МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Воронежский государственный технический университет»

Кафедра радиотехники

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ НА ЭВМ»

для обучающихся по направлению
11.03.01 «Радиотехника»,
профиль «Радиотехнические средства передачи, приема и обработки сигналов» всех форм обучения

Воронеж 2021

Лабораторная работа №1
Знакомство с интегрированной средой разработки Code::Blocks и изучение структуры программы на языке Си++

Цель работы: знакомство с программным пакетом Code::Blocks. Приобретение навыков создания простых программ на языке программирования С++. Ознакомление со структурой программы. Запуск программы на исполнение. Поиск ошибок в тексте программы. Изучение команд ввода-вывода.

Теоретические сведения

1. Основы работы в Code::Blocks

Интегрированная среда разработки (ИСР) (англ. IDE, Integrated Development Environment или Integrated Debugging Environment) - система программных средств, используемая программистами для разработки программного обеспечения (ПО).

Обычно среда разработки включает в себя:

1. **Текстовый редактор**, который используется для ввода исходного кода программы.

2. **Компилятор** и / или интерпретатор. Компилятор - это программа, транслирующая исходный (высокоуровневый) код программы (файлы *.c для языка С, *.cpp для языка С++) в конечный (низкоуровневый) код. Процесс компиляции - это процесс преобразования высокоуровневого исходного текста программы в эквивалентный текст программы, но уже на низкоуровневом языке (машинном языке).

3. **Средства автоматизации сборки**.

4. **Отладчик** (debugger). Это программа, позволяющая исследовать внутреннее устройство разрабатываемой программы. Отладчик обеспечивает пошаговое исполнение программы, просмотр текущих значений переменных, вычисление значения любого выражения программы и другие функции.

Работа в интегрированной среде дает программисту следующие преимущества:

- возможность использования встроенного многофайлового текстового редактора, специально ориентированного на работу с исходными текстами программ;
- диагностика выявленных при компиляции ошибок, и исходный текст программы, доступный редактированию, выводятся одновременно в многооконном режиме;
- возможность организации и ведения параллельной работы над несколькими проектами. Менеджер проектов позволяет использовать любой проект в качестве шаблона для вновь создаваемого проекта;
- перекомпиляции подвергаются только редактировавшиеся модули;

- возможность загрузки отлаживаемой программы в имеющиеся средства отладки, и работы с ними без выхода из оболочки;

- возможность подключения к оболочке практически любых программных средств.

Существуют следующие ИСР для языка C++: Anjuta, C++ Builder, Code::Blocks, Codeforge, CodeLite, Dev-C++, Eclipse CDT, Geany, GNAT Programming Studio, KDevelop, Kuzya, Microsoft Visual Studio, Microsoft Visual Studio Express, MonoDevelop, NetBeansC/C++ pack, Open Watcom, Pelles CQt, CreatorRational, Software Architect, Sun Studio,Turbo C++ Explorer, Turbo C++ Professional, Ultimate++, wxDev-C++, Xcode. Сравнительный анализ сред разработки производят по разным критериям: лицензия (например, универсальная общественная лицензия GNU, GPL; проприентарная, т.е. патентованная и т.д.), возможность установки на разные платформы (Windows, Linux, McOC и др.) и т.д. Для перечисленных ИСР такой анализ можно найти в различных источниках (например, в Википедии).

Code::Bloks - свободная кроссплатформенная интегрированная среда разработки. Code::Blocks написана на Си++ и использует библиотеку wxWidgets. Имея открытую архитектуру, может масштабироваться за счёт подключаемых модулей. Поддерживает языки программирования C, C++, D (с ограничениями). Code::Blocks разрабатывается для Windows, Linux и Mac OS X. Распространяется по лицензии GNU GPL.

Пакет представляет собой интегрированную программную среду, обеспечивающую возможность набора и редактирования текста программы на языке программирования C или C++, ее компиляцию и отладку. Тем самым существенно облегчается работа программиста по созданию приложений.

На сайте www.codeblocks.org ИСР Code::Blocks доступна для скачивания.

К Code::Blocks можно подключать различные компиляторы через удобный интерфейс. Для разных проектов можно подключить разные компиляторы, для одного и того же проекта можно использовать различные компиляторы. Последнее удобно при разработке open source-проектов, которые должны компилироваться всеми распространенными компиляторами. Список компиляторов, с которыми умеет работать Code::Blocks: GNU GCC (Linux), MinGW GCC (Win32), Microsoft's Visual C++ Free Toolkit 2003 (Win32), Borland's C++ Compiler 5.5 (Win32), DigitalMars (Win32), OpenWatcom (Win32), Small Device C Compiler (SDCC).

2. Структура программы на языке C++

Сама по себе программа на языке C++ представляет собой текстовый файл, в котором представлены конструкции и операторы данного языка в заданном программистом порядке. В самом простом случае этот текстовый файл может содержать такую информацию:

```
// программа вывода на экран текстового сообщения
/* при таком способе написания комментария он может быть
   многострочным*/

#include <iostream>
    using namespace std;
    int main()
    {
       cout << "Hello, world!" << endl;
       return 0;
    }
```

Обычно файл имеет расширение cpp, например, «ex1.cpp».

Следующий шаг – это компиляция исходного кода. Под компиляцией понимают процесс, при котором содержимое текстового файла преобразуется в исполняемый машинный код, понимаемый процессором компьютера. Однако компилятор создает не готовую к исполнению программу, а только объектный код (файл с расширением *.obj). Этот код является промежуточным этапом при создании готовой программы. Дело в том, что создаваемая программа может содержать функции стандартных библиотек языка C++, реализации которых описаны в объектных файлах библиотек. Это означает, что объектный файл (например, ex1.obj) будет содержать лишь инструкции по вызову данной функции, но код самой функции в нем будет отсутствовать.

Для того чтобы итоговая исполняемая программа содержала все необходимые реализации функций, используется компоновщик объектных кодов. Компоновщик – это программа, которая объединяет в единый исполняемый файл объектные коды создаваемой программы, объектные коды реализаций библиотечных функций и стандартный код запуска для заданной операционной системы. В итоге и объектный файл, и исполняемый файл состоят из инструкций машинного кода. Однако объектный файл содержит только результат перевода на машинный язык текста программы, созданной программистом, а исполняемый файл – также и машинный код для используемых стандартных библиотечных подпрограмм и для кода запуска.

Рассмотрим более подробно пример программы.

Первые строки задают комментарии, т.е. замечания, помогающие лучше понять программу. Они предназначены только для чтения и игнорируются компилятором. Комментарии помогают человеку читать текст программы; писать их грамотно считается правилом хорошего тона. Комментарии могут характеризовать используемый алгоритм, пояснять назначение тех или иных переменных, разъяснять непонятные места. При компиляции комментарии выкидываются из текста программы поэтому размер получающегося исполняемого модуля не увеличивается.

В C++ есть два типа комментариев.

1) Комментарий, использующий символы /* для обозначения начала и */ для обозначения конца комментария. Между этими парами символов может находиться любой текст, занимающий одну или несколько строк: вся последовательность между /* и */ считается комментарием. Комментарии не могут быть вложенными.

2) Однострочный комментарий. Он начинается последовательностью символов // и ограничен концом строки. Часть строки вправо от двух косых черт игнорируется компилятором.

Обычно в программе употребляют сразу оба типа комментариев. Строчные комментарии удобны для кратких пояснений – в одну или полстроки, а комментарии, ограниченные /* и */, лучше подходят для развернутых многострочных пояснений.

Далее в программе записана директива препроцессора. Заголовочные файлы включаются в текст программы с помощью директивы препроцессора #include. Директивы препроцессора начинаются со знака "диез" (#), который должен быть самым первым символом строки. Программа, которая обрабатывает эти директивы, называется препроцессором (в современных компиляторах препроцессор обычно является частью самого компилятора). Директива #include включает в программу содержимое указанного файла. Имя файла может быть указано двумя способами:

#include <some_file.h>

#include "my_file.h"

Если имя файла заключено в угловые скобки (<>), считается, что нам нужен некий стандартный заголовочный файл, и компилятор ищет этот файл в предопределенных местах. (Способ определения этих мест сильно различается для разных платформ и реализаций.) Двойные кавычки означают, что заголовочный файл - пользовательский, и его поиск начинается с того каталога, где находится исходный текст программы. Заголовочный файл также может содержать директивы #include. Поэтому иногда трудно понять, какие же конкретно заголовочные файлы включены в данный исходный текст, и некоторые заголовочные файлы могут оказаться включенными несколько раз. Избежать этого позволяют условные директивы препроцессора, например, #ifndef, #define и т.д. В данном примере используется библиотека iostream, которая является частью стандартной библиотеки C++ и которая реализована как иерархия классов и обеспечивает базовые возможности ввода/вывода.

В третьей строке определена функция с именем main, которая возвращает целое число (тип int) и не принимает никаких аргументов (тип void). Функция main() является обязательной функцией для всех программ на языке C++ и без ее наличия уже на этапе компиляции появляется сообщение об ошибке, указывающее на отсутствие данной функции. Обязательность данной функции обусловливается тем, что она является точкой входа в программу. В данном случае под точкой входа понимается функция, с которой начинается и которой заканчивается работа программы. Например, при запуске exe-файла происходит ак-

тивизация функции main(), выполнение всех операторов, входящих в нее и завершение программы. Таким образом, логика всей программы заключена в этой функции.

В приведенном примере при вызове функции main() происходит вызов функции cout, которая выводит на экран монитора сообщение “Hello World!”, а затем выполняется оператор return, который возвращает нулевое значение. Это число возвращается самой функцией main() операционной системе и означает успешное завершение программы.

Фигурные скобки {} служат для определения начала и конца тела функции, т.е. в них содержатся все возможные операторы, которые описывают работу данной функции.

Следует отметить, что после каждого оператора в языке C++ ставится символ ‘;’.

3. Текст на кириллице в консоли

Самый лучший способ изучения языка программирования C++ — это составление консольных программ. Структура консольного проекта максимально упрощена, так как нет графического режима, для которого необходимо подключение файлов ресурсов, классов и т.д. При составлении программы может понадобиться вывести некоторое текстовое сообщение в консоль. И если это сообщение написано на латинице, то в командной строке Windows оно будет отображаться корректно. А если текстовое сообщение написано на кириллице, то вместо передаваемого сообщения, будет отображаться непонятная последовательность букв и символов.

Это происходит из-за представления символов букв в компьютерах.

Природа вычислительных машин такова, что они могут работать только с числами. Поэтому для представления букв или символов необходимо их закодировать, то есть каждой букве или символу присвоить определённое число, которое будет являться его кодом. Так образовались таблицы кодирования символов. В связи с тем, что в мире существует более 2,5 тысяч языков, то для каждого алфавита создавались свои таблицы кодирования символов, вот почему существует большое количество таблиц кодирования символов. Так как мы программируем под Windows, то нас будут интересовать такие кодировочные таблицы: **cp866, cp1251** и **utf-8**(стандарт Unicode). Хотя уже давно разработан единый стандарт кодирования символов — **Unicode**, в Windows до сих пор используются несколько кодировочных таблиц, а именно — **cp866**, **cp1251.** Использование нескольких таблиц кодирования символов и является причиной появления неправильных символов вместо корректного сообщения.

Так уж повелось, в командной строке Windows кодировка символов соответствует стандарту **cp866**. То есть все символы в командной стро-

ке **Windows** закодированы по кодировочной таблице **cp866**. Причём поменять кодировку в командной строке Windows нельзя.

Во всех русскоязычных Windows кодировка **cp1251** является стандартной 8 — битной кодировкой. И при создании проекта этот стандарт кодирования символов наследуется проектом, то есть программой. Хотя кодировку для проекта можно поменять, это не решает проблемы, так как консоль понимает только одну кодировку **cp866**. В итоге получается, что программа передаёт коды символов сообщения стандарта **cp1251**. Командная строка принимает эти коды и переводит их в символы, но уже по стандарту **cp866**, так как другого стандарта не знает. В итоге сообщение передано в консоль, но символы интерпретированы не правильно, вот так и появляются некорректные символы.

Решить данную проблему можно только одним способом — перед тем, как передать текст в консоль, необходимо его перекодировать в стандарт кодирования символов **cp866**. Существует несколько способов преобразования кодов знаков из одного стандарта в другой, мы воспользуемся самым простым — настройка локали.

Локаль — это набор параметров: набор символов, язык пользователя, страна, часовой пояс и др. Локаль необходима для быстрой настройки пользовательского интерфейса, в зависимости от географического положения. В C++ есть функция setlocale(), которая выполняет перекодировку символов в соответствии с требуемым языком.

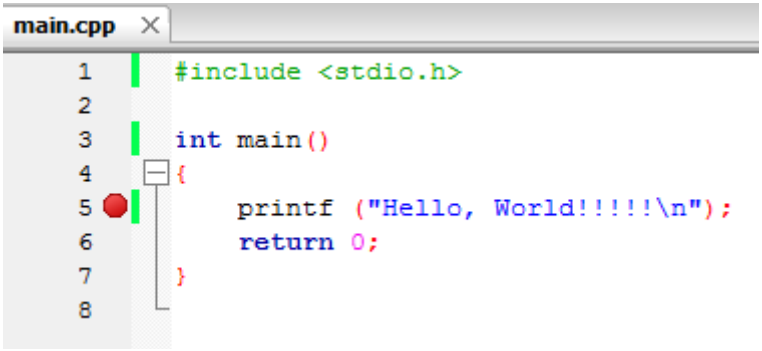
```
#include <iostream>
using namespace std;
int main()
{
    setlocale(LC_ALL, "Russian");
    cout << "Русский текст в консоли" << endl;
    return 0;
}
```

Чтобы излишне не усложнять выполнение заданий всех последующих лабораторных работ, допускается в процессе ввода-вывода использовать транслитерацию.

4. Организация ввода-вывода в Си++

Приложение, написанное на любом языке программирования, должно взаимодействовать с окружающим миром. Как правило, такое взаимодействие осуществляется посредством ввода-вывода информации на монитор или в файл. В стандартном Си++ существует два основных пути ввода-вывода информации:

1. Посредством традиционной системы ввода-вывода, унаследованной от C. Для реализации этого подхода необходимо включить заголовочный файл библиотеки <stdio.h>(std – стандартный, io- ввод-вывод). Для вывода на экран используется функция printf(), для ввода с клавиатуры – scanf().



2. С помощью потоков, реализованных в STL(Standart Template Library). Для использования объектно-ориентированного консольного ввода-вывода с помощью потоков (stream), в программу необходимо включить заголовочный файл <iostream>, а для файлового - <fstream>. В результате при запуске консольного приложения автоматически создаются виртуальные каналы связи, например, cin – для ввода с клавиатуры, cout – для буферизированного вывода на монитор, а также операции помещения в поток << и чтения из потока >>.

Для ввода данных используется *стандартный поток ввода* ***cin***, привязанный к стандартному устройству ввода данных (обычно к клавиатуре). Операция

cin >> i;

осуществляет передачу значения переменной i из объекта ***cin*** в память. Тип переменной при этом определяется автоматически.

Для вывода данных используется *стандартный поток вывода* ***cout***, привязанный к стандартному устройству вывода данных (обычно к дисплею). Операция

cout << i;

выводит значение переменной из памяти на экран.

Например, нужно вывести n чисел на экран:

```
#include <iostream.h>
void main()
    {
    int n = 5;

    for (int i = 1; i <=n; i++)
          cout << i;
    }
```

Результат работы программы будет следующий:

12345

В выходные данные можно включить параметры табуляции. Для этого предназначена функция установки ширины поля ***cout.width(int i)***. Она задает

ширину поля, отводимого под следующее выводимое на экран значение (при меньшей ширине автоматически будут добавлены пробелы).

