МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Самарский государственный технический университет»
(ФГБОУ ВО «СамГТУ»)

УТВЕРЖДАЮ:
Проректор по учебной работе

_____
(подпись, ФИО)
«____»_____20__ г.

РАБОЧАЯ ПРОГРАММА МОДУЛЯ

Языки программирования

УГСН	01.00.00 Математика и механика
Направление подготовки (специальность)	01.03.02 Прикладная математика и информатика
Квалификация	*Бакалавр*
Форма обучения	*Очная*
Год начала подготовки	*2025*
Объем дисциплины, ч. / з.е. / недели	*216/6/4*
Образовательные технологии	*Традиционные*
Форма контроля (промежуточная аттестация)	*Семестр 1: Экзамен; Семестр 2: Экзамен*
Подразделение – разработчик	*Кафедра «Информатика и вычислительная техника»*

Самара 2025

Языки программирования

Разработчик

Разработчик РПМ
Золин Алексей Георгиевич
Руководитель подразделения
Чуваков Александр Владимирович

СОДЕРЖАНИЕ

1. Перечень планируемых результатов обучения по модулю, соотнесенных с планируемыми результатами освоения образовательной программы 3

2. Место модуля в структуре образовательной программы 3

3. Объем модуля в зачетных единицах с указанием количества академических часов, выделенных на контактную работу обучающихся с преподавателем (по видам учебных занятий) и на самостоятельную работу обучающихся 4

4. Содержание модуля, структурированное по темам (разделам), с указанием отведенного на них количества академических часов и видов учебных занятий 5

4.1. Содержание лекционных занятий 6

4.2. Содержание лабораторных занятия 7

4.3. Содержание практических занятий 8

4.4. Содержание самостоятельной работы 9

5. Перечень учебной литературы и учебно-методического обеспечения по модулю 9

6. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», профессиональных баз данных, информационно-справочных систем 10

7. Описание материально-технической базы, необходимой для осуществления образовательного процесса по модулю 10

ПРИЛОЖЕНИЕ 11

Фонд оценочных средств 11

1. Перечень планируемых результатов обучения по модулю, соотнесенных с планируемыми результатами освоения образовательной программы

Таблица 1

Наименование категории (группы) компетенций	**Код и наименование компетенции**	**Код и наименование индикатора достижения компетенции**	**Результаты обучения** (знать, уметь, владеть)
Общепрофессиональные компетенции			
Теоретические и практические основы профессиональной деятельности	ОПК-2 Способен использовать и адаптировать существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач	ОПК-2.1 Знает существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы; основные принципы организации и управления памятью; основные принципы диспетчирования процессов и потоков в системах; взаимосвязь архитектурных особенностей аппаратуры ЭВМ и компонентов системного программного обеспечения; основные подходы к обеспечению безопасности функционирования операционных систем	Знать существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы.
		ОПК-2.2 Умеет разрабатывать и реализовывать алгоритмы решения прикладных задач, используя существующие математические методы, системы программирования и программные комплексы	Уметь разрабатывать алгоритмы решения прикладных задач, используя современные системы программирования и программные комплексы.
Информационно-коммуникационные технологии для профессиональной деятельности	ОПК-5 Способен разрабатывать алгоритмы и компьютерные программы, пригодные для практического применения	ОПК-5.1 Знает классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации	Знать классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации.
		ОПК-5.2 Умеет разрабатывать алгоритмические и программные решения системного и	Уметь разрабатывать алгоритмические и программные решения прикладного программного обеспечения.

Наименование категории (группы) компетенций	Код и наименование компетенции	Код и наименование индикатора достижения компетенции	Результаты обучения (знать, уметь, владеть)
		прикладного программного обеспечения	

2. Место модуля в структуре образовательной программы

Модуль входит в Обязательную часть.

Входные требования:

Предшествующие осваиваемые модули: Язык программирования C#; *Последующие осваиваемые модули:* Инфраструктура разработки ПО (DevOps инженер), Программирование баз данных, Операционные системы, Численные методы, Микропроцессорные системы, Математические основы методов защиты информации, Производственная практика: научно-исследовательская работа, Математические основы и численные методы искусственного интеллекта, Генеративный искусственный интеллект, Базы данных и методы BigData, Инфраструктура разработки ПО (DevOps инженер).

3. Объем модуля в зачетных единицах с указанием количества академических часов, выделенных на контактную работу обучающихся с преподавателем (по видам учебных занятий) и на самостоятельную работу обучающихся

Таблица 3

Вид учебной работы	Всего часов	В том числе часов в электронной форме	В том числе часов в форме практической подготовки	Семестр 1 часов/часов в электронной форме/в форме практической подготовки	Семестр 2 часов/часов в электронной форме/в форме практической подготовки
Аудиторная контактная работа (всего), в том числе:	**96**	**0**	**0**	**48**	**48**
лекционные занятия (ЛЗ)	32	0	0	16	16
лабораторные работы (ЛР)	64	0	0	32	32
практические занятия (ПЗ)	0	0	0	0	0
Внеаудиторная контактная работа, КСР	**6**	**0**	**0**	**3**	**3**
Самостоятельная работа (всего), в том числе:	**114**	**0**	**0**	**57**	**57**
Подготовка к лабораторным работам	*96*	*0*	*0*	*48*	*48*
Подготовка к экзамену	*18*	*0*	*0*	*9*	*9*
ИТОГО: час.	**216**	**0**	**0**	**108**	**108**
ИТОГО: з.е.	**6.0**	**0.0**	**0.0**	**3.0**	**3.0**

4. Содержание модуля, структурированное по темам (разделам), с указанием отведенного на них количества академических часов и видов учебных занятий

Таблица 4

№ раздела	Наименование раздела дисциплины	Виды учебной нагрузки и их трудоемкость, часы				
		ЛЗ	**ЛР**	**ПЗ**	**СРС**	**Всего часов**
1.	Основные методологии и технологии программирования: структурное программирование.	16	32	0	57	105
2.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	16	32	0	57	105
	КСР					*6*
	Итого:	*32*	*64*	*0*	*114*	*216*

4.1. Содержание лекционных занятий

Таблица 4.1

№ ЛЗ	Наименование раздела	Тема лекции	**Содержание лекции** (перечень дидактических единиц: рассматриваемых подтем, вопросов)	Количество часов	Количество часов в электронной форме
Семестр 1					
1.	Основные методологии и технологии программирования: структурное программирование.	Введение в дисциплину «Языки программирования».	Основные понятия курса. Новейшие направления в области технологий программирования.	2	0
2.	Основные методологии и технологии программирования: структурное программирование.	Понятия алгоритма, исполнитель алгоритма, свойства алгоритмов.	Характеристика понятия алгоритм, свойства алгоритма. Величины в алгоритмах. Базовые структуры алгоритмов: линейный алгоритм, разветвленный алгоритм, циклический алгоритм.	2	0
3.	Основные методологии и технологии программирования: структурное программирование.	Основные конструкции языка программирования С++.	Структура программы. Директивы прекомпилятору. Основные конструкции: следование, ветвление, циклы.	2	0
4.	Основные методологии и технологии программирования: структурное программир	Обработка пользовательских типов данных: одномерные массивы.	Необходимость создания пользовательских типов данных. Одномерные массивы, их реализация. Многомерные массивы, особенности обработки.	2	0

№ ЛЗ	Наименование раздела	Тема лекции	Содержание лекции (перечень дидактических единиц: рассматриваемых подтем, вопросов)	Количество часов	Количество часов в электронной форме
	ование.				
5.	Основные методологии и технологии программирования: структурное программирование.	Строки в С++.	Строки как частный случай массива. Обработка строк с использованием функций библиотеки string.h. Использование стандартных средств ввода/вывода, библиотека манипуляторов iomanip.h.	2	0
6.	Основные методологии и технологии программирования: структурное программирование.	Работа с динамической памятью.	Указатели, работа с динамической памятью, распределение памяти. Представление массива в виде указателя.	2	0
7.	Основные методологии и технологии программирования: структурное программирование.	Модульное программирование.	Разбивка задачи на подзадачи. Выделение функциональных составляющих. Проектирование модулей. Объединение модулей в единую программу.	2	0
8.	Основные методологии и технологии программирования: структурное программирование.	Параметры функций.	Локальные и глобальные переменные. Формальные и фактические параметры. Передача параметров в функцию.	2	0
			Итого за семестр:	*16*	*0*
			Семестр 2		
1.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Файлы в С++.	Операции ввода-вывода. Обработка файлов средствами языка С++. Текстовые и двоичные файлы. Режимы работы с файлами.	2	0
2.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Пользовательский тип данных структура.	Разработка пользовательских типов данных. structure. Объединение данных разных типов для упрощения обработки. Доступ к полям структур. Хранение и обработка массивов структур. Недостатки использования массивов структур при создании баз данных.	2	0
3.	Объектно-ориентированное программирование. Реализация основных	Соотношение основных понятий ООП.	Виды доступа к полям и методам класса. Требования к объектам класса Разделы private, protected, public при создании класса. Описание объектов типа класс. Взаимодействие нескольких объектов одного и различных классов.	2	0

№ ЛЗ	Наименование раздела	Тема лекции	Содержание лекции (перечень дидактических единиц: рассматриваемых подтем, вопросов)	Количество часов	Количество часов в электронной форме
	алгоритмов обработки данных.				
4.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Конструкторы и деструкторы.	Конструкторы и деструкторы. Методы построения классов. Объектный тип данных. Переменные объектного типа.	2	0
5.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Потоковые классы.	Стандартные потоки. Форматирование данных. Манипуляторы. Строковые потоки. Файловые потоки.	2	0
6.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Контейнерные классы.	Последовательные контейнеры (векторы, списки, стеки, очереди). Ассоциативные контейнеры (словари, множества).	2	0
7.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Алгоритмы сортировки.	Задача сортировки (внешней и внутренней). Быстрая сортировка Хоара. Процедура разделения. Рекурсивный и не рекурсивный алгоритмы быстрой сортировки. Анализ сложности. Пирамидальная сортировка: турнирная сортировка, построение пирамиды и полное упорядочение. Анализ сложности алгоритма.	2	0
8.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Стандартные алгоритмы сортировки и поиска.	Классификация алгоритмов сортировки. Алгоритмы, связанные с сортировкой: binary search, equal_range, inplace_merge, lexicographical_compare, lower_bound, upper_bound, max, min, max_element, min_element, merge, next_permutation, prev_permutation, partial_sort, partition, stable_partition, sort.	2	0
			Итого за семестр:	*16*	*0*
			Итого:	**32**	**0**

4.2. Содержание лабораторных занятий

Таблица 4.2

№ ЛабЗ	Наименование раздела	Наименование лабораторной работы	Содержание лабораторной работы (перечень рассматриваемых дидактических единиц: подтем, вопросов)	Количество часов	Количество часов в электронной форме	Количество часов в форме практической подготовки
Семестр 1						
1.	Основные методологии и технологии программирования: структурное программирование.	Условные операторы на языке С++.	Составление графической схемы алгоритма решения задачи; написание программы на языке С++ для простейших математических расчетов; написание программы, содержащей условные операторы.	4	0	*0*
2.	Основные методологии и технологии программирования: структурное программирование.	Циклические алгоритмы.	Составление графической схемы циклических алгоритмов; реализация циклических алгоритмов на языке С++.	4	0	*0*
3.	Основные методологии и технологии программирования: структурное программирование.	Обработка одномерных массивов.	Написание программы на языке С++ с использованием одномерных массивов и их обработка.	4	0	*0*
4.	Основные методологии и технологии программирования: структурное программирование.	Указатели. Динамические массивы.	Работа с указателями. Выделение динамической памяти. Обработка одномерных массивов с помощью указателей.	4	0	*0*
5.	Основные методол	Обработка многомерных массивов.	Написание программы на языке С++ с использованием двумерных массивов, стандартные возможности	4	0	*0*

№ ЛабЗ	Наименование раздела	Наименование лабораторной работы	Содержание лабораторной работы (перечень рассматриваемых дидактических единиц: подтем, вопросов)	Количество часов	Количество часов в электронной форме	Количество часов в форме практической подготовки
	огии и технологии программирования: структурное программирование.		библиотек манипуляторов.			
6.	Основные методологии и технологии программирования: структурное программирование.	Символьные и строковые данные.	Описание строковых данных в виде массивов символов и в виде объектов типа string и их обработка.	4	0	*0*
7.	Основные методологии и технологии программирования: структурное программирование.	Функции в С++.	Создание функций. Вызов функций. Передача параметров.	4	0	*0*
8.	Основные методологии и технологии программирования: структурное программирование.	Прием отчетов по лабораторным работам.	Прием отчетов по лабораторным работам.	4	0	*0*
			Итого за семестр:	*32*	*0*	*0*
			Семестр 2			
1.	Объектно-ориентированное программирование. Реализация	Массивы структур.	Пользовательский тип данных структура – объединение данных различных типов в виде единого объекта и возможности его обработки. Обработка массивов структур, как подготовка к реализации баз данных.	4	0	*0*

№ ЛабЗ	Наименование раздела	Наименование лабораторной работы	Содержание лабораторной работы (перечень рассматриваемых дидактических единиц: подтем, вопросов)	Количество часов	Количество часов в электронной форме	Количество часов в форме практической подготовки
	основных алгоритмов обработки данных.					
2.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Создание классов в C++.	Описание класса, создание объектов класса. Конструкторы и деструкторы. Методы класса. Виды доступа.	4	0	*0*
3.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Обработка текстовых файлов.	Создание, модификация, сохранение текстовых файлов. Одновременная обработка нескольких файлов.	4	0	*0*
4.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Двоичные (бинарные) файлы.	Создание, модификация, сохранение бинарных файлов. Обработка наборов структур, записанных в бинарный файл.	4	0	*0*
5.	Объектно-ориентированное программирован	Линейные односвязные списки.	Динамические структуры данных. Односвязные и двусвязные списки. Создание списка, добавление узлов, удаление узлов. Сохранение списка в бинарный файл, чтение списка из файла.	4	0	*0*

№ ЛабЗ	Наименование раздела	Наименование лабораторной работы	Содержание лабораторной работы (перечень рассматриваемых дидактических единиц: подтем, вопросов)	Количество часов	Количество часов в электронной форме	Количество часов в форме практической подготовки
	ие. Реализация основных алгоритмов обработки данных.					
6.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Алгоритмы сортировки.	Виды алгоритмов сортировки. Их достоинства и недостатки. Реализация для различных типов данных.	4	0	*0*
7.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Алгоритмы сортировки.	Виды алгоритмов сортировки. Их достоинства и недостатки. Реализация для различных типов данных.	4	0	*0*
8.	Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Прием отчетов по лабораторным работам.	Прием отчетов по лабораторным работам.	4	0	*0*
			Итого за семестр:	*32*	*0*	*0*
			Итого:	**64**	**0**	***0***

4.3. Содержание самостоятельной работы

Таблица 4.4

Наименование раздела	Вид самостоятельной работы	Содержание самостоятельной работы (перечень дидактических единиц: рассматриваемых подтем, вопросов)	Количество часов
Семестр 1			
Основные методологии и технологии программирования: структурное программирование.	Подготовка к лабораторным работам	Подготовка к практическим занятиям.	48
Основные методологии и технологии программирования: структурное программирование.	Подготовка к экзамену	Подготовка к экзамену по пройденным темам.	9
		Итого за семестр:	*57*
Семестр 2			
Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Подготовка к лабораторным работам	Подготовка к практическим занятиям.	48
Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	Подготовка к экзамену	Подготовка к экзамену по пройденным темам.	9
		Итого за семестр:	*57*
		Итого:	**114**

5. Перечень учебной литературы и учебно-методического обеспечения по модулю

Таблица 5

№ п/п	Библиографическое описание	Ресурс НТБ СамГТУ	Ссылка на ресурс
1.	Золин, А.Г. Языки и методы программирования: динамические структуры данных, сортировка и поиск : учебное пособие / А. Г. Золин, Е. А. Халикова; Самарский государственный технический университет, Информационные технологии.- Самара, 2021.- 152 с.	Электронная библиотека СамГТУ (ЭБС Политех)	http://platform.samgtu.ru/getinfo?uid=els_samgtu-elib-5461
2.	Золин, А.Г. Языки и методы программирования. Введение в разработку на С++ (первый семестр) : учебное пособие / А. Г. Золин, А. Е. Колоденкова, Е. А. Халикова; Самарский государственный технический университет, Информационные технологии.- Самара, 2020.- 128 с.	Электронная библиотека СамГТУ (ЭБС Политех)	http://platform.samgtu.ru/getinfo?uid=els_samgtu-elib-3978
3.	Алгоритмические языки и программирование: учебное пособие / Токманцев Т.Б., Профобразование, Уральский федеральный университет, ред. Костоусова В.Б.: 2024	ЭБС IPR Books	http://platform.samgtu.ru/getinfo?uid=els_samgtu-iprbooks-139513
4.	Основы программирования. Введение в язык Си. Ч.1: учебное пособие / Журавлева М.Г., Алексеев В.А., Домашнев П.А., Липецкий государственный технический университет, ЭБС АСВ: 2019	ЭБС IPR Books	http://platform.samgtu.ru/getinfo?uid=els_samgtu-iprbooks-101463
5.	Седжвик, Роберт Алгоритмы на С++ : анализ структуры данных, сортировка, поиск, алгоритмы на графах [Текст] / [пер. с англ. А. А. Моргунова; под ред. Ю. Н. Артеменко].- Москва; Санкт-Петербург; Киев, Вильямс, 2014.- 1056 с. : ил.	Книжный фонд АСА СамГТУ	http://platform.samgtu.ru/getinfo?uid=els_samgtu-asa-69036

Доступ обучающихся к ЭР НТБ СамГТУ (elib.samgtu.ru) осуществляется посредством электронной информационной образовательной среды университета и сайта НТБ СамГТУ по логину и паролю.

6. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», профессиональных баз данных, информационно-справочных систем

Таблица 6

№ п/п	Наименование	Ссылка на веб-страницу ресурса	Режим доступа
1.	*Библиотека проекта [Электронный ресурс]. - Электрон. текстовые, граф., зв. дан., Сайт в сети Интернет*	*http://www.museum.ru/ museum/1812/Library/ contents.html*	*Ресурсы открытого доступа*
2.	*eLIBRARY.RU (НЭБ - Научная электронная библиотека "Российский индекс научного цитирования") [Электронный ресурс]. - Электрон. текстовые, граф., зв. дан., Сайт в сети Интернет*	*http://www.elibrary.ru/*	*Ресурсы открытого доступа*
3.	*УИС РОССИЯ - Университетская информационная система РОССИЯ [Электронный ресурс]. - Электрон. текстовые, граф., зв. дан., Сайт в сети Интернет*	*http://www.cir.ru/*	*Ресурсы открытого доступа*
4.	*Единое окно доступа к образовательным ресурсам [Электронный ресурс]. - Электрон. текстовые, граф., зв. дан., Сайт в сети Интернет*	*http://window.edu.ru/*	*Ресурсы открытого доступа*

7. Описание материально-технической базы, необходимой для осуществления образовательного процесса по модулю

Лекционные занятия проводятся в:

Корпус № 1/432 - Учебная аудитория (Аудитория на 103 посадочных места оснащена следующими техническими средствами обучения и оборудованием: учебная мебель, доска аудиторная; экран. Мультимедийное оборудование: проектор Mitsubishi DLP; усилитель Apart MA 125; 6 аудио-колонок; радиосистема SR40, SR45.)

Корпус № 8/414 - Учебная аудитория (Аудитория на 36 посадочных мест оснащена следующими техническими средствами обучения и оборудованием: учебная мебель, доска аудиторная магнитно-маркерная, 1 персональный компьютер iRU 320 FX-4100/Gigabyte Ga-880GMA-USB3/4Gb/320Gb/k+M/DVDRW, с операционной системой Microsoft Windows 7 Professional. На ПК установлены ПО: Oracle VirtualBox, K-LiteCodecPack, Far Manager, 7-Zip,Notepad++, Foxit Reader, PostgreSQL, Google Chrome, OpenJDK, Apache NetBeans, PVS-Studio, Microsoft Visual Studio, Microsoft SQL Server 2012, Microsoft SQL Server Managment Studio, Kaspersky Endpoint Security, Microsoft Office Project Профессиональный 2007, MATLAB R2010a, Scilab-6.0.1 (64bit), Microsoft Visio Профессиональный 2010, Microsoft .NET Framework 4.8, Android Studio, Mozilla Firefox 68.0.2 (64 ru), Deductor Academic. 9 ПК Intel Core i5 с операционной системой Microsoft Windows 7 Professional. На ПК установлены ПО: офисный пакет MS Office 2007, Oracle VirtualBox, K-LiteCodecPack, Far Manager, 7-Zip,Notepad++, Foxit Reader, PostgreSQL, Google Chrome, OpenJDK, Apache NetBeans, PVS-Studio, Microsoft Visual Studio, Microsoft SQL Server 2012, Microsoft SQL Server Managment Studio, Kaspersky Endpoint Security, Microsoft Office Project Профессиональный 2007, MATLAB R2010a, Scilab-6.0.1 (64bit), Microsoft Visio Профессиональный 2010, Microsoft .NET Framework 4.8, Android Studio, Mozilla Firefox 68.0.2 (64 ru), Deductor Academic. Обеспечено проводное подключение ПК к локальной сети и сети Интернет.).

Практические занятия не предусмотрены.

Самостоятельная работа проводятся в:

Корпус № 8/200 - Читальный зал (Аудитория на 70 посадочных мест оснащена следующими техническими средствами обучения и оборудованием: учебная мебель, столы учебные.).

Лабораторные занятия проводятся в:

Корпус № 8/419 - Компьютерный класс (Аудитория на 33 посадочных места оснащена следующими техническими средствами обучения и оборудованием: парты, столы учебные, доска аудиторная магнитно-маркерная 100х180 эмаль. 12 ПК LG FLATRON 22EN43 в комплекте. На ПК, на базе операционной системы MS Windows 7 Professional, установлено следующее ПО: офисный пакет MS Office 2007, Oracle VirtualBox, K-LiteCodecPack, Far Manager, 7-Zip,Notepad++, Foxit Reader, PostgreSQL, Google Chrome, OpenJDK, Apache NetBeans, PVS-Studio, Microsoft Visual Studio, Microsoft SQL Server 2012, Microsoft SQL Server Managment Studio, Kaspersky Endpoint Security, Microsoft Office Project Профессиональный 2007,

MATLAB R2010a, Scilab-6.0.1 (64bit), Microsoft Visio Профессиональный 2010, Microsoft .NET Framework 4.8, Android Studio, Mozilla Firefox 68.0.2 (64 ru), Deductor Academic. Обеспечено проводное подключение ПК к локальной сети и сети Интернет.)

Корпус № 8/422 - Компьютерный класс (Аудитория на 34 посадочных места оснащена следующими техническими средствами обучения и оборудованием: парты, столы учебные, доска аудиторная магнитно-маркерная 100х180 эмаль, проектор Optome EX551; 1 ПК Intel Core i5 в комплекте, 10 ПК AMD Ruzen 5 5600G OEM. На базе операционной системы MS Windows 7 Professional на ПК установлены следующие ПО: офисный пакет MS Office 2007, Oracle VirtualBox, K-LiteCodecPack, Far Manager, 7-Zip,Notepad++, Foxit Reader, PostgreSQL, Google Chrome, OpenJDK, Apache NetBeans, PVS-Studio, Microsoft Visual Studio, Microsoft SQL Server 2012, Microsoft SQL Server Managment Studio, Kaspersky Endpoint Security, Microsoft Office Project Профессиональный 2007, MATLAB R2010a, Scilab-6.0.1 (64bit), Microsoft Visio Профессиональный 2010, Microsoft .NET Framework 4.8, Android Studio, Mozilla Firefox 68.0.2 (64 ru), Deductor Academic. Обеспечено проводное подключение ПК к локальной сети и сети Интернет.).

ПРИЛОЖЕНИЕ

Фонд оценочных средств

Фонд оценочных средств для текущего контроля успеваемости, промежуточной аттестации обучающихся по модулю

Настоящий Фонд оценочных средств (ФОС) по модулю Языки программирования является неотъемлемым приложением к рабочей программе модуля. На данный ФОС распространяются все реквизиты утверждения, представленные в рабочей программе модуля.

1. Перечень оценочных средств

Таблица 1

№ п/п	Оценочное средство	Краткая характеристика оценочного средства	Количество	Представление оценочного средства в ФОС
1	Отчёт по лабораторной работе	Оценка способности студента применить полученные ранее знания для проведения анализа, опыта, эксперимента и выполнения последующих расчетов, а также составления выводов.	16	Требования к содержанию и оформлению отчета
2	Контрольная точка в форме тестирования	Инструмент, с помощью которого преподаватель оценивает степень достижения студентом требуемых знаний, умений, навыков в формате рубежного контроля по итогам изучения разделов. Составление теста включает в себя создание выверенной системы вопросов, собственно процедуру проведения тестирования и способ измерения полученных результатов.	2	Спецификация тестов и примеры вариантов тестовых заданий

Раздел 2. Этапы формирования компетенций, соотнесенные с видами оценочных средств

Таблица 2

Контролируемые разделы модуля	Код и формулировка индикатора компетенции	Результат обучения	Оценочные средства	Номер оценочного средства
Основные методологии и технологии программирования: структурное программирование.	ОПК-2.1 Знает существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы; основные принципы организации и управления памятью; основные принципы диспетчирования процессов и потоков в системах;	Знать существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы.	Отчёт по лабораторной работе	1-8

Контролируемые разделы модуля	Код и формулировка индикатора компетенции	Результат обучения	Оценочные средства	Номер оценочного средства
	взаимосвязь архитектурных особенностей аппаратуры ЭВМ и компонентов системного программного обеспечения; основные подходы к обеспечению безопасности функционирования операционных систем			
Основные методологии и технологии программирования: структурное программирование.	ОПК-2.2 Умеет разрабатывать и реализовывать алгоритмы решения прикладных задач, используя существующие математические методы, системы программирования и программные комплексы	Уметь разрабатывать алгоритмы решения прикладных задач, используя современные системы программирования и программные комплексы.	Отчёт по лабораторной работе	1-8
Основные методологии и технологии программирования: структурное программирование.	ОПК-5.1 Знает классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации	Знать классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации.	Отчёт по лабораторной работе	1-8
Основные методологии и технологии программирования: структурное программирование.	ОПК-5.2 Умеет разрабатывать алгоритмические и программные решения системного и прикладного программного обеспечения	Уметь разрабатывать алгоритмические и программные решения прикладного программного обеспечения.	Отчёт по лабораторной работе	1-8
Основные методологии и технологии программирования: структурное программирование.	ОПК-2.1 Знает существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы; основные принципы организации и управления памятью; основные принципы диспетчирования процессов и потоков в системах; взаимосвязь архитектурных	Знать существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы.	Контрольная точка в форме тестирования	1

Контролируемые разделы модуля	Код и формулировка индикатора компетенции	Результат обучения	Оценочные средства	Номер оценочного средства
	особенностей аппаратуры ЭВМ и компонентов системного программного обеспечения; основные подходы к обеспечению безопасности функционирования операционных систем			
Основные методологии и технологии программирования: структурное программирование.	ОПК-5.1 Знает классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации	Знать классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации.	Контрольная точка в форме тестирования	1
Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	ОПК-2.1 Знает существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы; основные принципы организации и управления памятью; основные принципы диспетчирования процессов и потоков в системах; взаимосвязь архитектурных особенностей аппаратуры ЭВМ и компонентов системного программного обеспечения; основные подходы к обеспечению безопасности функционирования операционных систем	Знать существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы.	Отчёт по лабораторной работе	1-8
Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	ОПК-2.2 Умеет разрабатывать и реализовывать алгоритмы решения прикладных задач, используя существующие математические методы, системы программирования и программные	Уметь разрабатывать алгоритмы решения прикладных задач, используя современные системы программирования и программные комплексы.	Отчёт по лабораторной работе	1-8

Контролируемые разделы модуля	Код и формулировка индикатора компетенции	Результат обучения	Оценочные средства	Номер оценочного средства
	комплексы			
Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	ОПК-5.1 Знает классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации	Знать классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации.	Отчёт по лабораторной работе	1-8
Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	ОПК-5.2 Умеет разрабатывать алгоритмические и программные решения системного и прикладного программного обеспечения	Уметь разрабатывать алгоритмические и программные решения прикладного программного обеспечения.	Отчёт по лабораторной работе	1-8
Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	ОПК-2.1 Знает существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы; основные принципы организации и управления памятью; основные принципы диспетчирования процессов и потоков в системах; взаимосвязь архитектурных особенностей аппаратуры ЭВМ и компонентов системного программного обеспечения; основные подходы к обеспечению безопасности функционирования операционных систем	Знать существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач; методы и алгоритмы управления процессами и ресурсами операционной системы.	Контрольная точка в форме тестирования	2
Объектно-ориентированное программирование. Реализация основных алгоритмов обработки данных.	ОПК-5.1 Знает классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации	Знать классификацию языков программирования, основные методы программного обеспечения, стандарты оформления программной документации.	Контрольная точка в форме тестирования	2

3. Содержание оценочных средств текущего контроля

Оценочное средство 1:
Пример отчета по результатам практических работ того-то прилагаются к рабочей программе.
Оценочное средство 2:
Пример тестов прилагаются к рабочей программе.

4. Содержание оценочных средств промежуточной аттестации

Семестр 1:

Вид аттестационного испытания 1: Устное собеседование.
Форма проведения промежуточной аттестации: Экзамен.
Типовые задания: Вопросы к экзамену прилагаются к рабочей программе.

Семестр 2:

Вид аттестационного испытания 1: Устное собеседование.
Форма проведения промежуточной аттестации: Экзамен.
Типовые задания: Вопросы к экзамену прилагаются к рабочей программе.

5. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующие процесс формирования компетенций

Модуль состоит из набора разделов. Объем учебного раздела раскрывает отдельную (ые) тему(ы) модуля / этап прохождения практики. Модуль разбивается, как правило, на 2-4 раздела в семестр.

Раздел состоит из учебных элементов (дидактических единиц), сгруппированных по темам в рамках раздела. Объем учебного материала учебного элемента раскрывает отдельный вопрос темы. Путем набора соответствующих тем и учебных элементов формируется раздел, содержание которого соответствует запланированным результатам освоения ОП согласно учебному плану.

По каждому разделу устанавливается перечень обязательных видов работы студента, включающий: ответы на теоретические вопросы на семинаре; решение практических задач и выполнение заданий на практических занятиях; выполнение лабораторных работ; выполнение контрольных работ; написание рефератов; участие в коллоквиумах; тестирование по теме (группе тем); другие виды работ, определяемые преподавателем (в соответствии с установленными требованиями балльно-рейтинговой технологии).

Каждый раздел включает определенные формы контроля для оценки уровня усвоения учебного материала и определения рейтинговой оценки качества усвоения учебного материала.

Рейтинговая оценка знаний студентов определяется по 100-балльной шкале в каждом семестре и включает текущий, рубежный и итоговый контроль.

Накопленный рейтинг (фиксированная величина по окончании семестра) — это фактический рейтинг по всем освоенным к данному моменту разделам дисциплины, включая их текущий, рубежный и итоговый контроль, составляет N баллов (N – индивидуальная величина, зависящая от проделанной работы студента по дисциплине).

Проходной рейтинг (включает текущий контроль, рубежный контроль в семестре и промежуточную аттестацию) — это минимум баллов, набрав который студент будет считаться аттестованным по дисциплине, составляет 60 баллов в семестр по одной дисциплине.

Пороговый рейтинг (включает текущий контроль и рубежный контроль в семестре) — это минимальный фактический рейтинг семестрового контроля, набрав который, студент проходит на итоговый контроль, составляет 50% от суммы баллов, которые студент может набрать за текущий контроль и рубежный контроль в течение семестра.

Текущий контроль служит для получения первичной информации о ходе усвоения отдельных элементов содержания учебной дисциплины. Основное его назначение состоит в осуществлении контроля за всеми видами аудиторной и самостоятельной внеаудиторной деятельности студентов, результаты которого оцениваются в баллах.

Рубежный контроль (контрольные точки текущего контроля) проводится после изучения завершенной в смысловом и логическом плане части учебного материала (раздела, модуля). Этот вид контроля позволяет получить информацию о ходе усвоения некоторой совокупности содержательных элементов учебной дисциплины (раздела, модуля) и оценивается в баллах.

Промежуточный итоговый контроль (контрольные точки промежуточной аттестации) – курсовой проект, курсовая работа и экзамен, зачет, зачет с оценкой. Итоговый контроль по дисциплине в семестре предназначен для аттестации студента по учебной дисциплине: процедура установления соответствия уровня и качества подготовки студентов системе требований, зафиксированной в ФГОС ВО.

Распределение баллов рейтинговой оценки между видами контроля

Таблица 3

Номер семестра	Распределение баллов					Проходной рейтинг
	Текущий и рубежный контроль			Промежуточный итоговый контроль		
	Текущий контроль	Рубежный контроль	Пороговый рейтинг	Курсовая работа/курсовой проект	Зачет/зачет с оценкой/экзамен	
1	40	30	35	0	30	60
2	40	30	35	0	30	

Перевод оценок 100-балльной шкалы в оценки промежуточной аттестации

Таблица 4

100-балльная шкала	85-100	75-84	60-74	0-59
5-балльная шкала	Отлично	Хорошо	Удовлетворительно	Неудовлетворительно
Бинарная шкала	Зачтено			Не зачтено

Распределение рейтинговых баллов по разделам модуля с указанием применяемых для их начисления оценочных средств представлено в Технологической карте БРТ.

Технологическая карта БРТ

Семестр 1

Раздел учебного модуля	Оценочные средства			Итого
	Отчёт по лабораторной работе	**Контрольная точка в форме тестирования**	**Экзамен**	
Раздел 1				
ЛР 1.1	5	0		5
ЛР 1.2	5	0		5
ЛР 1.3	5	0		5
ЛР 1.4	5	0		5
ЛР 1.5	5	0		5
ЛР 1.6	5	0		5
ЛР 1.7	5	0		5
ЛР 1.8	5	30		35
Итого по разделу 1	40	30		70
Промежуточная аттестация			30	30
Итого	40	30	30	100

Семестр 2

Раздел учебного модуля	Оценочные средства			Итого
	Отчёт по лабораторной работе	**Контрольная точка в форме тестирования**	**Экзамен**	
Раздел 1				
ЛР 1.1	5	0		5
ЛР 1.2	5	0		5
ЛР 1.3	5	0		5
ЛР 1.4	5	0		5
ЛР 1.5	5	0		5
ЛР 1.6	5	0		5
ЛР 1.7	5	0		5
ЛР 1.8	5	30		35
Итого по разделу 1	40	30		70
Промежуточная аттестация			30	30
Итого	40	30	30	100

А.Г. ЗОЛИН

А.Е. КОЛОДЕНКОВА

Е.А. ХАЛИКОВА

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ. ВВЕДЕНИЕ В РАЗРАБОТКУ НА C++

(первый семестр)

Учебное пособие

Самара

Самарский государственный технический университет

2020

МИНОБРНАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

Кафедра «Информационные технологии»

А.Г. ЗОЛИН

А.Е. КОЛОДЕНКОВА

Е.А. ХАЛИКОВА

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ. ВВЕДЕНИЕ В РАЗРАБОТКУ НА C++ (первый семестр)

Учебное пособие

Самара

Самарский государственный технический университет

2020

Печатается по решению редакционно-издательского совета СамГТУ

УДК 519.682.2

Г 94

Золин А.Г.

Языки и методы программирования. Введение в разработку на C++ (первый семестр): учебное пособие / *А.Г. Золин, А.Е. Колоденкова, Е.А. Халикова*. – Самара: Самар. гос. техн. ун-т, 2020. – с.: ил.

Рассмотрены основы программирования на языке *C++* в целях получения навыков программной реализации алгоритмов, структур и методов обработки данных. Предназначено для студентов высших технических учебных заведений, обучающихся по направлениям 01.03.02 "Прикладная математика и информатика", 09.03.01 «Информатика и вычислительная техника», 09.03.03 «Прикладная информатика», 09.03.04 «Программная инженерия», 10.03.01 «Информационная безопасность», 11.03.01 «Радиотехника»

Рецензент

УДК 519.682.2

Г 94

© А.Г. Золин, А.Е. Колоденкова, Е.А. Халикова 2020

© Самарский государственный технический университет, 2020

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

ПК – персональный компьютер;

ПО – программное обеспечение;

ИСР – интегрированная среда разработки;

GUI – графический интерфейс пользователя;

ПРЕДИСЛОВИЕ

Языки программирования служат двум взаимосвязанным целям: они предоставляет программисту инструмент для описания подлежащих выполнению действий и набор концепций, которыми оперирует программист, обдумывая, что можно сделать. Первая цель требует языка, близкого к компьютеру, чтобы все важные элементы компьютера управлялись просто и эффективно способом, достаточно очевидным для программиста. Для реализации второй цели необходим язык, близкий к решаемой задаче, позволяющий описывать объекты и процессы предметной области.

C++ – это объектно-ориентированный язык программирования, хорошо известный своей эффективностью, экономичностью и переносимостью. Указанные преимущества C++ обеспечивают высокое качество разработки программного продукта. Использование C++ в качестве инструментального языка позволяет получать быстрые и компактные программы. Во многих случаях программы, написанные на C++, сравнимы по скорости с программами, написанными на языке ассемблера.

Первые шаги при программировании на C++ требуют от студента тщательного проектирования программы, а также определенной дисциплины при программировании. Общий подход к разработке программного обеспечения включает в себя три этапа. Сначала необходимо ясно понять задачу (анализ требований, постановка задачи), затем определить ключевые концепции, в терминах которых выражается решение задачи (проектирование, алгоритмизация), и, наконец, воплотить решение в программе (программирование).

Предлагаемое пособие служит для самостоятельного изучения основ C++ и является дополнением к лекционным курсам по таким дисциплинам, как «Языки и методы программирования», «Техноло-

гия программирования» и др. Может использоваться бакалаврами, которые обучаются по направлениям: 01.03.02 "Прикладная математика и информатика", 09.03.01 «Информатика и вычислительная техника», 09.03.03 «Прикладная информатика», 09.03.04 «Программная инженерия», 10.03.01 «Информационная безопасность», 11.03.01 «Радиотехника», а также будет полезно всем, кто начинает первые шаги в изучении основ программирования и пытается освоить язык программирования C++.

ВВЕДЕНИЕ

Несмотря на то, что язык программирования C++ был создан сотрудником фирмы Bell Laboratories Бьёрном Страуструпом в начале 1980-х годов, в последние 10 лет были проведены существенные обновления стандартов, добавившие новые дополнения, как в сам язык, так и в стандартную библиотеку. В настоящее время C++ позволяет решать широкий круг задач в области информационных технологий: создание высокопроизводительных систем обработки, создание компьютерных игр, обработка мультимедиа. С его помощью возможна разработка клиентских приложений с графическим пользовательским интерфейсом (*GUI*), работа с базами данных и др.

Как инструмент разработки программных приложений, язык C++ обладает широкими возможностями для создания высококачественного программного обеспечения, однако является достаточно сложным в освоении и требует внимательности в изучении и постоянной практики написания программ.

Данное учебное пособие предназначено для студентов, изучающих дисциплину «Языки и методы программирования», базовым языком которой был выбран C++. Пособие охватывает материал первого семестра обучения и предоставляет как теоретический материал по основным разделам курса, так и задания для самостоятельного выполнения.

Теоретический материал разбит на девять глав. В первой главе даются основные сведения о языке C++ и средствах разработки. Данное пособие ориентировано на разработку программ на компьютерах с операционной системой MS Windows и средой программирования Visual Studio.

Вторая глава вводит понятие типа данных и описывает механизмы создания переменных, использования различных операций с ними. Здесь же рассмотрена простейшая программа на C++.

В третьей главе рассмотрен логический тип данных. Описываются логические операции и операции сравнения, позволяющие составлять логические выражения произвольной сложности. Рассмотрены операторы if и switch.

В четвертой главе рассматриваются циклические алгоритмы и конструкции языка C++ для их реализации.

Пятая глава знакомит читателя с понятием массива данных. В ней рассмотрены основные подходы к обработке одномерных массивов, их инициализации и выводу на экран.

В шестой главе рассматривается такой элемент языка C++, как указатель. Даются способы инициализации указателя, описывается взаимосвязь указателей и массивов. Также в этой главе рассмотрен механизм динамического выделения памяти.

Седьмая глава дает базовые рекомендации для работы с многомерными массивами. Здесь рассмотрены варианты создания и обработки многомерных массивов, размещаемых, как в стеке, так и динамически, в «куче».

Восьмая глава посвящена обработке символьной информации. В ней описываются способы задания в программном коде текстовых строк, объясняются основы работы с ними.

В девятой главе рассматриваются вопросы разработки собственных функций. Объясняются основные принципы использования этого механизма.

В завершение пособия в приложении 1 даются задания для лабораторного практикума. В приложении 2 представлен список книг, рекомендуемых для самостоятельного изучения.

1. ОСНОВЫЕ СВЕДЕНИЯ О ЯЗЫКЕ И СРЕДСТВАХ РАЗРАБОТКИ

1.1. Разработка программ на языке C++

Язык программирования C++ относится к классу компилируемых языков. Это означает, что для создания программного обеспечения (ПО) необходима специальная программа – компилятор, которая преобразует текстовый файл с исходным кодом в двоичный файл с машинными инструкциями, который может быть запущен на выполнение. На практике, процесс получения исполняемого файла более сложен и состоит из следующих этапов:

1. В текстовом редакторе набираются файлы с исходным кодом. В общем случае таких файлов может быть несколько для одной программы.
2. С помощью компилятора языка C++, каждый исходный файл преобразуется в объектный файл, содержащий машинные инструкции.
3. Полученные объектные файлы имеют неудовлетворенные связи в виде глобальных переменных и вызовов функций, реализованных в других файлах (в том числе и библиотечных), что делает невозможным выполнять их как полноценные программы. Поэтому, на третьем шаге запускается специальная программа – редактор связей, которая собирает все объектные файлы в единый исполняемый файл (с расширением «exe»).

Таким образом для создания программ на языке C++ необходима установка специального пакета со средствами разработки, включающего в себя компилятор, редактор связей и другие полезные утилиты. Часто такой пакет упрощенно называют «компилятором», подразумевая обязательное наличие полного набора необходимых программ.

Этапы 2 и 3 называют «сборкой программы» или «сборкой приложения».

1.2. Выбор среды разработки

В настоящий момент для операционной системы Windows существуют несколько компиляторов для языка C++. Самыми распространенными из них являются:

- компилятор фирмы Microsoft;
- mingw – компилятор с открытым исходным кодом, портированный из операционной системы Linux;
- Clang – кроссплатформенный компилятор с открытым исходным кодом.

Каждый из этих пакетов может быть установлен на персональный компьютер (ПК), после чего можно приступать к полноценной разработке ПО. Однако, в этом случае придется отдельно, вручную, запускать компилятор, редактор связей и другие утилиты из окна командной строки, для чего необходимо дополнительно настраивать переменные окружения, возможно писать сборочные файлы сценариев или изучать одну из многих систем автоматизации сборки ПО. К тому же, при данном подходе усложняется процесс запуска отладки приложения.

В связи с этим, гораздо удобнее, особенно начинающим разработчикам, использовать, так называемую, «Интегрированную среду разработки», включающую в себя: удобный текстовый редактор с подсветкой синтаксиса языка, компилятор, средства отладки. Интегрированная среда разработки (ИСР) позволяет избавиться от рутинных операций сборки ПО и сосредоточится на решении практических задач обучения программированию.

В данном учебном пособии рассматривается ИСР фирмы Microsoft – MS Visual Studio. Это одно из самых популярных средств

разработки для операционной системы MS Windows, поддерживающее несколько языков программирования, в том числе и язык C++. MS Visual Studio имеет несколько редакций:

- Community – бесплатная версия, включающая базовый набор возможностей;
- Professionl – платная версия с возможностью устанавливать различные дополнения для увеличения удобства написания исходного кода;
- Enterprise - платная версия с возможностью разработки корпоративных информационных систем, продвинутыми средствами отладки и другими дополнительными инструментами.

Для освоения навыков программирования, в данном пособии, рекомендуется использовать MS Visual Studio Community edition, поскольку эта ИСР содержит все необходимые компоненты для разработки программ на языке C++.

1.3. Установка среды разработки

Свежую версию Visual Studio Community edition можно найти по адресу: http://www.microsoft.com/visualstudio/rus/downloads. На момент написания данного пособия, свежей версией является *Visual Studio 2019*. Установка включает в себя несколько шагов.

Шаг 1. Проверка системных требований.

Перед установкой необходимо ознакомиться с системными требованиями к программной и аппаратной частям вашего ПК. Эти требования помогут вам узнать, поддерживает ли ваш компьютер *Visual Studio 2019*. Системные требования, необходимые для установки версии 2019, указаны по ссылке:

https://docs.microsoft.com/en-us/visualstudio/releases/2019/system-requirements

Примените последние обновления *Windows*. Это гарантирует, что на вашем компьютере будут установлены, как последние обновления безопасности, так и необходимые системные компоненты для *Visual Studio*.

Перезагрузите ПК. Перезагрузка завершит любые другие ожидающие установки ПО или обновления, и они не помешают установке *Visual Studio*.

Шаг 2. Загрузка *Visual Studio*.

Загрузите файл загрузчика *Visual Studio* по ссылке данной выше.

Шаг 3. Запуск установщика *Visual Studio*

Запустите файл начальной загрузки:

```
vs_community.exe
```

Эта программа включает в себя все необходимое для установки и настройки *Visual Studio*.

Подтвердите условия лицензии *Microsoft* и заявление о конфиденциальности. Выберите кнопку *«Дальше»*.

Шаг 4. Выбор рабочей нагрузки.

После запуска установщика вы можете использовать его для настройки установки, выбрав нужные наборы функций или рабочие нагрузки (*workload*). Найдите нужную рабочую нагрузку для разработки на языке C++ в установщике *Visual Studio* (на рисунке 1, показана красной стрелкой).

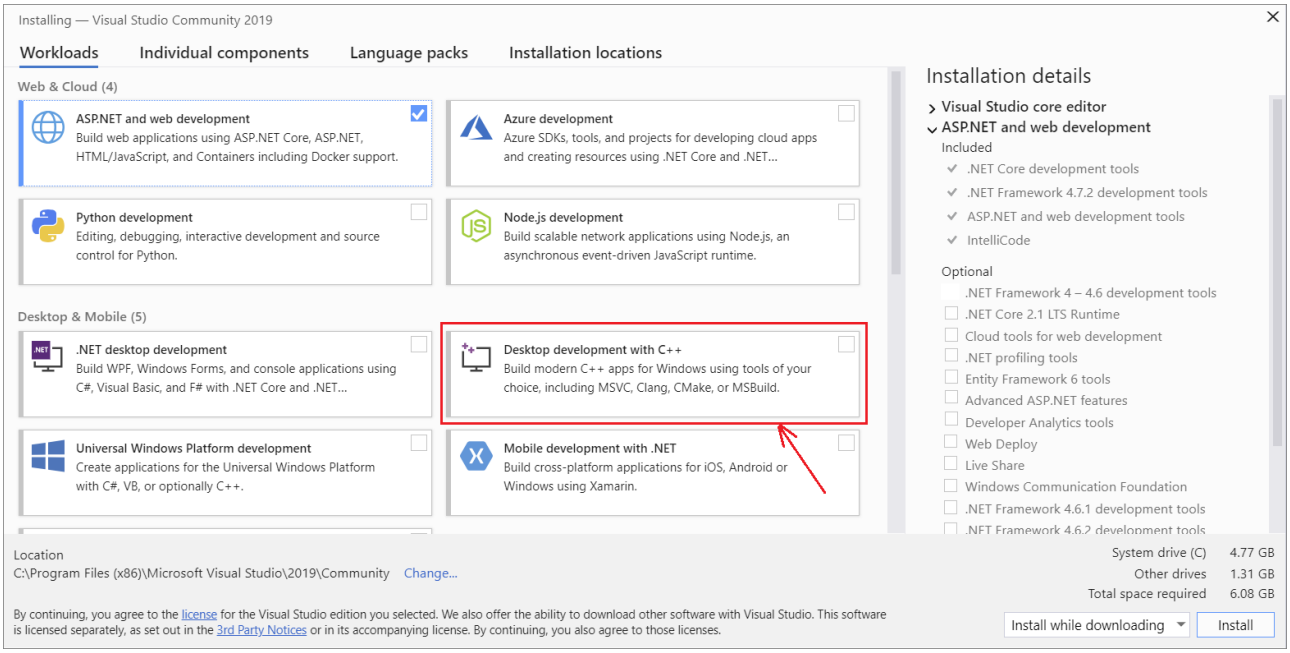


Рис. 1. Окно выбора рабочей нагрузки.

Шаг 5. Выбор места установки (необязательно).

Занимаемый объем после установки зависит от выбранных компонент. Можно уменьшить объем установки *Visual Studio* на системный диск: для этого требуется перенести кэш загрузки, общие компоненты, SDK и инструменты на разные диски и оставить *Visual Studio* на том диске, на котором он работает быстрее всего (рисунок 2).

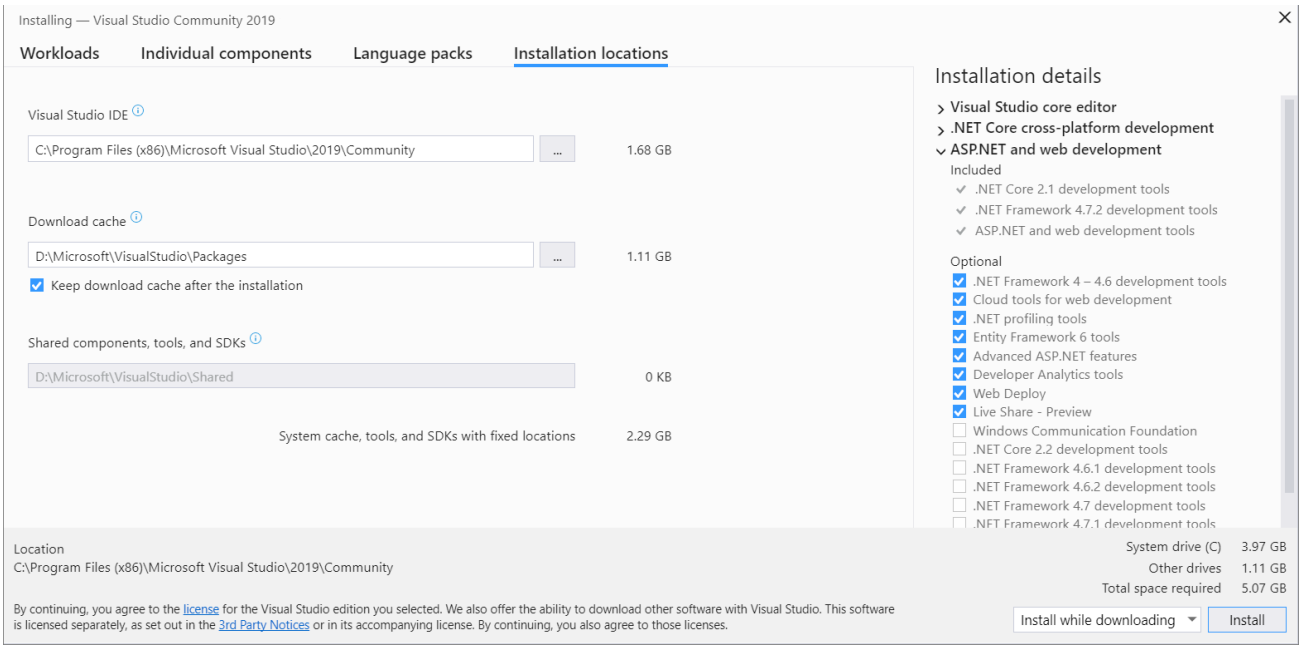


Рис. 2. Окно выбора расположения компонентов.

Шаг 6. Завершение установки.

Нажмите кнопку *«Установить»* (*Install*) в правом нижнем углу и дождитесь окончания установки программы.

1.4. Начало разработки

При работе с *Visual Studio,* вводятся понятия «Программный проект» (далее просто «Проект») и «Решение». Проект представляет собой папку на диске ПК, имеющую определенное имя (совпадающее с именем проекта) и содержащую различные файлы, обеспечивающие построение одного программного модуля (исполняемого файла или динамически загружаемой библиотеки):

- параметры настройки проекта;
- файлы с исходным кодом;
- промежуточные файлы, полученные в процессе построения проекта;
- исполняемый файл (exe) или динамически загружаемая библиотека (dll).

Проект представляет собой единицу построения программы и позволяет одной командой (нажатием кнопки мыши или «горячей» клавиши) собрать проект (скомпилировать все файлы с исходным кодом и запустить редактор связи) и/или выполнить программу, а также запустить отладку. Работа с проектами упрощает разработку ПО, позволяя манипулировать не множеством отдельных разрозненных файлов, а более крупными логическими единицами.

«Решение» позволяет объединить два (и более) проекта в некую единую проектную сущность. Это часто требуется в случае разработки многокомпонентных и/или распределенных приложений. В данном курсе такие случаи не рассматриваются.

Таким образом, каждый раз, когда мы хотим приступить к разработке новой программы, необходимо создавать новый проект. Для переноса одного проекта на другой ПК, необходимо копировать всю папку проекта целиком, не разбивая на отдельные файлы. Стороннее вмешательство в структуру проекта (например, через «Проводник») может нарушить его конфигурацию и проект перестанет корректно собираться.

Для создания нового проекта необходимо:

- или запустить *Visual Studio* и в появившемся окне выбрать «Создание проекта»,
- или, если *Visual Studio* уже запущена, выбрать пункт меню «Файл-Создать-Проект…».

В результате появится окно выбора типа проекта (рисунок 3).

В данном учебном пособии будет использоваться только один вид приложений – консольные приложения. Это приложения, запускаемые из командной строки и не имеющие графического интерфейса (GUI). Для создания такого проекта в окне (рисунок 3) необходимо выбрать:

- язык программирования: C++;
- платформа: *Windows*;
- тип проекта: «Консольное приложение».

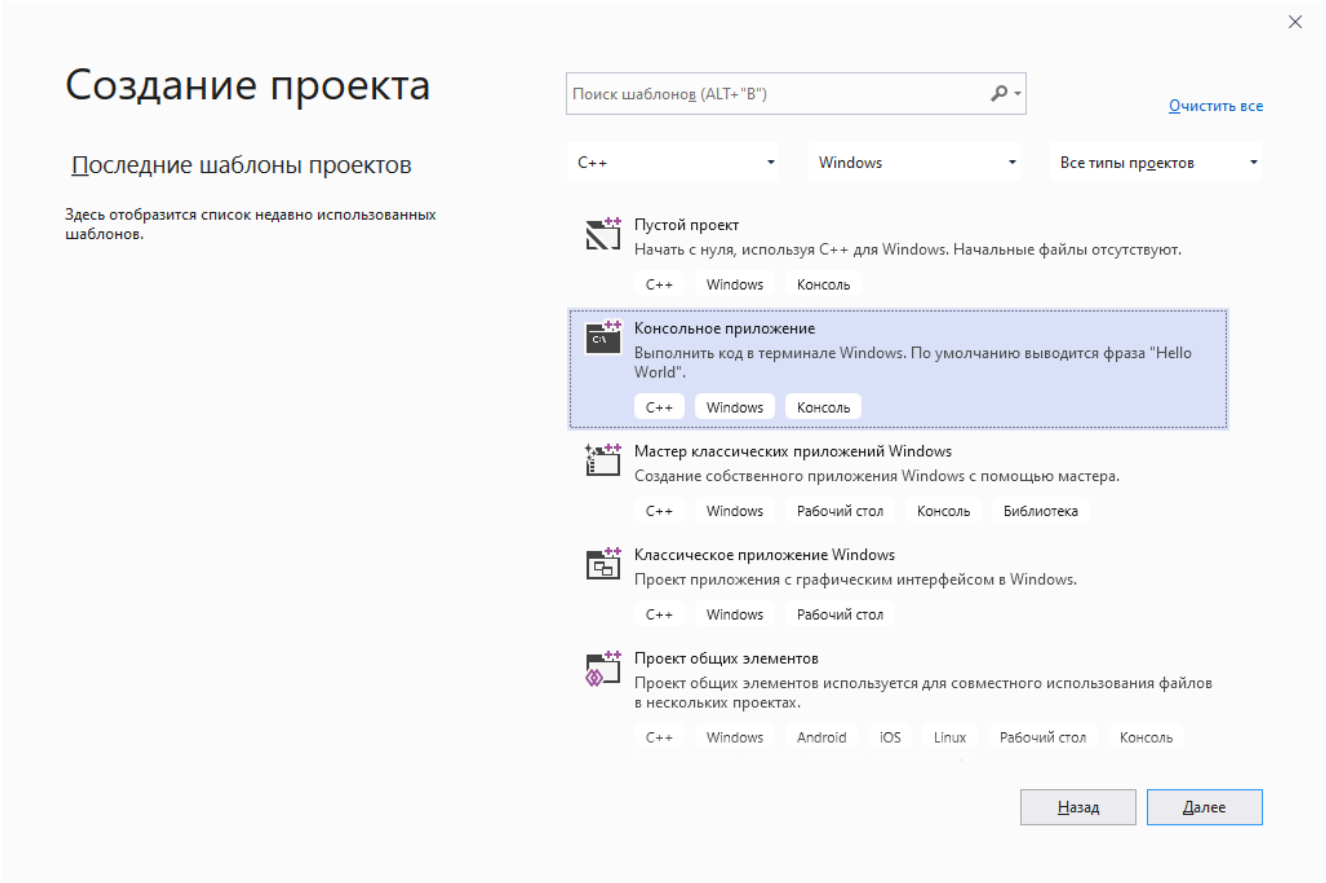


Рис. 3. Окно выбора типа проекта

После нажатия кнопки «Далее», появляется окно настройки проекта (рисунок 4).

