ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Федеральное государственное образовательное учреждение
высшего профессионального образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ЧЕРДЫНЦЕВА М.И.

РАЗРАБОТКА ПРОГРАММ НА ЯЗЫКЕ C++ В СРЕДЕ CODE::BLOCKS

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

для студентов 1–2 курса дневного и вечернего отделений
факультета математики, механики и компьютерных наук

Ростов-на-Дону
2008

Методические указания разработаны сотрудником кафедры прикладной математики и программирования: кандидатом технических наук, доцентом М.И.Чердынцевой.

В методических указаниях представлены сведения об использовании визуальной среды программирования Code::Blocks при разработке программ на C++. Описаны основные приемы управления проектами, использование отладчика. Все этапы работы демонстрируются на подробно разобранных примерах.

Методические указания разработаны с учетом кредитно-модульной системы обучения. Представленный в методических указаниях материал используется в модуле курса «Языки программирования и методы трансляции» – «Основы языка C++, основные конструкции, типы данных, указатели и массивы. Многомодульная структура программы». Объем методических указаний соответствует четырем лабораторным работам. Излагаемый материал сопровождается контрольными вопросами и заданиями для самостоятельного выполнения.

Методические указания предназначены для студентов 1–2 курса специальностей «прикладная математика» и «информационные технологии», начинающих изучение языка C++ как второго языка программирования.

Печатается в соответствии с решением кафедры прикладной математики и программирования факультета математики, механики и компьютерных наук ЮФУ, протокол № 2 от 2 октября 2008 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Конкретные действия по выполнению программы зависят от операционной среды, в которой предполагается работать, от используемого компилятора с языка C++, а также возможно от среды разработки. В настоящее время имеются удобные визуальные среды для разработки программ.

Однако в любом случае, для того чтобы выполнить программу на компьютере, необходимо выполнить следующие действия:

1. подготовить текст программы и сохранить его в файле. Файл, содержащий текст программы называется исходным модулем.

2. выполнить компиляцию исходного модуля. В процессе компиляции будет произведена трансляция с языка C++ в язык машинных команд. Результат компиляции, если она прошла без ошибок, сохраняется в файле. Файл, содержащий откомпилированную программу, называется объектным модулем.

3. выполнить компоновку программы. В процессе компоновки объектный модуль объединяется с другими объектными модулями, содержащими функции, используемые в программе. Результат компоновки дополняется стандартным кодом начальной загрузки, при этом получается выполняемая версия программы. Файл, содержащий окончательный результат, называется исполняемым модулем.

Каждый из вышеперечисленных шагов может быть выполнен путем вызова соответствующей программы (текстового редактора, компилятора, редактора связей или компоновщика) и передачи ей в качестве входных параметров соответствующих имен файлов – входных и выходных.

Удобство использования визуальных сред разработки, заключатся в том, что они позволяют выполнить весь цикл работы, используя текстовые и/или кнопочные меню. Они могу включать такие средства как контекстные подсказки, шаблоны, которые облегчают подготовку программ.

МОДУЛЬ 1

Начальные сведения о программировании на языке C++ и работе в визуальной среде Code::Blocks

Задачи и цели: Ознакомление с организацией простейшей программы на языке C++. Овладение навыками создания проекта в визуальной среде Code::Blocks. По завершении изучения модуля студент должен уметь создавать новые проекты, работать (открывать, модифицировать и выполнять) с существующими проектами.

1 РАБОТА В СРЕДЕ CODE::BLOCKS

В методических указаниях рассматриваются приемы разработки и отладки программы в интегрированной среде разработки Code::Blocks. Описание ориентировано на версию 1.0 в сборке от ноября 2006 года.

Окно IDE Code::Blocks при запуске выглядит, как показано на рисунке 1.

Для работы необходимо или создать новый проект, или открыть существующий. *Проект* – это набор файлов: заголовков, текстов программ, ресурсов, установок, конфигураций.

2 ПРОЕКТЫ, СОДЕРЖАЩИЕ ОДИН МОДУЛЬ

2.1 Создание проекта

Для создания нового проекта необходимо в меню выбрать команду File–New–Project.

Появляющееся после этого диалоговое окно предназначено для определения шаблона создаваемого проекта. Решение задач будет рассматриваться для консольных приложений. Поэтому среди предлагаемых шаблонов используем шаблон Console Application. После выбора соответствующей шаблону пиктограммы следует нажать на кнопку “Go”.

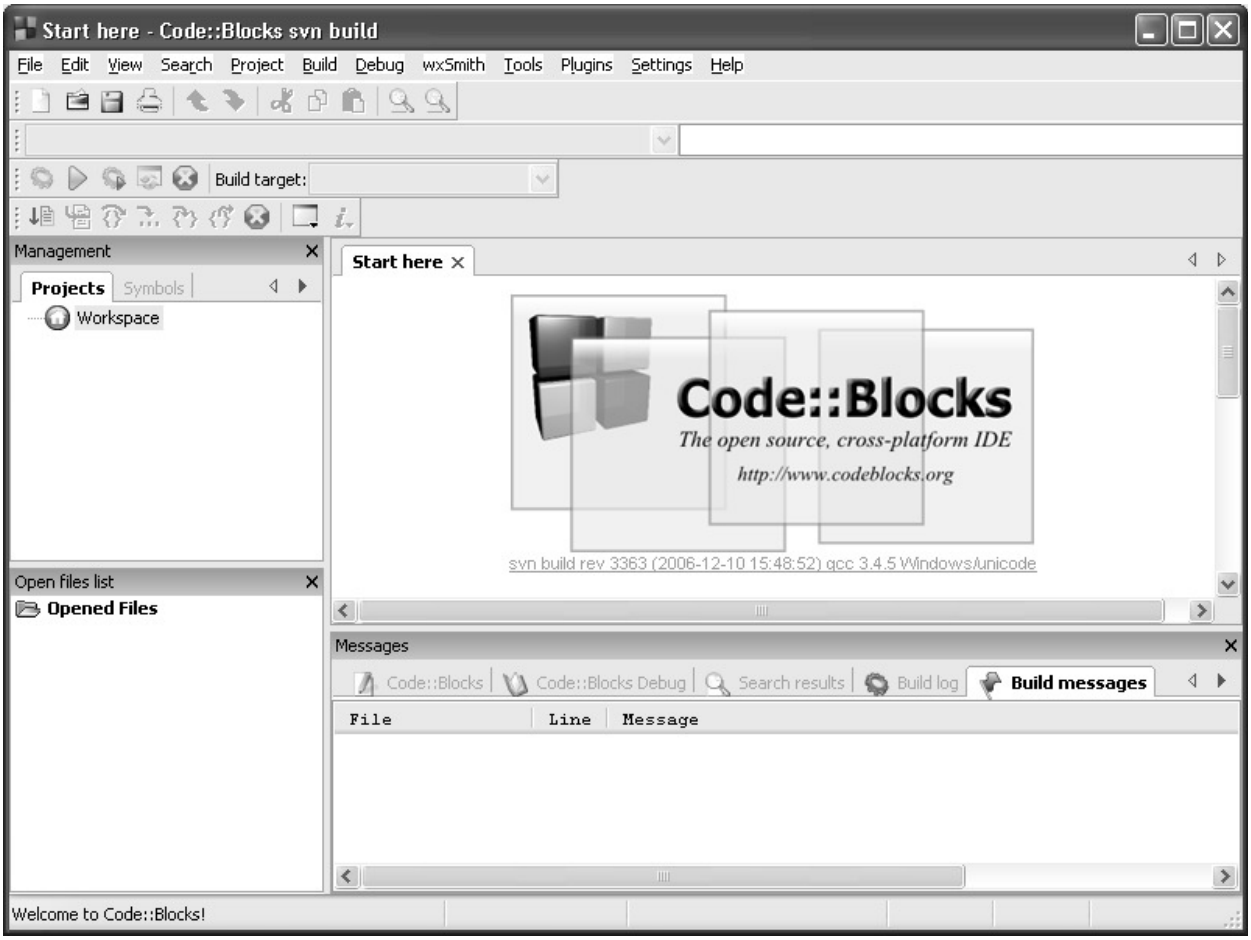

Рисунок 1

Создание шаблона предполагает выполнение последовательности шагов под управлением мастера. Первое окно, представленное на рисунке 2, предназначено для определения имени проекта и его местоположения.

В процессе ввода имени создаваемого проекта и выбора каталога для расположения файлов проекта, будет меняться информация в окнах Project filename и Resulting filename – эти окна служат для контроля ввода. Следует отметить, что в процессе работы в указанном каталоге будут создаваться различные файлы, в том числе и исполнимые файлы с расширением exe. Это следует учесть при выборе каталога для размещения проекта. Например, в учебных компьютерных классах может быть запрещено сохранение исполнимых файлов на сетевых дисках и системных дисках персональных компьютеров.