Для перевода курсора на новую строку можно использовать манипулятор форматирования ***endl*** или символ «***n**».

Модифицируем программу следующим образом:

```
#include <iostream.h>
void main()
    {
    int n;
    cout << "**************************\n";
    cout << "Введите количество чисел:   ";
    cin >> n;
    cout << endl;

    for (int i = 1; i <=n; i++)
        {
        cout.width(4);
        cout << i;
        }

    cout << endl;
    cout << "**************************\n";
    }
```

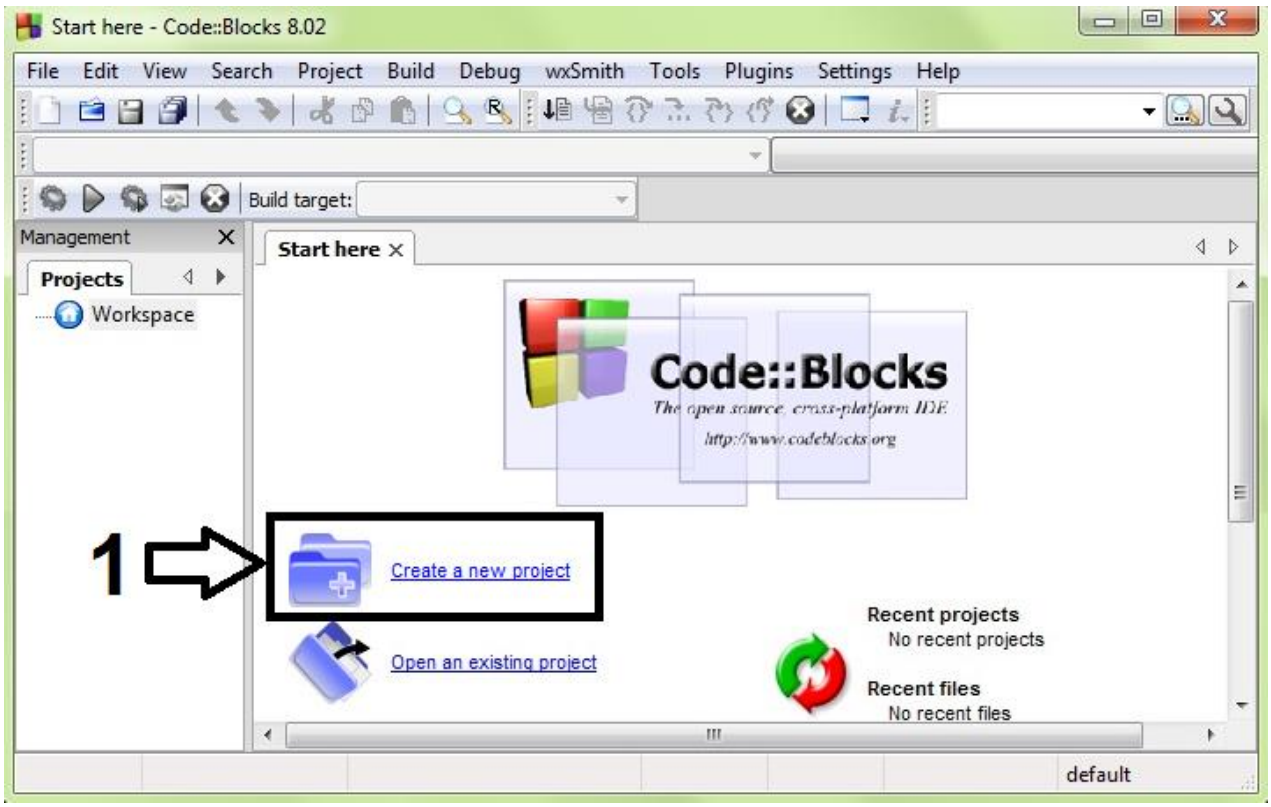
Результат работы программы:

```
**************************
Введите количество чисел:  6
   1   2   3   4   5   6
**************************
```

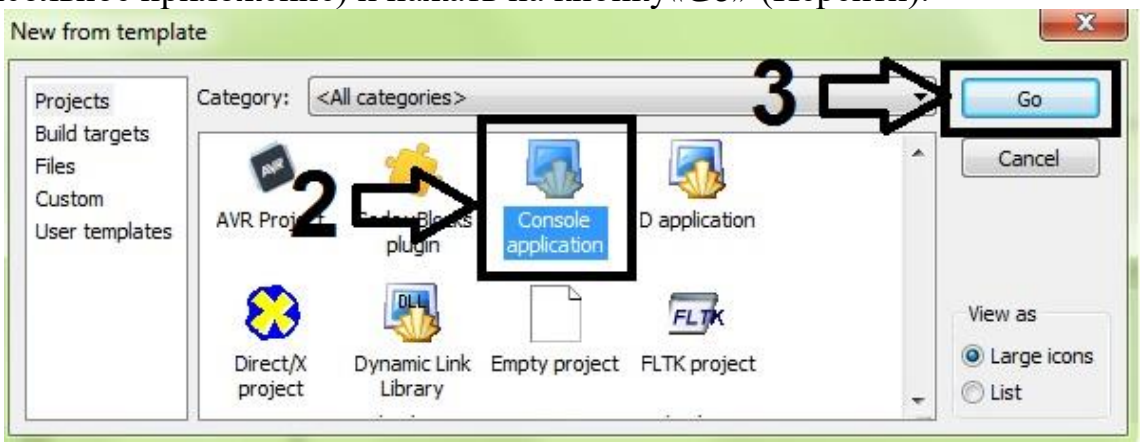
При выводе на экран вещественных чисел возможно указание точности представления данных с помощью функции ***cout.precision(int i)***.

Порядок выполнения лабораторной работы

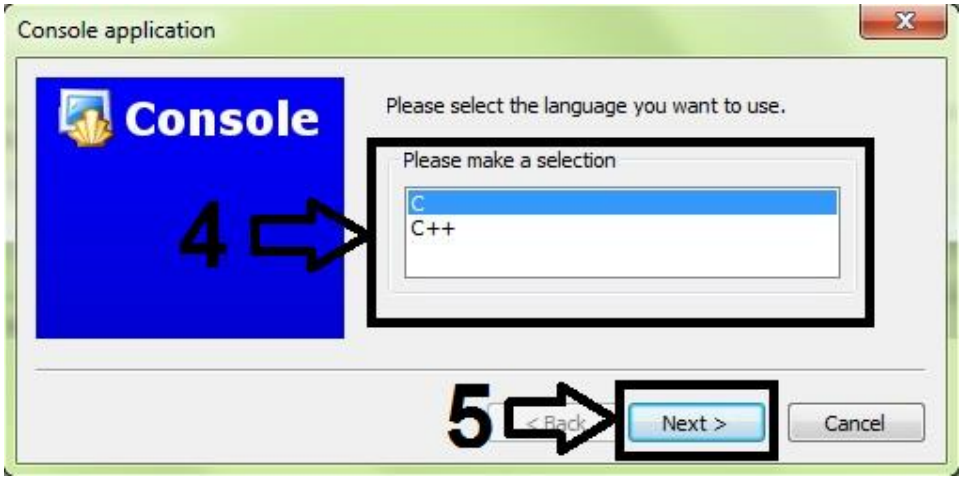
Запуск ***Code::Blocks*** производится через файл *codeblocks.exe*. При этом откроется окно редактирования с меню.



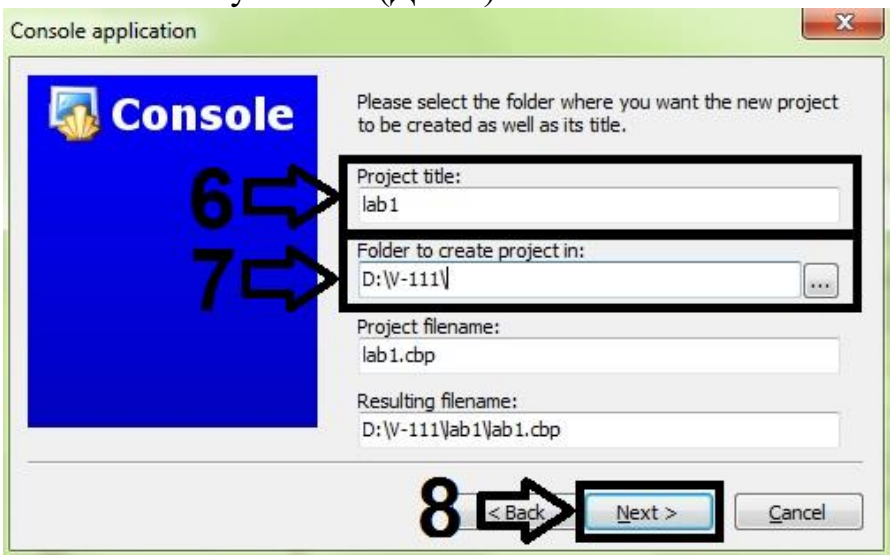
Для создания нового проекта необходимо перейти по ссылке ***Create a new project*** или ***File-New-Project.*** При этом откроется окно создания проекта, в котором нужно выбрать тип нового проекта –***Console application*** (Консольное приложение) и нажать на кнопку***«Go»*** (Перейти).



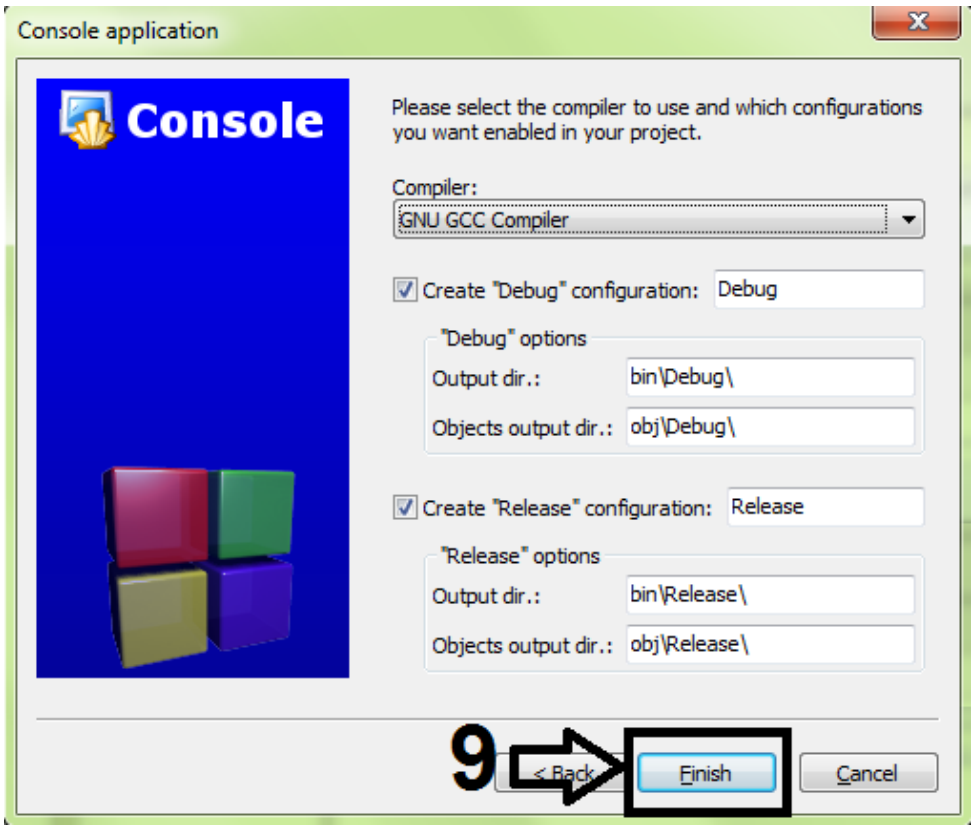
Далее при успешном создании приложения откроется окно *Console application*, в котором необходимо выбрать из списка язык C++ и нажать на кнопку***«Next»*** (Далее).



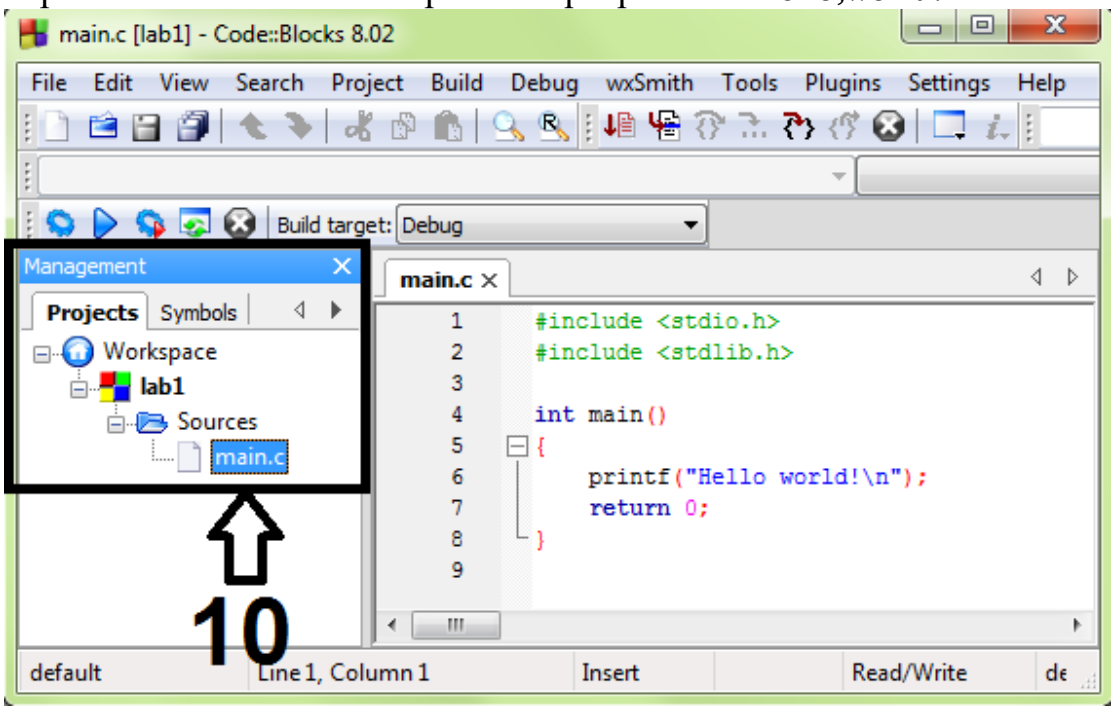
В следующем окне нужно ввести имя проекта и путь для создаваемого проекта и нажать на кнопку ***«Next»*** (Далее).



