Московский государственный технический университет
имени Н. Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети

«УТВЕРЖДАЮ»
Заведующий кафедрой ИУ-6
_____ Сюзев В.В.

Г. С. Иванова, Т.Н. Ничушкина

Консольные приложения Visual C++ в среде Microsoft Studio 2008

Методические указания по выполнению лабораторных работ

Москва 2013

Оглавление

Введение

Интегрированная среда программирования **Microsoft Visual Studio 2008** предназначена для создания различных 32^x разрядных программ, в том числе и консольных приложений WINDOWS, написанных на C++.

1 Создание заготовки консольного приложения

Главное окно среды при запуске имеет вид, представленный на рисунке 1. Окно включает меню, панели инструментов, текстовый редактор со стартовой страницей и окно навигатора **Solution Explorer**. Причем последнее включает несколько вкладок, позволяющих просматривать различную информацию.

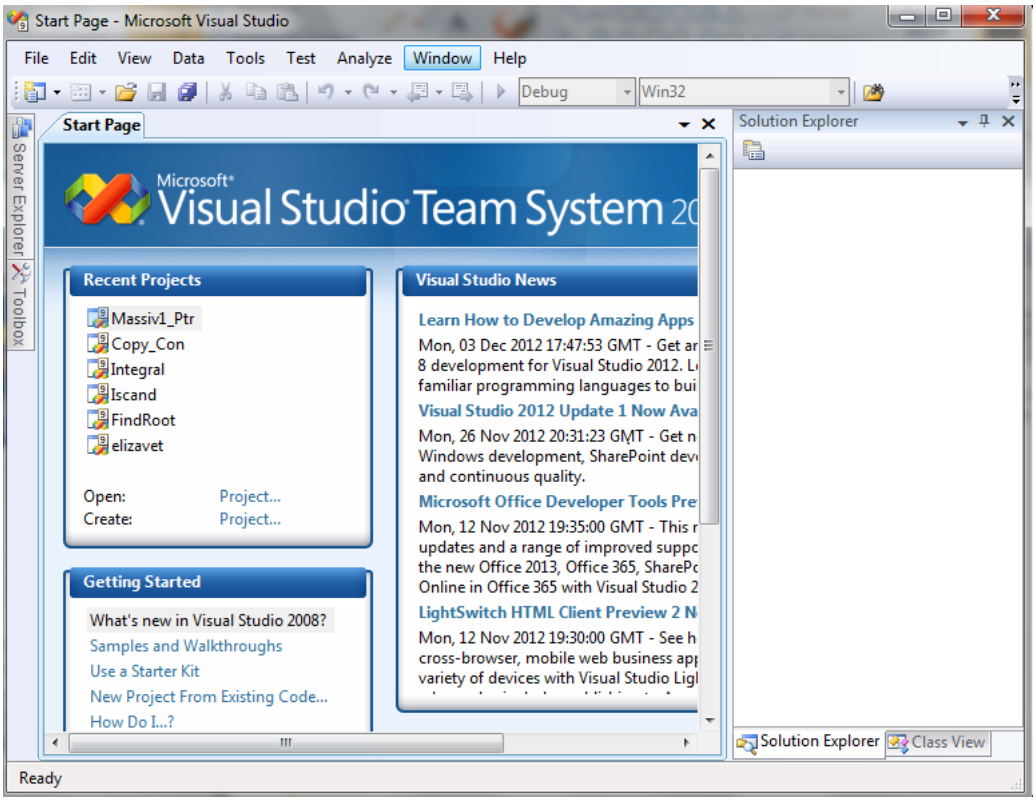

Рисунок 1 – Вид главного окна среды при отсутствии открытого проекта

Примечание. Вид главного окна зависит от профиля пользователя, который настраивается на вкладке **My Profile** стартовой страницы. Именно на этой вкладке **Microsoft Visual Studio** открывается при самом первом запуске. Никаких настроек профиля можно не выполнять, а сразу перейти на закладку **Projects**.

