ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

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

РУСАНОВА Я. М.

РАЗРАБОТКА ПРОГРАММ НА ЯЗЫКЕ C++ В СРЕДЕ VISUAL STUDIO

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

для студентов 1–2 курса дневного и вечернего отделений
специальностей «информационные технологии» и «прикладная математика»
факультета математики, механики и компьютерных наук

Ростов-на-Дону
2008

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

В методических указаниях представлены сведения об использовании визуальной среды программирования Visual Studio при разработке программ на языке 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. Модули сопровождаются тестами рубежного контроля, 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. Методические указания разработаны в поддержку курса «Языки программирования и методы трансляции».

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

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.

Microsoft Visual C++ является одним из компонентов Microsoft Visual Studio. Microsoft Visual Studio носит название *интегрированной среды разработки* (Integrated Development Environment – IDE).

После запуска Microsoft Visual Studio, например Microsoft Visual Studio 2005, открывается главное окно (Рисунок 1).

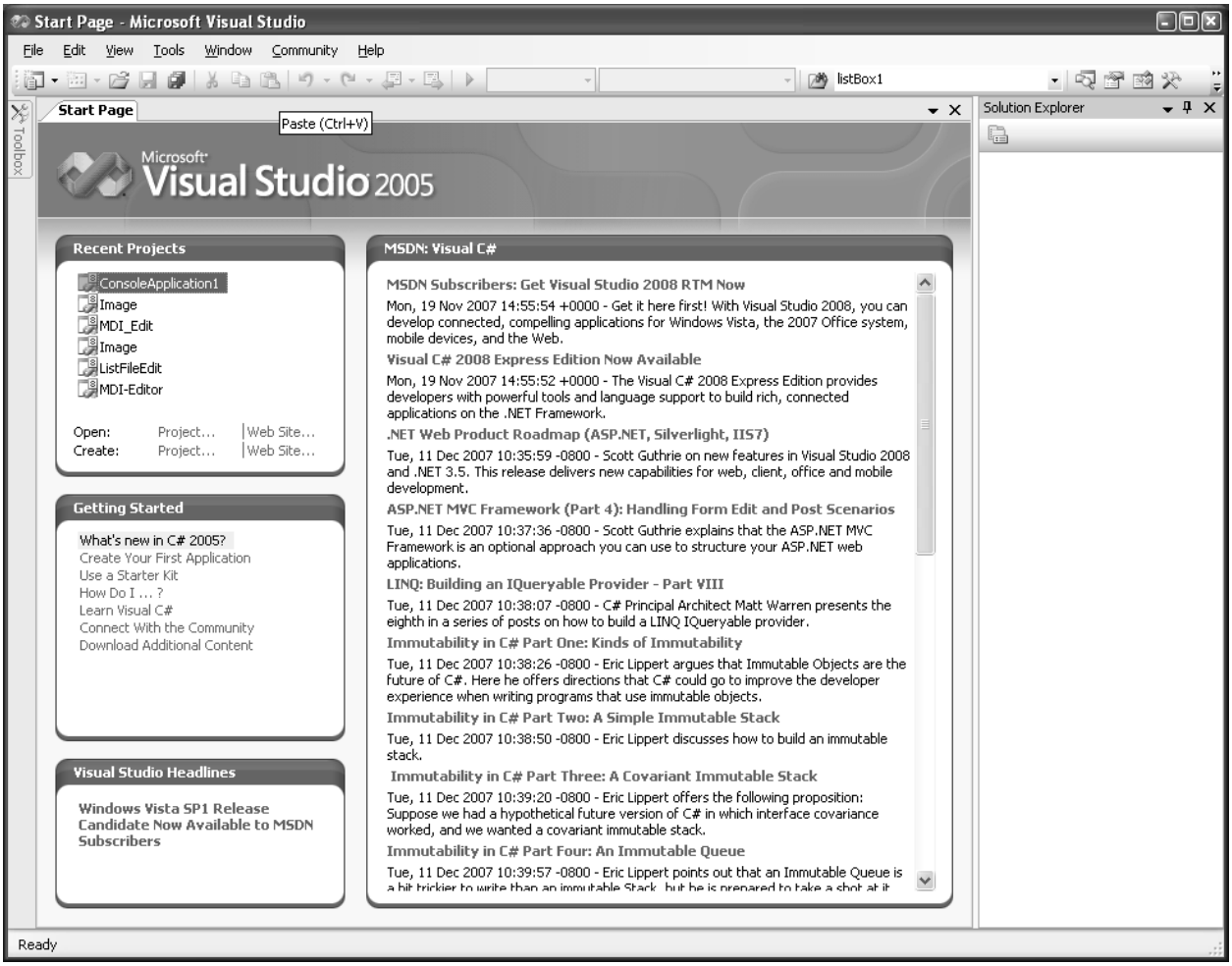

Рисунок 1

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

МОДУЛЬ 1. ПРОЕКТЫ, СОДЕРЖАЩИЕ ОДИН МОДУЛЬ

Цель. Познакомить с языком программирования C++ и со средой разработки Visual Studio. Научить разрабатывать проекты простой структуры на языке C++.

Требуемый начальный уровень подготовки. Владение навыками программирования средствами языка Паскаль. Знакомство с модульной структурой.

1.1 Создание нового проекта

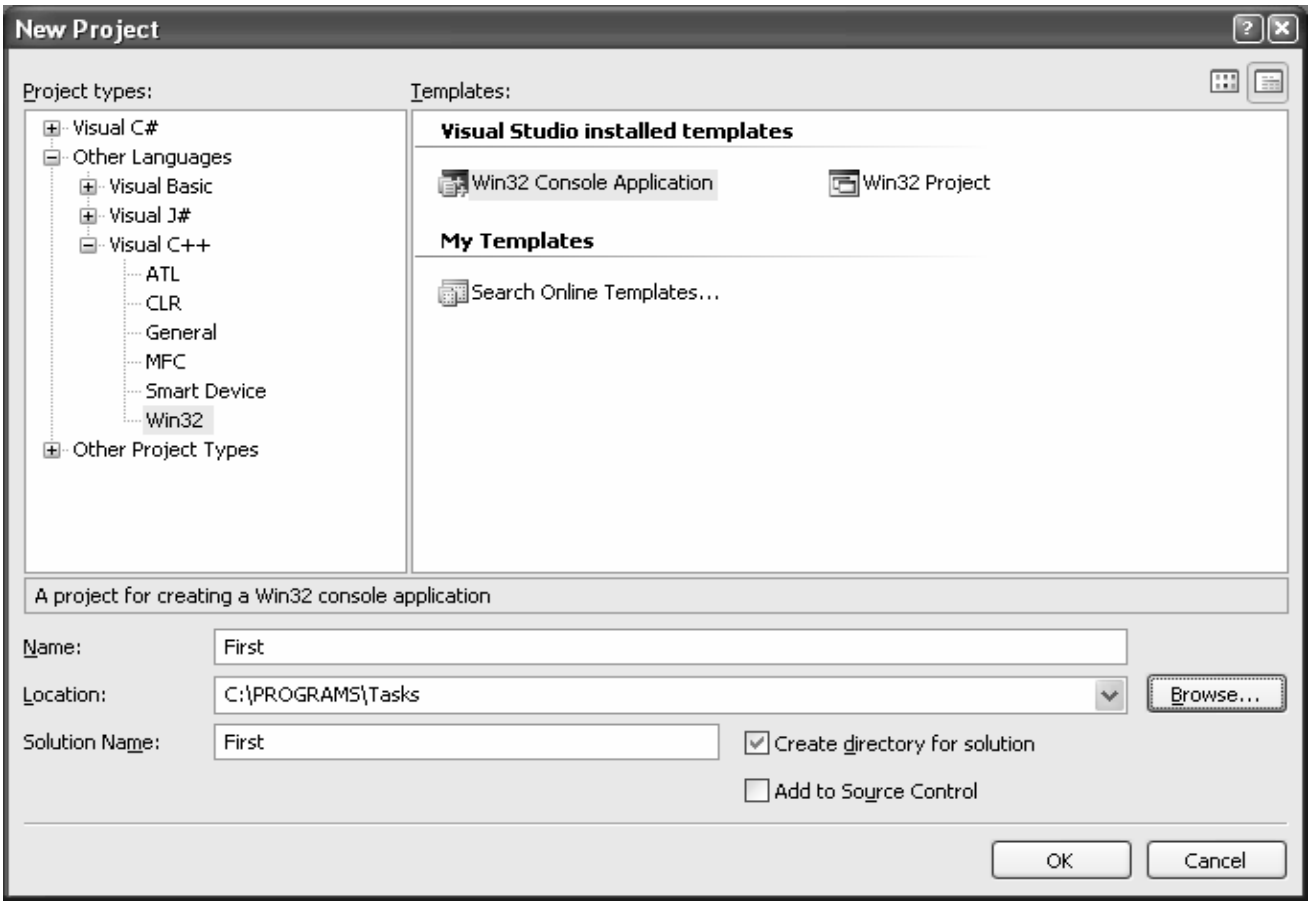

Рисунок 2

Для создания нового проекта необходимо выбрать один из вариантов:

— в строке меню выбрать File→New→Project.

— на панели пиктограмм выбрать , а затем выбрать New→Project.

— нажать горячие клавиши Ctrl+Shft+N.