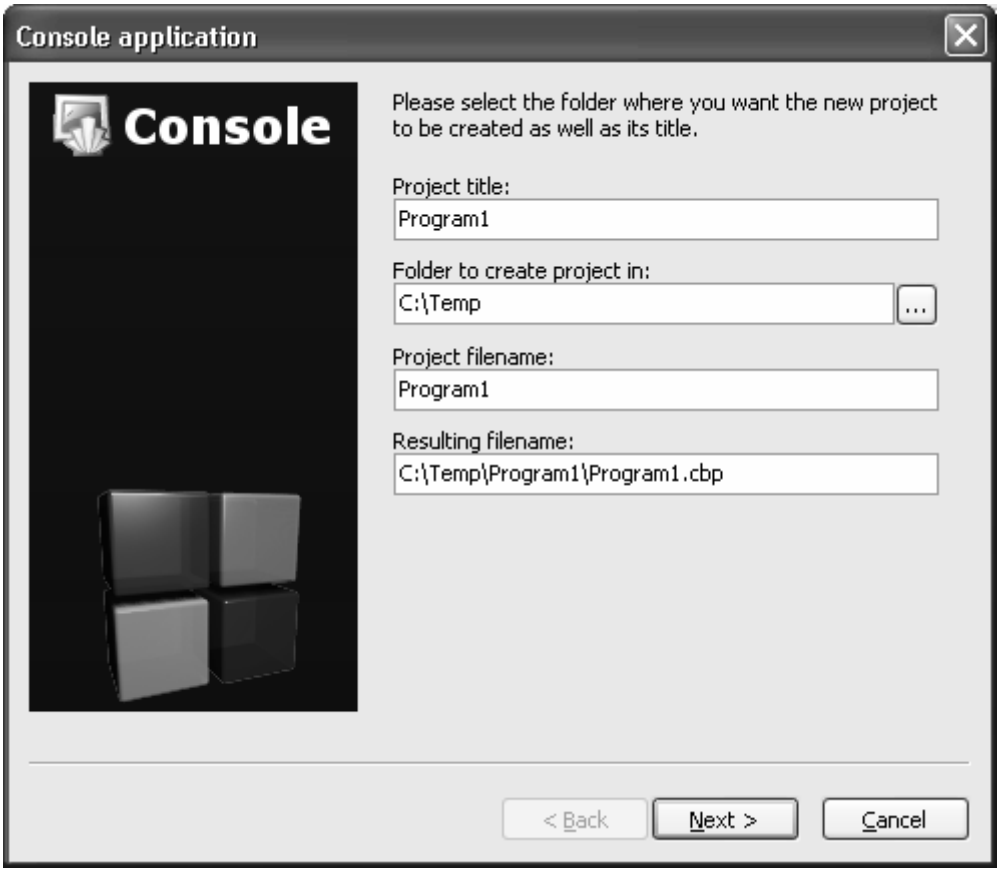

Рисунок 2

После определения имени и положения проекта мастер предлагает выбрать используемый компилятор и конфигурацию проекта, как показано на рисунке 3. Можно использовать параметры, предлагаемые по умолчанию.

Последний шаг в определении создаваемого файла – выбор используемого языка программирования – представлен на рисунке 4.

Для завершения работы с мастером создания проектов необходимо нажать кнопку Finish.

Созданный новый проект добавляется к рабочему пространству (Workspace) Code::Blocks, состояние которого отображается в специальном окне менеджера (Management). Проект содержит папку исходных модулей (Sources), в ней находится один файл с исходным модулем с именем main.cpp. При создании непустого проекта в него всегда помещается файл, содержащий функцию `main()` и ему дается имя main.cpp.

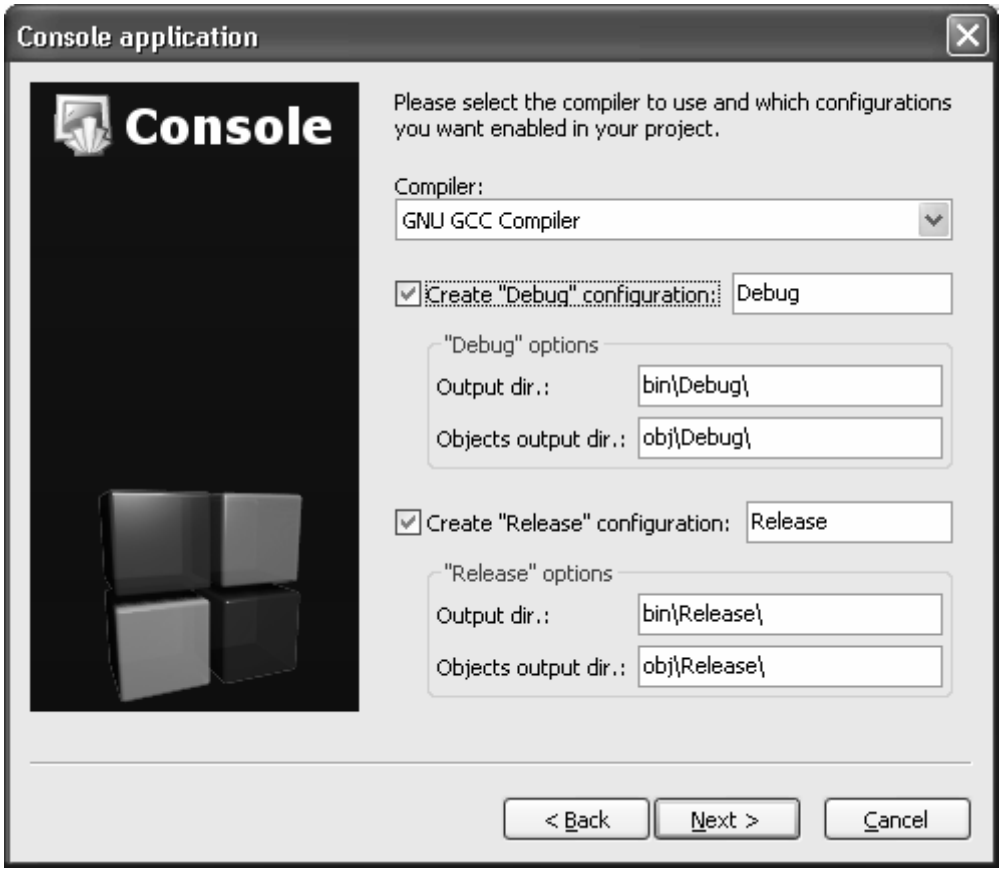

Рисунок 3

Если посмотреть папку C:\Temp, то в ней будет расположена папка нового созданного проекта C:\Temp\Program1. В этой папке находятся два файла: файл проекта – Program1.cbp; и файл main.cpp.

Чтобы увидеть содержимое файла main.cpp достаточно щелкнуть на нем в окне менеджера. При этом в рабочей области Code::Blocks появится вкладка, на которой будет представлен этот файл, доступный для редактирования (рисунок 5).

Создаваемый для консольных приложений файл main.cpp содержит готовую к выполнению программу “Hello world!”. Текст этой программы выглядит следующим образом:

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

Рисунок 4

Первая строка содержит директиву препроцессора:

```
#include <iostream>
```

В языке C++ определены специальные файлы заголовков (header), которые содержат информацию, необходимую компилятору. Директива препроцессора `#include` сообщает о том, какой заголовочный файл должен быть включен в исходный текст программы.

Подключение файла `<iostream>` необходимо для того, чтобы можно было использовать для вывода объект `cout`. Наиболее просто организовать ввод–вывод в программах на C++ используя стандартные потоковые объекты `cin` и `cout`. Эти объекты связаны со стандартными устройствами консольного ввода и вывода – клавиатурой и дисплеем.