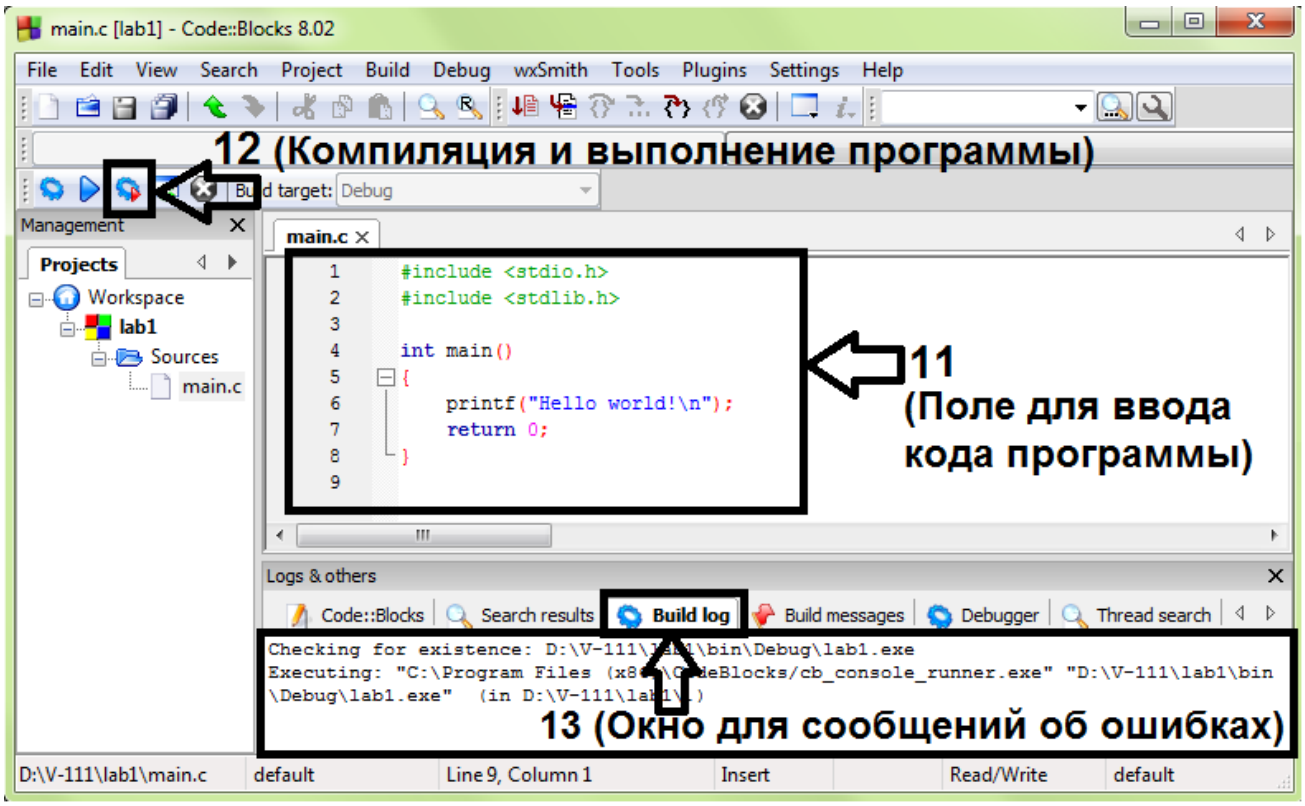
Поля следующего окна создания проекта должны быть заполнены так, как показано на рисунке.



По нажатию кнопки ***«Finish»*** в менеджере проектов во вкладке ***Projects*** открыть папку ***Sources***, найти файл ***main.c*** (если был выбран язык С) или ***main.cpp*** (если был выбран язык С++) и открыть его. После чего откроется окно редактирования с заготовкой короткой программы «*Hello,world!*».



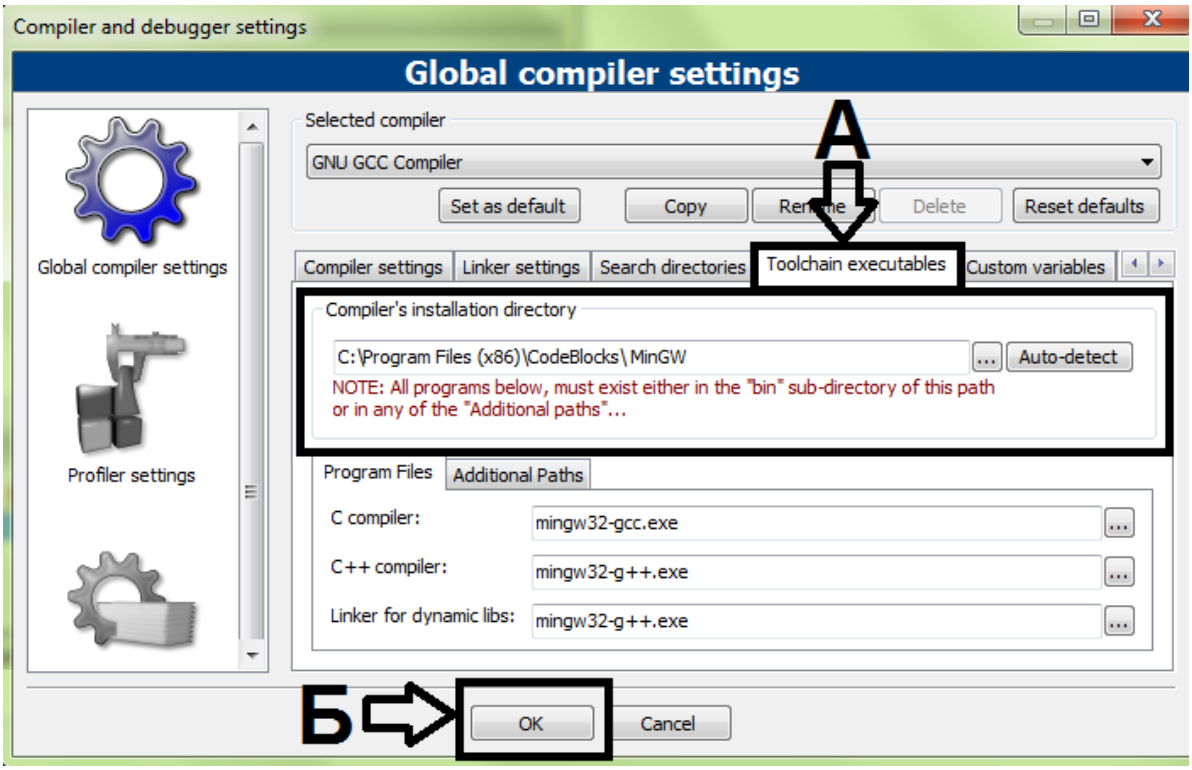
Далее набирается код программы, компилируется и выполняется нажатием клавиши <F9> или иконки, выделенной на следующем рисунке. При наличии ошибок в коде программы сообщения о них выводятся в окне***«Build log»***.



При компиляции программы в среде ***Code::Blocks***наиболее частыми ошибками являются следующие.

1. Компиляция предыдущей программы вместо текущей. Необходимо перезапустить ***Code::Blocks*** и еще раз скомпилировать нужную программу.

2. "Имя программы - Debug" uses an invalid compiler. Skipping... Необходимо правильно указать среде путь до компилятора (до папки ***CodeBlocks/MinGW***) в меню ***Settings – Compiler and debugger*** во вкладке***Toolchain executables***.



Задание на лабораторную работу

1. Ознакомиться с интегрированной средой разработки Code::Blocks.

1.1. Запустить Code::Blocks.

1.2. Создать новый файл для кода программы: File → New → File… → C/C++ Source File, далее выбрать язык C++ и указать полный путь к новому файлу (нажать «…», выбрать каталог и ввести имя файла).

1.3. Слева расположена панель **Management,** где в виде дерева отображена иерархическая структура проекта, состоящего из одного файла main.cpp, находящегося внутри виртуального каталога *Sources*. В свою очередь, каталог *Sources* находится внутри созданного проекта. А проект принадлежит рабочему пространству *Workspace*. Открыть файл main.cpp в главном окне. Добиться работы программы «Hello, world!».

2. Изучить структуру программы.

2.1 Скомпилировать и запустить программу, нажав F9 (или Build → Build and run в меню). Убедиться, что программа запускается и печатает требуемую строку.

2.2. Освоить чтение сообщений об ошибках и предупреждений. Полученные при выполнении этого пункта сообщения компилятора внести в отчет.

2.3. Внести в программу изменения (по одному за раз):

1) допустить ошибку при указании заголовочного файла;

2) удалить одну из круглых скобок;

3) удалить одну из фигурных скобок;

4) удалить одну из кавычек;

5) удалить точку с запятой.

Попытаться скомпилировать программу (Ctrl + F9 или Build → Build в меню). Найти область «Build messages», прочитать возникшие сообщения об ошибках. Вернуть программу к работоспособному состоянию.

2.4. Удалив в программе одновременно открывающую фигурную скобку и точку с запятой, попытаться скомпилировать код. Двойными щелчками мыши по сообщениям в области «Build messages» переместиться к месту каждой ошибки. Вернуть программу к работоспособному состоянию.

2.5. Заменить оператор << на <. Скомпилировать и запустить программу, осмотреть ее вывод. Перейти к области «Build messages», прочитать, перевести и понять текст предупреждения. Вернуть программу к работоспособному состоянию.

2.6. Проверьте, будет ли программа выводить русскоязычный текст. Воспользовавшись подсказкой, изучите причины этой проблемы и добейтесь ее решения.

2.7. Выясните назначение символов \n или endl в составе строки текста.

2.8. Измените программу так, чтобы она печатала какую-либо фразу по Вашему выбору в рамке, образованной символами *.

2.9. Изучите, что такое комментарии и по каким правилам они располагаются в тексте программы. Снабдите каждую строку программы комментариями.

3. Написать программу, считывающую введенные пользователем два числа, и выводящую на экран сумму этих чисел.

Контрольные вопросы

1. Какова структура программы на языке Си++.
2. Как происходит компиляция исходного кода?
3. Для чего нужны комментарии?
4. Типы комментариев в С++.
5. Что такое директива препроцессора?
6. Как включаются в текст программы заголовочные файлы?
7. Способы указания заголовочного файла.
8. Структура и назначение функции main.
9. Для чего используются фигурные скобки?
10. Что такое ошибки и предупреждения компилятора? В чем их отличие?
11. Способы организации ввода-вывода информации в Си++.

Лабораторная работа №2

ОСНОВЫ АЛГОИТМИЗАЦИИ И РАЗРАБОТКА ПРОГРАММЫ С ЛИНЕЙНОЙ СТРУКТУРОЙ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ СИ++

1. ОБЩИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

1.1. Цель работы

Изучение основных принципов разработки алгоритмов программ, знакомство с основными операциями и стандартными функциями языка Си++. Получение практических навыков реализации программ с линейной структурой.

1.2. Используемое оборудование и программное обеспечение

Для выполнения лабораторной работы требуется ПЭВМ типа IBM PC с установленной ОС Windows 2000 и выше, интегрированная среда разработки Code::Bloks.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Основы алгоритмизации

Решение любой задачи на ЭВМ происходит в несколько этапов: постановка задачи; конструирование алгоритма решения задачи; составление программы по разработанному алгоритму; ввод в ЭВМ программы и исходных данных; отладка и тестирование программы; получение решения и анализ результатов.

Алгоритм - конечная последовательность точно определенных действий, приводящих к однозначному решению поставленной задачи.

Главная особенность любого алгоритма - формальное исполнение, позволяющее выполнять заданные действия (команды) не только человеку, но и различным техническим устройствам (исполнителям). Процесс составления алгоритма называется *алгоритмизацией.* Алгоритмы могут быть заданы: словесно, таблично, графически (с помощью блок-схем).

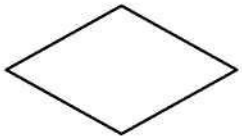
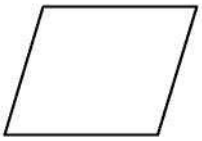
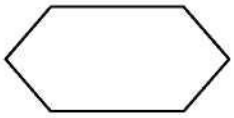
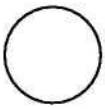
Словесное задание описывает алгоритм с помощью слов и предложений.

Табличное задание служит для представления алгоритма в форме таблиц и расчетных формул.

Графическое задание, или ***блок-схема,*** - способ представления алгоритма с помощью геометрических фигур, называемых *блоками.* Последовательность блоков и соединительных линий образуют блок-схему. Описание алгоритмов с помощью блок-схем является наиболее наглядным и распространенным способом задания алгоритмов. Блок-схемы располагаются сверху вниз. Линии соединения отдельных блоков показывают направление процесса обработки в схеме. Каждое такое направление называется ветвью. Алгоритм независимо от его структуры всегда имеет по одному блоку «Начало» и «Конец». Его ветви должны в конце сойтись, и по какой бы ветви не было бы начато движение, оно всегда должно привести к блоку «Конец».

При задании алгоритма с помощью блок-схемы используются строго определенные блоки. Основные типы блоков приведены в таблице 1. Следует отметить, что все блоки нумеруются. В этом случае номера проставляются вверху слева от блока (блоки «Начало», «Конец» и соединительные блоки не нумеруются). Стрелки на соединяющих линиях обычно не ставят при направлении сверху вниз и слева направо; если направление противоположное, то его показывают стрелкой на линии.

Таблица 1- Основные типы блоков.

Начало Конец	*Начало и конец алгоритма* (для функций «Вход», «Выход»)
	*Блок обработки*. Внутри блока записываются формулы, обозначения и функции
	*Блок условия*. Внутри блока записываются условия выбора направления действия алгоритма
	*Блок предопределенного процесса* (функция/ подпрограмма)
	*Блок ввода информации*
	*Блок цикла с известным количеством повторений*
	*Блок вывода информации на печатающее устройство*
	*Соединительный блок*

Алгоритмы бывают линейные, разветвляющиеся и циклические. Линейный алгоритм не содержит логических условий, имеет одну ветвь обработки и изображается линейной последовательностью связанных друг с другом блоков. Условное изображение линейного алгоритма представлено на рисунке 1.

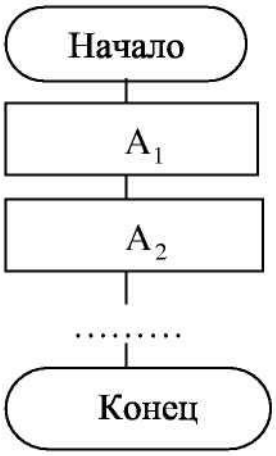


Рисунок 1 – Условное изображение линейного алгоритма

Например, составим блок-схему вычисления $z = \varphi(y)$, $y = f(x)$, где φ, f − известные функции при заданном значении переменной *x* (рис. 3.2).

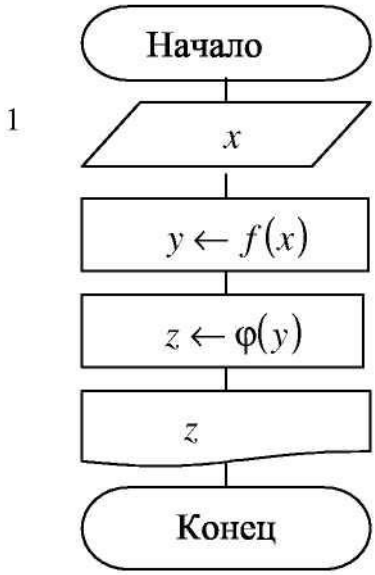


Рисунок 2 - Блок-схема алгоритма решения задачи

Знак «←» означает: внести значение переменной в ячейку памяти с определенным именем (операция присваивания). Разветвляющиеся и циклические алгоритмы будут рассмотрены в следующих работах.

2.2 Типы данных языка Си++

Правила Си++ требуют, чтобы в программе у всех переменных был задан тип данных. Стандартные целочисленные и вещественные типы данных языка Си++ приведены в табл. 1.

Таблица 1.

Тип	Размер	Область значений
char	1 байт	от -128 до 127
int	зависит от реализации	
short *int*	2 байта	от -32768 до 32767
long *int*	4 байта	от -2147483648 до 2147483647
unsigned char	1 байт	от 0 до 255
unsigned *int*	зависит от реализации	
unsigned short *int*	2 байта	от 0 до 255
unsigned long *int*	4 байта	от 0 до 4294967295
float	4 байта	от -3.4e-38 до 3.4e+38
double	8 байт	от – 1.7e-308 до 1.7e+308
long double	10 байт	от -3.4e-4932 до 1.1e+4932

Можно пользоваться сокращенной записью, опуская int там, где в таблице оно дано курсивом.

