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

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).

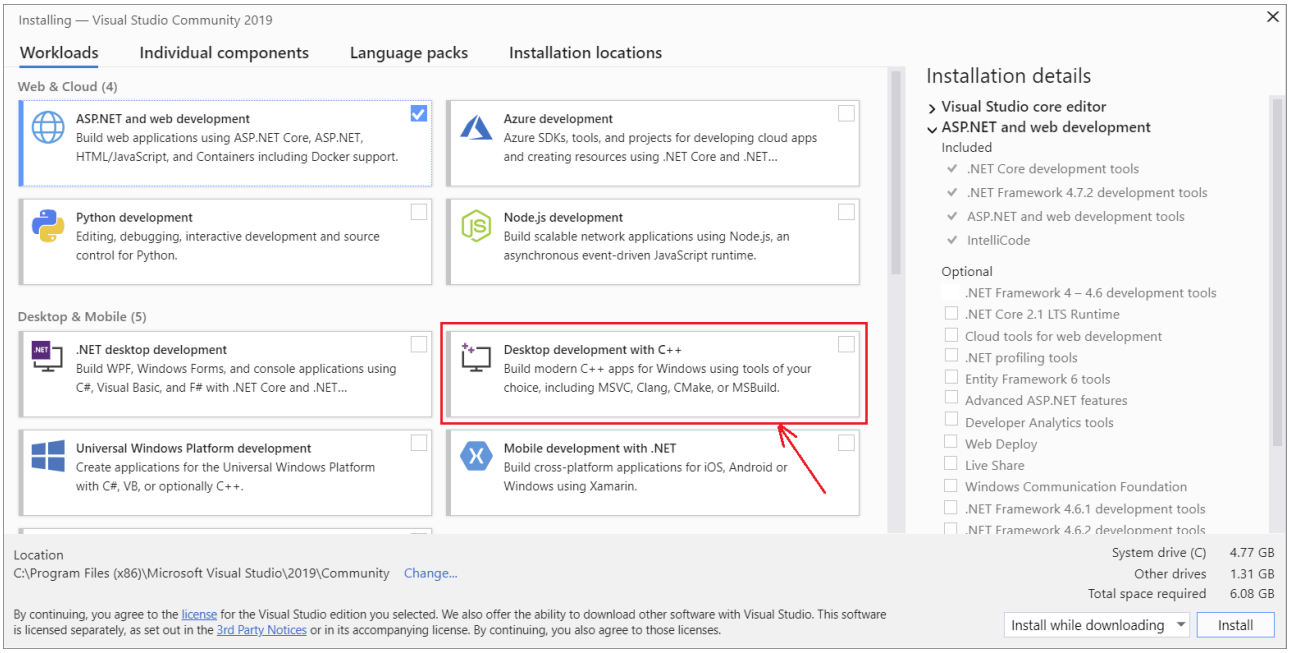


Рис. 1. Окно выбора рабочей нагрузки.

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).