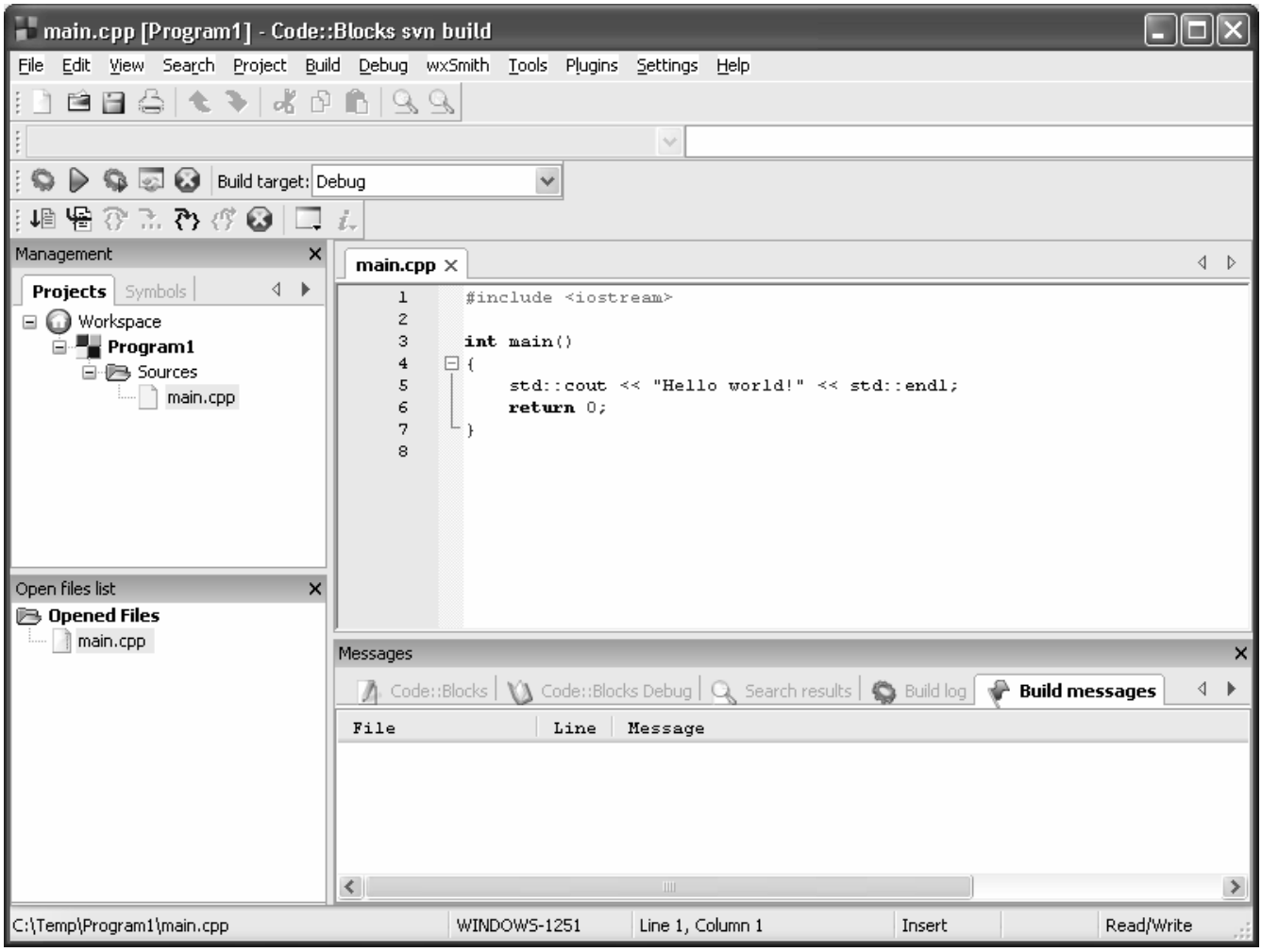

Рисунок 5

Поскольку имена `cin` и `cout` в приводимой программе не описаны, необходимо указать пространство имен, в котором они определены. Для всех стандартных имен языка C++ пространство имен называется `std`.

При обращении к объектам, объявленным в этом пространстве, имя пространства имен может быть указано явно. Для этого используется операция расширения области видимости **::**. Такой способ удобно использовать, если к какому-то пространству имен потребуется одно–два обращения. В противном случае, чтобы не загромождать текст программы лучше выносить определение используемого пространства имен оператор `using namespace`.

Тогда текст программы “Hello World” будет выглядеть так:

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

Выполнение программы на C++ всегда начинается с вызова функции с именем `main()`. Эта же функция должна вернуть управление операционной системе после своего завершения и сообщить код возврата. В случае аварийного завершения операционной системе будет сообщено ненулевое целочисленное значение – код ошибки (код возврата). Естественно предположить, что в случае безошибочной работы нужно возвратить признак нормального завершения `0`. Именно поэтому в заголовке функции `main()` указано, что возвращаемый ею результат должен быть целого типа. А последним оператором тела функции `main()` является оператор возврата:

```
return 0;
```

Чтобы проверить работоспособность программы Hello World, выполним ее. Для этого нужно последовательно провести компиляцию, компоновку и выполнение. Сделать это можно через команду меню Build – Build and run, или нажав клавишу F9, или выбрав соответствующую пиктограмму на кнопочной панели Compiler Toolbar , приведенной на рисунке 6.

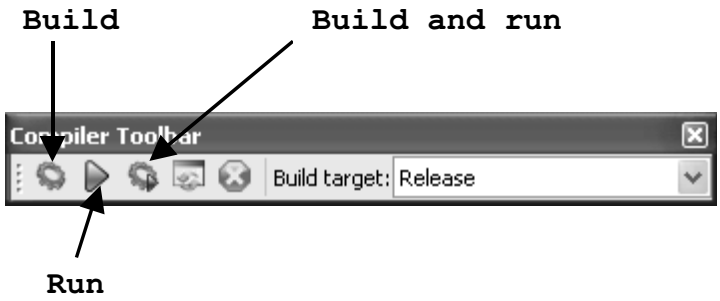

Рисунок 6

Выбрав любой из вышеперечисленных способов запуска программы, можно увидеть в окне Messages–Build log сообщения о последовательном выполнении всех этапов. В этом же окне будут выдаваться и сообщения об ошибках. Поскольку программа не содержит ошибок, для ее выполнения будет открыто подчиненное окно выполнения консольного приложения. Как показано на рисунке 7, выполнение программы заключается в выводе строки “Hello world!”. Результат вывода виден в первой строке окна выполнения. Ниже выдается сообщение о том, что процесс выполнения программы завершен с кодом возврата 0 и время, затраченное на выполнение. В последней строке выводится подсказка о том, что для закрытия окна выполнения консольного приложения и возврата в среду Code::Blocks необходимо нажать любую клавишу.

Упражнение. Измените текст программы, так как было показано выше, вынеся определение используемого пространства имен. Проверьте, что результат выполнения не изменится.

Если теперь открыть папку, в которой располагается проект (C:\Temp\Program1), то можно увидеть, что в ней появился исполнимый файл program1.exe. Кроме этого должна появиться одна из папок \obj\Release или \obj\Debug , в которой находится файл с именем main.o – это объектный модуль программы.

При создании проекта были выбраны две предлагаемые конфигурации – для отладки (Debug) и для реализации (Release). Они отличаются тем, что при создании конфигурации Debug в код объектного модуля вставляется дополнительная информация, которая может понадобиться отладчику. В код для реализации эта информация не добавляется. Чтобы получить объектный модуль для отладчика следует переключить конфигурацию (Build target) с Release на Debug как показано на рисунке 6.

Рисунок 7

Если сравнить размеры полученных объектных файлов, то можно увидеть, насколько отладочная информация увеличивает размер объектного модуля.

Когда не нужно запускать программу на выполнение, то компиляцию и компоновку можно выполнить, используя команду Build (Ctrl-F9) или выбрать соответствующую кнопку на панели Compiler Toolbar (рисунок 6). Уже готовую к выполнению программу можно запустить командой Run (Ctrl-F10) или соответствующей кнопкой на панели.

Поскольку каждая законченная программа будет содержать, по крайней мере, все вышеперечисленные модули и файлы, для каждого проекта формируется отдельный каталог. Следует учитывать, что просто собрать нужные файлы и разместить их в соответствующих каталогах – это не значит создать проект. Описание проекта хранится в файле с расширением cbp. Это текстовый файл, содержимое которого можно просмотреть в любом редакторе. Однако создавать или править содержимое файла проекта вручную не рекомендуется.

Рассмотрим еще раз окно для работы с проектом на рисунке 5. Окно разделено на несколько панелей, каждая из которых может быть скрыта или

выведена на экран через пункты меню View. Размеры и положение панелей может быть изменено по желанию.

В панели Management представлена информация об открытых в рабочем пространстве проектах. Если в рабочем пространстве открыто несколько проектов одновременно, то следует внимательно следить за тем, какой из проектов является в данный момент активным. Именно для него будут выполняться команды Build и Run. Чтобы переключиться на другой проект, нужно сделать его активным, используя команду контекстного меню Activate project. Ненужные в данный момент проекты лучше закрыть.

Панель Open file list содержит список файлов открытых в панели редактора. Файлы, входящие в проект проще всего открыть двойным щелчком на их имени. Если нужны другие файлы, они открываются командой меню File–Open.

Панель редактора является основной рабочей областью, в которой каждый открытый файл представляется на отдельном листе–закладке. Имя листа совпадает с именем открытого файла.