— выбрать Create Project из части 

в зоне Recent Projects.

В результате откроется окно New Project (Рисунок 2).

В этом окне необходимо выполнить ряд действий.

1. На панели Project types необходимо выбрать тип проекта: Visual C++.
2. Приложение будет рассматриваться как приложение под Windows. Поэтому необходимо уточнить тип проекта как Win32.
3. Решение задач будет рассматриваться для консольных приложений. Поэтому на панели Templates выбрать Win32 Console Application.
4. В поле ввода Location необходимо выбрать каталог – место расположения проекта.
5. В поле Name – задать имя проекта.

Рисунок 3

В нашем примере Name – First, Location – C:\PROGRAMS\Tasks.

После нажатия на кнопку OK появится окно с описанием выбранного типа создаваемого проекта (Рисунок 3).

Если что-либо требуется изменить, то следует нажать Cancel. Если все нормально, то следует выбрать Next для дальнейших уточнений настроек проекта или Finish в случае согласия с настройками по умолчанию.

Для рассматриваемых учебных программ можно соглашаться с настройками по умолчанию и сразу выбирать Finish.

После нажатия Finish открывается окно нового только что созданного проекта (Рисунок 4).

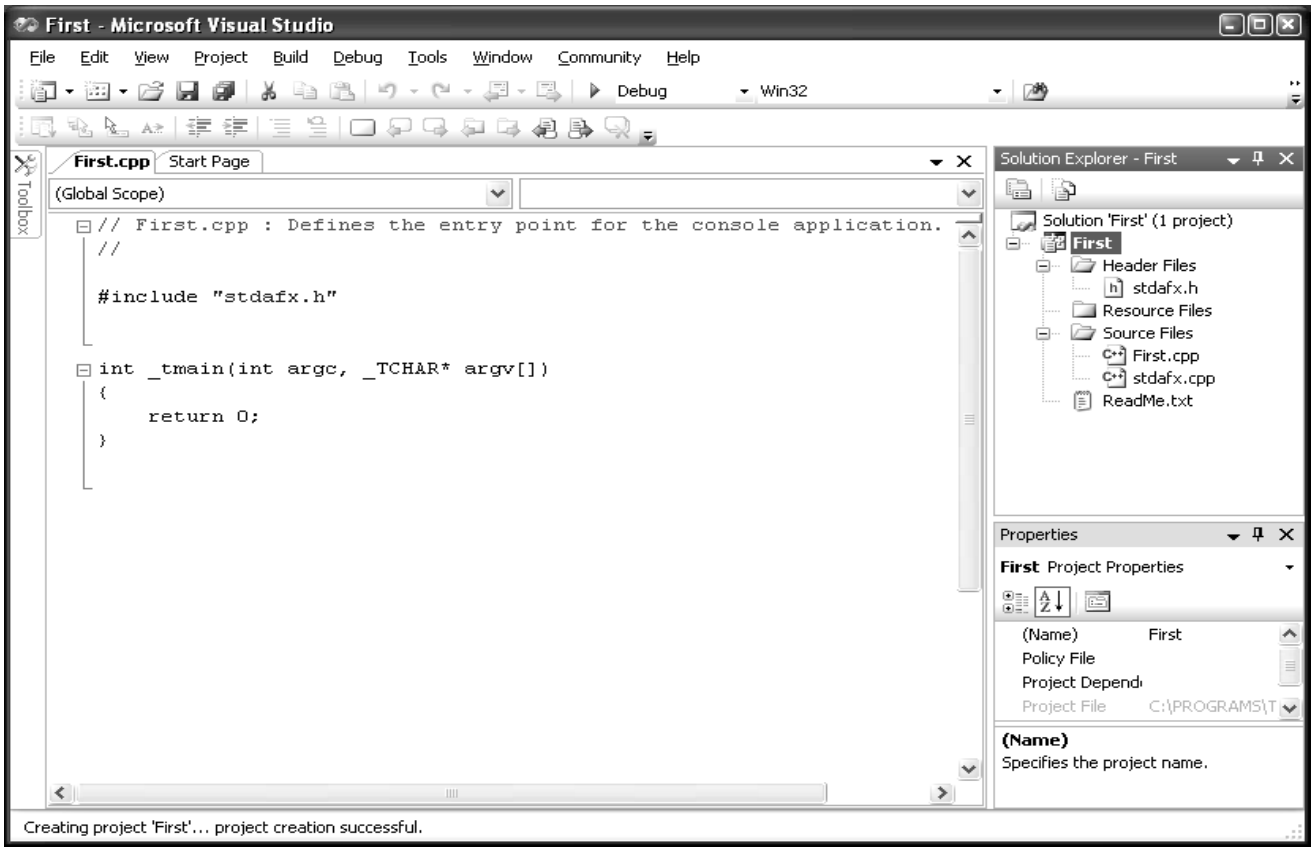

Рисунок 4

1.1 Зоны окна для работы с проектом

Окно Visual Studio разделено на зоны. Большую часть окна занимает основная рабочая область, в которой можно редактировать файлы.

В конкретном примере в ней находится файл First.cpp. Это файл, в котором будет находиться текст создаваемой программы на языке C++.

В зоне Solution Explorer – First видна структура проекта First, т.е. видны все файлы, входящие в состав данного проекта. Зона Solution Explorer определяет, с каким файлом проекта выполняется работа в текущий момент, и соответственно организует вывод информации в основную рабочую область.

Зона Solution Explorer отражает логическую структуру проекта. В ней представлены несколько групп файлов: cpp-файлы, .h-файлы и файлы ресурсов (в рамках создания консольных приложений они нами рассматриваться не будут). Эти файлы объединены в логические разделы: Header Files (*.h), Resource Files и Source Files (*.cpp).

Договоримся эти логические разделы называть папками, а физические каталоги на диске так и будем называть каталогами.

После создания проекта для консольного приложения в каталоге C:\PROGRAMS\Tasks создан каталог First, в котором созданы два файла First.ncb и First.sln – файлы решения – и вложенный каталог First. Итак, искомый файл First.cpp расположен в следующем каталоге: C:\PROGRAMS\Tasks\First\First.

Кроме этого в каталоге C:\PROGRAMS\Tasks\First\First автоматически созданы следующие файлы:

1. First.vcproj – главный файл проекта. Он содержит информацию о версии Visual C++, о платформе, конфигурации и настройках проекта.
2. Уже упомянутый файл First.cpp – главный файл с исходным кодом на языке C++.

3. Файлы StdAfx.h, StdAfx.cpp. Они используются средой для компиляции проекта.
4. Остальные файлы являются служебными, и мы их рассматривать не будем.

На данный момент нас интересует только файл First.cpp.

Он находится в папке Source Files зоны Solution Explorer. Если проект будет состоять из нескольких файлов *.cpp, то все они будут находиться в данной папке. Такой вариант создания проекта будет рассмотрен позже.

Рассмотрим содержимое файла First.cpp подробнее.

```
// First.cpp:
// Defines the entry point for the console application.

#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
     return 0;
}
```

Этот текст является сгенерированным системой автоматически.

`int _tmain(int argc, _TCHAR* argv[])` – это точка входа в программу. Функция `int _tmain(int argc, _TCHAR* argv[])` является аналогом функции `int main()`.

Программа 1. Рассмотрим процесс компиляции, запуска и отладки программы в описываемой среде на примере задачи поиска максимума из `n` целых чисел.

Для решения пока не будут задействованы никакие другие функции, кроме `main()`. Текст программы необходимо набрать внутри тела функции `int _tmain(int argc, _TCHAR* argv[])`.

1.2 Работа с редактором.

Интерфес редактора является максимально дружественным. Например, в нем реализованы следующие моменты:

— выделение цветом,

— подсказка синтаксиса операторов,

— автоматические отступы при переходе к новой строке,

— контекстная помощь по параметрам функций и т.д.

Зарезервированные слова выделяются цветом. Если этого не происходит, проверьте написание слов, возможно, они набраны с ошибкой.

Напомним, что текст программы должен быть хорошо структурирован.

При наборе текста программы можно не отслеживать отступы. Процесс расстановки необходимых отступов выполняется автоматически при переходе к следующей строке нажатием клавиши Enter. Не стоит самостоятельно добавлять или удалять отступы.

Листинг программы `First.cpp`.

Рассмотрим листинг набранной программы. Например, он будет выглядеть следующим образом.

```
// First.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>

int _tmain(int argc, _TCHAR* argv[])
{
     int n, i, x, max;
     cout<<"Input n"<<endl;
     cin>>n;
     cout<<"Input "<<n<<" elements"<<endl;
```

```
    cin>>max;
    for (i=1;i<n;i++)
    {
        cin>>x;
        if (x>max)
            max=x;
    }
    cout<<"Maximum = "<<max<<endl;
    return 0;
}
```

Рассмотрим вначале фрагмент программы:

```
for (i=1;i<n;i++)
{
    cin>>x;
    if (x>max)
        max=x;
}
```

В нем задействованы управляющие конструкции `if` и `for`.

Оператор `if-else` существует в двух вариантах: с секцией `else` и без нее.

Первый вариант:

```
if (выражение)
  оператор
```

Второй вариант:

```
if (выражение)
  оператор
else
  оператор
```

В программе представлен первый вариант:

```
    if (x>max)
        max=x;
```

В данном примере под выражением понимается условное выражение или просто условие. Результат выражение равен «истина» или «ложь».