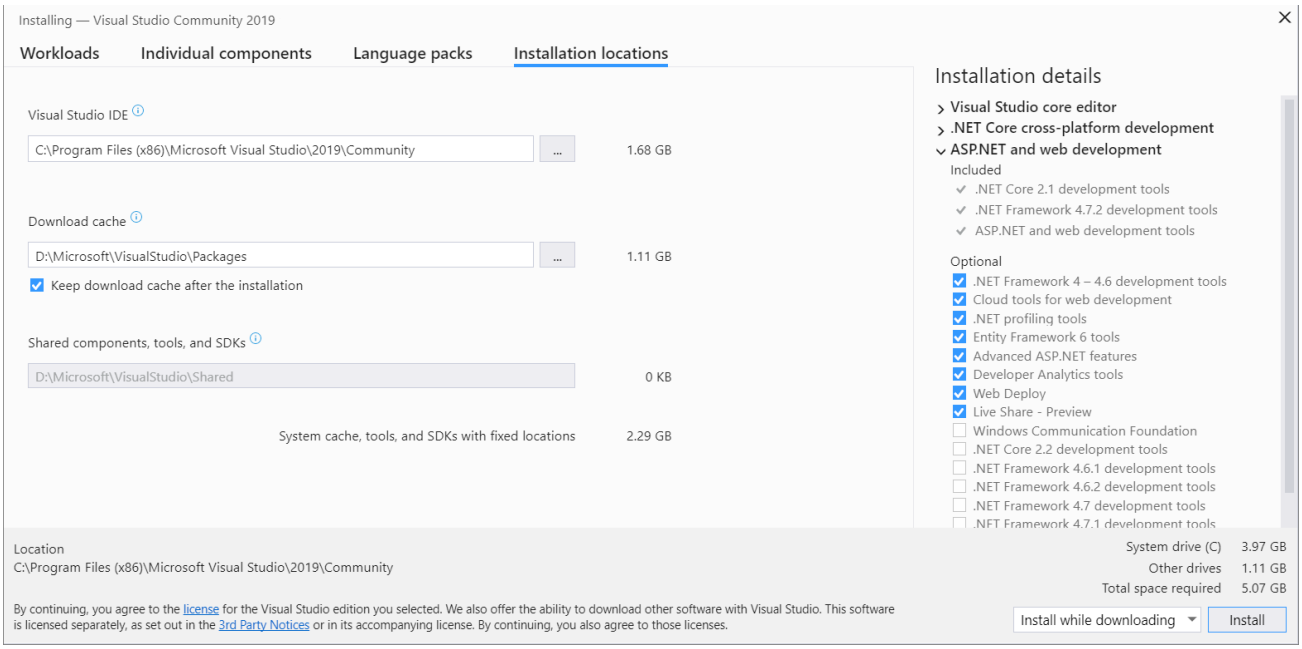


Рис. 2. Окно выбора расположения компонентов.

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».