Панель Messages содержит несколько вкладок для отображения различных сообщений. На вкладке Build log выдается информация о процессе компиляции и построения исполнимого модуля программы. Если в процессе компиляции или компоновки возникают ошибки, они отражаются на вкладке Build messages.

2.2 Сохранение проекта

Существуют различные варианты сохранения, например: сохранение всего проекта, целиком со всеми изменениями; сохранение текущего файла, открытого в окне редактирования, сохранение рабочего пространства.

Сохранение активного проекта выполняется через команду меню File–Save project. Чтобы сохранить все проекты, открытые в рабочей области используется команда File–Save all projects. Если необходимо сохранить проект, изменив его имя, используют команду File–Save project as. При закрытии проекта, если в

него были внесены изменения и не сохранены, выдается предупреждающее сообщение (рисунок 8).

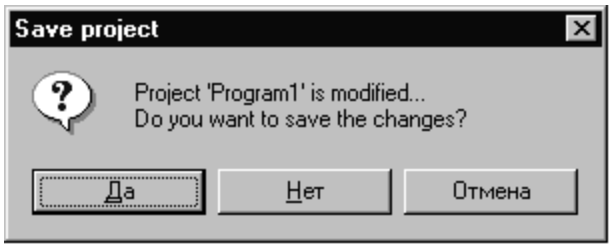

Рисунок 8

Для сохранения текущего файла, открытого в окне редактирования, с сохранением имени файла можно выполнить команду File–Save из меню (или горячие клавиши Ctrl+S или пиктограмма на панели пиктограмм).

Для сохранения текущего файла, открытого в окне редактирования, с возможностью изменения имени файла нужно выполнить команду File–Save as.

Для сохранения всех открытых в окне редактирования файлов предназначена команда меню File–Save all files. При попытке закрыть в окне редактирования файл, в который были внесены изменения, выдается предупреждающее сообщение.

Сохранение рабочего пространства удобно в тех случаях, когда желательно иметь быстрый способ открыть все в том же виде, как и при завершении работы.

2.3 Открытие существующего проекта

Если необходимо продолжить работу над сохраненным ранее проектом, следует его открыть.

Универсальный способ – использовать команду меню File–Open (Ctrl–O) или пиктограмму со стартовой страницы программы Code::Blocks (рис. 9).

На стартовой странице приводится список последних проектов, с которыми велась работа. Можно выбрать нужный проект из этого списка. Аналогичную возможность предоставляет команда меню File–Recent projects.

С помощью команд File–Open и File–Recent projects можно открыть не только проект, но и рабочее пространство. Файлы проектов имеют расширение cbp, файлы рабочих пространств – расширение workspace.

Файлы, не относящиеся к проекту, открываются командой File–Open или могут быть выбраны из списка недавно открытых файлов File–Recent files.

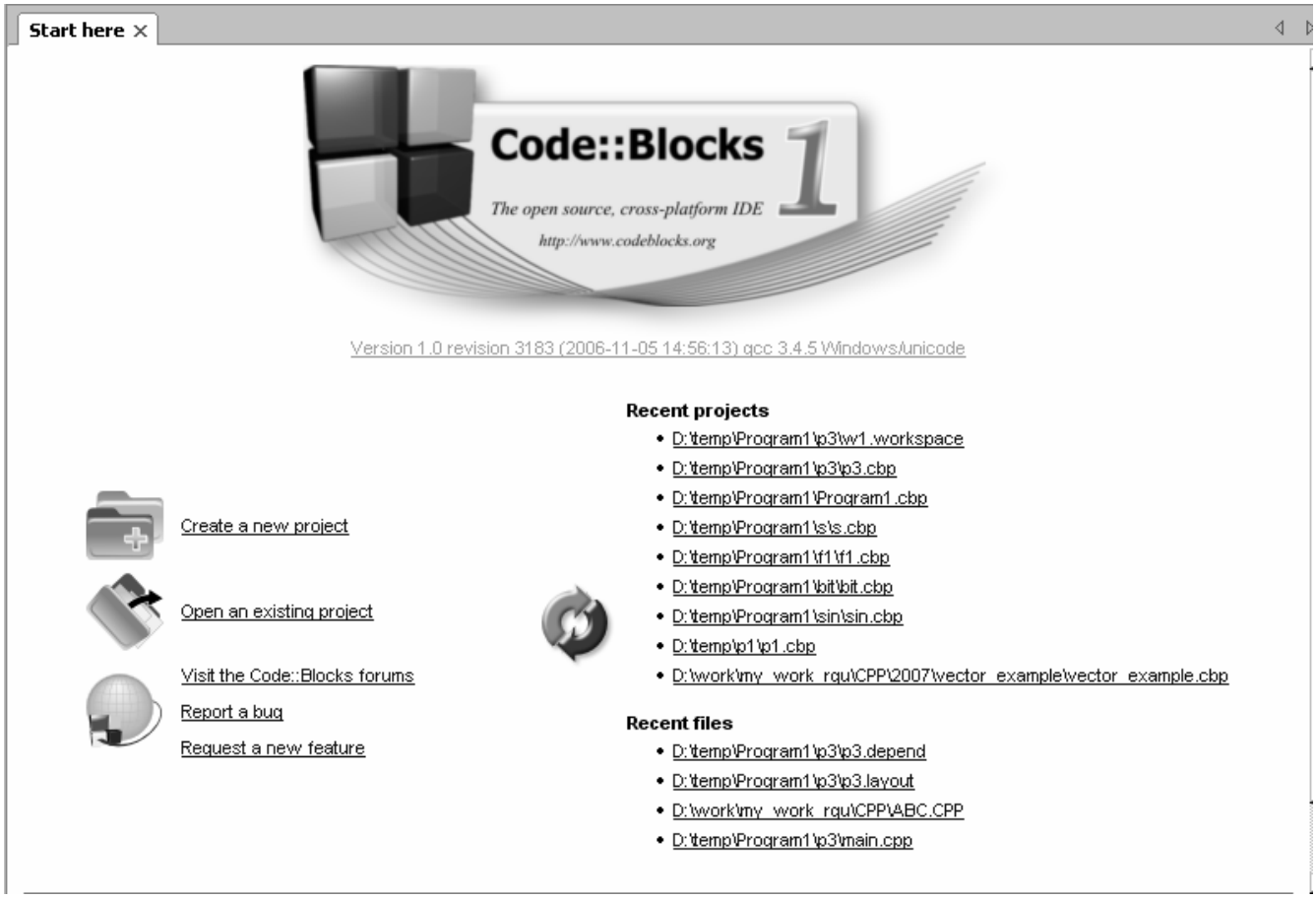

Рисунок 9

Тесты рубежного контроля

1) Что является основным элементом управления при разработке программ на C++ в визуальной среде программирования:
   a) Файл, содержащий текст программы на языке C++
   b) Отдельная функция на языке C++
   c) Несколько файлов, содержащихся в одной папке
   d) Проект

2) Что происходит при запуске программы на выполнение
   a) Начинается выполнение функции, записанной первой в исходном файле
   b) Начинается выполнение функции main().
   c) Начинается выполнение функции, имя которой совпадает с именем проекта.
3) В чем заключается отличие конфигурации Release от Debug
   a) Между ними нет отличий
   b) Конфигурация Debug не позволяет создать исполнимый модуль
   c) Конфигурация Debug добавляет в объектный код программы дополнительную информацию, необходимую при отладке.

Задания к модулю

1. Написать функцию, вычисляющую площадь треугольника по заданным длинам его сторон. Создать проект для решения треугольников по трем сторонам.

2. Написать функцию определения расстояния между двумя точками, заданными декартовыми координатами на плоскости. Создать проект для отладки данной функции.

3. Написать функцию, определяющую наибольший общий делитель двух целях чисел с использованием алгоритма Евклида. Написать функцию, преобразующую обыкновенную дробь, заданную в виде числителя и знаменателя к несократимому виду. Написать функцию, печатающую обыкновенную дробь. Создать проект, в котором будут использованы все вышеперечисленные функции.

В результате изучения модуля студент должен уметь:

1. Создавать новые проекты для работы с программами на C++, сохранять их, компилировать, отлаживать и выполнять.
2. Работать (открывать, модифицировать и выполнять) с существующими проектами.
3. Писать простейшие программы на языке C++, содержащие одну или несколько функций.

МОДУЛЬ 2

Принцип раздельной компиляции, многофайловая структура проекта

Задачи и цели: Ознакомление с принципами организации программ на C++ в виде нескольких исходных модулей. Изучение структуры и назначения заголовочных файлов. По завершении изучения модуля студент должен уметь выделять при решении сложных задач логически связные функции и оформлять их в виде отдельных файлов, писать заголовочные файлы для подключения своих функций.