Тип ***char*** используется для хранения какого-либо символа (например, цифры или пробела). При присваивании используются одинарные кавычки. Тип ***int*** служит для хранения целых чисел, типы ***float*** и ***double*** – для хранения чисел с плавающей точкой. Беззнаковая форма ***unsigned*** дает возможность более эффективно работать с положительными целыми числами. Формы ***short***, ***long*** и ***unsigned*** позволяют экономно расходовать ресурсы компьютера.

Кроме целочисленных и вещественных типов в Си++ есть логический тип (***bool***), переменные которого могут принимать значение 1 (true) или 0 (false), а также «пустой» тип ***void***.

Приведем примеры описания переменных:

int a, b, c;

unsigned int k = 0;
float e = 3.5;
long double d = 1e+21;
char f = 'f', g = '$';
bool h = true;

Множество типов данных порождает проблемы при выполнении операций с операндами различных типов. Непосредственно арифметические действия производятся над числами одного типа, поэтому исходные константы нужно преобразовать к одному типу.

При преобразовании «длинного» (требующего большего объема памяти) типа к «короткому» происходит потеря разрядов и, следовательно, потеря точности.

Преобразование от знакового типа к беззнаковому ведет к потере знака отрицательного числа. Преобразование символьного типа к численному выявляет код символа, при обратном преобразовании – возвращает символ.

Для приведения одного типа к другому можно использовать явное преобразование:

int i = 5;
double a = 0.5;
i = (int)a;
a = (double)i;

Добавление к описанию переменной слова ***const*** означает, что значение переменной нельзя изменить после инициализации, например,

const double pi = 3.1416;

При попытке в тексте программы изменить значение константы компилятор фиксирует ошибку.

2.3 Арифметические и логические операции

Синтаксис языка Си++ позволяет выполнять операции, перечисленные в табл. 2.

Таблица 2

Название операции	Знак
Арифметическое сложение	+
Арифметическое вычитание	-
Умножение	*
Деление	/
Остаток от деления	%
Логическое сложение	||

Логическое умножение	&&
Отрицание	!
Присваивание	=
Равно	==
Не равно	!=
Больше	>
Меньше	<
Больше или равно	>=
Меньше или равно	<=

В арифметических операциях поддерживается тот же *приоритет выполнения операций*, что и в обычных алгебраических, т. е. сначала выполняются умножение, деление и вычисление остатка, потом – сложение и вычитание.

Например, в выражении

*i = k + d * i / m % n – 1;*

действия выполняются в следующем порядке: умножение, деление, вычисление остатка, сложение, вычитание.

Для выполнения действий с нарушением естественного приоритета используются круглые скобки по обычным алгебраическим правилам.

Операция *присваивания* выполняется справа налево. Например,

i = j = 5; // равнозначно j = 5; i = j;

В Си++ имеются сокращенные формы записи арифметических операций (табл. 3).

Таблица 3

Сокращенная форма	Равнозначная форма записи
c += 3;	c = c + 3;
c -= 3;	c = c – 3;
c *= 3;	c = c * 3;
c /= 3;	c = c / 3;
c %= 3;	c = c % 3;
Операция инкремента	
i++; (постфиксная форма)	i = i + 1;
++i; (префиксная форма)	i = i + 1;
Операция декремента	
i--; (постфиксная форма)	i = i – 1;
--i; (префиксная форма)	i = i – 1;

Постфиксная и префиксная формы записи имеют важное различие. *Префиксный* оператор применяется ***до*** вычисления остальной части выражения, а *постфиксный* – ***после***. Например, после выполнения операторов

```
x = 4;
y = x++;
```

переменная x получит значение 5, а y – значение 4. В случае операторов

```
x = 4;
y = ++x;
```

обе переменные получат значение 5. Это объясняется тем, что «++x» выполняется до того, как значение x будет использовано в выражении, а «x++» - после.

2.4 Стандартные математические функции языка Си++

Все математические функции хранятся в файле ***math.h***, следовательно, чтобы их использовать, надо подключить его директивой ***#include***:

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

В этом файле содержатся следующие стандартные математические функции:

int abs(int i) – возвращает абсолютное значение целого аргумента i;

double acos(double x) – арккосинус; значение аргумента должно находиться в диапазоне от -1 до +1;

double asin(double x) – арксинус; значение аргумента должно находиться в диапазоне от -1 до +1;

double atan(double x) – акртангенс;

double cos(double x) – косинус; аргумент (угол) задается в радианах;

double cosh(double x) – гиперболический косинус x;

double exp(double x) – экспоненциальная функция;

double floor(double x) – находит наибольшее целое, не превышающее значения x;

double fmod(double x, double y) – остаток от деления x на y;

double log(double x) – натуральный логарифм;

double log10(double x) – десятичный логарифм;

double pow(double x, double y) – возвращает значение x в степени y;

double sin(double x) – синус; аргумент задается в радианах;

double sinh(double x) – гиперболический синус для x;

double sqrt(double x) – квадратный корень из x;

double tan(double x) – тангенс; аргумент задается в радианах;

double tanh(double x) – гиперболический тангенс x.

3. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Пусть требуется разработать программу, вычисляющую площадь треугольника, если известны значения трех его сторон. Для решения этой задачи используется формула Герона:

$$S = \sqrt{p * (p - a) * (p - b) * (p - c)},$$

где p – полупериметр, вычисляющийся по формуле:

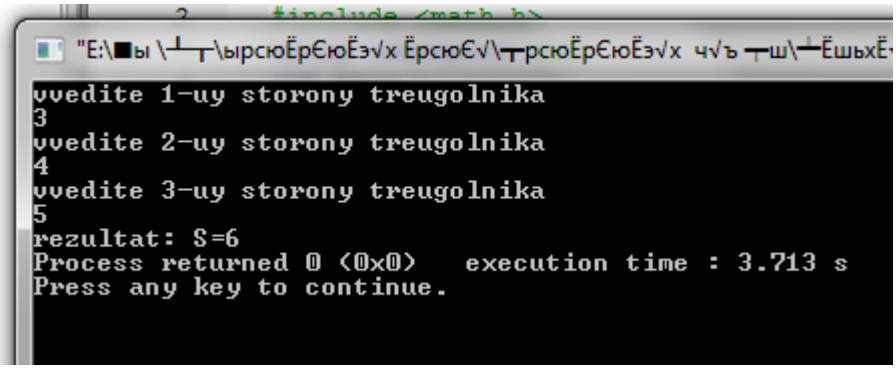
$$p = \frac{a + b + c}{2}.$$

Тогда программа может быть реализована следующим образом:

main.cpp

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    double a,b,c,p,S;
    cout <<"vvedite 1-uy storony treugolnika"<<endl;
    cin>>a;
    cout <<"vvedite 2-uy storony treugolnika"<<endl;
    cin>>b;
    cout <<"vvedite 3-uy storony treugolnika"<<endl;
    cin>>c;
    p=(a+b+c)/2;
    S=sqrt(p*(p-a)*(p-b)*(p-c));
    cout<< "rezultat: S="<<S;
    return 0;
}
```

Тогда для треугольника со сторонами 3,4,5 получится следующий результат:



Очевидно, что необходимо осуществлять проверку входных данных на непротиворечивость, т.е. проверять, соответствуют ли введенные пользователем значения сторон треугольника невырожденному треугольнику (сумма любых двух сторон должна быть больше третьей стороны). Но это можно реализовать, используя только разветвляющиеся алгоритмы.

В рамках данной лабораторной работы необходимо разработать блок-схему алгоритма решения предложенной задачи и реализовать программу с линейной структурой. Отчет должен содержать:

1. Постановку задачи.
2. Графическое представление блок-схемы алгоритма.
3. Листинг программы.
4. Тестовый пример.
5. Пример работы программы.

Варианты заданий выбираются в соответствии с номером в списке подгруппы.

Вычислить значение функции нескольких переменных при заданных значениях параметров:

1. $x=4y^2/(4z-2t^3)$ при t=1 ; z=3; y=sin t.

2 $x=4y^3-z/t$ при t=2; z=3; y=cos(t+z).

3. $x=6t^2-(z+1)/y^2$ при y=2; z=4; t=sin(2+z).

4. $x=(8z^2+1)/(y+t^2)$ при z=1; t=2; y=t+z.

5 $x=8z / (e^t+2)-y^2$ при t=3; z=ctg t +2; y=4.

6. $x=8z/(e^t+2)-y^2$ при t=1; z=t+2; y=4.

7. $x=2y+3$ sh t- z при y=2; t=5 / $(1+y^2)$; z=4

8. $x=3\ y^2/ (4$ tg z-$2t^2)$ при t=0.5; z=6; y=t+2 ctg z.

9. $x=4y^2 /(4y\ e^z - 2t^3)$ при t=1 ; z=3; y=sin t.

10. $x=4 \ln y^3-z / t$ при $t=2; z=3; y=\cos(t+z)$.

11. $x=6 t^2-(\operatorname{ctg} z+1)/ y^2$ при $y=2; z=4; t=\sin(2+z)$.

12. $x=(8z^2+1)/(y e^t +t^2)$ при $z=1; t=2; y=\operatorname{tg} t+z$.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Определение алгоритма.
2. Способы задания алгоритмов.
3. Основные типы блоков.
4. Классификация алгоритмов.
5. Типы данных языка Си++.
6. Арифметические операции.
7. Логические операции.
8. Сокращенные формы записи арифметических операций.
9. Операция инкремента.
10. Операция декремента.
11. Различие постфиксной и префиксной формы.
12. Стандартные математические функции языка Си++.

Лабораторная работа №3

Программы разветвлённой структуры

1. Цель работы: приобретение практических навыков в составлении алгоритмов и программ разветвленной структуры на языке программирования Си++.

2. Теоретические сведения

Алгоритм разветвленной структуры - это алгоритм, в котором вычислительный процесс осуществляется по одной из ветвей. Если количество ветвей две – то используется условный блок, если больше – то множественный выбор. В программах используют соответственно условный оператор IF или оператор SWITCH для реализации разветвленного алгоритма.

Условный оператор позволяет выполнять или пропускать операторы программы в зависимости от некоторого условия, которое представляет собой выражение логического типа. Схема записи условного оператора:

if (*условие****)*** *оператор_1****;***
else *оператор_2****;***

Если *условие* истинно (***true***), то выполнится *оператор_1*, если же условие ложно (***false***), то выполнится *оператор_2*. Например:

```
if (a == 0) a += 1;        // выполняется при a = 0
else a + = 2;              // выполняется при a ≠ 0
```

Часть ***else*** в условном операторе является необязательной.

Если в качестве операторов выступает несколько действий, то необходимо ставить фигурные скобки:

```
if (a == 0)
    {
    a += 1;
    b -= 2;
    }
```

Синтаксис языка Си++ допускает, что в качестве условия может выступать не только логическая, но и арифметическая операция, результатом которой является 0 (***false***) или не 0 (***true***), например:

*if (a % 2) {...} /*действия в скобках выполнятся, если число a не делится на 2 без остатка*/*

*else {...} /*действия в скобках выполнятся, если a делится на 2 без остатка*/*

СИ++ имеет еще одну ***условную операцию***, близкую по структуре к ***if/else***:

условие **?** *оператор_1* **:** *оператор_2***;**

Если *условие* истинно, то выполнится *оператор_1*, иначе выполнится *оператор_2*, например:

a == 0 ? a += 1 : a += 2;

Условные операторы могут быть вложенными друг в друга:

```
if (a == 0) a += 1;
else
    if (a == 1) a += 2;
    else
        if (a == 2)
        else a += 3;
```

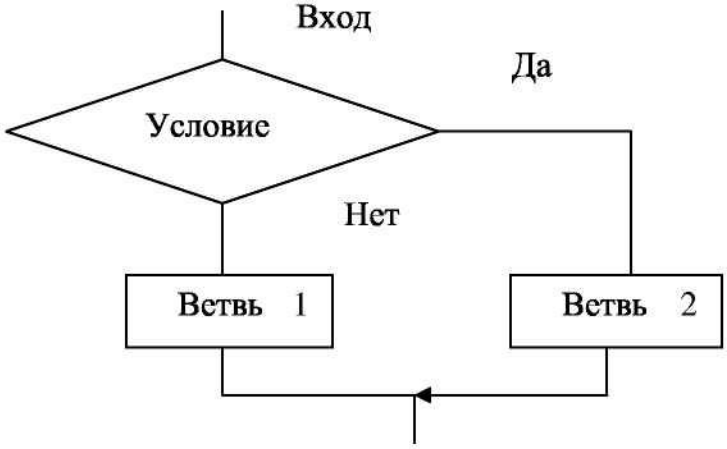


Рисунок 1- Условное изображение разветвляющегося алгоритма.

Разработаем блок-схему алгоритма, определяющего, попадает ли точка внутрь круга, если радиус круга известен, а его центр и точка задаются своими координатами.

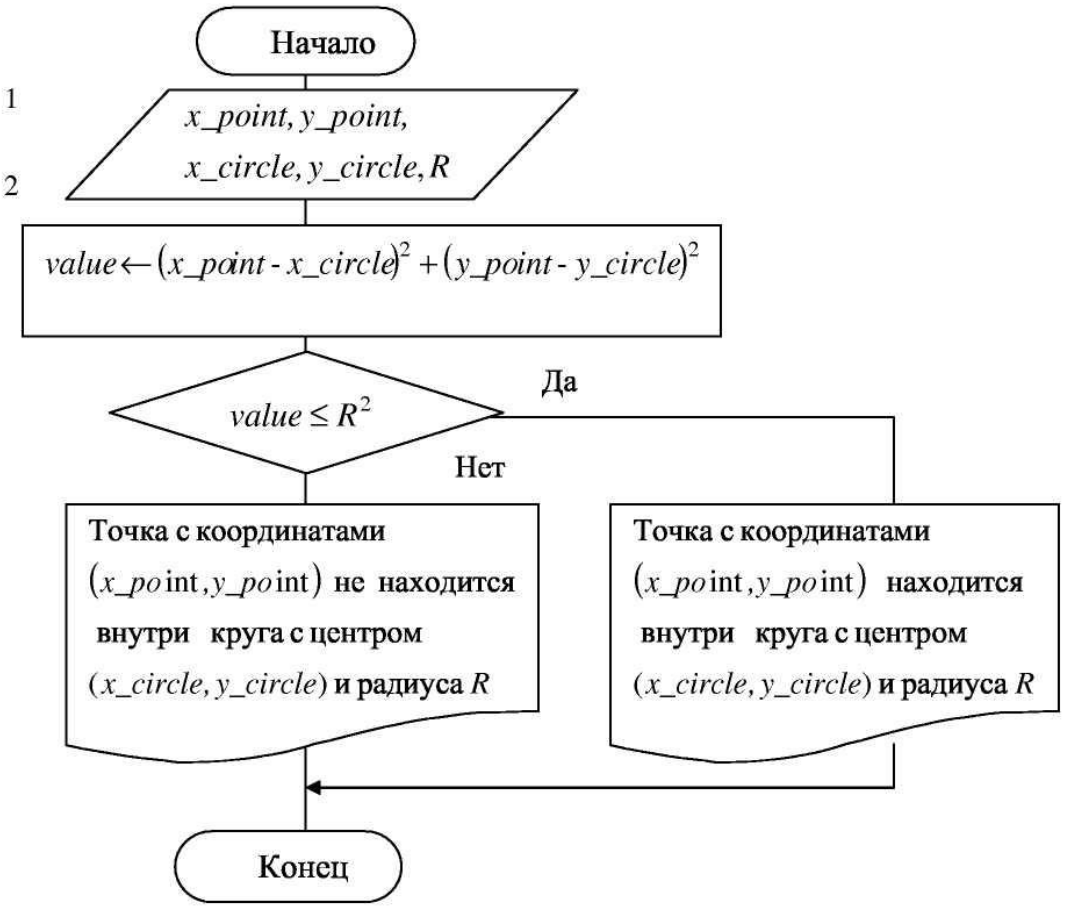


Рисунок 2 Блок-схема алгоритма

Но для реализации множественного ветвления можно использовать ***оператор выбора***:

```
switch (селектор)
    {
    case метка_1:
        оператор_1;
        break;
    case метка_2:
        оператор_2;
        break;
    …
    default:
        оператор_n;
    }
```

Действие оператора **switch** начинается с операции, стоящей за *меткой*, значение которой совпадает со значением *селектора*, и продолжается до ближайшего оператора **break**. Если выражение **break** пропущено, то выполняются и операторы, соответствующие ниже расположенным меткам.

Если значение селектора не совпадает ни с одной из меток, то выполняется оператор после метки ***default***, если же этой метки нет, то осуществляется выход из оператора выбора.

Заметим, что *селектор* в операторе ***switch*** может быть целочисленного или любого перечислимого типа, но не вещественного.

С помощью оператора выбора приведенный выше пример можно записать следующим образом:

```
switch (a)
    {
    case 0:
        a += 1;
        break;
    case 1:
        a += 2;
        break;
    case 2:
        a += 3;
        break;
    }
```

Например, разработаем программу вычисления наибольшего из трех значений функции $y1=x^2+1$, $y2=7-x^2$, $y3=x+1$ для переменной x, заданной пользователем.

main.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  { float x,y1,y2,y3,max;
5       cout <<"vvedite x"<<endl;
6       cin >>x;
7       y1=x*x+1;
8       y2=7-x*x;
9       y3=x+1;
10      if (y1>y2)
11           max=y1;
12    else
13           max=y2;
14    if (y3>max)
15           max=y3;
16    cout <<"naibolshee znachenie="<<max;
17  }
18
```

То же самое, только с использованием трехместной условной операции:

main.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  { float x,y1,y2,y3,max;
5       cout <<"vvedite x"<<endl;
6       cin >>x;
7       y1=x*x+1;
8       y2=7-x*x;
9       y3=x+1;
10      y1>y2? max=y1:max=y2;
11      y3>max? max=y3:max=max;
12    cout <<"naibolshee znachenie="<<max;
13  }
14
```

В качестве примера множественного ветвления разработаем программу, определяющую количество дней по номеру месяца. Как видно из листинга программы, если номер месяца превышает 12, выводится сообщение о неверном вводе месяца, для чего используется default. Оператор break служит для прерывания цикла проверки и перехода в конец переключателя. В случае отсутствия break, происходит переход на следующую ветвь.

main.cpp

```cpp
#include <iostream>
using namespace std;
int main()
{
    int m;
    cout << "введите номер месяца" << endl;
    cin>>m;
    switch (m) {
        case 1:case 3:case 5:case 7:case 8:case 10:case 12:
            cout << "тридцать один 31" << endl;
            break;
        case 2:
            cout << "двадцать восемь 28" << endl;
            break;
        case 4:case 6:case 9:case 11:
            cout << "тридцать 30" << endl;
            break;
        default:
            cout << "номер месяца неверен" << endl; }
    return 0;
}
```

Задание на лабораторную работу.

Разработать блок-схемы алгоритмов и реализовать программы разветвляющейся структуры согласно вариантам задания 1 (используя IF) и задания 2 (используя SWITCH).

Варианты задания 1

1. Даны четыре числа. На сколько их сумма меньше их произведения?
2. Даны четыре числа. Вычислить сумму положительных среди них чисел.
3. Даны четыре числа. Вычислить произведение отрицательных среди них чисел.
4. Даны четыре числа. Все отрицательные среди них числа заменить на 0.
5. Даны четыре числа. Определить сколько среди них отрицательных и сколько положительных.
6. Даны четыре числа. Определить порядковый номер наименьшего среди них.

7. Даны два числа. Большее из этих двух чисел заменить их полусуммой, а меньшее удвоенным произведением.

8. Даны три числа. Меньшее среди них заменить на 0.

9. Даны четыре числа. Найти разность между наибольшим и наименьшим среди них.

10. Даны три числа K, M и N. Поменять их значения местами таким образом, чтобы K < M < N.

11. Даны четыре разных числа. Найти среди них два наибольших.

12. Даны три числа. Поменять местами большее и меньшее из этих чисел.

Варианты задания 2

1. Разработать программу, которая запрашивает у пользователя номер месяца и выводит соответствующее название времени года, например: «2 - ЗИМА». Если ввести число, не являющееся номером месяца (например, 14), то программа выведет: «Такого номера месяца нет».

2. Группу детей, приехавшую в пионерский лагерь, распределяют по отрядам по следующему принципу:

с 6 до 7 лет - 5-й отряд;

с 7 до 9 лет - 4-й отряд;

с 9 до 11 лет - 3-й отряд;

с 11 до 13 лет - 2-й отряд;

с 13 до 15 лет (включительно) - 1-й отряд.

Составить программу-меню, которая позволила бы каждому приезжающему самому определять свой отряд.

3. Составить программу-меню, реализующую эпизод сказки: она спрашивает, куда предпочитает пойти герой (налево, направо или прямо), и печатает, что его ждет в каждом случае. Текст вопросов и ответов ЭВМ задать самостоятельно. Признак эпизода вводится с клавиатуры.

4. Составить программу, которая вычисляет площадь круга, заданную либо радиусом R, либо диаметром D, либо длиной окружности L:

$$S = \begin{cases} \pi \cdot R^2 \\ \pi \cdot D^2 / 4 \\ L^2 / 4\pi \end{cases}$$

5. Самолет летит из пункта A в пункт B со средней скоростью V. Составить программу нахождения времени в полете t, если возможны три варианта:

а) дует встречный ветер (K = 1);

б) ветра нет (K = 2);

в) дует попутный ветер (K = 3).

6. Написать программу-меню, которая спрашивала бы сокращенное имя, а печатала полное (например: Саша - Александр) для пяти ваших друзей. Ввод

незнакомого имени должен провоцировать заявление типа: «Я с Вами не знаком». Необходимые данные задать самостоятельно.

7. Ввести два целых числа X и У. Составить программу-меню для арифметических операций умножения, деления, сложения, вычитания и выполнить в зависимости от этого соответствующую операцию над числами X и У.

8. Вводится число М - номер месяца. Определить номер квартала по введенному номеру месяца и номер полугодия.

9. Вводится число М - номер месяца. Определить время года по введенному номеру месяца и номер полугодия.

Лабораторная работа №4
Программы циклической структуры

Цель работы: приобретение практических навыков в составлении алгоритмов и программ циклической структуры.

1. Теоретические сведения

1.1. Построение блок-схем циклических вычислительных процессов

Циклический алгоритм содержит один или несколько циклов. *Цикл* - это многократно повторяемая часть алгоритма. Цикл, не содержащий внутри себя других циклов, называют простым. Если он содержит внутри себя другие циклы или разветвления, то цикл называют сложным или вложенным. Любой цикл характеризуется одной или несколькими переменными, называемыми параметрами цикла, от анализа значений которых зависит выполнение цикла. *Параметр цикла* - переменная, принимающая при каждом вхождении в цикл новое значение. Условное изображение циклического алгоритма представлено на рисунке 1.

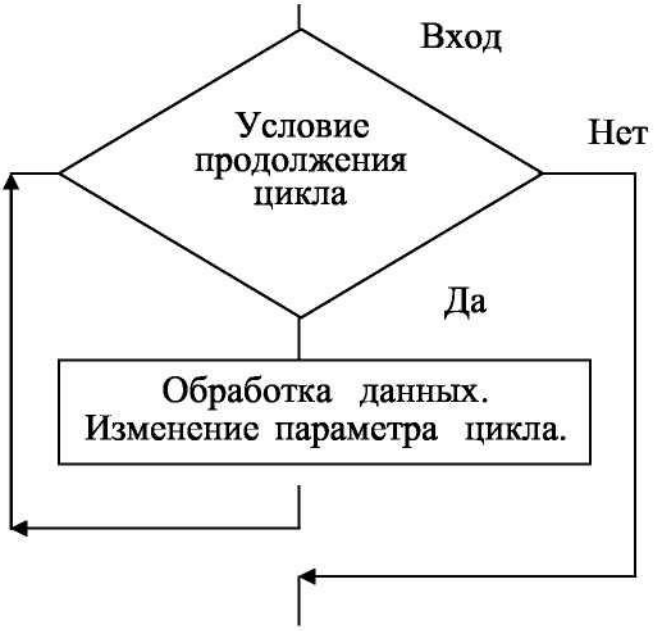


Рисунок 1 - Условное изображение циклического алгоритма.

Например, разработаем блок-схему нахождения суммы N первых целых чисел от 0 до $N-1$ (рисунок 2, рисунок 3).

Способ 1.

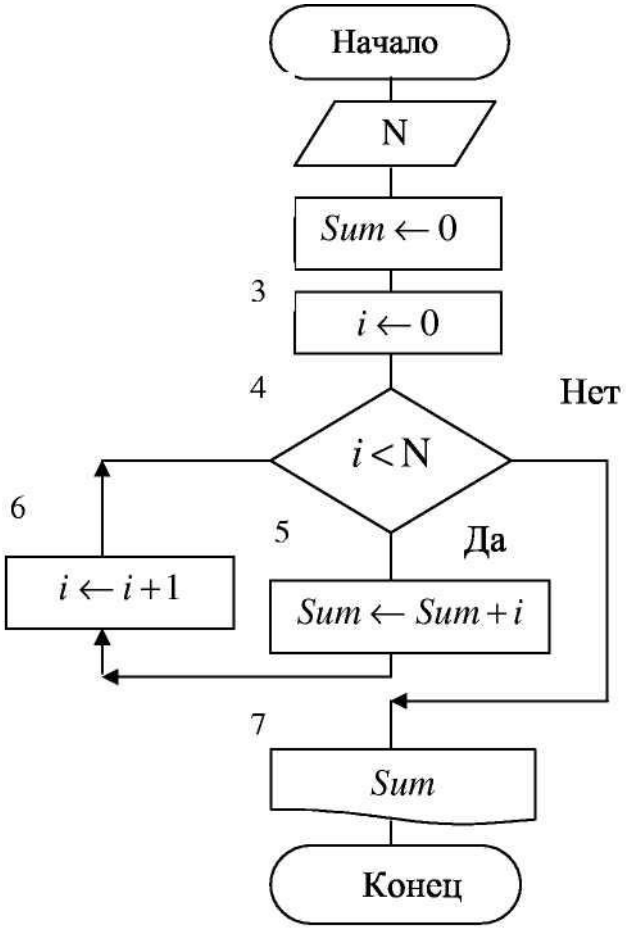


Рисунок 2 - Блок-схема алгоритма

Блок-схема любого циклического процесса независимо от многообразия сводящихся к нему задач должна содержать блок задания начального значения параметра цикла (блок 3), блок реализации необходимых вычислений (блок 5), блок изменения параметра цикла (блок 6) и блок проверки условия окончания цикла (блок 4).

Способ организации цикла зависит от условия задачи. Во многих задачах количество повторений цикла указывается (рисунок 3). Это так называемые *циклы с известным количеством повторений* или *циклы со счетчиком.*

Способ 2.

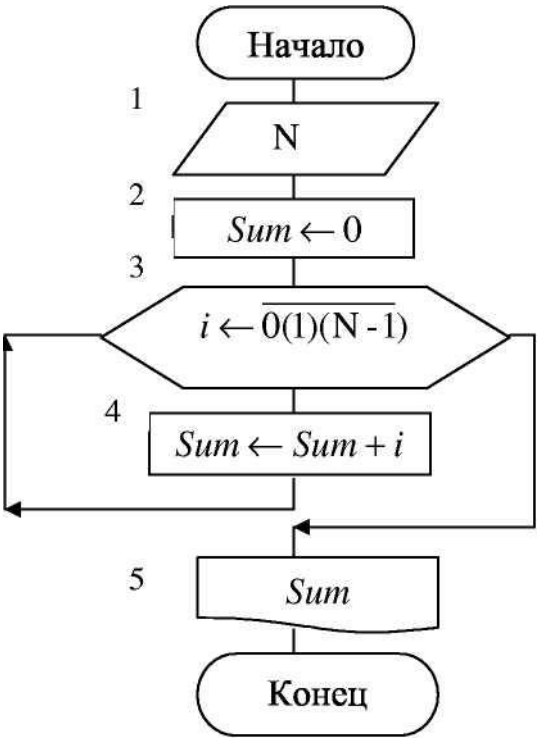


Рисунок 3 - Блок-схема алгоритма

Во многих задачах встречаются так называемые *циклы по простой переменной.* В этом случае задается только начальное значение переменной, закон (правило), по которому она изменяется, и конечное ее значение. В процессе решения задачи текущее значение переменной каждый раз сравнивается с ее конечным значением. Выход из цикла происходит, когда переменная достигла своего конечного значения или впервые превысила его. В таких циклах количество повторений неизвестно, но оно может быть вычислено, поскольку значение переменной изменяется по некоторому закону.

Например, разработаем блок-схему нахождения значений функции $f(x) = \sin x$ для значений переменной x, изменяющейся в диапазоне от a до b шагом h (рисунок 4).

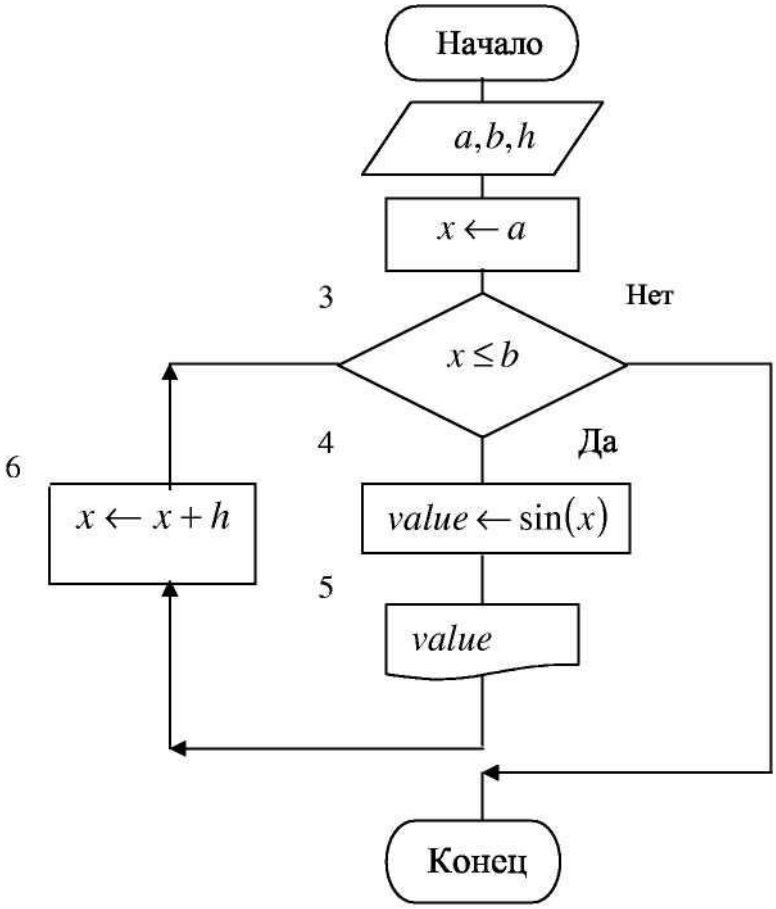


Рисунок 4 – Блок-схема алгоритма

1.2. Операторы циклов языка Си++.

Операторы циклов позволяют программисту определить действия, которые должны многократно повторяться при соблюдении заданных условий.

В Си++ предусматривает использование трех видов циклов:

1) цикл с параметром:

for (*инициализация_цикла****;***
*условие****;***
*изменение_параметра****)***
тело_цикла

2) цикл с предусловием:

while (*условие****)***
тело_цикла

3) цикл с постусловием:

do
тело_цикла
while (*условие****)***

В ***цикле с параметром*** в разделе *инициализация_цикла* задается последовательность выражений, разделяемых запятыми. Все выражения, входящие в

инициализацию цикла, вычисляются только один раз при входе в цикл. Чаще всего здесь устанавливаются начальные значения счетчиков и параметров ИКла. Затем вычисляется условие и, если оно истинно, выполняется тело цикла, после этого изменяется параметр цикла и проверяется его значение. Далее цепочка действий повторяется.

Например, нужно вычислить сумму квадратов натурального ряда чисел от 1 до n.

$$S=\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \ldots + n^2.$$

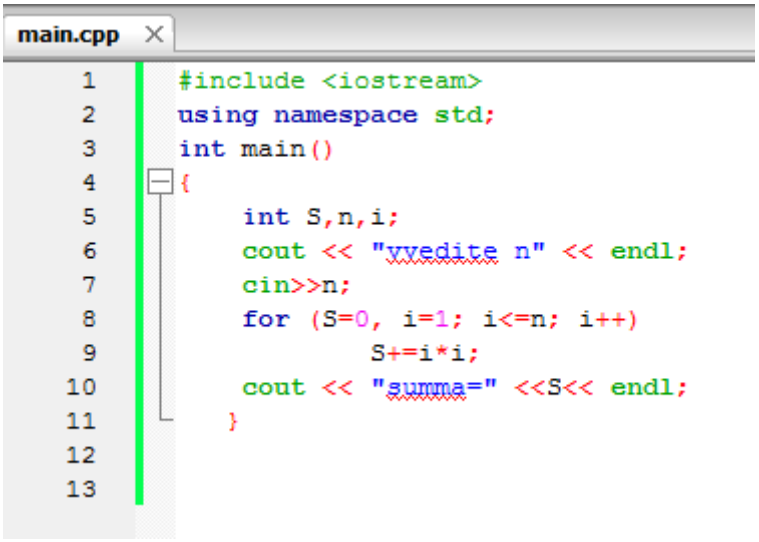


Рисунок 5 – Листинг программы с циклом со счетчиком

Если переменная i используется только внутри цикла, то ее описание можно внести в инициализацию цикла (в этом случае *областью видимости* переменной будет только тело цикла):