В отличие от языка Паскаль «условие» всегда заключается в скобки.

Под оператором понимается либо одиночный оператор, либо блок операторов в фигурных скобках (аналог составного оператора в языке Паскаль).

В отличие от языка Паскаль, любой оператор, кроме блока, завершается символом точки с запятой (;). Поэтому перед `else` может стоять символ точки с запятой (;).

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.

В действительности возможности цикла `for` в C++ гораздо шире.

1.3 Запуск программы на выполнение

Теперь проект можно запустить на компиляцию и затем на выполнение или сразу на выполнение.

Запуск сразу на выполнение подразумевает автоматичеки следующие действия:

1. если исполняемый файл с учетом всех последних изменений уже существует, то происходит просто запуск этого исполняемого файла.
2. если в наличии имеются все откомпилированные модули с учетом всех последних изменений, то выполняется компоновка и создается исполняемый файл. Затем происходит запуск исполняемого файла.
3. если некоторые модули являются не откомпилированными после последних изменений, произведенных в их тексте, то вначале выполняется компиляция таких модулей. Затем в случае успешной компиляции всех модулей, происходит компоновка и затем запуск исполняемого файла.

Запуск на выполнение возможен несколькими способами:

— пиктограмма  , расположенная на панели пиктограмм  . Этот вариант равносилен варианту Debug → Start Debugging.

— Debug → Start Debugging из строки меню (или горячая клавиша F5).

— Debug → Start Without Debugging из строки меню (или горячие клавиши Ctrl+F5).

Если проект сразу запустить на выполнение, предварительно его не откомпилировав, то появится следующее диалоговое окно (Рисунок 5).

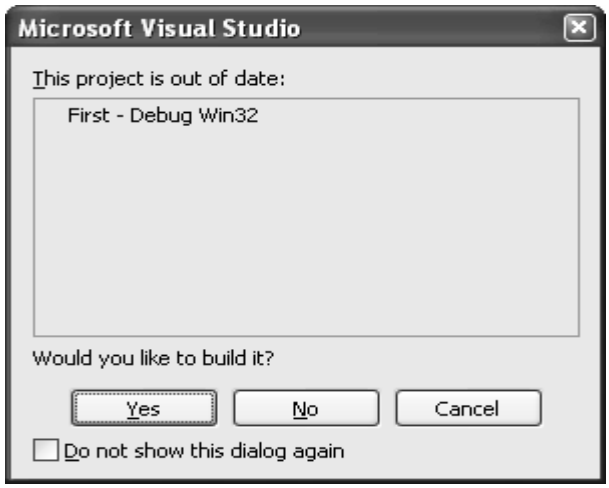

Рисунок 5

Следует выбрать вариант Yes. Тогда начнется процесс компиляции.

Можно предусмотрительно перед запуском программы выбрать запуск процесса компиляции.

1.4 Компиляция программы.

Запуск процесса компиляции возможен несколькими способами:

— Build → Compile из строки меню (или горячие клавиши Ctrl+F7). Это компиляция только одного текущего модуля (т.е. расположенного в активном окне редактора).

Остальные варианты подразумевают запуск компоновки проекта и компиляцию еще не откомпилированных модулей.

— Build → Build First из строки меню (или горячие клавиши Shft+F6).

— Build → ReBuild First из строки меню.

— Build → Build Solution из строки меню (или горячая клавиша F6).

— Build → ReBuild Solution из строки меню.

Различия между ними будут рассмотрены позже.

1.5 Отладка программы. Анализ и исправление ошибок

После завершения процесса компиляции в случае наличия ошибок появится диалоговое окно (Рисунок 6).

Рисунок 6

Следует выбрать No и посмотреть список ошибок. Список ошибок представлен в окне Error List (Рисунок 7).

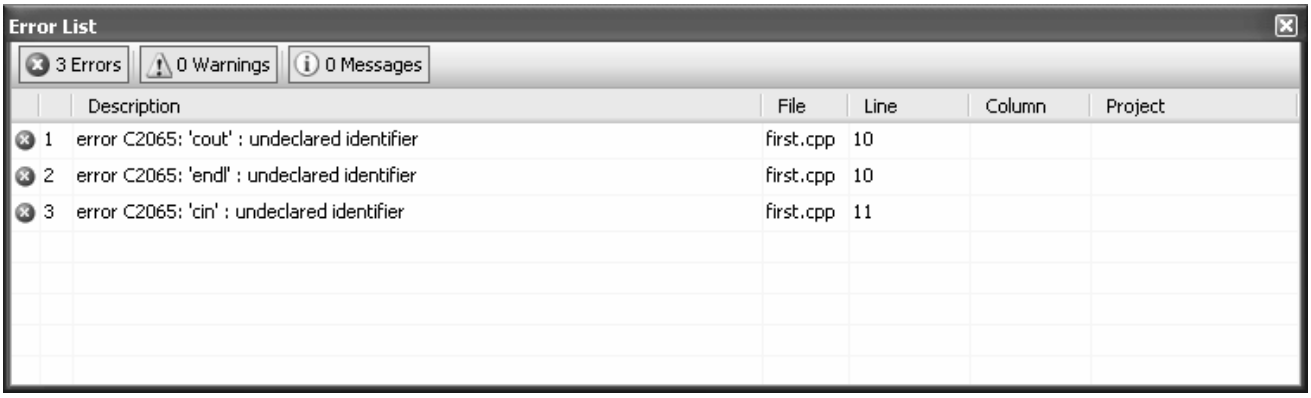

Рисунок 7

Проанализировав сообщения об ошибках, можно сделать вывод, что пропущено поключение пространства имен `std`.

Листинг программы `First.cpp` после исправления ошибки.

```
// First.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
     int n, i, x, max;
     cout<<"Input n"<<endl;
     cin>>n;
     cout<<"Input "<<n<<" elements"<<endl;
     cin>>max;
     for (i=1;i<n;i++)
     {
          cin>>x;
          if (x>max)
               max=x;
     }
     cout<<"Maximum = "<<max<<endl;
     return 0;
}
```

После исправления ошибок необходимо повторно запустить процесс компиляции.

В нашем примере компиляция проходит успешно. Окно ошибок Error List оказывается пустым (Рисунок 8).

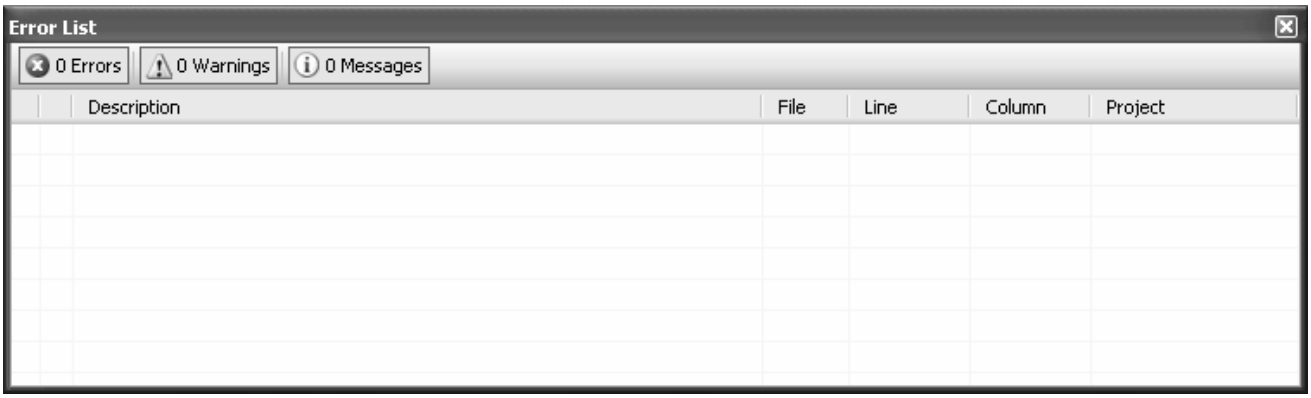

Рисунок 8

1.6 Просмотр результатов выполнения программы

После запуска программы на выполнение и ввода данных результат появляется в окне вывода результатов консольного приложения.

Если запуск был произведен через Debug → Start Without Debugging меню (горячая клавиша Ctrl+F5), то по окончании работы программы окно не закрывается, а происходит пауза, появляется фраза Для продолжения нажмите любую клавишу… и программа соотвественно ждет нажатия.

Если запуск был произведен через Debug → Start Debugging (горячая клавиша F5)или  , что является равносильным действием, то окно вывода результатов закрывается сразу при выполнении оператора `return`.

Чтобы этого не происходило, необходимо перед строкой с оператором

```
return 0;
```

добавить строку с оператором

```
system("PAUSE");
```

Тогда по окончании работы программы окно не закрывается, а происходит пауза, появляется фраза Для продолжения нажмите любую клавишу… и программа соотвественно ждет нажатия.

Результат работы программы (консольного приложения) выглядит следующим образом (Рисунок 9).