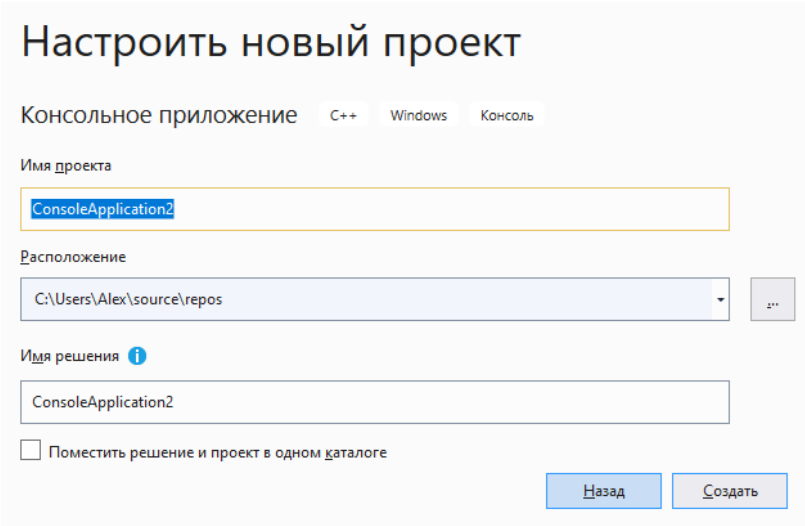


Рис. 4. Окно настройки проекта

Здесь необходимо ввести имя проекта, а также можно задать расположение папки проекта на диске ПК. После нажатия на кнопку

«Создать», проект будет создан. По указанному пути будет создана соответствующая папка, содержащая автоматически сгенерированный файл с простейшей программой, выводящий на экран строку: «Hello world!». Внешний вид рабочего окна Visual Studio с открытым проектом показан на рисунке 5.

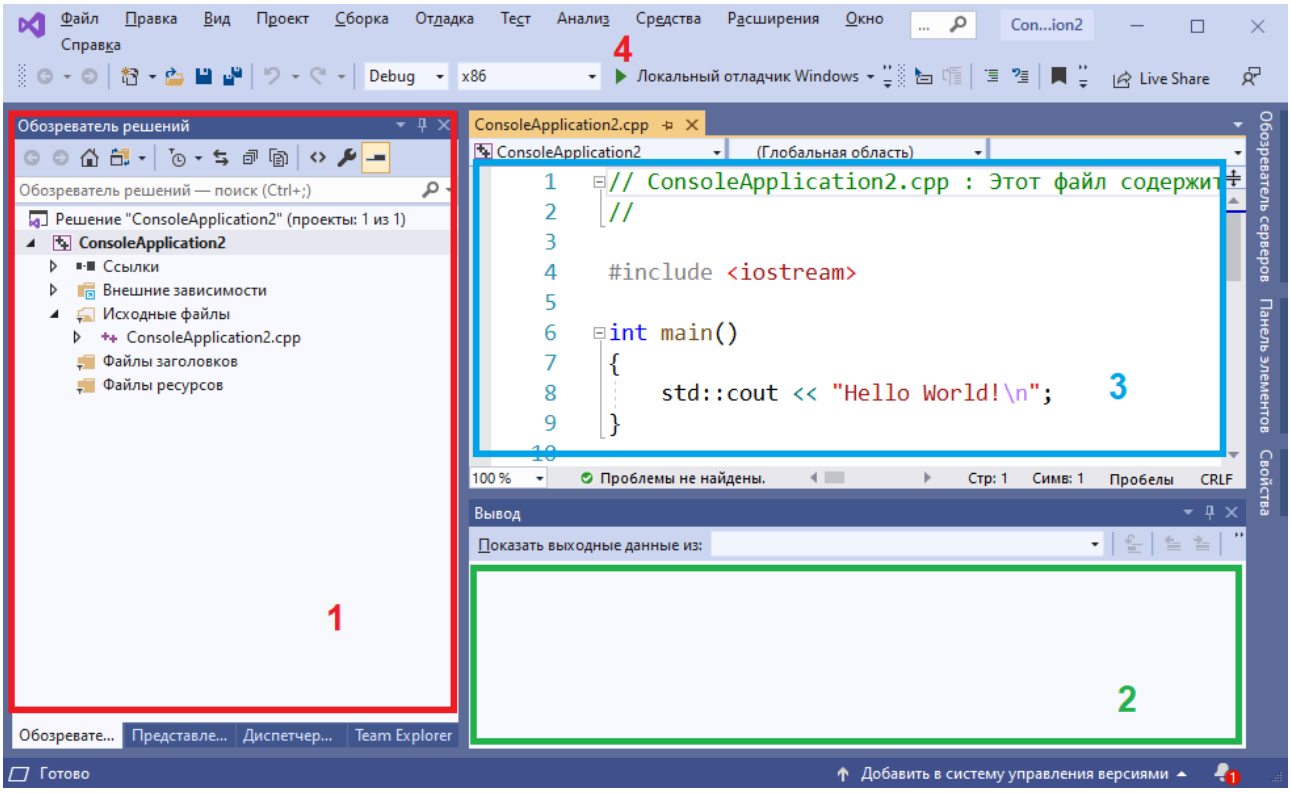


Рис. 5. Внешний вид *Visual Studio*

На рисунке 5 разными цветами выделены 3 области, важные для разработки ПО:

1. Обозреватель решений. Позволяет просматривать структуру и содержимое проекта. Здесь можно выполнять действия над исходными файлами: добавлять новые, открывать на редактирование существующие, удалять из проекта.
2. Окно вывода. Сюда выводятся предупреждения и сообщения об ошибках, возникших в процессе сборки.
3. Окно редактирования исходного кода. Текстовый редактор с подсветкой синтаксиса, подсказками, автодополнением и

другими возможностями, облегчающими работу программиста.

Также, цифрой 4 (рисунок 5), обозначена кнопка для сборки и запуска текущего проекта «Локальный отладчик Windows». Нажатие этой кнопки приводит к выполнению разрабатываемой программы или же ее отладке (рисунок 6).

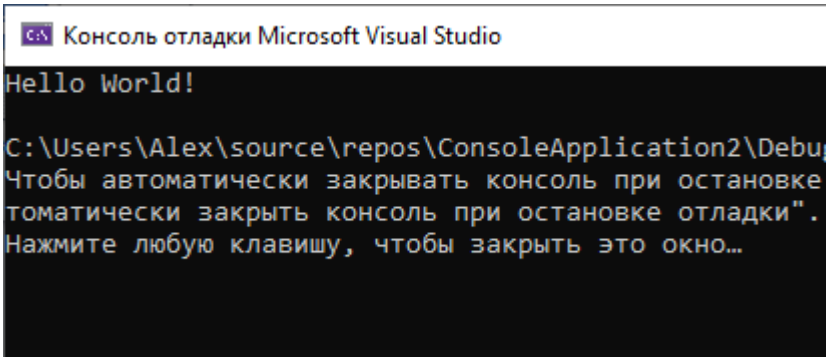


Рис. 6. Фрагмент окна командной строки после выполнения программы

1.5. Вопросы для самопроверки

1. Какие этапы сборки приложения вы знаете?
2. Для чего используется редактор связей?
3. Что такое «Интегрированная среда разработки»?
4. Что нужно сделать для создания нового консольного приложения с использованием *Visual Studio*?

1.6. Задания для самопроверки

1. Установите Visual Studio , если эта программа у вас еще не установлена.
2. Создайте проект типа «Консольное приложение» с именем MyFirstApp и расположенный в папке <Ваша_домашняя_папка>/Documents.
3. Запустите приложение и убедитесь, что оно работает.

4. Замените слово «World» на русское «Мир». Запустите приложение. Что отображается на экране?

2. ТИПЫ ДАННЫХ И РАБОТА С НИМИ

2.1. Работа с типами и их отличие

При выполнении программы, промежуточные результаты работы сохраняются в переменных. Переменная – это область памяти, имеющая имя и тип.

Обращение к участку памяти, с которым связана переменная, происходит по имени. Имена переменных определяет разработчик ПО, в процессе написания программы. Они должны быть «описательными», то есть нужно, чтобы по имени переменной можно было понять, для чего она предназначена. Определение имён переменных подчиняются правилу составления идентификаторов: любое символьное имя может состоять из латинских букв (верхнего и/или нижнего регистров), цифр и знака подчеркивания «_»; первым символом должна быть буква или знак подчеркивания. Регистр букв имеет значение: «abc» и «Abc» - это разные идентификаторы.

Тип переменной указывает размер, занимаемый в оперативной памяти и формат данных, определяющий, как компилятор должен интерпретировать содержимое. Язык C++ является строго типизированным. Это накладывает ограничения в виде необходимости объявлять тип каждого объекта, который создается в программе (числа с плавающей точкой, целые числа, кнопки, строки, окна, и т. д.). С другой стороны, компилятор позволяет избегать ошибок, которые связаны с присвоением переменным значений не того типа, который им соответствует. Тип переменной должен быть определен до присвоения ей значения. Общий вид объявления переменных следующий:

```
тип имя_переменной;
```

Пример:

```
int a;
```

здесь: `int` – тип переменной, `a` – имя переменной

Можно объявить сразу несколько переменных одного типа:

```
тип имя_перем1, имя_перем2, имя_перем3;
```

Пример:

```
int a, b, sum;
```

Также C++ позволяет при объявлении инициализировать переменные (то есть задавать им начальные значения):

```
int a = 1;
int b=2, c, sum=0;
```

Типы данных можно разделить на целочисленные, вещественные, символьные, логические (таблица 1). Целочисленные типы позволяют хранить только целые числа и, в свою очередь, бывают знаковые и беззнаковые. Беззнаковые типы позволяют хранить только положительные значения. В переменных знаковых типов можно хранить, как положительные, так и отрицательные значения, но диапазон у них в два раза меньше (это происходит потому, что старший бит отводится под знак: 0 – число положительное, 1 – число отрицательное).

Таблица 1

Встроенные типы данных

Тип данных	Диапазон	Размер в байтах
Знаковые целочисленные типы		
char	-128 .. 127	1
short	-32,768 .. 32,767	2
int	-2 147 483 648 .. 2 147 483 647	4
long	-2 147 483 648 .. 2 147 483 647	4
long long	-9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807	8

Тип данных	Диапазон	Размер в байтах
Беззнаковые целочисленные типы		
unsigned char	0..255	1
unsigned short	0..65535	2
unsigned int	0 .. 4,294,967,295	4
unsigned long	0 .. 4,294,967,295	4
unsigned long long	0 .. 18 446 744 073 709 551615	8
Вещественные типы		
float	-3,402823e38 .. 3,402823e38	4
double	-1,79769313486232e308 .. 1,79769313486232e308	8
Символьные типы		
char	один ASCII символ	1
wchar_t	расширенный символ	2
Логический тип		
bool	true или false	1

Размеры некоторых типов данных могут различаться на разных программных и аппаратных платформах. В таблице 1 приведены размеры для компьютеров, имеющих процессор семейства Intel и операционную систему Windows.

Также следует обратить внимание на тип `char`, который, с одной стороны, представляет собой целое однобайтовое число, а с другой стороны, может быть интерпретирован как символ – хранящееся в переменной типа `char` число представляет собой код символа в таблице ASCII.

2.2. Пространства имен

Пространство имен (*namespace*) – позволяет обеспечить уникальность всех имен, используемых в конкретной программе или проекте. Иногда программисту при работе над крупным проектом не хватает удобочитаемых глобальных имен или нужны библиотеки классов сторонних разработчиков, в которых конфликтуют имена. Тогда он может объявить собственные пространства имен и использовать их. Пространство имен объявляется следующим образом:

```
namespace Имя_пространства_имен {
    … область действия пространства имен…
}
```

Пример:

```
namespace MySpace {
        int myVar;
}
```

Здесь мы объявили пространство имен `MySpace` и внутри него поместили переменную `myVar`.

Доступ к идентификаторам внутри пространства имен осуществляется по схеме:

```
Имя_пространства_имен::имя_переменной
```

Например, для обращения к переменной `myVar` из примера выше будет происходить следующем образом:

```
MySpace::myVar = 3;
```

Разработчики стандартной библиотеки языка C++ создали специальное пространство имен **`std`** для типов данных, функций и других элементов, используемых в этой библиотеке. При обращении к ним необходимо каждый раз использовать префикс **`std::`** , например:

```
std::cout<<«Hello»;
```

Язык C++ позволяет объявить некоторое пространство имен, используемым по умолчанию и не добавлять его префикс далее в программе:

```
using namespace Имя_простр_имен;
```

Для того чтобы сделать стандартное пространство имен пространством по умолчанию, необходимо в программе набрать строку:

```
using namespace std;
```

После этой строки использовать префикс `std::` уже не обязательно.

2.3. Арифметические операции

Различают 5 операций, выполняющих арифметические действия (таблица 2) над числами.

Таблица 2

Арифметические операции

Операция	Назначение	Пример
+	Сложение	z = x + y
-	Вычитание	z = x - y
*	Умножение	z = x * y
/	Деление	z = x / y
%	Деление по модулю (остаток от деления)	z = x % y

Операция – это конструкция в языках программирования, которая аналогична по записи математическим операциям, то есть особый способ записи некоторых действий. Чаще других используются арифметические, логические и операции сравнения.

Операции делятся по количеству принимаемых аргументов на:

– унарные – один аргумент (отрицание, унарный минус)

– бинарные – два аргумента (сложение, вычитание, умножение и т.д.)
– тернарные – три аргумента («условие ? выражение1 : выражение2»)

К арифметическим унарным операциям в C++ относят инкремент и декремент: ++ и --

```
int n = 0;
++n; //  значение n стало 1, аналогично n = n + 1;
--n;  // значение n стало 0, аналогично n = n - 1;
```

Показанная выше форма называется префиксной, поскольку операция стоит перед переменной. Существуют также и постфиксные варианты инкремента и декремента:

```
int n = 0;
n++; //  значение n стало 1
n--;  // значение n стало 0
```

Разница между префиксной и постфиксной формой заключается в следующем: постфиксное выражение возвращает *старое значение* переменной и только потом увеличивает/уменьшает число. Это проявляется, когда постфиксное выражение является частью другого выражения. Старайтесь все время использовать префиксную форму и, только в случаях, когда это действительно нужно, постфиксную.

Основные арифметические бинарные операции представлены в таблице 2. Однако, язык C++ предоставляет их сокращенные варианты, для случаев, когда нужно изменить одну и ту же переменную:

```
int num = 0;
num += 5; // значение n стало 5, аналогично n = n + 5;
num *= 6; // значение n стало 30, аналогично n = n * 6;
num /= 2; // значение n стало 15, аналогично n = n / 2;
```

и т.д.

Тернарная операция в языке C++ всего одна, относится к операциям сравнения и будет рассмотрена позже.

2.4. Целочисленное деление

При выполнении арифметических операций особое внимание необходимо обратить на операцию деления одного целого числа на другое. В результате такой операции получается также *целое число*. Результат деления всегда округляется до *ближайшего наименьшего целого*.

Во избежание ошибки при вычислениях, использующих целые числа необходимо один из операндов привести к вещественному типу: если в выражении участвуют переменные целого и вещественного типа, результат будет вещественным.

Рассмотрим два примера. Первый пример – в выражении участвуют переменная и литерал (значение, прописанное непосредственно в коде программы):

```
1 int num = 10;
2 double res = num / 6; // значение res = 1, целочисленное деление
3 res = num / 6.0; // значение res = 1.6666
```

Литерал 6 имеет тип `int` (по правилу языка C++) и во второй строчке примера имеет место целочисленное деление: `10/6 = 1`. Результат деления имеет тип `int`. Потом этот результат приводится к типу `double` и в переменную `res` записывается значение `1.0`.

В строке 3 примера, литерал `6.0` имеет тип `double` (по правилу языка C++). Целочисленного деления не происходит и дробная часть не теряется.

Пример второй – в выражении участвуют только переменные:

```
1 int num1 = 10;
2 int num2 = 6;
3 double res = num1 / num2; // значение res = 1, целочисл. деление
4 res = num1 / (double)num2; // значение res = 1.6666
```

В строке 4 знаменатель явно приводится к типу `double` (для явного приведения одного типа в другой необходимо перед приводимым выражением в скобках указать желаемый тип данных).

Аналогично, для второго примера к типу `double` можно было бы привести и числитель.

2.5. Простейшая программа на C++

Рассмотрим простейшую программу на языке C++, выводящую на экран значение целочисленной переменной:

```
1 /* пример 1 */
2 #include <iostream>
3 using namespace std;
4 void main()
5 {
6     int year;
7     year = 2020;
8     cout << «Сейчас « << year << « год \n»;
9 }
```

Номера строк (слева) не являются частью программного кода и приведены здесь для простоты понимания.

Строка 1 – `/* пример 1 */` – является комментарием. Исходный код, находящийся внутри комментария не компилируется. Комментарии используются для добавления пояснений к коду программы или для исключения (возможно временного) части кода из работы программы. В данном случае, это пояснение.

Строка 2 – `#include <iostream>` – сообщает компилятору о необходимости подключить файл `iostream`. Этот файл содержит информацию, необходимую для правильного выполнения операций ввода/вывода языка C++. Язык C++ предусматривает использование специальных файлов, которые называются заголовочными файлами (часто имеют расширение `*.h`). В файле `iostream` находится информация о стандартном потоке вывода `cout`, который используется в программе.

Любая программа на языке C++ состоит из одной или нескольких функций. Выполнение программы начинается с вызова функции

`main()`. Каждая программа на языке C++ должна ее содержать. Разработчику необходимо самостоятельно написать содержимое этой функции. Строка 4 содержит объявление `main()`. Тело функции заключается в блок кода внутри фигурных скобок.

`{` – открывающаяся фигурная скобка, обозначает начало тела функции `main ()`. Фигурные скобки в языке C++ всегда используются парами (открывающаяся и закрывающаяся).

`int year;` – объявление переменной `year`, и сообщает компилятору, что эта переменная имеет тип `int`. Все переменные должны быть объявлены прежде, чем они будут использованы. Все операторы в языке C++ заканчиваются символом «точка с запятой».

`year = 2020;`– является оператором присваивания. В этой строке переменной с именем `year` присваивается значение 2020. В языке C++ оператор присваивания, это просто знак равенства.

`cout << «Сейчас « << year << « год \n»;` – является вызовом стандартного потока вывода `cout`, который выводит данные на экран. Эта строка состоит из имени потока `cout` и выводимых данных «`Сейчас`», `year` и `«год \n»,` разделенных символами направления потока вывода `<<`, которые помещают аргументы в выходной поток. В языке C++ нет операторов ввода/вывода, но библиотеки языка C++ содержат много полезных и удобных средств для этого.

Первое и третье выводимые данные `cout`– это строки «`Сейчас` » и « `год \n»`. Строка может содержать обычные символы и ESC-последовательности, начинающиеся с символа `«\»`. Обычные символы просто отображаются на экран в том порядке, в котором они следуют. ESC-последовательности вызывают определенные действия. Комбинация символов «`\n»` сообщает `cout` о необходимости перехода на новую строку. Этот символ называется символом новой строки. Другой способ обращения к этому символу – с помощью именной константы `endl`.

Вторым выводимым значением потока `cout` является значение переменной `year`.

`}` – последняя строка программысодержит закрывающуюся фигурную скобку. Она обозначает конец функции `main ()`.

Рассмотрим второй пример, в котором будет реализовываться ввод данных с клавиатуры. Для этого будет использоваться стандартный поток ввода `cin`, который позволяет пользователю вводить данные с клавиатуры во время выполнения программы.

Вычисление длины окружности по значению радиуса.

```
/* Вычисление длины окружности */
#include <iostream>
using namespace std;
void main()
{
    int radius;
    double length;
    cout << «Введите значение радиуса : «;
    cin >> radius;
    length = 2 * 3.1415 * radius;
    cout << «Радиус - « << radius << endl << « Длина - « << length;
}
```

Здесь объявлены две переменные двух разных типов: `radius` – типа целое (`int`); `length`– вещественного типа (`double`), содержащая дробную часть. Используется стандартный поток ввода `cin` для ввода с клавиатуры значения радиуса окружности. Операция `>>` извлекает из входного потока `cin` значение и присваивает его переменной `radius`. Для того, чтобы ею воспользоваться, подключен библиотечный файл math.h, содержащий информацию, необходимую для правильного выполнения математических функций языка C++.

2.5. Средства ввода-вывода

В C++ существуют специальные средства для выполнения операций ввода-вывода данных. Все операции ввода-вывода реализуются с

помощью функций, которые находятся в заголовочном файле <iostream>. Язык C++ поддерживает три уровня ввода-вывода:

– потоковый ввод-вывод;
– ввод-вывод нижнего уровня;
– ввод-вывод для консоли и портов (зависит от ОС).

Поток – это абстрактное понятие, которое можно отнести к любому последовательному переносу данных от источника к приемнику.

Поток определяется как последовательность байт и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер). Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти – буфер. Буфер накапливает байты, и фактическая передача данных выполняется после заполнения буфера (рис. 7). При вводе это дает возможность исправить ошибки, если данные из буфера еще не отправлены в программу.

Функции ввода-вывода языка C++, поддерживающие обмен данными с файлами в парадигме потока, позволяют обрабатывать данные различных размеров и форматов, обеспечивая при этом буферизованные операции ввода и вывода. Чтение данных из потока называется извлечением, вывод в поток – помещением (включением).

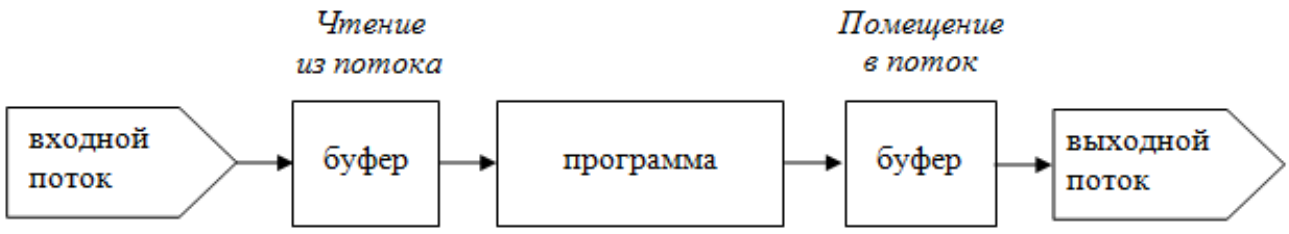


Рис. 7. Схема организации буферизации в операциях ввода-вывода

В момент начала выполнения программы, автоматически открываются пять потоков, из которых основными являются три:

– стандартный поток ввода (на него ссылаются, используя предопределенный указатель на поток `stdin`);
– стандартный поток вывода (`stdout`);
– стандартный поток вывода сообщений об ошибках (`stderr`).

По умолчанию стандартному потоку ввода `stdin` ставится в соответствие клавиатура, а потокам `stdout` и `stderr` соответствует экран монитора (окно консольного приложения).

Для работы со стандартными потоками ввода/вывода данных используются два оператора.

`cin` – оператор, который определяет стандартные потоки ввода данных.

`cout` – оператор, который определяет стандартные потоки вывода данных.

`<<` – операция записи данных в поток;

`>>` – операция чтения данных из потока.

Для использования этих операторов необходимо подключить файл заголовков `<iostream>`.

2.6. Вопросы для самопроверки

1. Для чего нужны типы данных?

2. В чем заключается правило формирования идентификаторов?

3. Что такое «литерал»?

4. Какой числовой диапазон входит в тип `usigned short`? `long long`?

5. Что такое пространство имен? Для какой цели служит пространство имен `std`?

2.7. Задания для самопроверки

1. Создайте новый проект типа «Консольное приложение».

2. Подключите библиотечный файл `math.h`, содержащий информацию, необходимую для правильного выполнения математических функций языка C++.

3. Разработайте программу, позволяющую вводить с клавиатуры 3 числа *A*, *B* и *C*. Программа должна вывести *cos*((*A***B*)/*C*).

4. Добавьте расчет (*exp*^*A*)/*B* и выведите полученное значение на экран.

5. Добавьте код, позволяющий пользователю ввести целое число и определяющий, четное оно или нет. Выведите на экран соответствующее сообщение.

3. УСЛОВНЫЕ ОПЕРАТОРЫ

3.1. Логический тип данных и логические операции

Логический, булевский (англ. *Boolean* или *logical data type*) тип данных – тип данных, переменные которого могут принимать два возможных значения: «истина» (`true`) и «ложь» (`false`). В C++ за истину также принимается целое число `1`, за ложь – `0`. Для объявления типа используется ключевое слово `bool`.

Тип данных `bool` получил своё название в честь английского математика и логика Джорджа Буля, среди прочего, занимавшегося вопросами математической логики в середине 19 века [5].

Традиционным применением логического типа данных являются значения «да»/»нет» в отношении результата более сложных операций.

Все операции сравнения двух величин (равно, больше, меньше возвращают в качестве результата логический тип. В языке *C++* применяются следующие операторы сравнения: < («меньше»), > («больше»), <= («меньше или равно»), >= («больше или равно»), == («равно»), != («не равно»).

```
bool isTrue = 4 > 5; // значение isTrue будет false
isTrue = 14 > 5;     // значение isTrue будет true
```

Две и более операции сравнения могут быть объединены в единое логическое выражение с помощью логических операций: «И», «ИЛИ», «НЕ». Обозначение и правила работы этих операций приведены в таблице 3.

Таблица 3

Логические операции

Логическая величина X	Логическая величина Y	Операция «И» X && Y	Опер. «ИЛИ» X || Y	Опер. «НЕ» !X
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Пример:

```
bool isTrue = 14 > 5 && 4 > 5; // значение isTrue будет false
```

3.2. Условный оператор if

Для организации ветвления кода используют языковую конструкцию (оператор) `if – else`.

Простейшая форма оператора `if` соответствует конструкции на рис. 8а и имеет вид:

```
if (условие) оператор
```

Условие – это логическое выражение, которое принимает значение либо «истина», либо «ложь». Оператор внутри `if` выполняется только в том случае, если условие истинно.

Оператор

```
if (0 < 1) cout << «0 меньше 1»;
```

выводит на экран сообщение: «0 меньше 1».

Оператор

```
if (0 > 1) cout << «0 больше 1»;
```

не выводит на экран никакого сообщения, т.к. значение выражения `0 > 1` ложно, и оператор, следующий за условием, не выполняется. Заметим, что при сравнении отношение равенства в языке C++ записывается двумя знаками равенства «==» (не путать с операцией присваивания «=»).

Оператор

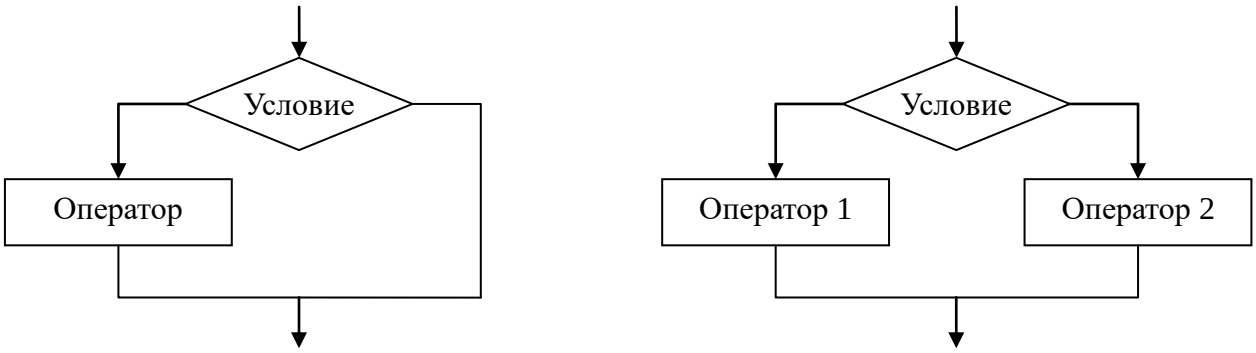
```
if (0 == 1) cout << «0 равно 1»;
```

не выводит на экран никакого сообщения, т.к. значение выражения `0 == 1` ложно.

Полная форма оператора `if` имеет вид:

```
if (условие) оператор1 else оператор2;
```

Если значение условия «истинно», то выполняется `оператор1`. Если же условие принимает значение «ложно», то выполняется `оператор2`. Полной форме оператора `if` соответствует конструкция, представленная на рис. 8б.



а б

Рис. 8. Конструкция условного оператора: а – неполной, б – полной формы

Пример: определить максимальное значение из трех целых чисел, вводимых с клавиатуры.

```
#include <iostream>

using namespace std;

void main()      // Поиск максимального значения
{                // из трех целых переменных a,b,c
    int a, b, c, m;
    cout << « Введите исходные данные : «;
    cin >> a >> b >> c;
    if (a > b) m = a; else m = b;
    if (c > m) m = c;
    cout << «максимум = «, m;
}
```

В данном примере продемонстрировано использование обеих форм оператора `if`.

`// <текст>` – эта часть строки демонстрирует еще один вид комментария, ограниченного пределами строки.

Пример: треугольник задан своими сторонами: a, b, c. Требуется найти площадь треугольника.

```
#include <iostream>
#include <math.h>

using namespace std;
/* Расчет площади треугольника по его сторонам a,b,c */
void main()
{
```

```
    double a, b, c, p, s;
    cout << « Введите исходные данные : «;
    cin >> a >> b >> c;
    if (a <= 0 || b <= 0 || c <= 0) {
        cout << «Ошибка.»;
        return;
    }
    if (a + b <= c) { cout << «Ошибка.»; return; }
    if (b + c <= a) { cout << «Ошибка.»; return; }
    if (a + c <= b) { cout << «Ошибка.»; return; }
    p = (a + b + c) / 2;    // вычисление полупериметра
    s = sqrt(p * (p - a) * (p - b) * (p - c)); // формула Герона
    cout << « Площадь треугольника = « << s;
}
```

Конструкция `{оператор1 оператор2 … операторN}` называется составным оператором, представляет собой блок кода размещенный внутри фигурных скобок и рассматривается как единый оператор. Эта конструкция позволяет в условном операторе if выполнить несколько действий.

В последнем примере также продемонстрировано использование стандартной математической функции вычисление квадратного корня `sqrt()`. Для того, чтобы ею воспользоваться, подключен библиотечный файл `math.h`, содержащий информацию, необходимую для правильного выполнения математических функций языка C++.

Логическое условие `a<=0 || b<=0 || c<=0` принимает значение «истинно», если будет истинным хотя бы одно из простых условий `a<=0, b<=0, c<=0`. Здесь использована логическая операция || («ИЛИ»).

3.2. Условный оператор switch

Иногда алгоритм содержит серию решений, проверяющих переменную или выражение отдельно для каждого постоянного целочисленного представления, которое может принять переменная или выражение. Постоянное целочисленное представление – это любое выражение, включающее в себя символ и целочисленные константы

(например, значения типа `int` или `char`). Затем алгоритм выполняет разные операции, исходя из своих значений. Для управления таким принятием решений в *C++* имеется оператор выбора `switch`.

В следующем примере предположим, что группа из студентов сдавала тест, и каждый студент получил некую оценку `A` или `B`. Программа просит ввести буквенные оценки и вычисляет результаты с помощью оператора `switch` для расчета количества появления каждой из них:

```
#include <iostream>

using namespace std;

void main()
{
    int aCount = 0, bCount = 0;
    char grade;
    int aCount = 0, bCount = 0;
    char grade;
    // ...
    cout << "Input 'a' or 'b': " << endl;
    cin >> grade;
    switch (grade)
    {
    case 'A':
    case 'a':
        aCount++;
        break;
    case 'B':
    case 'b':
        bCount++;
        break;
    default:
        cout << "Invalid character" << endl;
        break;
    }
    // ...
    cout << "aCount " << aCount << " bCount " << bCount << endl;
}
```

Когда поток управления достигает оператора `switch` программа оценивает управляющее выражение (в данном примере переменную `grade`) в скобках, следующих за ключевым словом `switch`. Значение этого выражения сравнивается со значением каждой метки `case`, до тех

пор, пока не будет найдено соответствие. Например, предположим, что пользователь ввел букву ‘B’ в качестве оценки студента. После этого введенное значение сравнивается с каждым вариантом case оператора switch до тех пор, пока в строке не будет иметь место соответствие (case ‘B’:). Теперь будут выполняться операторы для этого case. В случае буквы ‘B’ увеличивается количество оценок в переменной bCount, и оператор switch сразу завершается оператором break(оператор break обеспечивает передачу управления программы на первый оператор, следующий за оператором switch).

Если между значением управляющего выражения и любой меткой case не возникает соответствия, тогда выполняется секция default. В данном примере в ней отображается сообщение об ошибке. Обратите внимание, что секция default в операторе switch – не обязательная. Если управляющее выражение не совпадает с case, и нет случая default, тогда управление программы переходит к следующему оператору после оператора switch. Еще стоит обратить внимание, что в операторе switch могут выполняться операторы только для одного варианта case, после которого будет выполнен оператор break для текущего case, что обусловит немедленный выход из switch.

Каждый case может содержать одну операцию, несколько операций или вообще не иметь операций. Случай case без оператора называется пустым и может не содержать оператор break. Оператор break необходим для каждого случая case (включая default), только если в нем имеются операторы.

Последний вариант case в операторе switch не может быть пустым. Если управляющее выражение совпадает с пустым вариантом case, тогда будут выполняться операторы в следующем не пустом случае case. Такое поведение называется сквозным. Оно обеспечивает программиста способом указания выполнения определенных операторов для нескольких case. В коде продемонстрировано сквозное

поведение. Например, выполнится одинаковый код при введении оценок *A* или *a* (*B* или *b*).

При создании `case` рекомендуется проверять все возможные значения для подтверждения того, что в операторе `switch` не присутствуют два варианта `case` для одного проверяемого значения. Если какие-то значения одинаковы, то будет иметь место ошибка компиляции. Наконец, важно отметить, что оператор `switch` отличается от других управляющих структур тем, что в `case` множественные операции не обязательно заключаются в фигурные скобки.

Несмотря на то, что варианты `case` в операторе `switch` могут встречаться в любом порядке, хорошим тоном программирования считается размещение случая `default` в последнюю очередь. При использовании оператора `switch` помните, что выражение после каждого `case` в конкретном операторе `switch` должно быть либо вычисляемым значением, либо строкой. Символьная константа представлена как специальный символ в одинарных кавычках (например, `'A'`). Целочисленная константа – это просто целочисленное значение. Выражение после каждого случая `case` также может быть константой – переменной, содержащей значение, которое не меняется на протяжении выполнения всей программы.

3.3. Тернарная операция

Тернарная операция – это единственная операция в языке C++, имеющая три операнда. Ее использование напоминает использование условного оператора. Дело в том, что оператор if не может возвращать никакого результата и не может быть частью какого-либо выражения. Тернарная операция дублирует функционал оператора if, с той разницей, что она, как правило, является частью выражения и результатом ее работы будет некоторое значение. В общем случае ее синтаксис выглядит следующим образом:

```
условие ? выраж_если_истина : выраж_если_ложь
```

здесь:

`условие` – логическое выражение;

`выраж_если_истина` – произвольное выражение, которое будет выполнено, если результатом `условия` будет «истина»; результат всей операции – значение этого выражения;

`выраж_если_ложь` – произвольное выражение, которое будет выполнено, если результатом `условия` будет «ложь»; результат всей операции в этом случае – значение этого выражения.

Пример:

```
int a;
int b;
  . . .
int c = 10 * ( (a < b)?(a + b):(a * b) );
```

В данном примере, в случае, если `a < b` , то `c = 10 * (a+b)`,

иначе `c = 10 * (a*b).`

3.4. Вопросы для самопроверки

1. Как определяется логический тип в языке C++?

2. Приведите примеры наиболее частого использования булевского типа.

3. Является ли выражение `0` с точки зрения компилятора *C++* одинаковым с `true`?

4. Для чего используют оператор `if – else` ?

5. Можно ли внутрь скобок выражения `if` передавать операции с типом результата отличного от `bool`?

6. Какие элементы выражения `if – else` являются обязательными? Когда фигурные скобки можно опустить?

7. В чем отличие оператора `switch` от конструкции `if – else`?

8. Когда оператор `break` в конструкции `switch` обязателен?

3.5. Задания для самопроверки

1. Выполните индивидуальные задания из лабораторной работы №1 (приложение 1).

4. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ

4.1. Циклы с условием

В программировании цикл – это повторяющееся выполнение одних и тех же простых или составных операторов. Алгоритмы, содержащие циклы, называются циклическими.

Существует несколько типов операторов цикла. Рассмотрим первый из них – оператор цикла с предусловием, или оператор цикла «пока». С использованием псевдокода, этот оператор записывается так:

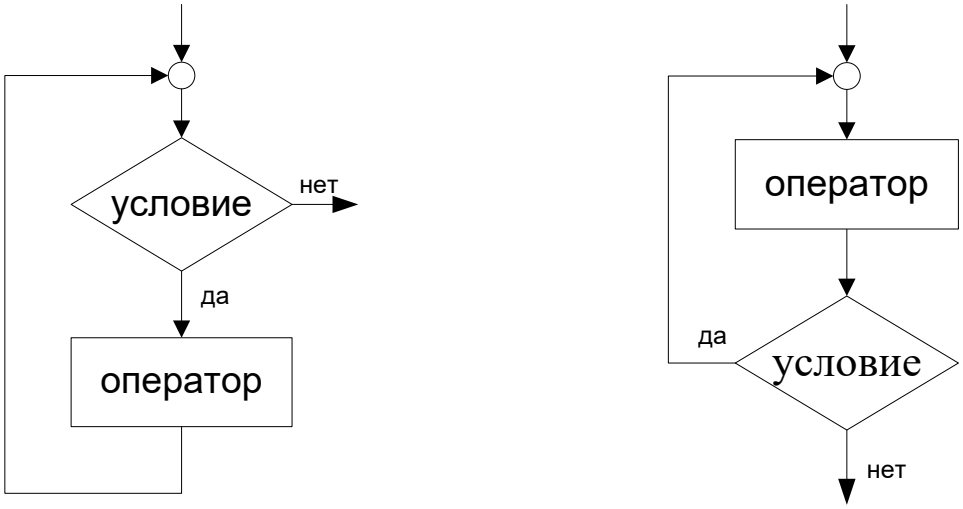
```
пока условие
 повторить
      оператор
```

Оператор (простой или составной), стоящий после служебного слова «повторить» и называемый телом цикла, будет выполняться циклически, пока значение «условия» равно «истина». Само условие цикла может быть выражением с логическим результатом, переменной или логической константой.

В языке C++ оператор цикла «пока» имеет вид:

```
while (условие) оператор
```

Конструкция, соответствующая циклу «пока», приведена на рисунке 9а.



а б

Рис. 9. Конструкция оператора цикла
а – с предусловием; б – с постусловием

Условие выполнения тела цикла “пока” проверяется до начала каждого его выполнения (итерации). Отсюда и название – оператор цикла с предусловием. Если условие сразу не выполняется, то тело цикла игнорируется и будет выполняться оператор, стоящий сразу за телом цикла.

Рассмотренный оператор цикла с предусловием позволяет до первой итерации решить: выполнять ли тело цикла или нет. Если по логике алгоритма необходимо выполнить тело цикла хотя-бы один раз, можно использовать оператор цикла с постусловием, который сначала выполняет итерацию, а затем решает продолжать ли выполнение цикла. Такой оператор называется оператором цикла “до”. Это имеет принципиальное значение лишь на первой итерации, а далее циклы ведут себя идентично. Цикл с постусловием на псевдокоде имеет вид:

```
повторить
   оператор
до  условие
```

В языке C++ оператору с постусловием соответствует оператор

```
do {
      операторы
} while (условие);
```

Оператор `do-while` называется оператором цикла с постусловием. Оператору `do-while` соответствует конструкция, представленная на рисунке 9б.

4.2. Циклы с параметром

Для организации циклов, которые должны быть проделаны заданное число раз используются циклы с параметром. Операторы циклов с пред- и постусловием, хотя и обладают значительной гибкостью, не слишком удобны в этих случаях. В цикле с параметром для каждого значения параметра выполняется тело цикла (очередная итерация). Пределы изменения параметра и способ получения следующего значения параметра по текущему указываются в заголовке оператора цикла. На псевдокоде оператор цикла с параметром имеет вид:

```
для  инициализация, условие, изменение
 повторить
      оператор
```

где “инициализация” используется для присвоения начального значения параметру цикла, “условие” определяет, выполнять тело цикла (оператор) или завершить цикл, “изменение” определяет способ изменения параметра цикла. Выполнение цикла происходит до тех пор, пока “условие” истинно. Как только условие становится ложным, начинает выполняться следующий за циклом оператор. Проверка “условия” происходит до выполнения тела цикла, т.е. этот оператор является частным случаем оператора цикла с предусловием. “Изменение” параметра цикла осуществляется после очередного выполнения тела цикла (в конце итерации). Если обозначить параметр цикла через i, начальное значение через $i_н$, конечное $i_к$, а изменение параметра $i_ш$, $i = i+i_ш$, то оператор можно записать так:

```
для i = iн, i≤iк, i = i+iш
      повторить
            оператор
```

или

```
для i = iн, iк, iш
      повторить
            оператор
```

Последнюю форму записи оператора цикла с параметром используют в случаях, когда заданы начальное и конечное значения параметра цикла, а шаг изменения его постоянный. Графическая схема оператора цикла с параметром определяется конструкциями, приведенными на рисунке 10.

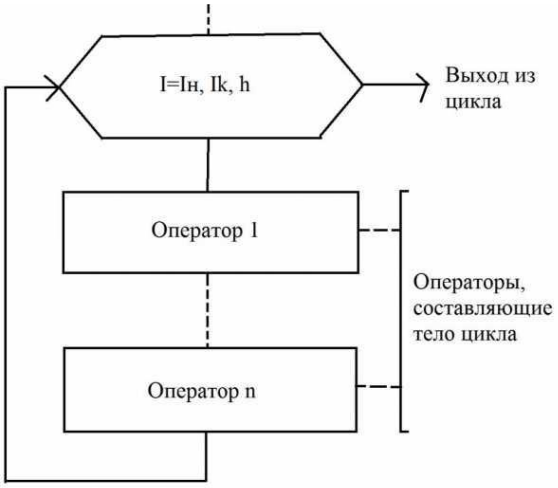


Рис. 10. Конструкция оператора цикла с параметром

На языке C++ этот оператор имеет следующий вид:

```
for (инициализация; условие; изменение)
      оператор
```

или в общем виде:

```
for (выражение1; выражение2; выражение3)
      оператор
```

Оператор цикла с параметром, соответствующий частному случаю, выглядит так:

```
for (i = iн; i <= iк; i = i + iш)
      оператор
```

Пример 1. Вывести на экран значения кубов натуральных чисел от 1 до 10.

Программа с использованием цикла `while`.

```
    int i; // описание переменной, отвечающей за число
    i = 1; // инициализация
    while (i <= 10)// проверка условия
    {
       // вывод на экран значения числа и его куба
        cout << i << " " << pow(i, 3) << endl;
        i++; // увеличение значения числа
    }
```

Программа с использованием цикла `for`.

```
    int i; // описание переменной-счетчика
// задание начального и конечного значений счетчика и шага изменения
    for (i = 1; i <= 10; i++)
    {
       // вывод на экран значения числа и его куба
        cout << i << " " << pow(i, 3) << endl;
    }
```

Результат работы обеих программ идентичен, однако код с использованием цикла `for` является более компактным и понятным для восприятия.

Пример 2. Написать программу вычисления факториала f натурального числа n по формуле $f = n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$, где n вводится с клавиатуры.

Псевдокод алгоритма

```
Алгоритм факториал
начало
  скаляры n, f, k- целые
  ввод (n)
  f=1
  дляk=1, n, 1
            повторить
      f=f*k
  вывод (f)
конец
```

Программа на языке С++

```
#include <iostream>
using namespace std;
void main()
{
      int k, n;
      double f;
      cout << "Input number : ";
      cin >> n;
      f = 1;
      for (k = 1; k <= n; k++)
            f = f * k;
      cout << "факториал = " << f;
}
```

Пример 3. С клавиатуры вводятся целые числа. Суммировать числа, пока пользователь подтверждает продолжение ввода. Считать подтверждением продолжения ввода символ ‘y’ (yes).

Программа на языке С++:

```
#include <iostream>
using namespace std;
void main()
{
    // описание переменных и инициализация значения суммы
    int number, summ = 0;
    // символьная переменная для подтверждения продолжения ввода
    char ch;
    do // начало цикла с постусловием
    {
        cout << "Input number: "; // запросчисла
        cin >> number; // вводчисла
        summ += number; // операциясложения
        // запрос на подтверждение продолжения вввода
        cout << "Continue? ";
        cin >> ch; // считывание символа
    } while (ch == 'y'); // проверка условия
    // вывод результата:
    cout << "Summis " << summ << endl;

}
```

Пример 4. На интервале $[a, b]$ с шагом dx вычислить значения кусочно-заданной функции $F(x)$

$$F(x) = \begin{cases} \dfrac{6-x}{5x^2}, x < -3, \\ \sin 2x, -3 \le x \le 3, \\ 3, x > 3. \end{cases}$$

Значения a, b, dx вводятся пользователем. При выводе результатов провести выравнивание при помощи манипуляторов.

Программа на языке С++.

```
#include<iostream>
#include<iomanip> // Библиотека манипуляторов
#include<math.h> // Математическая библиотека
using namespace std;
int main()
{
    double a, b, x, dx, f;
    cout << left; // выравнивание по левому краю
    cout << "Input a: "; // Запрос начала интервала
    cin >> a;              // Ввод начала интервала
    cout << "Input b: "; // Запрос конца интервала
    cin >> b;              // Ввод конца интервала
    cout << "Input dx: "; // Запрос шага изменения аргумента
    cin >> dx;         // Ввод шага изменения аргумента
```

```
    //Проверка корректности ввода значений
    if ((a <= b) && (dx > 0))
    {
        // Вывод заголовка
        cout << endl << setw(15) << "Results" << endl << endl;
        // Использование манипуляторов для выравнивания:
        cout << left << setw(15) << "x" << setw(15) << "F(x)" << endl;
        for (x = a; x <= b; x += dx)
        {
            if (x < -3)
                f = (6 - x) / (5 * pow(x, 2));
            else
                if (x <= 3)
                    f = sin(2 * x);
                else
                    f = 3;
            // Вывод результатов на экран
            cout << setw(15) << x << setw(15) << f << endl;
        }
    }
    else
        cout << "Incorrect data" << endl;
    return 0;
}
```

Для вычисления значения функции необходимо подключить заголовочный файл `<math.h>` или `<cmath>`, которые разработаны для выполнения математических операций. Большинство функций привлекают использование чисел с плавающей точкой. Все эти функции принимают значения `double`, если не определено иначе. Для работы с типами `float` и `long double` используются функции с постфиксами `f` и `l` соответственно. Все функции, принимающие или возвращающие угол, работают с радианами.

Также в программе используются манипуляторы вывода из заголовочного файла `<iomanip>`, в частности, выравнивание по левому краю `left` и установка ширины поля вывода `setw()`.

4.3. Безусловные циклы

Часто в программах применяются циклы, выход из которых не предусмотрен программной логикой. Такие циклы называются безусловными, или бесконечными. Специальных синтаксических конструкций для создания бесконечных циклов, ввиду их узкой направленности, языки программирования не предусматривают, поэтому такие циклы создаются с помощью конструкций, определённых для создания классических (или условных) циклов. Для обеспечения бесконечного повторения проверку условия в таком цикле либо убирают, либо заменяют константным значением:

```
while (true)
    cout << "not enouth..." << endl;

for (;;)
    cout << "not enouth2 ..." << endl;
```

4.4. Вопросы для самопроверки

1. Какие виды циклов бывают? Приведите пример кода на языке C++, описывающего каждый из видов циклов.

2. Для какого типа цикла не предусматриваются специальные синтаксические средства в языках программирования?

3. Можно ли в условии продолжения цикла использовать логические операции «И», «ИЛИ», «НЕ»?

4. Может ли условие продолжения цикла представлять собой целочисленное выражение?

4.5. Задания для самопроверки

1. Выполните индивидуальные задания из лабораторной работы №2 (приложение 1).

5. ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ

5.1. Понятие массива

Основная цель любой программы состоит в обработке данных. Данные различного типа хранятся и обрабатываются по-разному. В любом алгоритмическом языке каждая константа, переменная, результат вычисления выражения или функции должны иметь определенный тип.

Все типы языка C++ можно разделить на основные и составные. В языке C++ определены основные типы данных для представления целых, вещественных, символьных и логических величин (таблица 1). На основе этих типов программист может вводить описание составных типов. К ним относятся массивы, перечисления, структуры, ссылки, указатели, объединения и классы.

При обработке данных достаточно часто приходится создавать некоторое количество (иногда большое) переменных одинакового типа (и описывающих одинаковые объекты). В этом случае эти переменные имеет смысл объединить одним идентификатором. Это позволяют сделать массивы.

Массив – это поименованный набор однотипных компонентов (элементов массива), расположенных в памяти непосредственно друг за другом.

Массивы обладают рядом достоинств:

- лёгкость вычисления адреса элемента по его индексу (поскольку элементы массива располагаются один за другим);
- одинаковое время доступа ко всем элементам;
- малый размер элементов: они состоят только из информационного поля.

Из объявления массива компилятор должен получить информацию о типе элементов массива и их количестве. Каждый элемент мас-

сива определяется именем массива и порядковым номером элемента, который называется индексом. Индекс в языке C++ – всегда целое число. Индекс в языке C++ указывается в квадратных скобках после имени массива. Количество используемых индексов в массиве определяет его размерность. Если в массиве один индекс, то это одномерный массив. Основная форма объявления (описания) одномерного массива имеет вид:

```
тип  имя_массива [размер];
```

где `тип` – тип элементов массива, `размер` – количество элементов одномерного массива.

Размер массива в языке C++ может задаваться константой или выражением, состоящим из констант. Нельзя задавать массив переменного размера. Для этого существует отдельный механизм, называемый динамическим выделением памяти.

Примеры описания массивов:

```
const int  N = 15;
double a[10], arr[N];
```

В языке C++ индекс всегда начинается с нуля, под первым элементом массива подразумевается элемент с индексом 0.

Если объявлен массив:

```
int x[100];
```

то это означает, что массив содержит 100 элементов от `x[0]` до `x[99]`. Для одномерного массива легко подсчитать, сколько байт в памяти будет занимать этот массив:

```
колич.байт = размер_типа_элемента*колич.элементов
```

В языке C++ под массив всегда выделяется непрерывное место в оперативной памяти.

В языке C++ не проверяется выход индекса за пределы массива. Если массив `x[100]` описан как целочисленный массив, имеющий 100 элементов, а в программе указан элемент `x[200]`, то сообщение об ошибке не будет выдано, а в качестве значения элемента `x[200]` будет выдано некоторое число, занимающее 2 байта.

Используя циклы, можно осуществить перебор всего массива и через индексы обратиться к его элементам. Наиболее пригодными для решения этой задачи являются циклы со счетчиком.

Пример 1. Осуществить ввод элементов целочисленного массива с клавиатуры. Вывести значения элементов массива на экран в строку через пробел.

Программа на языке C++.

```
#include<iostream>
#include <locale.h>
using namespace std;
int main()
{
    setlocale(0, "");
    const  int  N = 10; // размер массива
    double  a[N]; // описание массива
    int i; // переменная-счетчик
    cout << "Введите " << N << " чисел" << endl;
    for (i = 0; i < N; i++)
        cin << a[i]; // ввод элементов массива
    cout << "Массив" << endl; // вывод заголовка
    for (i = 0; i < N; i++)
        cout << a[i] << " "; // // вывод элементов
    cout << endl;
}
```

В программе продемонстрировано использование при выводе символов русского алфавита (кириллицы). Для этого подключен заголовочный файл локализации `locale.h`, который содержит функции и классы для потоковой обработки данных в форме естественной для разных языков (денежный формат, представление символов, сортировка строк). Региональные настройки (локаль) содержит информацию о том, как интерпретировать и выполнять определенные опера-

ции ввода/вывода и преобразования с учетом специфики языков в определённых условиях.

Функция `setlocale()` задает региональные настройки, которые будут использоваться текущей программой. Можно изменить все параметры локали, или конкретные её части. Эта функция также может быть использована для получения имени текущей локали, передав `NULL` в через параметр `locale`.

Использование русского алфавита осуществляется вызовом функции `setlocale(LC_ALL, "Russian")`, для русифицированных версий операционной системы можно опустить последний аргумент функции и заменить на `setlocale(0, "")`.

5.2. Генерация случайных чисел

Элементы массива могут задаваться в зависимости от условий задачи различными способами:

- вводиться с клавиатуры;
- рассчитываться по формуле;
- генерироваться из некоторого диапазона.

Случайные числа в языке программирования C++ могут быть сгенерированы функцией `rand()` из стандартной библиотеки C++. Функция `rand()` генерирует числа в диапазоне от `0` до `RAND_MAX`. `RAND_MAX` – это константа, определённая в библиотеке `<stdlib.h>`. Для инициализации генератора случайных чисел используется функции `srand()`.

Зачастую, не нужен такой большой диапазон чисел от `0` до `RAND_MAX`. Например, имитируя бросание игрального кубика необходимо получить число от 1 до 6, то есть генерация чисел должна выполняться в пределе от 1 до 6-х. Бросая монету, может возникнуть только два случая, когда монета упадёт «орлом» или «решкой» вверх,

нужный интервал – от 1 до 2. Для того чтобы масштабировать интервал генерации чисел нужно воспользоваться, операцией нахождения остатка от деления «%». Например, какое бы не выдал число генератор случайных чисел `rand()`, запись `rand() % 3` в итоге выдаст число из диапазона от 0 до 2. Для того чтобы сместить диапазон, прибавляется единица: `rand() % 3 +1` тогда диапазон изменится от 1 до 3 включительно.

Однако, при повторном запуске программы, генератор будет выдавать те же самые случайные числа. Дело в том, что функция `rand()` генерирует случайные числа один раз, а при последующих запусках программы всего лишь возвращает уже сгенерированный набор. Такая особенность функции `rand()` нужна для того, чтобы можно было правильно отладить разрабатываемую программу. При отладке программы, внеся какие-либо изменения, необходимо удостовериться, что программа срабатывает правильно, а это возможно, если входные данные остались те же (берем ранее сгенерированные числа). После того, как программа успешно отлажена, нужно, чтобы при выполнении каждый раз генерировались новые случайные числа. В этом случае для инициализации генератора случайных чисел, часто используют функцию `time()` с аргументом `NULL` как параметр функции `srand()`. Чтобы использовать функцию `time()`, необходимо подключить заголовочный файл `<time.h>`.

Пример 2. Заполнить массив 20 случайными целыми числами из диапазона от 0 до 99. Вычислить сумму его элементов.

Программа на языке C++.

```
#include <iostream>
#include <time.h>
#include <stdlib.h>
#include <locale.h>

using namespace std;

int main()
{
```

```
    setlocale(0, "");
    const int N = 20; //размер массива
    int i, Array[N], Summ;
    //инициализируем генератор
    //случайных чисел:
    srand(time(NULL));
    // генерация элементов массива
    for (i = 0; i < N; i++)
        Array[i] = rand() % 100;
    // Вывод элементов массива
    cout << "Сгенерированный массив" << endl;
    for (i = 0; i < N; i++)
        cout << Array[i] << " ";
    cout << endl;
    // вычисление суммы и вывод результата на экран
    Summ = 0;
    for (i = 0; i < N; i++)
        Summ += Array[i];
    cout << "Summ = " << Summ << endl;

   //не дает программе завершиться немедленно:
    system("pause");
    return 0;
}
```

5.3. Вопросы для самопроверки

1. В каких случаях целесообразно использовать массивы?

2. Чему равен индекс последнего элемента массива?

3. Что произойдет в случае обращения к элементу массива по индексу, большему чем длина массива?

4. Может ли массив содержать элементы различных типов?

5.4. Задания для самопроверки

1. Выполните индивидуальные задания из лабораторной работы №3 (приложение 1).

6. УКАЗАТЕЛИ. ДИНАМИЧЕСКИЕ МАССИВЫ

6.1. Указатели

При обработке оператора определения переменной, например,

```
int a = 10;
```

компилятор в памяти выделяет участок памяти в соответствии с типом переменной и записывается в этот участок указанное значение. Все обращения к переменной `a` компилятор заменит на адрес оперативной памяти, в которой хранится эта переменная. Операция `&a` является операцией взятия адреса ее операнда.