```
int sum = 0;

for (int i = 1; i <= 5; i++)
    sum += i * i;
```

Если заданное условие всегда *ложно*, например

(i = 1; i < 1; i++)

то тело цикла не исполняется ни разу. Если заданное условие всегда *истинно*, например

(i = 1; i > 1; i++)

то тело цикла будет повторяться бесконечно.

Если тело цикла содержит не одну операцию, а несколько, то тогда все операции тела цикла следует заключить в фигурные скобки.

Для досрочного выхода из цикла и перехода на следующий оператор после цикла используется оператор break. Для пропуска всех операторов, оставшихся до конца тела цикла и перехода к следующему повторению цикла используется оператор continue.

Циклы с неизвестным количеством повторений используются тогда, когда количество повторений цикла заранее неизвестно и не может быть вычислено.

При входе в ***цикл с предусловием*** вначале вычисляется условие. Если его значение истинно, то выполняется тело цикла, затем опять проверяется условие и т. д. до тех пор, пока условие не станет ложным.

При использовании цикла с предусловием необходимо следить за тем, чтобы внутри тела цикла каким-либо образом изменялась переменная, проверяемая в условии. Только таким образом можно избежать зацикливания.

Запишем подсчет суммы квадратов с помощью цикла ***while***:

```
int i = 1;
int sum = 0;

while (i <= 5)
    {
    sum += i * i;
    i++;
    }
```

Цикл с постусловием отличается от цикла while тем, что сначала выполняется тело цикла, а потом проверяется условие, поэтому тело цикла обязательно выполнится как минимум один раз.

Например,

```
int i = 1;
int sum = 0;

do
    {
    sum += i * i;
    i++;
    }
while (i <= 5);
```

Для изменения хода цикла можно использовать операторы ***break*** и ***continue***.

При использовании оператора ***break*** цикл досрочно прерывается и осуществляется выход из цикла.

Оператор ***continue*** пропускает все следующие за ним операции тела цикла, но из цикла не выходит.

Изменим формулировку примера вычисления суммы квадратов натурального ряда чисел от 1 до n.

$$S=\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \ldots + n^2.$$

Пусть необходимо определить число n, при котором сумма чисел S предыдущего ряда не превысит величину K, введенную с клавиатуры. Такую программу можно реализовать с помощью циклов предусловия или постусловия следующим образом:

main.cpp | main.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int S,S1,K,i;
6      cout << "vvedite K" << endl;
7      cin>>K;
8      S=0;
9      i=0;
10     while ( S<K) { S1=S; i++ ; S+=i*i; };
11     cout << "chislo n=" <<--i<<"summa="<<S1<< endl;
12   }
13
14
15
```

Рисунок 6 – Листинг программы с циклом с предусловием

ain.cpp | main.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int S,S1,K,i;
6      cout << "vvedite K" << endl;
7      cin>>K;
8      S=0;
9      i=0;
10     do
11         {  S1=S; i++;    S+=i*i; }
12     while ( S<K) ;
13     cout << "chislo n=" <<--i<<"summa="<<S1<< endl;
14   }
15
16
17
18
19
```

Рисунок 7 – Листинг программы с циклом с постусловием

Задания на лабораторную работу.

Для своего варианта задания, выбранного в соответствии с номером в списке, разработать алгоритм решения задачи, представить его графически и реализовать программу.

Вариант	Операторы циклов while и for	Оператор цикла do while
1	Вычислить $\sum_{i=1}^{50} 1/i^2$ с использованием оператора for	Написать программу ввода произвольных чисел до тех пор, пока не будет введено число 0
2	Вычислить $f(x) = kx + b$, при $x = 1,2,...,100$ с использованием оператора while	Написать программу ввода произвольных символов до тех пор, пока не будет введен символ q
3	Вычислить $\sum_{i=1}^{50}\sum_{j=1}^{30} i + j$ с помощью вложенных циклов for	Написать программу подсчета суммы 10 чисел, вводимых с клавиатуры
4	Вычислить $S = \sum_{i=1}^{\infty} i$ пока $S < 50$ с помощью цикла while	Написать программу вычисления произведения 5 чисел, введенных с клавиатуры
5	Вычислить $S = \sum_{i=1}^{\infty} i^2$ пока $S < 100$ с помощью цикла for	Написать программу вычисления модулей введенных чисел до тех пор, пока пользователь не введет 0
6	Вычислить $\sum_{i=1}^{50}\sum_{j=1}^{10} 1/(i + j)$ с помощью вложенных циклов while	Написать программу определения знака введенных чисел до тех пор, пока пользователь не введет 0
7	Вычислить $f(x) = x^2 + b$, при $x = -10,-9,...,10$ с использованием оператора for	Написать программу определения минимального введенного числа из 10 чисел
8	Вычислить $\sum_{i=-10}^{10} 1/i^3$, $i \neq 0$ с использованием оператора for	Написать программу определения максимального введенного числа из 5 чисел
9	Вычислить $\sum_{i=-10}^{20}\sum_{j=0}^{10} 1/(i + j)^2$, $i + j \neq 0$ с помощью вложенных циклов for	Написать программу определения минимального среди положительных введенных 10 чисел
10	Вычислить $f(x) = 1/x$, $x \neq 0$ при $x = -10,-9,...,10$ с использованием оператора for	Написать программу определения максимального среди отрицательных введенных 7 чисел

Лабораторная работа №5
Обработка статических массивов

Цель работы: приобретение практических навыков в составлении программ с массивами на языке Си++.

Теоретические сведения

Массивы - структурированный тип данных с элементами одного и того же типа (int, float, char, и т.д.), имеющий одно имя и определенное количество элементов. Количество элементов определяет размер массива. Порядковый номер элемента массива называется его индексом. Число индексов называется размерностью массива, например, массив с двумя индексами называется двумерным массивом. Строка символов является массивом символов, вектор – массив чисел, матрица – массив векторов. Обработка массивов выполняется следующим образом: объявление, ввод или инициализация элементов массива, преобразование и вывод.

Объявление массива

Чтобы использовать массив, надо его объявить – выделить место в памяти компьютера, объём которой зависит от количества элементов и типа массива. При объявлении массива компилятор выделяет для него последовательность ячеек памяти, для обращения к которым в программе применяется одно и то же имя. В то же время массив позволяет получить прямой доступ к своим отдельным элементам.

Оператор описания массива имеет следующий синтаксис:

тип_данных имя_массива [размер_массива];

Доступ к элементам массива осуществляется с помощью индексирования. С помощью операции [] можно обратиться к произвольному элементу массива по имени массива и индексу – целочисленному смещению от начала:

имя_массива[индекс]

Индексация массивов в Си++ начинается с ***нуля***.

Например, объявим массив целых чисел int_array и заполним его элементы:

```
int array[3];
array[0] = 5;
array[1] = 7;
array[2] = 10;
```

Можно инициализировать элементы массива сразу же после объявления:

int array[3] = {5, 7, 10};

В этом случае размер массива можно не указывать.

С элементами объявленного массива можно выполнять все действия, допустимые для обычных переменных этого типа:

int s = array[0] + array[1];
double d = array[1] / array[2];

Массивы могут быть многомерными, например,

int array2x2[2][2] = {{3, 5}, {8, 9}};

Внутренние фигурные скобки при инициализации массива используются для большей наглядности, их можно опустить.

Для обращения к элементу многомерного массива нужно указывать соответствующее число индексов:

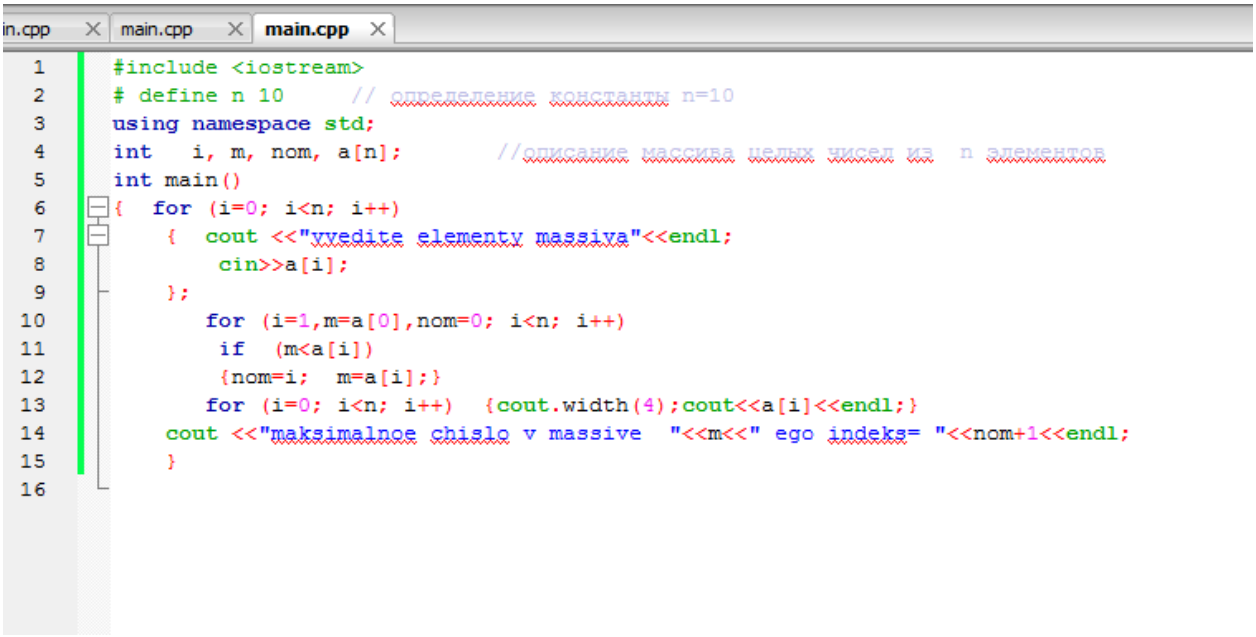
*int s = array2x2[0][0] * array2x2[1][1];*

Например, пусть нужно ввести массив из 5 элементов с клавиатуры и подсчитать сумму его элементов:

main.cpp | main.cpp | **main.cpp**

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5  const int n = 5;
6  int array[n];
7  cout << "vvedite elementy massiva" << endl;
8  for(int i = 0; i < n; i++)
9          cin >> array[i];
10 int s = 0;
11 for (int i = 0; i < n; i++)
12         s += array[i];
13 cout << "summa elementov massiva = " << s << endl;
14 }
15
```

Рассмотрим пример обработки одномерного массива. Дан массив из 10 целых чисел. Найти наибольший элемент в массиве и его порядковый номер.



Рассмотрим процесс заполнения массива случайными числами. Пусть требуется заполнить массив равномерно распределенными случайными числами в интервале [a,b]. Поскольку для целых и вещественных чисел способы вычисления случайного числа в заданном интервале отличаются, рассмотрим оба варианта.

Описание функции-датчика случайных чисел находится в библиотеке cstdlib. Удобно также добавить в программу функцию random:

int random (int N) { return rand() % N; },

которая выдает случайные числа с равномерным распределением в интервале [0,N-1].

Для получения случайных чисел с равномерным распределением в интервале [a,b] надо использовать формулу:

$$k = random (b – a + 1) + a.$$

Для вещественных чисел используется формула:

$$x = rand()*(b - a)/RAND\_MAX + a.$$

Здесь константа RAND_MAX – это максимальное случайное число, которое выдает стандартная функция rand.

В приведенной ниже программе массив A заполняется случайными целыми числами в интервале [-5,10], а массив X - случайными вещественными числами в том же интервале. Для организации ввода-вывода используется традиционный подход, реализованный подключением стандартной библиотеки cstdio.

```
in.cpp | main.cpp | main.cpp | *main.cpp
1  #include <cstdlib>
2  #include <cstdio>
3  const int N = 10;
4  int random (int N) { return rand()%N; }
5  int main()
6  { system("CLS");//очищает экран консоли
7    int i, A[N], a = -5, b = 10;
8    float X[N];
9    for ( i = 0; i < N; i ++ )
10      A[i] = random(b-a+1) + a;
11   for ( i = 0; i < N; i ++ )
12      X[i] = (float)rand()*(b-a)/RAND_MAX + a;
13   printf("\nzelye\n");
14   for ( i = 0; i < N; i ++ )
15      printf("%4d",A[i]);
16   printf("\nveshestvennye\n");
17   for ( i = 0; i < N; i ++ )
18      printf("%6.2f",X[i]);
19   system("PAUSE");//команда задержки экрана
20   return 0;
21 }
22
23
```

Задания на лабораторную работу.

Для своего варианта задания, выбранного в соответствии с номером в списке, разработать алгоритм решения задачи, представить его графически и реализовать программу.