Создание заготовки консольного приложения выполняется либо нажатием кнопки **New Project** стартовой страницы, либо с использованием меню **File/New/Project...**.

В появившемся окне **New Project** выбираем тип проекта **Visual C++** шаблон **Win32 Console Application** и вводим имя проекта **Con1** в поле **Name** (см. рисунок 2). При желании можно указать отличное от стандартного местонахождение папки будущего проекта в позиции **Location** и отличное от имени проекта имя программы **Solution Name**, но особой необходимости в этом нет.

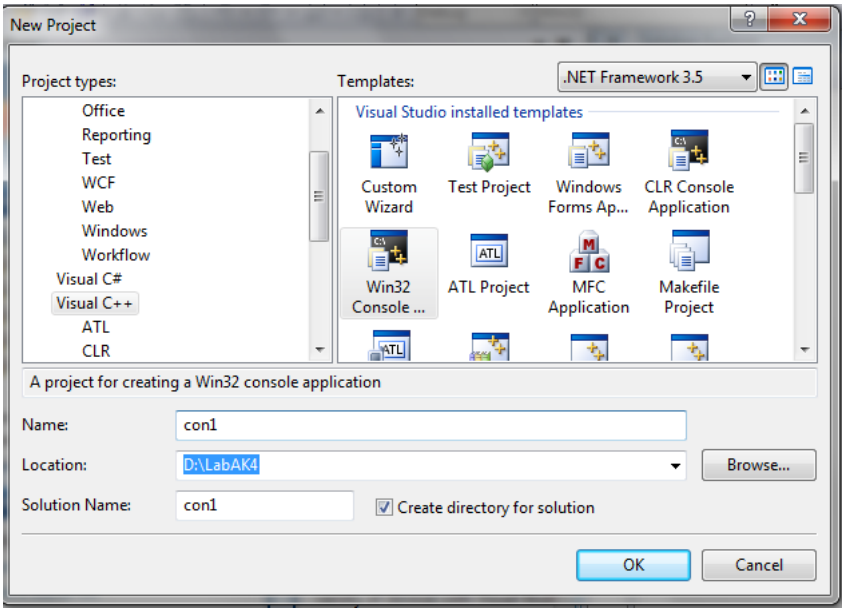

Рисунок 2 – Вид окна выбора типа проекта и задания его имени и местоположения

В процессе создания проекта мастер предоставляет описание приложения на вкладке **Overview** окна **Win32 Application Wizard** (рисунок 3).