Программист может определить собственные специальные переменные, в которых будут храниться адреса областей памяти, а не значения. Такие специальные переменные называются указателями.

Указатель – именованный объект, предназначенный для хранения адреса области памяти (объекта, непоименованной области оперативной памяти либо точки входа в функцию).

Указатель не является самостоятельным типом, он всегда связан с каким-то другим типом.

В общем случае синтаксис определения указателя на объект:

```
Тип *Описатель;
```

При определении указателя задается имя указателя и тип объекта, на который он ссылается. `Тип` задает тип объекта, адрес которого будет содержать определяемая переменная и может соответствовать базовому, пустому (`void`), перечислению, структурному типу и типу объединения. Реально указатель на `void` ни на что не указывает, но обладает способностью указывать на область любого размера после приведения его к типу другого объекта.

`Описатель` – это идентификатор, определяющий имя объявляемой переменной типа указатель или конструкция, которая организует непосредственно доступ к памяти. Описателю обязательно должна предшествовать звездочка (`*`).

Знак '`*`' является унарной операцией косвенной адресации, его операнд – указатель, а результат – адрес объекта, на который указывает операнд. Адресация является косвенной, так как обращение к области памяти осуществляется не напрямую по адресу (например, `1A2B`), а через объект, которому в памяти соответствует определенный участок. Объем памяти, который выделяется для хранения данных, определяется типом данных и моделью памяти. Для приведенной на рисунке 11 модели памяти адресом переменной типа `float` с именем summa является `0012FF48`, адресом переменной типа `int` с именем `date` является `0012FF54`, адресом переменной типа `char` с именем `ch` является `0012FF63`.

Машинный адрес	0012FF48 байт	0012FF49 байт	0012FF4A байт	0012FF4B байт	0012FF54 байт	0012FF55 байт	0012FF56 байт	0012FF57 байт	0012FF63 байт
Значение в памяти	$2.015*10^{-6}$				1937				'G'
Имя	summa				date				ch

Рис. 11. Пример модели памяти.

6.2. Способы инициализации указателя

Существует 4 основных способа инициализации указателей.

1. Присваивание указателю адреса области памяти существующего объекта:

   a. с помощью операции получения адреса:

```
int a = 5;
int *pa = &a;
```

   b. с помощью проинициализированного указателя

```
int *pb = pa;
```

2. Присваивание указателю адреса области памяти в явном виде:

```
char* cp = (char*)0x B8000000;
```

где `0xB8000000` – шестнадцатеричная константа, (`char*`) операция приведения типа (в практических задачах явное присваивание адреса указателю не используется).

3. Присваивание указателю нулевого (пустого) значения:

```
int *pN = NULL; //или
int *pN = 0;
```

Этот вариант используется для того, чтобы отложить инициализацию способами 1 и 4 до нужного момента. Обращение по нулевому указателю гарантированно вызовет ошибку и аварийное завершение программы, что даст возможность разработчику найти это место в программе и исправить ошибку.

4. Динамическое выделение памяти.

С указателями можно выполнять следующие операции:

- разыменование (`*`) – получение значения величины, адрес которой хранится в указателе;
- взятие адреса (`&`);
- присваивание;
- арифметические операции
- сложение указателя только с константой,
- вычитание: допускается разность указателей и разность указателя и константы,
- инкремент (`++`) увеличивает значение указателя на величину `sizeof(тип)`;
- декремент (`--`) уменьшает значение указателя на величину `sizeof(тип)`;
- сравнение;
- приведение типов.

Для взятия адреса переменной также используется оператор &, размещаемый перед объектом, адрес которого хочется получить.

Пример:

```
int a = 15;
// выводится адрес переменной, а не её значение:
cout << &a << endl;
```

Для перехода по известному адресу используется оператор *, размещаемый перед адресом или указателем хранящем адрес. Под переходом по адресу понимается, что от адреса мы переходим к действиям над значением, хранимом по данному адресу. Это операция называется иногда разыменованием.

```
int a = 15; // переменная a со значением
int* p = &a; // указатель с адресом переменной a
cout << *p << endl; // увидим 15, т.е. значение переменной
```

Таким образом, к любой переменной можно обратиться как по её имени, так и по её адресу, применив к нему операцию разыменования. Справедливо тождество: выражение `a == *(&a)`, где `a` – любой тип данных.

Существует три основных области хранения информации в оперативной памяти. В первой хранятся глобальные переменные. Выделяемые под них поля памяти остаются неизменными на все время выполнения программы. Под локальные переменные программа отводит память в специальной области, называемой стеком. Это вторая область хранения данных. Для размещения данных в стеке требуется заранее (в момент написания программы) знать объем памяти, выделяемой для каждой ситуации.

Третья область, в которой может храниться информация, называется «кучей» (heap). Работа с ней заключается в использовании системы динамического выделения памяти. При этом, память распределяется из свободного пространства кучи по мере необходимости. Об-

ласть свободной памяти находится между кодом программы с ее постоянной областью памяти и стеком (рис. 12).

Рис. 12. Схематичное представление областей памяти, выделяемой программе

Динамическое распределение памяти – способ выделения оперативной памяти компьютера для объектов в программе, при котором выделение памяти под объект осуществляется во время выполнения программы. При конструировании объекта указывается размер запрашиваемой под объект памяти, и, в случае успеха, выделенная область памяти, образно говоря, «изымается» из «кучи», становясь недоступной при последующих операциях выделения памяти. Противоположная по смыслу операция – освобождение занятой ранее под какой-либо объект памяти: освобождаемая память, также образно говоря, возвращается в «кучу» и становится доступной при дальнейших операциях выделения памяти.

В языке программирования C++ для динамического распределения памяти существуют операции `new` и `delete`. Эти операции используются для выделения и освобождения блоков памяти. Область памяти, в которой размещаются эти блоки, называется свободной памятью.

Операция new позволяет выделить и сделать доступным свободный участок в основной памяти, размеры которого соответствуют типу данных, определяемому именем типа.

Синтаксис:

```
переменная_указатель = new ИмяТипа;
```

Операцию delete следует использовать только для указателей на память, выделенную с помощью операции new.

```
delete переменная_указатель;
```

Пример.

```
#include <iostream>
using namespace std;
int main() {
    int* pa, * pb;
    pa = new int; // выделяется 4 Б из «кучи»
    *pa = 21;
    pb = pa; // pb и pa указывают на одну область памяти
    cout << *pa << "\t" << *pb << "\t" << pa << "\t" << pb << endl;
    pb = new int; // выделяется 4 Б из «кучи»
    *pb = 28;
    cout << *pa << "\t" << *pb << "\t" << pa << "\t" << pb << endl;
    //освобождаем pb
    delete pb;
    cout << *pa << "\t" << *pb << "\t" << pa << "\t" << pb << endl;
    //освобождаем pa
    delete pa;

    system("pause");
    return 0;
}
```

6.3. Связь указателей с массивами

При создании любого массива в C++, вместе с ним естественным образом создаётся константный указатель. Имя этого указателя совпадает с именем массива. Тип этого указателя – «указатель на базовый тип массива». В появившемся указателе хранится адрес начального элемента массива.

С учётом того, что в массиве все элементы располагаются в памяти последовательно, начав с указателя, направленного на начальный элемент, мы сможем обойти все элементы массива, смещая указатель на каждом шаге вправо на минимально возможную дистанцию (то есть, на соседний справа элемент подходящего типа). Смещение указателя может производиться с помощью операторов инкремента и декремента.

```
int ar[] = { -72, 3, 402, -1, 55, 132 };
int* p = ar;
for (int i = 0; i < 6; i++) {
    cout << *p << ' ';
    p++;
}
for (int i = 0; i < 6; i++) {
    cout << &ar[i] << ' ';
}
```

В примере все элементы массива будут выведены на экран без использования индексов (первый цикл `for`). Далее выведены адреса элементов массива (второй цикл `for`). Они будут отличаться на размер базового типа массива в байтах (в данном случае, `int` – 4 байта).

Справедливо тождество:

```
a[i] == *(a + i),
```

где `a` – указатель на массив любого типа и `i` допустимый индекс этого массива.

6.4. Динамические массивы

Динамический массив – это массив, размер которого заранее не фиксирован и может меняться во время исполнения программы. Для изменения размера динамического массива язык программирования C++, предоставляет специальные встроенные функции. Динамические массивы дают возможность более гибкой работы с данными, так как позволяют не прогнозировать хранимые объемы данных, а регу-

лировать размер массива в соответствии с реально необходимыми объемами.

Под объявлением одномерного динамического массива понимают объявление указателя на переменную заданного типа. Он будет указывать на область памяти, в дальнейшем выделяемую из «кучи».

Описание.

```
Тип * ИмяМассива;
```

Таким образом, сначала следует описать соответствующий указатель, которому будет присвоено значение адреса начала области выделенной памяти.

Выделение памяти производится при помощи операции `new`, которая выделяет для размещения массива участок динамической памяти соответствующего размера и не позволяет инициализировать элементы массива.

Описание.

```
ИмяМассива = new Тип [Размер];
```

Освобождение памяти, выделенной под одномерный динамический массив, осуществляется при помощи операции `delete`, которая освобождает участок ранее выделенной памяти.

Описание.

```
delete [] ИмяМассива;
```

Пример. Найти сумму элементов динамического целочисленного массива. Все отрицательные элементы заменить нулями, при их отсутствии – вывести сообщение.

Программа на языке C++:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
```

```
#include <locale.h>

using namespace std;

int main()
{
    setlocale(0, "");
    srand(time(NULL));
    int i, N;
    //размер массива заранее не известен
    cout << "Введите размер массива: ";
    cin >> N;

    //объявление одномерного динамического массива
    int* A = new int[N];
    // Генерация элементов массива
    for (i = 0; i < N; i++) {
        *(A + i) = rand() % 101 - 50;
    }

    // Вывод элементов исходного массива
    cout << endl << "Исходный массив" << endl;
    for (i = 0; i < N; i++) {
        cout << *(A + i) << " ";
    }
    cout << endl;

    // Вычисление суммы элементов массива
    int summ = 0;
    for (i = 0; i < N; i++) {
        summ += *(A + i);
    }
    cout << "Сумма элементов массива = " << summ << endl;

    // Обработка массива
    bool flag = false; // наличие отрицательных элементов
    for (i = 0; i < N; i++)  {
        if (*(A + i) < 0) {
            *(A + i) = 0;
            flag = true; // отрицательные элементы найдены
        }
    }

    // Вывод элементов результирующего массива
    if (flag) {
        cout << endl << "Результирующий массив" << endl;
        for (i = 0; i < N; i++) {
            cout << *(A + i) << " ";
        }
        cout << endl;
```

```
    }
    else
        cout << "Отрицательные элементы не обнаружены" << endl;

// освобождение памяти
    delete[]A;
    A = NULL;
    return 0;
}
```

В данном примере обращение к элементам массива намеренно осуществляется через указатель для демонстрации такого способа. На практике можно использовать любой вариант обращения (по индексу, как в массивах, и через указатель).

6.5. Вопросы для самопроверки

1. Чем область памяти «куча» отличается от области памяти «стек»?

2. Что может произойти при обращении к неинициализированному указателю? Для чего используется присваивание указателю значения NULL?

3. Что произойдет, если освободить одну и ту же область памяти дважды?

4. В чем заключается связь указателей и массивов?

6.6. Задания для самопроверки

1. Выполните индивидуальные задания из лабораторной работы №4 (приложение 1).

7. ОБРАБОТКА МНОГОМЕРНЫХ МАССИВОВ

7.1. Многомерные массивы

Многомерные массивы – поименованная структура данных, хранящая набор значений (элементов массива), идентифицируемых по набору индексов, принимающих целые значения из некоторого заданного непрерывного диапазона.

Размерность массива – это количество индексов, необходимое для однозначной адресации элемента в рамках массива. Размерность двумерных массивов равна 2, трехмерных – 3 и т.д. Ограничений на количество размерностей массива не существует.

Визуально массив может быть представлен в виде набора ячеек: вектора, таблицы, короба (рис. 13).

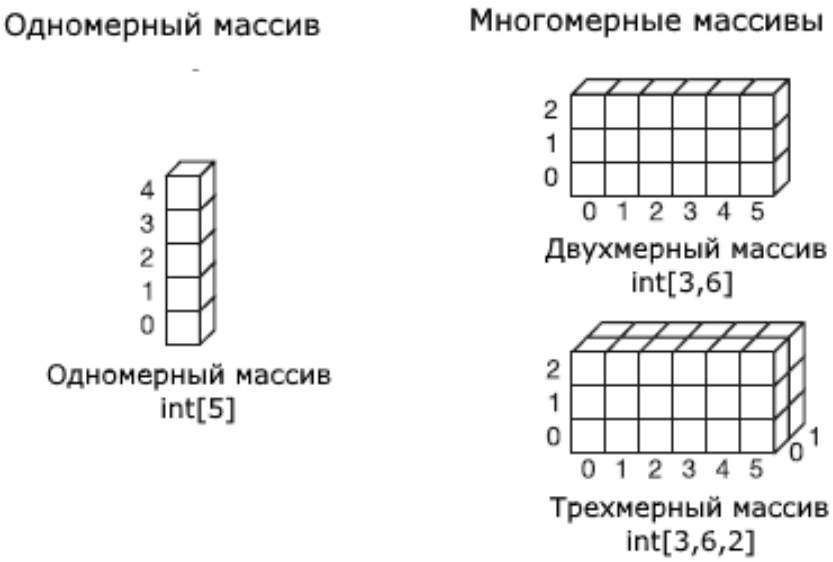


Рис. 13. Визуальное представление массивов

Определение многомерных массивов совпадает с определением одномерных массивов, за исключением того, что вместо одной размерности может быть указано несколько:

```
const int COLS = 3, ROWS = 5;
int array[COLS][ ROWS];
```

В этом примере определяется статический двумерный массив с именем array из 3 строк по 5 значений типа int в каждой строке, таким образом, массив array содержит 15 целых чисел.

Двумерные массивы располагаются в памяти в порядке быстрого изменения последнего индекса. Так, например, данные двумерного массива (состоящего из 3 строк и 10 столбцов и определенного как float A[3][10]), располагаются следующим образом:

Строки	Столбцы		
	1	2	9
1	A[0][0]	A[0][1] ...	A[0][9]
2	A[1][0]	A[1][1] ...	A[1][9]
3	A[2][0]	A[2][1] ...	A[2][9]

Двумерные массивы инициализируются так же, как и одномерные.

Например:

```
//инициализируется целочисленный массив размерностью 3x3
int W[3][3] = {
                {2, 3, 4}, //1-я строка
                {3, 4, 8}, //2-я строка
                {1, 0, 9}  //3-я строка
};
```

Однако вполне допустим и такой вариант:

```
int W[3][3] = { 2, 3, 4, 3, 4, 8, 1, 0, 9 };//все строки разом
```

Здесь главное, чтобы количество элементов в фигурных скобках совпадало с общим количеством элементов в массиве. Распределение по строкам компилятор сделает самостоятельно. Тем не менее, отделять содержимое строк фигурными скобками более предпочтительно, поскольку в этом случае, исходный код воспринимается легче.

Обращение к элементам двумерного массива осуществляется так же, как и к элементам одномерного:

```
ИмяМассива[ЗначениеИндекса][ЗначениеИндекса];
```

Например:

```
a[0][0] – индексы задаются как константы
s[i][j] – индексы задаются как переменные
```

Во многих случаях многомерный массив удобно заполнять значениями и обрабатывать с помощью вложенных циклов; как правило, количество циклов совпадает с размерностью массива.

Пример 1. Заполнить двумерный целочисленный статический массив размера `ROWS × COLS` таким образом, чтобы каждому элементу массива было присвоено значение, первая цифра которого указывает номер строки, а вторая цифра – номер столбца для этого значения, и вывести его на экран (нумерация должна начинаться с 1).

В результате должен быть получен следующий вывод на консоль:

```
11 12 13 14 15
21 22 23 24 25
31 32 33 34 35
```

Программа на языке C++:

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    const unsigned int ROWS = 3; // количество строк
    const unsigned int COLS = 5; // количество элементов в строке

    int A[ROWS][COLS], i, j;
    for (i = 0; i < ROWS; i++) {
        for (j = 0; j < COLS; j++) {
            A[i][j] = (i + 1) * 10 + (j + 1);
        }
    }
    for (i = 0; i < ROWS; i++) {
        for (j = 0; j < COLS; j++) {
            cout << setw(5) << A[i][j];
        }
```

```
        cout << endl;
    }

    return 0;
}
```

Ввиду необходимости ввода большого количества значений, для заполнения многомерных массивов в процессе обучения часто используют генератор псевдослучайных чисел.

Пример 2. Найти сумму положительных элементов двумерного целочисленного массива размера ROWS × COLS, заданного с помощью генератора случайных чисел, принадлежащих диапазону [-30; 50]. При их отсутствии вывести сообщение.

Программа на языке C++:

```
#include <iostream>
#include <iomanip>
#include <ctime>

using namespace std;

int main()
{
    const unsigned int ROWS = 3; // количество строк
    const unsigned int COLS = 5; // количество элементов в строке
    int A[ROWS][COLS], i, j, sum = 0, k = 0;
    srand(time(NULL));
    // Заполнение массива псевдослучайными числами
    // из заданного диапазона [-30; 50]
    for (i = 0; i < ROWS; i++) {
        for (j = 0; j < COLS; j++) {
            A[i][j] = -30 + rand() % 81;
        }
    }
    // Вывод элементов массива с выравниванием
    cout << "Массив" << endl;
    for (i = 0; i < ROWS; i++) {
        for (j = 0; j < COLS; j++) {
            cout << setw(7) << A[i][j];
        }
        cout << endl;
    }
    // Вычисление суммы положительных элементов
    // массива
    for (i = 0; i < ROWS; i++) {
        for (j = 0; j < COLS; j++) {
```

```
            if (A[i][j] > 0) {
                sum += A[i][j];
                k++;
            }
        }
    }
    if (k > 0)
        cout << "Сумма положительных элементов массива = " <<
                                                        sum << endl;
    else
        cout << "Положительные элементы отсутствуют";

    return 0;
}
```

7.2. Двумерные динамические массивы

Существует несколько способов представления двумерных динамических массивов. Первым способом является представление двумерного массива в виде одномерного. В этом случае все «двумерные строки» записываются одна за другой в одномерный массив.

Пример 3. Осуществить вод и вывод двумерного массива.

```
#include <iostream>
#include <iomanip>
#include <ctime>

using namespace std;

int main()
{
    int ROWS; // количество строк
    int COLS; // количество элементов в строке
    int i,  j;
    cout << "Введите количество строк: ";
    cin >> ROWS;
    cout << "Введите количество столбцов: ";
    cin >> COLS;
    //выделяем память под "двумерный" массив
    //в виде одномерного
    int *a = new int[ROWS * COLS];
    srand(time(NULL));

    //заполняем случайными значениями
    //обращаемся к эл. массива через указатель
```

```
    for (i = 0; i < ROWS; i++)
        for (j = 0; j < COLS; j++)
            *(a + COLS * i + j) = rand() % 10;

    // выводим на экран в виде матрицы
    cout << endl <<"Array " << endl;
    for (i = 0; i < ROWS; i++) {
        for (j = 0; j < COLS; j++)
            cout << *(a + COLS * i + j) << " ";
        cout << endl;
    }

    //заполняем случайными значениями
    //обращаемся к эл. массива по индексу
    for (i = 0; i < ROWS; i++)
        for (j = 0; j < COLS; j++)
            a[COLS * i + j] = rand() % 10;

    // выводим на экран в виде матрицы
    cout << endl<< "Array " << endl;
    for (i = 0; i < ROWS; i++) {
        for (j = 0; j < COLS; j++)
            cout << a[COLS * i + j] << " ";
        cout << endl;
    }
    // освобождаем память, выделенную под массив
    delete [] a;

    return 0;
}
```

Особенностью данного подхода является то, как мы обращаемся к элементу массива:

a[COLS * i + j]

Номер текущей строки `i` умножается на количество элементов в строке `COLS`, что дает нам смещение на начало нужной «двумерной строки» в массиве. К этому числу мы добавляем смещение элемента внутри «двумерной строки» `j`.

7.3. Указатели и двумерные массивы

При размещении элементов двумерных массивов они располагаются в единой области памяти по строкам. При последовательной обработке массива с помощью вложенных циклов, быстрее всего изменяется последний индекс (номер элемента в строке), а медленнее – первый (номер строки). Такой порядок дает возможность обращаться к любому элементу двумерного массива, используя адрес его начального элемента и только одно индексное выражение.

```
int arr[ROWS][COLS];
Адрес(arr[i][j]) = Адрес(arr[0][0]) + (COLS * i + j) * k,
```

где `k` – количество байтов, выделяемое для элемента массива (в зависимости от типа).

Указатели на двумерные массивы в языке C++ – это массивы массивов, т.е. такие массивы, элементами которых являются массивы. При объявлении таких массивов в памяти компьютера создается несколько различных объектов. Например, при выполнении объявления двумерного массива:

```
int arr[4][3];
```

в памяти выделяется участок для хранения значения переменной `arr`, которая является указателем на массив из четырех указателей.

Для этого массива из четырех указателей тоже выделяется память. Каждый из этих четырех указателей содержит адрес одномерного массива из трех элементов типа `int`.

Следовательно, в памяти компьютера выделяется четыре участка для хранения четырех массивов чисел типа `int`, каждый из которых состоит из трех элементов.

Таким образом, объявление `arr[4][3]` порождает в программе три разных объекта:

– указатель с идентификатором `arr`,

– безымянный массив из четырех указателей: `arr[0], arr[1], arr[2], arr[3]`

– 4 безымянных массива из двенадцати чисел типа `int`.

Для доступа к безымянным массивам используются адресные выражения с указателем `arr`. Доступ к элементам одномерного массива указателей осуществляется с указанием одного индексного выражения в форме `arr[2]` или `*(arr+2)`.

Для доступа к элементам двумерного массива чисел типа `int arr[i][j]` должны быть использованы следующие выражения:

Например, пусть `i=1, j=2`, тогда обращение к элементу `arr[1][2]`:

`arr[i][j]` или `arr[1][2]=10`

`*(*(arr+i)+j)` или `*(*(arr+1)+2)=10`

`(*(arr+i))[j]` или `(*(arr+1))[2]=10`

Причем внешне похожее обращение `arr[5]` выполнить невозможно, так как указателя с индексом `5` не существует.

Для создания двумерного динамического массива вначале нужно распределить память для массива указателей на одномерные массивы, а затем выделить память для одномерных массивов. При динамическом распределении памяти для массивов следует описать соответствующий указатель, которому будет присвоено значение адреса начала области выделенной памяти.

7.4. Двумерные динамические массивы

Под объявлением двумерного динамического массива понимают объявление двойного указателя, то есть объявление указателя на указатель.

Синтаксис:

```
Тип ** ИмяМассива;
```

`ИмяМассива` – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

`Тип` – тип элементов объявляемого динамического массива. Элементами динамического массива не могут быть функции и элементы типа `void`.

Например:

```
int **a;
float **m;
```

При формировании двумерного динамического массива сначала выделяется память для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется память под каждый одномерный массив отдельно. На рисунке 14 представлена схема динамической области памяти, выделенной под двумерный массив.

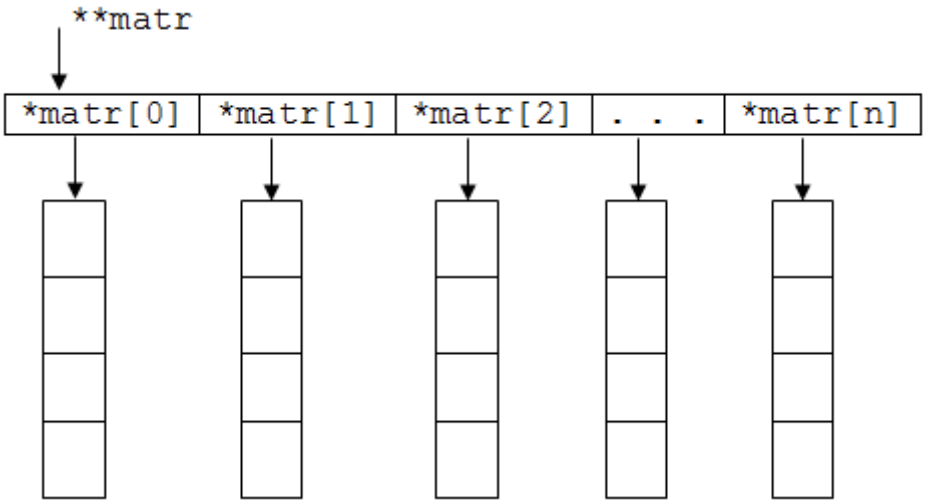


Рис. 14. Схема динамической области памяти,
выделенной под двумерный массив

Синтаксис выделения памяти под массив указателей следующий:

```
ИмяМассива = new Тип* [Кол-воЭлементов];
```

Синтаксис выделения памяти для массива значений:

```
ИмяМассива[ЗначИндекса] = new Тип[Кол-воЭлементов];
```

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

Тип – тип указателя на массив.

Кол-воЭлементов – задает количество элементов (размерность) массива.

Например:

```
int ROWS = 3; // количество строк
int COLS = 5; // количество элементов встроке
int i;

float** matr; //указатель для массива указателей
//выделение динамической памяти под массив указателей
matr = new float* [ROWS];
for (int i = 0; i < ROWS; i++)
    //выделение динамической памяти массива значений
    matr[i] = new float[COLS];
. . .
```

На момент выделения динамической памяти размеры массивов должны быть полностью определены.

Удаление из динамической памяти двумерного массива осуществляется в порядке, обратном его созданию: сначала освобождается память, выделенная под одномерные массивы с данными, а затем память, выделенная под одномерный массив указателей.

Освобождение памяти, выделенной под двумерный динамический массив, осуществляется при помощи операции delete, которая освобождает участок памяти ранее выделенной операцией new.

Синтаксис освобождения памяти, выделенной для массивов значений:

```
delete ИмяМассива [ЗначениеИндекса];
```

Синтаксис освобождения памяти, выделенной под массив указателей:

```
delete [] ИмяМассива;
```

ИмяМассива – идентификатор массива, то есть имя двойного указателя для выделяемого блока памяти.

Например:

```
//освобождает память, массивов значений
for (int i = 0; i < ROWS; i++)
    delete matr[i];

//освобождает память, выделенную под массив указателей
delete[] matr;
```

Квадратные скобки [] означают, что освобождается память, занятая всеми элементами массива, а не только первым.

Пример 4. Запросить у пользователя размеры массива ROWS и COLS. Описать двумерный динамический массив с указанными параметрами. Заполнить массив с помощью генератора случайных чисел значениями из диапазона [1; 10]. Вывести элементы массива на экран в виде матрицы.

Программа на языке C++.

```
#include <iostream>
#include <iomanip>
#include <ctime>

using namespace std;

int main()
{
    int ROWS; // количество строк
    int COLS; // количество элементов в строке
    int i, j;

    // объявление переменной массива
    int** matrix = NULL;

    cout << "Введите размеры массива ROWS и COLS: ";
    cin >> ROWS >> COLS;

    // выделение памяти
    matrix = new int* [ROWS];
    for (i = 0; i < ROWS; i++)
```

```
        matrix[i] = new int[COLS];

    // заполнение массива
    srand(time(NULL));
    for (i = 0; i < ROWS; i++) {
        for (j = 0; j < COLS; j++) {
            matrix[i][j] = rand() % 10 + 1;
        }
    }

    // вывод массива на экран
    cout << "Matrix:" << endl;
    for (i = 0; i < ROWS; i++) {
        for (j = 0; j < COLS; j++) {
            cout << setw(3) << matrix[i][j];
        }
        cout << endl;
    }

    // освобождение памяти
    for (i = 0; i < ROWS; i++)
    {
        delete[]matrix[i];
        matrix[i] = NULL;
    }

    delete[]matrix;
    matrix = NULL;

    return 0;
}
```

7.5. Вопросы для самопроверки

1. Что такое двумерный массив? Чем он отличается от одномерного массива?

2. Существует ли ограничение на количество размерностей массива?

3. Существует ли ограничение на тип элемента двумерного массива?

4. Сколько байт в оперативной памяти будет занимать массив, объявленный следующим образом:

`char A[10][20][30][40];` ?

7.6. Задания для самопроверки

1. Выполните индивидуальные задания из лабораторной работы №5 (приложение 1).

8. СИМВОЛЬНЫЕ И СТРОКОВЫЕ ДАННЫЕ

8.1. Объявление символьных данных и строк

Международным стандартом для кодировки символьной информации в области информационных технологий стала таблица ASCII (читается «аски») – Американский стандартный код для информационного обмена. Таблица кодов ASCII состоит из двух столбцов: код символа и его изображение. Международным стандартом является лишь первая половина таблицы, т.е. символы с номерами от `0 (00000000)`, до `127 (01111111).` Вторая половина используется для определения символов национальных алфавитов и некоторых дополнительных символов (например, символов «псевдографики» для отображения рамок, таблиц и т.д.).

В настоящее время существуют несколько различных кодировок кириллицы (CP1251, CP866 , КОИ8-Р, ISO и др.), из-за этого часто возникают проблемы с переносом русского текста с одного компьютера на другой, из одной программной системы в другую.

Для представления текстовой информации в языке C++ используются символьные литералы (константы), символьные переменные и строки (строковые литералы). Для строк в языке C++ изначально не введено отдельного типа данных.

Для описания символьных данных в C++ используется тип `char`. Этот тип описывает целое число размером в один байт. В контексте обработки символьной информации это число интерпретируется компилятором, как код символа в таблице ASCII.

Описание символьных переменных:

```
char список_имен_переменных;
```

Например:

```
// символ – занимает один байт, его значение не меняется:
const char ch = 'c';
// символьные переменные, занимают по одному байту:
char a, b;
```

Переменные `a` и `b` могут быть интерпретированы и как целые числа и как символы.

Язык C++ не определяет специального типа для объявления строк, поэтому поддержка строковых операций осуществляется с помощью стандартных алгоритмов обработки массивов. Наиболее типичным использованием одномерного массива типа `char`, являются строки.

Строка – это символьный массив произвольной длины, оканчивающийся нулевым символом или нуль-терминатором (это байт, у которого все биты равны нулю; обозначается `'\0'`). Поэтому символьные массивы должны содержать на один символ больше, чем максимальный размер, хранящейся в этом массиве строки. Например, если необходимо объявить массив, содержащий строку максимальная длина которой будет 10 символов, следует написать:

```
char Str[11];
```

В результате этого выделяется место в конце строки для нулевого символа.

Внимание! Все функции стандартной библиотеки для работы со строками определяют конец строки по нулевому символу и, если по каким-то причинам его там не окажется, будут обрабатывать смежные области память, выходящие за пределы указанной строки, до тех пор, пока им случайно не встретится `'\0'`.

В исходном коде на языке C/C++ можно объявить строковые литералы – это набор символов, заключенных в двойные кавычки. Например:

```
"HELLO WORLD"
"Введите исходные данные: "
```

При этом компилятор автоматически добавляет нуль-терминатор '\0' в конец строкового литерала.

Символьные литералы, в отличие от строковых, содержат только один символ и объявляются с помощью апострофов:

```
char ch = 'c';
```

В общем случае строка может содержать и один символ, например `"A"` (в двойных кавычках), однако, в отличие от символьного литерала `'A'` (используются апострофы), длина выделенной памяти под строку `"A"` равна 2 байтам. Под символьный литерал `'A'` - один байт (рис. 15).

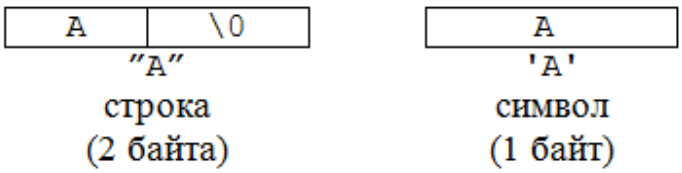


Рис. 15. Выделение памяти под строковые и символьные литералы

В языке C++ строка – это пронумерованная последовательность символов (массив символов), она всегда имеет тип `char[]` (или `char*`). Все символы строки нумеруются, начиная с нуля. Символ конца строки также нумеруется – ему соответствует наибольший из номеров. Таким образом, строка считывается значением типа «массив символов».

Присваивать значение строке с помощью оператора присваивания (`=`) нельзя, так как для массивов не определена операция прямого присваивания. Поместить строку в символьный массив можно либо при вводе, либо с помощью инициализации:

```
//инициализация строк
char s1[] = "ABCDEF"; //1-й способ
char s2[] = { 'A','B','C','D','E','F','\0' };//2-й способ
```

Для ввода-вывода символов и строк в стандартные потоки данных, также используются операторы `cin, cout` и операции `<<, >>`.

Например:

```
char s[20];
cin >> s; //ввод строки из стандартного потока
cout << s; //вывод строки в стандартный поток
```

При этом, считывание производится до пробела.

Если нужно извлечь данные из входного потока до строкового разделителя (`'\n'` , `endl`), то можно воспользоваться функцией `getline()`. При этом, сам разделитель не записывается в получившийся массив данных. Использование `getline()` на практике выглядит так:

```
cin.getline(string, streamsize, separator);
```

где `string` – переменная типа `char*`, в которую запишется строка, `streamsize` - максимально количество символов, которое может быть записано в строку, и `separator` – строковый разделитель, показывающий на конец строки. Последний параметр функции можно опустить, тогда будет задан сепаратор по умолчанию, то есть символ перевода строки – `'\n'`. Приведем пример работы функции `getline()` в программе:

```
char str[256];
cout << "Введите предложение: " << endl;
cin.getline(str, 256, ';'));
cout << str;
```

Программа записывает в переменную `str` символы из стандартного потока ввода. Предложение является законченным, только если в конце стоит `';'` . Если этот символ отсутствует, будет считано 256 символов. После записи предложение выводится на экран.

Пример 1. Задана строка символов. Определить, есть ли заданный символ с в этой строке символов.

Программа на языке С++.

```
#include <iostream>

using namespace std;

int main()
{
    char Str[80]; // строка символов
    char ch; // искомый символ
    int i = 0;
   // flag - символ есть в строке, иначе flag – false:
    bool flag = false;
    // ввод строки Str
    cout << "Введите строку: " << endl;
    cin.getline(Str, 80);
    // ввод символа ch
    cout << "Введите символ: " << endl;
    cin >> ch;
    while (Str[i] != '\0' && !flag) {
        if (Str[i] == ch)
        {
            flag = true; // символ ch есть в строке
        }
        i++;
    }
    if (flag)
        cout << "Символ \'" << ch << "\' есть в строке \"" <<
                                                    Str << "\" \n";
    else
        cout << "Символ \'" << ch << "\' отсутствует в строке \"" <<
                                                    Str << "\" \n";


    return 0;
}
```

В данном примере продемонстрирован вывод на экран символов апостроф (') и кавычки (").

Пример 2. Задана строка символов. Определить, является ли она палиндромом.

Палиндромы (перевертыши) – слова или фразы, читающиеся одинаково в обоих направлениях, например, шалаш. Самым длинным

в мире словом-палиндромом принято считать финское слово SAIPPUAKIVIKAUPPIAS – продавец мыла). В русском языке самым известным палиндромом считается предложение «А роза упала на лапу Азора».

Программа на языке C++:

```
#include <iostream>

using namespace std;

int main()
{
    char Str[80]; // строка символов
    char ch; // искомый символ
    int i = 0, len = 0;
// flag – true если палиндром, иначе flag – false
    bool flag = false;
    // ввод строки Str
    cout << "Введите строку: " << endl;
    cin.getline(Str, 80);
    // определение длины строки
    len = 0;
    while (Str[len] != '\0')
        len++;
    // решение задачи
    while (i < len / 2 && !flag) {
        if ((Str[i] != Str[len - i - 1])) {
            flag = true; // обнаружены отличия символов
        }
        i++;
    }
    if (flag)
        cout << "Строка НЕ является палиндромом\n";
    else
        cout << "Строка является палиндромом\n";


    return 0;
}
```

Так как строки представляет собой массивы символов, заканчивающиеся нулем, то появляется возможность работать с ними с использованием указателей.

Например, оператор:

```
    char* str = "First";
```

создает не строковую переменную, а указатель на строковый литерал и изменить этот литерал невозможно (к примеру, оператор `str[l] = 'R'` не допускается). Знак равенства перед строковым литералом означает инициализацию, а не присваивание.

Операция присваивания одной строки другой не определена (поскольку строка является массивом) и может выполняться с помощью цикла или функций стандартной библиотеки.

8.2. Функции стандартной библиотеки для работы со строками

Для работы с нуль-терминированными строками чаще всего пользуются функциями, описанными в заголовочном `cstring` (или `string.h`).

Функции, объявленные в `cstring` являются частью стандартной библиотеки и гарантированно работают на всех платформах, поддерживающих C++. Большинство функций `cstring` не производят никакого выделения памяти и контроля границ; эта обязанность целиком возлагается на программиста. В таблице 4 приведены некоторые наиболее часто употребляемые функции для работы со строками.

Таблица 4.

Функции для работы со строками

Функция	Пояснение
strlen(имя_строки**)**	определяет длину указанной строки, без учёта нуль-символа
Копирование строк	
strcpy(s1,s2**)**	выполняет побайтное копирование символов из строки s2 в строку s1
strncpy(s1,s2, n**)**	выполняет побайтное копирование n символов из строки s2 в строку s1. возвращает значения s1

Функция	Пояснение
Конкатенация строк	
strcat(s1,s2**)**	объединяет строку s2 со строкой s1. Результат сохраняется в s1
strncat(s1,s2,n**)**	объединяет n символов строки s2 со строкой s1. Результат сохраняется в s1
Сравнение строк	
strcmp(s1,s2**)**	сравнивает строку s1 со строкой s2 и возвращает результат типа int: 0 – если строки эквивалентны, >0 – если s1<s2, <0 — если s1>s2 С учётом регистра
strncmp(s1,s2,n**)**	сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 – если строки эквивалентны, >0 – если s1<s2, <0 — если s1>s2 С учётом регистра
stricmp(s1,s2**)**	сравнивает строку s1 со строкой s2 и возвращает результат типа **int**: 0 –если строки эквивалентны, >0 – если s1<s2, <0 — если s1>s2 Без учёта регистра
strnicmp(s1,s2,n**)**	сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если s1<s2, <0 — если s1>s2 Без учёта регистра
Обработка символов	
isalnum(c**)**	возвращает значение true, если c является буквой или цифрой, и false в других случаях
isalpha(c**)**	возвращает значение true, если c является буквой, и false в других случаях
isdigit(c**)**	возвращает значение true, если c является цифрой, и false в других случаях
islower(c**)**	возвращает значение true, если c является буквой нижнего регистра, и false в других случаях

Функция	Пояснение
isupper(c**)**	возвращает значение true, если c является буквой верхнего регистра, и false в других случаях
isspace(c**)**	возвращает значение true, если c является пробелом, и false в других случаях
toupper(c**)**	если символ c, является символом нижнего регистра, то функция возвращает преобразованный символ c в верхнем регистре, иначе символ возвращается без изменений.
Функции поиска	
strchr(s,c**)**	поиск первого вхождения символа c в строке s**.** В случае удачного поиска возвращает указатель на место первого вхождения символа c**.** Если символ не найден, то возвращается ноль.
strcspn(s1,s2**)**	определяет длину начального сегмента строки s1, содержащего те символы, которые не входят в строку s2
strspn(s1,s2**)**	возвращает длину начального сегмента строки s1, содержащего только те символы, которые входят в строку s2
strprbk(s1,s2**)**	Возвращает указатель первого вхождения любого символа строки s2 в строке s1
Функции преобразования	
atof(s1**)**	преобразует строку s1 в тип double
atoi(s1**)**	преобразует строку s1 в тип int
atol(s1**)**	преобразует строку s1 в тип long int

8.3. Вопросы для самопроверки

1. Что хранит переменная типа `char`: символ или число?

2. Почему символ и строка, состоящая из одного символа, занимают разный объем памяти?

3. Допустима ли операция сравнения над символами?

4. Можно ли сравнивать строкам? Если да, то как?

5. Что будет являться результатом работы функции побайтового копирования строк, если длина строки-источника превосходит допустимый размер строки-приемника?

8.4. Задания для самопроверки

1. Выполните индивидуальные задания из лабораторной работы №6 (приложение 1).

9. ФУНКЦИИ

9.1. Краткие теоретические сведения

Любая программа на C++ состоит из функций. Выполнение программы начинается с выполнения главной функции приложения, которая должна иметь имя `main()`.

Функция – это поименованный набор описаний и операторов, выполняющих определенную задачу. Функция может принимать параметры (аргументы) и возвращать значение. Информация, передаваемая в функцию для обработки называется параметром (аргументом), а результат работы функции – ее возвращаемым значением. Обращение к функции называют вызовом.

Перед вызовом функция должна быть описана. Описание функции состоит из заголовка и тела функции:

```
тип имя_функции(список_переменных)
{
```

```
    тело_функции
}
```

Заголовок функции содержит:

- `тип` возвращаемого функцией значения, он может быть любым; если функция не возвращает значения, указывают тип `void`;
- `имя_функции`, с которым она будет вызываться;
- `список_переменных` – перечень передаваемых в функцию аргументов, которые отделяются друг от друга запятыми; для каждой переменной из списка указывается тип и имя;

Тело функции представляет собой последовательность описаний и операторов, заключенных в фигурные скобки.

В общем виде структура программы на C++ может иметь вид:

```
директивы препроцессора

тип имя_1(список_переменных)
{
   тело_функции_1;
}
…
тип имя_n(список_переменных)
{
   тело_функции_n;
}
тип main(список_переменных)
{
   тело_основной_функции;
}
```

Завершает выполнение функции оператор `return`. Он возвращает управление вызывающей функции. Выполнение программы возобновляется в точке сразу после вызова. Также оператор `return` может возвращать результат выполнения функции, передавая его вызывающей функции.

Пример 1. Написать функцию нахождения минимального значения двух целых чисел.

```
int min2(int a, int b)
{
    int result;
    if (a < b)
        result = a;
    else
        result = b;
    return result;
}
```

В данном примере в теле функции введена промежуточная переменная `result`. Она может отсутствовать, например, функция `min2(int a, int b)` описана короче. При этом сразу производится возврат результата расчетов:

```
int min2(int a, int b)
{
    if (a < b)
        return a;
    else
        return b;
}
```

Вызвать функцию можно в любом месте программы. Для этого необходимо указать ее имя и в круглых скобках через запятую перечислить имена или значения аргументов, если такие имеются:

```
имя_функции (список_переменных);
```

Пример 2. Продемонстрировать использование разработанной функции `min2(int a, int b)`, вызвав ее в главной функции.

```
int main()
{
    int x, y, min;
    cout << "Введите два числа\n";
    cin >> x >> y;
    min = min2(x, y);
    cout << "Минимум равен " << min << endl;

    return 0;
```

```
}
```

Если тип возвращаемого значения не `void`, то функция может входить в состав выражений. При этом возвращается значение, определенное оператором `return`. Также возможен вывод значения, возвращаемого функцией. Функция из примера 2 может быть записана так, как показано ниже.

```
int main()
{
    int x, y;
    cout << "Введите два числа\n";
    cin >> x >> y;
    cout << "Минимум равен " << min2(x, y) << endl;
    return 0;
}
```

Стоит отметить, что тексты функции могут следовать после главной функции `main()`. Однако заголовки необходимо перечислить до нее. Такого рода опережающие заголовки называют прототипами функций. Прототип указывает компилятору тип данных, возвращаемых функцией, тип переменных, выступающих в роли аргументов, и порядок их следования. Прототипы используются для проверки правильности вызова функций в основной программе. Если в программе используются прототипы функций, то ее структура имеет другой вид.

```
директивы препроцессора

//список прототипов
тип имя_1(список_переменных);
…
тип имя_n(список_переменных);

int main(список_переменных)
{
    тело_основной_функции;
}

тип имя_1(список_переменных)
{
    тело_функции_1;
```

```
}
…
тип имя_n(список_переменных)
{
   тело_функции_n;
}
```

Имена переменных, указанные в прототипе функции, компилятор игнорирует. Два следующих прототипа функций являются идентичными.

```
int func(int a, int b);
int func(int, int);
```

Пример 3. Описать функцию нахождения минимального значения двух целых чисел и продемонстрировать ее выполнение с использованием прототипа функции `min2(int a, int b)`.

Программа на языке C++:

```
#include <iostream>

using namespace std;

int min2(int a, int b);

int main()
{
    int x, y;
    cout << "Введите два числа\n";
    cin >> x >> y;
    cout << "Минимум равен " <<
        min2(x, y) << endl;
    return 0;
}


int min2(int a, int b)
{
    if (a < b)
        return a;
    else
        return b;
}
```

Функция `main()` не является единственной функцией, которая может вызывать другие функции. Любая функция может вызывать любую другую функцию. Возврат результата из функции в вызывающую её функцию осуществляется оператором `return` выражение;

Работает оператор следующим образом. Вычисляется значение выражения, указанного после `return`, и преобразуется к типу возвращаемого функцией значения. Выполнение функции завершается, а вычисленное значение передаётся в вызывающую функцию. Любые операторы, следующие в функции за оператором `return`, игнорируются. Программа продолжает свою работу с оператора, следующего за оператором вызова данной функции.

Оператор `return` может отсутствовать в функциях типа `void`, если возврат происходит перед закрывающейся фигурной скобкой, и в функции `main()`.

Также функция может содержать несколько операторов `return`, если это определено потребностями алгоритма.

Пример 4. Описать функцию нахождения минимального значения четырех целых чисел с использованием функции `min2(int a, int b)` и продемонстрировать ее выполнение.

Программа на языке C++:

```
#include <iostream>

using namespace std;

int min2(int a, int b)
{
    if (a < b)
        return a;
    else
        return b;
}

int min4(int a, int b, int c, int d)
{
    int min_ab = min2(a, b);
    int min_cd = min2(c, d);
```

```
    int result = min2(min_ab, min_cd);
    return result;
}

int main()
{
    cout << "Минимум чисел 1, 2, 3, 4 равен " <<
        min4(1, 2, 3, 4) << endl;
    return 0;
}
```

Функция `min4(int a,int b, int c, int d)` может иметь более простой вид.

```
int min4(int a, int b, int c, int d)
{
    return min2(min2(a, b), min2(c, d));
}
```

Пример 5. Описать функцию, которая вычисляет корни квадратного уравнения `ax^2+dx+c=0`. Если `a=0` (уравнение не является квадратным), то в вызывающую программу передаётся значение равное -1, если дискриминант отрицательный (уравнение не имеет действительных корней), то 1, а если положительный, то вычисляются корни уравнения и в вызывающую программу передаётся 0.

Программа на языке C++:

```
#include <iostream>
#include <cmath>

using namespace std;

int equation(float a, float b, float c, float* x1, float* x2)
{
     float D = b * b - 4 * a * c;
     if (a == 0) return -1;
     else
          if (D < 0) return 1;
          else
          {
               *x1 = (-b + sqrt(D)) / 2 / a;
               *x2 = (-b - sqrt(D)) / 2 / a;
               return 0;
          }
}
```

```
int main()
{
      float A, B, C, X1, X2;
      int P;
      cout << " Введите коэффициенты уравнения:" <<
            endl;
      cout << "A="; cin >> A;
      cout << "B="; cin >> B;
      cout << "C="; cin >> C;
      P = equation(A, B, C, &X1, &X2);
      if (P == -1)
            cout << "Уравнение не является квадратным" <<
            endl;
      else if (P == 1)
            cout << "No real roots" << endl;
      else
            cout << "X1=" << X1 << "; X2=" << X2 << endl;
      return 0;
}
```

Переменные описанные внутри функции, а также переменные из списка аргументов, называются локальными. Например, если программа содержит пять разных функций, в каждой из которых описана переменная n, то для C++ это пять различных переменных. Область действия локальной переменной не выходит за рамки функции.

Переменные, определенные до объявления всех функций и доступные всем функциям, называют глобальными. В функции глобальную переменную можно отличить, если не описана локальная переменная с теми же именем. Глобальные переменные применяют для передачи данных между функциями, но это затрудняет отладку программы. Для обмена данными между функциями используют параметры функций и значения, возвращаемые функциями.

9.2. Передача параметров в функции

Механизм параметров является основным способом обмена информацией между вызываемой и вызывающей функциями. Параметры, перечисленные в заголовке описания функции, называются формальными параметрами, или просто параметрами, а записанные в

операторе вызова функции – фактическими параметрами, или аргументами.

При вызове функции в первую очередь вычисляются выражения, стоящие на месте аргументов; затем в стеке выделяется память под формальные параметры функции в соответствии с их типом, и каждому из них присваивается значение соответствующего аргумента. При этом проверяется соответствие типов и при необходимости выполняются их преобразования. При несоответствии типов выдается диагностическое сообщение.

Существует два способа передачи параметров в функцию: по значению и по адресу.

При передаче по значению в стек заносятся копии значений аргументов, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а, следовательно, нет и возможности их изменить.

При передаче по адресу в стек заносятся копии адресов аргументов, а функция осуществляет доступ к ячейкам памяти по этим адресам и может изменить исходные значения аргументов:

Например, в примере 6 решается задача обмена значениями двух переменных. Функция swap1() при передаче параметров по значению, которая должна менять значение параметров местами, не будет этого делать:

Для решения этой задачи необходимо в качестве параметра передавать адрес переменной, которую надо изменять, т. е. передавать указатель на переменную. Такой прием в программировании называется передачей параметра по ссылке. Вызванная функция должна описывать аргумент как ссылку и обращаться к фактической переменной косвенно, через эту ссылку.

Пример 6. Описать функцию, которая производит обмен значениями двух целочисленных переменных. Провести тестирование функции.

Программа на языке C++:

```
#include <iostream>

using namespace std;

void swap1(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}

void swap2(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

int main()
{
    int x, y;
    cout << "Введите два числа: ";
    cin >> x >> y;
    cout << "Входные значения\nx = " << x << " y = " << y << endl;
        swap1(x, y);
    cout << "Функция 1\nx = " << x << "; y = " << y << endl;
    swap2(&x, &y);
    cout << "Функция 21\nx = " << x << "; y = " << y << endl;
    return 0;
}
```

Пример работы программы показан на рисунке 16.

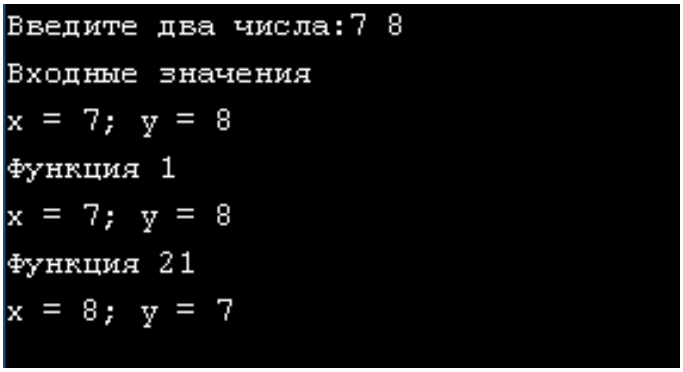


Рис. 16. Результат работы программы из примера 6

9.3. Обработка целочисленных данных.

Целочисленные данные являются одними из наиболее часто используемых типов данных в программировании. Любой символ в компьютере связан с целым числом – кодом этого символа, например в таблице ASCII. Сам символ необходим, когда информация выводится на экран или на принтер, или, наоборот, вводится с клавиатуры.

В стандарте ANSI языка C++ имеются модификаторы:

```
signed      - знаковый,
unsigned  - беззнаковый,
long          - длинный,
short         - короткий.
```

Модификаторы ставятся перед соответствующим типом и, в результате, получаются новые типы данных. Модификаторы `signed` и `unsigned` могут применятся к типам `char` и `int`. Модификаторы `short` и `long` могут применятся к типу `int`. Модификаторы `signed` и `unsigned` могут комбинироваться с модификаторами `short` и `long` в применении к типу `int`.

Размер в байтах и интервал изменения целых типов данных с различными комбинациями модификаторов могут варьироваться в зависимости от компилятора, процессора и операционной системы.

Можно заметить, что типы `int, short int, signed int` и `signed short int` имеют одни и те же пределы изменения. Эти типы, хотя и имеют разные названия, являются совершенно одинаковыми в данной реализации компилятора языка. Типы `int` и `signed int` всегда обозначают один и тот же тип, так как тип `int` всегда знаковый и без модификатора `signed`. Модификатор `short` в данной реализации не оказывает никакого воздействия на тип.

При работе с целыми данными нередко приходится иметь дело с операциями нахождения остатка от деления нацело и частного от деления нацело.

При работе с целыми данными используются операции, не применимые к действительным числам: `mod` – остаток от деления нацело, и `div` – частное от деления нацело. Эти операции определяются следующим образом:

```
a  div  b  =  [ a / b ]
a  mod  b  =  a  -  b * [a / b ],
```

где [x] - целая часть числа x.

Например,

```
11 div 4 = [2.75] = 2
(-11) div 4 = [-2.75] = -2
11 div (-4) = [-2.75] = -2
(-11) div (-4) = [2.75] = 2
11 mod 4 = 11 – 4*2 = 3
(-11) mod 4 = (-11) – 4*(-2) = -3
11 mod (-4) = 11 – (-4)*(-2) = 3
(-11) mod (-4) = (-11) – (-4)*2 = -3
```

Результат деления нацело по знаку положителен, если оба операнда одного знака, и отрицателен, если операнды разного знака. Это аналогично обычному делению. Остаток же от деления нацело, если не равен нулю, по знаку совпадает с делимым. Делить на нуль нельзя, предполагается, что $b \neq 0$.

В языке C++ операция `mod` обозначается `%`, т.е. `a mod b` записывается как `a % b`. Эта операция определена только для целых `a, b`. Специального символа для обозначения операции `div` в языке C++ нет. Операция `a div b` в языке C++ реализуется как `a/b`, где `a` и `b` – целые данные. Дело в том, что в языке C++ операция деления (`/`) для целых операндов вычисляет частное от деления нацело. Если же хотя бы один операнд вещественный, то это будет обычная операция деления. Делить на нуль нельзя, т.е. в операциях `a/b, a % b` предполагается, что $b \neq 0$.

Пример 7. Вычислить сумму цифр целого числа n.

Программа на языке C++:

```
#include <iostream>

using namespace std;

int main()
{
    long int n;
    int S = 0;
    cout << "Введите число: ";
    cin >> n;
    if (n < 0)
        n = -n;
    while (n!= 0) {
        S += n % 10;
        n /= 10;
    }
    cout << "Сумма цифр=" << S;

    return 0;
}
```

Число `n` может быть отрицательным. Для вычисления суммы цифр знак числа роли не играет, поэтому если число отрицательное, то в начале алгоритма производится изменение знака. Если число не равно нулю, то оно содержит цифры. Вычислив остаток от деления нацело числа `n` на 10, определяется последняя цифра числа. Применяя ту же процедуру к частному от деления нацело на 10 числа `n`, опреде-

ляются остальные цифры. Суммируя в переменной S выделяемые из числа n цифры, решается поставленная задача.

Пример 8. Для заданного натурального числа n определить, является ли оно совершенным. Натуральное число – совершенное, если оно равно сумме своих делителей меньших самого себя. Например, число 6 совершенное, т.к. 6=1+2+3.

Программа на C++:

```
#include <iostream>

using namespace std;

int main()
{
    unsigned int n;
    int k, S;
    cout << "Введите n: ";
    cin >> n;
    S = 0;
    for (k = 1; k < n; k++) {
        if (n % k == 0)
            S = S + k;
    }
    if (n == S)
        cout << "Число " << n << " – совершенное";
    else
        cout << "Число " << n << " – несовершенное";

    return 0;
}
```

Замечание: если остаток от деления нацело числа n на k равен нулю, то это значит, что число n делится на k без остатка.

Пример 9. Оформить решение задачи из примера 8, является ли натуральное число совершенным, в виде функции. Найти все совершенные числа на интервале [a, b], где a, b вводятся пользователем.

Программа на C++

```
#include <iostream>

using namespace std;

bool IsPerf(int m)
```

```
{
    int k, Sum = 0;
    for (k = 1; k < m; k++)
        if (m % k == 0)
            Sum += k;
    if (m == Sum)
        return true;
    else
        return false;
}

int main()
{
    unsigned int n;
    // наличие совершенных чисел на интервале:
    bool flag = false;
    int a, b, i;
    cout << "Введите границы интервала" <<
        " натуральные числа a и b: ";
    cin >> a >> b;
    if (a > b || a < 1)
        cout << "Некорректный интервал" << endl;
    else
    {
        for (i = a; i <= b; i++) {
            if (IsPerf(i))
            {
                cout << i <<
                    " - совершенное число" << endl;
                flag = true;
            }
        }
        if (!flag)
            cout << "На интервале [" << a << "; " << b <<
                            " отсутствуют совершенные числа" << endl;
    }
    return 0;
}
```

9.4. Вопросы для самопроверки

1. Как введение функции позволяет избежать избыточности кода?
2. Что такое прототип функции?
3. В каких случаях используется передача параметров по ссылке?
4. Что такое локальные и глобальные переменные?

5. Можно ли возвращать из функции указатель на локальную переменную?

9.5. Задания для самопроверки

1. Выполните индивидуальные задания из лабораторной работы №7 (приложение 1).