Вариант	Одномерный массив	Двумерный массив
1	Записать в массив значения функции $f(x) = kx + b$, при $x = 1,2,...,100$ и вывести его на экран	Занести в массив значения функции $f(x, y) = x + y$, $0 \le x \le 20$, $0 \le y \le 10$ и вывести его на экран
2	Записать в массив значения функции $f(x) = a\sin(x/100)$, при $x = 1,2,...,100$ и вывести его на экран	Написать программу ввода в массив 5x4 элемента чисел и поиска в нем максимального значения
3	Написать программу ввода в массив 20 чисел и поиска в нем максимального значения	Занести в массив значения функции $f(x, y) = 1/(x + y)$, $0 \le x \le 30$, $1 \le y \le 20$ и вывести его на экран
4	Записать в массив значения функции $f(x) = a\cos(x/50)$, при $x = 1,2,...,100$ и вывести его на экран	Написать программу ввода в массив 6x3 элемента чисел и поиска в нем минимального значения
5	Написать программу ввода в массив 10 чисел и поиска в нем минимального значения	Занести в массив значения функции $f(x, y) = (x + y)^2$, $0 \le x \le 5$, $0 \le y \le 3$ и вывести его на экран
6	Записать в массив значения функции $f(x) = x^2 + b$, при $x = 1,2,...,10$ и вывести его на экран	Написать программу ввода в массив 6x5 элементов чисел и вычисления суммы элементов полученного массива
7	Написать программу ввода в массив 20 чисел и вычисления суммы элементов полученного массива	Занести в массив значения функции $f(x, y) = 1/((x - y)^2 + 1)$, $0 \le x \le 5$, $0 \le y \le 10$ и вывести его на экран
8	Написать программу ввода в массив 5 чисел и вычисления произведения элементов полученного массива	Написать программу ввода в массив 3x3 элемента чисел и вычисления произведения элементов полученного массива
9	Записать в массив значения функции $f(x) = 1/x + b$, при $x = 1,2,...,50$ и вывести его на экран	Занести в массив значения функции $f(x, y) = x - y$, $0 \le x \le 20$, $0 \le y \le 10$ и вывести его на экран
10	Написать программу ввода в массив 10 чисел и поиска в нем модуля максимального значения	Написать программу ввода в массив 4x4 элементов чисел и поиска в нем модуля максимального значения

Лабораторная работа № 6
Функции

Цель работы: Изучение методов использования функций языка Си.

1. Теоретические сведения

Часто в программе требуется повторить определенную последовательность операторов в разных частях программы. Для того, чтобы описывать эту последовательность один раз, а применять многократно, в языках программирования применяются подпрограммы. Подпрограмма - автономная часть программы, выполняющая определенный алгоритм и допускающая обращение к ней из различных частей общей программы.

В языке Си существует один вид подпрограмм - функции. Каждая программа в своем составе должна иметь главную функцию main(), служащую точкой входа в программу. Кроме функции main(), в программу может входить произвольное число функций, выполнение которых инициализируется либо прямо, либо вызовами из функции main(). Каждая функция по отношению к другой является внешней. Для того, чтобы функция была доступной, необходимо, чтобы до ее вызова о ней было известно компилятору. Форма записи функции:

<тип > <имя функции>(<формальные параметры>){<тело функции >}

Если тип возвращаемого функцией значения не указан, то подразумевается int. Если с именем функции не связан результат, то нужно указать тип функции void. Параметры, записываемые в обращении к функции, называются фактическими; параметры, указанные в описании функции - формальными. Фактические параметры должны соответствовать формальным по количеству, порядку следования и типу. Объекты, объявленные вне функции, действуют в любой функции и называются глобальными. Объекты, объявленные в функции, действуют только в ней и называются локальными. В теле функции обычно присутствует оператор return <выражение>, определяющий возвращаемое функцией значение.

Все параметры функции, кроме массивов, передаются по значению, т.е. внутри функции создаются локальные копии параметров. Если необходимо передать саму переменную, а не её копию, то в функцию передаётся адрес этой переменной. Таким образом, через параметры можно передавать результат выполнения функции. То есть, параметры, с помощью которых результаты должны передаваться из функции в точку вызова, описываются как указатели. Вызов функции может быть оформлен в виде оператора, если с именем функции не связано возвращаемое значение, или в виде выражения, если возвращаемое значение связано с именем функции.

Прототип функции может указываться до вызова функции вместо описания функции для того, чтобы компилятор мог выполнить проверку соответ-

ствия типов аргументов и параметров. Прототип функции по форме такой же, как и заголовок функции. В конце него ставится «;».

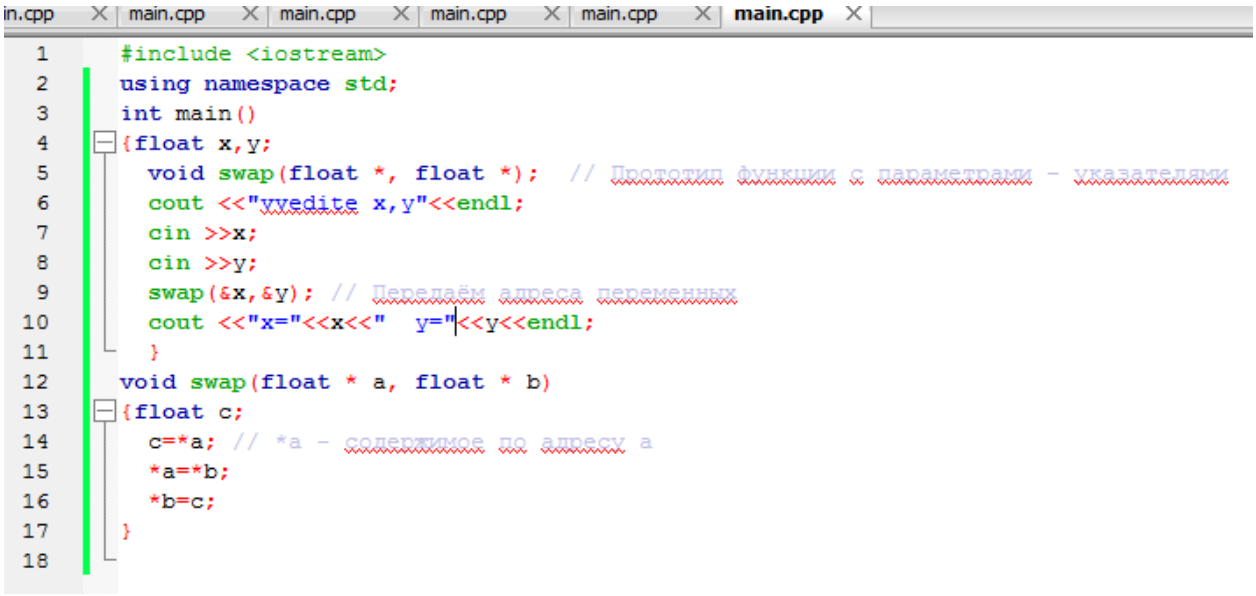
Функции можно подключать с помощью директивы #include <имя файла>. Такие файлы с функциями удобно использовать в диалоговых программах с пользовательским меню, позволяющих выбрать один из режимов.

Пример 1: Функция с параметрами-значениями. Результат связан с именем функции. В программе объявляется прототип функции, а сама функция описывается ниже.

ain.cpp | main.cpp | main.cpp | main.cpp | **main.cpp**

```
1   #include <iostream>
2   using namespace std;
3   int max(int,int); //Прототип функции
4   int main()
5   {
6       int x,y,z;
7       cout<<"vvedite x,y "<<endl;
8       cin>>x;
9       cin>>y;
10      z=max(x,y);//Вызов функции с фактическими параметрами
11      cout<<"x="<<x<<"y="<<y<<"z="<<z<<endl;
12      return(0);
13  }
14  int max(int a ,int b) //Заголовок функции с формальными параметрами
15  { int c;
16    if (a>b) c=a;
17    else c=b;
18    return c;
19  }
20
```

Пример 2: Функция с параметрами-указателями. Здесь передаются адреса фактических параметров, по которым и получаем результат. Функция меняет местами переменные x,y.



Пример 3: Подключение файлов с функциями и создание меню.

! Внимание! Следите за тем, чтобы константы, объявленные директивой #define, не переобъявлялись в функциях.

```
//lab8_3
#include <stdio.h>
#include <conio.h>
#include "lab3.c"
#include "lab5.c"
#include "lab6.c"
main()
{ int nom;
  while(1)
  {  clrscr();
     printf("\n  1. Сумма ряда \n");
     printf("  2. Матрица \n");
     printf("  3. Строки \n");
     printf("  4. Выход \n");
     scanf("%d",&nom);
     switch(nom)
     {
       case 1:lab3();break;
       case 2:lab5();break;
       case 3:lab6();break;
       case 4:return 0;
       default:printf("Неверный режим");
      }
  }
  getch();
```

```
}
```

Пример 4: Передача в функцию массива с использованием указателя. Результат – элементы массива возводятся в квадрат. Функция описывается до вызова, поэтому прототип не объявляется.

```
//lab8_4
#include <stdio.h>
#include <conio.h>
void quart(int n, float * x)  // Можно void quart(int n, float x[])
{ int i;
  for (i=0;i<n;i++)
     x[i]=x[i]*x[i];
}
main()
{ float z[]={1,2,3,4};int j;
  clrscr();
  for (j=0;j<4;j++)
  printf(" %6.2f",z[j]);
  quart(4,z);
  for (j=0;j<4;j++)
     printf("\n %6.2f",z[j]);
  getch();
}
```

3. Выполнение работы

3.1. Проанализировать приведенные выше программы.

3.2. Оформить свои задания по лабораторным работам 5, 6, 7 в виде функций. Организовать меню с вызовом этих функций.

3.3. Используя функции, написать программу по своему варианту.

Варианты заданий

1. Написать функцию, выводящую в порядке возрастания элементы одномерного массива. В главной программе вызвать функцию для двух разных массивов.

2. Написать функцию вычисления произведения прямоугольной матрицы A размера k x m на прямоугольную матрицу B размера m x n. В главной программе обратиться к этой функции.

3. Написать функцию вычисления суммы ряда $s=s(1)+\ldots+s(n)$,

где $s(n)=(-1)^n x^{(2n-1)}/(2n+1)$ с точностью до eps=0.001. В качестве параметров выбрать x и eps.

4. Написать функцию, которая вычисляет для заданной квадратной матрицы A её симметричную часть S(ij)=(A(ij)+A(ji))/2 и кососимметричную часть K(ij)=(A(ij)-A(ji))/2.

5. Написать функцию “шапочка” f(x), зависящую от параметров a и b: если |x| >a то f(x)=0 иначе f(x)=b*exp(-a^2/(a^2-$|x|^2$)). В качестве параметров передать a,b,x.

6. Написать функцию поиска максимального и минимального элементов одномерного массива. В основной программе вызвать эту функцию для двух разных массивов.

7. Написать функцию, которая сортирует одномерный массив в порядке убывания методом пузырька. В основной программе вызвать эту функцию для двух разных массивов.

8. Написать функцию, которая по двум заданным одномерным массивам (A размера m и B размера n) вычисляет максимальное значение двумерного массива c(ij)=a(i)*b(j).

9. Написать функцию определителя квадратной матрицы A размера 3x3:

detA=a(1,1)a(2,2)a(3,3)+a(3,1)a(1,2)a(2,3)+a(2,1)a(3,2)a(1,3)-
a(3,1)a(2,2)a(1,3)-a(1,1)a(3,2)a(2,3)-a(2,1)a(1,2)a(3,3).

10. Написать функцию вычисления суммы ряда y=sinx-(sin2x)/2+…

+$(-1)^{n+1}$sin(nx)/n с точностью до eps=0.001. В качестве параметров передать x (в радианах) и eps.

11. Написать функцию вычисления ряда y=x+x^3/3!+…+x^{2n+1}/(2n+1)! с точностью до eps=0.0001. В качестве параметров передать x и eps.

12. Написать функцию обработки матриц A и B одинакового размера m x n. Получить матрицу C =max(a(i,j),b(i,j)), и матрицу D=min(a(i,j),b(i,j)). Матрицы C и D вывести в главной программе.

4. Контрольные вопросы

4.1. Описание функции. Для чего объявляется прототип?

4.2. Что такое формальные и фактические параметры? Локальные и глобальные?

4.3. Как можно передавать массив в функцию?

4.4. Способы вызова функций.

Лабораторная работа №7
Работа со строками и структурами

Цель работы: Изучение основных принципов работы со строками; получение практических навыков использования структур в Си++.

1. Теоретические сведения

1.1. Функции для работы со строками

В Си++ строки хранятся и обрабатываются в виде массивов символов.

Как и обычные массивы, строки могут быть инициализированы при объявлении:

char s[] = "string";

В данном случае компилятор подсчитывает количество символов в двойных кавычках и добавляет служебный символ окончания строки «\0». При объявлении массива с фиксированным размером следует предусмотреть элемент для нулевого символа, т. е. строку s можно было бы объявить следующим образом:

char s[7] = "string";

Поскольку строки – это массивы символов, то их нельзя приравнивать и сравнивать с помощью операций «=» и «==».

Для работы со строками используются следующие функции:

double atof(char * s) – преобразует строку s в вещественно число типа double;

int atoi(char * s), ***long atol(char * s)*** – преобразует строку s в цело число типа int и long int, соответственно;

char * itoa(int v, char * s, int base), ***char * ltoa(long v, char * s, int base)*** – преобразует целое и длинное целое, соответственно, v в строку s; при изображении число используется основание base;

strcpy(char * s1, char * s2) – копирует строку s2 в строку s1;

strcat(char * s1, char * s2) – приписывает строку s2 в строку s1;

int strcmp(char * s1, char * s2) – сравнивает строки s1 и s2; возвращает 0, если строки равны и не 0 в противном случае;

unsigned int strlen(char * s) – вычисляет длину строки s (без учета символа ‘\0’).

Приведем примеры использования данных функций:

```
char s0[] = "100";
int n = atoi(s0);            //n = 100
char s1[10];
char s2[] = "string2";
strcpy(s1, s2);                    //s1 = string2
char s3[] = "string3";
strcat(s2, s3);                    //s2 = string2string3
```

int c = strcmp(s2, s3); //c = -1, т. к. строки не равны
int len = strlen(s3); //len = 7

Для ввода строк с клавиатуры пригоден оператор >>, но его применение ограничено, поскольку этот оператор считает пробелы разделителями. Более удобной оказывается функция ***getline(char * s, int len)***, которая позволяет получить от пользователя строку с пробелами s длиной (len – 1) символов (последний символ - служебный):

```
const max_length = 20;
char s[max_length];
cin.getline(s, max_length);
```

1.2. Хранение данных в виде структур

Как уже было сказано, массивы используются для хранения элементов, являющихся объектами одного и того же типа. Но массивы не предназначены для хранения разнотипной информации. В этом случае удобно использовать ***структуры*** – составной тип данных, объединяющий в одно целое множество элементов в общем случае разных типов.

Описание структуры в языке Си++ имеет следующий синтаксис:

```
struct имя_структуры
    {
    тип_данных1 имя_переменной1;
    тип_данных2 имя_переменной2;
    …
    };
```

Внутренние переменные структуры называются ***полями*** или компонентами структуры.