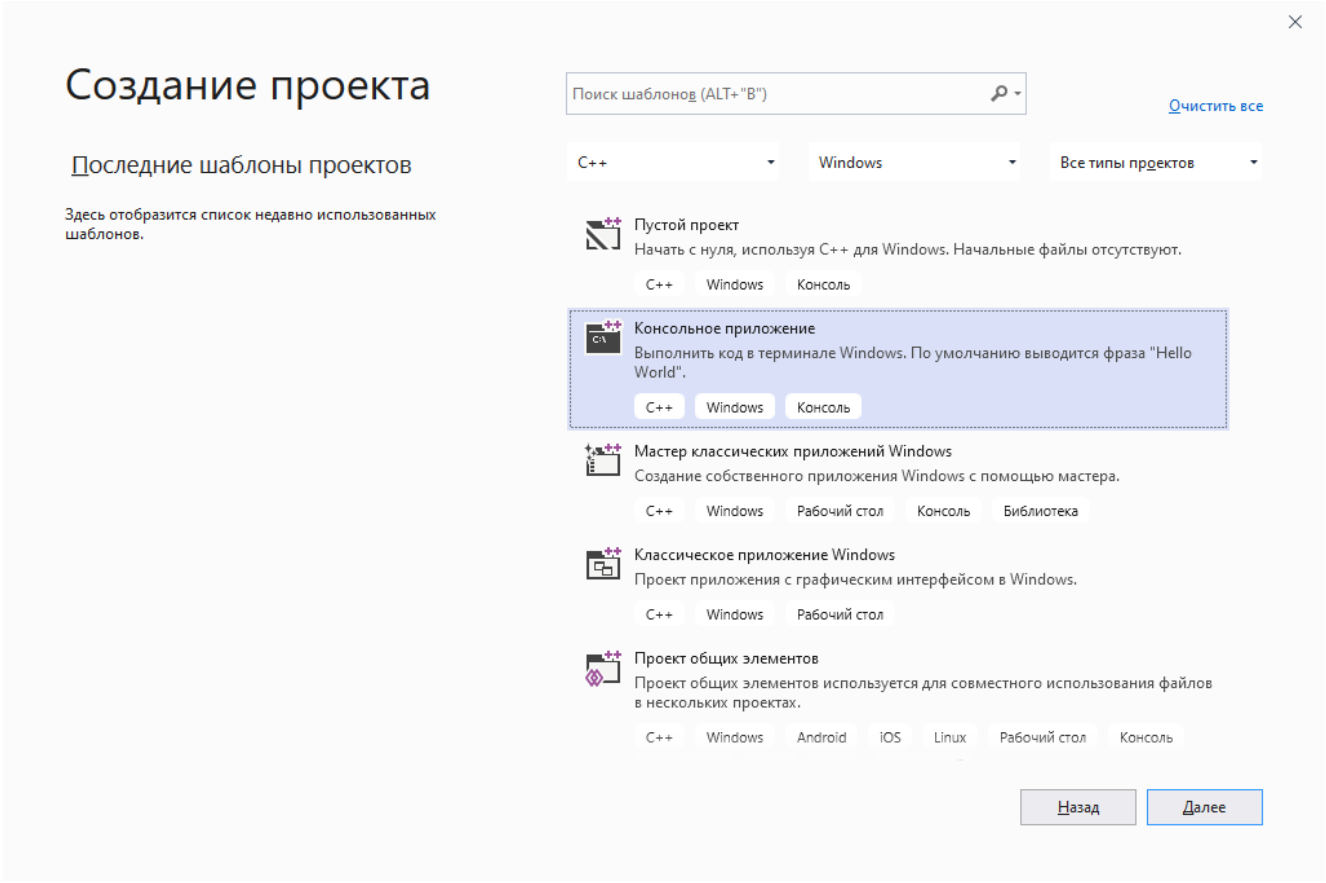


Рис. 3. Окно выбора типа проекта

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

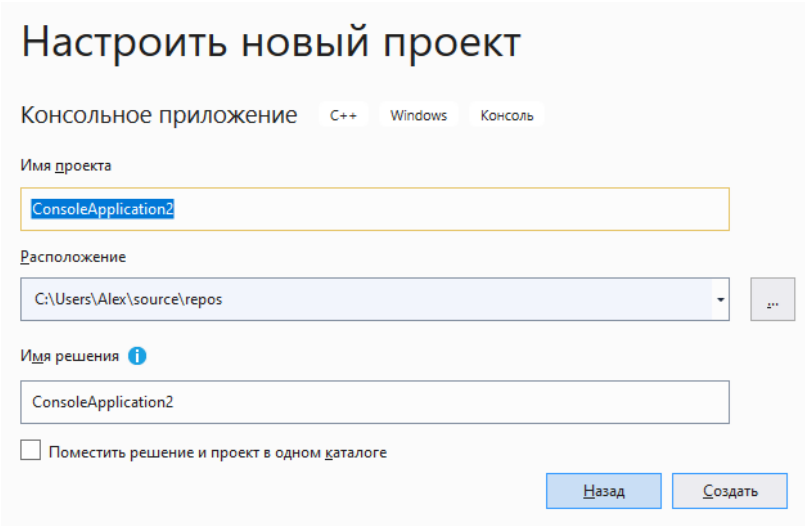


Рис. 4. Окно настройки проекта

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.