ЗАКЛЮЧЕНИЕ

В данном учебном пособии изложен материал первого семестра изучения дисциплины «Языки и методы программирования», позволяющий получить общее представление о возможностях практического использования языка C++ при решении задач начального уровня.

Были рассмотрены базовые синтаксические конструкции. Рассмотрены ключевые элементы технологии программирования такие как: работа с памятью, реализация алгоритмов с ветвлениями, циклических алгоритмов, обработка символьной информации, разработка повторно используемого кода с помощью применения функций.

В учебное пособие не вошли такие темы, как работа со структурированными типами данных, разработка рекурсивных функций, принципы построения и использование динамических списочных типов данных, базовые алгоритмы поиска. Данные разделы будут рассмотрены в материалах второго семестра обучения.

Дополнительные сведения по разработке приложений, а также задания для самостоятельного решения и множество различных примеров можно получить из книг, приведенных в списке рекомендованной и дополнительной литературы.

ПРИЛОЖЕНИЕ 1. ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА

Порядок выполнения работ

1. Получение варианта задания на лабораторную работу.
2. Разработка алгоритма и графической схемы решения задачи.
3. Составление программы на языке C++.
4. Отладка программы.
5. Тестирование программы.
6. Составление отчета о проделанной работе.

Требования к оформлению отчета

1. Оформить титульный лист с указанием темы работы.
2. Сформулировать цель и задачи работы.
3. Привести формулировку задания.
4. Построить графическую схему алгоритма.
5. Записать текст программы.
6. Привести результаты тестирования программы.
7. Сформулировать вывод по проделанной работе.

Критерии результативности выполнения работы

Лабораторная работа считается успешно выполненной, если:

- представленный отчет содержит все необходимые пункты, согласно требованиям;
- успешно демонстрируется работа программы как на выбранных студентом исходных данных, так и на заданных преподавателем исходных данных;
- студент правильно отвечает на контрольные вопросы.

ЛАБОРАТОРНАЯ РАБОТА № 1

УСЛОВНЫЕ ОПЕРАТОРЫ НА ЯЗЫКЕ С++

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью работы является получение обучающимися навыков составления и отладки простейших программ на языке С++.

Задачами работы являются:

- составление графической схемы алгоритма решения задачи;
- написание программы на языке С++ для простейших математических расчетов;
- тестирование разработанной программы для различных исходных данных.

2. ВАРИАНТЫ ЗАДАНИЙ

Разработать консольные приложения на языке С++, решающие следующие задачи:

1. а) Треугольник задан своими сторонами: a, b, c. Требуется найти медианы треугольника l_a, l_b, l_c.
   б) Написать программу решения уравнения $ax^3 + bx = 0$ для произвольных действительных чисел a, b.
2. а) Даны три действительных числа. Найти среднее арифметическое и среднее геометрическое их модулей.
   б) Треугольник задан своими вершинами (x_1, y_1), (x_2, y_2), (x_3, y_3). Определить вид треугольника (остроугольный, прямоугольный, тупоугольный), если он существует.
3. а) Найти все высоты треугольника, заданного длинами своих сторон.
   б) Заданы размеры A, B прямоугольного отверстия и размеры X, Y, Z кирпича. Определить, пройдет ли кирпич через отверстие так, чтобы каждое из его ребер было параллельно или перпендикулярно сторонам отверстия.
4. а) Найти сумму членов бесконечно убывающей геометрической

прогрессии по заданным значениям первого члена и знаменателя.

б) Даны три положительных числа. Определить, можно ли построить треугольник с длинами сторон, равным этим числам. Если можно, то ответить на вопрос, является ли он остроугольным.

5. а) Смешано v_1 литров воды с температурой t_1 с v_2 литрами воды с температурой t_2. Найти объем и температуру образовавшейся смеси.

б) Задана точка с координатами (x, y). Определить, на какой оси или в какой четверти она находится.

6. а) Три сопротивления R_1, R_2, R_3 соединены параллельно. Найти сопротивление соединения.

б) Заданы величины a, b, c. Определить, являются ли они упорядоченными, т.е. выполняется ли одно из условий:
$a \le b \le c$ или $a \ge b \ge c$.

7. а) Определить время падения камня на поверхность земли с заданной высоты h.

б) Даны два целых числа: D (день) и M (месяц), определяющие правильную дату не високосного года. Вывести значения даты, следующей за указанной.

8. а) Определить силу притяжения F между телами массы m_1 и m_2, находящимися на расстоянии r друг от друга.

б) Проверить, можно ли из четырех отрезков, заданных длинами, составить параллелограмм.

9. а) Найти площадь кольца, внутренний радиус которого r, а внешний R.

б) Определить максимальное и минимальное значения из трех различных вещественных чисел.

10. а) Найти сумму членов арифметической прогрессии a, $a + d$, $a + 2d$,…,$a + (n\text{-}2)d$, $a + (n\text{-}1)d$ по заданным значениям a, d, n.

б) Даны значения трех углов в градусах. Определить, может ли существовать треугольник с такими углами. Если да, то будет ли он прямоугольным.

11. а) Треугольник задан своими сторонами: a, b, c. Требуется найти площадь описанного круга

б) Решить квадратное уравнение $ax^2 + bx + c = 0$ с действительными коэффициентами для произвольных чи-

сел a, b, c.

12. а) Задан круг своим радиусом r. Найти площадь равностороннего треугольника, в который вписан этот круг.
    б) Заданы величины a, b, c, d. Найти максимальное значение.
13. а) Треугольник задан координатами своих вершин на плоскости. Найти площадь треугольника.
    б) Решить линейное уравнение $ax + b = 0$ для произвольных коэффициентов a и b.
14. а) Треугольник задан длинами своих сторон: a, b, c. Найти углы треугольника.
    б) Задана точка (x_0, y_0) и прямая $y = ax + b$. Определить, где находится точка относительно прямой: на прямой, выше, или ниже.
15. а) Определить время, через которое встретятся два тела, равноускоренно движущиеся навстречу друг другу, если известны их начальные скорости, ускорения и начальное расстояние между ними.
    б) Заданы три числа a, b, c. Определить, существует ли треугольник со сторонами a, b, c; если существует, то является ли полученный треугольник равнобедренным.
16. а) Дана длина ребра куба. Найти объем куба и площадь его поверхности.
    б) Определить, войдет ли в конверт с размерами a и b мм прямоугольная открытка с размерами c и d мм. Для размещения открытки в конверте необходим зазор в 1 мм с каждой стороны.
17. а) Вычислить расстояние между двумя точками с координатами (x_1, y_1) и (x_2, y_2)
    б) Определить вид треугольника, заданного своими тремя углами (остроугольный, прямоугольный, тупоугольный, равносторонний, равнобедренный)
18. а) Задан четырехугольник на плоскости координатами своих вершин в порядке обхода. Найти его периметр.
    б) Найти количество точек пересечения прямой $y = kx + b$ и окружности радиуса R с центром в начале координат.
19. а) Вычислить объем и площадь полной поверхности цилиндра высота которого h, а радиус основания r.
    б) Даны три стороны a_1, b_1, c_1одного и три стороны a_2, b_2,

c_2другого треугольника. Определить, будут ли эти треугольники равновеликими, т.е. равные ли у них площади.

20. а) Треугольник задан своими сторонами: a, b, c. Требуется найти площадь вписанного круга.
    б) Заданы величины a, b, c, d. Найти минимальное по модулю значение.
21. а) Треугольник задан на плоскости координатами своих вершин. Найти его периметр.
    б) Определить количество действительных корней квадратного уравнения $ax^2 + bx + c = 0$ с коэффициентами a, b, c.
22. а) Треугольник задан на плоскости координатами своих вершин. Найти координаты центра этой фигуры.
    б) Даны два целых числа: D (день) и M (месяц), определяющие правильную дату не високосного года. Вывести значения D и M для даты, предшествующей указанной.
23. а) Задан прямоугольник на плоскости координатами своих вершин. Найти координаты центра этой фигуры.
    б) Заданы две прямые $ax + by + c = 0$ и $px + qy + r = 0$. Определить, как расположены прямые друг относительно друга: пересекаются, совпадают или параллельны.
24. а) Найти сумму n первых членов геометрической прогрессии по заданным значениямb, q, n.
    б) Задан треугольник своими вершинами (x_1, y_1), (x_2, y_2), (x_3, y_3). Определить вид треугольника (равносторонний, равнобедренный, общего вида).
25. а) Треугольник задан своими сторонами: a, b, c. Требуется найти медианы треугольника ma, mb, mc.
    б) Значения переменных A, B, C поменять местами так, чтобы они оказались упорядоченными по убыванию их модулей.

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что делает строка программы # include<iostream.h>?
2. В чем особенность функции main( )?
3. Что обозначают имена cin, cout и для чего они используются?
4. Каково назначение операций “>>” и “<<”?

5. Какие управляющие структуры используются в языке C++ для организации разветвляющихся алгоритмов?

6. Что такое условие и как формируются сложные условия?

7. Каково назначение оператора if? Каковы его формы?

8. Какой вид имеют графические изображения полной и неполной форм условного оператора?

9. Поясните порядок выполнения оператора switch. Каково здесь назначение оператора break?

10. Как разрешается неоднозначность, которая может возникнуть при использовании вложенных условных операторов?

ЛАБОРАТОРНАЯ РАБОТА № 2
ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью работы является получение обучающимися навыков реализации циклических алгоритмов на языке C++.

Задачами работы являются:

- написание программы на языке C++ с использованием различных типов циклов;
- тестирование работоспособности разработанной программы для различных исходных данных.

2. ВАРИАНТЫ ЗАДАНИЙ

Разработать консольные приложения на языке C++, решающие следующую задачу: на интервале $[a, b]$ с шагом dx вычислить значения кусочно-заданной функции $F(x)$. Значения a, b, dx вводятся пользователем. При выводе результатов провести выравнивание при помощи манипуляторов.

1. $$F(x) = \begin{cases} \sin 2x + \operatorname{tg}\frac{x}{x+1}, x \le -2, \\ \sqrt[3]{|4x|+1} - \ln(3+x), -2 < x \le 1, \\ 7, x > 1. \end{cases}$$

2. $F(x)=\begin{cases}\sqrt[4]{|\sin^2(x)-2x-1|^3}, x<1,\\ \frac{2\,\mathrm{arctg}(x+1)+3x^7}{\cos^2\ln x+1}, 1\le x\le 4,\\ \pi, x>4.\end{cases}$

3. $F(x)=\begin{cases}\sqrt[3]{|25x+1|}-\cos\pi x\,, x\le 0,\\ 4\,\mathrm{tg}\frac{\pi x}{5}+\sin 8x\,, 0<x<5,\\ 11, x\ge 5.\end{cases}$

4. $F(x)=\begin{cases}\sin\frac{x}{6}-3\sqrt{|x-5|}, x<-1{,}5,\\ \log_x(10+|x|)\ +10\cos\frac{2x+3}{7}, -1{,}5\le x\le 0,\\ 1, x>0.\end{cases}$

5. $F(x)=\begin{cases}\sqrt[5]{|x^3-4|+1}+4\sin(3x+2)\,, x<2,\\ 2-xe^x+\mathrm{tg}\frac{x}{x+1}, 2\le x\le 6,\\ -4, x>6.\end{cases}$

6. $F(x)=\begin{cases}3x^2+\sqrt[5]{10|x|^3+2}, x\le -3,\\ \sin\frac{\pi x}{5}-3\sqrt{\ln|x|^2}\,, -3<x\le 1,\\ 1, x>1.\end{cases}$

7. $F(x)=\begin{cases}\sqrt[2]{|5\sin x|+2}-\cos\pi x\,, x<2,\\ 3\,\mathrm{tg}\frac{\pi x}{7}+\sin\cos x\,, 1\le x\le 4,\\ \pi, x>4.\end{cases}$

8. $F(x)=\begin{cases}\log_2\left(1+\frac{|3+x|}{3-x}\right)+4\sin\frac{x}{2}, x\le 1,\\ 8\cos\frac{\pi x}{3}-\left(10+\frac{x^3}{4}\right)^2\,, 1<x<6,\\ \pi, x\ge 6.\end{cases}$

9. $F(x)=\begin{cases}\sqrt[3]{|24-x^2|}-\cos\frac{x-1}{x+1}, x<5,\\ 2+\mathrm{tg}\frac{\pi x}{2}-x(|x|+1)^2\,, 5\le x\le 9,\\ -4, x>9.\end{cases}$

10. $F(x)=\begin{cases}\sqrt[3]{2+\left|4-\frac{\sin x}{x+1}\right|}+\cos\pi x\,, x<\pi,\\ e^{x+1}+3\ln(|x^2-2x|+2)+5\,, \pi\le x\le 3\pi,\\ 1, x>3\pi.\end{cases}$

11. $$F(x)=\begin{cases}\lg(5+|7x|)+3\pi x\,, x<1,\\ \frac{2\sin x^2+2x^5}{\cos 4x+1}, 1\le x<5,\\ -2, x\ge 5.\end{cases}$$

12. $$F(x)=\begin{cases}\lg(x^4+x^2)\,, x\le -6,\\ 4\,\mathrm{tg}\frac{\pi x}{10}-\sqrt[3]{|5x^2+x|+3}\,, -6<x<-4,\\ 5, x\ge -4.\end{cases}$$

13. $$F(x)=\begin{cases}\sqrt[4]{|x^2-\cos x+2|^5}, x<5,\\ \frac{3x}{x+2}\mathrm{tg}(x+1)+2\pi\lg(x+35)\,, 5\le x\le 10,\\ 10, x>10.\end{cases}$$

14. $$F(x)=\begin{cases}20-\frac{3x}{5}\cos\frac{x}{2}+\mathrm{tg}\frac{x}{x-1}, x<-3,\\ \sqrt[2]{|5x^2-\pi|}+4, -3\le x\le -1,\\ 0, x>-1.\end{cases}$$

15. $$F(x)=\begin{cases}2e^{x+3}+3\sin(|x^3-x|+2)\,, x<2,\\ \frac{\ln(x+1)+3x^5}{\cos^2 x}, 2\le x<4,\\ 1, x\ge 4.\end{cases}$$

16. $$F(x)=\begin{cases}\lg(|x^2+5x|-3)+x^2\sin x\,, x\le 1,\\ \cos\frac{x+7}{x-7}+3\,\mathrm{tg}\frac{x}{x+2}, 1<x<4,\\ 11, x\ge 4.\end{cases}$$

17. $$F(x)=\begin{cases}3xe^{3x}+\sin(x^2+2)\,, x<-3,\\ \frac{x}{x+2}\mathrm{tg}(x^2+1)+2\pi\ln(x+30)\,, -3\le x\le 4,\\ 0, x>4.\end{cases}$$

18. $$F(x)=\begin{cases}\sin^2(2x+3)-x\sqrt[3]{4x^2+3}+9, x<4,\\ \lg(x^3+x^2+3)-5\sin x+1\,, 4\le x\le 6,\\ 8, x>6.\end{cases}$$

19. $$F(x)=\begin{cases}\pi x-\frac{x}{5}\lg\left(15+\frac{3|x|}{10}\right)+\frac{x}{x+1}, x<0,\\ 5+\frac{x}{3x-2}-\sqrt{8+(x+1)^3}, 0\le x<4,\\ 3, x\ge 4.\end{cases}$$

20. $$F(x)=\begin{cases}\sqrt[3]{x^4+1}-\ln(x+3), x\le -1,\\ \cos\frac{\pi x}{5}+\sin 3x+e^{2x}, -1<x<1,\\ -3, x>1.\end{cases}$$

21. $$F(x)=\begin{cases}7\sin^3 x+\sqrt[5]{10x^2+3}, x<-6,\\ \lg\left(1+\frac{x+3}{x-3}\right)+\cos\frac{x}{2}, -6\le x\le -1,\\ -\pi, x>-1.\end{cases}$$

22. $$F(x)=\begin{cases}\sqrt[5]{|x^3-4|+1}+4\sin\frac{\pi x}{5}, x<0,\\ \operatorname{ctg}(x+1)+\pi x^5, 0\le x\le 3,\\ 9, x>3.\end{cases}$$

23. $$F(x)=\begin{cases}\log_4(x^2+|x|+5)-4\sin x, x<7,\\ \sqrt{x^3+2}-\sin\frac{2x}{4x+1}, 7\le x<9,\\ 2, x\ge 9.\end{cases}$$

24. $$F(x)=\begin{cases}\sqrt[3]{|4x^2-x|+1}+2\sin\frac{\pi x}{3}, x\le 2,\\ \cos\frac{x+3}{2}+\operatorname{tg}\frac{x-2}{x+2}, 2<x<5,\\ 15, x\ge 5.\end{cases}$$

25. $$F(x)=\begin{cases}\frac{2x}{x-5}\sin(2x+1)+2\pi\ln|x-6|, x<-6,\\ \sqrt[3]{4|x|^5+1}-\ln\left|\frac{x+6}{x-6}\right|, -6\le x\le -1,\\ 1, x>-1.\end{cases}$$

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое циклический алгоритм?
2. Какие виды циклов существуют, в чем их принципиальные отличия?
3. Что означает цикл с предусловием и какие операторы реализуют его в C++?
4. Что означает цикл с постусловием и как он реализуется на языке C++?
5. Как изображаются циклы в графической схеме алгоритма?
6. Что такое тело цикла и каким образом в него можно включить несколько операторов?

7. Что означает вычислить с точностью ε сумму бесконечного числа слагаемых?
8. Как записать вычисление суммы с заданной точностью ε при использовании оператора цикла while?
9. Назовите особенности использования цикла со счетчиком.
10. Какие функции содержит заголовочный файл <math.h>?
11. Какие параметры вывода можно изменить с помощью манипуляторов?

ЛАБОРАТОРНАЯ РАБОТА № 3
ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью работы является получение обучающимися навыков работы со средствами обработки одномерных массивов в языке C++.

Задачами работы являются:

- написание программ на языке C++ с использованием одномерных массивов;
- тестирование работоспособности программы для различных исходных данных.

2. ВАРИАНТЫ ЗАДАНИЙ

Разработать консольные приложения на языке C++, решающие следующие задачи:

1. Дан целочисленный массив A размера N. Вывести номера тех его элементов A[i], которые удовлетворяют двойному неравенству: A[0] < A[i] < A[N-1]. Если таких элементов нет, то вывести сообщение. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [0; 10k], где k – номер студента в списке группы.

2. Дан целочисленный массив размера N. Преобразовать его, прибавив к четным числам первый элемент. Первый и последний элементы массива не изменять. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-10k; 5k], где k – номер студента в списке группы.
3. Дан целочисленный массив размера N. Вывести сначала все его элементы, являющиеся четными числами, а затем – нечетными. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-5k; 2k], где k – номер студента в списке группы.
4. Поменять местами минимальный и максимальный элементы одномерного массива размера N. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-6k; 0], где k – номер студента в списке группы.
5. Заменить все отрицательные элементы целочисленного массива размера N на минимальное значение элементов массива. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-3k; 4k], где k – номер студента в списке группы.
6. Дан массив размера N. Найти среднее арифметическое отрицательных элементов массива. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [0; 20k], где k – номер студента в списке группы.
7. Проверить, образуют ли элементы целочисленного массива размера N арифметическую прогрессию. Если да, то вывести разность прогрессии, если нет – вывести сообщение. Прове-

сти тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-k; 3k], где k – номер студента в списке группы.

8. Дан массив ненулевых целых чисел размера N. Проверить, чередуются ли в нем положительные и отрицательные числа. Если чередуются, то вывести сообщение, если нет, то вывести номер первого элемента, нарушающего закономерность. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-5k; 5k], где k – номер студента в списке группы.

9. Даны два одномерных массива a и b одинакового размера N. Найти max(a[i]-b[i]), $0 \le i<N$. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [0; 20k], где k – номер студента в списке группы.

10. Расположить элементы данного массива в обратном порядке (первый элемент меняется с последним, второй – с предпоследним и т.д. до среднего элемента). Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-3k; 0], где k – номер студента в списке группы.

11. В заданном одномерном массиве максимальный элемент заменить значением суммы предшествующих ему элементов. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [0; 10k], где k – номер студента в списке группы.

12. В данном массиве поменять местами элементы, стоящие на нечетных местах, с элементами, стоящими на четных местах. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива

из диапазона [-2k; 5k], где k – номер студента в списке группы.

13. Найти сумму нечетных элементов, стоящих на нечетных местах (то есть имеющих нечетные номера). Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-4k; 2k], где k – номер студента в списке группы.
14. Дан одномерный целочисленный массив. Необходимо “сжать” массив, удалив из него каждый второй элемент, а оставшиеся элементы заменить нулями. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-k; 0], где k – номер студента в списке группы.
15. Дан целочисленный массив и числа a и b (вводятся пользователем). Переместить в начало массива элементы, значение которых находится в промежутке [a, b]. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-3k; 4k], где k – номер студента в списке группы.
16. Задан массив с количеством элементов N. Сформируйте два массива: в первый включите элементы исходного массива с четными номерами, а во второй – с нечетными. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-k; 3k], где k – номер студента в списке группы.
17. Дана последовательность N целых чисел и целое число m. Указать пары чисел этой последовательности таких, что их сумма равна m. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [0; k], где k – номер студента в списке группы.

18. Дано N чисел. Наименьший член этой последовательности заменить значением среднего арифметического всех элементов, остальные элементы оставить без изменения. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-2k; 2k], где k – номер студента в списке группы.
19. Дана последовательность из N различных целых чисел. Найти сумму ее членов, расположенных между максимальным и минимальным значениями (в сумму включить и оба этих числа). Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [0; 100k], где k – номер студента в списке группы.
20. Даны два массива a и b одинаковой размерности N. Найти min|a[i]-b[i]|, 0<i<N. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-k; 5k], где k – номер студента в списке группы.
21. Найти сумму всех четных элементов массива, стоящих на четных местах, то есть имеющих четные номера. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [0; 10k], где k – номер студента в списке группы.
22. В одномерном массиве заменить нулями все положительные элементы, идущие после минимального элемента массива. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-4k; 4k], где k – номер студента в списке группы.
23. В заданном массиве определить среднее арифметическое значение положительных и среднее арифметическое значение

отрицательных элементов. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-2k; 3k], где k – номер студента в списке группы.

24. Найти среднее геометрическое положительных элементов одномерного вещественного массива. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [-2k; 2k], где k – номер студента в списке группы.
25. Даны два одномерных целочисленных массива, состоящие из одинакового числа элементов. Получить третий массив той же размерности, каждый элемент которого равен большему из соответствующих элементов данных массивов. Провести тестирование программы при пользовательском вводе элементов массива и при генерации элементов массива из диапазона [0; 3k], где k – номер студента в списке группы.

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое массив
2. Какие виды массивов существуют?
3. Как описывается массив?
4. Каких типов могут быть элементы массива?
5. Какими способами можно обращаться к элементам массива?
6. Каково назначение индекса и какие значения он может принимать на языке C++?
7. Какое начальное значение имеет индекс массива?
8. Что произойдет при выходе индекса массива за пределы указанного размера?
9. Как заполнить массив с помощью генератора случайных числе?
10. Как выводить символы русского алфавита?

ЛАБОРАТОРНАЯ РАБОТА № 4
УКАЗАТЕЛИ. ДИНАМИЧЕСКИЕ МАССИВЫ.

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью работы является получение обучающимися навыков работы с динамической памятью в языке C++.

Задачами работы являются:

- написание программы на языке C++ с использованием указателей при обработке динамических массивов;
- тестирование работоспособности программы для различных исходных данных.

2. ВАРИАНТЫ ЗАДАНИЙ

Разработать консольные приложения на языке C++, решающие следующие задачи:

1. Решить задачу для динамического массива, не используя индексацию. Дан массив размера N. Найти среднее арифметическое отрицательных элементов массива. Вывести адреса четных элементов исходного массива.
2. Решить задачу для динамического массива, не используя индексацию. Проверить, образуют ли элементы целочисленного массива размера N арифметическую прогрессию. Если да, то вывести разность прогрессии, если нет – вывести сообщение. Вывести адреса отрицательных элементов исходного массива.
3. Решить задачу для динамического массива, не используя индексацию. Дан массив ненулевых целых чисел размера N. Проверить, чередуются ли в нем положительные и отрицательные числа. Если чередуются, то вывести сообщение, если нет, то вывести номер первого элемента, нарушающего зако-

номерность. Вывести адрес минимального элемента исходного массива.

4. Решить задачу для динамического массива, не используя индексацию. Заменить все положительные элементы целочисленного массива, состоящего из N элементов, на значение максимального элемента. Вывести адреса нечетных элементов исходного массива.
5. Решить задачу для динамического массива, не используя индексацию. Расположить элементы данного массива в обратном порядке (первый элемент меняется с последним, второй – с предпоследним и т.д. до среднего элемента). Вывести адреса k последних элементов исходного массива.
6. Решить задачу для динамического массива, не используя индексацию. В заданном одномерном массиве максимальный элемент заменить значением суммы предшествующих ему элементов. Вывести адреса отрицательных элементов исходного массива.
7. Решить задачу для динамического массива, не используя индексацию. В данном массиве поменять местами элементы, стоящие на нечетных местах, с элементами, стоящими на четных местах. Вывести адреса нулевых элементов исходного массива.
8. Решить задачу для динамического массива, не используя индексацию. Найти сумму нечетных элементов, стоящих на нечетных местах (то есть имеющих нечетные номера). Вывести адрес максимального элемента исходного массива.
9. Решить задачу для динамического массива, не используя индексацию. Дан одномерный целочисленный массив. Необходимо “сжать” массив, удалив из него каждый второй элемент, а оставшиеся элементы заменить нулями. Вывести адреса k первых элементов исходного массива.

10. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив и числа a и b. Переместить в начало массива элементы, значение которых находится в промежутке [a, b]. Вывести адрес минимального элемента исходного массива.
11. Решить задачу для динамического массива, не используя индексацию. Задан массив с количеством элементов N. Сформируйте два массива: в первый включите элементы исходного массива с четными номерами, а во второй – с нечетными. Вывести адреса элементов исходного массива, больших M.
12. Решить задачу для динамического массива, не используя индексацию. Дана последовательность N целых чисел и целое число m. Указать пары чисел этой последовательности таких, что их сумма равна m. Вывести адреса нулевых элементов исходного массива.
13. Решить задачу для динамического массива, не используя индексацию. Дано N чисел. Наименьший член этой последовательности заменить значением среднего арифметического всех элементов, остальные элементы оставить без изменения. Вывести адреса четных элементов исходного массива.
14. Решить задачу для динамического массива, не используя индексацию. Дана последовательность из N различных целых чисел. Найти сумму ее членов, расположенных между максимальным и минимальным значениями (в сумму включить и оба этих числа). Вывести адреса отрицательных элементов исходного массива.
15. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив размера N. Преобразовать его, прибавив к отрицательным числам последний элемент. Первый и последний элементы массива не изменять. Вывести адрес минимального элемента исходного массива.

16. Решить задачу для динамического массива, не используя индексацию. Найти сумму всех четных элементов массива, стоящих на четных местах, то есть имеющих четные номера. Вывести адреса k последних элементов исходного массива.
17. Решить задачу для динамического массива, не используя индексацию. В одномерном массиве заменить нулями все положительные элементы, идущие после минимального элемента массива. Вывести адреса элементов, значение которых находится в промежутке [a, b].
18. Решить задачу для динамического массива, не используя индексацию. В заданном массиве определить среднее арифметическое значение положительных и среднее арифметическое значение отрицательных элементов. Вывести адреса нечетных элементов исходного массива.
19. Решить задачу для динамического массива, не используя индексацию. Даны два одномерных целочисленных массива, состоящие из одинакового числа элементов. Получить третий массив той же размерности, каждый элемент которого равен большему из соответствующих элементов данных массивов. Вывести адреса нулевых элементов исходного массива.
20. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив A размера N. Вывести номера тех его элементов, которые больше первого и меньше последнего элементов. Если таких элементов нет, то вывести сообщение. Вывести адреса k первых элементов исходного массива.
21. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив размера N. Преобразовать его, прибавив к четным числам первый элемент. Первый и последний элементы массива не изменять. Вывести адрес максимального элемента массива.

22. Найти среднее геометрическое положительных элементов одномерного вещественного массива. Вывести адреса элементов, значение которых находится в промежутке [a, b].
23. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив размера N. Вывести сначала все его элементы, являющиеся четными числами, а затем – нечетными. Вывести адреса элементов, значение которых находится вне интервала [a, b].
24. Решить задачу для динамического массива, не используя индексацию. Поменять местами минимальный и максимальный элементы одномерного массива размера N. Вывести адрес минимального элемента исходного массива.
25. Решить задачу для динамического массива, не используя индексацию. Дан массив, состоящий из N элементов. Переставить в обратном порядке элементы массива, расположенные между его минимальным и максимальным элементами. Вывести адреса элементов исходного массива, больших M.

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое указатель, динамическая память?
2. Всегда ли эффективно использование динамической памяти в программах?
3. Почему ошибки при работе с динамической памятью относят к опасным?
4. Опишите последствия использования в программах неинициализированных указателей.
5. Почему выделенную ранее динамическую память следует освобождать после использования?
6. К каким последствиям в работе программы приводит попытка освободить динамическую память, не выделенную ранее?

7. Почему к нулевому указателю нельзя применить операцию разыменования?
8. Какая связь между массивом и указателем на его нулевой элемент?

ЛАБОРАТОРНАЯ РАБОТА № 5
ОБРАБОТКА МНОГОМЕРНЫХ МАССИВОВ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью работы является получение обучающимися навыков работы со средствами обработки двумерных массивов в языке C++.

Задачами работы являются:

- написание программы на языке C++ с использованием двумерных массивов;
- тестирование работоспособности программы для различных исходных данных.

2. ВАРИАНТЫ ЗАДАНИЙ

Разработать консольное приложение, выполняющее задания, представленные ниже тремя способами:

- с использованием статического массива;
- с использованием одномерного динамического массива;
- с использованием двумерного динамического массива.

1. Дана целочисленная матрица размера M × N. Указать номера столбцов, все элементы которых различны.
2. Дана матрица размера A × B. В каждой строке найти количество элементов, больших среднего арифметического всех элементов этой строки.

3. Дана целочисленная матрица размера A × B. Вывести номер ее первой строки, содержащей равное количество положительных и отрицательных элементов (нулевые элементы не учитываются). Если таких строк нет, то вывести сообщение.
4. Дана действительная матрица размером M × N. Требуется преобразовать матрицу следующим образом: поэлементно вычесть последнюю строку из всех строк, кроме последней.
5. Дана целочисленная матрица размера M × N. Указать номера строк, все элементы которых различны.
6. Даны два целых числа m и n и матрица размера A × B. Поменять местами столбцы матрицы с номерами m и n и найти произведение их элементов.
7. Дана матрица размера A × B. Преобразовать матрицу, поменяв местами минимальный и максимальный элемент в каждой строке.
8. Дана квадратная матрица размера M. Найти сумму положительных элементов ее главной и побочной диагонали. При их отсутствии вывести сообщение.
9. Задана квадратная матрица и число m. Поменять местами строку с максимальным элементом на главной диагонали со строкой с номером m.
10. Дана матрица размера A × B. Найти минимальное значение в каждой строке и столбце.
11. Вычислить сумму и число положительных элементов квадратной матрицы размера N × N, находящихся над главной диагональю.
12. Дана матрица размером M × N. Определить k – количество особых элементов массива, считая его элемент особым, если он больше суммы остальных элементов его столбца.
13. Дана квадратная матрица размера M. Заменить нулями элементы матрицы, лежащие ниже главной диагонали.

14. Дана целая квадратная матрица N × N. Определить, является ли она магическим квадратом, т.е. такой матрицей, в которой суммы элементов во всех строках и столбцах одинаковы.
15. Дана матрица размера A × B. Найти сумму элементов всех ее нечетных столбцов.
16. Заданы целочисленная квадратная матрица N × N и число k. Разделить элементы m -й строки на диагональный элемент, расположенный в этой строке. Разделить элементы m -го столбца на диагональный элемент, расположенный в этом столбце.
17. Дана квадратная матрица порядка M. Заменить нулями отрицательные элементы матрицы, лежащие выше главной диагонали.
18. Дана целочисленная матрица размера A × B. Найти произведения четных элементов ее четных строк.
19. Дана целочисленная квадратная матрица. Найти в каждой строке наибольший элемент и поменять его местами с элементом главной диагонали.
20. Дана целочисленная матрица размера A × B. Преобразовать матрицу, поменяв местами минимальный и максимальный элементы в каждом столбце.
21. Даны два числа m и n и матрица размера A × B. Поменять местами строки матрицы с номерами m и n и найти произведение их элементов.
22. Дана прямоугольная матрица размера A × B. Найти строки с наибольшей и наименьшей суммой элементов. Вывести на печать найденные строки и суммы их элементов.
23. Определить, является ли целочисленная квадратная матрица N-го порядка симметричной относительно главной диагонали.
24. Дана целочисленная матрица размера M×N. Указать номера столбцов, все элементы которых различны.

25. Дана целочисленная матрица размера A × B. Вывести номер ее первого столбца, содержащего равное количество положительных и отрицательных элементов (нулевые элементы не учитываются).

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое многомерный массив и как он объявляется?
2. Какими способами можно обращаться к элементам массива?
3. Каково назначение индексов в многомерном массиве?
4. Какова общая схема работы с многомерным массивом?
5. Как организовать вложенные циклы?
6. Как представить двумерный массив в виде одномерного?
7. Как осуществляется работа с двумерным динамическим массивом?

ЛАБОРАТОРНАЯ РАБОТА № 6
СИМВОЛЬНЫЕ И СТРОКОВЫЕ ДАННЫЕ

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью работы является получение обучающимися навыков реализации алгоритмов обработки строк на языке C++.

Задачами работы являются:

- написание программы на языке C++ для обработки информации, представленной в виде строк;
- написание программы на языке C++ с использованием функций библиотеки string;
- тестирование работоспособности разработанной программы для различных исходных данных.

2. ВАРИАНТЫ ЗАДАНИЙ

Разработать консольное приложение, выполняющее задания, представленные ниже двумя способами:

- без использования функций заголовочного файла <string.h> (обрабатывая строки, как массивы символов в циклах);
- с использованием функций заголовочного файла <string.h>.

1. Ввести предложение длиной не более 80 символов. Определить длину его первого слова и количество слов короче первого. Вывести эти слова. Количество пробелов между словами произвольно.
2. Ввести предложение длиной не более 80 символов. Вывести слова, которые заканчиваются на ту же букву, что и последнее слово, и их количество. Количество пробелов между словами произвольно
3. Ввести предложение длиной не более 80 символов. Вывести слова, которые начинаются на букву из эталонной строки, и их количество. Количество пробелов между словами произвольно.
4. Ввести предложение длиной не более 80 символов. Определить длину его последнего слова и количество слов короче последнего. Вывести эти слова. Количество пробелов между словами произвольно.
5. Ввести предложение длиной не более 80 символов. Определить среднюю длину слов и количество слов короче средней. Вывести эти слова. Количество пробелов между словами произвольно
6. Ввести предложение длиной не более 80 символов. Вывести слова, которые заканчиваются на ту же букву, что и второе слово, и их количество. Количество пробелов между словами произвольно.

7. Ввести предложение длиной не более 80 символов. Вывести слова, которые начинаются на ту же букву, что и первое слово, и их количество. Количество пробелов между словами произвольно.
8. Ввести предложение длиной не более 80 символов. Определить самое длинное слово и количество слов короче его. Вывести эти слова. Количество пробелов между словами произвольно.
9. Ввести предложение длиной не более 80 символов. Определить длину его предпоследнего слова и количество слов короче предпоследнего. Вывести эти слова. Количество пробелов между словами произвольно.
10. Ввести предложение длиной не более 80 символов. Вывести слова, которые заканчиваются на ту же букву, что и первое слово, и их количество. Количество пробелов между словами произвольно.
11. Ввести предложение длиной не более 80 символов. Определить среднюю длину слов и количество слов длиннее средней Вывести эти слова. Количество пробелов между словами произвольно.
12. Ввести предложение длиной не более 80 символов. Вывести слова, которые начинаются на ту же букву, что и предпоследнее слово, и их количество. Количество пробелов между словами произвольно
13. Ввести предложение длиной не более 80 символов. Определить самое короткое слово и количество слов длиннее его. Вывести эти слова. Количество пробелов между словами произвольно.
14. Ввести предложение длиной не более 80 символов. Вывести слова, которые оканчиваются на букву из эталонной строки, и их количество. Количество пробелов между словами произвольно.

15. Ввести предложение длиной не более 80 символов. Вывести слова, которые начинаются на ту же букву, что и последнее слово, и их количество. Количество пробелов между словами произвольно.
16. Ввести предложение длиной не более 80 символов. Определить длину его последнего слова и количество слов длиннее последнего. Вывести эти слова. Количество пробелов между словами произвольно.
17. Ввести предложение длиной не более 80 символов. Определить длину его второго слова и количество слов короче второго. Вывести эти слова. Количество пробелов между словами произвольно.
18. Ввести предложение длиной не более 80 символов. Вывести слова, которые заканчиваются на ту же букву, что и предпоследнее слово, и их количество. Количество пробелов между словами произвольно.
19. Ввести предложение длиной не более 80 символов. Вывести слова, которые начинаются и заканчиваются на одну ту же букву, и их количество. Количество пробелов между словами произвольно.
20. Ввести предложение длиной не более 80 символов. Определить длину его второго слова и количество слов длиннее второго. Вывести эти слова. Количество пробелов между словами произвольно.
21. Ввести предложение длиной не более 80 символов. Определить длину его первого слова и количество слов длиннее первого. Вывести эти слова. Количество пробелов между словами произвольно
22. Ввести предложение длиной не более 80 символов. Определить длину его предпоследнего слова и количество слов длиннее предпоследнего. Вывести эти слова. Количество пробелов между словами произвольно.

23. Ввести предложение длиной не более 80 символов. Вывести слова, которые начинаются на ту же букву, что и второе слово, и их количество. Количество пробелов между словами произвольно.
24. Ввести предложение длиной не более 80 символов. Определить длину его второго слова и количество слов короче второго. Вывести эти слова. Количество пробелов между словами произвольно.
25. Ввести предложение длиной не более 80 символов. Вывести слова, которые заканчиваются на ту же букву, что и предпоследнее слово, и их количество. Количество пробелов между словами произвольно

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Почему в языке C++ определена строгая типизация данных, используемых в программе?
2. Почему в C++ символьный тип считается подмножеством целочисленного типа?
3. Что такое строка в C++
4. Почему в C++ не выполняется операция прямого присваивания значения строке?
5. Почему символ и строка, состоящая из одного символа, занимают разный объем памяти?
6. Допустима ли операция сравнения над символами? Если да, то каким образом определены отношения "больше" и "меньше"?
7. Можно ли выполнить присваивание символьной переменной числового значения?
8. Что будет являться результатом работы функции побайтового копирования строк, если длина строки-источника превосходит допустимый размер строки-приемника?

9. Что будет являться результатом работы функции побайтового копирования строк, если длина строки-источника меньше размера строки-приемника?
10. Почему при сравнении строк важен регистр символов?
11. Как сравниваются строки разной длины?

ЛАБОРАТОРНАЯ РАБОТА № 7
ФУНКЦИИ В C++

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Целью работы является получение обучающимися навыков разработки и использования функций на языке C++.

Задачами работы являются:

- создании функций на языке C++ для обработки целочисленных данных;
- написание программы с использованием разработанных функций для решения комплексной задачи;
- тестирование работоспособности разработанной программы для различных исходных данных.

2. ВАРИАНТЫ ЗАДАНИЙ

1. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию int NumToDigit(int n) для определения итоговой цифры числа (итоговой цифрой числа называется число, состоящее из одной цифры, получаемое при последовательном сложении цифр числа, сложении цифр полученного числа и т.д. до конечного результата, состоящего из одной цифры). Решить задачу: определить количество чисел на интервале [a, b], итоговая цифра которых совпадает с введенной пользователем цифрой с использованием разработанных функций.

2. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsPres(int n, int m) для определения, содержит ли число n цифру m. Решить задачу: определить количество нечетных чисел, содержащих введенную пользователем цифру, на интервале [a, b] с использованием разработанных функций.

3. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsDivOnDigit(int n) для определения, делится ли число на каждую свою цифру. Решить задачу: сколько чисел на интервале [a, b] делятся на каждую свою цифру с использованием разработанных функций.

4. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsSimple(int n) для определения, является ли число простым. Описать функцию bool HasRep(int n) для определения, содержит ли число хотя бы две одинаковые цифры. Решить задачу: сколько простых чисел на интервале [a, b] содержат одинаковые цифры с использованием разработанных функций.

5. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool DigitEven(int n) для определения являются ли все цифры в числе четными. Описать функцию bool HasRep(int n) для определения, содержит ли число все одинаковые цифры. Решить задачу: сколько чисел, содержащих одинаковые цифры и все цифры являются четными, на интервале [a, b] с использованием разработанных функций.

6. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsSimple(int n) для определения, является ли число простым. Описать функцию bool IsAcsend(int n) для определе-

ния, является ли последовательность цифр в числе возрастающей. Решить задачу: сколько простых чисел, цифры которых образуют возрастающую последовательность, на интервале [a, b] с использованием разработанных функций.

7. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool HasDif(int n) для определения, содержит ли число различные цифры. Описать функцию bool IsAcsend(int n) для определения, является ли последовательность цифр в числе возрастающей. Решить задачу: сколько чисел на интервале [a, b] содержат различные цифры и образуют возрастающую последовательность с использованием разработанных функций.

8. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool SumDigIsM(int n, int m) для определения, равна ли сумма цифр числа n числу m. Решить задачу: определить количество чисел на интервале [a, b], сумма цифр которых совпадает с введенным пользователем числом с использованием разработанных функций.

9. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool Acsend(int n), которая упорядочивает цифры числа в порядке возрастания и формирует из них новое целое число. Решить задачу: сколько чисел на интервале [a, b] содержит те же цифры, что и число, введенное пользователем, с использованием разработанных функций.

10. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsSimple(int n) для определения, является ли число простым. Описать функцию int CountZeros(int n) для определения количества нулей в числе n. Решить задачу: определить количе-

ство простых чисел, содержащих максимальное количество нулей, на интервале [a, b] с использованием разработанных функций и вывести их на экран.

11. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию int Divisor(int n), которая возвращает количество простых множителей в разложение числа. Решить задачу: сколько чисел на интервале [a, b] содержит максимальное количество простых множителей с использованием разработанных функций.

12. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsCubeDigits(int n) которая определяет, равно ли число сумме кубов своих цифр. Решить задачу: определить количество таких чисел на интервале [a, b] с использованием разработанных функций.

13. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsSimple(int n) для определения, является ли число простым. Описать функцию bool IsFail(int n, int m) для определения, отсутствует ли в числе n цифра m. Решить задачу: сколько простых чисел, в которых отсутствует введенная пользователем цифра, на интервале [a, b] с использованием разработанных функций.

14. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsSimple(int n) для определения, является ли число простым. Описать функцию bool DigitOdd(int n) для определения, являются ли все цифры в числе нечетными. Решить задачу: определить количество простых чисел, все цифры которых являются нечетными цифрами, на интервале [a, b] с использованием разработанных функций.

15. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsAcsend(int n) для определения, является ли последовательность цифр в числе возрастающей. Описать функцию bool IsPres(int n, int m) для определения, содержит ли число n цифру m. Решить задачу: сколько чисел состоят из цифр, образующих возрастающую последовательность и содержащих введенную пользователем цифру, на интервале [a, b] с использованием разработанных функций.

16. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsHappy(int n) для определения, является ли число «счастливым» («счастливым» называется число, в котором сумма первых n/2 цифр равна сумме последних n/2 цифр). Решить задачу: определить количество «счастливых» чисел на интервале [a, b] с использованием разработанных функций.

17. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsSimple(int n) для определения, является ли число простым. Описать функцию bool IsDescend(int n) для определения, является ли последовательность цифр в числе убывающей. Решить задачу: сколько простых чисел, цифры которых образуют убывающую последовательность, на интервале [a, b] с использованием разработанных функций.

18. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsPres(int n, int m) для определения, содержит ли число n цифру m. Решить задачу: определить количество четных чисел, содержащих введенную пользователем цифру, на интервале [a, b] с использованием разработанных функций.

19. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool HasDif(int n) для определения, содержит ли число различные цифры. Описать функцию bool IsHappy(int n) для определения, является ли число «счастливым» («счастливым» называется число, в котором сумма первых n/2 цифр равна сумме последних n/2 цифр). Решить задачу: определить количество «счастливых» чисел, содержащих различные цифры, на интервале [a, b] с использованием разработанных функций.

20. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsSimple(int n) для определения, является ли число простым. Описать функцию bool HasDif(int n) для определения, содержит ли число различные цифры. Решить задачу: сколько простых чисел на интервале [a, b] не содержат повторяющиеся цифры с использованием разработанных функций.

21. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool HasDif(int n) для определения, содержит ли число различные цифры. Описать функцию bool IsDescend(int n) для определения, является ли последовательность цифр в числе убывающей. Решить задачу: сколько чисел на интервале [a, b] содержат различные цифры и образуют убывающую последовательность с использованием разработанных функций.

22. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsSimple(int n) для определения, является ли число простым.Описать функцию bool SumDigIsM(int n, int m) для определения, равна ли сумма цифр числа n числу m. Решить задачу: определить количество простых чисел на интервале [a, b], сумма

цифр которых совпадает с введенным пользователем числом с использованием разработанных функций.

23. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool HasRep(int n) для определения, содержит ли число одинаковые цифры. Решить задачу: определить количество четных чисел, содержащих одинаковые цифры, на интервале [a, b] с использованием разработанных функций.

24. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию bool IsSimple(int n) для определения, является ли число простым. Описать функцию bool IsPres(int n, int m) для определения, содержит ли число n цифру m. Решить задачу: определить количество простых чисел, содержащих введенную пользователем цифру, на интервале [a, b] с использованием разработанных функций.

25. Описать функцию bool IsCorrect(int n1, int n2) для определения, является ли введенный пользователем интервал корректным. Описать функцию int SumDivisor(int n), которая возвращает сумму простых множителей при разложение числа. Решить задачу: у какого количества чисел на интервале [a, b] сумма простых множителей совпадает с числом, введенным пользователем, с использованием разработанных функций.

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое функция?
2. Как введение функции позволяет избежать избыточности кода?
3. Что такое прототип функции?
4. В каких случаях используется передача параметров по ссылке?

5. Что такое локальные и глобальные переменные?
6. Можно ли возвращать из функции указатель на локальную переменную?
7. Перечислите преимущества и недостатки использования глобальных переменных.
8. Как реализуется в C++ операция «частное от деления нацело»?
9. Как реализуется в C++ операция «остаток от деления нацело»?

ПРИЛОЖЕНИЕ 2. СПИСОК ЛИТЕРАТУРЫ ДЛЯ САМОСТОЯТЕЛЬНОГО ИЗУЧЕНИЯ

1. Павловская Т.А. C/C++ . Процедурное и объектно-ориентированное программирование: учеб. / Т. А. Павловская, .- М.: 2015.- 495 с
2. Павловская Т.А. C/C++.Программирование на языке высокого уровня: Учеб. / Т. А. Павловская.- М.: 2006.- 460 с
3. Фридман А.Л. Язык программирования Си++: Курс лекций / А. Л. Фридман, .- М.: 2003.- 282 с
4. Аляев Ю.А., Козлов О.А. Алгоритмизация и языки программирования Pascal, C++, Visual Basic: Учеб.-справ.пособие / Ю.А.Аляев, О.А.Козлов.- М.: 2002.- 319 с
5. Кучеренко В. Язык программирования C++ для начинающих и не только.- М.: 2001.- 160 с
6. Тюгашев А. А. Языки программирования: учеб. пособие / А. А. Тюгашев. - СПб. : Питер , 2014. - 333 с
7. Эллайн А. C++. От ламера до программера / А. Эллайн. - М. ; СПб. ; Нижний Новгород : Питер , 2015. - 475 с
8. Прата С. Язык программирования C++ : лекции и упр. / С. Прата. СПб. : Вильямс, 2007. - 1181 с
9. Васильев А.Н. Программирование на C++ в примерах и задачах / А.Н. Васильев. – М.: Эксмо- Пресс : , 2017. - 368 с

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Павловская Т.А. C/C++ . Процедурное и объектно-ориентированное программирование: учеб. / Т. А. Павловская, .- М.: 2015.- 495 с
2. Павловская Т.А. C/C++.Программирование на языке высокого уровня: Учеб. / Т. А. Павловская.- М.: 2006.- 460 с
3. Фридман А.Л. Язык программирования Си++: Курс лекций / А. Л. Фридман, .- М.: 2003.- 282 с
4. Аляев Ю.А., Козлов О.А. Алгоритмизация и языки программирования Pascal, C++, Visual Basic: Учеб.-справ.пособие / Ю.А.Аляев, О.А.Козлов.- М.: 2002.- 319 с
5. Кучеренко В. Язык программирования C++ для начинающих и не только.- М.: 2001.- 160 с

СОДЕРЖАНИЕ

Список используемых сокращений 3
ПРЕДИСЛОВИЕ 4
ВВЕДЕНИЕ 6
1. ОСНОВЫЕ СВЕДЕНИЯ О ЯЗЫКЕ И СРЕДСТВАХ РАЗРАБОТКИ 8
1.1. Разработка программ на языке C++ 8
1.2. Выбор среды разработки 9
1.3. Установка среды разработки 10
1.4. Начало разработки 13
1.5. Вопросы для самопроверки 17
1.6. Задания для самопроверки 17
2. ТИПЫ ДАННЫХ И РАБОТА С НИМИ 18
2.1. Работа с типами и их отличие 18
2.2. Пространства имен 21
2.3. Арифметические операции 22
2.4. Целочисленное деление 24
2.5. Простейшая программа на C++ 25
2.5. Средства ввода-вывода 27
2.6. Вопросы для самопроверки 29
2.7. Задания для самопроверки 30
3. УСЛОВНЫЕ ОПЕРАТОРЫ 30
3.1. Логический тип данных и логические операции 30
3.2. Условный оператор if 31
3.2. Условный оператор switch 34

3.3. Тернарная операция 37
3.4. Вопросы для самопроверки 38
3.5. Задания для самопроверки 39
4. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ 39
4.1. Циклы с условием 39
4.2. Циклы с параметром 41
4.3. Безусловные циклы 46
4.4. Вопросы для самопроверки 46
4.5. Задания для самопроверки 46
5. ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ 47
5.1. Понятие массива 47
5.2. Генерация случайных чисел 50
5.3. Вопросы для самопроверки 52
5.4. Задания для самопроверки 52
6. УКАЗАТЕЛИ. ДИНАМИЧЕСКИЕ МАССИВЫ 53
6.1. Указатели 53
6.2. Способы инициализации указателя 54
6.3. Связь указателей с массивами 58
6.4. Динамические массивы 59
6.5. Вопросы для самопроверки 62
6.6. Задания для самопроверки 62
7. ОБРАБОТКА МНОГОМЕРНЫХ МАССИВОВ 63
7.1. Многомерные массивы 63
7.2. Двумерные динамические массивы 67
7.3. Указатели и двумерные массивы 69
7.4. Двумерные динамические массивы 70

7.5. Вопросы для самопроверки 74
7.6. Задания для самопроверки 75
8. СИМВОЛЬНЫЕ И СТРОКОВЫЕ ДАННЫЕ 75
8.1. Объявление символьных данных и строк 75
8.2. Функции стандартной библиотеки для работы со строками 81
8.3. Вопросы для самопроверки 83
8.4. Задания для самопроверки 84
9. ФУНКЦИИ 84
9.1. Краткие теоретические сведения 84
9.2. Передача параметров в функции 91
9.3. Обработка целочисленных данных. 94
9.4. Вопросы для самопроверки 98
9.5. Задания для самопроверки 99
ЗАКЛЮЧЕНИЕ 99
ПРИЛОЖЕНИЕ 1. ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА 100
Порядок выполнения работ 100
Требования к оформлению отчета 100
Критерии результативности выполнения работы 100
ЛАБОРАТОРНАЯ РАБОТА № 1 101
ЛАБОРАТОРНАЯ РАБОТА № 2 105
ЛАБОРАТОРНАЯ РАБОТА № 3 109
ЛАБОРАТОРНАЯ РАБОТА № 4 115
ЛАБОРАТОРНАЯ РАБОТА № 5 120
ЛАБОРАТОРНАЯ РАБОТА № 6 123

ЛАБОРАТОРНАЯ РАБОТА № 7 128
ПРИЛОЖЕНИЕ 2. СПИСОК ЛИТЕРАТУРЫ ДЛЯ САМОСТОЯТЕЛЬНОГО ИЗУЧЕНИЯ..... 136
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 137
СОДЕРЖАНИЕ 138

Учебное издание

ЗОЛИН Алексей Георгиевич
КОЛОДЕНКОВА Анна Евгеньевна
ХАЛИКОВА Елена Анатольевна

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ.
ВВЕДЕНИЕ В РАЗРАБОТКУ НА C++
(первый семестр)

Редактор В.В. Прокопова
Компьютерная верстка *И.О. Миняева*
Выпускающий редактор *Ю.А. Петропольская*

Подписано в печать
Формат 60x84 1/16. Бумага офсетная
Усл. 8,11 п. л. Уч.-изд. л.
Тираж 100 экз. Рег. №

Федеральное государственное
бюджетное образовательное учреждение
высшего профессионального образования
«Самарский государственный технический университет»
443100, г. Самара, ул. Молодогвардейская, 244. Главный корпус

Отпечатано в типографии
Самарского государственного технического университета
443100, г. Самара, ул. Молодогвардейская, 244. Корпус №8

Вопросы к экзамену по дисциплине

«Языки и методы программирования»

1. Понятие алгоритма. Свойства алгоритма
2. Классификация алгоритмов по виду представления
3. Базовые алгоритмические структуры
4. Структурная (графическая) схема алгоритма
5. Оценка сложности алгоритма
6. Языки программирования. Поколения языков программирования
7. Парадигмы программирования
8. Области программирования
9. Язык программирования C++. Версии языка. Основные возможности. Сравнение с другими языками программирования
10. Структура программы на языке программирования C++
11. Этапы создания исполняемой программы
12. Алфавит языка программирования C++
13. Идентификаторы, знаки операций, зарезервированные слова, комментарии
14. Типы данных языка программирования C++, классификация
15. Целочисленный тип данных, специфика обработки целочисленных данных в C++
16. Константы, константные переменные, переменные (классификации, примеры)
17. Явное и неявное преобразования типов
18. Выражения, приоритет операций
19. Логический тип данных, логические выражения в языке программирования C++
20. Условный оператор, графическая схема, пример
21. Тернарная условная операция, пример
22. Оператор выбора, графическая схема, пример
23. Операторы цикла, классификация
24. Оператор цикла с предусловием, графическая схема, пример
25. Оператор цикла с постусловием, графическая схема, пример
26. Оператор цикла со счетчиком, графическая схема, пример
27. Вложенные алгоритмические структуры, графические схемы, примеры
28. Структура данных массив: основные понятия, достоинства, недостатки
29. Реализация статических одномерных массивов на языке программирования C++: заполнение, вывод на экран
30. Заполнение целочисленного одномерного массива с использованием генератора случайных чисел
31. Решение задачи определения минимального значения в целочисленном одномерном массиве
32. Решение задачи поиска элемента в целочисленном одномерном массиве
33. Решение задачи сортировки одномерного массива

34. Динамическая память. Хранение информации в оперативной памяти. Распределение памяти
35. Указатели на языке программирования C++: основные понятия, способы инициализации
36. Указатели на языке программирования C++: операции с указателями
37. Взаимосвязь указателей и одномерных массивов
38. Работа с динамической памятью с помощью операторов new и delete
39. Многомерные массивы: основные понятия, способы размещения в памяти
40. Статические двумерные массивы на языке программирования C++
41. Динамические двумерные массивы на языке программирования C++
42. Поток данных в программировании: основные понятия, классификация
43. Поток данных в программировании: буферизированный выходной поток
44. Поток данных в программировании: буферизированный входной поток
45. Потоковый ввод/вывод на языке программирования C++
46. Символьные данные на языке программирования C++
47. Строковые данные на языке программирования C++
48. Функции в программировании
49. Функции на языке программирования C++: описание, объявление, прототип, примеры
50. Функции на языке программирования C++: формальные и фактические параметры, примеры
51. Передача параметров в функцию по значению и по адресу (по ссылке), пример
52. Глобальные и локальные переменные
53. Функции на языке программирования C++: вызов функции, возврат результата с помощью оператора return
54. Передача массивов в качестве параметров функции
55. Функции на языке программирования C++: параметры по умолчанию
56. Функции заголовочного файла string.h (csting) для обработки строк
57. Функции заголовочного файла string.h (csting) для обработки символов
58. Функции заголовочного файла math.h (cmath)
59. Функции заголовочного файла iomanip.h
60. Интегрированная среда разработки. Среда разработки Microsoft Visual Studio

А.Г. ЗОЛИН

Е.А. ХАЛИКОВА

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ: ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ, СОРТИРОВКА И ПОИСК

Учебное пособие

Самара

Самарский государственный технический университет

2021

МИНОБРНАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

Кафедра «Информационные технологии»

А.Г. ЗОЛИН

Е.А. ХАЛИКОВА

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ: ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ, СОРТИРОВКА И ПОИСК

Учебное пособие

Самара

Самарский государственный технический университет

2021

Печатается по решению редакционно-издательского совета СамГТУ

УДК 519.682.2

Г 94

Золин А.Г.