3 ПРОЕКТ, СОСТОЯЩИЙ ИЗ НЕСКОЛЬКИХ ФАЙЛОВ

Программа на языке C++ состоит из одной или нескольких функций, объявлений и определений глобальных объектов, подключения пространства имен и заголовочных файлов. Среди функций должна быть одна и только одна с именем `main()`. Все функции, составляющие программу, могут быть расположены в одном файле – исходном модуле. Реальные программы на C++, как правило состоят из нескольких файлов с исходными модулями. Это возможно благодаря тому, что C++ и C поддерживают раздельную компиляцию модулей.

Пример 1.

Дано целое число, определить количество и сумму цифр в десятичной записи этого числа. Выдать само число и число, получающееся из него при вычеркивании первой и последней цифр.

Оформим действия определения количества цифр в десятичной записи числа, вычисления суммы цифр числа, преобразования числа путем вычеркивания из него первой и последней цифры в виде функций.

Рассмотрим сначала вариант, когда все пользовательские функции и функция `main()` располагаются в одном файле.

```
#include <iostream>
using namespace std;
/*   прототипы заголовков функций
     сами функции описаны после функции main()
*/
int count_dig(int a);//параметр – значение
int sum_dig(int a);
void del_last_dig(int& a);//параметр-ссылка будет изменен в функции
int del_first_dig(int& a);

int main()
{
    int x;
    cin>>x;
    cout<<count_dig(x)<<endl;
    cout<<sum_dig(x)<<endl;
//следующие две функции изменяют значение параметра
//поэтому сохраним исходное число во вспомогательной переменной
    int r=x;
    del_last_dig(r);
    cout<<x<<' '<<r<<endl;
    cout<<del_first_dig(r)<<' '<<r;
}

int count_dig(int a)
{
    int k=0;
    while (a!=0)
    {
        k++;        //операция увеличения
        a/=10;      //составная операция присваивания
    }
    return k;
}

int sum_dig(int a)
{
    int s=0;
```

```
    a=abs(a);
    while (a!=0)
    {
        s+=a%10;
        a/=10;
    }
    return s;
}

void del_last_dig(int& a)
{
    a/=10;
}
int del_first_dig(int& a)
{
    int k=count_dig(a);
    int r=1;
    for (int i=0; i<k-1; i++) r*=10;
    a%=r;
    return a;
}
```

Рассмотрим некоторые моменты, касающиеся синтаксиса.

Комментарии не являются синтаксическими единицами, но играют важную роль в оформлении текста программы. В языке C++ имеются комментарии двух видов. Комментарий, начинающийся парой символов `//` и заканчивающийся концом строки, является однострочным. Многострочный комментарий начинается парой символов `/*` и заканчивается парой символов `*/`. Многострочный комментарий может занимать одну или несколько строк, а также только часть строки.

Перед определением функции `main()` расположены *прототипы* заголовков пользовательских функций. В отличие от языка C в C++ рекомендуется всегда описывать прототипы заголовков функций. Прототипы необходимы в том случае, когда программа имеет многомодульную структуру и описание функции

располагается не в том файле, где она используется. Поэтому написание прототипов функций является хорошей подготовкой к организации многомодульной структуры программы.

Первые две функции – для вычисления количества и суммы цифр числа – используют *передачу параметров по значению*.

```
int count_dig(int a);//параметр – значение
int sum_dig(int a);
```

Если фактический параметр будет передаваться по значению, то при описании формального параметра указывается только его имя и тип.

В функциях, которые вычеркивают одну из цифр числа (первую или последнюю), параметр *передается по ссылке*.

```
void del_last_dig(int& a);//параметр-ссылка будет изменен в функции
int del_first_dig(int& a);
```

Такой способ передачи параметров появился только в C++. Чтобы определить способ передачи параметра по ссылке, после указания типа формального параметра ставится модификатор ссылки `&`.

Различие в прототипах заголовков не связано с различием в алгоритмах соответствующих функций. Оно призвано только продемонстрировать два стиля описания и использования функций. В первом случае описание более соответствует стилю описания процедур: функция не возвращает никакого результата. Второй вариант описания более соответствует стилю описания функций в языках C и C++. Если в результате выполнения функции изменяется один из параметров, то полученное после изменения значение возвращается как результат функции. В коде функции `main()` видно, что функцию `del_last_dig()` необходимо вызывать оператором вызова функции, а затем использовать полученное значение фактического параметра.

Функцию `del_first_dig()` можно вызвать аналогичным образом, но в программе показано, что возвращаемое функцией значение можно использовать

сразу, например, поместив его в поток вывода. Для сравнения сразу за значением возвращаемым функцией выдается значение фактического параметра.

Тело функции `main()` представляет собой линейную программу, содержащую *операторы* следующих видов:

1) операторы объявления (или описания)

```
int r;
int r=x;
```

Описание вспомогательной переменной `r` размещено не вначале блока, а в том месте, где эта переменная используется. Одновременно с объявлением переменной происходит ее инициализация.

2) оператор ввода

```
cin>>x;
```

3) операторы вывода

```
cout<<count_dig(x)<<endl;
cout<<sum_dig(x)<<endl;

cout<<x<<' '<<r<<endl;
cout<<del_first_dig(r)<<' '<<r;
```

В операторах ввода-вывода указываются имена *стандартных потоковых объектов* `cin` и `cout`.

Эти объекты связаны со стандартными устройствами консольного ввода и вывода – клавиатурой и дисплеем.

Для ввода данных объект `cin` использует операцию >>. Справа от этого знака находится переменная, принимающая вводимую информацию. В процессе ввода последовательность символов, вводимых с клавиатуры, преобразуется к типу, соответствующему переменной, принимающей информацию. Если это невозможно, генерируется ошибочная ситуация.

Для вывода используется объект `cout` и операция <<.

Использование объектов `cin` и `cout` возможно потому, что в программе подключен файл `<iostream>` и объявлено пространство имен `std`, где определены эти объекты. Отметим также, что операции >> и << могут быть применены последовательно многократно. Символьные строки при выводе должны заключаться в двойные кавычки, одиночные символы – в апострофы. При выводе используется манипулятор `endl` – он тоже определен в пространстве имен `std`. Манипулятор `endl` очищает буфер вывода и добавляет в поток `cout` символ новой строки.

4) оператор вызова функции

```
del_last_dig(r);
```

5) оператор возврата результата выполнения функции

```
return 0;
```

Основные действия по обработке данных в языке C++, как и в других процедурных языках, реализуются в виде подпрограмм. В C++ единственным видом подпрограммы является функция. Функция может быть вызвана как операция, т.е. использована в выражении соответствующего типа. Кроме этого функция может быть вызвана оператором вызова функции.

Оператор вызова функции (в отличие от операции вызова функции) используется в одном из следующих случаев:

– если результат функции не будет использован, а функция является функцией с побочным эффектом;

– когда функция является функцией с побочным эффектом, но не возвращает результата. В последнем случае функция семантически эквивалентна процедуре.

Теперь рассмотрим определение пользовательских функций. В общем случае определение функции состоит из заголовка функции и тела функции. Заголовок совпадает с прототипом функции (с тем только исключением, что в прототипе имена формальных параметров можно опускать). Тело функции всегда

представляет собой блок и должно быть заключено в фигурные скобки. Определения функций следуют одно за другим.

При реализации функций, кроме уже рассмотренных операторов, используются операторы:

1) цикла

```
while (a!=0)
{
        k++;
        a/=10;
}
```

```
for (int i=0; i<k-1; i++) r*=10;
```

Оператор цикла с предусловием является универсальным оператором цикла:

```
while (условие)
 оператор;
```

Общая форма цикла for выглядит так:

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

В данном примере можно провести аналогию оператора `for` в языке C++ и цикла `for` в языке Паскаль.

2) инкремента

```
k++;
```

Операция инкремента `k++` является сокращенной формой операции присваивания следующего вида: `k=k+1`. Однако компилятор языка при трансляции такой операции должен использовать более эффективную реализацию – заменить, по возможности, операцию сложения на машинную операцию увеличения (`inc`). Аналогичный смысл имеет операция декремента `(--)`.

Операция присваивания, завершающаяся точкой с запятой, представляет оператор присваивания.

3) комбинированные или составные операторы присваивания

```
a/=10;
```

```
s+=a%10;
```

Составная операция присваивания вида `s+=a` является сокращенной записью операции присваивания вида `s= s+a`.

Для вычисления остатка от деления двух целых чисел используется операция `%`. Операция деления `/` для целых операндов определена как целочисленное деление.

Упражнение 1. Прежде чем разбивать программу на несколько модулей, убедитесь, что она работает правильно – создайте новый проект, наберите вышеприведенный текст и выполните программу для различных целых чисел.

Разместим теперь описание функций, работающих с цифрами десятичного представления числа в отдельном файле.