Приведем примеры описания структур для представления комплексных чисел и дат:

```
struct ComplexNumber
    {
    double re;
    double im;
    };
struct Date
    {
    int year;
    int month;
    int day;
    };
```

Переменные типа структуры описываются аналогично переменным простых типов:

```
ComplexNumber c1;
```

```
ComplexNumber c2 = {7.8, 3.6};
Date d1;
Date d2 = {2012, 8, 10};
```

Доступ к полям конкретного структурного объекта осуществляется операцией ‘.’. Сначала указывается имя переменной-структуры, а затем – имя компоненты, например:

```
c.re = 2.5;
c.im = 4;
d.year = 2011;
d.month = 12;
d.day = 30;
```

Структурные объекты могут быть собраны в массив:

```
Date dates_array[10];
```

Например, используем структуры для хранения данных о людях:

```
#include <iostream.h>
//описание структуры
struct Person
{
char surname[20];
char name[20];
int year;
};
//прототипы функций
void ReadPerson(Person & p);
void WritePeople(Person p[], int count);

void main()
{
Person p;
Person array_people[10];

cout << "**************************" << endl;

ReadPerson(p);
array_people[0] = p;
ReadPerson(p);
array_people[1] = p;

WritePeople(array_people, 2);

cout << "**************************" << endl;
}
```

```
void ReadPerson(Person & p)
{
cout << "Фамилия : ";
cin >> p.surname;
cout << "Имя: ";
cin >> p.name;
cout << "Год рождения: ";
cin >> p.year;
}

void WritePeople(Person p[], int count)
{
cout << "Список людей:" << endl;
for (int i = 0; i < count; i++)
      cout << p[i].surname << " " << p[i].name << " " <<
            p[i].year << endl;
}
```

Результат работы программы:

****

Фамилия: Степанов
Имя: Олег
Год рождения: 1998
Фамилия: Орлов

Имя: Александр
Год рождения: 2001

Список людей:
Степанов Олег 1998
Орлов Александр 2001

****

2. Выполнение работы

Изучить основные функции работы со строками в Си++, а также правила использования структур для хранения разнотипных данных; выполнить полученное от преподавателя задание по применению полученных знаний и закреплению практических навыков.

3. Указания по оформлению отчета

Отчет должен содержать:
- наименование и цель работы;
- краткие теоретические сведения;

- задание на лабораторную работу;
- текст и результаты работы программы.

Варианты заданий

Дана последовательность символов S1, …, SN. Группы символов, разделенные пробелом (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами.

1. Определить число символов в самом длинном слове строки. Слова отделяются знаком “/”.
2. В произвольном тексте выделить и отпечатать слова, начинающиеся с буквы А.
3. В произвольном тексте вставить между первым и вторым словом новое слово.
4. В произвольном тексте найти и отпечатать слова, содержащие букву Е.
5. Отпечатать второе и третье слова произвольного текста.
6. В произвольном тексте вставить между вторым и третьим словом новое слово.
7. В произвольном тексте найти и отпечатать все слова длиной 5 символов.
8. В произвольном тексте найти самое короткое слово.
9. В последовательности из 10 пятибуквенных слов найти и поменять местами пару слов, у которых первые три буквы одного совпадают с последними тремя буквами другого.
10. Упорядочить в алфавитном порядке последовательность из 10 пятибуквенных слов.
11. В строке из 50 символов отдельные слова разделены пробелом. Упорядочить строку так, чтобы каждое следующее слово было не короче предыдущего.
12. Расположить слова строки в порядке, обратным исходному.
13. Подсчитать количество букв ‘а’ в последнем слове строки.
14. Найти количество слов, у которых первый и последний символы совпадают между собой.
15. Исключить из строки слова, расположенные между скобками (,). Сами скобки должны быть исключены.
16. В произвольном тексте найти и отпечатать слова, содержащие букву А.
17. Отпечатать первое и второе слова произвольного текста.
18. В произвольном тексте вставить после первого слова новое слово.
19. В произвольном тексте найти и отпечатать все слова длиной 4 символа.

20. В произвольном тексте найти самое длинное слово.

21. Выполнить сравнение двух строк s и d. Результат вывести в виде сообщения «идентичны» или «не идентичны».

22. Добавить в конец строки новое слово, длинною 5 символов, иначе выдать сообщение об ошибке.

23. Добавить в начало строки новое слово, начинающееся с буквы а, иначе, если слово начинается с другой буквы вывести сообщение о невозможности добавления.

24. Посчитать какое количество раз встречается буква n (задается при каждом выполнении алгоритма).

25. Проанализировать массив символов, состоящий из n символов. Если массив состоит из n-5 символов, добавить в конец набор символов rrrrr.

4. Контрольные вопросы

1. Как задать строку в Си++?
2. Какие функции есть в Си++ для работы со строками?
3. Что такое структура? Чем она отличается от массива?
4. Как обратиться к конкретному полю структуры?

Лабораторная работа №8
Текстовые файлы

Цель работы: Изучение основных принципов работы с текстовыми файлами в Си++.

1. Теоретические сведения

1.1. Доступ к файлам с помощью потоков

Хранение данных в переменных и массивах является временным (до конца выполнения программы). Для постоянного хранения предназначены файлы, которые размещаются на запоминающих устройствах компьютера.

В Си++ работа с файлами осуществляется с помощью ***файловых потоков*** - объектов, которые передают и принимают данные и связываются с файлами.

Потоки для работы с файлами создаются как объекты следующих *классов* (типов данных, объединяющих в себе переменные и функции):

- ***ofstream*** – выходной поток для записи данных в файл;
- ***ifstream*** – входной поток для чтения данных из файла;
- ***fstream*** – для чтения и для записи данных (двунаправленный обмен).

Для реализации операций файлового ввода-вывода нужно включить в программу заголовочный файл ***fstream.h***.

Перед началом работы с файлами нужно создать поток:

ifstream in_stream;

ofstream out_strem;

Здесь создается поток с именем in_stream, являющийся объектом класса ifstream и поток с именем out_stream, являющийся объектом класса ofstream.

После создания потока его можно подключить к файлу (т. е. открыть файл) с помощью функции ***open()***:

имя_потока.open(имя_файла);

Например, операторы

in_stream.open("file.txt");

out_stream.open("file.txt");

подключат потоки in_stream и out_stream к началу файла file.txt. При подключении выходного потока возможны следующие ситуации:

- если файла нет, то он создается;
- если файл есть, то все его содержимое удаляется.

После открытия файла желательно проверять, не произошло ли ошибки во время этой операции. Для проверки можно вызвать функцию ***fail()***, которая возвращает истинное значение, если последняя операция потока привела к ошибке (например, попытка открытия несуществующего файла):

in_stream.fail();

Для отключения потока от файла надо вызвать функцию ***close()***:

in_stream.close();

```
out_stream.close();
```

1.2. Выполнение операций с файлами

После того, как файл открыт для чтения, из него можно считывать отдельные символы. Для этого служит функция ***get()***:

```
char ch;
in_stream.get(ch);
```

Функция ***get()*** считывает символ из файла и записывает его в переменную ch.

В открытый файл можно записывать отдельные символы с помощью функции ***put()***:

```
out_stream.put('k');
```

Функции ***get()*** и ***put()*** предназначены для чтения (записи) только отдельных символов, следовательно, данные других типов (int, double и др.) для записи в файл должны быть преобразованы в последовательность символов. При чтении из файла эти последовательности должны быть преобразованы обратно.

Некоторые преобразования типов данных автоматически выполняются операторами '<< и >>.

При работе с входным потоком надо следить за тем, достигнут ли конец файла или нет. Для этого существует функция ***eof()***, которая равно ***true***, если конец файла достигнут, и ***false*** в противном случае

Например, нужно скопировать текстовый файл file.txt одновременно и на экран, и в другой файл copy_of_file.txt:

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

void main()
{
ifstream in_stream;
ofstream out_stream;

in_stream.open("file1.txt");
out_stream.open("copy_of_file.txt");
cout << "**************************" << endl;

if (in_stream.fail())
  cout << "Ошибка при открытии файла" << endl;
else
  if (out_stream.fail())
    cout << "Ошибка при создании файла" << endl;
  else
    {
    char ch;
```

```
    in_stream.get(ch);

    while (!in_stream.eof())
      {
      cout << ch;
      out_stream.put(ch);
      in_stream.get(ch);
      }
  }

in_stream.close();
out_stream.close();
cout << "**************************" << endl;
}
```

Если в каталоге программы есть текстовый файл file.txt и не возникнет ошибок при создании файла copy_of_file.txt, то в результате работы программы содержимое первого файла будет выведено на экран и записано во второй файл.

2. Выполнение работы

Изучить основные принципы работы с текстовыми файлами в Си++; выполнить полученное от преподавателя задание по применению полученных знаний и закреплению практических навыков.

3. Указания по оформлению отчета

Отчет должен содержать:

- наименование и цель работы;
- краткие теоретические сведения;
- задание на лабораторную работу;
- текст и результаты работы программы.

Варианты заданий

1. Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде:

X Y
5 1
2 8
12 3 и т.д.

Считать из файла пары значений и в тех из них, где X>Y , поменять значения X и Y местами. Результат записать в другой текстовый файл такого же формата.

2. Ввести с клавиатуры попарно значения вещественного типа и записать их в текстовый файл в виде таблицы следующего формата:

X : Y

2.1 : 3.7

6.2 : 5.4 и т.д.

Считать из файла полученные пары значений и создать из них другой файл вида:

sin(x) : cos(y)

значение sin(2.1) : значение cos(3.7) и т.д.

3. Ввести с клавиатуры попарно значения вещественного типа и записать их в текстовый файл в виде таблицы следующего формата:

X : Y

2.1 : 3.7

6.2 : 5.4 и т.д.

Считать из файла полученные пары значений и создать из них другой файл вида:

tg(x) : ctg(y)

значение tg(2.1) : значение ctg(3.7) и т.д.

4. Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде:

X Y

5 1

2 8

12 3 и т.д.

Считать из файла пары значений и в тех из них, где X кратен Y , пометить строку таблицы:

X Y

5 1 ***

2 8

12 3 ***

в том же файле.

5. Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде:

X Y

5 8

2 1

1 3 и т.д.

Считать из файла пары значений и в тех из них, где X меньше Y , пометить строку таблицы:

X Y

5 8 ###

2 1

1 3 ###

в том же файле.

6. Случайным образом создать таблицу значений и записать её в текстовый файл в виде:

a b c

5.2 4.6 2.5

1.2 8.9 2.3 и т.д.

Считать из файла записанные данные и определить, можно ли построить треугольник с такими сторонами. Пометить соответствующие строки таблицы (в том же файле).

a b c

5.2 4.6 2.5 можно

1.2 8.9 2.3

7. Создать текстовый файл, содержащий вещественные значения. Считать из файла записанные данные и определить максимальное значение. Если оно находится в первой половине файла, заменить его суммой последующих элементов, если во второй – суммой предыдущих элементов.

8. Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде:

X Y

5 25

1 3

49 7 и т.д.

Считать из файла пары значений и в тех из них, где X является точным квадратом Y или наоборот, найти сумму значений X и Y. Результат записать в другой текстовый файл в виде

X Y sum

5 25 30

1 3

49 7 56

9. Создать текстовый файл, содержащий целые числовые значения, например : 5 3 21 4 37 52 9 2. Считать из файла записанные данные и определить минимальное значение. Если оно кратно трем, заменить каждое третье значение файла нулем, если кратно пяти – заменить его суммой первого и последнего элементов.

10. Создать текстовый файл, содержащий целые числовые значения, например: 15 13 21 42 37 50 9 2. Считать из файла записанные данные и заменить нулем каждое значение файла, кратное минимальному числу.

11. Ввести с клавиатуры значения вещественного типа и записать их в текстовый файл в виде таблицы следующего формата:

X : Y : Z

2.1 : 3.7 : 0.9

6.2 : 5.4 : 4.2 и т.д.

Считать из файла полученные значения и создать из них другой файл вида:

sin(max{X,Y,Z}) : cos(min{X,Y,Z})
значение sin(3.7) : значение cos(0.9)

4. Контрольные вопросы

1. Что такое файловые потоки?

2. Какие классы файловых потоков есть в Си++?

3. Как связать файловый поток с конкретным файлом?

4. С помощью каких функций Си++ можно читать из файла и записывать в файл?

5. Как проверить, достигнут ли конец файла?

Лабораторная работа №9
Программные модули

Цель работы: Изучение основных принципов написания программных модулей; получение практических навыков вызовов функций из стандартных и собственных модулей.

1. Теоретические сведения

Модульное программирование предполагает разделение текста программы на несколько файлов, в каждом из которых сосредоточены независимые части программы (сгруппированные по смыслу функции).

Библиотека функций Си++ содержит достаточно большое количество стандартных функций, разбитых на модули. Чтобы иметь возможность пользоваться функциями, их описания в виде прототипов необходимо включать в программу дополнительно. Обычно это делается с помощью препроцессорных директив

#include имя_файла

Здесь *имя_файла* определяет *заголовочный* файл, содержащий прототипы той или иной группы стандартных для данного компилятора функций. По правилам Си++ заголовочный файл имеет расширение ***.h***.

Подобно тому, как это сделано в библиотеке стандартных функций компилятора, следует поступать и при разработке своих программ, состоящих из достаточно большого количества функций, размещенных в разных модулях. Прототипы функций и описания внешних объектов (переменных, массивов и т. д.) помещают в *заголовочный файл*, а исходный текст (описания функций) – в *файл реализации* (с расширением ***.cpp***).

Заголовочный файл подключается к каждому модулю программы, но, в отличие от библиотечных функций, имя такого заголовочного файла в команде ***#include*** записывают не в угловых скобках <>, а в кавычках *""*.

Например, создадим файл с функциями для нахождения максимального и минимального элементов целочисленного массива, а также вычисления среднего значения:

```
// unit.cpp – файл реализации
int Max(int array[], int size)
{
int max = -1e5;
for (int i = 0; i < size; i++)
  if (array[i] > max) max = array[i];
return max;
}
```

```
int Min(int array[], int size)
{
int min = 1e5;
for (int i = 0; i < size; i++)
  if (array[i] < min) min = array[i];
return min;
}

double Average(int array[], int size)
{
int s = 0;
for (int i = 0; i < size; i++)
  s += array[i];
return ((double)s / size);
}
```

Создадим одноименный заголовочный файл, куда поместим прототипы функций:

```
// unit.h – заголовочный файл
int Max(int array[], int size);
int Min(int array[], int size);
double Average(int array[], int size);
```

Текст основной программы будет иметь вид:

```
#include <iostream.h>
#include "unit.h" // подключение модуля unit.h

void main()
{
int n;
int array[20];

cout << "Введите размер массива = ";
cin >> n;
cout << endl;

cout << "Введите элементы массива " << endl;
for (int i = 0; i < n; i++)
  cin >> array[i];

cout << "Max = " << Max(array, n) << endl;
cout << "Min = " << Min(array, n) << endl;
cout << "Average = " << Average(array, n) << endl;
```

}

Программа демонстрирует основное достоинство модульного подхода: при изменении деталей реализации в файле *unit.cpp* не обязательно вносить изменения в файл *unit.h* или в главный файл программы.

2. Выполнение работы

Изучить основные правила создания и использования программных модулей; выполнить задание, описанное в методических указаниях. Проверить, действительно ли при изменении деталей реализации в файле *unit.cpp* не обязательно вносить изменения в файл *unit.h* или в главный файл программы.

3. Указания по оформлению отчета

Отчет должен содержать:
- наименование и цель работы;
- краткие теоретические сведения;
- задание на лабораторную работу;
- текст и результаты работы программы.

4. Контрольные вопросы

1. Что такое модульное программирование?
2. Какие достоинства имеет принцип модульного программирования?
3. Что такое файл реализации и заголовочный файл?
4. Как подключить модуль к основной программе?