Языки и методы программирования: динамические структуры данных, сортировка и поиск: учебное пособие / *А.Г. Золин, Е.А. Халикова* – Самара: Самар. гос. техн. ун-т, 2021. – с.: ил.

Рассмотрены реализация связных списков, алгоритмов сортировки и поиска на языке *C++*, в целях получения навыков программной реализации методов обработки данных. Описаны основные принципы объектно-ориентированного программирвания. Предназначено для студентов высших технических учебных заведений, обучающихся по направлениям 09.03.01 «Информатика и вычислительная техника», 09.03.03 «Прикладная информатика», 09.03.04 «Программная инженерия», 10.03.01 «Информационная безопасность», 11.03.01 «Радиотехника»

Р е ц е н з е н т

УДК 519.682.2

Г 94

© А.Г. Золин, Е.А. Халикова 2021

© Самарский государственный технический университет, 2021

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

ПК – персональный компьютер;

ПО – программное обеспечение;

ИСР – интегрированная среда разработки;

GUI – графический интерфейс пользователя;

LIFO (Last Input – First Output) – стековый метод доступа

FIFO (First Input – First Output) – метод доступа очереди

ПРЕДИСЛОВИЕ

Практика показывает, что при изучении программирования, наиболее сложным для студентов аспектом является развитие навыка разработки алгоритмов решения поставленной задачи. Формирование алгоритмического мышления является ключевым моментом в процессе становления будущего специалиста в области информационных технологий. Однако, процедуры составления последовательных шагов, приводящих к определенному результату, являются плохо формализуемыми, включают в себя элементы творчества, что приводит к необходимости искать различные эффективные способы обученя студентов основам программирования.

Одним из извстных подходов к развитию алгоритмического мышления, является изучение и анализ существующих программ, разбор реализованных в них алгоритмов, и далее, на этой основе переход к выполнению разработки собственных решений похожих задач, используя базовые принципы и приемы.

С другой стороны, разработка алгоримов тесно связана с организацией хранения данных, как в оперативной памяти, так и на внешних носителях. По мере усложнения процессов обработки, обучающийся неизбежно приходит к усложнению структур используемых данных. Поэтому при изучении программирования, необходимо развивать у студентов навыки структурировать информацию, уметь выполнять декомпозицию сущностей предметной области и на основе полученных результатов, разарабатывать высокоэффективные структуры данных, используемые в программах. Для этого обучающийся должен быть знаком с современными подходами к организации данных, знать наиболее используемые на практике виды структур и способы их формирования.

Для достижения поставленных задач обучения, хорошо подходит язык программирования C++, известный своей эффективностью, экономичностью и переносимостью, позволяющий также использовать объектно-ориентированный подход в программировании: C++ имеет средства для создания сложных типов данных, объединяющих в себе и наборы информации, описывающих тот или иной объект, и методы обработки этих данных. При этом язык имеет строгий, лаконичный синтаксис, что способствует развитию дисциплины написания исходного кода.

Использование C++ в качестве инструментального языка позволяет получать быстрые и компактные программы, а указанные преимущества обеспечивают высокое качество разработки программного продукта.

Предлагаемое пособие служит для самостоятельного изучения основ C++ в области разработки структурированных типов данных, алгоритмов сортировки и поиска и является дополнением к лекционным курсам по дисциплине «Языки и методы программирования» (второй семестр). Может использоваться бакалаврами, которые обучаются по направлениям: 09.03.01 «Информатика и вычислительная техника», 09.03.03 «Прикладная информатика», 09.03.04 «Программная инженерия», 10.03.01 «Информационная безопасность», 11.03.01 «Радиотехника», а также будет полезно всем, кто начинает первые шаги в изучении основ программирования и пытается познакомиться с основами алгоритмов сортировки, поиска и динамических структур данных.

ВВЕДЕНИЕ

Данное учебное пособие предназначено для студентов, изучающих дисциплину «Языки и методы программирования», базовым языком которой был выбран C++. Пособие охватывает материал второго семестра обучения и предоставляет как теоретический материал по основным разделам курса, так и задания для самостоятельного выполнения.

Данное пособие ориентировано на разработку программ на компьютерах с операционной системой MS Windows и средой программирования Visual Studio.

Теоретический материал разбит на семь глав. В первой главе даются основные сведения о структурированных типах данных. Приводится обобщенное понятие типа данных. Описываются средства языка для создания пользовательских структур и работы с ними.

Во второй главе даются основы работы с файлами. Дается общее понятие файла, а также средства языка C++ для работы с ними. Данная глава описывает работу с текстовыми файлами с использованием потоков ввода/вывода из стандартной библиотеки языка.

В третьей главе продолжено рассмотрение работы с файлами. Здесь представлен материал по работе с двоичными файлами. Описаны методы произвольного доступа к содержимому файла. Даны способы записи объектов в бинарные файлы.

Четвертая глава вводит понятие класса C++. Приводится краткое описание основных парадигм объектно-ориентированного программирования. Более подробно рассмотрен принцип инкапсуляции. Дается определение конструктора, деструктора класса, а также дружественных функций.

Пятая глава посвящена связным спискам. В ней рассмотрены линейные односвязные и двусвязные списки. Представлена примерная реализация линейного односвязного списка.

В шестой главе рассматриваются алгоритмы сортировок. Приводятся алгоритмы элементарных методов сортировки: выбором, пузырьком, Шелла. Дается сравнительный анализ различных алгоритмов сортировки.

Седьмая глава посвящена алгоритмам поиска. Здесь рассмотрены варианты поиска элементов в массивах: поиск перебором, поиск с барьером, бинарный поиск. Отдельно рассматриваются алгоритмы нахождения подстроки в тексте: алгоритм Кнута, Морриса и Пратта и алгоритм Боуера и Мура.

В завершение пособия в приложении 1 даются задания для лабораторного практикума. В приложении 2 представлен список книг, рекомендуемых для самостоятельного изучения.

1. СТРУКТРИРОВАННЫЕ ТИПЫ ДАННЫХ

1.1. Типы данных в языке C++

Тип данных в языке программирования позволяет задавать множество значений и операций над этими значениями. Он определяет диапазон возможных значений, их смысл, операции, допустимые с этими значениями, а также способы и формат хранения. Неотъемлемой частью большинства языков программирования являются системы типов, позволяющие контролировать процесс компиляции (интерпретации) на предмет корректного использования данных в программе и выполнять оптимизацию сгенерированного кода на более высоком уровне.

Тип данных определяет:

- внутреннее представление данных в памяти компьютера;

– множество значений, которые могут принимать величины этого типа;
– операции и функции, которые можно применять к величинам этого типа.

В программировании используется низкоуровневое определение типа – как заданных размерных и структурных характеристик ячейки памяти, в которую можно поместить некое значение, соответствующее этим характеристикам. На практике с этим связан ряд важных свойств (обусловленных особенностями организации памяти компьютера).

Существует несколько классификаций типов данных в зависимости от признака-классификатора, в дальнейшем используется классификация, представленная на рисунке 1.1.

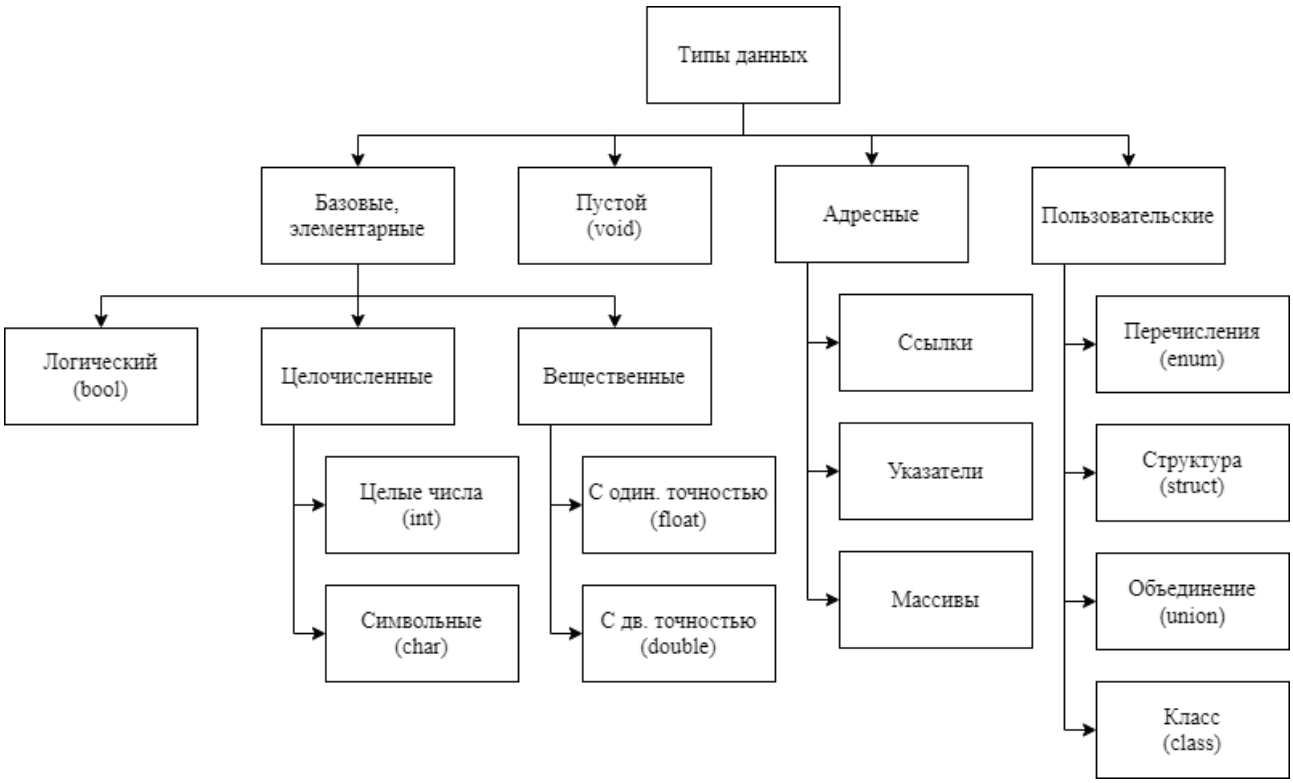


Рис.1.1 – Классификация типов данных на языке С++

Языки программирования C/C++ поддерживают типы данных, определяемые пользователями, одним из которых является структурированный тип данных.

Структура – это совокупность переменных, объединенных одним именем, предоставляющая общепринятый способ совместного хранения информации. Объявление структуры приводит к образованию шаблона, используемого для создания объектов структуры. Переменные, образующие структуру, называются полями (элементами, членами) структуры.

Таким образом, структура является объединением одного или более объектов (переменных встроенных типов, массивов, указателей, других структур и т.д.), которые для удобства работы с ними сгруппированы под одним именем.

Достоинства структур:

- облегчают написание и понимание программ;
- помогают сгруппировать данные, объединяемые каким-либо общим понятием;
- позволяют использовать группу связанных между собой переменных как единое целое.

Подобно массиву, структура представляет собой совокупность данных, но отличается от него тем, что к ее элементам (компонентам) необходимо обращаться по имени и ее элементы могут быть различного типа. Структуры целесообразно использовать там, где необходимо объединить данные, относящиеся к одному объекту. Объявление структуры приводит к образованию нового типа данных, используемого для создания переменных этого типа - объектов структуры.

Использование структур состоит из двух шагов:

- задание нового типа данных, определенного пользователем (объявление структуры);
- создание переменных типа объявленной структуры.

Общий синтаксис объявления шаблона (типа) структуры:

```
struct имя_типа
{
  тип1 имя_поля1;
  …
  тип1 имя_поляN;
};
```

Подробнее о структурах можно посмотреть, например, в [1,2,3, 4].

Пример. Описать структуру `DateBase` (база данных), которая имеет 4 поля: два символьных массива `fam` (фамилия) и `name` (имя), целое число `TelNumber`(номер телефона), символьный указатель `Adress`.

Результат будет следующим:

```
struct DateBase
{
    char fam[20];
    char name[15];
    long TelNumber;
    char* Adress;
};
```

Имена структур должны быть уникальными в пределах их области определения для того, чтобы компилятор мог различать различные типы структур. Задание типа осуществляется с помощью ключевого слова `struct`, за которым следует имя типа структуры и список элементов, заключенных в фигурные скобки.

Имена элементов в одном типе также должны быть уникальными. Однако в разных структурах можно использовать одинаковые имена элементов.

Задание только типа, не влечет резервирования памяти компилятором. Шаблон (тип данных) представляет компилятору необходимую

информацию об элементах структурной переменной для резервирования места в оперативной памяти и организации доступа к ней при определении структурной переменой и использовании отдельных элементов структурной переменной.

Среди членов данных структуры могут также присутствовать, кроме стандартных типов данных(`int, float, char` и т.д.), ранее определенные типы (другие структуры и т.д).

Другой пример: объявить структуру `person` (человек), имеющую среди прочих составное поле `date` (дата рождения), содержащее число, месяц и год рождения.

Вот полученный результат:

```
/* объявление шаблона структуры типа date */
struct date
{
    int day, month, year;
};
/* шаблон структуры person */
struct person
{
    char fam[30], im[20], otch[15];
    float weight;
    date birthday;
};
```

Структура `date` имеет три поля типа `int`. Тип структуры `person` в качестве элемента включает поле `birthday`, которое, в свою очередь, имеет ранее объявленный тип данных: `struct date`. Элемент `birthday` содержит в себе все компоненты `struct date`.

Определение структуры-переменной не отличается от объявления обычной переменной базового типа. Общий синтаксис определения переменной типа структуры:

```
struct имя_типа имя_переменной;
```

или

```
имя_типа имя_переменной;
```

Например:

```
struct person student;
struct date date1;
```

или

```
person student;
date date1;
```

Компилятор выделит под каждую переменную количество байтов памяти, необходимое для хранения всех ее элементов.

Разрешается совмещать описание шаблона и определение структурной переменной.

```
struct book {
    char title[20];
    char autor[30];
    double cast;
} book1, book2, * ptr_bk = &book1;
```

Доступ к полям осуществляется с помощью оператора «.» при непосредственной работе со структурой. Если же мы имеем указатель на объект структуры, то используется оператор «->». Эти операторы называются селекторами членов класса.

Прямой доступ к полям осуществляется следующим образом:

```
имя_переменной_структуры.поле_данных;
```

```
1) date1.day=10;
2) date1.year=1991;
3) strcpy(book1.title, “Война и мир”);
```

Используя прямое обращение к элементу, присваиваем значение выбранной переменной. Текст помещается в переменную, используя функцию копирования строковых переменных – `strcpy()`.

```
4) student.birthday.month=1;
5) student.birthday.year=2000;
```

Доступ по указателю осуществляется аналогично:

```
имя_указателя->имя_поля;
```

Примеры:

```
1) date1->day=10;
2) student->birthday.month=1;
```

Используя доступ к полям по указателю на структуру, можно использовать и такой синтаксис:

```
(*имя_указателя).имя_поля;
```

Примеры:

```
1) (*date1).day=10;
2) (*student).birthday.month=1;
```

Однако использование оператора «->» на практике предпочтительнее.

При определении структурных переменных можно инициализировать их поля. Эта возможность подобна инициализации массива и следует тем же правилам:

```
имя_типа имя_переменной = {значение1, значение2…};
```

Компилятор присваивает `значение1` первой переменной в структуре, `значение2` – второй переменной структуры и т.д., и тут необходимо следовать некоторым правилам:

– присваиваемые значения должны совпадать по типу с соответствующими полями структуры;

– можно объявлять меньшее количество присваиваемых значений, чем количество полей, компилятор присвоит нули остальными полями структуры;
– список инициализации последовательно присваивает значения полям структуры, вложенных структур и массивов.

```
struct date
{
    int day, month, year;
}
d = {1,3,2002};
```

Язык C++ позволяет оператору присваивания копировать значения одной структуры-переменной в другую переменную, при условии, что обе структуры-переменные относятся к одному и тому же типу.

```
book2 = book1;
```

При этом выполняется полное побайтовое копирование данных. Таким образом, единственный оператор может скопировать несколько членов данных, которые включают массивы и вложенные структуры.

Часто используются совокупности структур, объединяемые в массив структур. Для объявления массива структур следует сначала определить структуру, а затем объявить массив переменных данного типа. Например, для объявления 100-элементного массива структур типа date следует написать:

```
struct date date_array[100];
```

В результате получаем набор из 100 переменных, устроенных, как объявлено в типе структуры `date`.

Для доступа к отдельным структурам массива `date_array` следует сначала обратиться по индексу к нужному элементу массива. Напри-

мер, для вывода содержимого поля month третьей структуры, следует написать:

```
cout << date_array[2].month;
```

Как и в случае массивов переменных, индексация массивов структур начинается с нуля.

При описании массива структур возможна инициализация значений его элементов, как для массива, содержащего элементы базовых типов.

```
   struct date date_array[100] = { {1,3,2002}, {5,1,2002}, {1,1,2003} };
```

Если указаны значения не всех элементов, то остальным будут присвоены нулевые значения.

Часто объявления структур выносят в отдельный заголовочный файл.

Заголовочный файл (подключаемый файл, header file, file.h) – это файл, содержимое которого автоматически добавляется препроцессором в исходный текст в том месте, где располагается директива #include:

```
#include <file.h>.
```

В языках программирования C/C++ заголовочные файлы являются основным способом подключить к программе типы данных, структуры, прототипы функций, перечисляемые типы и макросы, используемые в другом модуле. По умолчанию используется расширение .h; иногда для заголовочных файлов языка C++ используют расширение .hpp.

Заголовочный файл в общем случае может содержать любые конструкции языка программирования, но на практике исполняемый код (за исключением inline-функций) в заголовочные файлы не помещают.

Например, идентификаторы, которые должны быть объявлены более чем в одном файле, удобно описать в заголовочном файле, а затем его подключать по мере надобности. Также в заголовочных файлах объявляют функции стандартной библиотеки.

Заголовочные файлы состоят из двух частей:

- директивы препроцессора – в частности, header guards, которые предотвращают вызов заголовочного файла больше одного раза из одного и того же файла.
- содержимое заголовочного файла – набор объявлений.

Пример 1. Задан массив структур «Студент» (фамилия; имя; баллы по трем предметам ЕГЭ). Задание данных осуществляется вручную до указания пользователем прекращения ввода. Определить суммарный балл каждого студента. Вывести на экран список студентов с указанием суммы баллов.

Отдельно будут разработаны два файла в одном проекте: заголовочный (`student.h`) и собственно программа (`exampl01t.cpp`).

Заголовочный файл создается для объявления шаблона структуры и далее это объявление включается в программу директивой `#include "student.h"`.

Программа на языке C++.

Файл student.h:

```
#ifndef STUDENT_H
#define STUDENT_H

struct student // Описание структуры student
{
    char name[20];
    char fam[30];
    int math;
    int phys;
    int rus;
```

```
    double sum;
};

#endif
```

Файл exampl01t.cpp:

```
#include <iostream>
#include <iomanip>
#include "student.h"

using namespace std;


int main()
{
    ios_base::left;
    setlocale(0, "");

    // Объявление массива
    // структур students типа student
    student students[10] = {}; //обнуляем поля
    // Объявление
    // вспомогательной переменной buffer типа student
    student buffer;
    int i, j, records = 0;
    // Объявление символьной переменной-флага
    char flag = 'y';
    cout << setw(40) << "Ввод данных о студентах" << endl;
    // Ввод данных
    do {
        cout << "Студент " << records + 1 << endl;
        cout << "Введите фамилию: ";
        cin >> students[records].fam;
        cout << "Введите имя: ";
        cin >> students[records].name;
        cout << "Введите балл по математике: ";
        cin >> students[records].math;
        cout << "Введите балл по физике: ";
        cin >> students[records].phys;
        cout << "Введите балл по русскому языку: ";
        cin >> students[records].rus;
        records++;
```

```
        cout << "Продолжить ввод? (y/n)";
        cin >> flag;
    } while (flag == 'y');
    // подсчет суммы баллов
    for (i = 0; i < records; i++)
    {
        students[i].sum += students[i].math;
        students[i].sum += students[i].phys;
        students[i].sum += students[i].rus;
    }

    cout << endl << "Список студентов" << endl;
    cout << setw(16) << "Фамилия" << setw(10) <<
        "Имя" << setw(10) << "Сумма баллов" << endl;
    for (i = 0; i < records; i++) {
        cout << setw(16) << students[i].fam << setw(10) <<
            students[i].name << setw(10) <<
            students[i].sum  <<  endl;
    }

    return 0;
}
```

1.2 Передача структур в качестве параметров

Переменные структурного типа и элементы структуры можно передавать в функции в качестве параметров.

Передача параметров может выполняться:

- по ссылке или указателю;
- по значению.

При передаче параметра по указателю передается только указатель на структуру, при передаче по значению в стек копируется все содержание структуры. Например:

```
struct structA {
    int i;
    char c;
} sA, *psA=&sA;
```

```
void F1(structA sA);   // Передача параметров по значению
void F2(structA *psA); // Передача параметров по указателю
void F3(structA &sA);  // Передача параметров по ссылке
…
void F2(structA *psA) {
    psA->i =10; // Доступ к элементу структуры
}
```

При большой вложенности вызовов и использовании большого числа структур или их значительных размерах вызов по значению может привести к переполнению стека.

Функция может возвращать значение структурного типа или типа указателя на структуру. Пример:

```
struct structA { int i; char c;};

// Функция возвращает значение структурного типа
structA Function3(void);

// Функция возвращает указатель на структуру
structA *Function4(void);
```

1.3 Формат времени с использованием struct tm

Структура `tm` очень важна при работе с датой и временем в программах на C++. Большинство функций, связанных со временем, используют структуру `tm`. Поля этой структуры приведены в таблице 1.1

Таблица 1.1

Список полей структуры `tm`

Член	Тип	Имея в виду	Диапазон
tm_sec	int	секунды после минуты	0-60*
tm_min	int	минуты после часа	0-59
tm_hour	int	часов с полуночи	0-23
tm_mday	int	день месяца	1-31
tm_mon	int	месяцев с января	0-11
tm_year	int	лет с 1900	
tm_wday	int	дней с воскресенья	0-6
tm_yday	int	дней с 1 января	0-365

tm_isdst	int	Флаг перехода на летнее время	

Ниже приведен пример, который использует различные функции, связанные с датой и временем, и структуру tm.

```
#include <iostream>
#include <ctime>
using namespace std;

int main() {
   // current date/time based on current system
   time_t now = time(0);
   cout << "Number of sec since January 1,1900:" <<
   now << endl;

   tm ltm;
   localtime_s(&ltm, &now);

   // print various components of tm structure.
   cout << "Year" << 1900 + ltm.tm_year<<endl;
   cout << "Month: "<< 1 + ltm.tm_mon<< endl;
   cout << "Day: "<<  ltm.tm_mday << endl;
   cout << "Time: "<< 1 + ltm.tm_hour << ":";
   cout << 1 + ltm.tm_min << ":";
   cout << 1 + ltm.tm_sec << endl;
}
```

Обратите внимание, что функция localtime_s() используется только в компиляторе Microsoft (который идет с Visual Studio). В других компиляторах нужно использовать функцию localtime().

Когда приведенный выше код компилируется и выполняется, он производит следующий результат:

```
Number of sec since January 1, 1900:1294548238
Year: 2021
Month: 1
Day: 8
Time: 22: 44:59
```

Структуры – это один из видов структурированных типов данных, содержащий данные, представленные в виде полей. Кроме структур, другими случаями структурированных типов данных являются объединения.

1.4 Объединения

Объединение (англ. Union) представляет собой специальный тип данных, позволяющий нескольким переменным различных типов, занимать один участок памяти. Объединения применяют для экономии памяти в тех случаях, когда известно, что больше одного поля одновременно не требуется.

Использование объединений началось во времена, когда стояла проблема экономии оперативной памяти. Объединение переменных призвано помогать программисту сделать оптимальную и экономную программу.

Чтобы понять, в чем смысл объединения, нужно вспомнить как хранятся переменные. Разные переменные различных типов (например, `int, double и short`) занимают в памяти разное количество байт. `double` в любом случае занимает максимальное количество байт в памяти, при этом в память для переменной этого типа вполне можно записать значения `int` или `short`.

При этом не все зарезервированные байты могут быть востребованы. Если поместить, к примеру число 325 в `int`, будут заняты два байта (зависит от разрядности процессора), а остальные байты заполнятся нулями.

Таким образом, компилятор резервирует место для типа данных с максимальным запросом объема памяти, а какие значения в них хранить решается отдельно в каждой реализации.

Объединение – это такой формат данных, который внешне подобен структуре: в него можно включить несколько полей разных типов. И оно способно хранить в пределах одной зарезервированной области памяти различные данные. Но в каждый определенный момент времени в объединении хранится только один из этих типов данных и возможно использовать лишь значение этого элемента (компонента).

Синтаксически определяют объединение так:

```
union имя_объединения {
 //Открытые члены по умолчанию.
private:
 //Закрытые члены.
} список_объектов; //не обязательный
```

Доступ к элементам объединения осуществляется так же, как и к элементам структур.

К данным, которые хранят элементы структуры (например `short, int, long`) мы можем обращаться в любой момент (хоть к одному, хоть и ко всем сразу). А в объединении могут храниться данные либо `short`, либо `int`, либо `long`. Например:

```
#include <iostream>
using namespace std;

union MyUnion
{
    short int name1;
    int name2;
    long int name3;
};

int main()
{
   MyUnion myUnion;
   myUnion.name1 = 22;
   cout << myUnion.name1 << endl;
   myUnion.name3 = 222222222;
   cout << myUnion.name3 << endl;
```

```
    cout << myUnion.name1 << endl;
    cout << "sizeof(myUnion) = " << sizeof(myUnion);
    return 0;
}
```

Результат работы программы показан на рисунке 1.2.

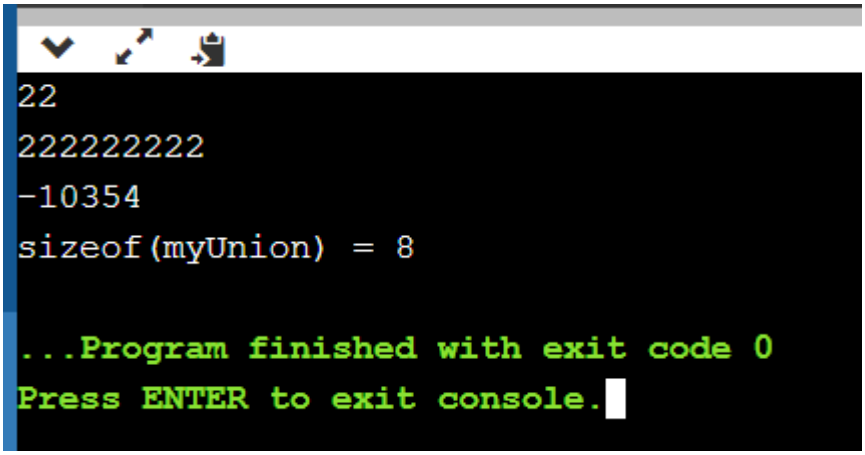


Рис 1.2. Результат работы программы, с помощью on-line компилятора

После того, как записано значение в элемент `name3` типа `long int`, уже невозможно нормально обращаться к элементу `name1`. Все потому, что в их общую память уже записано значение `long int`, а переменная типа `short int` неспособна работать с данными такого объема.

Схематически можно это представлено в таблице 1.2

Таблица 1.2.

Распределение памяти в объединении

Объединение myUnion занимает 8 байта							
1 байт	2 байт	3 байт	4 байт	5 байт	6 байт	7 байт	8 байт
Два байта для short		Не использует для short	Не используется для short	Не используется для short	Не используется для short	Не используется для short	Не используется для short
Четыре байта для int				Не используется для int	Не используется для int	Не используется для int	Не используется для int

Восемь байт для long

Поэтому, чтобы опять перейти к работе с данными типа `short int` необходимо снова присвоить элементу `name1` новое значение. Таким образом, в одной и той же памяти размещаются разные переменные. К какому участку памяти происходит обращение, такое значение запрашивает из памяти.

Элементы объединения располагаются в памяти, начиная с одного участка (как бы накладываются друг на друга), в отличие от элементов структуры (они располагаются в памяти последовательно один за другим).

При описании набора переменных, не объединенных в `union`, потребовалось бы 2 + 4 + 8 байта = 14 байт. Фактически объединение занимает 8 байт.

1.5 Перечисления.

Перечисление (перечисляемый тип, англ. enumeration, enumerated type) – тип данных с фиксированным множеством значений, определяемых пользователем. Множество значений перечисления представляет собой список идентификаторов-констант, которым может быть присвоено конкретное значение.

Описание

```
enum [имя_типа] {список_констант};
```

Имя перечисляемого типа задается в том случае, если в дальнейшем требуется определять переменные этого типа. Компилятор обеспечивает, чтобы эти переменные принимали значения только из списка констант. Константы должны быть целочисленными и могут инициализироваться обычным образом. При отсутствии инициализатора

первая константа обнуляется, а каждой следующей присваивается на единицу большее значение, чем предыдущей.

Перечисления удобно использовать там, где определенная сущность предметной области может принимать некоторое (не очень большое) количество фиксированных значений. Например, описать перечисление, задающее времена года:

```
enum seasons
{
   spring,
   summer,
   autumn,
   winter
};
```

В этом примере каждой константе присваивается по умолчанию числовое значение, начиная с нуля: `spring=0 и т.д.`, а `winter=3`.

Мы можем определить переменную типа `seasons` и присвоить этой переменной значение одной из констант, объявленных в перечислении. Но фактически это будет числовое значение.

```
seasons currentSeason = autumn;
cout << "Season: " << currentSeason << endl;
```

Результатом вывода фрагмента этой программы является следующий вывод:

```
Season: 2
```

Так как перечисления - это отдельный тип данных, то присвоить переменной числовое значение напрямую нельзя:

```
seasons currentSeason = 2;  // ошибка
```

Значения по умолчанию для констант можно задать явным образом. Например, установить начальное значение для каждой константы перечисления:

```
enum seasons
{
   spring = 1,
   summer, //2
   autumn, //3
   winter  //4
};
```

В этом случае значения второй и последующих констант будет увеличиваться на единицу.

При необходимости можно задать значение для каждой константы:

```
enum seasons
{
   spring = 1,
   summer = 2,
   autumn = 4,
   winter = 8
};
```

Пример. Описать перечисление, содержащее количество дней в каждом месяце года.

```
enum DAYS_IN_MONTHS
{
   January = 31,
   February = 28,
   March = 31,
   April = 30,
   May = 31,
   June = 30,
   July = 31,
   August = 31,
   September = 30,
   October = 31,
   November = 30,
   December = 31
};
```

В перечислениях допускается использовать константные значения следующих типов:

- целочисленных типов (`int, long int` и т.д.);
- символьного типа `char`;
- логического типа `bool`.

Использовать константные вещественные значения типов `float`, `double` нельзя. При попытке объявить константное значение типа `double` или `float` компилятор выдаст сообщение об ошибке

```
enum ENUM_DBL { a = 1.234 };
```

При необходимости эти типы могут сочетаться в одном перечислении, например

```
enum VALUES {
    val1 = 2,
    val2 = 'c',
    val3 = true,
    val4 = -9
};
```

1.5. Вопросы для самопроверки

1. Дайте определение структуры.
2. Объявление и определение структур.
3. Доступ к членам структур
4. Инициализация структур
5. Размер переменной типа структура
6. Для чего используются вложенные структуры?

1.6. Задания для самопроверки

Выполните задания из лабораторной работы №1 (приложение 1).

2. РАБОТА С ТЕКСТОВЫМИ ФАЙЛАМИ

2.1. Общие сведения о файлах

Большинство компьютерных программ работают с файлами, и поэтому возникает необходимость производить обработку: создавать, удалять, записывать читать, открывать файлы.

Файл (англ. file) – именованная область данных на носителе информации (материальный объект, используемый человеком для хранения информации).

Под файлом понимается некоторая последовательность байтов на внешнем носителе, которая имеет своё, уникальное имя, например «File.txt».

Имя файла – массив символов, однозначно определяющий файл в некотором пространстве имён файловой системы, обычно называемом папкой, каталогом или директорией. Имена файлов строятся по тем правилам, которые приняты в той или иной файловой и операционной системах.

Имя файла является частью полного имени файла, также называемого полным или абсолютным путём к файлу. Полное имя может включать следующие компоненты:

- протокол или способ доступа (http, ftp и т.д.);
- имя или адрес компьютера, узла сети;
- устройство хранения, диск (C:, /, SYSLIB и т. п.);
- путь к каталогу (/usr/sbin, \TEMP, [USR.LIB.SRC] и т. п.);
- собственно имя файла: название и расширение, разделенные точкой(.txt, .exe, .COM и т. п.);
- версия или номер ревизии (perl6, ping6 и т. п.).

Имя файла необходимо знать для того, чтобы была возможность обращения к файлу. В одной папке не может быть двух файлов

с одинаковыми именами. При этом «file.txt» и «file.doc» – разные файлы, хоть и имеют одинаковые названия.

Для каждого файла важным является его полное имя – это полный адрес к местоположению файла с указанием имени файла, например в C++:

D:\\docs\\file.txt.

По аналогии относительное имя файла – это его имя относительно текущей рабочей папки (каталога), например, из которой запущена программа, работающая с файлом. При использовании относительных имен файлов, можно использовать специальные обозначения «.» и «..»:

- . – это ссылка, которая содержит в себе адрес на текущую папку;
- .. – это ссылка, которая содержит адрес на предыдущую папку.

Обычно говорят об имени файла, подразумевая только имя и расширение.

При работе с файлами в программировании часто их классифицируют на два основных типа: текстовые и двоичные.

Текстовый файл – файл, содержащий текстовые данные: набор символов в соответствующей кодировке и символы перевода строки. Посмотреть содержимое такого файла можно с помощью любого текстового редактора, например «Блокнот» (но не MS Word!). При записи информации в текстовый файл все данные хранятся в виде одного сплошного массива символов, включая символы перевода строки.

Двоичный (бинарный) файл – файл, который содержит данные, не рассчитанные на интерпретацию в качестве текстовой информации (например, файлы, хранящие текст в закодированном или сжатом виде, или хранящие не текст, а звук, изображение или иные данные). В двоичных файлах информация считывается и записывается в виде

блоков определенного размера, в которых могут храниться данные любого вида и структуры.

Рассмотрим работу с текстовыми файлами более подробно.

Как было сказано выше, текстовые файлы организуются по строкам, каждая из которых заканчивается символом перевода строки (конца строки).

Признак конца файла EOF (аббревиатура англ. end of file – конец файла) в компьютерной терминологии является индикатором операционной системы, означающим, что данные в источнике закончились. Источниками данных обычно являются файлы и потоки.

В стандартной библиотеке языка C функции ввода-вывода, в частности доступа к файлам, могут возвращать значение, равное символьной константе (точнее макроопределению) EOF для индикации, что достигнут конец файла. Реальное значение EOF является отрицательным числом, зависящим от системы (в основном −1), что гарантирует несовпадение с кодом символа.

Механизм ввода-вывода, разработанный для языка программмирования C, не соответствует общепринятому сегодня стилю объектно-ориентированного программирования, кроме того, он активно использует операции с указателями, считающиеся потенциально небезопасными в современных защищённых средах выполнения кода. Альтернативой при разработке прикладных приложений является механизм стандартных классов потоков ввода-вывода, предоставляемый стандартом языка C++

Для работы с файлами с использованием потоков C++, необходимо подключить заголовочный файл <fstream>. В <fstream> определены несколько классов и подключены заголовочные файлы <ifstream> – файловый ввод и <ofstream> – файловый вывод.

Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие – это то, что ввод/вывод выполнятся не на

экран, а в файл. Если ввод/вывод на стандартные устройства выполняется с помощью объектов cin и cout, то для организации файлового ввода/вывода достаточно создать собственные объекты, которые можно использовать аналогично операторам cin и cout.

2.2 Класс ofstream: запись в файл

Пример. Создать текстовый файл «`text.txt`» и записать в него строку «`Hello world!`». Для этого необходимо проделать следующие шаги:

- создать объект класса ofstream;
- связать объект класса с текстовым файлом, в который будет производиться запись;
- записать информацию в текстовый файл;
- закрыть файл.

Приведем фрагмент программы для решения этой задачи с пояснениями.

1. Создать объект класса `ofstream`. Объект необходим, чтобы можно было выполнять запись в файл. Назовём объект `fout` по аналогии с выводом информации на экран cout.

```
ofstream fout;
```

Объект создан, но не связан с файлом, в который нужно записать информацию.

2. Связать объект класса с текстовым файлом, в который будет производиться запись. Для этого для созданного объекта используется метод `open()` класса `ofstream`.

```
fout.open("text.txt"); // связываем объект с файлом
```

Через операцию точка получаем доступ к методу класса `open()`, в круглых скобочках которого указываем имя файла. Указанный файл будет создан в текущей папке с программой. Если файл с таким име-

нем существует, то будет создан новый файл, при этом содержимое предыдущего тестового файла будет удалено.

В общем случае функция открытия потока имеет вид:

```
f.open(«file», mode);
```

где `f` – объект класса `ofstream`,

`file` – полное имя файла, `mode` – режим работы с открываемым файлом.

3. Записать информацию в файл.

Используя операцию передачи в поток совместно с объектом `fout` строка «Hello world!» будет записана в файл.

```
fout << " Hello world!"; // запись строки в файл
```

4. Закрыть файл. Так как обработка файла окончена и нет больше необходимости изменять содержимое файла, его нужно закрыть, то есть отделить объект от файла.

```
fout.close(); // закрываем файл
```

В результате в текущей папке создан файл "`text.txt`" со строкой «Hello world!».

Шаги 1 и 2 можно объединить, то есть в одной строке создать объект и связать его с файлом.

```
ofstream fout("text.txt"); // создаём объект класса
            //ofstream и связываем его с файлом "text.txt".
```

Объединим весь код и получим следующую программу.

```
ofstream fout("text.txt "); // создаём объект класса
fout << "Hello world!"; // запись строки в файл
fout.close(); // закрываем файл
```

Осталось проверить правильность работы программы. Для этого файл нужно открыть в текстовом редакторе. В нем должна быть записана строка «`Hello world!`».

Если необходимо указать полный путь, необходимо прописать его при объявлении объекта класса ofstream, например,

```
ofstream fout("C:\\Temp\\text.txt ");
```

Нужно обратить внимание, что числовые значения записываются в файл в виде текстовых строк, а не двоичных значений.

Для работы с объектами класса ofstream используются методы, описанные в таблице 2.1.

Таблица 2.1

Методы класса ofstream

Метод	Описание
open	Открывает файл для записи
put	Записывает одиночный символ в файл
write	Записывает заданное число байт из памяти в файл
seekp	Перемещает указатель позиционирования в указанное положение
tellp	Возвращает текущее значение указателя позиционирования файла
close	Закрывает файл

2.3 Класс ifstream: чтение файлов

Для того чтобы прочитать файл понадобится выполнить те же шаги, что и при записи в файл с небольшими изменениями:

- создать объект класса ifstream
- связать его с файлом, из которого будет производиться считывание;
- прочитать файл;
- закрыть файл.

Пример. Считать текст из текстового файла "text.txt" и вывести его на экран.

```
char buff[80]; // буфер промежуточного
               //хранения считываемого из файла текста
ifstream fin("text.txt"); // открыли файл для чтения
fin >> buff; // считали первое слово из файла
cout << buff << endl; // напечатали это слово
fin.getline(buff, 80); // считали строку из файла
fin.close(); // закрываем файл
cout << buff << endl; // напечатали эту строку
```

В программе показаны два способа чтения из текстового файла. Первый способ использует операцию передачи в поток, второй – метод getline(). В первом случае считывается только первое слово, а во втором случае считывается строка, длиной до 80 символов. В файле осталось меньше 80 символов, поэтому происходит считывание символов до конца строки включительно. При этом считывание во второй раз с использованием метода getline() продолжилось со второго слова, так как первое слово было прочитано с использованием операции передачи информации в поток и удалено из потока.

Для работы с объектами класса ifstream используются методы, описанные в таблице 2.2.

Таблица 2.2

Методы класса ifstream

Метод	Описание
open	Открывает файл для чтения
get	Читает один или более символов из файла
getline	Читает символьную строку из текстового файла или данные из бинарного файла до определенного ограничителя
read	Считывает заданное число байт из файла в память
eof	Возвращает ненулевое значение (true), когда указатель потока достигает конца файла
peek	Выдает очередной символ потока, но не выбирает его

Метод	Описание
seekg	Перемещает указатель позиционирования файла в заданное положение
tellg	Возвращает текущее значение указателя позиционирования файла
close	Закрывает файл

Нужно иметь ввиду, что программа может быть выполнена некорректно, даже в том случае, если код написан правильно. Например, в программу будет передано имя несуществующего файла или в имени файла допущена ошибка. В этом случае запрашиваемый файл не будет найден и произвести чтение будет невозможно. Поэтому компилятор проигнорирует строки, в которых выполняется работа с файлом. В результате работа программы завершится корректно, но результат на экране показан не будет.

В C++ предусмотрена возможность проверки объекта потока как логической переменной, связанной с открытием файла:

true – если файл был успешно открыт,

false – если файл открыт не был.

Доработаем программу с открытием файла, таким образом, что если файл не открыт, выводилось соответствующее сообщение.

```
setlocale(LC_ALL, "rus"); // корректное отображение
                          //Кириллицы
ofstream fout("data_types.txt "); // создаём объект класса
                  // ofstream для записи и связываем его
                  // с файлом output.txt

char buff[50]; // буфер промежуточного хранения
               //считываемого из файла текста
ifstream fin("text.txt"); // здесь намеренно введено
                          //имя несуществующего файла

if (!fin) // здесь проверяем открытие файла
          //если файл не открыт:
```

```
    cout << "Ошибка открытия файла.\n"; // сообщить об
                                        // ошибке открытия
else
{
    fin >> buff; // считали первое слово из файла
    cout << buff << endl; // напечатали это слово

    fin.getline(buff, 50); // считали строку из файла
    fin.close(); // закрываем файл
    cout << buff << endl; // напечатали эту строку
}
```

Результат работы программы показан на рисунке 2.1.

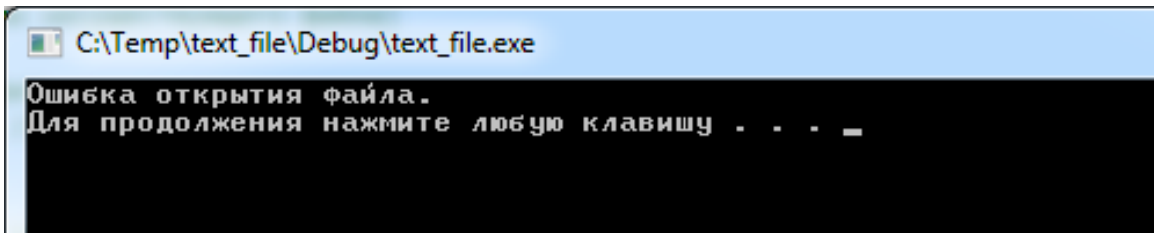


Рис.2.1 –Результат выполнения программы при отсутствии файла

Как видно из рисунка 2.1, программа сообщила об ошибке и невозможности открыть файл. Поэтому, если программа работает с файлами, рекомендуется использовать проверку открытия файла, даже если точно известно, что файл существует.

Особенно важно контролировать ситуацию и осуществлять эту проверку в случаях, когда производится работа с файлом, имя которого вводит пользователь. Сделать это можно следующим образом.

```
char buff[50]; // буфер промежуточного хранения
               // считываемого из файла текста
char file_name[20];
cout << "Введите имя файла: ";
cin >> file_name;
ifstream fin(file_name); // (введено имя несущ. файла)

if (!fin) // если файл не открыт
```

```
        cout << "Ошибка открытия файла.\n";
    else
    {
         fin >> buff; // считали первое слово из файла
        cout << buff << endl; // напечатали это слово
        fin.getline(buff, 50); // считали строку из файла
        fin.close(); // закрываем файл
        cout << buff << endl; // напечатали эту строку
    }
```

2.4 Режимы открытия файлов

Режимы открытия файлов устанавливают характер использования файлов. Для установки режима в классе `ios_base` предусмотрены константы, которые определяют режим открытия файлов:

`ios::in` – открыть файл в режиме чтения данных; режим является режимом по умолчанию для потоков `ifstream`;

`ios::out` – открыть файл в режиме записи данных (при этом информация о существующем файле уничтожается); режим является режимом по умолчанию для потоков `ofstream`;

`ios::app` – открыть файл в режиме записи данных в конец файла;

`ios::ate` – при открытии переместить указатель в конец файла;

`ios::trunc` – очистить файл;

`ios::nocreate` – не выполнять операцию открытия файла, если он не существует;

`ios::noreplace` – не открывать существующий файл.

`ios_base::binary` – открытие файла в двоичном режиме

Параметр `mode` может отсутствовать,

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове функции `open()`.

Например,

```
ofstream fout("text.txt", ios_base::app); // открытие файла для добавления информации к концу файла.
```

или

```
fout.open("text.txt", ios_base::app); // открытие файла для добавления информации к концу файла.
```

Режимы открытия файлов можно комбинировать, оператор логического ИЛИ (|) позволяет составить режим с любым сочетанием флагов.

Например,

1) `file.open ("test.txt", ios::out | ios::app)`

– открыть файл для записи и при этом случайно не удалить существующий файл с тем же именем;

2) `file.open (("test.txt", ios_base::out | ios_base::trunc)`

– открытие файла для записи с предварительной очисткой.

Объекты класса `ofstream`, при связке с файлами по умолчанию содержат режимы открытия файлов `ios_base::out | ios_base::trunc`. То есть файл будет создан, если не существует. Если же файл существует, то его содержимое будет удалено, а сам файл будет готов к записи. Объекты класса `ifstream`, связываясь с файлом, имеют по умолчанию режим открытия файла `ios_base::in` – файл открыт только для чтения.

Обратите внимание на то, что флаги `ate` и `app` по описанию очень похожи, они оба перемещают указатель в конец файла, но флаг app позволяет производить запись, только в конец файла, а флаг ate просто переставляет флаг в конец файла и не ограничивает места записи.

После работы с файлом необходимо его обязательно закрыть. Это выполняет метод `close()`. Ввиду механического устройства жесткого диска, данные в файл попадают не сразу. Сначала данные записыва-

ются в буфер (область оперативной памяти), и когда он заполнится, то данные из буфера будут записаны в файл. Такая схема разработана для ускорения работы с более медленными устройствами хранения данных. Когда производится запись в файл любым методом, то на самом деле в первую очередь данные попадают в буфер. И только тогда, когда буфер будет заполнен до конца, он будет сразу весь записан на жесткий диск. По этой причине, если файл не закрыт методом `close()`, то последние данные из буфера не будут записаны в файл. Отсутствие этой операции может привести к потере данных в файле, который был открыт для записи или дозаписи.

Файлы, открытые только для чтения, также необходимо закрывать. Если не закрывать файлы (которые открыты даже для чтения), то это может привести к ограничению доступа к файлу для других программ. Какие именно ограничения будут использоваться, это зависит от операционной системы. Например, в операционной системе Windows, если файл открыт для чтения и не закрыт, то из другой программы его нельзя удалить.

Приведем несколько примеров.

Пример 1. Создать текстовый файл `C:\\Temp\\numbers.txt` и записать в него *n* вещественных чисел. Значение *n* вводится пользователем.

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
int main()
{
   setlocale (LC_ALL, "RUS");
   int i, n;
   double a;
 //описывает поток для записи данных в файл
   ofstream fout;
 //открываем файл в режиме записи,
 //режим ios::out устанавливается по умолчанию
```

```
  fout.open("C:\\Temp\\numbers.txt", ios::out);
 //вводим количество вещественных чисел
  cout<<"n="; cin>>n;
 //цикл для ввода вещественных чисел
 //и записи их в файл
  for (i=0; i<n; i++)
  {
     cout<<"a=";
     //ввод числа
     cin>>a;
     fout<<a<<"\t";
     cout << "Число "<< a <<" записано в файл" << endl;
  }
 //закрытие потока
  cout << "Работа успешно завершена" << endl;
  fout.close();
  system("pause");
  return 0;
}
```

Пример 2. В текстовом файле C:\\Temp\\numbers.txt хранятся вещественные числа, вывести их на экран и вычислить их количество.

```
int n=0;
float a;
ifstream fin;
//открываем файл в режиме чтения
fin.open("C:\\Temp\\numbers.txt", ios::in);
//если открытие файла прошло корректно, то
if (fin)
{
     // цикл для чтения значений из файла;
     // выполнение цикла прервется,
     // когда достигнем конца файла, в этом случае
     // fin.eof() вернет истину.
     while (!fin.eof())
     {
          //чтение очередного значения из потока fin
          // в переменную a
          fin >>a;
          //вывод значения переменной a на экран
          cout<<a<<"\t";
```

```
            //увеличение количества считанных чисел
            n++;
        }
        //закрытие потока
        fin.close();
        //вовод на экран количества считанных чисел
        cout<<endl<<"Количество чисел равно "<<n<<endl;
    }
    //если открытие файла прошло некорректно, то вывод
    //сообщения об отсутствии такого файла
    else cout<<"Ошибка открытия файла"<<endl;
```

Возможно, исходный текстовый файл не содержит информации, то есть является пустым. Результат работы программы в этом случае показан на рисунке 2.3.

Рис. 2.3 – Результат работы программы,
если исходный файл является пустым.

Для проверки файла на пустоту, т.е. отсутствие какой-либо информации, можно использовать метод peek(), который используется с потоками ввода и возвращает следующий символ потока или EOF, если достигнут конец файла. При этом метод peek() не удаляет символ из потока.

```
if (fin.peek() == EOF) cout << "File is empty\n";
// если первый символ конец файла
```

Тогда программа будет иметь вид:

```
int n=0;
 float a;
 ifstream fin;
 //открываем файл в режиме чтения
 fin.open("C:\\Temp\\numbers.txt", ios::in);
 //если открытие файла прошло некорректно, то вывод
 //сообщения об отсутствии такого файла
 if (!fin)
```

```
{
 cout<<"Ошибка открытия файла"<<endl;
 return 0;
}
// если первый символ конец файла, то файл пустой
 if (fin.peek() == EOF)
{
 cout << "Файл пустой\n";
 return 0;
}
 // цикл для чтения значений из файла;
  // выполнение цикла прервется,
 // когда достигнем конца файла, в этом случае
 // fin.eof() вернет истину.
 while (!fin.eof())
 {
 //чтение очередного значения из потока fin
  // в переменную a
 fin >>a;
 //вывод значения переменной a на экран
 cout<<a<<"\t";
 //увеличение количества считанных чисел
 n++;
 }
 //закрытие потока
 fin.close();
 //вовод на экран количества считанных чисел
 cout<<endl<<"Количество чисел равно "<<n<<endl;
```

2.5. Вопросы для самопроверки

1. Дайте определение файла

2. Что такое абсолютный (полный) путь к файлу. Приведите пример записи абсолютного имени файла на языке C++.

3. В чем отличие текстовых и двоичных файлов?

4. Для чего предназначен заголовочный файл <fstream>?

5. Приведите алгоритм записи информации в текстовый файл.

2.6 Задания для самопроверки

Выполните задания из лабораторной работы №2 (приложение 1).

3. РАБОТА С ДВОИЧНЫМИ (БИНАРНЫМИ) ФАЙЛАМИ

3.1. Основы работы с двоичными файлами

Двоичный (бинарный) файл – файл, который содержит данные, не рассчитанные на интерпретацию в качестве текстовых (например, файлы, хранящие текст в закодированном или сжатом виде, или хранящие не текст, а звук, изображение или иные данные).

В принципе, любой файл (даже содержащий текст) можно рассматривать как двоичный, однако, как правило, работа с текстовыми файлами, описанная в главе 2, может существенно упростить разработку программы, обрабатывающей символьные данные.

Как известно, информация хранится в компьютере в виде наборов 0 и 1, т. е. в бинарном виде. Поэтому бинарные (двоичные) файлы отличаются от текстовых только методами работы с ними. Например, если записать в текстовый файл цифру «5», то она записывается как символ, и для ее хранения нужен один байт. Соответственно и размер файла будет равен одному байту. Текстовый файл, содержащий запись: «123456», будет иметь размер шесть байт. Если же записать целое число 123456 в бинарный файл, то он будет иметь размер четыре байта, так как именно столько необходимо для хранения данных типа `int`. Таким образом, бинарные файлы более компактны и в некоторых случаях более удобны для обработки.

Открытие бинарного файла на языке программирования C++ производится в режиме `ios::binary`

Бинарные файлы обрабатываются аналогично текстовым. Отличие состоит в том, что если бинарные данные имеют определенную

логическую структуру, то они должны считываться из файла в переменные тех же типов (структур, массивов, базовых типов и т.д.).

Для записи и чтения данных, при работе с бинарными данными, используются методы `write()` и `read()` потоков ввода/вывода.

Наиболее распространенный способ использования этих методов основывается на следующих прототипах:

```
istream &read(unsigned char *buf, int num);
ostream &write(const unsigned char *buf, int num);
```

Метод `read()` читает `num` байт из ассоциированного потока ввода и записывает их в буфер по адресу `buf`. Метод `write()` записывает `num` байт в ассоциированный поток вывода из буфера `buf`.

Первый параметр методов `write()` и `read()` (адрес блока записи/чтения) должен иметь тип символьного указателя, поэтому необходимо произвести явное преобразование адреса, откуда будет выполняться чтение/запись к (`char *`). Второй параметр указывает количество байт чтения/записи.

Пример 1. Создать два файла: текстовый и бинарный. Записать в них целое и вещественное число. Сравнить размеры файлов.

Решение. Создадим объект класса `fstream` и произведем его открытие в текстовом режиме для записи.

```
fstream text_file("C:\\Temp\\text.txt", ios::out);
```

Файл создан в папке `C:\\Temp`, размеры составляют 0 байт. Запишем в него информацию.

```
int Int=987654;
double Double=3.14159265358;
```

Так как точность `cout` по умолчанию равна 6, т.е. 6 значащих цифр. Чтобы изменить точность, нужно использовать метод `precision`:

```
text_file.precision(12);
```

```
text_file << int << endl << double <<endl;
text_file.close();
```

Теперь запишем данные в бинарный файл. Создадим другой объект класса `fstream` произведем его открытие в бинарном режиме для записи.

```
fstream bin_file("C:\\Temp\\bin.dat",
                              ios::out| ios::binary);
```

Запись в бинарный файл осуществляется с помощью метода `write()`,

```
bin_file.write((char*)&int,4);
```

В случае, если размер записываемой переменной неизвестен, используется функция sizeof.

```
bin_file.write((char*)&double,sizeof(double));
bin_file.close();
```

Обратная задача. Открыть и прочитать информацию из созданного бинарного файла. Необходимо учесть, что считывать данные из бинарного файла возможно лишь в той же последовательности, в которой они были записаны.

```
fstream bin_f("C:\\Temp\\bin.dat",
                               ios::in| ios::binary);
bin_f.read((char*)&Int,sizeof(int));
bin_f.read((char*)&double,sizeof(double));
cout<<int<<endl<<double<<endl;
bin_f.close();
```

Пример 2. Написать программу, которая записывает и потом читает массив целых чисел.

```
int n[5] = {1, 2, 3, 4, 5};
int i;
ofstream fout("C:\\Temp\\bin.dat", ios::out|ios::binary);
fout.write((char *)&n, sizeof(n));
fout.close();
```

```
for (i=0; i<5; i++) // очистка массива
     n[i] = 0;
ifstream fin("C:\\Temp\\bin.dat", ios::in | ios::binary);
fin.read((char *) &n, sizeof n);
for (i=0; i<5; i++){ // вывести значения, прочитанные
                     // из файла
     cout << n[i] << " ";
}
fin.close();
```

Следует обратить внимание, что приведение типов в вызовах `read()` и `write()` необходимо для работы с буфером, который не определен как массив символов.

Если конец файла достигается до того, как будет прочитано заданное число символов, функция `read()` просто прекращает работу и буфер содержит столько символов, сколько было прочитано.

3.2 Методы произвольного доступа

Метод `seekg()` производит установку текущей позиции в нужную, указываемую числом. В этот метод так же передается способ позиционирования:

`ios_base::end` – Отсчитать новую позицию с конца файла;

`ios_base::beg` – Отсчитать новую позицию с начала файла (абсолютное позиционирование);

`ios_base::cur` – Перейти на n байт начиная от текущей позиции в файле (по умолчанию);

```
file.seekg(0,ios_base::end); //Стать в конец файла
file.seekg(10,ios_base::end); //Стать на 10 байтов с конца
file.seekg(30,ios_base::beg); //Стать на 31-й байт
file.seekg(3,ios_base::cur); //перепрыгнуть через 3 байта
file.seekg(3); //перепрыгнуть через 3 байта - аналогично
```

Метод tellg(). Иногда нужно получать информацию о том, сколько уже прочитано. В этом поможет метод tellg():

```
cout << "Считано байт: " << file.tellg();
```

Он возвращает значение типа int, которое показывает, сколько уже пройдено в байтах. Его можно использовать в паре с методом seekg(), чтоб получать размер файла:

```
//становимся в конец файла
file.seekg(0,ios_base::end);
//Получаем текущую позицию
cout << "Размер файла (в байтах): " << file.tellg();
```

3.3 Запись и чтение структуры из бинарного файла

В качестве примера воспользуемся структурой студент, содержащей два поля – фамилия и возраст.