С точки зрения любого компилятора с языка C++ файл является единицей компиляции – исходным модулем. В результате компиляции каждого исходного модуля (в случае отсутствия ошибок) получаются объектные модули. Каждому исходному модулю будет соответствовать свой объектный модуль. Такая структура программы называется многомодульной. Если при этом компилятор может работать с исходными модулями по отдельности, то говорят, что компилятор поддерживает принцип раздельной компиляции.

Назовем файл, содержащий описание функций cifr.cpp. Для добавления нового файла в проект используется команда File–New–Empty file(Ctrl–Shift–N). При этом появляется диалоговое окно, изображенное на рисунке 10, в котором спрашивается, хотите ли Вы включить этот новый фай в активный проект. Для

включения файла в активный проект необходимо сохранить файл в каталоге активного проекта.

Если ответить «Да», то откроется окно для сохранения файла с новым именем и после того, как вы укажите имя файла, он будет включен в активный проект.

Рисунок 10

Если ответить «Нет», то файл в активный проект добавлен не будет и ему будет дано имя Untitled. В дальнейшем файл можно будет сохранить с другим именем через команду File–Save as. Его также можно будет включить в проект, используя команду меню Project–Add files.

В нашем случае достаточно воспользоваться первым вариантом. Добавляем новый файл в активный проект, сохраняем его с именем cifr.cpp. В открывшееся окно редактора для нового файла переносим описание функций. В этом случае содержимое файла будет выглядеть так:

```
int count_dig(int a)
{
    int k=0;
    while (a!=0)
    {
        k++;        //операция увеличения
        a/=10;      //составная операция присваивания
    }
    return k;
}
```

```
int sum_dig(int a)
{
    int s=0;
    a=abs(a);
    while (a!=0)
    {
        s+=a%10;
        a/=10;
    }
    return s;
}
void del_last_dig(int& a)
{
    a/=10;
}

int del_first_dig(int& a)
{
    int k=count_dig(a);
    int r=1;
    for (int i=0; i<k-1; i++) r*=10;
    a%=r;
    return a;
}
```

Содержимое файла довольно необычно для C, поскольку не содержит ни одной команды подключения заголовочных файлов. Если попробовать откомпилировать такой файл отдельно Build–Compile current file (Ctrl–Shift–F9), то выдается сообщение об ошибке компиляции, показанное на рисунке 11.

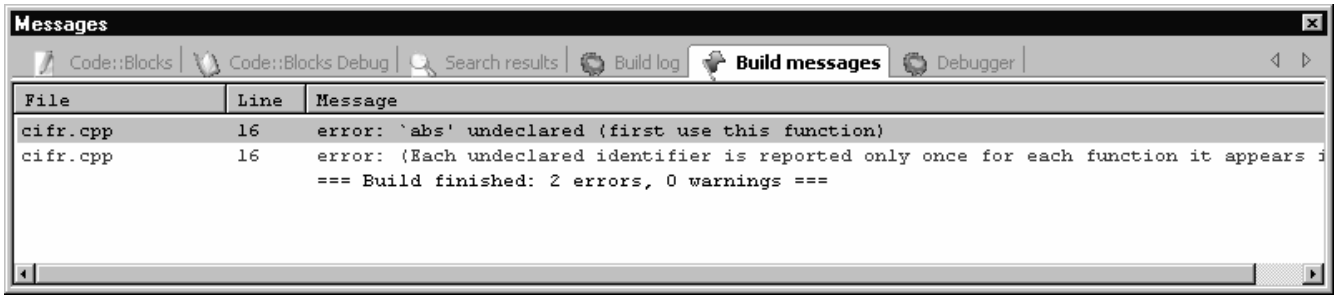

Рисунок 11

Ошибка означает, что имя функции `abs()` неизвестно, необходимо подключить заголовочный файл, в котором находится описание прототипа функции `abs()` для целочисленного аргумента. Чтобы исправить эту ошибку достаточно подключить заголовочный файл `<cstdlib>`:

```
#include <cstdlib>
```

Почему, когда программа была представлена одним файлом, не понадобилось подключать этот заголовочный файл? Это связано с тем, что многие заголовочные файлы C++ сами подключают другие заголовочные файлы. Такая ситуация имеет место и в случае использования заголовочного файла `<iostream>`.

Теперь, когда компиляция файла завершится успешно, можно увидеть, что в каталоге проекта \obj\Debug (или \obj\Release если используется соответствующая конфигурация) появился файл объектного модуля cifr.o.

Для того чтобы убедиться, что программа не потеряла работоспособность, необходимо перекомпилировать файл с функцией `main()`, выполнить редактирование связей и запустить вновь полученную программу. Все это можно сделать с помощью команды Build and run.

В полученном варианте программы присутствует еще один недостаток – явное перечисление прототипов всех функций, вынесенных нами в отдельный файл, перед описанием функции `main()`. Однако убрать их нельзя, т.к. в этом случае все имена функций станут неизвестными и при компиляции будет получено сообщение об ошибках, аналогичное предыдущему (рисунок 11).

Очевидно, что хорошим решением в этом случае было бы наличие заголовочного файла, содержащего прототипы всех функций, вынесенных в отдельный файл.

Правило хорошего стиля требует в случае вынесения функций в отдельный файл создавать соответствующий ему заголовочный файл с расширением h,

содержащий прототипы соответствующих функций. Причем рекомендуется, чтобы имена cpp-файла и соответствующего ему h-файла совпадали.

В нашем случае это будет заголовочный файл cifr.h с таким содержимым:

```
int count_dig(int a);
int sum_dig(int a);
void del_last_dig(int& a);
int del_first_dig(int& a);
```

Создание и включение заголовочного файла в проект аналогично созданию файла cifr.cpp. Отличие будет только в том, что для заголовочных файлов в окне менеджера проектов создается специальная папка.

Подключение заголовочного файла, используемого в проекте, выглядит так:

```
#include "cifr.h"
```

Задания к модулю

1. Расширьте список функций, работающих с цифрами десятичного представления, добавив, например, функцию замены цифры в числе, функцию перестановки цифр и другие. Внесите соответствующие изменения в файлы cifr.cpp и cifr.h. Создайте новый проект, в котором в функции main() будут использованы новые функции, включите в этот проект уже готовые файлы cifr.cpp и cifr.h.

2. Составьте набор функций (b_cifr.cpp) для работы с цифрами двоичного представления целого числа. Создайте проект, в котором используются функции для работы с цифрами десятичного (cifr.cpp) и двоичного (b_cifr.cpp) представления целого числа.

Для вывода двоичного представления целого неотрицательного числа можно использовать рекурсивную функцию, выводящую число без последней цифры, а затем – последнюю цифру:

```
void print_b(int a)
{
    if (a)
    {
        print_b(a/2);
        cout<<a%2;
    }
}
```

3. Написать набор функций (korni.cpp) для вычисления квадратного, кубического корня и арифметического корня степени k ($y = \sqrt[k]{x}$), используя итерационную формулу Ньютона:

$$\begin{cases} y_{n+1} = \frac{1}{k}\left[(k-1)y_n + \frac{x}{y_n^{k-1}}\right] \\ y_0 = x \end{cases}$$

Итерационный процесс вычисления заканчивается, когда два последовательных приближения будут удовлетворять условию $|y_{n+1} - y_n| < \varepsilon$.

Используйте написанные функции в проекте, который выдает таблицу значений арифметических корней для заданной последовательности целых положительных чисел.

4. Написать набор функций (elem.cpp) для вычисления элементарных функций *sin x*, *cos x*, e^x, используя формулы Маклорена. Составить проект, в котором выполняется табулирование указанных функций на произвольном отрезке с заданным шагом. Сравнить получаемые значения со значениями, вычисляемыми функциями из библиотеки <cmath>.

5. Написать набор функций (q_dig.cpp) для проверки, обладает ли десятичная запись целого положительного числа заданными свойствами. Например, в десятичной записи числа нет нулей, десятичная запись числа состоит точно из k цифр, в десятичной записи числа встречается заданная цифра, сумма цифр равна заданному значению и другие. Составить программу, в которой

вводимая последовательность целых положительных чисел проверяется на наличие/отсутствие указанных свойств.

6. Написать набор функций (compl.cpp) для выполнения операций с комплексными числами, заданными в тригонометрической форме. Составить программу, вычисляющую значение любого выражения, операндами которого являются комплексные числа.

Рубежный контроль к модулю

Самостоятельная работа.

В результате изучения модуля студент должен уметь:

1. Выделять при решении сложных задач логически связные функции и оформлять их в виде отдельных файлов – библиотек функций.
2. Писать заголовочные файлы для подключения своих функций.
3. Создавать проекты, содержащие уже существующие библиотеки функций и расширять их новыми.