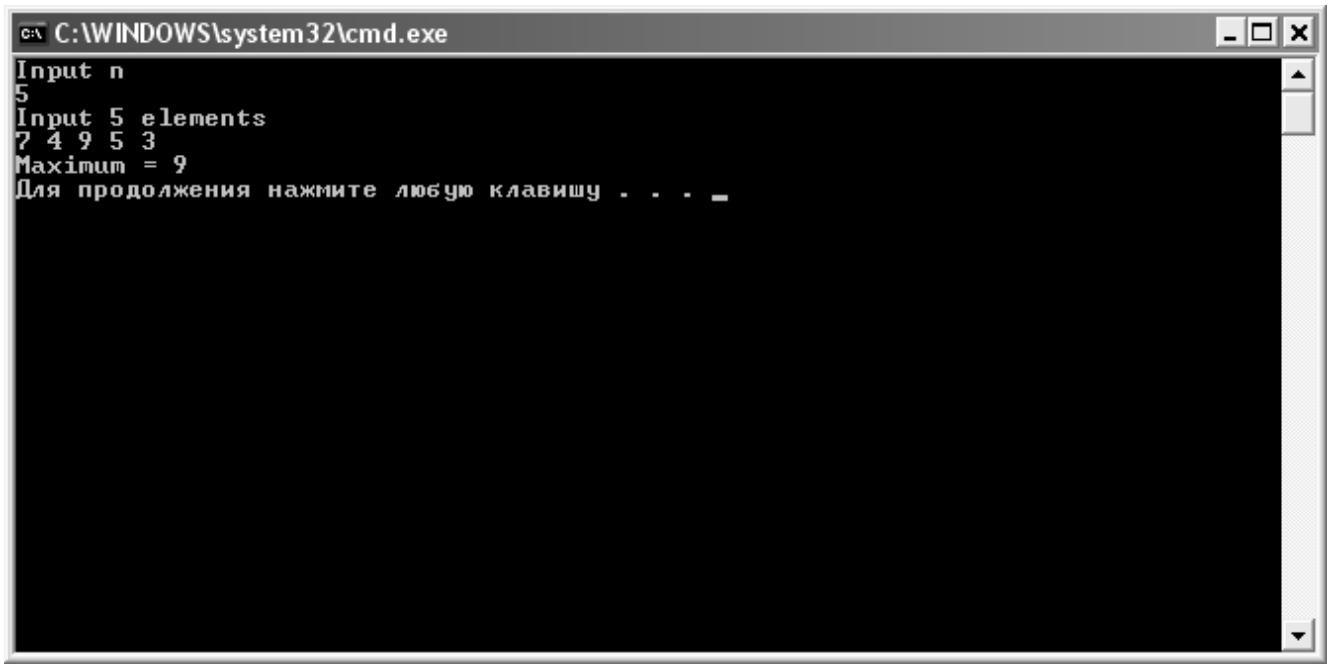

Рисунок 9

1.7 Сохранение проекта

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

— Для сохранения всего проекта, т.е. для сохранения всех изменений во всех файлах проекта можно выполнить команду File → Save All из строки меню (или горячие клавиши Ctrl+Shft+S или пиктограмма  на панели пиктограмм).

— Для сохранения текущего файла, открытого в окне редактирования, с сохранением имени файла можно выполнить команду File → Save имя файла из строки меню (или горячие клавиши Ctrl+S или пиктограмма на панели пиктограмм). Для первого рассмотренного проекта команда выглядит так: File → Save First.cpp.

— Для сохранения текущего файла, открытого в окне редактирования, с возможностью изменения имени файла можно выполнить команду File → Save имя файла as из строки меню. Для первого рассмотренного проекта команда выглядит так: File → Save First.cpp as.

1.8 Открытие существующего проекта

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

Открыть существующий проект можно несколькими способами:

— Выбрать из списка недавно открывавшихся проектов на вкладке Recent Projects (Рисунок 10) в окне, открывающемся при запуске Microsoft Visual Studio (Рисунок 11).

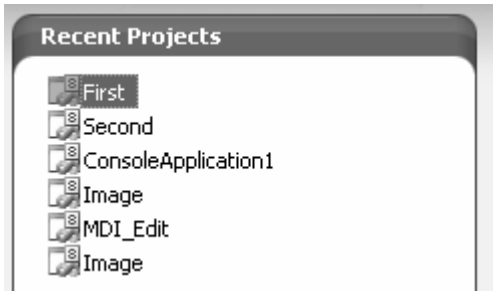

Рисунок 10

— Выбрать Open Project из части 

в зоне Recent Projects.

— Выбрать File → Recent Projects из строки меню.

— Выбрать Open → Project/Solution из строки меню (или горячие клавиши Ctrl+Shft+O).

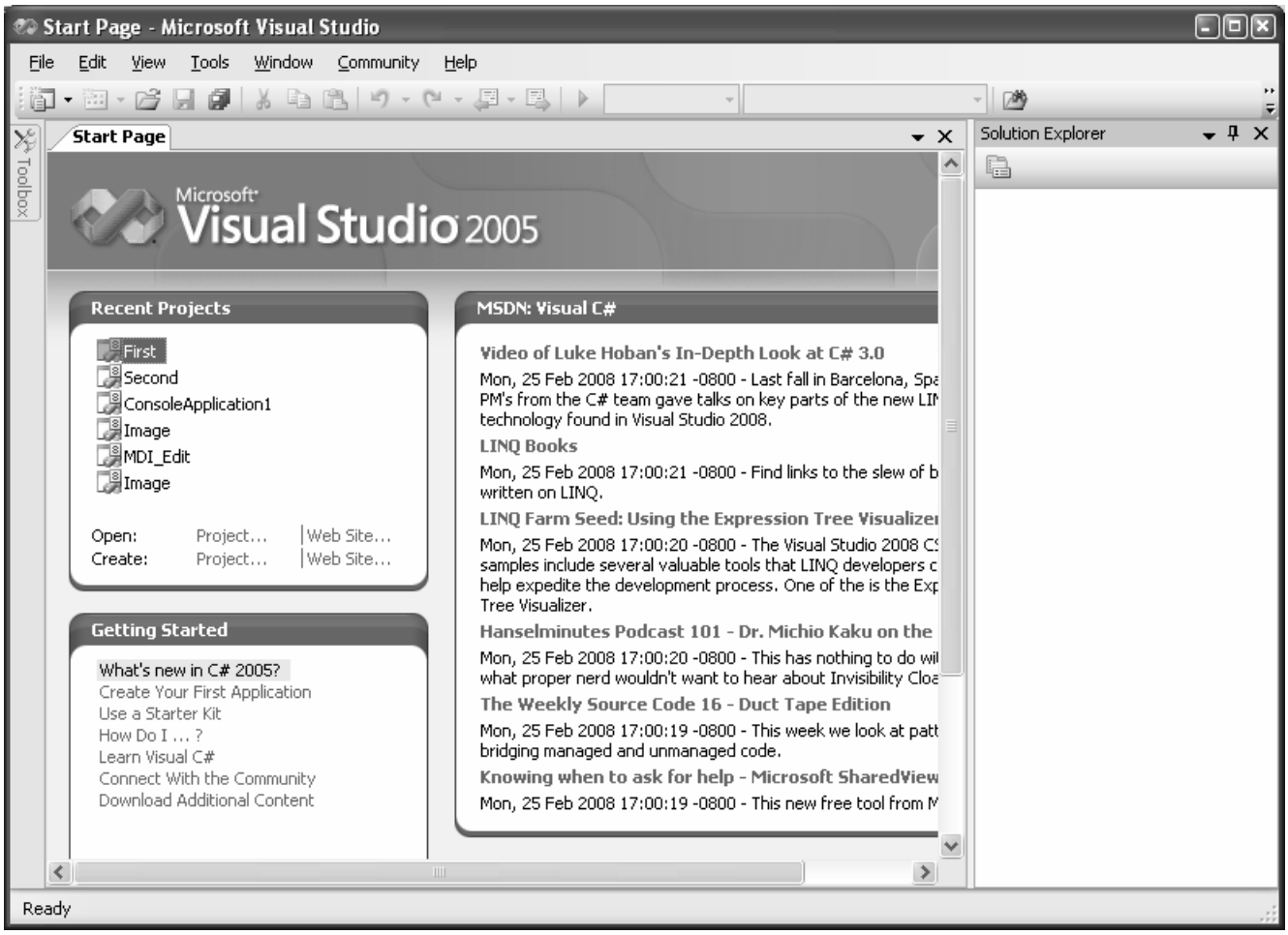

Рисунок 11

— Выбрать пиктограмму  на панели инструментов.

Первые три варианта подходят только для тех нескольких проектов, которые открывались последними. Их количество ограничено. В остальных случаях необходимо использовать последние два варианта.

1.9 Вопросы для самопроверки

1. Какие этапы проходит программа на языке C++ ?
2. Перечислить модули, которые получаются в процессе трансляции программы на языке C++.
3. Для чего нужно описание прототипов функций? Куда рекомендуется помещать прототипы функций?

1.10 Тесты рубежного контроля

Примечание: возможно несколько вариантов правильного ответа

№	Вопрос	Варианты ответа		
		1	2	3
1	Где может размещаться прототип функции?	Внутри функции main	Внутри любой функции	За пределами функции
2	Какой тип данных может иметь параметр цикла for	Целый	Вещественный	Любой
3	Для чего служит оператор return	Чтобы вернуть значение и выйти из функции	Чтобы вернуть значение	Чтобы выйти из функции
4	Всегда ли символ «;» завершает оператор	Да	Нет	Зависит от конкретной ситуации

1.11 Проектные задания для самостоятельного выполнения

Найти ошибки в каждом из следующих фрагментов программы и объяснить, как можно исправить ошибки. Для проверки внесенных исправлений необходимо написать программу, вставить в нее фрагменты и откомпилировать.