```
struct student {
    char Name[30];     //Фамилия студента
    int Age;            //Возраст студента
};
```

Для записи информации в файл и чтения информации из файла введены три дополнительные переменные:

```
int main()
{
    const char *FileName="D:\\student.dat"; //Путь к файлу
                                            // вписывайте свой.
    student st_out; //st_out – записываемый в файл
                    // объект структуры
    student st_in;  //st_in – читаемый из файл
                    // объект структуры

    /*----    ЗАПОЛНЯЕМ СТРУКТУРУ    ----*/
    strncpy(st_out.Name, "Ivanov", 30); //в случае работы
                  // со строками в виде массива символов,
                  // нужно копировать строку в строку
    st_out.Age = 20;
```

```
/*Запись структуры в файл*/
ofstream f1(FileName, ios::binary | ios::out);
f1.write((char*)&st_out, sizeof(student));
f1.close();

/*Чтение структуры из файла */
ifstream f2(FileName, ios::binary | ios::in);
f2.read((char*)&st_in, sizeof(student));
f2.close();

/*Вывод данных на экран*/
cout << st_in.Name << '\t' << st_in.Age << '\n';
```

Дополнительную информацию о системе ввода-вывода на языке C++ можно получить в [5].

3.4. Вопросы для самопроверки

1. Дайте определение бинарного файла

2. В каком режиме производится обработка двоичных файлов?

3. Приведите алгоритм записи информации в бинарный файл. В чем специфика использования метода write?

4. Перечислите методы класса ofstream для обработки бинарного файла

3.5. Задания для самопроверки

Выполните задания из лабораторной работы №3 (приложение 1).

4. КЛАССЫ В C++

4.1. Знакомство с объектно-ориентированным программированием

Язык C++ является объектно-ориентированным языком, т.е. поддерживает принципы объектно-ориентированного программирования (ООП).

Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса. Классы могут образовывать иерархию наследования.

Объектно-ориентированный подход к программированию как к моделированию информационных объектов, решает на новом уровне основную задачу структурного программирования.

Структурирование информации улучшает управляемость самим процессом моделирования, что особенно важно при реализации крупных проектов.

Основными принципами ООП (тремя парадигмами ООП) являются инкапсуляция, наследование и полиморфизм.

1) Инкапсуляция – это размещение в одном компоненте данных и методов их обработки при скрытии внутренней реализации от других компонентов. Например, доступ к скрытой переменной может предоставляться не напрямую, а с помощью методов для чтения и изменения её значения [1, 3, 4].

Свойство инкапсуляция повышает степень абстракции программы: данные разработанного класса и реализация его методов находятся ниже уровня абстракции, и при написании программы информация о них не нужна. Также инкапсуляция позволяет модифицировать реализацию класса без изменения основной части программы, если интерфейс остался прежним.

Инкапсуляция позволяет создавать библиотеки классов для применения во других программах, а также быть уверенным, что класс не модифицирует не принадлежащие ему области памяти.

2) Наследование – концепция объектно-ориентированного программирования, согласно которой тип данных может наследовать данные и функциональность некоторого существующего типа, спо-

собствуя повторному использованию компонентов программного обеспечения. Наследование предоставляет возможность создания иерархии классов, когда потомки наследуют все свойства своих предков, могут их изменять и добавлять новые. Свойства при наследовании повторно не описываются, что сокращает объем программы. Выделение общих черт различных классов в один класс-предок является мощным механизмом абстракции [1, 3, 4].

Иерархия классов представляется в виде направленного графа близкого к древовидной структуре, в котором более общие классы располагаются ближе к условному корню, а более специализированные - на условных ветвях и листьях. В C++ каждый класс может иметь сколько угодно потомков и предков. Иногда предки называются базовыми классами, надклассами или суперклассами, а потомки – производными классами, подклассами или субклассами.

3) Полиморфизм – принцип ООП, согласно которому возможно использовать в различных классах одно имя для обозначения сходных по смыслу действий и гибко выбирать требуемое действие во время выполнения программы [1, 3, 4].

Понятие полиморфизма используется в C++ весьма широко. Простым примером полиморфизма может служить перегрузка функций, когда из нескольких вариантов выбирается наиболее подходящая функция по соответствию ее прототипа передаваемым параметрам (статический полиморфизм). Другой пример – использование шаблонов функций, когда один и тот же код видоизменяется в соответствии с типом, переданным в качестве параметра. Чаще всего понятие полиморфизма связывают с механизмом виртуальных методов (динамический полиморфизм).

Использование при программировании понятий, более близких к предметной области, благодаря представлению программы в терминах поведения объектов является большим преимуществом ООП. Однако проектирование такой программы представляет собой весьма

сложную задачу, поскольку в процесс добавляется еще один важный этап – разработка иерархии классов. Неправильно спроектированная иерархия приводит к созданию сложных и запутанных программ. Важно до начала проектирования правильно определить, требуется ли вообще применять объектно-ориентированный подход. Если в иерархии классов нет необходимости, то, как правило, достаточно ограничиться модульной технологией.

Класс, также как и структура, представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов.

Другие абстрактные типы данных – метаклассы, интерфейсы, структуры, перечисления, – характеризуются своими специфическими особенностями. Класс является ключевым понятием в ООП.

Класс – это описание определяемого типа. Любой тип данных представляет собой множество значений и набор действий, которые разрешается выполнять с этими значениями. Например, сами по себе числа не представляют интереса – нужно иметь возможность ими оперировать: складывать, вычитать, вычислять квадратный корень и т. д. В C++ множество значений нового типа определяется задаваемой в классе структурой данных, а действия с объектами нового типа реализуются в виде функций и перегруженных операций C++.

Данные класса называются полями (по аналогии с полями структуры), а функции класса - методами. Поля и методы называются элементами класса. Описание класса в первом приближении выглядит так:

```
class <имя>{
[ private: ]
<описание скрытых элементов>
public:
<описание доступных элементов>
}; // Описание заканчивается точкой с запятой
```

Спецификаторы доступа `private` и `public` управляют видимостью элементов класса. Элементы, описанные после служебного слова `private`, видимы только внутри класса. Этот вид доступа принят в классе по умолчанию. Интерфейс класса описывается после спецификатора `public`. Действие любого спецификатора распространяется до следующего спецификатора или до конца класса. Можно задавать несколько секций `private` и `public`, порядок их следования значения не имеет.

Поля класса:

– могут быть простыми переменными любого типа, указателями, массивами и ссылками (т.е. могут иметь практически любой тип, кроме типа этого же класса, но могут быть указателями или ссылками на этот класс);

– могут быть константами (описаны с модификатором `const`), при этом они инициализируются только один раз (с помощью конструктора) и не могут изменяться;

– могут быть описаны с модификатором `static`, но не как `auto`, `extern` и `register`.

Классы могут быть глобальными (объявленными вне любого блока) и локальными (объявленными внутри блока, например, внутри функции или внутри другого класса). Обычно классы определяются глобально.

Локальные классы имеют некоторые особенности:

– локальный класс не может иметь статических элементов (уточнение: современные стандарты допускают это);
– внутри локального класса можно использовать из охватывающей его области типы, статические (`static`) и внешние (`extern`) переменные, внешние функции и элементы перечислений;

– запрещается использовать автоматические переменные из охватывающей класс области;
– методы локальных классов могут быть только
– встроенными (inline); (уточнение: согласно современным стандартам это необязательно)
– если один класс вложен в другой класс, они не имеют каких-либо особых прав доступа к элементам друг друга и могут обращаться к ним только по общим правилам.

Пример 1. Описать класс Student, содержащий следующую информацию: фамилия, имя, оценки по трем дисциплинам. По аналогии с типом данных «структура» сначала производится описание полей класса.

Листинг 1. Описание класса Student

```
class Student // Описание класса студент
{
// список данных класса Student,
// по умолчанию являются скрытыми
      string LastName;
      string Name;
      int Math;
      int Phys;
      int Rus;
};
```

В этом классе все поля закрытые по умолчанию, получить их значения напрямую нельзя. Доступ к полям с помощью методов в данном случае кажется искусственным усложнением, но надо учитывать, что полями реальных классов могут быть сложные динамические структуры, и получение значений их элементов не так тривиально. Кроме того, очень важной является возможность вносить в эти структуры изменения, не затрагивая интерфейс класса.

Методами называются функции-члены класса. Методы класса имеют неограниченный непосредственный доступ к его полям. Внут-

ри метода можно объявлять объекты, указатели и ссылки как на свой, так и на другие классы.

4.2. Конструкторы

Для создания объекта класса используется специальный метод, который называется конструктором. Формально, это даже не метод, а специальная конструкция языка. Имя конструктора всегда совпадает с именем класса и, в отличие от обычных методов, конструктор не имеет типа возвращаемого значения. Он вызывается автоматически при создании объекта класса и предназначен для его инициализации.

Автоматический вызов конструктора позволяет избежать ошибок, связанных с использованием неинициализированных переменных.

Ниже перечислены основные свойства конструкторов:

- нельзя получить указатель на конструктор;
- класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации (при этом используется механизм перегрузки);
- параметры конструктора могут иметь любой тип, кроме этого же класса.

Конструктор без параметров, называется конструктором по умолчанию. Можно задавать значения параметров по умолчанию. Их может содержать только один из конструкторов. Если программист не указал ни одного конструктора, компилятор создает его автоматически (кроме случая, когда класс содержит константы и ссылки, поскольку их необходимо инициализировать). Такой конструктор вызывает конструкторы по умолчанию для полей класса и конструкторы базовых классов.

При объявлении объектов вызывается один из конструкторов. При отсутствии инициализирующего выражения в объявлении объекта вызывается конструктор по умолчанию, при инициализации другим объектом того же типа – конструктор копирования, при инициализации полей – один из явно определенных конструкторов инициализации (т.е. конструкторов, которым передаются параметры для инициализации полей объекта).

Конструкторы часто вызываются неявно для создания временных объектов. Обычно это происходит в следующих случаях:

- при инициализации;
- при выполнении операции присваивания;
- для задания значений параметров по умолчанию;
- при создании и инициализации массива;
- при создании динамических объектов;
- при передаче параметров в функцию и возврате результатов по значению.

Как уже упоминалось выше, при отсутствии явного описания конструктора, компилятор создает его автоматически. Для класса `Student` (листинг 1) конструктор по умолчанию будет создавать объект, значениями полей которого будут случайные числа. Для того, чтобы новый объект инициализировался нулевыми значениями, в классе нужно явно объявить конструктор (листинг 2).

Для повышения эффективности и наглядности, описание класса помещено в файл «`student.h`», а описание его методов – в файле «`student.cpp`». В описании класса в разделе `public` указывается метод, далее его описание. Обратите внимание на способ инициализации полей класса в конструкторе.

Листинг 2. Описание класса студент с конструктором по умолчанию.

```
// файл student.h ------------
class Student // Описание класса студент
{
// список данных класса Student
```

```
    string LastName;
    string Name;
    int Math;
    int Phys;
    int Rus;
public:
// конструктор по умолчанию в объявлении класса
    Student();

// сюда далее будут добавлены методы класса Student
};

// файл student.cpp  -----------
#include "student.h"
// Описание конструктора.
// конструктор по умолчанию
Student::Student():LastName(""), Name(""),
                   Math(0), Phys(0), Rus(0)
{ } // пустое тело конструктора
```

Конкретные переменные типа данных "класс" называются экземплярами класса, или объектами. Время жизни и видимость объектов зависит от вида и места описания и подчиняется общим правилам C++. Создание объекта осуществляется по аналогии с созданием переменных других типов.

```
Имя_класса имя_объекта;
```

Например, создание объекта `Student1` класса `Student` производится следующим образом:

```
Student Student1;
```

При создании каждого объекта выделяется память, достаточная для хранения всех его полей, и автоматически вызывается конструктор, выполняющий их инициализацию. При выходе объекта из области действия он уничтожается, при этом автоматически вызывается деструктор. При создании объекта `Student1` будет выделена память для всех элементов и присвоены им нулевые значения.

Кроме конструктора по умолчанию, может возникнуть необходимость создания объекта с конкретными параметрами. Для этого необходимо описать конструктор с нужным набором действий.

Например, конструктор класса `Student`, инициализирующий все поля, имеет 5 параметров.

Листинг 3. Описание класса студент с двумя конструкторами.

```
// файл student.h -----------

class Student // Описание класса студент
{
// список данных класса Student
    string LastName;
    string Name;
    int Math;
    int Phys;
    int Rus;
public:
// методы класса Student
// конструктор по умолчанию в объявлении класса
    Student();

// конструктор со всеми параметрами
    Student(string TempLastName, string TempName,
                  int TempMath, int TempPhys, int TempRus);
};

// файл student.cpp ------------------

#include "student.h"

// конструктор со всеми параметрами
Student::Student(string TempLastName, string TempName,
                  int TempMath, int TempPhys, int TempRus):
          LastName(TempLastName),
          Name(TempName),
          Math(TempMath),
          Phys(TempPhys),
          Rus(TempRus)
{} // пустое тело конструктора
```

Создание объекта с использованием полного конструктора:

```
Student Student2("Ivanov","Ivan",90,90,90);
```

Для вывода информации об объекте на экран нужно создать метод класса, реализующий эти действия.

Листинг 4. Добавление метода PrintStudent.

```
// файл student.h --------------

class Student // Описание класса студент
{
// список данных класса Student
    string LastName;
    string Name;
    int Math;
    int Phys;
    int Rus;
public:
// методы класса Student
// конструктор по умолчанию в объявлении класса
    Student();

// конструктор со всеми параметрами
    Student(string TempLastName, string TempName,
                int TempMath, int TempPhys, int TempRus);

// метод вывода данных об одном объекте на экран
    void PrintStudent();
};

// фрагмент файла student.cpp -------------------

// метод вывода данных об одном объекте на экран
void Student::PrintStudent()
{
     cout << setw(10) << LastName << setw(10) <<
             Name << setw(10) << Math <<
             setw(10) << Phys << setw(10) << Rus << endl;
}
```

Доступ к открытым (public) элементам объекта аналогичен доступу к полям структуры. Для этого используются операция селектор . (point) при обращении к элементу через имя объекта и операция -> (arrow) при обращении через указатель:

```
объект.поле
указатель -> поле
(*указатель).поле
объект.метод( параметры )
указатель -> метод( параметры )
(*указатель).метод( параметры )
```

Для вывода данных объектов Student на экран вызов функции осуществляется с помощью операций селекторов point:

```
Student1.PrintStudent();
Student2.PrintStudent();
```

Значения полей объекта класса могут быть введены с помощью метода класса. Метод SetConsole() осуществляет запрос всех полей с клавиатуры.

Листинг 5. Добавление метода SetConsole().

```
// файл student.h ---------------------
class Student // Описание класса студент
{
// список данных класса Student
    string LastName;
    string Name;
    int Math;
    int Phys;
    int Rus;
public:
// методы класса Student
// конструктор по умолчанию в объявлении класса
    Student();

// конструктор со всеми параметрами
    Student(string TempLastName, string TempName,
                 int TempMath, int TempPhys, int TempRus);
```

```
// метод вывода данных об одном объекте на экран
    void PrintStudent();

// метод задания данных с клавиатуры
    void SetConsole();
};

// файл student.cpp ----------------------

#include "student.h"

// метод задания данных с клавиатуры
void Student::SetConsole()
{
    cout << "NameLast: "; cin >> LastName; //Ввод фамилии
    cout << "Name: "; cin >> Name; //Ввод имени
    cout << "Math: "; cin >> Math; //Ввод оценки по матем.
    cout << "Phys: "; cin >> Phys; //Ввод оценки по физике
    cout << "Rus: "; cin >> Rus; //Ввод оценки по русскому
    SumMates = Math + Phys + Rus;
}
```

Если тело метода определено внутри класса, он является встроенным (`inline`). Как правило, встроенными делают короткие методы. Например, встроенными делают методы установки значения поля («сеттеры») и методы получения значения поля («геттеры»), которые устанавливают и возвращают значения полей по отдельности.

Листинг 6. Добавление методов получения/установки значений полей.

```
// файл student.h -------------------

class Student // Описание класса студент
{
// список данных класса Student
    string LastName;
    string Name;
    int Math;
    int Phys;
    int Rus;
public:
// методы класса Student
```

```
// конструктор по умолчанию в объявлении класса
    Student();
// конструктор со всеми параметрами
    Student(string TempLastName, string TempName,
                int TempMath, int TempPhys, int TempRus);

// методы установки и получения значений
// закрытых полей (сеттеры и геттеры)
    void SetLastName(string str) { LastName = str; }
    void SetName(string str) { Name = str; }
    void SetMath(int data) { Math = data; }
    void SetPhys(int data) { Phys = data; }
    void SetRus(int data) { Rus = data; }
    string GetLastName() { return LastName; }
    int GetMath() { return Math; }
    int GetPhys() { return Phys; }
    int GetRus() { return Rus; }

// метод вывода данных об одном объекте на экран
    void PrintStudent();

// метод задания данных с клавиатуры
    void SetConsole();
};
```

Если внутри класса записано только объявление (заголовок) метода, сам метод должен быть определен в другом месте программы с помощью операции доступа к области видимости. Встроенные методы можно определить и вне класса с помощью директивы `inline` (как и для обычных функций, она носит рекомендательный характер).

В практической работе необходимо обрабатывать набор объектов. Для хранения такого набора можно использовать структуру данных массив. Описание одномерного статического массива объектов имеет вид:

```
имя_класса имя_массива[размер];
```

Например, описание массива, содержащего 10 элементов объектов класса `Student`:

```
Student ArrayStudent[10];
```

Обращение к открытому полю и вызов метода для массива объектов:

```
имя_массива[индекс].поле
имя_массива[индекс].метод(параметры)
Например:
int n = ArrayStudent[0].GetMath();
ArrayStudent[i].PrintStudent();
```

Аналогично создаются динамические массивы объектов и выделяется память для необходимого количества данных.

```
Student* ArrayStudent;
ArrayStudent = new Student[CountStudent];
```

Получить или изменить значения private элементов (полей и методов) можно только через обращение к соответствующим методам.

Можно создать константный объект, значения полей которого изменять запрещается. К нему должны применяться только константные методы.

Константный метод:

- объявляется с ключевым словом `const` после списка параметров;
- не может изменять значения полей класса;
- может вызывать только константные методы;
- может вызываться для любых (не только константных) объектов.

Рекомендуется описывать как константные те методы, которые предназначены для получения значений полей. Каждый объект содержит свой экземпляр полей класса. Методы места в классе не занимают и не дублируются для каждого объекта.

Единственный экземпляр метода используется всеми объектами совместно, поэтому нестатический метод должен "знать", для какого объекта он вызван.

Каждый нестатический метод, помимо явно объявленных параметров, получает еще один скрытый параметр: константный указатель на объект, для которого он вызван. В C++ это указатель обозначается зарезервированным словом `this`. Когда имя параметра метода совпадает с именем поля класса, доступ к полю выполняется через этот указатель.

Выражение `*this` представляет собой разыменование указателя и имеет тип определяемого класса. Обычно это выражение возвращается в качестве результата, если метод возвращает ссылку на свой класс (`return *this;`).

4.3. Деструкторы

Деструктор - это особый вид метода, применяющийся для освобождения памяти или других ресурсов, занимаемых объектом. Деструктор вызывается автоматически, когда объект разрушается:

– для локальных переменных - при выходе из блока, в котором они объявлены;

– для глобальных - как часть процедуры выхода из `main()` ;

– для объектов, заданных через указатели, деструктор вызывается неявно при использовании операции `delete` (автоматический вызов деструктора при выходе указателя из области действия не производится).

При уничтожении массива объектов, деструктор вызывается для каждого элемента удаляемого массива. Для динамических объектов деструктор вызывается при уничтожении объекта операцией `delete`.

При выполнении операции `delete[]` деструктор также вызывается для каждого элемента.

Имя деструктора начинается с символа тильды (~), непосредственно за которой следует имя класса. Деструктор:

– не имеет аргументов и возвращаемого значения;
– не может быть объявлен как `const` или `static`;
– не наследуется;
– может быть виртуальным;
– может вызываться явным образом путем указания полностью уточненного имени; это необходимо для объектов, которым с помощью new выделялся конкретный адрес.

Если деструктор явным образом не определен, компилятор создает его автоматически.

Описывать в классе деструктор явным образом требуется в случае, когда объект содержит указатели на память, выделяемую динамически – иначе при уничтожении объекта память, на которую ссылались его поля-указатели, не будет помечена как свободная.

Указатель на деструктор определить нельзя. Деструктор, например, может выглядеть так :

```
MyClass::~ MyClass()
{
    delete [] name;
}
```

4.4. Дружественная функция

Для ограничения доступа и повышения защиты информации, поля класса являются закрытыми (`private`). Однако может возникнуть ситуация, когда есть класс и функция, которая работает с этим классом, но которая не находится в его теле. Например, есть класс, в котором хранятся данные, и функция (или другой класс), которая выводит эти данные на экран. Хотя класс функция вывода разделены (для упрощения поддержки кода), код функции вывода тесно связан с данными класса. Следовательно, сделав члены класса private, нужный эффект достигнут не будет.

Существует два способа решения этой проблемы:

1. Сделать открытыми методы класса и через них функция будет взаимодействовать с классом. Однако здесь есть несколько нюансов. Во-первых, эти открытые методы нужно будет определить, на что потребуется время, и они будут загромождать интерфейс класса. Во-вторых, в классе нужно будет открыть методы, которые не всегда должны быть открытыми и предоставляющими доступ извне.

2. Использовать дружественные классы и дружественные функции, с помощью которых можно будет предоставить функции вывода доступ к закрытым данным класса. Это позволит функции вывода напрямую обращаться ко всем закрытым переменным-членам и методам класса, сохраняя при этом закрытый доступ к данным класса для всех остальных функций вне тела класса.

Дружественная функция — это функция, которая имеет доступ к закрытым членам класса, как если бы она сама была членом этого класса. Во всех других отношениях дружественная функция является обычной функцией. Ею может быть, как обычная функция, так и метод другого класса. Для объявления дружественной функции используется ключевое слово `friend` перед прототипом функции, которую вы хотите сделать дружественной классу. При этом дружественная функция может быть закрытой или открытой.

Для класса `Student` удобно перегрузить операции ввода `>>` и вывода `<<` для обработки объектов класса. Это можно сделать, используя определение дружественной функции. Эти операции могут быть использованы как для консольного ввода-вывода, так и для чтения-записи информации в файл.

```
// перегрузка операторов ввода и вывода
// внутри определения класса Student

// Оператор Вывода:
friend ostream& operator << (ostream & out, Student & S) {
```

```
    return out << setw(10) << S.LastName <<
                  setw(10) << S.Name << setw(10) <<
                  S.Math << setw(10) << S.Phys <<
                  setw(10) << S.Rus << endl;
}

. . .
// в cpp файле...

// Оператор Ввода:
friend istream& operator >> (istream& in, Student & S) {
    in >> S.LastName; //Ввод фамилии
    in >> S.Name; //Ввод имени
    in >> S.Math; //Ввод оценки по математике
    in >> S.Phys; //Ввод оценки по физике
    in >> S.Rus; //Ввод оценки по русскому
    return in;
}
```

4.5. Рекомендации по составу класса

Как правило, класс как тип, определенный пользователем, содержит скрытые поля и следующие функции:

- конструкторы, определяющие, как инициализируются объекты класса;
- набор методов доступа к скрытым элементам класса (эти методы описываются с модификатором const, указывающим, что они не должны изменять значения полей);
- набор методов, изменяющих значения полей;
- набор операций, позволяющих присваивать, сравнивать
- объекты и производить с ними другие действия, требующиеся по сути класса;
- класс исключений, используемый для сообщений об ошибках с помощью генерации исключительных ситуаций.

Пример. В текстовом файле «STUDENT.TXT» записана информация о студентах (файл считать корректным). Создать файл

«MAX.TXT», содержащий список студентов, имеющих максимальную сумму баллов.

Листинг 7. Создание списка студентов.

```
// файл student.h  ---------------------------
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string.h>

using namespace std;

// Описание класса студент
class Student
{
    // список полей (данных) класса Student
    string LastName;
    string Name;
    int Math;
    int Phys;
    int Rus;

public:
    // методы класса Student
    // конструктор по умолчанию в объявлении класса
    Student();

    // конструктор со всеми параметрами
    Student(string TempLastName, string TempName,
              int TempMath, int TempPhys, int TempRus);

    // деструктор
    ~Student();

    // сеттеры и геттеры
    void SetLastName(string str) { LastName = str; }
    void SetName(string str) { Name = str; }
    void SetMath(int data) { Math = data; }
    void SetPhys(int data) { Phys = data; }
    void SetRus(int data) { Rus = data; }
    string GetLastName() { return LastName; }
    string GetName() { return Name; }
```

```
    int GetMath() { return Math; }
    int GetPhys() { return Phys; }
    int GetRus() { return Rus; }

    // метод вычисления суммы баллов
    int SumMates() { return Math + Phys + Rus; }

    // метод задания данных с клавиатуры
    void SetConsole();

    // перегрузка оператора вывода (для записи):
    friend ostream& operator<<(ostream & out, Student & S);

    // перегрузка оператора ввода (для чтения):
    friend istream& operator>>(istream& in, Student & S);
};

// файл student.cpp ---------------------------------

#include <iostream>
#include <iomanip>
#include <fstream>
#include <string.h>
#include "student.h"

using namespace std;

// конструктор по умолчанию
Student::Student():LastName(""), Name(""), Math(0),
                   Phys(0), Rus(0)
{}

// конструктор со всеми параметрами
Student::Student(string TempLastName, string TempName,
                  int TempMath, int TempPhys, int TempRus):
        LastName(TempLastName),
        Name(TempName),
        Math(TempMath),
        Phys(TempPhys),
        Rus(TempRus)
{}

// деструктор (пустой деструктор не обязателен)
```

```
Student::~Student()
{}

// метод задания данных с клавиатуры
void Student::SetConsole()
{
    cout << "NameLast: "; cin >> LastName; //Ввод фамилии
    cout << "Name: "; cin >> Name; //Ввод имени
    cout << "Math: "; cin >> Math; //Ввод оценки по матем.
    cout << "Phys: "; cin >> Phys; //Ввод оценки по физике
    cout << "Rus: "; cin >> Rus; //Ввод оценки по русскому
}

// функции-друзья -----------
// перегрузка оператора ввода (для чтения):
istream& operator >> (istream& in, Student & S) {
    in >> S.LastName; //Ввод фамилии
    in >> S.Name; //Ввод имени
    18
    in >> S.Math; //Ввод оценки по математике
    in >> S.Phys; //Ввод оценки по физике
    in >> S.Rus; //Ввод оценки по русскому
    return in;
}

// перегрузка оператора вывода (для записи):
ostream& operator << (ostream & out, Student & S) {
    return out << setw(10) << S.LastName << setw(10) <<
           S.Name << setw(10) << S.Math << setw(10) <<
           S.Phys << setw(10) << S.Rus << endl;
}

// файл listing7.cpp --------------------------------

#include "student.h"

int main()
{
    system("chcp 1251 > nul");
    // подсчет строк в исходном файле
    ifstream InputFile("STUDENT.TXT");

    // проверка открытия файла
```

```
if (!InputFile)
{
    cout << "Ошибка открытия файла" << endl;
    return 0;
}

// проверка файла на пустой файл
if (InputFile.peek() == EOF)
{
    cout << "Файл не содержит данных" << endl;
    return 0;
}

// обработка файла
int CountStudent = 0;  // количество строк
char StringBuffer[80]; // буферная переменная для
                       // строк файла
while (!InputFile.eof())
{
    InputFile.getline(StringBuffer,80);
    CountStudent++;
}

InputFile.clear();
InputFile.seekg(0, ios_base::beg); // возврат в начало
                                   // файла

// создание массива объектов класса
Student* ArrayStudent;
ArrayStudent = new Student[CountStudent];

// считывание данных из файла
string StrBuffer = ""; // буферная строковая переменная
int DataBuffer = 0; // буферная целочисленная перем.

for (int i = 0; i < CountStudent; i++) {
    InputFile >> StrBuffer;
    ArrayStudent[i].SetLastName(StrBuffer);
    InputFile >> StrBuffer;
    ArrayStudent[i].SetName(StrBuffer);
    InputFile >> DataBuffer;
    ArrayStudent[i].SetMath(DataBuffer);
    InputFile >> DataBuffer;
```

```
        ArrayStudent[i].SetPhys(DataBuffer);
        InputFile >> DataBuffer;
        ArrayStudent[i].SetRus(DataBuffer);
    }

    InputFile.close();

    // вывод данных на экран
    cout << "\t\tList of Students" << endl;
    for (int i = 0; i < CountStudent; i++)
    {
          cout << ArrayStudent[i];
    }

    // нахождение максимальной суммы баллов
    int MaxIndex = 0;
    for (int i = 1; i < CountStudent; i++)
    {
         if (ArrayStudent[i].SumMates() >
                          ArrayStudent[MaxIndex].SumMates())
               MaxIndex = i;
    }

    // проверка записи в файл
    ofstream OutputFile("MAX.TXT");
    for (int i = 0; i < CountStudent; i++)
    {
        if (ArrayStudent[i].SumMates() ==
                          ArrayStudent[MaxIndex].SumMates())
              OutputFile << ArrayStudent[i];
    }

    OutputFile.close();

    delete [] ArrayStudent;
    system("pause");
    return 0;
}
```

4.6. Вопросы для самопроверки

1. Дайте определение инкапсуляции. Какие два аспекта инкапсуляции можно выделить. Как они реализуются в языке C++?

2. Что такое конструктор класса? Для чего нужны конструкторы? Когда и кем они вызываются?

3. Что такое деструктор класса? Для чего нужны деструкторы? Когда и кем они вызываются?

4. Что такое «дружественные функции» класса? Для чего они применяются? Какую парадигму объектно-ориентированного программирования они нарушают?

4.7. Задания для самопроверки

Выполните задания из лабораторной работы №4 (приложение 1).

5. ЛИНЕЙНЫЕ ОДНОСВЯЗНЫЕ СПИСКИ

5.1. Связные списки как динамические структуры данных

Структура данных – программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в программе. Каждая такая структура имеет некоторый набор функций, таких как добавление, поиск, изменение хранящихся данных и удаление элементов, используя которые, разработчик решает поставленные перед ним задачи.

Выделяют линейные и нелинейные структуры данных. К линейным относятся массив, стек, очередь, дек и другие; к нелинейным – графы, деревья и другие.

Линейные связные списки представляют собой динамическую структуру данных. Динамическими такие структуры называются потому, что позволяют легко изменять свой размер: добавлять и удалять

элементы. На практике наиболее часто используют односвязные (однонаправленные) и двусвязные (двунаправленные) списки. Способ хранения данных позволяет гибко менять конфигурацию списка. Так, если соединить начало списка и его конец, можно получить кольцевой (циклический) список.

Односвязный (однонаправленный) список – представляет собой набор элементов, каждый из которых содержит данные для хранения и указатель на следующий элемент (рисунок 5.1). В последнем элементе указатель на следующий элемент устанавливается равным нулю, как признак конца списка.

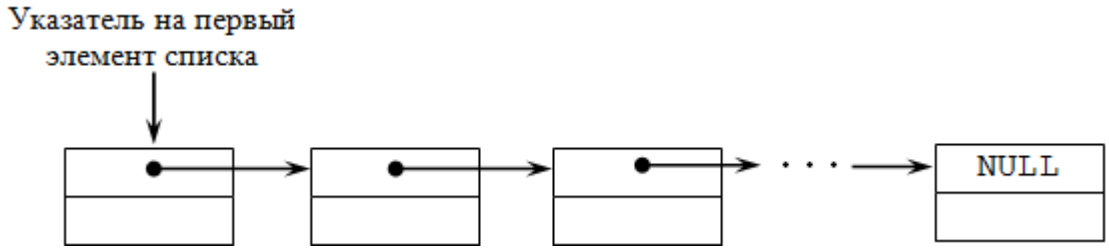


Рис. 5.1. Линейный однонаправленный односвязный список

Кроме указателя на следующий элемент, в каждой ячейке списка содержатся данные для хранения. Таким образом, каждый элемент списка разделён на две части: служебную (указатель на следующий элемент), для организации списка и, собственно, данные.

5.2. Основные операции со списком

Можно выделить следующие основные операции, необходимые для работы со списком:

- создание списка;
- вставка элемента в список (в начало, конец, середину и т.д.);
- удаление элемента из списка (из начала, конца и т.д.);
- поиск элемента с определенными данными;
- проверка, является ли список пустым;
- удаление списка.

Особое внимание следует обратить на то, что при выполнении любых операций с линейным однонаправленным списком необходимо

иметь указатель на первый элемент списка, который также называют указателем на вершину списка или указателем на голову списка. При потере такого указателя, весь список становится недоступен, а в программе происходит «утечка» памяти: выделенная под элементы списка оперативная память остается занятой и не доступна для освобождения.

Однонаправленные связные списки – это рекурсивные структуры данных: любая часть списка, также является однонаправленным связным списком. По сути, даже пустой (нулевой) указатель можно рассматривать как указатель на вершину пустого списка. В связи с вышесказанным, можно отметить, что при обработке данных в списках иногда удобно использовать рекурсивные реализации функций, выполняющих основные операции.

Создание односвязного списка

Изначально, когда список пуст, указатель на вершину равен нулю. В процессе работы, сначала создается «отдельно стоящий» элемент списка и после того, как указателю на вершину присваевается его адрес, он становится первым элементом. Затем добавяются (в начало, в конец или в середину) остальные элементы по тому же принципу.

Проход по односвязному списку

Операция прохода по списку заключается в последовательном получении указателя на очередной элемент списка и выполнении операций обработки данных, хранящихся в нем. Для этого создается функция, в которой нужно переставлять указатель на следующий элемент списка (начиная с первого) до тех пор, пока указатель не станет равен нулю, то есть будет достигнут конец списка. Такая функция может быть выполнена как с использованием итерационных алгоритмов, так и рекурсивно. Получив указатель на текущий элемент списка, можно выполнить над ним какие-либо действия. Алгоритм прохо-

да по списку является частью других алгоритмов и операций над списками.

Вставка элемента

Поскольку односвязные списки являются динамическими структурами данных, операция добавления (вставки) элемента выполняется достаточно просто. Для этого необходимо создать в памяти (обычно в куче) экземпляр элемента, а затем «подключить» его к списку, изменив адрес какого-либо «указателя на следующий элемент». Если элемент будет добавляться в начало списка, то изменять нужно указатель на вершину. Алгоритм вставки элемента в начало списка таков:

- создаем экземпляр элемента списка;
- указателю на следующий элемент нового элемента присваиваем зачение указателя на вершину списка;
- указателю на вершину списка присваиваем адрес нового элемента.

В случае вставки в середину или конец списка, необходимо получить указатель на элемент после которого будет выполнено добавление. Алгоритм вставки элемента в середину (конец) списка таков:

- создаем экземпляр элемента списка;
- проверяем пустой ли список; если список пуст, выполняем алгоритм вставки в начало списка;
- если список не пуст:
    - получаем указатель **P** на элемент, после которого будет осуществляться вставка;
    - указателю на следующий элемент нового элемента присваиваем зачение указателя на следующий элемент элемента **P**;
    - указателю на следующий элемент элемента **P** присваиваем зачение указателя на новый элемент.

Данный алгоритм проиллюстрирован на рисунке 5.2.

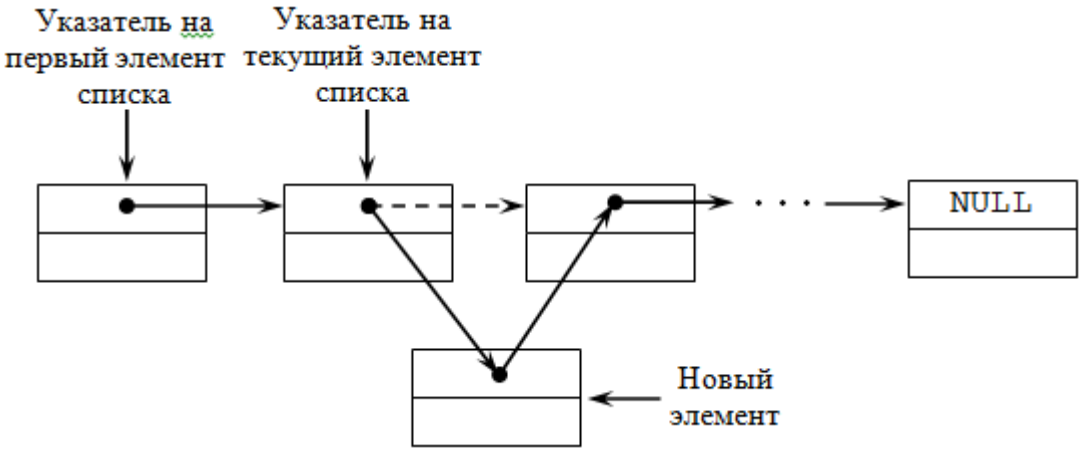


Рис. 5.2 – Вставка элемента в однонаправленный связный список

Удаление элемента из однонаправленного списка

Удаление элемента связного списка также представляет собой простую операцию. Для этого нам необходимо получить указатель на элемент, предшествующий удаляемому и изменить его указатель на следующий элемент, пропустив удаляемый элемент. Таким образом, мы исключаем элемент из списка, хотя он остается в оперативной памяти. Далее можно с ним работать, как с самостоятельным объектом, а можно уничтожить, вызвав оператор `delete`.

При выполнении удаления необходимо, чтобы операция выполнялась корректно (без аварийного завершения программы) в том числе и для пустого списка. В случае удаления первого элемента, данная операция изменяет указатель на вершину списка.

Алгоритмы удаления элемента из вершины списка следующий:

- проверяем пустой ли список; если список пуст, завершаем операцию;
- если список не пуст:
    - указатель на вершину списка сохраняем в переменной **P**;
    - указателю на вершину списка присваиваем значение указателя на следующий элемент первого списка;
    - освобождаем память по адресу **P**.

Данный алгоритм уничтожает удаляемый элемент.

В случае удаления из середины или конца списка, необходимо получить указатель на элемент предшествующий удаляемому. Алгоритм удаления элемента из середины (конца) списка таков:

- проверяем пустой ли список; если список пуст, завершаем операцию;
- если список не пуст:
    - получаем указатель **P1** на элемент, предшествующий удаляемому;
    - получаем указатель **P2** на удаляемый элемент (указатель на следующий элемент элемента **P1**);
    - указателю на следующий элемент элемента **P1** присваиваем зачение указателя на следующий элемент удаляемого элемента **P2**;
    - освобождаем память по адресу **P2**.

Данный алгоритм проиллюстрирован на рисунке (рис. 5.3)

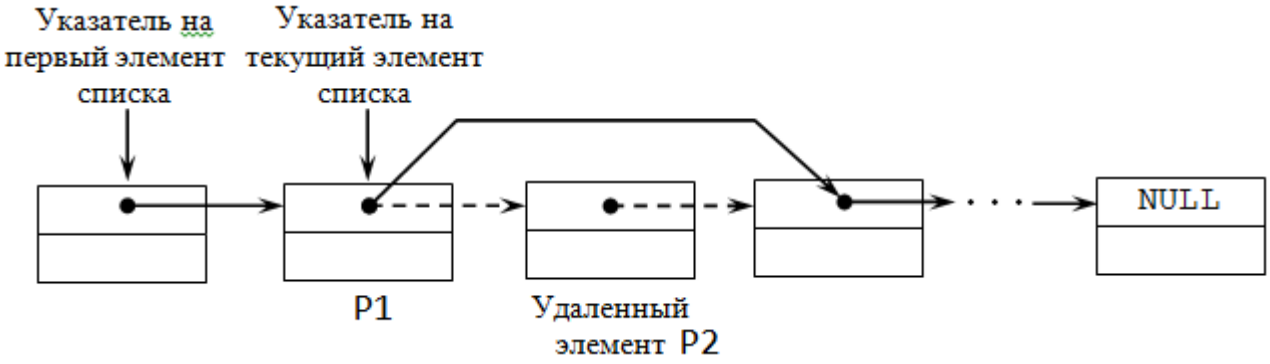


Рис. 5.3 – Удаление элемента из связного списка

Поиск элемента

Операция поиска элемента основывается на алгоритме прохода по списку. При этом каждый раз применяется операция сравнения данных, хранящихся в элементе списка с некоторым значением поиска. Поиск может осуществляться до первого совпадения, тогда функция обычно возвращает указатель (или ссылку) на найденный элемент

(нулевой указатель, если совпадений не было). Также возможна реализация, позволяющая находить все элементы списка, удовлетворяющие критерию поиска. В этом случае функция должна создавать новый связный список, состоящий из найденных элементов (из их копий) и возвращать указатель на его вершину. В случае, когда ни один элемент не найден, возвращается указатель на пустой список.

Удаление связного списка

Операция удаления списка целиком состоит из удаления всех элементов и присваиванию указателю на вершину списка значения нуля. При этом можно задействовать функции удаления (например из вершины списка) и проверки на пустой список.

5.3 Двусвязные (двунаправленные) списки

Односвязный список является простейшей реализацией связного списка. Однако, несомотря на простоту, у него есть один существенный недостаток: перебирать элементы можно только в одном направлении – от вершины к концу списка. Это вынуждает разработчика выполнять лишние проходы по списку, особенно при работе с элементами в конце.

Для решения этой проблемы были разработаны двусвязные (двунаправленные) списки. Технически двусвязный (двунаправленный) список очень похож на одновсязный, но к элементу списка обыл добавлено еще одно поле: указатель на предыдущий элемент. Таким образом, два соседних элемента содержат взаимные указатели друг на друга (рис. 5.4).

Наличие указателей на следующий и предыдущий элементы, позволяет двигаться по списку в любом направлении: к концу списка или к его вершине, поэтому такой список называют двунаправленным.

Основные операции, осуществляемые с двусвязными списками, такие же, что и с односвязными:

– создание списка;
– вставка элемента в список (в начало, конец, середину и т.д.);
– удаление элемента из списка (из начала, конца и т.д.);
– поиск элемента с определенными данными;
– проверка, является ли список пустым;
– удаление списка;
– и другие.

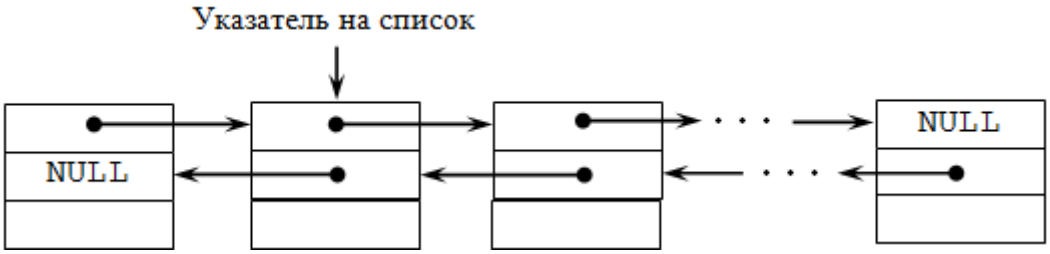


Рис. 5.4. Двусвязный (двунаправленный) список

Особое внимание следует обратить на то, что в отличие от односвязного списка нет необходимости строго хранить указатель на вершину (начало) списка. Достаточно иметь указатель на любой элемент: всегда есть возможность осуществлять движение в прямом и обратном направлениях. Тем не менее, во многих реализациях двусвязных списков указатель на вершину присутствует.

5.4. Стеки и очереди

В списках доступ к элементам происходит посредством адресации, при этом доступ к отдельным элементам не ограничен. Но существуют также и такие динамические структуры данных, в которых имеются ограничения доступа к элементам. Например, можно задать ограничение на порядок добавления и извлечения данных.

Стеки

Одной из таких структур является стек. Стек может быть создан как на базе обычного массива, так и на базе связного списка. Особен-

ностью реализации стека является порядок доваления/извлечения элементов.

Стек (англ. stack – стопка) – это динамическая структура данных, у которой добавление и извлечение нового элемента всегда осуществляется в одно определенное место: на вершину стека (рис. 5.5). Такой метод доступа к элементам получил название LIFO (Last Input – First Output, "последним пришел – первым ушел"). Принцип работы стека часто сравнивают со стопкой тарелок: взять очередную тарелку можно только сверху и положить новую туда же.

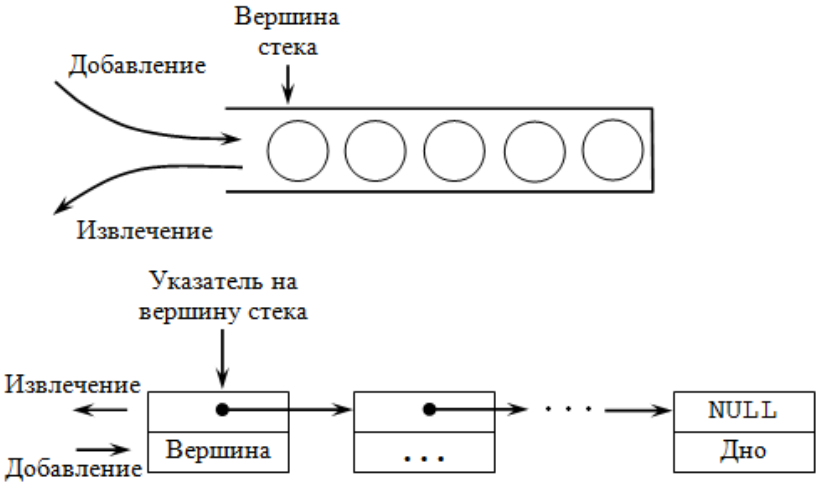


Рис. 5.5. Принцип работы стека

Таким образом, если добавить в стек несколько элементов, а потом извлечь их, мы получим эти элементы в обратном порядке. Например, если мы добавим последовательно в стек числа 5, 10, 15, то после извлечения получим 15, 10 , 5.

Очереди

Другим видом структур данных, определяемых порядком добавления и извлечения элементов, является очередь. Подобно очереди в магазине, новые элементы добавляются в конец очереди, а извлекаются из начала. В очередях используется принцип доступа к элемен-

там FIFO (First Input – First Output, "первый пришёл – первый ушёл"). Соответственно, очередь имеет одно место для добавления – это конец очереди и одно место для извлечения днных – это начало очереди (рис. 5.6).

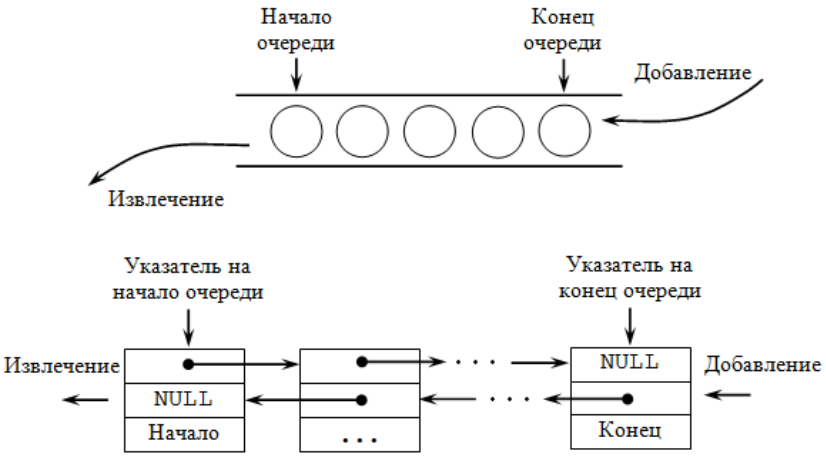


Рис. 5.6. Принцип работы очереди

Подобно стеку, очередь может быть реализована на базе различных видов хранлищ данных (массивов, связных списков и т.д.). Однако следует помнить, что для правильной работы необходимо предоставить одно место для добавления, и одно место для извлечения информации. Очереди обладают «упорядочивающим» эффектом и при этом не меняют порядок поступления элементов. Очереди часто используются для обработки асинхронных событий, возникающих в случайные моменты времени. Так, если в очередь в произвольные моменты времени будут добавлены значения 5, 10 и 15, то на другой стороне, подиситема обработки событий, сможет вполне синхронно, по порядку, извлечь их в той последовательности, в которой они попали в очередь.

5.5. Списки в библиотеке STL

Язык C++ имеет готовую реализацию некоторых рассмотренных ранее динамических структур данных. Эта реализация содержится в библиотеке стандартных шаблонов (STL – Standard Template Library).

Данная библиотека включает в себя различные обобщенные классы контейнеров для хранения данных, средства доступа к ним и реализацию алгоритмов обработки данных.

Так STL содержит класс `list` реализующий двусвязный список с возможностью добавления/удаления элементов в начало/конец/середину списка. Алгоритмы методов класса `list` хорошо продуманы. Так вставка/удаление выполняются за время `O(1).`

С полным списком методов класса `list` можно ознакомиться в документации по C++ и STL. Рассмотрим лишь некоторые:

- `front()` — возвращает ссылку на первый элемент списка;
- `back()` — возвращает ссылку на последний элемент списка;
- `push_back(value)` — добавляет новый элемент в конец списка;
- `push_front(value)` — добавляет новый элемент в начало списка;
- `insert(it, value)` — вставляет новый элемент `value` перед итератором `it`;
- insert(i, n, value) — вставляет n копий элементов `value` перед `i`;
- `pop_front()` — удаляет первый элемент списка;
- `pop_back()` — удаляет последний списка;
- `erase(it)` — удаляет элемент, на который указывает итератор `it`;
- `erase(start, fin)` — удаляет все элементы между занчениями итераторов `start` и `fin`, но элемент `fin` не удаляется;
- `clear()` — удаляет все элементы списка;
- `size()` — возвращает количество элементов списка;
- `empty()` — возвращает истину, если список пуст.

5.6 Примерная реализация однонаправленного связного списка

Задание: дана структура «Студент» (фамилия; имя; возраст, курс). Информация хранится в бинарном файле. Реализовать следующие операции работы со списком:

- выполнить чтение данных из файла и добавление их в список;
- разработать метод добавления новых элементов;
- разработать метод удаления элементов;
- вывести содержимое списка на экран;
- сохранить содержимое списка в двоичный файл.

Исходный код программы:

```
//содержимое Файла student.h
// заголовок структуры Student
#pragma once

#include <fstream>

class Student
{
      public:
      char surname[20]; // Фамилия студента
      char Name[20]; // Имя студента
      int Age; //Возраст студенат
      int Course; // Курс, на котором учится
      //запись информации в файл
      //файловый поток для записи должен быть открыт
      bool writeToFile( std::ofstream *outFile );
      //чтения информации из файла
      //файловый поток для чтения должен быть открыт
      bool readFromFile(std::ifstream *inFile );
};


// содержимое Файла student.cpp
// содержит методы класса Student
```

```
#include "student.h"

using namespace std;

//метод класса Student writeToFile()
//файловый поток для записи должен быть открыт
bool Student::writeToFile(ofstream* outFile) {
      //проверка на корректное открытие файла
      if (!outFile) return false;
      //запись в двоичный поток
      outFile->write((char*)this, sizeof(Student));
      //проверка корректности записи данных
      if (outFile->good())
           return true;

      return false;
}

//метод класса Student readFromFile()
//файловый поток для чтения должен быть открыт
bool Student::readFromFile(ifstream* inFile) {
      //проверка на корректное открытие файла
      if (!inFile) return false;
           //проверка на пустое содержимое файла
      if (inFile->peek() == EOF) return false;
      //чтение из двоичного потока
      inFile->readFromFile((char*)this, sizeof(Student));
      //проверка корректности чтения данных
      if (inFile->good())
           return true;

      return false;
}

//содержимое Файла  node.h
//файл заголовока структуры Node

#pragma once
#include "student.h"

//Элемент односвязного линейного списка
struct Node {
 Student data;
```

```
 Node* next;
 //конструктор с одним параметром
 Node(Student* st) {
      //копируем данные полей из st в data
      strcpy(data.surname, st->surname);
      strcpy(data.Name, st->Name);
      data.Age = st->Age;
      data.Course = st->Course;

      next = nullptr;
 }
};

// содержимое Файла  linkedlist.h
// файл заголовока класса LinkedList

#pragma once

#include "student.h"
#include "node.h"

class LinkedList
{
      Node* Top; //Указатель на начало списка
    public:
      //Конструктор по умолчанию
      LinkedList() :Top(nullptr) {};//здесь Top = nullptr
      ~LinkedList();//Деструктор

    // добавление записи в начало списка
      void AddToTop(Student* st);

    // добавление записи в конец списка
      void AddToTail(Student* st);

      void PrintList(); // печать содержим. списка на экран

      void Erase(); // удаление всех элементов из списка

    // запись содержимого списка в двоичный файл
          bool LoadToFile(char* nmFile);
      // чтение содержимого списка из двоичного файла
      // старое содержимое списка уничтожается
```

```
          bool ReadFromFile(char* nmFile);

    // проверка на пустой список
    // реализовано в *.h файле
      bool isEmpty() {
           if (Top == nullptr) return true;
           //если список не пустой, возвращаем "ложь"
           return false;
      }
};

// содержимое Файла  linkedlist.cpp
// содержит методы класса LinkedList

#include <iostream>
#include <iomanip>
#include <fstream>
#include <string.h>

#include "linkedlist.h"

using namespace std;

//Деструктор класса LinkedList
// очищаем список перед разрушением
LinkedList::~LinkedList()
{
      Erase();
}

//метод удаления всех элементов из списка
void LinkedList::Erase()
{
 while (Top != nullptr) //Пока не достигли конца списка
 {
      Node* tmp = Top->next; // tmp будет указывать на
                             // адрес следующего элемента
      delete Top; //Освобождаем память
      Top = tmp; //адрес на начало списка указывает на
                 // следующий элемент
 }
 Top = nullptr;
}
```

```
// метод добавления нового элемента в начало списка
void LinkedList::AddToTop(Student* st)
{
//Выделение памяти под новый элемент
      Node* tmp = new Node(st);
//Указываем, что адрес следующего элемента, это
    // старое начало списка
      tmp->next = Top;

      Top = tmp; //меняем адрес на начало списка на
                  // вновь добавленный элемент
}

// метод добавления нового элемента в конец списка
void LinkedList::AddToTail(Student* st)
{
      //Выделение памяти под новый элемент
      Node* newNode = new Node(st);

      // если в списке нет элементов, то
      // новый элемент - это вершина списка
      if (Top == nullptr) {
           Top = newNode;
           return;
      }

      // если в списке есть элементы, то переходим
      // на последний элемент и добавляем
      Node* pt = Top;
      while (pt->next != nullptr) {
           pt = pt->next;
      }
      //теперь pt указывает на последний элемент списка
      pt->next = newNode; //добавляем
}

// метод для распечатки содержимого списка на экран
void LinkedList::PrintList()
{
//Объявляем указатель, он указывает на начало
      Node* tmp = Top;
      system("cls"); // очистка экрана
```

```
// печать заголовка таблицы
 cout << setw(20) << "Фамилия" << setw(20) << "Имя" <<
      setw(10) << "Возраст" << setw(10) << "Курс"<<endl;
 cout << "_____________________________________________\n"
      << endl;

 while (tmp != nullptr) {
      //Выводим на экран содержимое полей:

      cout << setw(20)<< tmp->data.surname; //фамилия
      cout << setw(20) << tmp->data.Name; //имя
      cout << setw(10) << tmp->data.Age; //возраст
      cout << setw(10) << tmp->data.Course << endl;//курс

      //Переходим к следующему элементу
      tmp = tmp->next;
 }
 cout <<
       "_____________________________________________\n\n";
}

// метод чтения содержимого списка из двоичного файла
// старое содержимое списка уничтожается
bool LinkedList::ReadFromFile(char * nmFile)
{
 //проверка имени файла
 if (nmFile == nullptr) return false;

 ifstream inFile; //файловый поток на чтение
 inFile.open(nmFile, ios::binary | ios::in);
 //Если файл не открыт
 if (!inFile) return false;

 // если файл открылся успешно
 Erase(); // удаляем существующие элементы

 //читаем информацию из файла и добавляем в конец списка
 Student tmp;
 bool ok = false;
 while (!inFile.eof())  {
      if (inFile.peek() != EOF) {
           ok = tmp.readFromFile(&inFile);
```

```
          if (!ok) {
               inFile.close(); // если чтение не успешно,
                               // закрываем файловый поток
               return false;
          }
          AddToTail( &tmp );
     }
 }

 inFile.close(); // в конце закрываем файловый поток
 return true;
}

// метод записи содержимого списка в двоичный файл
bool LinkedList::LoadToFile(char* nmFile)
{
 //проверка имени файла
 if (nmFile == nullptr) return false;

 ofstream outFile(nmFile, ios::binary | ios::out);
 //Если файл не открыт
 if (!outFile) return false;

//Объявляем указатель, он указывает на начало списка
 Node* tmp = Top;
 while (tmp != nullptr) //Пока не конец списка
  {
      tmp->data.writeToFile( &outFile );
      //переходим на следующий элемент
      tmp = tmp->next;
      if (!outFile.good()) { //в случае ошибки записи