МОДУЛЬ 3

Отладка и тестирование программ на С++

Задачи и цели: Ознакомление со средствами обнаружения логических ошибок в программах. Изучение возможностей отладчика, встроенного в визуальную среду программирования. В результате изучения блока студент должен уметь трассировать программы с целью обнаружения в них логических ошибок.

4 ИСПОЛЬЗОВАНИЕ ОТЛАДЧИКА

Если в программе нет синтаксических ошибок, то она будет выполнена. Означает ли это, что при выполнении программы будет получен правильный результат. Нет, потому что кроме синтаксических ошибок, в программе могут

быть семантические (логические) ошибки. Эти ошибки компилятор обнаружить не может. Семантические ошибки могут привести к следующим ситуациям:

- программа выполняется, но выдает неправильный результат;
- программа завершается сообщением об ошибке времени выполнения;
- программа не завершает выполнения (зацикливается);
- программа при некоторых значениях входных данных завершается и выдает правильный результат, а при некоторых входных значениях возникает одна из вышеперечисленных ошибочных ситуаций.

Если анализ текста программы не помогает обнаружить семантическую ошибку, приходится прослеживать по шагам выполнение программы, следя за изменениями значений используемых переменных.

Для этого можно добавлять в текст программы (или ее фрагмента, где вероятно наличие ошибки) вспомогательные операторы вывода.

Современные визуальные средства разработки программ включают в себя специальные программы – отладчики, облегчающие этот процесс. Отладчики могут включать в себя довольно много средств, упрощающих процесс поиска логических ошибок в программах. Мы рассмотрим только наиболее важные из них: управление пошаговым выполнением программы и отслеживание изменения значений переменных при выполнении отдельных операторов.

4.1 Подготовка программы к отладке

Для того чтобы воспользоваться возможностями отладчика (Debugger) необходимо, чтобы проект был откомпилирован в конфигурации Debug. Если до этого использовалась конфигурация Release, нужно изменить ее командой Build – Select target – Debug или используя переключатель конфигурации на панели (рисунок 6). Конфигурация Debug отличается тем, что компилятор вставляет в машинный код дополнительные команды, которые будут использованы при отладке. При этом получается объектный модуль существенно большего объема, чем в конфигурации Release.

После переключения конфигурации необходимо выполнить одну из команд Build (Ctrl–F9) или Rebuild (Ctrl–F11). Отличие команды Rebuild заключается в том, что перед ее выполнением будут удалены выполнимый и объектный модули, если они были уже созданы.

Выполнение построенного в конфигурации Debug модуля с помощью команды Run ничем не отличается от выполнения модуля, созданного в конфигурации Release. Чтобы запустить процесс отладки используется команда Debug–Start (F8) или Debug–Run to cursor(F4). Это можно сделать и с помощью панели кнопочного меню отладчика, показанной на рисунке 12.

Чтобы иметь возможность останавливать процесс выполнения программы в нужных местах в тексте программы отмечают точки останова (breakpoints). Точка останова отмечается слева от текста программы красным кружком (рисунок 13).

После запуска отладчика командой Debug–Start (F8) будут выполнены все операции до первой точки останова. Чтобы прервать выполнение в нужном месте программы можно также поместить в соответствующую строку курсор и запустить отладку командой Debug–Run to cursor(F4).

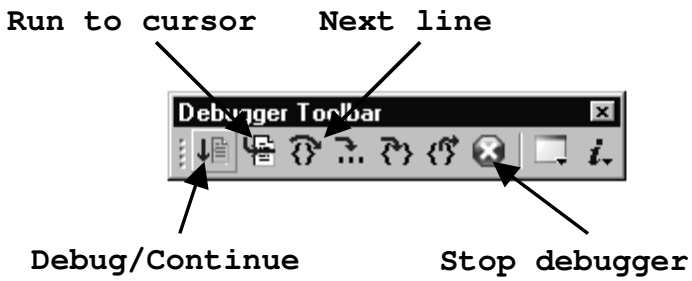

Рисунок 12

Управлять дальнейшим выполнением программы можно через команды меню или кнопки на панели меню отладчика (рисунок 12).

Продолжить выполнение до следующей точки останова Debug–Continue(Ctrl–F7). Выполнить до строки, отмеченной курсором Debug–Run to cursor(F4). Перейти при выполнении к следующей строке в тексте программы

Debug–Next line (F7). Если следующая выполнимая команда содержит вызов функции, то выполнить функцию по шагам Debug–Step into (Shift–F7). Завершить пошаговое выполнение функции и выйти из нее Debug–Step out (Ctrl–Shift–F7). Завершить отладку Debug–Stop debugger.

В процессе пошагового выполнения программы строка, перед выполнением которой произошел останов, отмечается слева от текста программы желтым треугольником (рисунок 13).

Следует обратить внимание, что после завершения выполнения программы под управлением отладчика окно вывода программы автоматически закрывается.

Чтобы проследить за изменением значений переменных при выполнении программы необходимо открыть окно Watches, представленное на рисунке 14.

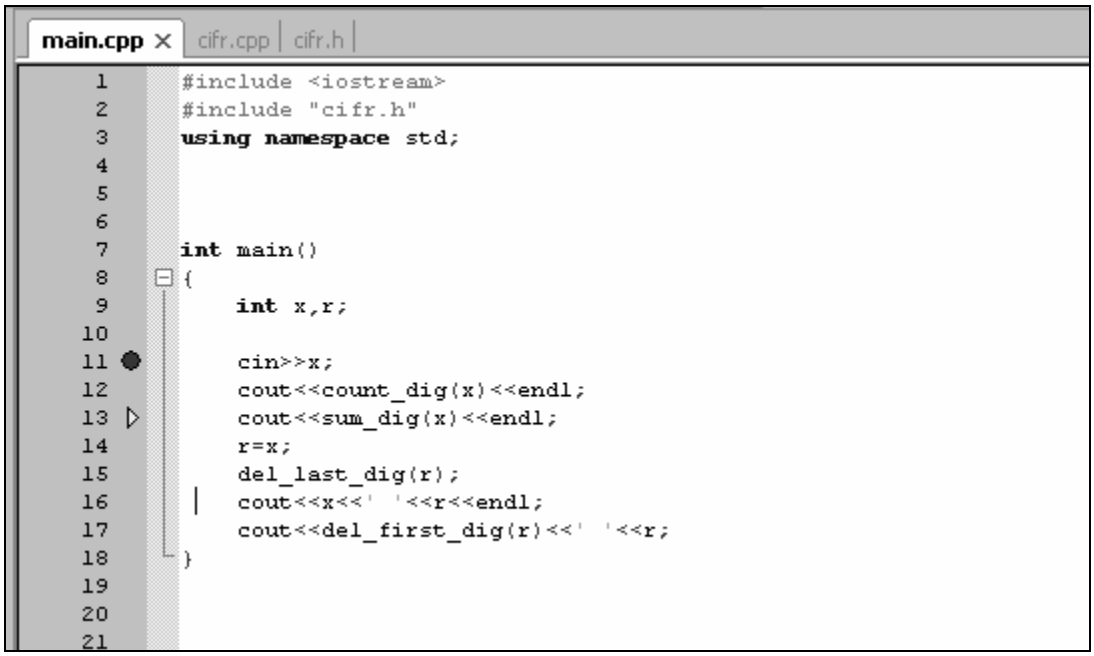

Рисунок 13

Окно Watches в каждый момент пошагового выполнения содержит значения локальных переменных и аргументов функции, которая выполняется в данный момент. В окно Watches могут быть добавлены дополнительные контролируемые значения, например, нелокальные переменные или выражения, значение которых может быть вычислено

Рисунок 14

4.2. Поиск ошибки в программе с помощью отладчика

Пример 2.

Рассмотрим следующую функцию, вычисляющую конечную сумму вида $\sum_{i=1}^{n}\frac{1}{i}$.

```
double sum(int n)
{
    double s=0;
    for (int i=1; i<n+1; i++)
        s+= 1/i;
    return s;
}
```

Создайте проект, в котором функция расположена в файле с именем summa.cpp, заголовочный файл summa.h содержит прототип

```
double sum(int n);
```

Файл main.cpp содержит следующую программу:

```
#include <iostream>
#include "summa.h"
using namespace std;

int main()
{
    int n;
     cin>>n;
```

```
    cout.precision(20);
    cout<<sum(n)<<endl;
    return 0;
}
```

Для объекта `cout` в данном примере продемонстрировано использование одного из методов, с помощью которых можно управлять форматом вывода на экран. Метод `precision(n)` изменяет формат вывода для вещественных данных, устанавливая количество цифр в дробной части числа равным `n`.