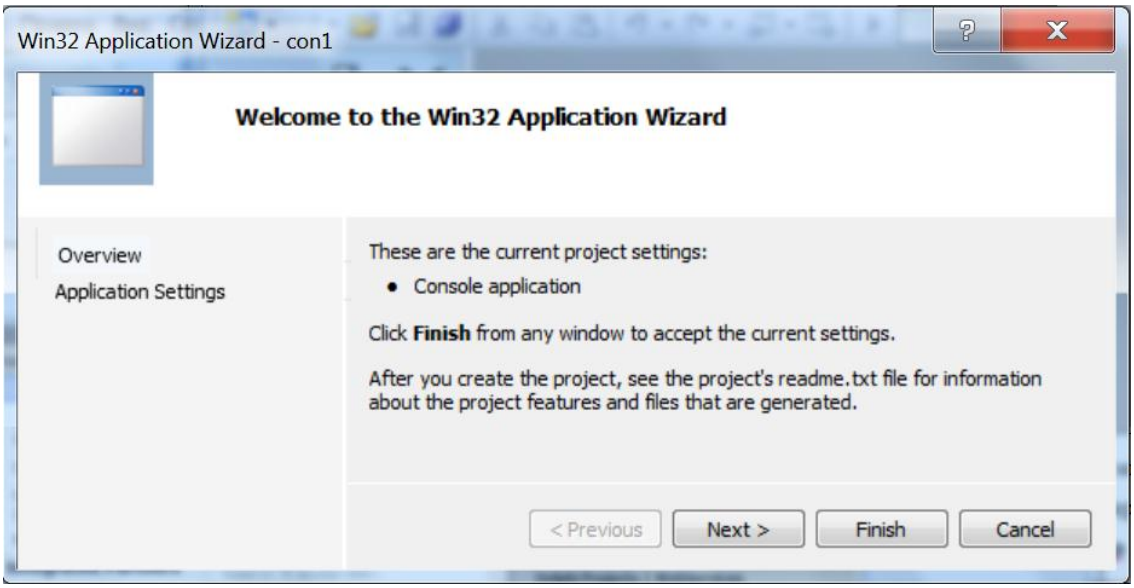

Рисунок 3 – Вкладка Overview мастера создания проектов

Переключившись на другую вкладку того же окна **Application Settings** мы получаем возможность указать параметры проекта (рисунок 4).

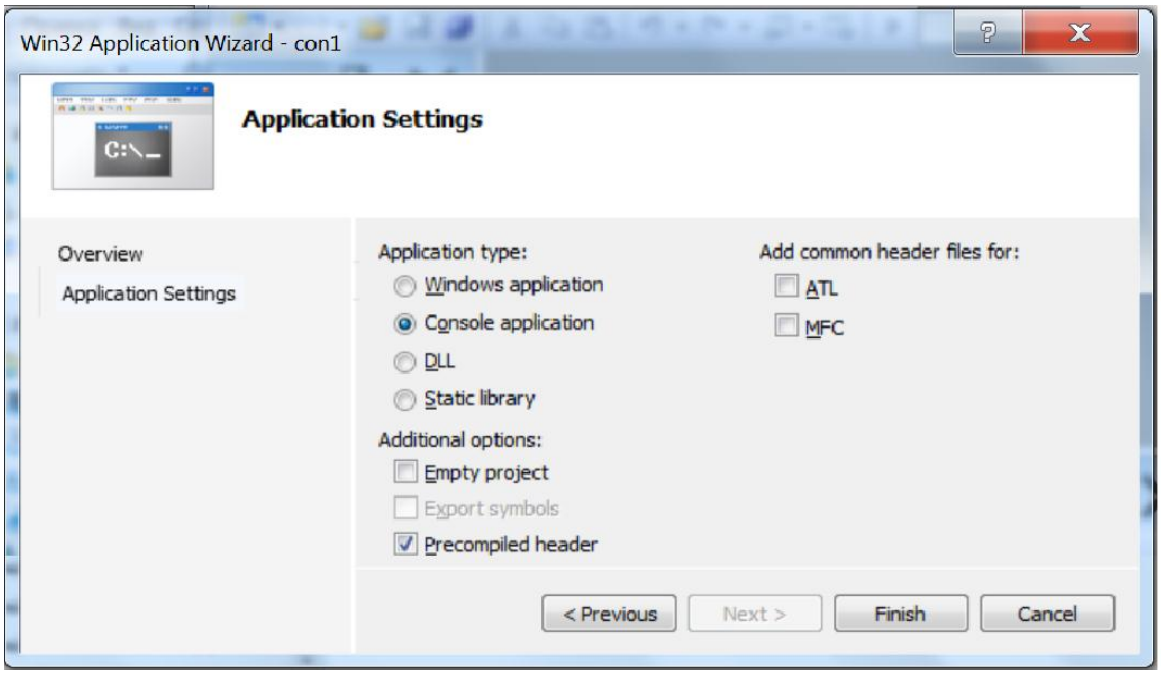

Рисунок 4 –Вкладка задания параметров проекта

Как следует из текста вкладки, мастер предлагает создать проект **Console Application** с параметром **Precompiled header**. Такой проект включает файлы `stdafx.h` и `stdafx.cpp` называемые предкомпилируемым заголовком и заготовку основной программы.