 // закрываем файловый поток
          outFile.close();
          return false;
      }
 }
 outFile.close();// закрываем файловый поток
 return true;
}

//Содержимое файла lab2_5_1.cpp с функцией main()
#include <iostream>
```

```
#include <iomanip>
#include <fstream>
#include <ctime>
#include <string.h>

#include "student.h"
#include "node.h"
#include "linkedlist.h"

using namespace std;

//--- прототипы функций реализованных в этом файле:

//функция реализации пункта меню 1 "Чтение из файла"
void ReadListFromFile(LinkedList* list);

//функция реализации пункта меню 3 "Добавление студента"
void AddStudToList(LinkedList* list);

//функция реализации пункта меню 4: "Сохранение в файл"
void LoadListToFile(LinkedList* list);


int main(int argc, char *argv[])
{
 bool isContinue = true; // флаг продолжения работы
 int select; // значение выбора пользователя
 Student stud; // переменная класса Студент
 LinkedList list; // связный список

 cout.setf(ios::left); // задаем выравнивание по левому
краю
 while (isContinue)
 {
     system("cls"); // выполняем очистку экрана
                    // после каждой операции
     cout << " Меню" << endl;
     cout << "_____________________________________\n";
     cout << "1: Чтение из файла" << endl;
     cout << "2: Печать списка" << endl;
     cout << "3: Добавление студента" << endl;
     cout << "4: Сохранение в файл" << endl;
     cout << "5: Удаление последнего элемента" << endl;
```

```
      cout << "6: Поиск" << endl;
      cout << "7: Сортировка" << endl;
      cout << "8: Выход" << endl;
      cout << "_____________________________________\n" << endl;
      cout << "Выберите операцию (1-8): ";
      cin >> select;
      switch (select)
      {
      case 1: ReadListFromFile( &list); system("PAUSE");
            break;
      case 2: list.PrintList(); system("PAUSE"); break;
      case 3: AddStudToList( &list ); break;
      case 4: LoadListToFile( &list ); break;
      case 5: cout << "Не реализовано\n "; break;
      case 6: cout << "Не реализовано\n "; break;
      case 7: cout << "Не реализовано\n "; break;
      case 8: isContinue = false; break;
      default: cout << "Выберите число от 1 до 8\n" ;
      }
 }
 system("PAUSE");
 return 0;
}

//функция реализации пункта меню 1 "Чтение из файла"
void ReadListFromFile( LinkedList * list) {
      if (list == nullptr) {
      cout << "Отсутствует указатель на список\n";
      return;
 }

 char nmFile[30]; // имя файла

 // если нужно указать абсолютный путь к файлу:
 //ifstream inFile("C:\\PATH\\student.dat",
                              //ios::binary | ios::in);
 cout << "Введите имя файла с данными: ";
 cin >> nmFile;
 if (! list->ReadFromFile(nmFile) )
      cout << "Неверное имя файла\n";
 else
      cout << "Данные считаны\n";
```

```
}

//функция реализации пункта меню 3 "Добавление студента"
void AddStudToList(LinkedList* list) {
 if (list == nullptr) {
      cout << "Отсутствует указатель на список\n";
      return;
 }

 Student stud;
 system("cls"); // Очищаем экран перед вводом данных
 cin.ignore(); //если буфер потока ввода не пустой,
               // игнорируем символ
  //Ввод фамилии:
 cout << "Фамилия: "; cin.getline(stud.surname, 20);
 //Ввод имени:
 cout << "Имя: "; cin.getline(stud.Name, 20);
 cout << "Возраст: "; cin >> stud.Age; //Возраст
 cin.ignore(); //после ввода числа, игнорируем символ
 cout << "Курс: "; cin >> stud.Course; //курс
 cin.ignore(); //после ввода числа, игнорируем символ

 list->AddToTop(&stud);
}

//функция реализации пункта меню 4 "Сохранение в файл"
void LoadListToFile(LinkedList* list) {
 if (list == nullptr) {
      cout << "Отсутствует указатель на список\n";
      return;
 }

 char nmFile[30]; // имя файла
 system("cls"); // Очищаем экран
 cout << "Введите имя файла: ";
 cin >> nmFile;
 if ( list->LoadToFile( nmFile ) )
 {
      //Если запись в файл прошла успешно
      cout << "Файл создан, данные записаны\n";
 }
 else cout << "Запись не выполнена\n";
 system("pause");
```

```
}
```

5.6. Вопросы для самопроверки

1. Что такое динамическая структура данных?
2. Приведите классификацию динамических структур данных.
3. Перечислите основные функции обработки данных
4. Опишите структуру данных «список»
5. Опишите структуру данных «стек»
6. Опишите структуру данных «очередь»

5.7. Задания для самопроверки

Выполните задания из лабораторной работы №5 (приложение 1).

6. АЛГОРИТМЫ СОРТИРОВОК

6.1. Сортировка. Общие сведения

Сортировкой (англ. sorting) называется процесс упорядочивания множества объектов по какому-либо признаку.

Так как данные могут храниться в разных структурах, то и алгоритмы для каждой структуры могут отличаться. В данном пособии, сортироваться будут данные, находящиеся в массивах или других хранилищах и упорядочиваться они будут в зависимости от значения ключа. Ключом называется поле элемента (или сам элемент, если это, например, число), по которому производится сортировку.

Существует несколько вариантов классификации сортировок в зависимости от признака-классификатора.

1. Время работы

Эту классификацию обычно считают самой важной. Оценивают худшее время алгоритма, среднее и лучшее. Лучшее время - минимальное время работы алгоритма на каком-либо наборе, обычно этим набором является тривиальный [1...n]. Худшее время - наибольшее время. У большинства алгоритмов временные оценки бывают O(n logn) и O(n2).

2. Память

Параметр сортировки, показывающий, сколько дополнительной памяти требуется алгоритму. Сюда входят и дополнительный массив, и переменные, и затраты на стек вызовов. Обычно затраты бывают O(1), O(logn), O(n).

3. Устойчивость

Устойчивой сортировкой называется сортировка, не меняющая порядка объектов с одинаковыми ключами.

4. Количество обменов

Количество обменов может быть важным параметром в случае, если объекты имеют большой размер. В таком случае при большом количестве обменов время алгоритма заметно увеличивается.

5. Детерминированность

Алгоритм сортировки называется детерминированным, если каждая операция присваивания, обмена и т.д. не зависит от предыдущих. Все сортирующие сети являются детерминированными.

Некоторые виды сортировки представлены в таблице 6.1.

Таблица 6.1

Некоторые виды сортировок

Название	Описание алгоритма
Сортировка пузырьком (Bubble Sort)	Подход заключается в выполнении повторяющихся проходов по сортируемому списку (массиву). На каждом проходе цикла сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами.
Сортировка вставками (Insertion Sort)	На каждом шаге выбирается один из элементов сортируемомого списка (массива) и вставляется на нужную позицию в отсортированной части. Операция выполняется до тех пор, пока массив не будет отсортирован.
Сортировка Шелла (Shellsort)	Это модификация сортировки вставками, при которой сравниваются и меняются местами элементы, стоящие на местах, кратных специально подобранному набору шагов.
Сортировка выбором (Selection Sort)	На каждом шаге из элементов сортируемомого списка (массива) выбирается элемент с максимальным (минимальным) значением ключа и вставляется на нужную позицию в отсортированной части (например, в начале массива). Операция выполняется до тех пор, пока массив не будет отсортирован.
Быстрая сортировка (Quick Sort)	Выбирается опорный элемент, разделеляющий массива на две части (все элементы, меньшие опорного и элементы большие). Полученные части сортируются используя рекурсивные вызовы.
Сортировка слиянием (Merge Sort)	Выполняется в виде разделения массива (списка) пополам. Затем, происходит сортировка половин и их слияние.
Многопоточная сортировка слиянием (Multithreaded merge sort)	Отличается от сортировки слиянием только тем, что при рекурсивном вызове будет создавать новый поток.
PSRS-сортировка (PSRS-sorting)	Разделим массив на несколько подмассивов и запустим в каждой быструю сортировку. После этого объединим все эти подмассивы.

На практике, наибольший интерес представляют алгоритмы сортировки, которые не выполняют дублирование исходных данных.

Например, если нужно отсортировать массив **a**, можно создать массив **b**, имеющий такой же размер и переписывать отсортированные данные из **a** в **b**. Это приведет к двойному расходу памяти и, в случае, когда **a** достаточно большого размера, может вызвать проблемы. Таким образом, перестановка элементов должна выполняться внутри сортируемого хранилища данных.

Кроме того эффективность алгоритмов сортировки часто оценивается по числу необходимых сравнений элементов и числу перестановок (замен) элементов.

Эти параметры, по сути, являются функциями от количества сортируемых элементов `n`. В данном пособии рассматриваются несколько простых алгоритмов, называемых прямыми. Они проигрывают с точки зрения эффективности более сложным алгоримам: простым требуется порядка n^2 сравнений, тогда как сложым может быть достаточнодка `n*log(n)` сравнений. Однако, прямые методы удобны для объяснения характерных черт основных принципов большинства сортировок; исходный код программ этих методов является компактным и легок для понимания.

Кроме того, нужно иметь ввиду, что при малом количестве сортируемых элементов, разница в скорости обработки массивов может быть пренебрежимо мала, а время на разработку и отладку сложных алгоритмов может быть потрачено значительно больше.

Методы сортировки, не выполняющие дублирование данных можно разбить на три основные категории:

- сортировки вставками;
- сортировки выбора;
- сортировки обменом.

Далее в пособии все рассматриваемые алгоритмы будут сортировать массив целых чисел **m**, который объявляется следующим образом:

```
int m[n];
```

6.2. Сортировка вставками

При сортировке вставками, массив разделяется на две части относительно некоторого i–го элемента: отсортированную $m_1, \ldots, m_{i-1}$ и неотсортированную $m_i, \ldots, m_n$.

В начале работы алгоритмы в качестве отсортированной части массива принимается только один первый элемент, а в качестве неотсортированной части – все остальные элементы.

На каждом шаге, начиная с `i = 2` и увеличивая i каждый раз на единицу, из неотсортированной последовательности извлекается i-й элемент и вставляется в отсортированную так, чтобы не нарушить в ней упорядоченности элементов. Каждый шаг алгоритма включает четыре действия:

1. Берется i-й элемент массива (это первый элементом во второй, неотсортированной части), и сохраняется во вспомогательной переменной.
2. Находится позиция j в первой, отсортированной части массива, и в нее вставляется i-й элемент, который мы получили на шаге 1. Вставка должна быть выполнена так, чтобы не нарушать упорядоченность элементов отсортированной части массива.
3. Выполняется сдвиг элементов массива с i – 1 позиции по j-ю на один элемент вправо, чтобы освободить найденную позицию для вставки.
4. Выполняется вставка взятого элемента в найденную j-ю позицию.

На рисунке 6.1 приведены два первых шага работы алгоритма сортировки вставками. Отсортированная и неотсортированная части разделены чертой.

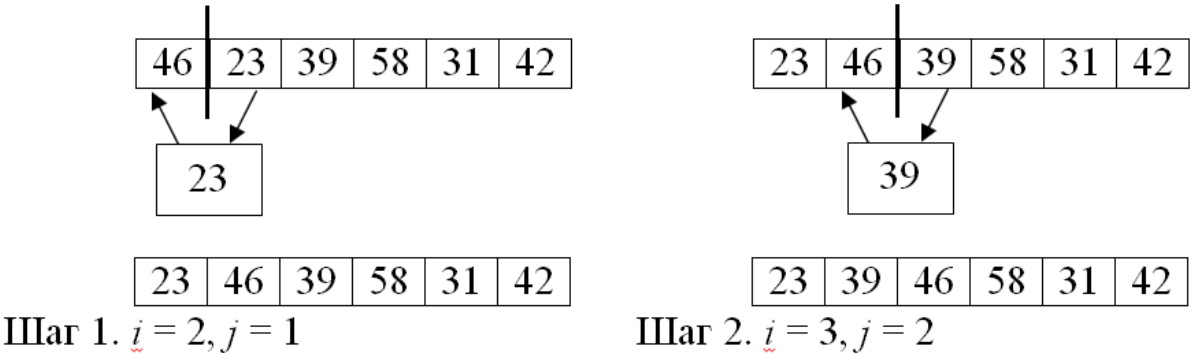


Рис. 6.1. Первые два шага алгоритма сортировки вставками

В таблице 6.2 показан пример сортировки вставками массива целых чисел из 6 элементов. Отсортированная часть массива подчеркнута.

Таблица 6.2

Пример сортировки вставками

	46	23	39	58	31	42
$i = 2$	23	46	39	58	31	42
$i = 3$	23	39	46	58	31	42
$i = 4$	23	39	46	58	31	42
$i = 5$	23	31	39	46	58	42
$i = 6$	23	31	39	42	46	58

Анализ алгоритма сортировки вставками

Число сравнений значений массива (C_i) при i-м проходе изменяется от $i - 1$ до 1. Если предположить, что все перестановки из n элементов равновероятны, то среднее число сравнений будет $i / 2$. Число же перестановок M_i равно $C_i + 2$.

Таким образом, сортировка вставками наиболее эффективна в случае, когда исходная часть массива уже упорядоченна, и показывает наихудшие результаты, когда элементы изначально расположены в обратном порядке.

Рассматриваемый алгоритм можно легко улучшить, если иметь ввиду, что готовая последовательность, в которую надо вставить новый элемент, уже отсортирована. Тогда для нахождения точки включения можно использовать бинарный поиск (алгоритм бинарного поиска рассмотрен в разделе 7.4 данного пособия). Такой модифицированный алгоритм сортировки называется *методом с двоичным включением*.

Однако, полученные улучшения (в случае применения бинарного поиска), влияют только на число сравнений, а не на число необходимых замен. Следует учитывать, что перестановка элементов является более ресурсоемкой операцией, чем сравнение и занимает больше времени, а количество замен так и остается порядка n^2.

6.3. Сортировка выбором

При сортировке выбором массив логически делится на две части: отсортированную m_1, …, m_{i-1} и неотсортированную – m_i, …, m_n относительно некоторого `i`–го элемента.

Общий принцип работы этого алгоритма достаточно прост: выполняется поиск наименьшего (наибольшего) элемента из неотсортированной части массива, и он помещается в конец готовой, отсортированной части.

В самом начале сортировки, первая часть считается пустой, а неотсортированная часть, соответственно, «включает» в себя все элементы массива.

Алгоритм сортировки выбором состоит из следующих шагов:

– Из всего массива выбирается элемент с наименьшим значением.
– Он меняется местами с первым элементом m_1.

– Затем этот процесс повторяется с оставшимися n – 1 элементами, n – 2 элементами и т. д. до тех пор, пока не останется один элемент с наибольшим значением.

Рассмотрим несколько первых шагов алгоритма сортировки с помощью прямого выбора на примере упорядочивания по возрастанию следующего массива:

46 23 39 58 31 42

На первом шаге отсортированная часть массива не содержит ни одного элемента. Мы находим наименьший элемент - сортировка будет по возрастанию. Это второй элемент массива: 23 и меняем его местами с первым элементом: 46 .

Теперь отсортированная часть включает в себя один элемент – 23. На рисунке 6.2, для текущего состояние массива, отсортированная часть подчеркнута сплошной линией.

Далее, находим следующий наименьший элемент в оставшейся части массива. Это будет элемент 31, его нужно поместить на место второго элемента отсортированной части. Эти элементы выделены на рисунке 6.2.

<u>23</u> **46** 39 58 **31** 42

Рис. 6.2. Шаг 1 алгоритма сортировки выбором

На втором шаге готовая последовательность состоит уже из двух элементов: 23 и 31. Наименьший элемент исходной последовательности – 39. Он является третьим элементом массива и именно эту позицию он должен занять. Поэтому на рисунке 6.3 полужирным шрифтом выделен только один данный элемент.

23 31 **39** 58 46 42

Рис. 6.3. Шаг 2 алгоритма сортировки выбором

В таблице 6.3 наглядно показаны все шаги алгоритма сортировки с помощью прямого выбора: отмечены готовые последовательности и переставляемые элементы.

Таблица 6.3

Пример сортировки выбором

	46	**23**	39	58	31	42
$i = 2$	23	**46**	39	58	**31**	42
$i = 3$	23	31	**39**	58	46	42
$i = 4$	23	31	39	**58**	46	**42**
$i = 5$	23	31	39	42	**46**	58
$i = 6$	23	31	39	42	46	58

Анализ алгоритма сортировки выбором

При работе данного алгоритма число сравнений элементов не зависит от начального порядка элементов. Количество сравнений в общем случае можно рассчитать по формуле:

$$C = (n^2 - n) / 2.$$

Если массив изначально был упорядоченным, число перестановок будет минимально и расчитывается по формуле:

$$M_{min} = 3 * (n - 1).$$

В наихудшем случае, когда первоначально элементы располагались в обратном порядке, число перестановок будет равно:

$$M_{max} = n^2 / 4 + 3 * (n - 1).$$

На практике, алгоритм сортировки выбором, обычно, предпочтительнее алгоритма сортировки вставками. Однако, если элементы исходного массива изначально упорядочены (или почти упорядочены), алгоритм вставками выполнит сортировку быстрее.

6.4. Сортировка методом «пузырька»

При данном алгоритме сортировки, выполняется последовательность проходов по исходному массиву. Во время каждого прохода наименьший (или наибольший) элемент сдвигается к началу (или к концу) массива и становится «на свое место», то есть становится отсортированным. В отличие от рассмотренных ранее алгоримов при сортировке методом «пузырька», сравниваются и меняются местами только соседние два элемента.

Если представить, что элементы массива расположены вертикально, то за первый проход элемент с наименьшим значением, т. е. самый «легкий», поднимется на самый верх массива и станет его первым элементом, подобно пузырьку воздуха в жидкости. В результате следующего прохода второй по «легкости» элемент поднимется к началу массива и станет вторым элементом. Отсюда и название данного метода – метод «пузырька» или «пузырьковая сортировка».

Алгоритм сортировки методом «пузырька» работает следующим образом. Рассмотрим k-ый проход ($1 \le k \le n$). На нем элементы массива с n-го по k-й включительно перебираются в обратном порядке и сравниваются соседние элементы. Если сортировка выполняется по возрастанию, а элемент $m[i]$ оказывается меньше предыдущего ($m[i - 1]$), то они меняются местами.

В таблице 6.4 представлены шаги сортировки методом «пузырька» по возрастанию. Элементы массива расположены вертикально. Движение элементов к началу массива показано стрелками.

Таблица 6.4

Пример пузырьковой сортировки

k =1	*k=2*	*k=3*	*k=4*	*k=5*	*k=6*	*k=7*	*k=8*
21	4	4	4	4	4	4	4
25	21	9	9	9	9	9	9
9	25	21	12	12	12	12	12
17	9	25	21	17	17	17	17
43	17	12	25	21	21	21	21
12	43	17	17	25	25	25	25
4	12	43	32	32	32	32	32
32	32	32	43	43	43	43	43

Как видно из таблицы 6.4, три в трех последних проходах перемещения элементов отсутствуют, так как они уже отсортированы. Следовательно, один из способов увеличения эффективности этого алгоритма – отслеживать, были ли перестановки во время текущего прохода. Если в последнем проходе ни одного перемещения элементов не было, то значит, массив отсортирован и работу алгоритма можно завершать.

С другой стороны, после первого прохода отсортированным становится самый первый элемент, после второго – второй и так далее. Очевидно, что все пары соседних элементов, находящиеся выше индекса равному номеру прохода *k*, уже упорядочены. Поэтому каждый раз очередной проход можно заканчивать на этом, *k*-ом индексе.

6.5. Шейкерная сортировка

Пример сортировки методом «пузырька», представленный в табл. 6.4, показывает некоторый дисбаланс работы алгоритма: легкий пузырек «всплывает» быстро, а тяжелый «тонет» медленно. Легкий пузырек «всплывает» за один проход цикла, а тяжелый «тонет» очень медленно - за один проход он перемещается на одну позицию. Например, массив

15 29 31 55 70 93 8

с помощью пузырьковой сортировки будет упорядочен за один проход, а для сортировки массива

93 8 15 29 31 55 70

потребуется шесть проходов. В этом случае можно попробовать чередование направления обработки массива на каждом последующем проходе.

Такой подход реализует, так называемая «шейкерная сортировка». Таблица 6.5 иллюстрирует данный алгоритм, примененный к тем же данным, что и в таблице 6.4. Переменные L и R содержат индексы элементов, до которых должен происходить просмотр при движении влево (или вверх) и вправо (или вниз) соответственно.

Таблица 6.5

Пример шейкерной сортировки

L =	1	2	2	3	3
R =	8	8	7	7	6
направление	↑	↓	↑	↓	↑
	21	4	4	4	4
	25	21	21	9	9
	9	25	9	21	12
	17	9	17	12	17
	43	17	25	17	21
	12	43	12	25	25
	4	12	32	32	32
	32	32	43	43	43

Если дополнительно ввести переменную, отслеживающую были ли перестановки элементов, то данный алгоритм отсортирует массив за пять проходов, а для сортировки «пузырьком» (без улучшений) потребуется семь проходов.

Если же при работе алгоритма шейкерной сортировки запоминать индекс последнего переставленного элемента, например в переменной h, то число проходов сократится до четырех, как показано в таблице 6.5.

Анализ шейкерной сортировки и сортировки «пузырьком»

Число сравнений в алгоритме сортировки обменом и методом «пузырька» равно

$$C = (n^2 - n) / 2.$$

Минимальное и максимальное число перестановок элементов равно

$$M_{min} = 0,\ M_{max} = 3 * (n^2 - n) / 4.$$

Если же рассматривать улучшенные методы, особенно шейкерную сортировку, можно отметить, что анализ их эффективности достаточно сложен. Заметим, что все перечисленные выше попытки оптимизации алгоритмов не влияют на число перемещений, они только сокращают число проверок, а как уже говорилось выше, более ресурсоемкой операцией является именно перемещение элементов. Поэтому очевидные на первый взгляд улучшения существенно не увеличивают производительность алгоритмов. Шейкерную сортировку лучше использовать в тех случаях, когда известно, что элементы исходного массива (списка, хранилища данных) почти упорядочены. Анализ показывает, что «обменная» сортировка и различные ее усовершенствования, оказываются хуже сортировок вставками и выбором.

6.6. Сортировка Шелла

Рассмотренные ранее алгоритмы сортировки передвигают нужный элемент на определенном шаге на одну позицию. Из этого следует, что общее количество таких шагов должно быть n^2. Одним из способов увеличения эффективности алгоритма может быть положен принцип перемещения элементов на большее расстояние.

В 1959 году Д. Шелл было предложил усовершенствованный алгоритм сортировки вставками. Для знакомства с его работой, посмотрим на следующий пример:

35 28 49 79 45 11 89 70 91 67 54 19 13 24

Принцип алгоритма следующий: зададим некоторый шаг между проверяемыми элементами. Сначала примем его равным 4 и будем сравнивать их друг с другом. На рисунке 6.4 (а-г) выбранные элементы показаны дугами.

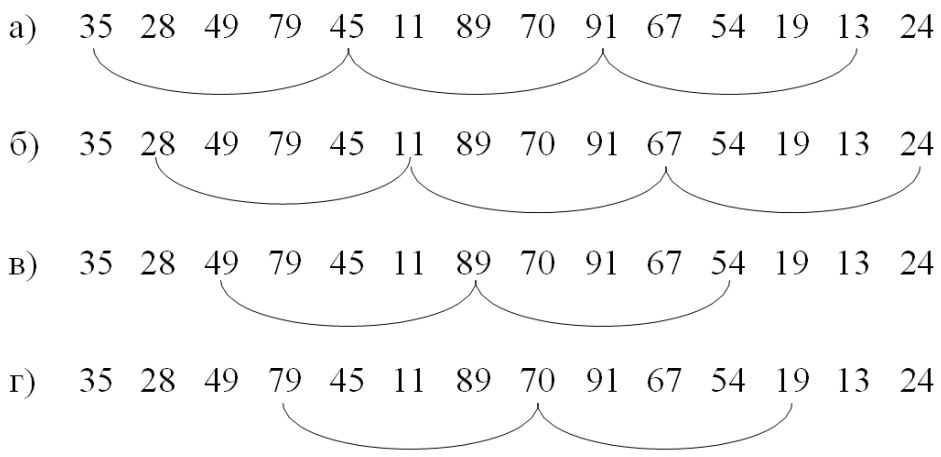


Рис. 6.4. Выделение элементов массива,
отстоящих друг от друга на 4 шага

Если порядок в паре выбранных элементов не соответствует порядку сортировки, они меняются местами (выполняется сортировка). Таким образом, сначала сортируются элементы 35, 45, 91, 13, затем 28, 11, 67, 24, следом 49, 89, 54, и, наконец, 79, 70, 19. В результате получаем массив:

13 11 49 19 35 24 54 70 45 28 89 79 91 67

На следующем этапе мы уменьшаем шаг в два раза и сравниваем элементы, отстоящие друг от друга на две позиции. Таких групп две, они показаны на рисунке 6.5 (а – б).

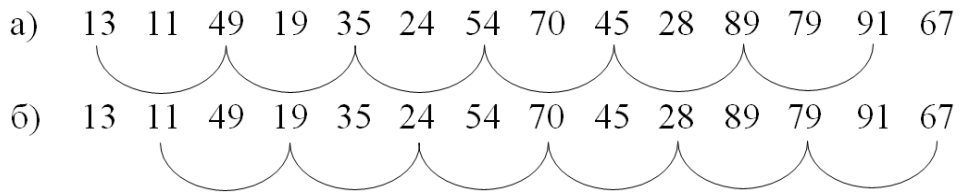


Рис. 6.5. Выбор элементов,
отстоящих друг от друга на два

Вновь среди выбранных элементов выполняется сортировка. В результате получим массив:

13 11 35 19 45 24 49 28 54 67 89 70 91 79

И наконец, на третьем шаге сравниваются все элементы подряд (рисунок 6.6).

Рис. 6.6. Третий шаг сортировки

Пройдя все шаги, получаем отсортированный массив:

11 13 19 24 28 35 45 49 54 67 70 79 89 91

На первый взгляд можно засомневаться: если необходимо несколько процессов сортировки, причем и каждый включаются все элементы, то не добавят ли они больше работы, чем сэкономят? Однако на каждом этапе либо сортируется относительно мало элементов, либо элементы уже довольно хорошо упорядочены и требуется сравнительно немного перестановок.

Очевидно, что такой метод в результате дает упорядоченный массив. Видно также, что каждый проход от предыдущих только выигрывает (т. к. каждая i-сортировка объединяет две группы, уже отсортированные $2i$-сортировкой).

Расстояния в группах можно уменьшать по-разному, лишь бы последнее было единичным, ведь в самом плохом случае последний проход и сделает всю работу. Однако совсем не очевидно, что такой прием «уменьшающихся расстояний» может дать лучшие результаты, если расстояния не будут степенями двойки. При выполнении лабораторных работ рекомендуется начальный шаг взять равным $h_{нач} = n / 3$, где n – количество элементов массива, и уменьшать его на каждом проходе вдвое:

$$h_{i-1} = h_i / 2.$$

Как отмечалось выше $h_{конеч} = 1$.

Анализ сортировки Шелла

Анализ этого алгоритма поставил несколько весьма трудных математических проблем, многие из которых так еще и не решены. В частности, не известно, какие расстояния дают наилучшие результаты. Известно, однако, что выбор расстояний должен быть таким, чтобы взаимодействие различных цепочек проходило как можно чаще.

В работе Кнут [3] показал, что имеет смысл использовать следующую последовательность (она записана в обратном порядке): 1, 4, 13, 40, 121, ..., где $h_{k-1} = 3h_k + 1$, h_t (или $h_{конеч}$) = 1 и $t = log_3(n) - 1$. Он рекомендует и другую последовательность: 1, 3, 7, 15, 31, ... где $h_{k-1} = 2h_k + 1$, $h_t = 1$ и $t = log_2(n) - 1$. Математический анализ показывает, что в последнем случае для сортировки n элементов методом Шелла затраты пропорциональны $n^{1.2}$, что значительно лучше n^2, необходимых для методов прямой сортировки.

6.7. Сравнение различных алгоритмов сортировки

Проанализируем эффективность различных алгоритмов сортировки массивов, состоящих из n сортируемых элементов. Анализ будет проводиться по двум критериям: числу необходимых сравнений

элементов *C* и числу перестановок элементов *M*. Для всех прямых методов сортировки можно дать точные аналитические формулы. Они приведены в табл. 6.6.

Таблица 6.6

Сравнение прямых методов сортировки

Метод		Минимальное	Среднее	Максимальное
Прямое включение	C =	$n - 1$	$(n^2 + n - 2) / 4$	$(n^2 - n) / 2 - 1$
	M =	$2(n - 1)$	$(n^2 - 9n - 10) / 4$	$(n^2 - 3n - 4) / 2$
Прямой выбор	C =	$(n^2 - n) / 2$	$(n^2 - n) / 2$	$(n^2 - n) / 2$
	M =	$3(n - 1)$	$n * (\ln n + 0.57)$	$n^2 / 4 + 3(n - 1)$
Прямой обмен	C =	$(n^2 - n) / 2$	$(n^2 - n) / 2$	$(n^2 - n) / 2$
	M =	0	$(n^2 - n) * 0.75$	$(n^2 - n) * 1.5$

В таблице 6.7 приведены фактические значения показателей *C* и *M* для массива с количеством элементов равным тысяче.

Таблица 6.7

Значения показателей *C* и *M* при *n* = 1000

Метод		Минимальное	Среднее	Максимальное
Прямое включение	C =	999	250250	499499
	M =	1998	247748	498498
Прямой выбор	C =	499500	499500	499500
	M =	2997	7478	252997
Прямой обмен	C =	499500	499500	499500
	M =	0	749250	1498500

Для сортировки Шелла, как и для других усовершенствованных методов, простых и точных формул не существует. Однако, в случае сортировки Шелла вычислительные затраты составляют в среднем $n^{1.2}$, в то время как для прямых методов затраты составили бы n^2.

Дополнительную информацию об эффективности простых методов сортировки, можно посмотреть в [6,7].

6.8. Вопросы для самопроверки

1. Дайте определение сортировки

2. Классификации сортировок

3. Чем можно объяснить многообразие алгоритмов сортировок? Почему не существует универсального алгоритма сортировки?

4. Графическая схема рассмотренных алгоритмов сортировки

5. Сравнение сложности сортировок

6.9. Задания для самопроверки

Выполните задания из лабораторной работы №6 (приложение 1).

7. АЛГОРИТМЫ ПОИСКА

7.1. Понятие поиска

Поиск – процесс нахождения заданной информации в имеющемся множестве данных или определение отсутствия совпадений. Как правило, данные в этом множестве являются конкретными записями, каждая из которых содержит в своем составе простой или составной ключ. Ключ поиска – это поле или набор полей записи, по значению которого осуществляется поиск, т.е. совпадение с ключом. Ключи записи во множестве данных используются для отличия одних записей от других. Целью поиска является нахождение всех записей (если они есть) с данным значением ключа или определение отсутствия совпадений с ключом в имеющемся множестве данных.

При обработке информации поиск данных является одним из наиболее часто выполняемых действий. Так как к решению данной проблемы разработано несколько подходов, то существует множество соответствующих алгоритмов.

Вводится ряд исходных допущений, не умаляющих общности рассматриваемых видов поиска.

1. Считается, что множество данных, в которых необходимо отыскать заданный элемент, является фиксированой и ее состав не изменяется в процессе выполнения алгоритма.
2. Пусть исходное множество из N элементов представляет собой некоторый массив $array$.
3. В этом случае задача поиска заключается определении элемента, равного заданному ключу x. Полученный в результате выполнения алгоритма индекс i искомого элемента должен удовлетворять условию:

$$array[i] = x$$

или должно быть выведено сообщение об отсутствии результата поиска.

7.2 Поиск перебором

В самом общем случае при отсутствии дополнительной информации об исходном множестве и разыскиваемых данных очевидным подходом является простой пошаговый последовательный просмотр массива и сравнение каждого очередного его элемента с ключом поиска. Этот метод называется поиском перебором или линейным поиском. Алгоритм завершается при выполнении одного из следующих условий:

1. Искомый элемент найден, т. е. $array[i] = x$.
2. Исходный массив просмотрен и совпадений с ключом не обнаружено.

Реализация алгоритма на языке C++ имеет следующий вид:

```
i = 0;
while ((i < N) && (array[i] != x))
{
   i++;
}
```

Порядок элементов в заголовке цикла имеет существенное значение. Перед каждым увеличением счетчика *i* выполняется условие:

$(0 \le i < N)$ AND $(\forall k: 0 \le k < i: array[k] \ne x)$.

Таким образом, для всех значений индекса *k*, меньших чем *i*, совпадения не обнаружены. Отсюда и из того факта, что поиск заканчивается только в случае ложности условия в заголовке цикла, можно вывести окончательное условие:

$((i = N)$ OR $(array[i] = x))$ AND $(\forall k: 0 \le k < i: array[k] \ne x)$.

Из полученного условия следует, что если искомый элемент найден, то это первый из таких элементов, т.е. желаемый результат найден вместе с минимально возможным индексом. При этом равенство значений $i = N$ свидетельствует, что в заданном множестве совпадение с ключом не обнаружено.

В рассматриваемом алгоритме гарантировано окончание цикла, поскольку значение *i* увеличивается на каждом шаге, и за конечное число шагов индекс достигнет предела *N*. Если совпадение с ключом в исходном множестве отсутствует, то это будет обнаружено по достижению конца массива ровно через *N* шагов.

7.3 Поиск с барьером

При реализации алгоритма поиска перебором на каждом шаге происходило увеличение индекса, а также производилось вычисление некоторого логического условия. Для ускорения и упрощения процедуры можно упростить само логическое условие, которое состоит из двух составляющих. Необходимо сформулировать более простое выражение, которое будет эквивалентно рассмотренному сложному условию. Это можно реализовать в случае, если существует гарантия, что совпадение рано или поздно произойдет.

Для этого достаточно в конец массива поместить дополнительный элемент со значением x. Такой вспомогательный элемент называется

барьером, поскольку не позволяет выйти за пределы массива. Сам массив может быть определен, например, следующим образом:

```
int array[N + 1];
```

Алгоритм поиска с барьером будет иметь следующий вид:

```
array[N] = x; i = 0;
while (array[i] != x)
{
     i++;
}
```

Результирующее условие, выведенное из того же инварианта, что и при поиске перебором:

(array[i] = x) AND (∀k: 0 ≤ k < i: array[k] ≠ x).

Как и в случае с линейным поиском равенство $i = N$ свидетельствует о том, что совпадение с ключом отсутствует (если не считать совпадения с барьером).

7.4 Бинарный поиск

Поиск перебором, в общем случае, улучшить практически невозможно, если отсутствует дополнительная информация о множестве данных, среди которых осуществляется поиск. Однако алгоритм поиска можно сделать значительно более эффективным, если данные будут отсортированы. Например, в журнале студентов, в котором фамилии упорядочены по алфавиту, искать нужного студента значительно легче.

Рассмотрим алгоритм, учитывающий то, что массив *array* упорядочен, т.е. удовлетворяет условию:

∀k: 1 ≤ k < N: array[k – 1] ≤ array[k].

Основная идея этого подхода – выбрать некоторый элемент, предположим, *array*[*m*], и сравнить его с ключом поиска *x*. Если *array*[*m*] равен ключу *x*, то нужный результат достигнут и процедура

поиска заканчивается. Если *array*[*m*] меньше *x*, то делается вывод, что все элементы отсортированного массива с индексами, меньшими или равными *m* можно исключить из дальнейшего поиска. Если же *array*[*m*] больше *x*, то исключаются элементы массива с индексами большими *m*. Эти рассуждения приводят к алгоритму, имеющему название бинарный или двоичный поиск.

Для реализации алгоритма бинарного поиска создадим две индексные переменные *Left* и *Right*, которые содержат соответственно левый и правый концы секции массива *array*:

```
Left = 0;
Right = N – 1;
Found = false;
while (Left <= Right)
{
    m = <некоторое значение между Left и Right >;
    if (array[m] == x)
       return array[m];
    else if (array[m] < x)
            Left = m + 1;
      else
         Right = m – 1;
}
```

Инвариант цикла, т. е. условие, выполняющееся перед каждым шагом, таков:

(Left ≤ Right) AND (∀k: 0 ≤ k < Left: array[k] < x) AND (∀k: Right ≤ k < N: array[k] > x),

из чего выводится результат:

((Left > Right) AND (∀k: 0 ≤ k < Left: array[k] < x) AND (∀k: Right ≤ k < N: array[k] > x)),

откуда следует:

(array[m] = x) OR (∀k: 0 ≤ k < i: array[k] ≠ x).

Корректность алгоритма не зависит от выбора значения m . Однако, принцип выбора значения m, может существенно повлиять на его эффективность. При этом основная задача - это исключение как можно большего числа элементов массива на каждом шаге. Наиболее распространенным на практике решением является выбор среднего элемента исходного массива, так как сразу исключается половина проверяемого диапазона. В результате максимальное число проведенных сравнений значений элементов массива с ключом будет равно $\log_2 N$, округленному до ближайшего большего целого. Таким образом, рассмотренный алгоритм существенно выигрывает по сравнению с поиском перебором, ожидаемое число сравнений которого равно $N / 2$.

Другим подходом в выборе m может стать использование интерполяционного поиска, когда вместо простого деления массива пополам, мы расчитываем более точное положение ключа искомого элемента. Следуя аналогии с телефонным справочником, если необходимо найти абонента с фамилией, начинающейся на букву «Б», необходимо сразу открывать книгу ближе к началу, а не посередине.

В этом случае, значение m вычисляется по формуле:

```
m = Left + ( x- array[Left] )*( Right-Left )/
      (array[Right] - array[Left] )
```

При интерполяционном поиске в массиве с произвольными ключами для каждого поиска (попадания или промаха) используется менее $\log_2(\log_2 N)$ сравнений.

Также, эффективность рассматриваемого алгоритма поиска можно улучшить, поменяв местами условные операторы. Проверку на равенство можно выполнять во вторую очередь, так как она удовлетворяется лишь единожды и приводит к окончанию работы.

Но более существенно следующее соображение: как и при поиске перебором, возможно найти такое решение, которое бы упростило

условие завершения алгоритма. Быстрый алгоритм основан на рассмотрении следующего условия:

$$(\forall k: 0 \le k < Left: array[k] < x) \text{ AND}$$
$$(\forall k: Right \le k < N: array[k] \ge x),$$

причем поиск продолжается до тех пор, пока обе секции не охватят массив целиком.

```
Left = 0;
Right = N – 1;
while (Left < Right)
{
      m = (Left + Right) / 2;
      if(array[m] == x ) return array[m];
      if (array[m] < x)
        Left = m + 1;
      else
        Right = m;
}
```

Однако, всегда ли достижимо условие окончания Left ≥ Right? Для его выполнения необходимо, чтобы на каждом шаге разность Right – Left убывала. В начале каждого шага Left < Right. Для среднего арифметического m справедливо условие Left ≤ m < Right. Следовательно, разность действительно убывает, ведь либо Left увеличивается при присвоении ему значения m + 1, либо Right уменьшается при присвоении ему значения m. При Left = Right повторение цикла заканчивается.

Однако и инвариант, и условие Left = Right еще не свидетельствуют о совпадении. Конечно, при Left = Right никаких совпадений нет. В других же случаях необходимо учитывать, что элемент array[Right] никогда в сравнении не участвует. Следовательно, необходима дополнительная проверка на равенство array[Right] = x. В отличие от начальной реализации рассматриваемого алгоритма (с логической переменной Found), приведенный алгоритм, как и в случае поис-

ка перебором, находит совпадающий элемент с наименьшим индексом.

Другие методы поиска элемента в списке/массиве представлены в [6,7].

7.5. Поиск подстроки в строке

Среди алгоритмов поиска отдельно выделяют специфический строковый поиск – поиск подстроки. Постановка задачи формулируется следующим образом. Задан исходный массив символов *str* из *N* элементов-символов и массив *sub* из *M* элементов- символов, причем выполняется условие $0 \leq M < N$. Исходные массивы символов задаются следующим образом:

```
char str[N];
char sub[M];
```

Задача поиск подстроки заключается в нахождении первого вхождения *sub* в *str*. Оба массива содержат некоторые наборы символов, таким образом, *str* можно считать некоторым текстом или строкой, а *sub* – подстрокой или образом. Необходимо найти первое вхождение подстроки *sub* в строку *str* или определить отсутствие результата поиска. Такая задача является типичной и актуальной для систем обработки текстов, отсюда и возникает необходимость разработки эффективного алгоритма для ее решения.

7.5.1. Прямой поиск подстроки в строке

Сначала разберем прямолинейный алгоритм поиска подстроки, который называется прямым поиском подстроки и не является эффективным для решения данной задачи. При реализации алгоритма прямого поиске производится сравнение первого символа подстроки *sub* и соответствующего ему символа из строки *str*. В начале работы алгоритма этим символом будет первый символ строки *str*. Если в результате сравнения оказывается, что эти символы совпадают, то рассмат-

ривается продолжение – вторые символы подстроки *sub* и строки *str*. При реализации алгоритма производится перемещение по подстроке *sub* от начала к концу. Если проход по подстроке *sub* завершен и все символы подстроки *sub* и символы части строки *str* совпали, то делается вывод, что поиск подстроки в строке выполнен успешно и совпадение найдено.

Если на некотором шаге сравнения символов подстроки и строки произошло несовпадение, то подстрока сдвигается по строке на один символ вправо, и снова производится сравнение подстроки сначала. Теперь осуществляется сравнение первого символа подстроки *sub* со вторым символом строки *str*, затем второго символа подстроки *sub* – с третьим символом строки *str* и т. д. Алгоритм поиск завершается, если будет найдено вхождение подстроки *sub* в строке *str* или же не закончится строка, что значит, что поиск не дал результата и вхождение не найдено.

Для реализации рассмотренного алгоритма и при разработке программы необходимо использовать два цикла. Внешний цикл отвечает за перемещение подстроки по строке, во внутреннем цикле происходит перемещение по самой подстроке. Фрагмент программы на языке C++ имеет вид:

```
i = -1; // счетчик до входа в цикл
do {
   i++; // перемещение по строке
   j = 0; // начало подстроки
// условия совпадания символов
   while((str[i + j] == sub[j]) && (j < m))
   {
      j++; // перемещение по подстроке
   }
}
// условия окончания поиска
while((j != m) && (i < n - m));
```

На практике значение (i < n – m) в условии окончания цикла определяет, не закончилась ли строка *str* и в случае отрицательного ответа показывает, что в строке *str* совпадения с подстрокой *sub* не выявлено. Невыполнение условия (j != m) соответствует ситуации, когда результат поиска положительный и найдено соответствие подстроки *sub* некоторой части строки *str*.

Рассмотренный алгоритм выполняется достаточно эффективно, если считать, что несовпадение пары символов подстроки и строки происходит после небольшого количества сравнений во внутреннем цикле. Например, можно предполагать, что для массивов символов, составленных из 128 печатных символов, несовпадение будет обнаруживаться после одной или двух проверок. Однако в неблагоприятном случае производительность алгоритма прямого поиска подстроки в строке является невысокой. Например, для строки, состоящей из $N - 1$ символов A и единственного символа B в конце, и подстроки, содержащей $M - 1$ символ A и опять B, чтобы обнаружить совпадение в конце строки, требуется произвести порядка $N \times M$ сравнений. Разработаны различные алгоритмы, которые имеют большую эффективность при решении задач рассматриваемого класса

7.5.2 Алгоритм Кнута, Морриса и Пратта

Алгоритм Кнута-Морриса-Пратта используется для поиска подстроки в строке. При использовании последовательного сравнения символов подстроки с очередными символами строки временная сложность алгоритма равна O(длина _подстроки × длина_строки). В 1977 г. Д. Кнут, В. Пратт и Д. Моррис разработали алгоритм поиска подстроки в строке, который имеет линейную временную сложность O(длина _подстроки + длина_строки). В честь разработчиков его называют алгоритм КМП или просто АКМП.

Пусть имеется подстрока *T* и строка *S*. Нужно найти номер символа, начиная с которого подстрока *T* входит в строку *S*. При отсут-

ствии вхождения подстроки в строку должно быть возвращено значение, которое не может быть интерпретировано как номер символа (например, 0).

Основой алгоритма Кнута-Морриса-Пратта является сдвиг после частичного совпадения некоторой начальной части подстроки с аналогичными символами строки. В этом случае пройденная часть строки считается известной, на основе подстроки возможно определить дополнительную информацию для поиска, которая позволит продвинуться по строке дальше, чем при прямом поиске.

АКМП при реализации сдвига подстроки использует префиксную функцию – числовую таблицу, которая формируется еще до начала поиска подстроки в строке. Можно выделить два этапа работы КМП-алгоритма:

1. формирование числовой таблицы, используемой при сдвиге подстроки по строке;

3. поиск подстроки в строке.

Таблица формируется на основе подстроки и содержит значения, которые в дальнейшем будут использованы при вычислении величины сдвига подстроки. Размер данной таблицы равен длине подстроки, которую можно определить при помощи функции int strlen(char *) библиотеки `<string.h>`. Таким образом, числовая таблица фактически является одномерным массивом, состоящим из числа элементов равного количеству символов в подстроке.

За формирование числовой таблицы отвечает префиксная функция. Префикс-функция строки $\pi(S,i)$ – это длина наибольшего префикса строки $S[1..i]$, который не совпадает с этой строкой и одновременно является ее суффиксом. Проще говоря, это длина наиболее длинного начала строки, являющегося также и ее концом. Для строки S удобно представлять префикс функцию в виде вектора длиной $|S|$-1. Можно рассматривать префикс-функцию длины $|S|$, положив $\pi(S,1)=0$.

Пример префикс функции для строки «abcdabcabcdabcdab» представлен в таблице 7.1.

Таблица 7.1

Пример префикс функции

$S[i]$	a	b	c	d	a	b	c	a	b	c	d	a	b	c	d	a	b
$\pi(S,i)$	0	0	0	0	1	2	3	1	2	3	4	5	6	7	4	5	6

Предположим, что $\pi(S,i)=k$. Отметим следующие свойства префикс-функции.

2. Если $S[i+1]=S[k+1]$, то $\pi(S,i+1)=k+1$.
3. $S[1..\pi(S,k)]$ является суффиксом строки $S[1..i]$. Действительно, если строка $S[1..i]$ оканчивается строкой $S[1\ldots\ \pi(S,i)]=S[1..k]$, а строка $S[1..k]$ оканчивается строкой $S[1..\pi(S,k)]$, то и строка $S[1..i]$ оканчивается строкой $S[1..\pi(S,k)]$.
4. $\forall\ j\in(k,i)$, $S[1..j]$ не является суффиксом строки $S[1..i]$. В противном случае было бы неверным предположение $\pi(S,i)=k$, так как $j>k$.

Рассмотренные свойства позволяют получить алгоритм вычисления префикс-функции:

```
Пусть π(S,i)=k. Необходимо вычислить π(S,i+1).
Если S[i+1]=S[k+1], то π(S,i+1)=k+1.
Иначе, если k=0, то π(S,i+1)=0.
Иначе положить k:=π(S,k) и перейти к шагу 1.
```

Ключевым моментом для понимания сути алгоритма является тот факт, что если найденный на предыдущем шаге суффикс не может быть расширен на следующую позицию, то мы пытаемся рассматривать меньшие суффиксы до тех пор, пока это возможно.

Покажем, что время работы алгоритма составляет O(n), где n=|S|. Заметим, что асимптотику алгоритма определяет итоговое количество итераций цикла `while`. Это так, поскольку без учета цикла `while` каждая итерация цикла for выполняется за время, не превышающее константу. На каждой итерации цикла `for k` увеличивается не более чем на единицу, значит максимально возможное значение `k=n-1`. Поскольку внутри цикла `while` значение `k` лишь уменьшается, получается, что `k` не может суммарно уменьшиться больше, чем `n-1` раз. Значит цикл `while` в итоге выполнится не более n раз, что дает итоговую оценку времени алгоритма $O(n)$.

Рассмотрим алгоритм Кнута-Морриса-Пратта, основанный на использовании префикс-функции. Как и в примитивном алгоритме поиска подстроки, образец «перемещается» по строке слева направо с целью обнаружения совпадения. Однако ключевым отличием является то, что при помощи префикс-функции мы можем избежать заведомо бесполезных сдвигов.

Пусть `S[0..m-1]` – подстрока, `T[0..n-1]` – строка, в которой ведется поиск. Рассмотрим сравнение строк на позиции i, то есть образец S[0..m–1] сопоставляется с частью строки `T[i..i+m-1]`. Предположим, первое несовпадение произошло между символами `S[j]` и `T[i+j]`, где i < j < m. Обозначим P = S[0..j–1] = T[i..i+j–1]. При сдвиге можно ожидать, что префикс S сойдется с каким-либо суффиксом строки P. Поскольку длина наиболее длинного префикса, являющегося одновременно суффиксом, есть префикс-функция от строки S для индекса j, приходим к следующему алгоритму:

5. Построить префикс-функцию образца S, обозначим ее F.
6. Положить k = 0, i = 0.
7. Сравнить символы S[k] и T[i]. Если символы равны, увеличить k на 1. Если при этом k стало равно длине подстроки, то вхождение образца S в строку T найдено, индекс вхождения равен i – |S| + 1. Алгоритм завершается.

Если символы не равны, используем префикс-функцию для оптимизации сдвигов. Пока k > 0, присвоим k = F[k–1] и перейдем в начало шага 3.

4. Пока i < |T|, увеличиваем i на 1 и переходим в шаг 3.

Точный анализ КМП-поиска, как и сам его алгоритм, весьма сложен. Его разработчики доказывают, что требуется порядка |T| + |S| сравнений символов, что значительно лучше, чем |T| × |S| сравнений из прямого поиска. Они так же отмечают то приятное свойство, что указатель сканирования строки i никогда не возвращается назад, в то время как при прямом поиске после несовпадения просмотр всегда начинается с первого символа образа и поэтому может включать символы строки, которые уже просматривались ранее. Это может привести к затруднениям, если строка читается из вторичной памяти, ведь в этом случае возврат обходится дорого. Даже при буферизованном вводе может встретиться столь большой образ, что возврат превысит емкость буфера.

7.6. Вопросы для самопроверки

1. Дайте определение поиска.
2. Приведите классификацию алгоритмов поиска.
3. Чем можно объяснить многообразие алгоритмов поиска? Почему не существует универсального алгоритма поиска?
4. Приведите графическую схему рассмотренных алгоритмов поиска.
5. Приведите сравнение сложности алгоритмов поиска.

7.7. Задания для самопроверки

Выполните задания из лабораторной работы №7 (приложение 1).

ЗАКЛЮЧЕНИЕ

В данном учебном пособии изложен материал второго семестра изучения дисциплины «Языки и методы программирования», позволяющий получить представление об использовании языка C++ в задачах по обработке данных, как в оперативной памяти, так и на внешних носителях.

Были рассмотрены базовые концепции построения сложных структурированных типов данных. Рассмотрены ключевые элементы технологии обработки информации такие как: работа с массивами структур, организация связных списков, реализация алгоритмов сортировки и поиска. Представлены основы объектно-ориентированного программирования в рамках реализации инкапсуляции и наследования. Весь теоретический материал проиллюстрирован программным кодом.

Дополнительные сведения по разработке приложений, а также задания для самостоятельного решения и множество различных примеров можно получить из книг, приведенных в списке рекомендованной и дополнительной литературы.

ПРИЛОЖЕНИЕ 1. ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА

Порядок выполнения работ

1. Получение варианта задания на лабораторную работу.
2. Разработка алгоритма решения задачи.
3. Составление программы на языке C++.
4. Отладка программы.
5. Тестирование программы.
6. Составление отчета о проделанной работе.

Требования к оформлению отчета

1. Оформить титульный лист с указанием темы работы.
2. Сформулировать цель и задачи работы.
3. Привести формулировку задания.
4. Записать текст программы.
5. Привести результаты тестирования программы. Количество элементов в массивах, списках, файлах (в зависимости от задания) должно быть не менее 10.
6. Сформулировать вывод по проделанной работе.

Критерии результативности выполнения работы

Лабораторная работа считается успешно выполненной, если:

- представленный отчет содержит все необходимые пункты, согласно требованиям;
- успешно демонстрируется работа программы как на выбранных студентом исходных данных, так и на заданных преподавателем исходных данных;
- студент правильно отвечает на контрольные вопросы.

ЛАБОРАТОРНАЯ РАБОТА № 1

МАССИВЫ СТРУКТУР

1. Цель и задачи работы

Целью работы является получение обучающимися навыков работы с массивами структур в языке C++.

Задачами работы являются:

получение практических навыков работы с интегрированными типами данных – структурами и массивами структур на языке C++;

тестирование работоспособности программы для различных исходных данных.

2. Варианты заданий

Разработать консольные приложения на языке C++, решающие следующие задачи:

Общие задания.

Задание 1. Задан массив структур «Студент» (фамилия; имя; баллы по трем предметам ЕГЭ). Объявить тип данных «структура Студент» в отдельном h-файле. Вывести на экран результаты поиска: студента с максимальной суммой баллов; студента с указанной фамилией.

Задание 2. Задан массив структур «Студент» (фамилия; имя; пол; вложенная структура дата рождения в формате день:месяц:год). Вывести фамилии студентов, подлежащих призыву.

Задание 3 (индивидуальное).

Для всех заданий объявить структурированный тип данных в отдельном h-файле.

1. Задан массив структур «Автосервис» (регистрационный номер автомобиля; фамилия владельца; пробег; стоимость ремонта; фамилия мастера, выполнившего ремонт). Вывести на экран общую сумму ремонта всех автомобилей.
2. Задан массив структур «Сотрудник» (фамилия сотрудника; должность; возраст; стаж; заработная плата). Вывести на экран фамилии и должности сотрудников требуемого возраста.
3. Задан массив структур «Режим дня» (наименования занятия в детском саду; время начала занятия; время окончания занятия). Вывести на экран суммарное количество свободного времени детей (наименование занятия – free_time).
4. Задан массив структур «Абонент» (фамилия; домашний адрес; номер телефона; тариф; баланс). Вывести на экран список абонентов с указанием номеров телефонов для рассылки сообщений о том, что достигнут порог отключения (нулевой или отрицательный баланс).
5. Задан массив структур «Склад» (вид; наименование товара; количество; стоимость; процент торговой надбавки). Вывести на экран перечень товаров с указанием общей стоимости товара, с учетом торговой надбавки.
6. Задан массив структур «Ученик» (фамилия; класс; оценка по математике; оценка по русскому языку; оценка по литературе). Вывести на экран значения средних оценок по всем предметам.
7. Задан массив структур «Афиша» (название фильма; номер зала; количество мест; стоимость билета; время сеанса). Вывести на экран информацию о максимально возможной выручке за каждый фильм.

8. Задан массив структур «Покупатель» (фамилия покупателя; имя; номер карты; наименование товара; стоимость покупки). Вывести на экран сведения о покупателях и суммарной стоимости их покупок.
9. Задан массив структур «Игрушка» (название игрушки; цена; количество; возрастные ограничения). Вывести на экран наименования и стоимость игрушек, подходящих ребенку заданного возраста и имеющихся в наличии.
10. Задан массив структур «Владелец» (фамилия владельца квартиры; имя; номер квартиры; площадь квартиры; задолженность по оплате коммунальных услуг). Вывести на экран список задолжников по оплате коммунальных услуг.
11. Задан массив структур «Рейс» (место отправления; время отправления; место прибытия; время прибытия). Вывести на экран список возможных маршрутов из заданного пункта с указанием времени в пути.
12. Задан массив структур «Читатель» (фамилия читателя; номер читательского билета; наименование книги; дата возврата). Вывести на экран фамилии и номера читательских билетов читателей, имеющих задолженности по возврату книг.
13. Задан массив структур «Тур» (наименование тура; страна; продолжительность; стоимость; количество свободных путевок). Вывести на экран список туров, имеющихся в наличии.
14. Задан массив структур «Преподаватель» (фамилия; имя; отчество; разряд; год рождения). Вывести на экран список преподавателей заданного разряда.
15. Задан массив структур «Специальность» (название университета; шифр специальности подготовки; количество бюд-

жетных мест; количество заявлений). Вывести на экран этот список специальностей с указанием конкурса (отношение количества заявлений к количеству бюджетных мест).

16. Задан массив структур «Спортсмен» (фамилия спортсмена; порядковый номер; результаты трех попыток). Вывести на экран список спортсменов с указанием суммарного результата каждого спортсмена.
17. Задан массив структур «Вклад» (фамилия клиента банка; номер лицевого счета; сумма на счете; дата операции). Вывести на экран данные о клиентах, вносивших изменения за последние три месяца.
18. Задан массив структур «Сотрудник» (фамилия; имя; квалификация; возраст; стаж работы). Вывести на экран данные о сотруднике, соответствующем требованиям на замещение вакантной должности.
19. Задан массив структур «Государство» (материк; название; столица; численность населения; площадь). Вывести на экран список государств и столиц с указанием плотности населения.
20. Задан массив структур «Кредит» (фамилия заемщика; сумма кредита; процентная ставка; срок кредитования). Вывести на экран сумму переплаты каждого заемщика.
21. Задан массив структур «Процесс» (наименование процесса; время начала; время окончания). Вывести на экран общую продолжительность выполнения работы.
22. Задан массив структур «Картина» (автор; название картины; стиль; размеры; цена). Вывести на экран сведения о картинах, выполненных в запрашиваемом стиле.

23. Задан массив структур «Телефон» (производитель; наименование модели; оперативная память; разрешение камеры; стоимость). Вывести на экран перечень телефонов, параметры которых превышают заданные значения параметров стоимость и объем оперативной памяти.
24. Задан массив структур «Пациент» (фамилия пациента; номер страхового полиса; домашний адрес; диагноз). Вывести на экран список всех пациентов, проживающих на заданной улице, с указанием номера страхового полиса и диагноза.
25. Задан массив структур «Тендер» (номер конкурса; наименование организации-участника тендера; стаж участия в работах; собственный капитал; заявленная стоимость работ). Вывести на экран организации, удовлетворяющие требованиям конкурса.

3. Контрольные вопросы

1. Как объявить структуру?
2. Как обращаться к полям структуры через объект?
3. Как обращаться к полям структуры через указатель на объект?
4. Как в памяти хранится переменная структурного типа?
5. Для чего используются вложенные структуры?

ЛАБОРАТОРНАЯ РАБОТА № 2
ОБРАБОТКА ТЕКСТОВЫХ ФАЙЛОВ

1. Цель и задачи работы

Целью работы является получение обучающимися навыков чтения и записи текстовых файлов на языке C++.

Задачами работы являются:

– получение практических навыков работы с текстовыми файлами на языке C++;
– тестирование работоспособности программы для различных исходных данных.

2. Варианты заданий

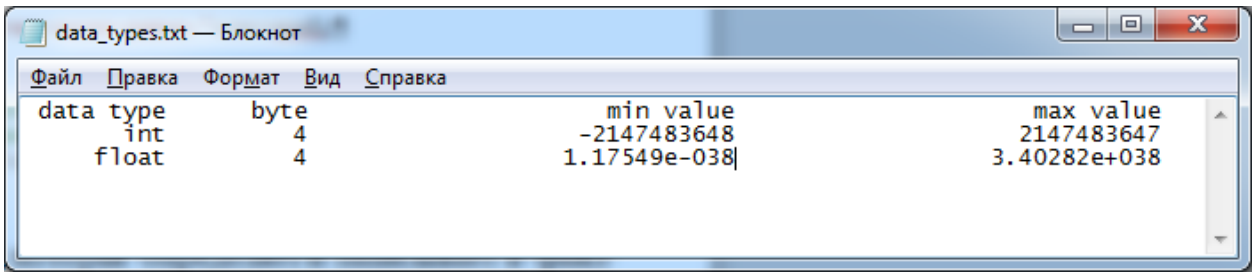
Задание 1. Исходный текстовый файл “data.txt” находится в папке Temp на диске C: и содержит целые числа. Переписать положительные числа в текстовый файл “positive.txt” с тем же местоположением. При выполнении задания сохранить строгую структуру строк заданного файла, т.е. в конечном файле количество строк и их порядок должны повторять исходный.

Задание 2. Разработать программу, которая определяет и записывает в текстовый файл характеристики основных типов данных в C++: char, short int, unsigned short int, int, unsigned int, long int, unsigned long int, float, double, long double.

Характеристики:

1. число байт, отводимое под тип данных;
2. минимальное значения, которое может хранить указанный тип данных;
3. максимальное значения, которое может хранить указанный тип данных.

Запись в файл должна выполняться с выравниванием в формате, приведенном на рисунке:



Задание 3 (индивидуальное).

1. Найти в текстовых файлах file1.txt и file2.txt совпадающие числа и записать их в новый текстовый файл result.txt. В каждом из исходных файлов числа не повторяются.
2. Дан текстовый файл 1.txt, содержащий целые числа, записанные в несколько строк. Найти суммы четных чисел в каждой нечетной строке и записать их в новый текстовый файл 2.txt.
3. В исходном текстовом файле input.txt записаны в несколько строк вещественные числа. Найти произведение чисел каждой строки. Если произведение положительное, то переписать строку чисел в файл output.txt.
4. Текстовый файл data.txt содержит целые числа. Переписать числа в три файла, сохраняя деление на строки: в файл prime.txt записать простые числа, в файл composite.txt – составные, в файл other.txt – остальные числа.
5. Задан исходный текстовый файл numbers.txt, содержащий вещественные числа, записанные в несколько строк. В новый текстовый файл sqrt.txt записать значения квадратных корней положительных чисел из исходного файла. При выполнении задания сохранить строгую структуру строк заданного файла, т.е. в конечном файле количество строк и их порядок должны повторять исходный.

6. Текстовый файл integer.txt содержит целые числа. В файл odd.txt выписать все нечетные числа из этого файла, в файл even.txt – все четные числа. В последней строке каждого файла указать количество чисел.
7. Сформировать текстовый файл 3.txt из двух заданный текстовых файлов 1.txt и 2.txt по следующему алгоритму: первая строка берется из первого файла, вторая – из второго, третья – из первого, четвертая - из второго и так далее с чередованием до полного завершения обоих исходных файлов.
8. Даны два текстовых файла f1.txt и f2.txt, содержащие числа. Сформировать из них текстовый файл f3.txt следующим образом: первая строка это сумма чисел первой строки из первого файла, вторая – это сумма чисел первой строки из второго, третья – это сумма чисел второй строки из первого, четвертая – это сумма чисел второй строки из второго и так далее с чередованием до полного завершения обоих исходных файлов.
9. В исходном текстовом файле input.txt записаны числа в несколько строк. Переписать в файл output.txt строки исходного файла, содержащие равное количество положительных и отрицательных элементов. При выполнении задания сохранить строгую структуру строк заданного файла, т.е. в конечном файле количество строк и их порядок должны повторять исходный.
10. Даны текстовые файлы numbers1.txt и numbers2.txt, содержащие числа. Создать текстовый файл numbers3.txt и заполнить его значениями по следующему правилу: в первую строку записать сумму четных чисел первых строк исходных файлов, во вторую строку – сумму нечетных чисел вторых строк, в третью – сумму четных чисел первых строк

и т.д. с чередованием до полного окончания исходных файлов.

11. Текстовый файл numbers.txt содержит числа в несколько строк. Найти сумму чисел каждой строки. Переписать в файл sum.txt строки исходного файла, сумма элементов которых превосходит введенное пользователем значение. При выполнении задания сохранить строгую структуру строк заданного файла, т.е. в конечном файле количество строк и их порядок должны повторять исходный.
12. Текстовый файл data1.txt содержит вещественные числа, записанные в виде нескольких строк. Пользователь с клавиатуры вводит некоторое значение m. Переписать числа из исходного файла, не превосходящие m, в файл data2.txt, остальные числа – в файл data3.txt, не нарушая строгую структуру строк заданного файла, т.е. в результирующих файлах количество строк и их порядок должны повторять исходный.
13. Даны два файла input1.txt и input2.txt, содержащие вещественные числа. Числа из первого файла input1.txt, не содержащиеся во втором файле input2.txt, переписать в файл output1.txt. Числа из второго файла input2.txt, не содержащиеся в первом файле input1.txt, переписать в файл output2.txt. В последних строках каждого полученного файла указать количество чисел.
14. Задан текстовый файл 1.txt, содержащий целые числа, записанные в несколько строк. С клавиатуры вводятся целые числа a и b. Записать в новый текстовый файл 2.txt числа из файла 1.txt сначала принадлежащие отрезку [a, b], затем оставшиеся числа. Взаимный порядок чисел в каждой из групп должен быть сохранен.

15. Переписать в новый текстовый файл reverse.txt положительные числа из исходного файла base.txt построчно в обратном порядке в строке, сохраняя порядок строк.
16. Дан текстовый файл source.txt, содержащий вещественные числа. Уникальные числа из исходного файла переписать в файл unique.txt, повторяющиеся числа – в файл repeat.txt без повторов. Взаимное расположение чисел должно остаться неизменным.
17. В текстовом файле data.txt построчно записаны целые числа. Создать новый текстовый файл total.txt, записать в него из исходного файла строки, содержащие только отрицательные числа. При выполнении задания сохранить строгую структуру строк заданного файла, т.е. в конечном файле количество строк и их порядок должны повторять исходный.
18. Даны текстовые файлы input1.txt и input2.txt, содержащие числа. Найти суммы положительных чисел соответствующих строк и записать их в новый текстовый файл output.txt. В последней строке записать количество положительных чисел исходных файлов.
19. Исходный текстовый файл 1.txt содержит вещественные числа. Создать новый текстовый файл 2.txt, записать в него только целые числа четных строк исходного файла.
20. Дан текстовый файл data.txt, содержащий вещественные числа. Найти среднее арифметическое каждой строки. Элементы строки, большие среднего арифметического, переписать в файл larger.txt, остальные – в файл less.txt. Взаимное расположение чисел должно сохраниться.
21. Дан текстовый файл data.txt, содержащий вещественные числа. Найти максимальное значение среди отрицательных

чисел каждой строки и записать в файл max.txt как отдельную строку. Если строка пустая или отрицательных чисел нет, записать «0.0». Взаимное расположение строк в исходном и результирующем файлах должно сохраниться.

22. Дан текстовый файл data.txt, содержащий целые числа. Переписать эти числа в файл data2.txt вычитая из каждого числа порядковый номер этого числа в строке. Т.е. из первого числа в строке вычитаем 1, из второго 2 и т.д. Из первого числа следующей строки опять вычитаем 1. В конечном файле количество строк и их порядок должны повторять исходный.

23. Дан текстовый файл data.txt, содержащий целые числа. Переписать эти числа в файл data2.txt построчно, попарно меняя местами числа внутри строки. Т.е. сначала записываем число с четным порядковым номером, а потом с нечетным. Например, строка исходного файла:
    4 6 8 10
    в новый файл нужно записать:
    6 4 10 8
    В конечном файле количество строк и их порядок должны повторять исходный.

24. Дан текстовый файл data.txt, содержащий целые числа. Переписать данные в файл data2.txt, попарно меняя местами строки: сначала записываем вторую строку, потом превую, затем записываем четвертую строку, потом третью и т.д.

25. Дан текстовый файл data.txt, содержащий целые числа. Переписать эти числа в файл data2.txt разделив каждое число на значение максимума числа в этой строке. Если значение максимума равно 0.0, записать «inf» В конечном файле количество строк и их порядок должны повторять исходный.

3. Контрольные вопросы

1. Что такое режим открытия файла?
2. Перечислите режимы открытия файла.
3. Как можно осуществить комбинацию режимов открытия файла?
4. Как произвести проверку открытия файла?
5. Зачем нужно закрывать файл после завершения работы?

ЛАБОРАТОРНАЯ РАБОТА № 3
ДВОИЧНЫЕ (БИНАРНЫЕ) ФАЙЛЫ

1. Цель и задачи работы

Целью работы является получение обучающимися навыков обработки бинарных файлов на языке C++.

Задачами работы являются:

– получение практических навыков работы с бинарными файлами на языке C++;
– тестирование работоспособности программы для различных исходных данных.

2. Варианты заданий

Задание 1. Создать бинарный файл, содержащий информацию о 10 студентах. Прочитать данные в массив и вывести их на экран.

Задание 2. Дан бинарный файл, содержащий информацию о студентах. Запросить у пользователя два числа a и b. Поменять местами в файле две записи с номерами a и b.

Задание 3 (индивидуальное).

1. Задан массив структур «Автосервис» (регистрационный номер автомобиля; фамилия владельца; пробег; стоимость ремонта; фамилия мастера, выполнившего ремонт). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
2. Задан массив структур «Сотрудник» (фамилия сотрудника; должность; возраст; стаж; заработная плата). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
3. Задан массив структур «Режим дня» (наименования занятия в детском саду; время начала занятия; время окончания занятия). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
4. Задан массив структур «Абонент» (фамилия; домашний адрес; номер телефона; тариф; баланс). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
5. Задан массив структур «Склад» (вид; наименование товара; количество; стоимость; процент торговой надбавки). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
6. Задан массив структур «Ученик» (фамилия; класс; оценка по математике; оценка по русскому языку; оценка по литературе). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.

7. Задан массив структур «Афиша» (название фильма; номер зала; количество мест; стоимость билета; время сеанса). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
8. Задан массив структур «Покупатель» (фамилия покупателя; имя; номер карты; наименование товара; стоимость покупки). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
9. Задан массив структур «Игрушка» (название игрушки; цена; количество; возрастные ограничения). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
10. Задан массив структур «Владелец» (фамилия владельца квартиры; имя; номер квартиры; площадь квартиры; задолженность по оплате коммунальных услуг). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
11. Задан массив структур «Рейс» (пункт отправления; время отправления; пункт прибытия; время прибытия). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
12. Задан массив структур «Читатель» (фамилия читателя; номер читательского билета; наименование книги; дата возврата). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.

13. Задан массив структур «Тур» (наименование тура; страна; продолжительность; стоимость; количество свободных путевок). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
14. Задан массив структур «Преподаватель» (фамилия; имя; отчество; разряд; год рождения). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
15. Задан массив структур «Специальность» (название университета; шифр специальности подготовки; количество бюджетных мест; количество заявлений). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
16. Задан массив структур «Спортсмен» (фамилия спортсмена; порядковый номер; результаты трех попыток). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
17. Задан массив структур «Вклад» (фамилия клиента банка; номер лицевого счета; сумма на счете; дата операции). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
18. Задан массив структур «Сотрудник» (фамилия; имя; квалификация; возраст; стаж работы). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.

19. Задан массив структур «Государство» (материк; название; столица; численность населения; площадь). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
20. Задан массив структур «Кредит» (фамилия заемщика; сумма кредита; процентная ставка; срок кредитования). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
21. Задан массив структур «Процесс» (наименование процесса; время начала; время окончания). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
22. Задан массив структур «Картина» (автор; название картины; стиль; размеры; цена). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
23. Задан массив структур «Телефон» (производитель; наименование модели; оперативная память; разрешение камеры; стоимость). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
24. Задан массив структур «Пациент» (фамилия пациента; номер страхового полиса; домашний адрес; диагноз). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.
25. Задан массив структур «Тендер» (номер конкурса; наименование организации-участника тендера; стаж участия в

работах; собственный капитал; заявленная стоимость работ). Создать бинарный файл, содержащий информацию о 10 объектах указанного типа данных. Прочитать данные из файла и вывести их на экран.

3. Контрольные вопросы

1. Приведите алгоритм чтения информации из бинарного файла. В чем специфика использования метода read?
2. Перечислите методы класса ifstream для обработки бинарного файла
3. Как можно осуществить комбинацию режимов открытия файла?
4. Как осуществляется запись и чтение структур в бинарный файл?

ЛАБОРАТОРНАЯ РАБОТА № 4
КЛАССЫ В C++

1. Цель и задачи работы

Целью работы является получение обучающимися навыков работы с классами в языке C++.

Задачами работы являются:

– получение практических навыков объектно-ориентированного программирования на языке C++;
– тестирование работоспособности программы для различных исходных данных.

2. Варианты заданий

Задание 1**.** В текстовом файле «STUDENTS.TXT» записана информация о студентах (файл считать корректным). Создать файл

«GRANT.TXT», содержащий список студентов-первокурсников, получающих государственную академическую стипендию в увеличенном размере (имеющих суммарный балл по ЕГЭ от 215 баллов и выше).

Задание 2**.** Разработать класс Date, хранящий дату в виде 3 полей: день, месяц, год. Класс должен содержать 3 конструктора.

- Конструктор, принимающий три параметра: день, месяц, год.
- Конструктор, принимающий дату в виду строки, например, «02.08.2021» или «2.8.21»
- Конструктор по умолчанию: поля инициализируются текущей датой.
- Добавить в класс следующие методы – члены класса:
    - Методы получения и установки значения каждого поля (всего 6 методов).
    - Метод представления строки в строковом виде `string toString() const`.
    - Метод сравнения двух дат:
      `bool equals(const Date &d) const`.
      (Примечание: вместо метода `equals()` можно перегрузить оператор сравнения «==»).
    - Метод, возвращающий разницу в днях между двумя датами:
      `int daysTo(const Date &d) const`.

Примечание: если дата `d` находится в прошлом по отношению к дате текущего объекта, то метод возвращает отрицательное значение.

Внимание! В конструкторах и методах класса не должно быть вывода на экран и ввода с клавиатуры.

Задание 3 (индивидуальное). Для всех вариантов описать заданный класс, создать текстовый файл, содержащий корректные данные в соответствии с описанием класса, считать данные из текстового файла в массив объектов класса и выполнить задание.

1. Задан класс «Автосервис» (регистрационный номер автомобиля; фамилия владельца; пробег; стоимость ремонта; фамилия мастера, выполнившего ремонт). Вывести на экран список автомобилей, стоимость ремонта которых превышает среднюю стоимость ремонта. Запросить фамилию мастера. Создать текстовый файл, содержащий список автомобилей, ремонт которых выполнил указанный мастер.
2. Задан класс «Сотрудник» (фамилия сотрудника; должность; возраст; стаж; заработная плата). Вывести на экран фамилии и должности сотрудников требуемого возраста. Запросить должность. Создать текстовый файл, содержащий список сотрудников, занимающих эту должность.
3. Задан класс «Режим дня» (наименования занятия в детском саду; время начала занятия; время окончания занятия). Вывести на экран суммарное количество свободного времени детей (наименование занятия – free_time). Создать текстовый файл, содержащий список занятий с указанием длительности занятия.
4. Задан класс «Абонент» (фамилия; домашний адрес; номер телефона; тариф; баланс). Вывести на экран список абонентов с указанием номеров телефонов для рассылки сообщений о том, что достигнут порог отключения (нулевой или отрицательный баланс). Запросить тариф. Создать текстовый файл, содержащий список абонентов, подключенных к этому тарифу.
5. Задан класс «Склад» (наименование товара; количество; стоимость; процент торговой надбавки). Вывести на экран перечень товаров, подсчитать общую стоимость товаров с учетом надбавки. Создать текстовый файл, содержащий список товаров с указанием прибыли за каждый товар.

6. Задан класс «Ученик» (фамилия; класс; оценка по математике; оценка по русскому языку; оценка по литературе). Вывести на экран учеников с максимальной успеваемостью. Запросить класс. Создать текстовый файл, содержащий список учеников этого класса.
7. Задан класс «Афиша» (название фильма; номер зала; количество мест; стоимость билета; время сеанса). Вывести на экран информацию фильмах с максимальной стоимостью билета. Запросить название фильма. Создать текстовый файл, содержащий список показов этого фильма.
8. Задан класс «Покупатель» (фамилия покупателя; имя; номер карты; наименование товара; стоимость покупки). Вывести на экран значение средней стоимости покупок. Запросить фамилию покупателя. Создать текстовый файл, содержащий список покупок этого клиента.
9. Задан класс «Игрушка» (название игрушки; цена; количество; возрастные ограничения). Вывести на экран наименования игрушек, подходящих ребенку заданного возраста и не превышающих заданной стоимости. Создать текстовый файл, содержащий список всех игрушек, имеющихся в наличии.
10. Задан класс «Владелец» (фамилия владельца квартиры; имя; номер квартиры; площадь квартиры; задолженность по оплате коммунальных услуг). Вывести на экран общую площадь дома. Создать текстовый файл, содержащий список всех задолжников по оплате коммунальных услуг.
11. Задан класс «Рейс» (пункт отправления; время отправления; пункт прибытия; время прибытия). Вывести на экран список возможных маршрутов из заданного пункта с указанием времени в пути. Создать текстовый файл, содержащий список всех рейсов.

12. Задан класс «Читатель» (фамилия читателя; номер читательского билета; наименование книги; дата возврата). Вывести на экран фамилии и номера читательских билетов читателей, имеющих задолженности по возврату книг на текущую дату. Запросить фамилию читателя. Создать текстовый файл, содержащий список книг, взятых этим читателем.
13. Задан класс «Тур» (наименование тура; страна; продолжительность; стоимость; количество свободных путевок). Вывести на экран список туров, имеющихся в наличии. Запросить страну. Создать текстовый файл, содержащий список туров в эту страну.
14. Задан класс «Преподаватель» (фамилия; имя; отчество; разряд; год рождения). Вывести на экран преподавателей, возраст которых не превышает 35 лет. Запросить разряд. Создать текстовый файл, содержащий список всех преподавателей указанного разряда.
15. Задан класс «Специальность» (название университета; шифр специальности подготовки; количество бюджетных мест; количество заявлений). Вывести на экран перечень специальностей с указанием конкурса (отношение количества заявлений к количеству бюджетных мест) и записать этот перечень в новый файл.
16. Задан класс «Спортсмен» (фамилия спортсмена; порядковый номер; результаты трех попыток). Вывести на экран список спортсменов с указанием суммарного результата. Записать в текстовый файл спортсменов, показавших максимальный результат.
17. Задан класс «Вклад» (фамилия клиента банка; номер лицевого счета; сумма на счете; дата операции). Вывести на экран данные о клиентах, вносивших изменения за послед-

ние три месяца. Запросить фамилию клиента. Создать текстовый файл, содержащий список счетов этого клиента.

18. Задан класс «Сотрудник» (фамилия; имя; квалификация; возраст; стаж работы). Вывести на экран данные о сотруднике, соответствующем требованиям на замещение вакантной должности. Запросить квалификацию. Создать текстовый файл, содержащий список сотрудников, имеющих эту квалификацию.
19. Задан класс «Государство» (материк; название; столица; численность населения; площадь). Вывести на экран список государств и столиц с указанием плотности населения и записать этот список в файл.
20. Задан класс «Кредит» (фамилия заемщика; сумма кредита; процентная ставка; срок кредитования). Вывести на экран сумму переплаты каждого заемщика. Запросить процентную ставку. Создать текстовый файл, содержащий список заемщиков, которые взяли кредит по этой ставке.
21. Задан класс «Процесс» (наименование процесса; время начала; время окончания). Вывести на экран общую продолжительность выполнения работы. Создать текстовый файл, содержащий список процессов с указанием длительности каждого процесса.
22. Задан класс «Картина» (автор; название картины; стиль; размеры; цена). Вывести на экран сведения о картинах, выполненных в запрашиваемом стиле. Запросить фамилию автора. Создать текстовый файл, содержащий список картин этого автора.
23. Задан класс «Телефон» (производитель; наименование модели; оперативная память; разрешение камеры; стоимость). Вывести на экран перечень телефонов с макси-

мальной стоимостью. Запросить производителя. Создать текстовый файл, содержащий список телефонов этого производителя.

24. Задан класс «Пациент» (фамилия пациента; номер страхового полиса; домашний адрес; диагноз). Вывести на экран список всех пациентов, проживающих на заданной улице, с указанием номера страхового полиса и диагноза. Запросить диагноз. Создать текстовый файл, содержащий список пациентов с указанным диагнозом.
25. Задан класс «Тендер» (номер конкурса; наименование организации-участника тендера; стаж участия в работах; собственный капитал; заявленная стоимость работ). Вывести на экран организации, удовлетворяющие требованиям конкурса и сохранить этот список в новый файл.

3. Контрольные вопросы

1. Дайте определение класса в программировании.
2. Назовите принципы объектно-ориентированного программирования.
3. Что такое модификатор доступа?
4. В чем разница между public, private, protected?
5. Для чего нужен указатель this.

ЛАБОРАТОРНАЯ РАБОТА № 5
ЛИНЕЙНЫЕ ОДНОСВЯЗНЫЕ СПИСКИ

1. Цель и задачи работы

Целью работы является получение обучающимися навыков создания и обработки линейных динамических списков.

Задачами работы являются:

– получение практических навыков работы с линейными динамическими списками структур;
– тестирование работоспособности программы для различных исходных данных.

2. Варианты заданий

Задание 1. Реализовать метод класса MyList для удаления последнего элемента списка.

Задание 2. Осуществить поиск элемента списка MyList по фамилии студента; по возрасту. Информация для поиска запрашивается у пользователя.

Задание 3. В каждом варианте задана некоторая структура. Информация должна хранится в бинарном файле. Реализовать следующие функции обработки информации:

– добавить данные,
– удалить данные,
– вывести на экран,
– считать данные из файла в список,
– сохранить в файл,
– индивидуальное задание.

В каждом индивидуальном задании предусмотрено выполнение группировки по определенному параметру, например, в первом задании группировка осуществляется по фамилии мастера. Для этого в исходный файл следует добавить несколько записей с одинаковым значением параметра группировки. Так для задания №1, следует добавить несколько разных записей мастера 1, несколько разных записей мастера 2 и т.д.

Внимание!! Пользоваться стандартными библиотечными классами (list, vector и т.п.) нельзя!

1. Задан список структур «Автосервис» (регистрационный номер автомобиля; фамилия владельца; пробег; стоимость ремонта; фамилия мастера, выполнившего ремонт). Вывести на экран общую сумму ремонта по каждому мастеру.
2. Задан список структур «Сотрудник» (фамилия сотрудника; должность; возраст; стаж; заработная плата). Вывести на экран среднюю заработную плату сотрудников каждой должности.
3. Задан список структур «Режим дня» (наименования занятия в детском саду; время начала занятия; время окончания занятия). Вывести на экран суммарное количество каждого занятия детей.
4. Задан список структур «Абонент» (фамилия; домашний адрес; номер телефона; тариф; баланс). Вывести на экран список тарифов с указанием количества абонентов.
5. Задан список структур «Склад» (вид; наименование товара; количество; стоимость; процент торговой надбавки). Вывести на экран суммарное количество товаров каждого вида.
6. Задан список структур «Ученик» (фамилия; класс; оценка по математике; оценка по русскому языку; оценка по литературе). Вывести на экран перечень классов с указанием среднего балла по классу.
7. Задан список структур «Афиша» (название фильма; номер зала; количество мест; стоимость билета; время сеанса). Вывести на экран информацию о максимально возможной выручке за каждый фильм.

8. Задан список структур «Покупатель» (фамилия покупателя; имя; номер карты; наименование товара; стоимость покупки). Вывести на экран сведения о покупателях и суммарной стоимости покупок каждого покупателя.
9. Задан список структур «Игрушка» (название игрушки; цена; количество; возрастные ограничения). Вывести на экран наименования и суммарное количество игрушек каждой возрастной группы согласно возрастным ограничениям (сначала 0+, потом 3+ и т.д.).
10. Задан список структур «Владелец» (фамилия владельца квартиры; номер дома; номер квартиры; площадь квартиры; задолженность по оплате коммунальных услуг). Вывести на экран сумму задолженности по оплате коммунальных услуг по каждому дому.
11. Задан список структур «Рейс» (пункт отправления; время отправления; пункт прибытия; время прибытия). Вывести на экран список возможных маршрутов из каждого пункта отправления места с указанием времени в пути.
12. Задан список структур «Читатель» (фамилия читателя; номер читательского билета; наименование книги; дата возврата). Вывести на экран фамилии и номера читательских билетов читателей с указанием количества взятых книг.
13. Задан список структур «Тур» (наименование тура; страна; продолжительность; стоимость; количество свободных путевок). Вывести на экран список стран с указанием суммарного количества путевок в наличии.
14. Задан список структур «Преподаватель» (фамилия; имя; отчество; разряд; год рождения). Вывести на экран количество преподавателей каждого разряда.

15. Задан список структур «Специальность» (название университета; шифр специальности подготовки; количество бюджетных мест; количество заявлений). Вывести на экран суммарное количество заявлений по каждой специальности по всем университетам.
16. Задан список структур «Спортсмен» (фамилия спортсмена; порядковый номер; результаты трех попыток). Вывести на экран список суммарных результатов с указанием количества спортсменов, показавших данный результат.
17. Задан список структур «Вклад» (фамилия клиента банка; номер лицевого счета; сумма на счете; дата операции). Вывести на экран статистику операций по счетам по каждому месяцу с указанием количества клиентов, проводившим операции в данном месяце.
18. Задан список структур «Сотрудник» (фамилия; имя; квалификация; возраст; стаж работы). Вывести на экран данные о количестве сотрудников каждой квалификации.
19. Задан список структур «Государство» (материк; название; столица; численность населения; площадь). Вывести на экран список материков с указанием плотности населения.
20. Задан список структур «Кредит» (фамилия заемщика; сумма кредита; процентная ставка; срок кредитования). Вывести на экран количество заемщиков, взявших кредит с каждой процентной ставкой.
21. Задан список структур «Процесс» (наименование процесса; время начала; время окончания). Вывести на экран общую продолжительность выполнения каждого процесса.
22. Задан список структур «Картина» (автор; название картины; стиль; размеры; цена). Вывести на экран сведения о картинах, каждого стиля с указанием их количества.

23. Задан список структур «Телефон» (производитель; наименование модели; оперативная память; разрешение камеры; стоимость). Вывести на экран количество телефонов каждого производителя.
24. Задан список структур «Пациент» (фамилия пациента; номер страхового полиса; домашний адрес; диагноз). Вывести на экран перечень диагнозов с указанием количества пациентов с данным диагнозом.
25. Задан список структур «Тендер» (номер конкурса; наименование организации-участника тендера; стаж участия в работах; собственный капитал; заявленная стоимость работ). Вывести на экран перечень организаций-участников с указанием количества заявок каждой организации.

3. Контрольные вопросы

1. Дайте определение структуры данных «список».
2. Какие виды списков существуют?
3. Опишите алгоритм добавления элемента в список.
4. Опишите алгоритм удаления элемента из списка.
5. Как осуществляется удаление списка?

ЛАБОРАТОРНАЯ РАБОТА № 6
АЛГОРИТМЫ СОРТИРОВКИ

1. Цель и задачи работы

Целью работы является получение обучающимися навыков внутренней и внешней сортировки.

Задачами работы являются:

– получение практических навыков реализации алгоритмов сортировки;
– оценка эффективности алгоритмов и их сравнение;
– тестирование работоспособности программы для различных исходных данных.

2. Варианты заданий

Задание 1. Реализовать алгоритмы сортировки массива из 100 элементов: следующими методами: пузырьковая сортировка, сортировка выбором, сортировка Шелла.

Задание 2. Провести анализ эффективности рассмотренных алгоритмов сортировки в зависимости от количества перестановок элементов.

Задание 3. В лабораторной работе № 5 задана некоторая структура. Информация хранится в бинарном файле. В исходном коде к 5 лабораторной работе добавить и реализовать метод сортировки списка любым методом по произвольному полю. Добавить в меню пункт “Сортировка”. Провести тестирование программы.

3. Контрольные вопросы

1. Дайте определение сортировки.
2. Чем можно объяснить многообразие алгоритмов сортировок?
3. Почему не существует универсального алгоритма сортировки?
4. Приведите графическую схему алгоритма сортировки методом «пузырька».
5. Приведите сравнение сложности алгоритмов сортировок.

ЛАБОРАТОРНАЯ РАБОТА № 7
АЛГОРИТМЫ ПОИСКА

1. Цель и задачи работы

Целью работы является получение обучающимися навыков реализации алгоритмов поиска.

Задачами работы являются:

- получение практических навыков реализации алгоритмов поиска;
- оценка эффективности алгоритмов поиска и их сравнение;
- тестирование работоспособности программы для различных исходных данных.

2. Варианты заданий

Задание 1. Реализовать алгоритмы поиска элемента в массиве, содержащем 100 элементов: следующими методами: поиск перебором, поиск с барьером, бинарный поиск элемента массива, равного заданному значению. Оценить эффективность работы алгоритмов по количеству сравнений.

Задание 2. Написать программу поиска образа в строке по методу Кнута, Морриса и Пратта. Предусмотреть возможность существования в образе пробела. Ввести опцию чувствительности / нечувствительности к регистру.

Задание 3. В лабораторной работе № 5 задана некоторая структура. Информация хранится в бинарном файле. Реализовать два метода поиска информации в списке произвольным методом по числовому и текстовому полю. Добавить в меню пункты «Поиск 1» и «Поиск 2». Провести тестирование программы.

3. Контрольные вопросы

1. Дайте определение поиска.

2. Приведите классификацию алгоритмов поиска.
3. Чем можно объяснить многообразие алгоритмов поиска? Почему не существует универсального алгоритма поиска?
4. Приведите графическую схему рассмотренных алгоритмов поиска.
5. Приведите сравнение сложности алгоритмов поиска.

ПРИЛОЖЕНИЕ 2. СПИСОК ЛИТЕРАТУРЫ ДЛЯ САМОСТОЯТЕЛЬНОГО ИЗУЧЕНИЯ

1. Павловская Т.А. C/C++ . Процедурное и объектно-ориентированное программирование: учеб. / Т. А. Павловская, .- М.: 2015.- 495 с
2. Аляев Ю.А., Козлов О.А. Алгоритмизация и языки программирования Pascal, C++, Visual Basic: Учеб.-справ.пособие / Ю.А.Аляев, О.А.Козлов.- М.: 2002.- 319 с
3. Павловская Т.А. C/C++.Программирование на языке высокого уровня: Учеб. / Т. А. Павловская.- М.: 2006.- 460 с
4. Фридман А.Л. Язык программирования Си++: Курс лекций / А. Л. Фридман, .- М.: 2003.- 282 с
5. Хайнеман, Джордж, Поояис, Гэри, Сеяков, Стэнли. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2-е изд.: Пер. с англ. — СпБ.: ООО “Альфа-книга”, 2017. — 432 с .: ил. — Парал. тит. англ.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

6. Павловская Т.А. C/C++ . Процедурное и объектно-ориентированное программирование: учеб. / Т. А. Павловская, .- М.: 2015.- 495 с
7. Аляев Ю.А., Козлов О.А. Алгоритмизация и языки программирования Pascal, C++, Visual Basic: Учеб.-справ.пособие / Ю.А.Аляев, О.А.Козлов.- М.: 2002.- 319 с
8. Павловская Т.А. C/C++.Программирование на языке высокого уровня: Учеб. / Т. А. Павловская.- М.: 2006.- 460 с
9. Кучеренко В. Язык программирования C++ для начинающих и не только.- М.: 2001.- 160 с
10. Фридман А.Л. Язык программирования Си++: Курс лекций / А. Л. Фридман, .- М.: 2003.- 282 с
11. Хайнеман, Джордж, Поояис, Гэри, Сеяков, Стэнли. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2-е изд.: Пер. с англ. — СпБ.: ООО “Альфа-книга”, 2017. — 432 с .: ил. — Парал. тит. англ.
12. Роберт Седжвик. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск: Пер. с англ. - К.: Издательство «ДиаСофт», 2001.- 688 с.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ 4
ВВЕДЕНИЕ..... 6
1. СТРУКТРИРОВАННЫЕ ТИПЫ ДАННЫХ 7
1.1. Типы данных в языке C++ 7
1.2 Передача структур в качестве параметров..... 18
1.3 Формат времени с использованием struct tm 19
1.4 Объединения 21
1.5 Перечисления. 24
1.5. Вопросы для самопроверки..... 27
1.6. Задания для самопроверки 27
2. РАБОТА С ТЕКСТОВЫМИ ФАЙЛАМИ 28
2.1. Общие сведения о файлах..... 28
2.2 Класс ofstream: запись в файл 31
2.3 Класс ifstream: чтение файлов..... 33
2.4 Режимы открытия файлов..... 37
2.5. Вопросы для самопроверки..... 42
2.6 Задания для самопроверки 43
3. РАБОТА С ДВОИЧНЫМИ (БИНАРНЫМИ) ФАЙЛАМИ 43
3.1. Основы работы с двоичными файлами 43
3.2 Методы произвольного доступа..... 46
3.3 Запись и чтение структуры из бинарного файла..... 47
3.4. Вопросы для самопроверки..... 48
3.5. Задания для самопроверки 48

4. КЛАССЫ В C++ 48
4.1. Знакомство с объектно-ориентированным программированием 48
4.2. Конструкторы 54
4.3. Деструкторы 63
4.4. Дружественная функция 64
4.5. Рекомендации по составу класса 66
4.6. Вопросы для самопроверки 72
4.7. Задания для самопроверки 72
5. ЛИНЕЙНЫЕ ОДНОСВЯЗНЫЕ СПИСКИ 72
5.1. Связные списки как динамические структуры данных 72
5.2. Основные операции со списком 73
5.3 Двусвязные (двунаправленные) списки 78
5.4. Стеки и очереди 79
5.5. Списки в библиотеке STL 81
5.6 Примерная реализация однонаправленного связного списка 83
5.6. Вопросы для самопроверки 93
5.7. Задания для самопроверки 93
6. АЛГОРИТМЫ СОРТИРОВОК 93
6.1. Сортировка. Общие сведения 93
6.2. Сортировка вставками 97
6.3. Сортировка выбором 99
6.4. Сортировка методом «пузырька» 102
6.5. Шейкерная сортировка 103
6.6. Сортировка Шелла 105

6.7. Сравнение различных алгоритмов сортировки 108
6.8. Вопросы для самопроверки 109
6.9. Задания для самопроверки 110
7. АЛГОРИТМЫ ПОИСКА 110
7.1. Понятие поиска 110
7.2 Поиск перебором 111
7.3 Поиск с барьером 112
7.4 Бинарный поиск 113
7.5. Поиск подстроки в строке 117
7.5.1. Прямой поиск подстроки в строке 117
7.5.2 Алгоритм Кнута, Морриса и Пратта 119
7.6. Вопросы для самопроверки 123
7.7. Задания для самопроверки 123
ЗАКЛЮЧЕНИЕ 124
ПРИЛОЖЕНИЕ 1. ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА 125
Порядок выполнения работ 125
Требования к оформлению отчета 125
Критерии результативности выполнения работы 125
ЛАБОРАТОРНАЯ РАБОТА № 1 МАССИВЫ СТРУКТУР 126
ЛАБОРАТОРНАЯ РАБОТА № 2 ОБРАБОТКА ТЕКСТОВЫХ ФАЙЛОВ 130
ЛАБОРАТОРНАЯ РАБОТА № 3 ДВОИЧНЫЕ (БИНАРНЫЕ) ФАЙЛЫ 137
ЛАБОРАТОРНАЯ РАБОТА № 4 КЛАССЫ В C++ 142
ЛАБОРАТОРНАЯ РАБОТА № 5 ЛИНЕЙНЫЕ ОДНОСВЯЗНЫЕ СПИСКИ 148

ЛАБОРАТОРНАЯ РАБОТА № 6 АЛГОРИТМЫ СОРТИРОВКИ 153
ЛАБОРАТОРНАЯ РАБОТА № 7 АЛГОРИТМЫ ПОИСКА.... 155
ПРИЛОЖЕНИЕ 2. СПИСОК ЛИТЕРАТУРЫ ДЛЯ САМОСТОЯТЕЛЬНОГО ИЗУЧЕНИЯ..... 157

Учебное издание

ЗОЛИН Алексей Георгиевич
ХАЛИКОВА Елена Анатольевна

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ: ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ, СОРТИРОВКА И ПОИСК

Редактор В.В. Прокопова
Компьютерная верстка *И.О. Миняева*
Выпускающий редактор *Ю.А. Петропольская*

Подписано в печать
Формат 60x84 1/16. Бумага офсетная
Усл. 8,11 п. л. Уч.-изд. л.
Тираж 100 экз. Рег. №

Федеральное государственное
бюджетное образовательное учреждение
высшего профессионального образования
«Самарский государственный технический университет»
443100, г. Самара, ул. Молодогвардейская, 244. Главный корпус

Отпечатано в типографии
Самарского государственного технического университета
443100, г. Самара, ул. Молодогвардейская, 244. Корпус №8

Вопросы к экзамену по дисциплине

“Языки и методы программирования”

(весенний семестр)

1. Рекурсивные функции. Рекурсивная триада.
2. Анализ трудоемкости рекурсивных алгоритмов
3. Тип данных структуры
4. Передача структур в качестве параметров
5. Массивы структур
6. Абстрактные типы данных. Понятие класса и объекта.
7. Описание класса (спецификаторы доступа, свойства атрибутов класса)
8. Работа с объектами класса. Присвоение объектов. Указатель this.
9. Конструкторы и деструктор
10. Наследование
11. Инкапсуляция
12. Полиморфизм
13. Перегрузка операций
14. Абстрактные классы
15. Потоки. Методы обмена с потоками
16. Основные понятия о файлах. Обработка файловых потоков
17. Режимы открытия файлов. Обработка ошибок открытия файлов.
18. Указатель позиционирования в файле
19. Запись в текстовые файлы
20. Чтение из текстовых файлов
21. Запись в бинарные файлы
22. Чтение из бинарных файлов
23. Запись структуры в бинарный файл
24. Чтение структуры из бинарного файла
25. Алгоритмы сортировки. Оценка сложности. Классификация
26. Сортировка методом "пузырька" (простого обмена)
27. Шейкерная сортировка
28. Сортировка методом простого выбора (простой перебор)
29. Бинарная пирамидальная сортировка
30. Сортировка методом Шелла
31. Сортировка слиянием
32. Алгоритмы внешней сортировки

33. Алгоритмы поиска. Оценка сложности. Классификация.
34. Последовательный (линейный) поиск
35. Бинарный (двоичный) поиск
36. Алгоритмы поиска строк: прямой поиск
37. Алгоритм Кнута, Морриса и Пратта
38. Контейнерные классы: Векторы (vector)
39. Контейнерные классы: Векторы логических значений (vector <booi>)
40. Контейнерные классы: Двусторонние очереди (deque)
41. Контейнерные классы: Списки (list)
42. Контейнерные классы: Стеки (stack)
43. Ассоциативные контейнеры: Словари (map)
44. Ассоциативные контейнеры: Множества (set)
45. Итераторы
46. Потоковые итераторы
47. Функциональные объекты
48. Алгоритмы стандартной библиотеки: не изменяющие последовательные операции
49. Алгоритмы стандартной библиотеки: изменяющие последовательные операции
50. Алгоритмы стандартной библиотеки: операции сортировки
51. Алгоритмы стандартной библиотеки: бинарные операции поиска
52. Алгоритмы стандартной библиотеки: операции слияния
53. Алгоритмы стандартной библиотеки: операции отношений
54. Алгоритм std::find() и поиск элемента по значению
55. Алгоритм std::find_if() и поиск элемента с условием
56. Алгоритмы std::count()/std::count_if() и подсчёт вхождений элемента
57. Алгоритм std::sort() и пользовательская сортировка
58. Алгоритм std::for_each() и все элементы контейнера
59. Кучи. Алгоритмы стандартной библиотеки для работы с кучей
60. Битовые поля

Направление подготовки: 01.03.02 Прикладная математика и информатика
(шифр и наименование направления подготовки/специальности)
Профиль: Прикладная математика и информатика
(наименование профиля)
Дисциплина: Б1.О.02.07 Языки программирования
(шифр и наименование дисциплины в соответствии с учебным планом)

Номер задания	Содержание задания	Ключ к ответу	Тема РПД	Семестр(ы)
ОПК-2. Способен использовать и адаптировать существующие математические методы и системы программирования для разработки и реализации алгоритмов решения прикладных задач				
1.	Прочитайте текст вопроса и выберите правильный ответ. Определите вид алгоритма, в котором команды выполняются последовательно одна за другой без проверок условий и повторений. а) Циклический алгоритм б) Линейный алгоритм в) Рекурсивный алгоритм г) Разветвленный алгоритм	Б	Понятия алгоритма, исполнитель алгоритма, свойства алгоритмов	1
2.	Прочитайте текст вопроса и выберите несколько правильных ответов. Отметьте характеристики, присущие циклическому алгоритму. а) Наличие условия продолжения или окончания работы б) Однократное выполнение всех команд в) Многократное выполнение тела цикла г) Отсутствие переходов между командами д) Наличие тела цикла	а, в, д	Понятия алгоритма, исполнитель алгоритма, свойства алгоритмов	1
3.	Прочитайте текст вопроса и установите последовательность. Запишите соответствующую последовательность цифр слева направо. Укажите порядок действий при выполнении разветвляющегося алгоритма с условием. 1. Проверка условия 2. Выполнение команды после ветвления 3. Выбор ветви исполнения (Да/Нет) 4. Начало алгоритма	4, 1, 3, 2	Понятия алгоритма, исполнитель алгоритма, свойства алгоритмов	1
4.	Прочитайте текст вопроса и выберите правильный ответ. Укажите ключевое слово языка C++, используемое для организации множественного выбора из нескольких вариантов. а) if б) for в) switch г) while	В	Основные конструкции языка программирования C++	1
5.	Прочитайте текст вопроса и выберите несколько правильных ответов. Отметьте виды циклов, поддерживаемые языком программирования C++. а) Цикл с параметром (for) б) Цикл с предусловием (while) в) Цикл с произвольным шагом (loop) г) Цикл с постусловием (do-while) д) Рекурсивный цикл (recurse)	А, Б, Г	Основные конструкции языка программирования C++	1
6.	Прочитайте текст вопроса и установите последовательность. Запишите соответствующую последовательность цифр слева направо. Укажите порядок выполнения операций в цикле for (int i=0; i<5; i++). 1. Проверка условия i<5	2, 1, 3, 4	Основные конструкции языка программирования C++	1

<table>
<tr><td></td><td>2. Инициализация int i=0
3. Тело цикла
4. Инкремент i++</td><td></td><td></td><td></td></tr>
<tr><td>7.</td><td>Прочитайте текст и запишите ответ. Как называется цикл, выполняющийся до тех пор, пока условие истинно?</td><td>while</td><td>Основные конструкции языка программирования C++</td><td>1</td></tr>
<tr><td>8.</td><td>Прочитайте текст вопроса и установите соответствие. К каждой позиции, данной в левом столбце, подберите соответствующую позицию из правого столбца.
<table>
<tr><th>Оператор</th><th>Функция</th></tr>
<tr><td>А) Оператор break</td><td>1. Осуществляет безусловный переход на метку</td></tr>
<tr><td>Б) Оператор continue</td><td>2. Прерывает выполнение цикла и передает управление после него</td></tr>
<tr><td>В) Оператор goto</td><td>3. Прерывает текущую итерацию и переходит к следующей проверке условия</td></tr>
<tr><td>Г) Оператор return</td><td>4. Завершает выполнение функции</td></tr>
</table></td><td>
<table>
<tr><td>А</td><td>Б</td><td>В</td><td>Г</td></tr>
<tr><td>2</td><td>3</td><td>1</td><td>4</td></tr>
</table></td><td>Основные конструкции языка программирования C++</td><td>1</td></tr>
<tr><td>9.</td><td>Прочитайте текст вопроса и выберите правильный ответ.
Определите индекс первого элемента в одномерном массиве языка C++.
а) 1
б) 0
в) -1
г) Зависит от размера массива</td><td>Б</td><td>Обработка пользовательских типов данных: одномерные массивы</td><td>1</td></tr>
<tr><td>10.</td><td>Прочитайте текст вопроса и выберите несколько правильных ответов.
Отметьте способы объявления двумерного массива целых чисел размером 3 на 3.
а) int arr[3][3];
б) int arr[3, 3];
в) int arr[][3] = {{1,2,3},{4,5,6},{7,8,9}};
г) int arr[3][] = {{1,2,3},{4,5,6},{7,8,9}};</td><td>а, в</td><td>Обработка пользовательских типов данных: одномерные массивы</td><td>1</td></tr>
<tr><td>11.</td><td>Прочитайте текст вопроса и установите последовательность. Запишите соответствующую последовательность цифр слева направо.
Укажите порядок доступа к элементам при построчной обработке двумерного массива.
1. Увеличение индекса столбца
2. Сброс индекса столбца в 0
3. Увеличение индекса строки
4. Проверка выхода за границы столбца</td><td>1, 4, 2, 3</td><td>Обработка пользовательских типов данных: одномерные массивы</td><td>1</td></tr>
<tr><td>12.</td><td>Прочитайте текст и запишите ответ. Какой тип данных в языке C++ имеет диапазон значений от -32767 до 32768 (16 бит со знаком)?</td><td>short</td><td>Обработка пользовательских типов данных: одномерные массивы</td><td>1</td></tr>
<tr><td>13.</td><td>Прочитайте текст вопроса и выберите несколько правильных ответов. Какие способы индексации применяются для обращения к элементам двумерного массива?
а) Прямой доступ по индексам
б) Через итерацию циклом
в) Через указатель
г) По имени элемента</td><td>А, б, в</td><td>Обработка пользовательских типов данных: одномерные массивы</td><td>1</td></tr>
<tr><td>14.</td><td>Прочитайте текст и запишите ответ.
Определите количество элементов двумерного массива int arr[3][4].</td><td>12</td><td>Обработка пользовательских типов данных: одномерные массивы</td><td>1</td></tr>
<tr><td>15.</td><td>Прочитайте текст вопроса и выберите правильный ответ. Для какого действия используется операция * перед именем указателя?</td><td>В</td><td>Работа с динамической памятью</td><td>1</td></tr>
</table>

	а) Получение адреса переменной б) Деструкция объекта в) Разыменование указателя г) Преобразование типа			
16.	Прочитайте текст вопроса и выберите правильный ответ. Укажите оператор языка С++, предназначенный для выделения памяти в динамической области (куче). а) malloc б) alloc в) new г) create	В	Работа с динамической памятью	1
17.	Прочитайте текст вопроса и выберите несколько правильных ответов. Отметьте операции, допустимые над указателями одного типа. а) Сложение двух указателей б) Вычитание одного указателя из другого в) Сравнение указателей г) Прибавление целого числа к указателю д) Умножение указателя на число	б, в, г	Работа с динамической памятью	1
18.	Прочитайте текст вопроса и выберите несколько правильных ответов. Отметьте функции библиотеки string.h, предназначенные для работы со строками. а) strlen б) printf в) strcpy г) strcmp д) cin	а, в, г	Строки в С++	1
19.	Прочитайте текст вопроса и выберите правильный ответ. Укажите область видимости локальной переменной, объявленной внутри функции. а) Вся программа б) Только блок функции, в которой она объявлена в) Все функции проекта г) Только главная функция main	Б	Параметры функций.	1
20.	Прочитайте текст вопроса и выберите несколько правильных ответов. Отметьте способы передачи параметров в функцию в языке С++. а) Передача по значению б) Передача по ссылке в) Передача по указателю г) Передача по имени д) Передача по потоку	а, б, в	Параметры функций.	1
ОПК-5 Способен разрабатывать алгоритмы и компьютерные программы, пригодные для практического применения				
1.	Прочитайте текст вопроса и выберите правильный ответ. Укажите ключевое слово, используемое для объявления структуры в С++. а) class б) union в) struct г) type	В	Пользовательский тип данных структура.	2
2.	Прочитайте текст вопроса и выберите несколько правильных ответов. Отметьте типы данных, которые могут входить в состав одной структуры. а) int б) double в) char массив г) Другая структура д) Сама структура (рекурсивно без указателя)	а, б, в, г	Пользовательский тип данных структура.	2
3.	Прочитайте текст вопроса и установите	А Б В	Пользовательский тип	2

<table>
<tr><td></td><td>соответствие. К каждой позиции, данной в левом столбце, подберите соответствующую позицию из правого столбца.
<table><tr><th>Доступ</th><th>Реализация</th></tr><tr><td>А) Доступ через объект</td><td>1. Используется оператор стрелки -></td></tr><tr><td>Б) Доступ через указатель на объект</td><td>2. Используется оператор точки .</td></tr><tr><td>В) Массив структур</td><td>3. Доступ по индексу и имени поля</td></tr></table></td><td><table><tr><td>2</td><td>1</td><td>3</td></tr></table></td><td>данных структура.</td><td></td></tr>
<tr><td>4.</td><td>Прочитайте текст вопроса и выберите правильный ответ.
Определите класс потока, используемый для записи данных в файл.
а) ifstream
б) ofstream
в) fstream
г) cin</td><td>Б</td><td>Файлы в C++.</td><td>2</td></tr>
<tr><td>5.</td><td>Прочитайте текст вопроса и выберите несколько правильных ответов.
Отметьте режимы открытия файла в C++.
а) ios::in
б) ios::out
в) ios::binary
г) ios::execute
д) ios::compile</td><td>а, б, в</td><td>Файлы в C++.</td><td>2</td></tr>
<tr><td>6.</td><td>Прочитайте текст вопроса и установите последовательность. Запишите соответствующую последовательность цифр слева направо.
Укажите порядок операций при записи в файл.
1. Открытие файла
2. Проверка успешности открытия
3. Запись данных
4. Закрытие файла</td><td>1, 2, 3, 4</td><td>Файлы в C++.</td><td>2</td></tr>
<tr><td>7.</td><td>Прочитайте текст и запишите ответ.
Напишите команду подключения заголовочного файла для работы с файловыми потоками.</td><td>#include <fstream></td><td>Файлы в C++.</td><td>2</td></tr>
<tr><td>8.</td><td>Прочитайте текст вопроса и выберите правильный ответ.
Укажите уровень доступа по умолчанию для членов класса (class) в C++.
а) public
б) protected
в) private
г) friend</td><td>В</td><td>Соотношение основных понятий ООП</td><td>2</td></tr>
<tr><td>9.</td><td>Прочитайте текст вопроса и выберите несколько правильных ответов.
Отметьте спецификаторы доступа, используемые в классах C++.
а) public
б) private
в) protected
г) internal
д) open</td><td>а, б, в</td><td>Соотношение основных понятий ООП</td><td>2</td></tr>
<tr><td>10.</td><td>Прочитайте текст вопроса и установите соответствие. К каждой позиции, данной в левом столбце, подберите соответствующую позицию из правого столбца.
<table><tr><th>Принцип ООП</th><th>Описание</th></tr><tr><td>А) Инкапсуляция</td><td>1. Выделение главных, наиболее значимых характеристик объекта и игнорирование второстепенных</td></tr><tr><td>Б) Наследование</td><td>2. Способность объектов с одинаковыми именами методов реализовывать</td></tr></table></td><td><table><tr><td>А</td><td>Б</td><td>В</td><td>Г</td></tr><tr><td>3</td><td>4</td><td>2</td><td>1</td></tr></table></td><td>Соотношение основных понятий ООП</td><td>2</td></tr>
</table>

		различное поведение			
	В) Полиморфизм	3. Объединение данных и методов, работающих с этими данными, внутри одного типа, а также скрытие внутренней реализации объекта			
	Г) Абстракция	4. Механизм, позволяющий создать новый тип (потомок) на основе существующего (родитель), заимствуя его поля и методы			
11.	Прочитайте текст вопроса и выберите правильный ответ. Определите имя конструктора класса. а) Произвольное имя б) Имя, совпадающее с именем класса в) Имя constructor г) Имя init		Б	Конструкторы и деструкторы	2
12.	Прочитайте текст вопроса и выберите несколько правильных ответов. Отметьте характеристики деструктора. а) Имеет имя, совпадающее с именем класса б) Имеет тильду (~) перед именем в) Не имеет параметров г) Не имеет возвращаемого типа д) Вызывается явно пользователем		б, в, г	Конструкторы и деструкторы	2
13.	Прочитайте текст вопроса и установите последовательность. Запишите соответствующую последовательность цифр слева направо. Укажите порядок вызова специальных методов при создании и уничтожении объекта. 1. Вызов конструктора 2. Выделение памяти 3. Выполнение методов объекта 4. Вызов деструктора 5. Освобождение памяти		2, 1, 3, 4, 5	Конструкторы и деструкторы	2
14.	Прочитайте текст вопроса и выберите несколько правильных ответов. Какие правила действуют при работе с объектами класса? а) Класс может иметь несколько конструкторов б) Каждый класс обязан иметь конструктор в) Все классы обязаны иметь деструктор г) Методы класса могут обращаться к приватным членам класса		А, Г	Конструкторы и деструкторы	2
15.	Прочитайте текст вопроса и выберите правильный ответ. Какой основной принцип построения линейного односвязного списка? а) Каждый узел содержит ссылку на предыдущий элемент б) Узлы расположены последовательно в памяти в) Каждый узел содержит ссылку на следующий элемент г) Узлы хранятся в случайном порядке		В	Контейнерные классы.	2
16.	Прочитайте текст и запишите ответ. Какой стандартный поток вывода используется в C++?		cout или std::cout	Потоковые классы.	2
17.	Прочитайте текст вопроса и выберите правильный ответ. Определите алгоритм сортировки, основанный на последовательном сравнении и обмене соседних элементов. а) Сортировка выбором б) Сортировка вставками в) Сортировка пузырьком г) Быстрая сортировка		В	Алгоритмы сортировки.	2

18.	Прочитайте текст вопроса и установите последовательность. Запишите соответствующую последовательность цифр слева направо. Укажите этапы одного прохода сортировки выбором для поиска минимума. 1. Сравнение текущего элемента с минимальным 2. Предположение, что первый несортированный элемент минимален 3. Обмен найденного минимума с первым несортированным 4. Переход к следующему элементу	2, 1, 4, 3	Алгоритмы сортировки	2
19.	Прочитайте текст вопроса и выберите правильный ответ. Определите обязательное условие для применения алгоритма бинарного поиска. а) Массив должен быть заполнен нулями б) Массив должен быть отсортирован в) Массив должен быть связным списком г) Массив должен быть двумерным	Б	Стандартные алгоритмы сортировки и поиска	2
20.	Прочитайте текст вопроса и выберите несколько правильных ответов. Отметьте характеристики алгоритма последовательного (линейного) поиска. а) Применим к любым данным б) Требует предварительной сортировки в) Просматривает элементы по очереди г) Имеет сложность O(log n) д) Имеет сложность O(n) в худшем случае	а, в, д	Стандартные алгоритмы сортировки и поиска	2