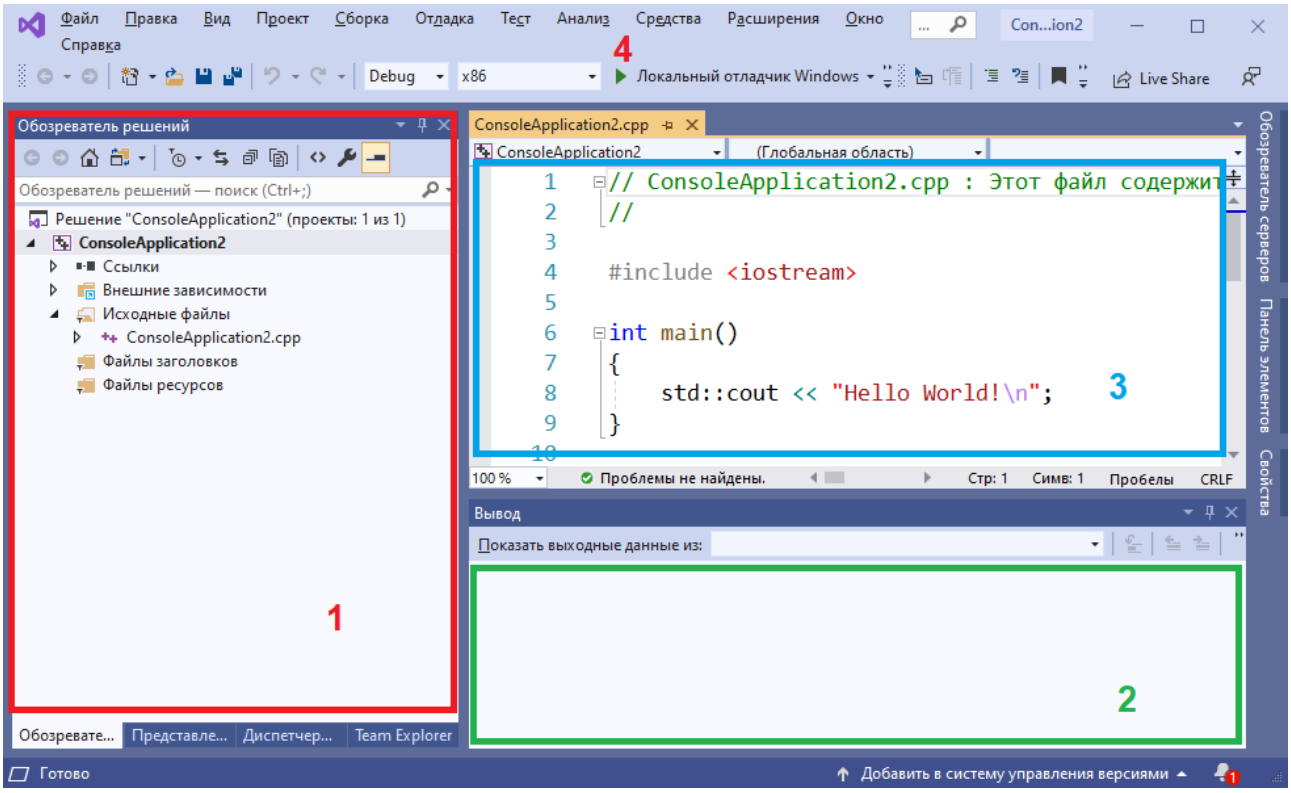


Рис. 5. Внешний вид *Visual Studio*

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).

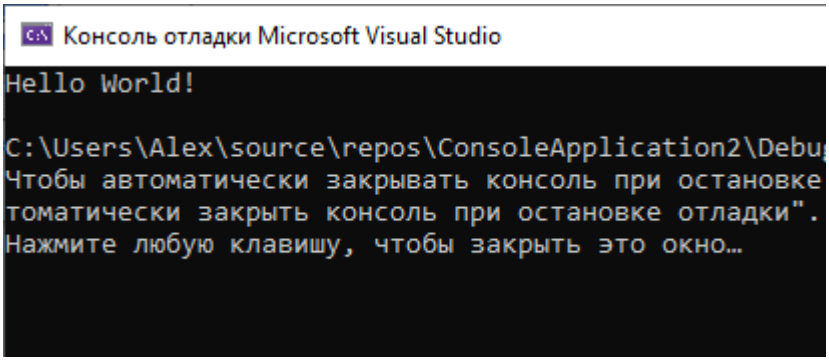


Рис. 6. Фрагмент окна командной строки после выполнения программы

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).