Предкомпилируемый заголовок оптимизирует компиляцию, поскольку при его использовании в файл `stdafx.cpp` собирают все заголовочные файлы, которые в результате компилируются один раз. При использовании проектов данного типа следует помнить, что в них файл `stdafx.h` должен подключаться ко всем файлам, содержащим исходные тексты, т.е. файлам с расширением `.cpp`.

В процессе обучения целесообразно использовать пустые проекты, которые создаются при выборе параметра **Empty project**. Такой проект не включает никаких файлов и, соответственно, заготовки.

Проект с предкомпилируемым заголовком и заготовкой основной программы. Заготовка находится в файле проекта с расширением .cpp, который называют *файлом реализации программы*. Для созданного проекта этот файл имеет имя **Con1.cpp**.

Чтобы увидеть текст заготовки программы необходимо открыть соответствующий файл в окне текстового редактора. Для этого переходим в окно Навигатора на вкладку

Solution Explorer и щелкаем мышью по символу «+» около имени проекта **Con1**, а затем – около **Source Files**, раскрывая список исходных файлов проекта (см. рисунок 5).

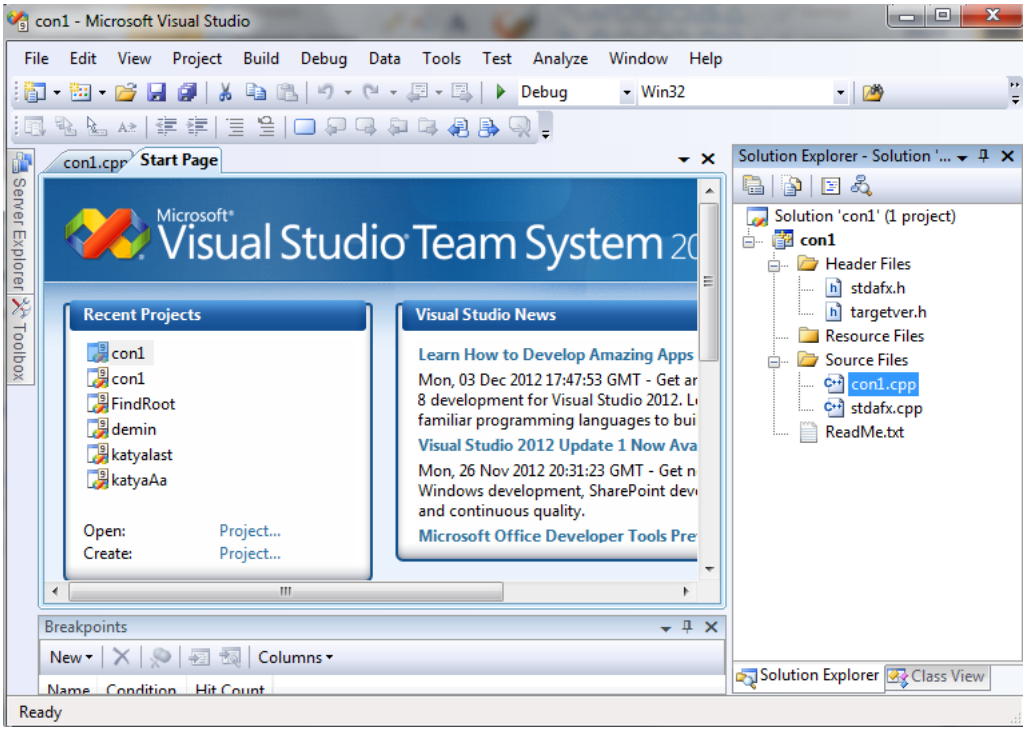

Рисунок 5 – Работа с вкладкой **Solution Explorer** Навигатора

Находим файл **Con1.cpp** и дважды щелкаем мышью по его имени. Содержимое файла будет показано на вкладке **Con1.cpp** текстового редактора, в другом окне которого по-прежнему высвечивалась стартовая страница (см. рисунок 6).