1.
```
int g()
{
     cout << "внутри функции g" << endl;
     int h()
     {
          cout << "внутри функции f" << endl;
```

```
        }
    }
```

2.
```
    int sum(int x, y)
    {
        int result;
        result=x+y;
    }
```

3.
```
    int sum(int n)
    {
        if (n==0)
            return 0;
        else
            n=sum(n-1);
    }
```

4.
```
    void f(float a);
    {
        float a;
        cout << a << endl;
    }
```

5.
```
    void product() {
        int a,b,c,result;
        cout << "Input a,b,c: ";
        cin >> a >> b >> c >>;
        result=a*b*c;
        cout >> "Result = " >> Result;
        return result;
    }
```

6.
```
    double max(double a,b)   {
        if a>b
            return a
        else
            return b;
    }
```

МОДУЛЬ 2. ПРОЕКТ, СОСТОЯЩИЙ ИЗ НЕСКОЛЬКИХ ФАЙЛОВ

Цель. Научить разрабатывать проекты, состоящие из нескольких файлов, на языке C++.

Требуемый начальный уровень подготовки. Владение навыками программирования. Знакомство с языком программирования C++. Умение разрабатывать простые проекты на языке C++. Знакомство с модульной структурой проектов.

Рассмотрим пример программы, состоящий из нескольких модулей.

Программа 2. Необходимо вычислить биномиальный коэффициент для заданных натуральных чисел `n` и `m`, используя формулу $C_n^m = \frac{n!}{m!(n-m)!}$.

Организуем вычисление факториала в виде функции, поместим ее в отдельном cpp-файле и создадим для этого cpp-файла заголовочный h-файл.

Создадим новый проект для консольного приложения и назовем его Second. Поместим функцию вычисления факториала в отдельном cpp-файле с именем functions.

2.1 Добавление в проект нового файла

В проект можно добавить либо уже существующий файл, либо вновь созданный.

Чтобы добавить в проект уже существующий файл, необходимо выполнить следующие действия:

1. Вызвать контекстное меню (нажав правой кнопкой мыши) для папки Source Files зоны Solution Explorer.
2. В контекстном меню выбрать Add → Existing Item.
3. В появившемся окне открытия файлов выбрать нужный файл.

А для того чтобы добавить в проект новый файл, необходимо выполнить соответственно следующие действия:

1. Вызвать контекстное меню (нажав правой кнопкой мыши) для папки Source Files зоны Solution Explorer.
2. В контекстном меню выбрать Add → New Item.
3. В появившемся окне на вкладке Categories выбрать Code, на вкладке Templates выбрать либо C++ File (*.cpp) либо Header File (*.h), задать имя файла и нажать Add.

В наш проект добавим новый cpp-файл. Назовем его functions.

Листинг файла functions.cpp.

```
#include "stdafx.h"

double fact(int n)
{
     double f=1;
     for (int i=1;i<=n;i++)
          f*=i;
     return f;
}
```

Переменные могут определяться в управляющих выражениях циклов `for` и `while`, в условиях оператора `if` и критериях выбора оператора `switch`.

В нашем проекте:

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

При таком объявлении время жизни для переменной `i` заканчивается за пределами данного цикла.

В начале каждого .cpp-файла нужно обязательно помещать следующую строку:

```
#include "stdafx.h"
```

Это вызвано особенностью построения исполняемого файла в среде Microsoft Visual Studio.

Листинг Second.cpp **– вариант 1.**

```
#include "stdafx.h"
#include <iostream>

using namespace std;

double fact(int);

int _tmain(int argc, _TCHAR* argv[])
{
     int n,m;
     cout<<"Input n and m"<<endl;
     cin>>n>>m;
     cout<<"rezult = "<<fact(n)/(fact(m)*fact(n-m))<<endl;
     return 0;
}
```

Вспомним правило о том, что всякое имя, прежде чем будет использовано, должно быть описано. Следовательно, чтобы в основной функции `main()`, расположенной в файле Second.cpp, можно было вызывать функцию `fact(n)`, необходимо в файл Second.cpp добавить строку с прототипом функции для вычисления факториала.

```
double fact(int);
```

2.2 Включение заголовочных файлов

Правило хорошего стиля требует в случае вынесения функций в отдельный cpp-файл создавать соответствующий ему h-файл, содержащий прототипы соответствующих функций. Причем рекомендуется, чтобы имена cpp-файла и соответствующего ему h-файла совпадали.

Это удобно тем, что не нужно размещать в основном файле, содержащем функцию `main()`, прототипы всех функций. Достаточно подключить соответствующий h-файл.

Листинг Second.cpp **– вариант 2.**

```
#include "stdafx.h"
#include <iostream>
#include "functions.h"

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
     int n,m;
     cout<<"Input n and m"<<endl;
     cin>>n>>m;
     cout<<"rezult = "<<fact(n)/(fact(m)*fact(n-m))<<endl;
     return 0;
}
```

Листинг файла functions.h.

```
double fact(int);
```

Обратите внимание, что подключение `#include "functions.h"` отличается от подключения `#include <iostream>`.

Они представляют две формы директивы `#include`.

```
#include <iostream>
#include "functions.h"
```

В угловых скобках обычно подключается заголовочный файл стандартной библиотеки, в двойных кавычках – заголовочный файл, определяемый программистом.

2.3 Библиотеки в C++

Многие библиотеки содержат большое количество функций и переменных. Чтобы избавить программиста от лишней работы и обеспечить логическую последовательность внешних объявлений, в C и C++ используется *механизм заголовочных файлов*.

Заголовочным файлом называется файл, содержащий внешние объявления библиотеки.

Создатель библиотеки представляет заголовочный файл. Чтобы объявить в программе функции и внешние переменные этой библиотеки, пользователь просто включает в программу заголовочный файл препроцессорной директивой `#include`.

Заставляя правильно работать с заголовочными файлами, язык C++ гарантирует согласованность библиотек и сокращает количество ошибок благодаря всеобщему использованию одного и того же интерфейса.

Заголовочный файл определяет *контракт* между разработчиком библиотеки и ее пользователем. Контракт описывает структуры данных, утверждает типы аргументов и возвращаемых значений при вызовах функций.

2.4 Подробнее о заголовочных файлах

Заголовочные файлы практически обязательны для упрощения разработки программ, в них включается предельно конкретная информация – объявления.

Директива `#include` указывает препроцессору, что он должен открыть файл с заданным именем и вставить его содержимое на место директивы.

При включении файла в угловых скобках препроцессором обычно используется некоторая разновидность «пути поиска», задаваемого в переменной окружения или в командной строке компилятора. Способ определения пути поиска зависит от компьютера, операционной системы и реализации C++.

Директива с именем файла, заключенным в кавычки, сообщает препроцессору, что поиск заданного файла начинается с текущего каталога. Если файл не обнаружен, то директива обрабатывается примерно так, как если бы вместо кавычек использовались угловые скобки.

Библиотеки, унаследованные из языка C, сохранили традиционное для заголовочных файлов расширение .h. Впрочем их можно использовать и при

включении в современном стиле C++, для чего имя заголовочного файла снабжается префиксом `c`.

Рассмотрим следующий фрагмент:

```
#include <stdio.h>
#include <stdlib.h>
```

В современном варианте он выглядит так:

```
#include <cstdio>
#include <cstdlib>
```

Это правило распространяется на все стандартные заголовки C. При просмотре кода программы сразу понятно, когда в программе используются библиотеки C, а когда – библиотеки C++.

В одной программе эти две формы лучше не смешивать.

Что же стоит включать в заголовочные файлы? Основное правило рекомендует ограничиться одними объявлениями.

Принцип единственного определения в C++:

объявлений может быть сколько угодно, но определение должно быть только одно.

2.5 Проблема многократного объявления в заголовочных файлах

При размещении объявления в заголовочном файле может оказаться, что файл многократно включается в сложную программу.

Если в файле B подключается файл A, а файле C подключаются и файл A и файл B, то возникает риск многократного включения заголовочного файла A и множественного объявления его содержимого.

Чтобы предотвратить ошибки при многократном включении заголовочного файла, необходимо использовать специальные директивы препроцессора. В стандартных заголовочных файлах C++, таких как `<iostream>`, такие директивы уже используются.

В каждом заголовочном файле следует сначала проверить, не был ли этот заголовочный файл уже включен в текущий cpp-файл. Это делается при помощи препроцессорного флага. Если флаг не установлен, значит, флаг еще не включался; установите флаг и выполните объявления. Если флаг уже установлен, то код файла с объявлениями игнорируется.

Примерный формат заголовочного файла:

```
#ifndef HEADER_FLAG
#define HEADER_FLAG
// необходимые объявления
#endif    // HEADER_FLAG
```

Имя `HEADER_FLAG` можно заменить любым уникальным именем, но существует надежная схема выбора имен: записать имя заголовочного файла символами верхнего регистра и заменить точки символами подчеркивания.

Например, для файла с именем `simple.h` можно, следуя данному правилу, получить имя `SIMPLE_H`.

2.6 Перенос в визуальную среду разработки Microsoft Visual Studio программы, уже разработанной в другой среде разработки.