Откомпилируйте проект и выполните программу. Выполнив несколько запусков с разными значениями `n`, можно убедиться, что результат не зависит от `n` и всегда равен `1`.

Чтобы обнаружить ошибку, воспользуемся отладчиком. Очевидно, что ошибка содержится в функции `sum()`, поэтому точку останова можно поставить на той строке, в которой идет вызов функции `sum()` или на первом операторе в теле функции `sum()`.

Запускаем отладчик и выполняем по шагам тело функции `sum()`. При этом анализ информации в окне Watches показывает, что в процессе выполнения цикла не изменяется значение переменной `s`. Оно остается равным `1`, т.е. величине первого слагаемого.

Чтобы понять, почему это происходит, добавим в окно отладчика выражение, которое равно слагаемому, добавляемому к сумме: `1/i`. Используем для этого команду Debug–Edit watches. Выполнив еще несколько шагов, можно увидеть, что слагаемое равно `0` (рисунок 15).

В данном примере ошибка связана с особенностью операции /, которая в языке C++ является перегруженной, т.е. для целочисленных операндов – это деление нацело, а если хотя бы один из операндов вещественный, то и результат будет вещественным.

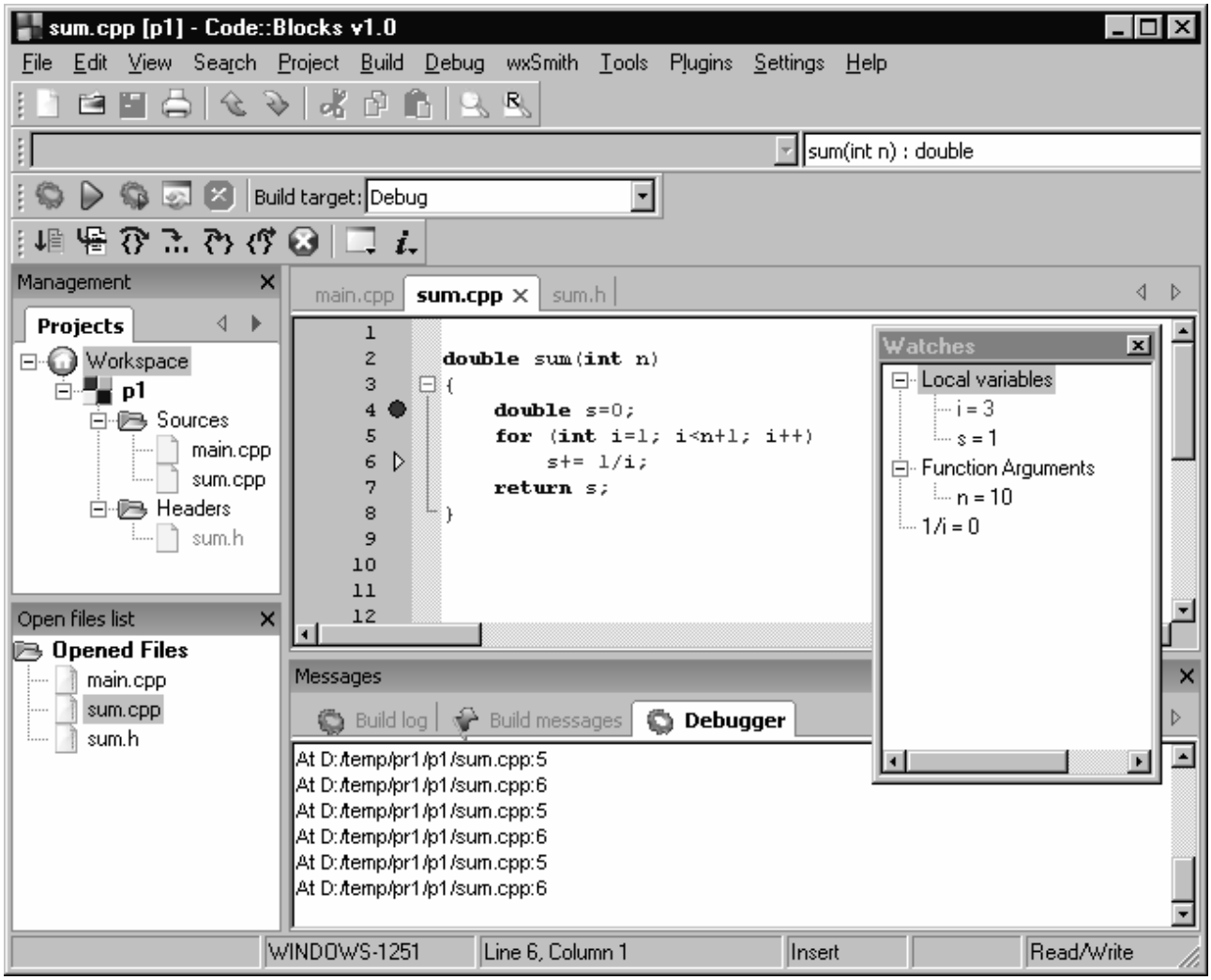

Рисунок 15

Чтобы исправить ошибку, достаточно записать числитель дроби как вещественную константу:

```
s+=1.0/i;
```

Можно поступить иначе – оператор цикла `for` не требует, чтобы переменная `i` была обязательно целого типа, т.к. этот оператор определен в C++ не так, как цикл `for` в Паскале.

```
for (double i=1; i<n+1; i++)
```

И в одном, и в другом случае после исправления мы получим правильный результат работы программы.

Тесты рубежного контроля

1. Для обнаружения логической ошибки в программе на C++ необходимо:
    a. Вставить в нее дополнительные операторы выдачи информации.
    b. Проверить алгоритм решения задачи.
    c. Написать программу заново, используя другой алгоритм.
    d. Выполнить трассировку программы средствами отладчика.
    e. Выполнить трассировку программы вручную.
2. Средства отладчика могут быть использованы:
    a. Если проект откомпилирован в конфигурации Release.
    b. Если проект откомпилирован в конфигурации Debug.
    c. При любой конфигурации компилятора.
3. Какая информация может быть отслежена через окноWatches
    a. Содержимое всех переменных программы.
    b. Содержимое локальных переменных функции.
    c. Содержимое локальных переменных функции и дополнительных переменных, время жизни которых не истекло.
    d. Значение выражений, которые могут быть вычислены в данной точке программы.
4. Для чего необходимы точки останова
    a. Чтобы прекратить выполнение программы.
    b. Чтобы управлять трассировкой программы.
    c. Чтобы выполнять программу по шагам.

В результате изучения модуля студент должен уметь:

1. Компилировать проекты с целью их дальнейшей отладки средствами отладчика.
2. Трассировать программы с целью обнаружения в них логических ошибок.

3. Управлять ходом выполнения программы с помощью точек останова и средств отладчика.
4. Правильно организовывать выдачу информации в окноWatches.
5. Использовать отладчик для изучения алгоритмов решения сложных задач.

5 ВОПРОСЫ ДЛЯ ПОВТОРЕНИЯ И САМОКОНТРОЛЯ

1. Какие этапы проходит при выполнении программа на языке C++ .
2. Перечислить модули, которые получаются в процессе трансляции программы на языке C++.
3. Что представляет собой проект в среде разработки Code::Blocks.
4. Как сохраняются файлы, относящиеся к одному проекту.
5. Чем отличаются конфигурации Debug и Release.
6. Для чего нужно описание прототипов функций. Куда рекомендуется помещать прототипы функций.
7. Какова роль функции `main()` в программе на C++.
8. В чем заключается принцип раздельной компиляции.
9. Если несколько функций записываются в отдельный исходный файл, то, что нужно сделать, чтобы использовать эти функции в другом файле.
10. Что делает следующая директива препроцессора:

```
#include <iostream>
```

11. Что выполняет следующий оператор:

```
using namespace std;
```

12. В каком случае при определении функции можно не использовать оператор `return`.
13. Каким образом можно создать проект, состоящий из нескольких файлов.
14. Для чего используются объекты `cin` и `cout`.

15. Напишите прототип функции с именем `mpg`, которая принимает два аргумента типа `int` и возвращает результат типа `double`.

16. Для чего предназначено окно Watches.

17. Что нужно сделать, чтобы в процессе отладки выполнение программы остановилось на конкретном операторе.

18. Для чего нужны операторы объявления.

19. Чем отличаются функции с возвращаемым значением и без возвращаемого значения.

20. Какие существуют способы включения прототипа функции в код программы.

21. Как при описании параметров функции указать, что передача параметра должна осуществляться по ссылке.

22. В чем особенность операции деления для целых чисел в C++.

ЛИТЕРАТУРА

1. Брюс Эккель. Философия C++. Введение в стандартный C++. –
2. Х.М.Дейтел, П.Дж.Дейтел. Как программировать на C++. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.