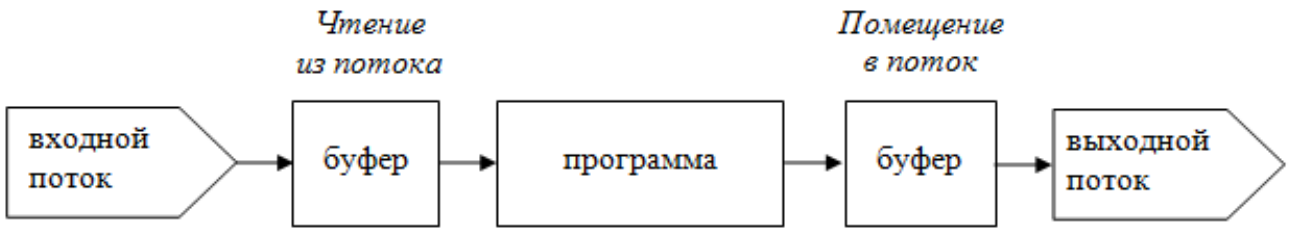


Рис. 7. Схема организации буферизации в операциях ввода-вывода

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

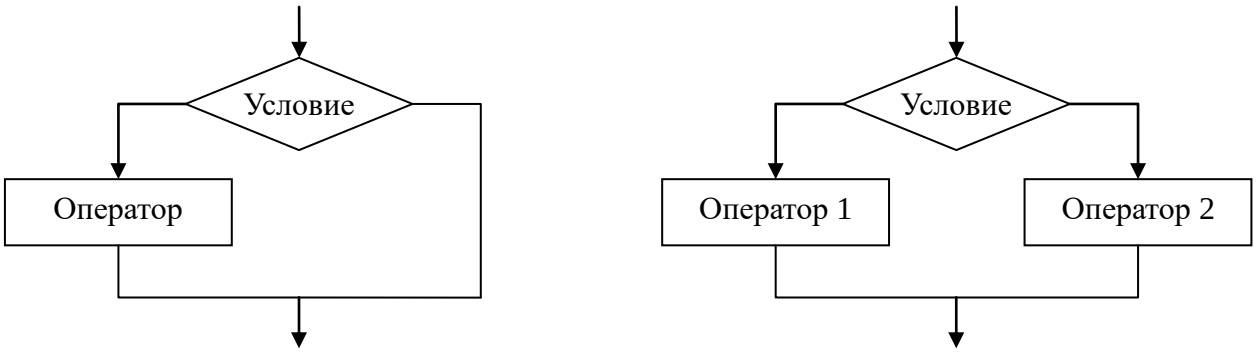
```
if (0 == 1) cout << «0 равно 1»;
```

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:

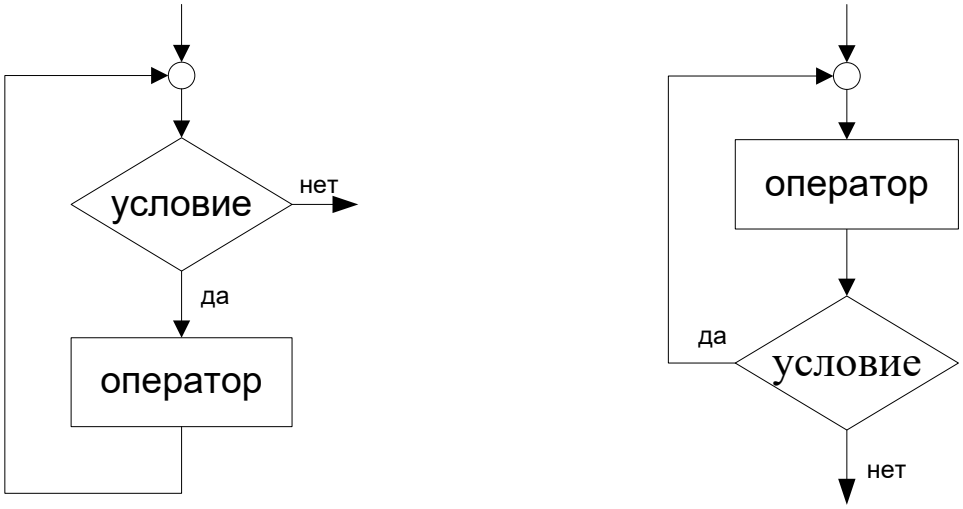
```
пока условие
 повторить
      оператор
```

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 C++:

```
#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 C++.

```
#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).