Необходимо создать новый проект.

Подключить в проект описанным выше способом (см. **Добавление в проект нового файла**) все файлы кроме файла, содержащего функцию `main()`.

Текст программы из cpp-файла, содержащего функцию `main()` нужно перенести в cpp-файл проекта, содержащий функцию

```
int _tmain(int argc, _TCHAR* argv[]).
```

При этом необходимо соблюдать все правила и рекомендации по созданию проектов, описанные в начале раздела.

Рекомендуется функцию `main()` делать небольшой по объему, вынося как можно больше фрагментов в другие функции и размещая их соответственно в других cpp-файлах.

Рекомендуется разбивать большой код на относительно независимые части и объединять в cpp-файлы группы взаимосвязанных функций.

2.7 Тесты рубежного контроля

Примечание: возможно несколько вариантов правильного ответа

№	Вопрос	Варианты ответа		
		1	2	3
1	В какие кавычки заключаются имена заголовочных файлов при подключении	<>	" "	' '
2	Требования к именам заголовочных файлов	Имя может быть любым	Имя обязано совпадать с именем файла на языке C++	Имя может быть любым, но желательно, чтобы оно совпадало с именем файла на языке C++
3	Сколько объявлений одного объекта может содержаться в проекте	1	2	Сколько угодно
4	Сколько определений одного объекта может содержаться в проекте	1	2	Сколько угодно

2.8 Вопросы для самопроверки

1. Чем отличаются конфигурации Debug и Release?
2. Зачем нужны заголовочные файлы?

3. Какие имена рекомендуется давать заголовочным файлам?

4. Что следует размещать в заголовочных файлах?

5. В чем состоит проблема многократного объявления в заголовочных файлах? Как ее необходимо решать?

МОДУЛЬ 3. ИСПОЛЬЗОВАНИЕ ОТЛАДЧИКА

Цель. Научить использовать отладчик для эффективного поиска ошибок, отладки и тестирования программ в среде Visual Studio .

Требуемый начальный уровень подготовки. Владение навыками программирования. Знакомство с языком программирования C++. Умение разрабатывать проекты на языке C++, в том числе и состоящие из нескольких модулей. Знакомство с модульной структурой проектов.

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:

1. Программа выполняется, но выдает неправильный результат.
2. Программа завершается сообщением об ошибке времени выполнения.
3. Программа не завершает выполнения (зацикливается).
4. 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) вспомогательных операторов 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.

3.1 Подготовка программы к отладке

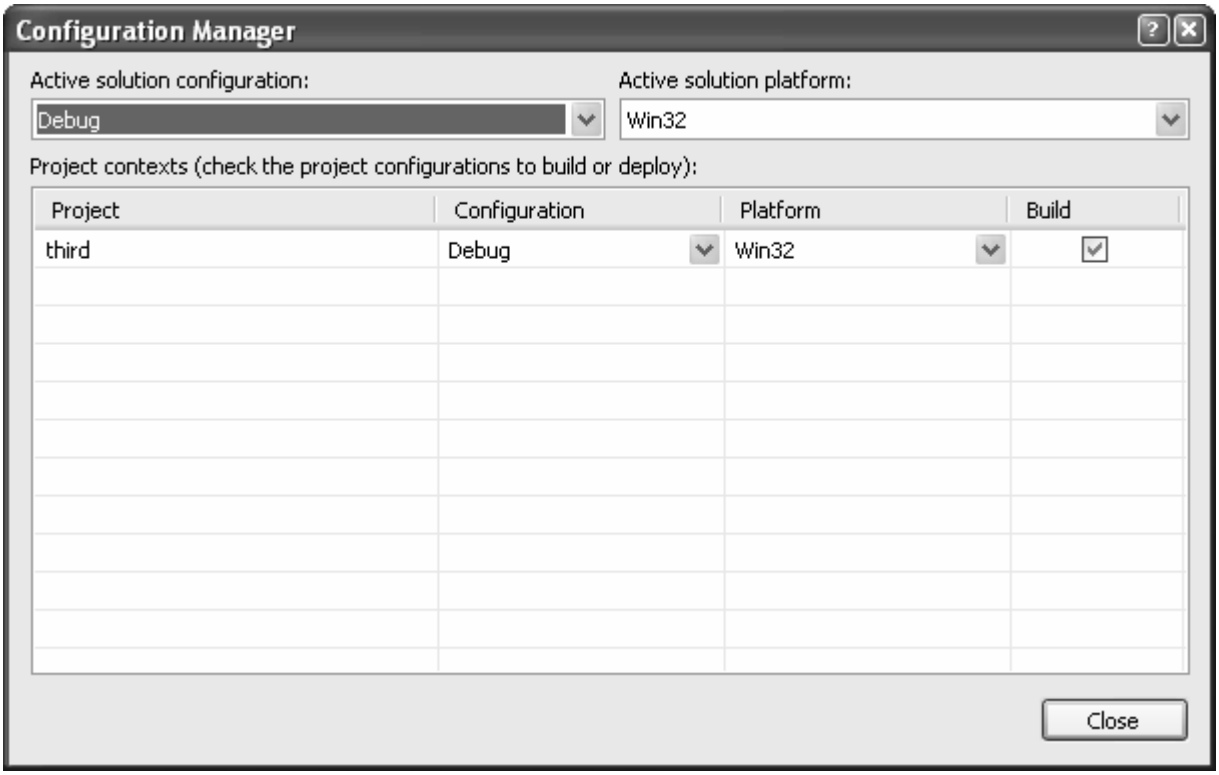

Рисунок 12

Для того чтобы воспользоваться возможностями отладчика (Debugger) необходимо, чтобы проект был откомпилирован в конфигурации Debug.

Конфигурация Debug отличается от конфигурации Release 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.

Переключение конфигурации можно осуществить следующим образом:

Build → Configuration Manager…

В открывшемся окне (Рисунок 12) необходимо выбрать значение Active solution configuration: Debug или Release.

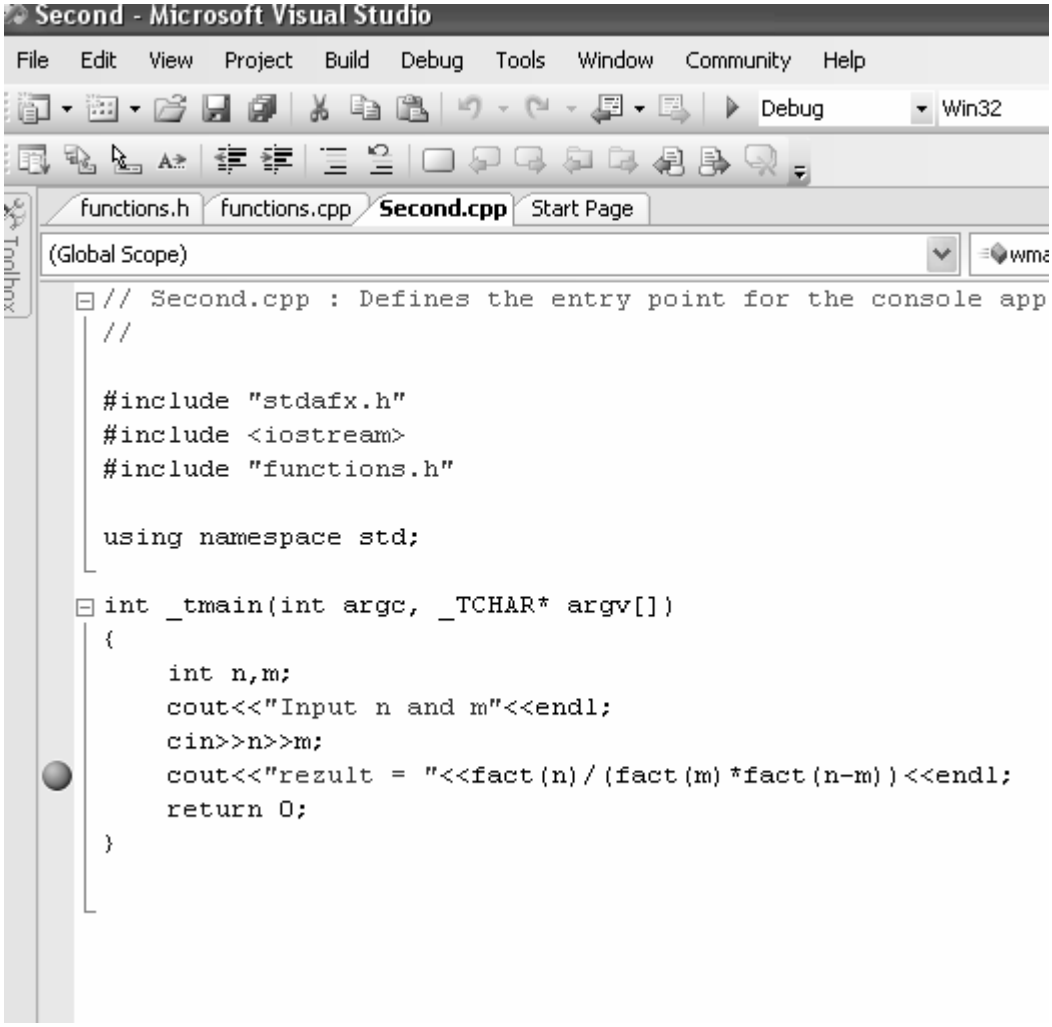

Рисунок 13

Эти же действия можно выполнить непосредственно на панели инструментов (Рисунок 13), выбрав значения Debug или Release.

Чтобы запустить процесс отладки используется один из вариантов команд:

Debug→Start Debugging (F5) или Debug→Step Into (F11) или Debug→ Step Over (F10).

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 (Рисунок13).

Рисунок 14

– Continue

– Break All (Ctrl+Alt+Break)

– Stop Debugging (Shift+F5)

– Restart (Ctrl+Shift+F5)

– Step Into (F11)

– Step Over (F10)

– Step Out (Shift+F11)

– Данную пиктограмму рассмотрим подробнее.

После запуска отладчика командой Debug→Start Debugging (F5) будут выполнены все операции до первой точки останова.

Если следующая выполнимая команда содержит вызов функции, то выполнить функцию по шагам Debug→Step into (Shift+F7).

После запуска отладчика командой Debug→Start Debugging (F5) активизируется панель отладчика вверху на панели инструментов (Рисунок 14).

Она позволяет активизировать различные окна, доступные в режиме Debug. Например, одно из наиболее часто используемых при отладке – окно для просмотра значений локальных переменных (Рисунок 15). Это окно по умолчанию располагается в нижней части экрана.

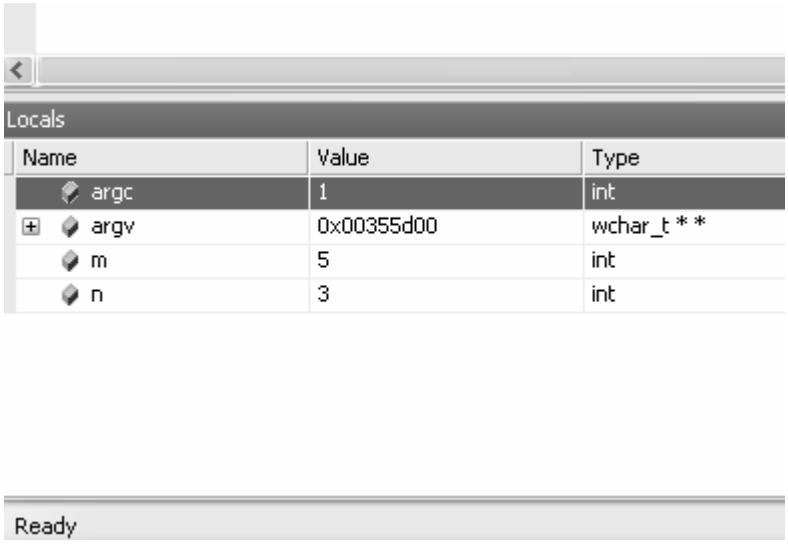

Рисунок 15

В процессе пошагового выполнения программы строка, перед выполнением которой произошел останов, отмечается слева от текста программы желтой стрелкой (Рисунок16).

Управлять дальнейшим выполнением программы можно через команды меню или кнопки на панели меню отладчика.

Выполнение команды Continue позволит выполнить все команды до следующей точки останова и так далее.

Если следующая выполнимая команда содержит вызов функции, то выполнить функцию по шагам Debug→Step Into (F11). Завершить пошаговое выполнение функции и выйти из нее Debug→Step Out (Shift+F11). Команда Debug→Step Over (F10) позволяет выполнять функцию, не заходя в ее реализацию. Завершить отладку позволяет команда Debug→Stop Debugging (Shift+F5).

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

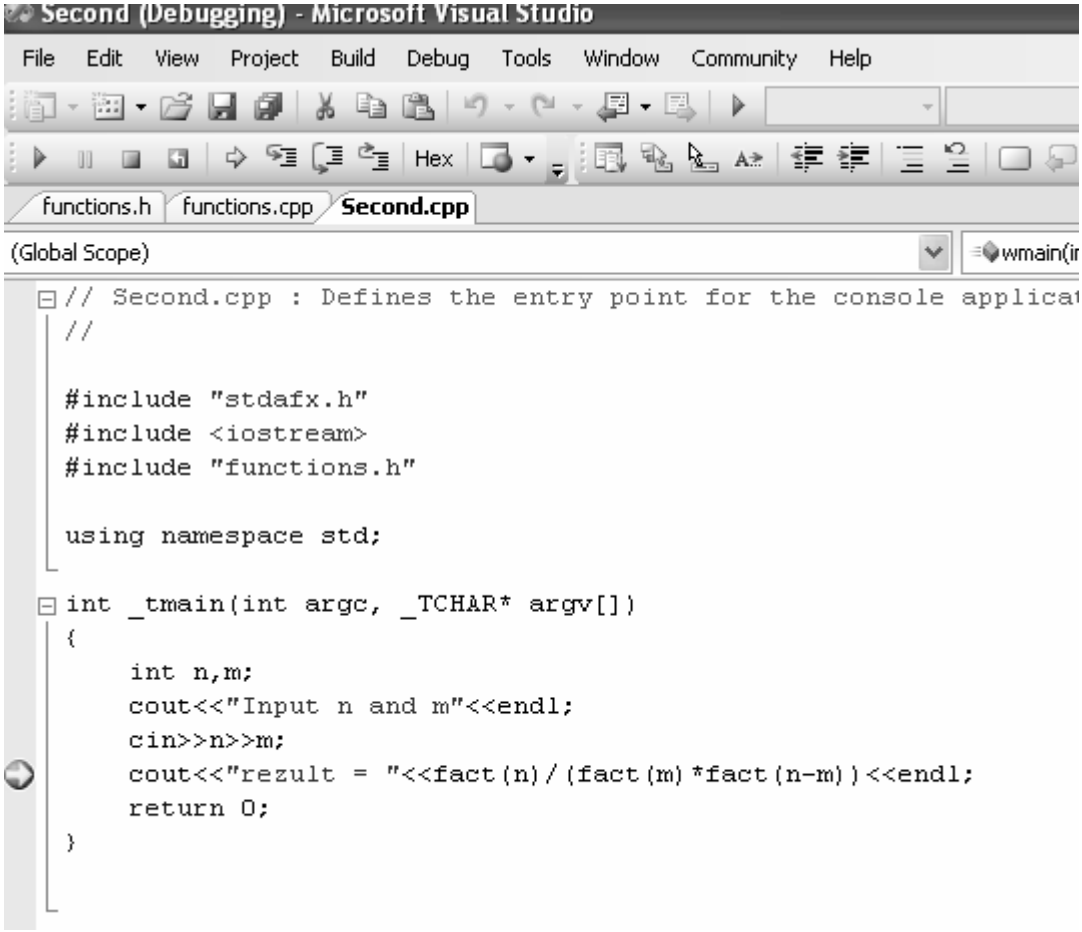

Рисунок 16

Окно локальных переменных (Рисунок 15) в каждый момент пошагового выполнения содержит значения локальных переменных и аргументов функции, которая выполняется в данный момент.

3.2 Поиск ошибки в программе с помощью отладчика

Программа 3. Рассмотрим программу для решения задачи поиска всех простых чисел в диапазоне от 2 до N.

Листинг Third.cpp – первый вариант

```
#include "stdafx.h"
#include <cmath>
#include <iostream>

using namespace std;
```

```
bool IsSimple(int x)
{
     for (int i=2;i<=int(sqrt((double)x));i++)
     {
          int y=x%i;
          if (y=0)
               return false;
     }
     return true;
}

int _tmain(int argc, _TCHAR* argv[])
{
  int N;
  cout<<"N = ";
  cin>>N;
  for (int i=2;i<=N;i++)
       if (IsSimple(i))
            cout<<i<<' ';
  cout<<endl;
  system("PAUSE");
  return 0;
}
```

После запуска программы на выполнение можно увидеть следующие результаты (Рисунок 17).

Легко заметить, что не все выведенные числа являются простыми. Следовательно, программа содержит логические ошибки. Вполне возможно, что после визуального изучения текста программы ошибку обнаружить не удается. В таком случае следует воспользоваться возможностями отладчика для поиска ошибок.

Программа обязательно должна быть откомпилирована в режиме Debug.