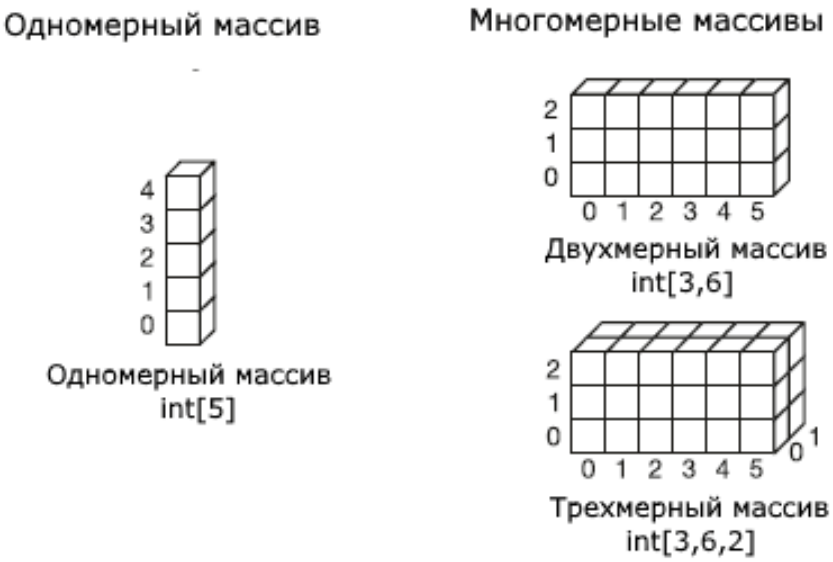


Рис. 13. Визуальное представление массивов

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.

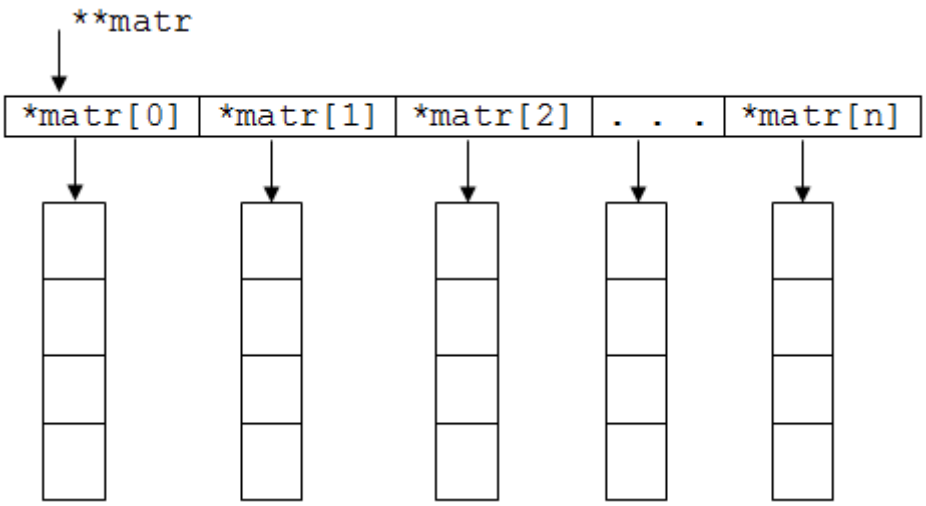


Рис. 14. Схема динамической области памяти,
выделенной под двумерный массив

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).

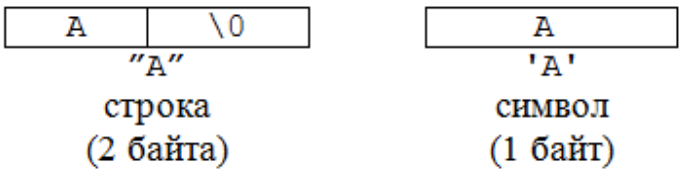


Рис. 15. Выделение памяти под строковые и символьные литералы

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 С++:

```
#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.

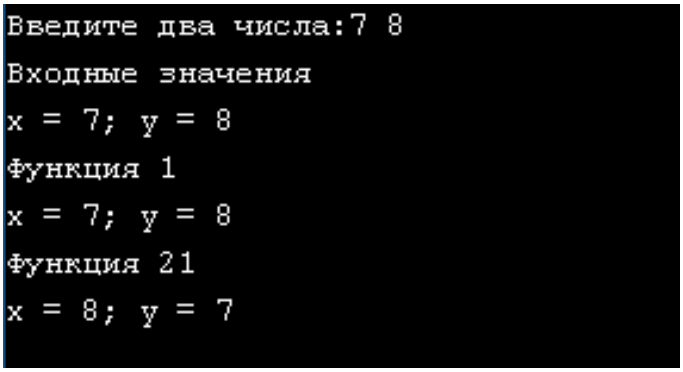


Рис. 16. Результат работы программы из примера 6

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

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