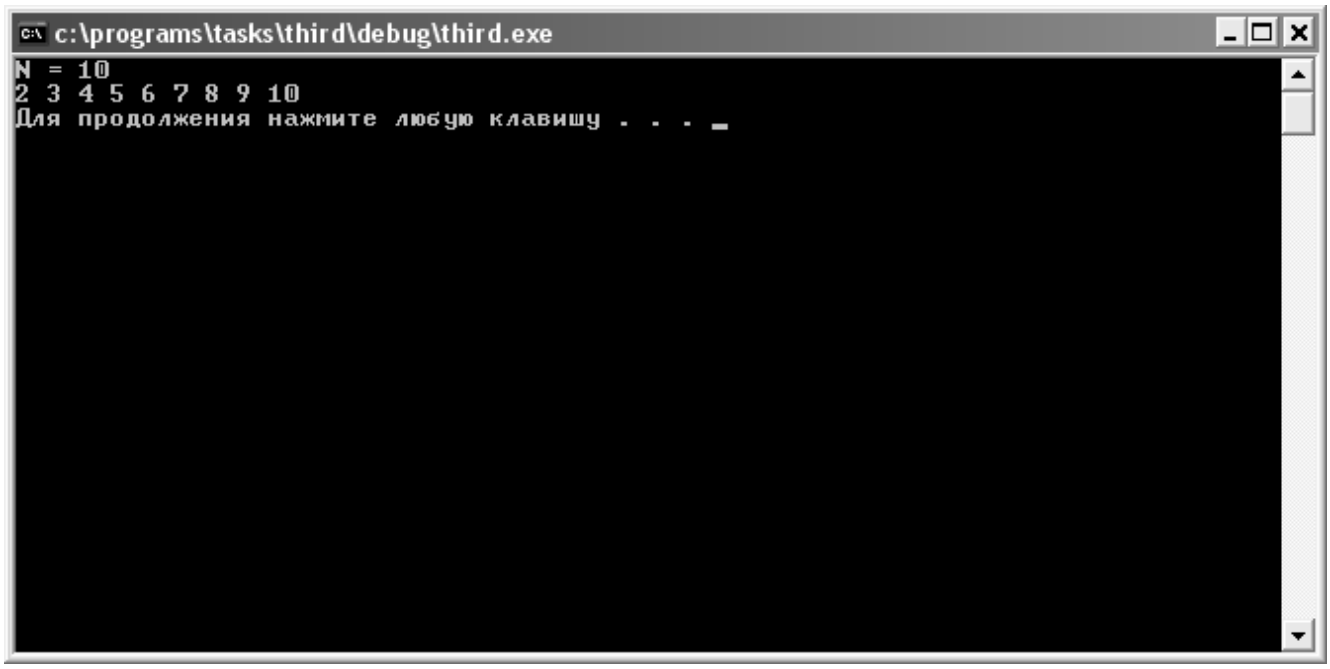

Рисунок 17

Исходя из текста программы, можно сделать вывод, что ошибки следует искать в реализации функции `IsSimple`. Анализ результатов работы программы показывает, что функция `IsSimple` всегда возвращает значение `true` (истина).

Выполним трассировку функции `IsSimple`, т.е. реализуем ее пошаговое выполнение. Поставим точку останова в строке

```
int y=x%i;
```

Запустим программу на выполнение при N=10.

Выполнение приостановится в точке останова. С этого момента будем выполнять программу пошагово, обязательно отслеживая изменение значений переменных, используемых в функции `IsSimple`: `i`, `x` и `y` – в окне локальных переменных (Рисунок 15).

При пошаговом выполнении программы можно обнаружить, что выражение `y=0` в операторе

```
if (y=0)
     return false;
```

всегда принимает значение `false` (ложь).

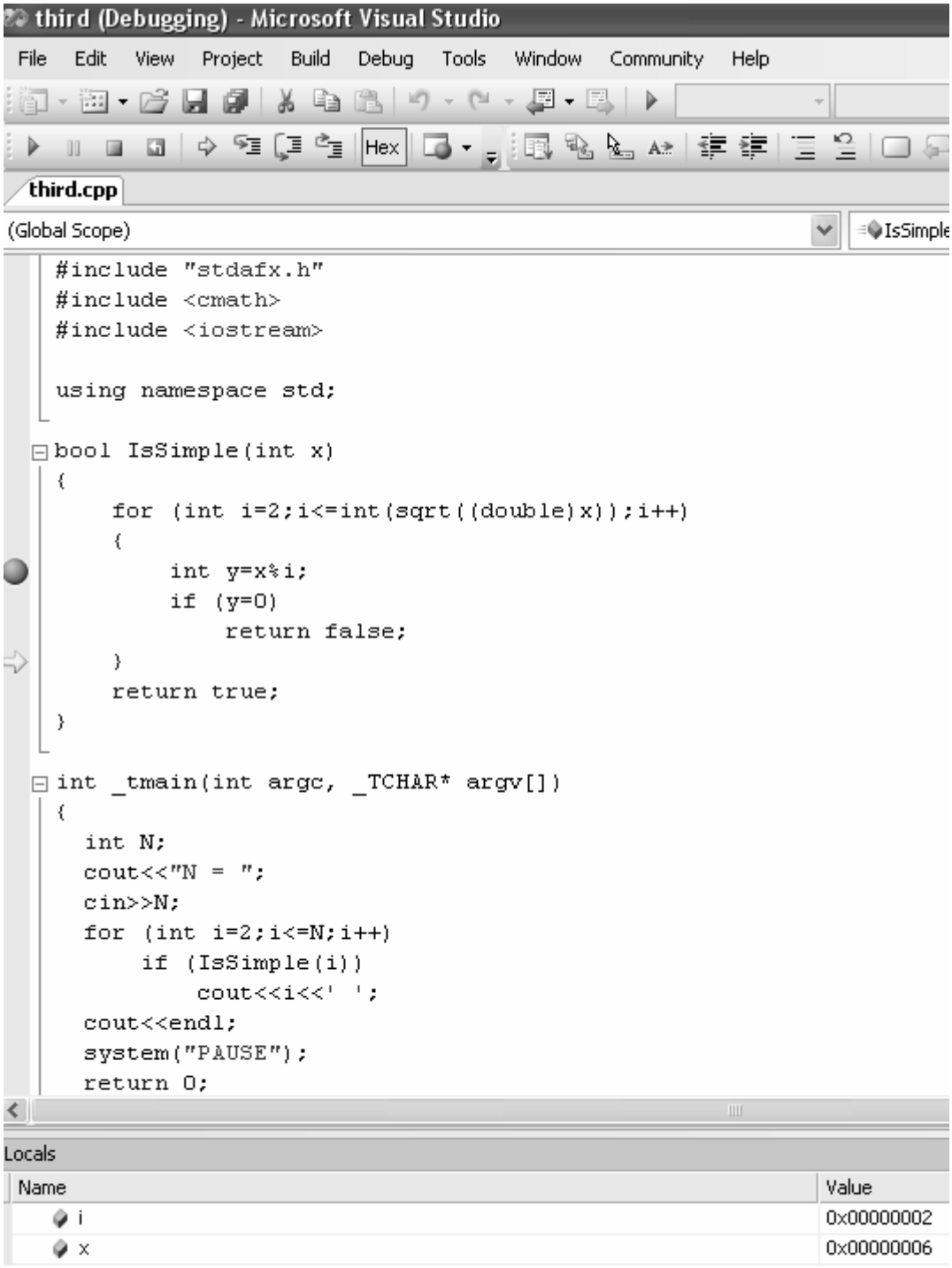

Рисунок 18

Детальное рассмотрение выражения `y=0` позволяет выявить ошибку. В выражении вместо операции сравнения используется операция присваивания, т.е. переменной `y` всегда присваивается 0, а это значит, что выражение всегда будет принимать значение `false` (ложь).

Можно заметить, что в списке локальных переменных y то появляется, то исчезает. Это связано с тем, что она писана внутри цикла. Чтобы все время отслеживать ее значение необходимо расположить ее описание перед циклом.

```
bool IsSimple(int x)
{
     int y;
     for (int i=2;i<=int(sqrt((double)x));i++)
     {
          y=x%i;
          if (y=0)
               return false;
     }
     return true;
}
```

После обнаружения ошибки осталось только ее исправить.

Листинг Third.cpp – второй вариант

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

bool IsSimple(int x)
{
     for (int i=2;i<=int(sqrt((double)x));i++)
     {
          int y=x%i;
          if (y==0)
               return false;
     }
     return true;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
  int N;
  cout<<"N = ";
  cin>>N;
  for (int i=2;i<=N;i++)
       if (IsSimple(i))
            cout<<i<<' ';
  cout<<endl;
  system("PAUSE");
  return 0;
}
```

3.3 Проектные задания для самостоятельного выполнения

1. Все рассмотренные программы оформить в виде библиотеки с соответствующими заголовочными файлами.

2. Найти логические ошибки в каждом из следующих фрагментов программы и исправить ошибки. Для выполнения фрагментов их необходимо написать программу и вставить в нее фрагменты. Для поиска ошибок можно воспользоваться отладчиком.

2.1.
```
int sum(int a, int b)
{
     for(int i=a, int s=0;i<=b;i++)
          s+=i;
     return s;
}
```

2.2.
```
int fact(int n)
{    int f;
     for (int i=1; i<n; i++)
          f*=i;
     return f;
```

```
}
```

2.3.

```
void swap(int a,int b)
{
     int c=a;
     a=b;
     b=c;
}
```

2.4.

```
int max(int a, int b, int c)
{
     int maxi;
     if (a>maxi)
          maxi=a;
     else if (b>maxi)
          maxi=b;
     else if (c>maxi)
          maxi=c;
     return maxi;
}
```

ЛИТЕРАТУРА

1. Брюс Эккель. Философия C++. Введение в стандартный C++. – СПб.: Питер, 2004.- 572 с.
2. Брюс Эккель, Чак Эллисон. Философия C++. Практическое программирование. – СПб.: Питер, 2004.- 608 с.
3. Х.М.Дейтел, П.Дж.Дейтел. Как программировать на C++. – М.: ЗАО «Издательство БИНОМ», 2000 г. – 1024 с.
4. Эндрю Кениг, Барбара Му. Эффективное программирование на C++. Практическое программирование на примерах. – М.: Издательский дом «Вильямс», 2002 г. – 384 с.
5. Бьерн Стауструп. Язык программирования C++. Специальное издание. – М.: ООО «Бином-Пресс», 2004 г. – 1104 с.