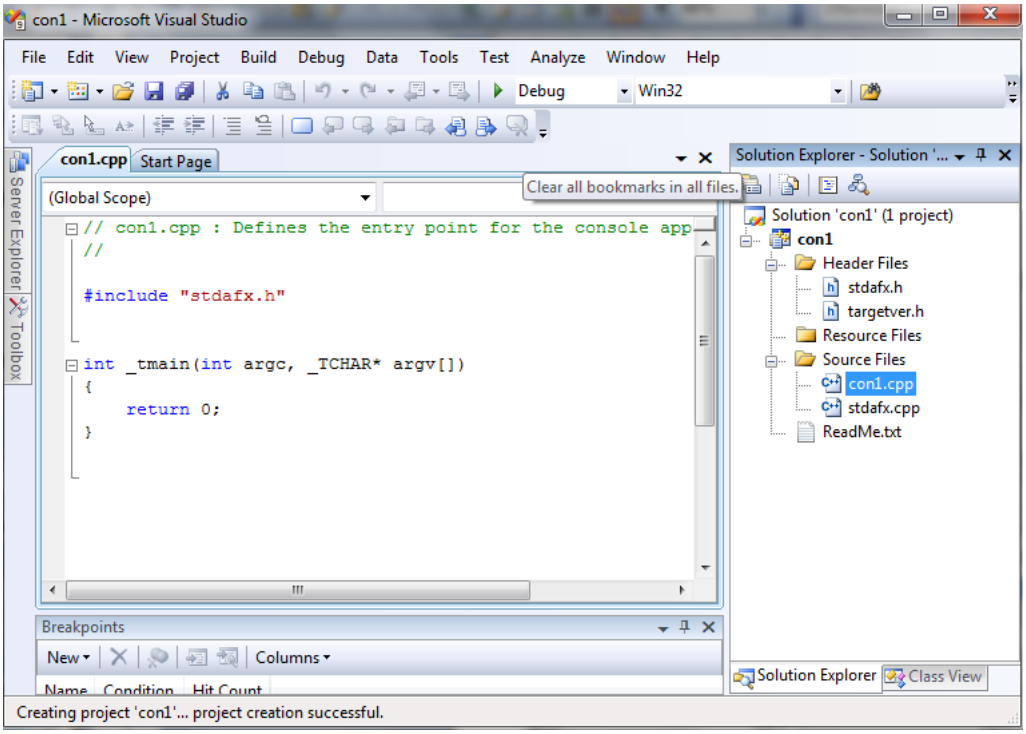

Рисунок 6 – Окно текстового редактора с заготовкой консольного приложения

В заготовке оператор `return 0;` возвращает 0 в качестве результата основной функции, поскольку функция предполагает возврат целого значения. Если заменить `int` при объявлении функции на `void`, т.е. использовать процедуру вместо функции, то этот оператор из текста заготовки следует удалить.

Имя основной функции `_tmain()`, предлагаемое по умолчанию, обеспечивает обрабатку символов кодировки `Unicode` при вводе с консоли. Для учебных программ его можно оставить или безболезненно поменять на `void main()` без параметров.

Если же программа собирается использовать парамеры командной строки, то заголовок следует изменить следующим образом

```
void main(int argc, char* argv[])
```

Пустой проект файла исходного текста и заготовки не содержит. Для ввода программы в проект следует включить файл исходного текста с расширением `.cpp`. Для этого можно:

а) установить курсор в новигаторе **Solution** на папку **Source Files** и щелкнуть правой кнопкой; в появившемся локальном меню выбрать подменю **Add** и в подменю – пункт **New Item...**; в открывшемся окне (рисунок 7) выбрать файл с расширением `.cpp`;

б) в меню выбрать пункт **Project/Add New Item** и в открывшемся окне выбрать файл с расширением `.cpp`.

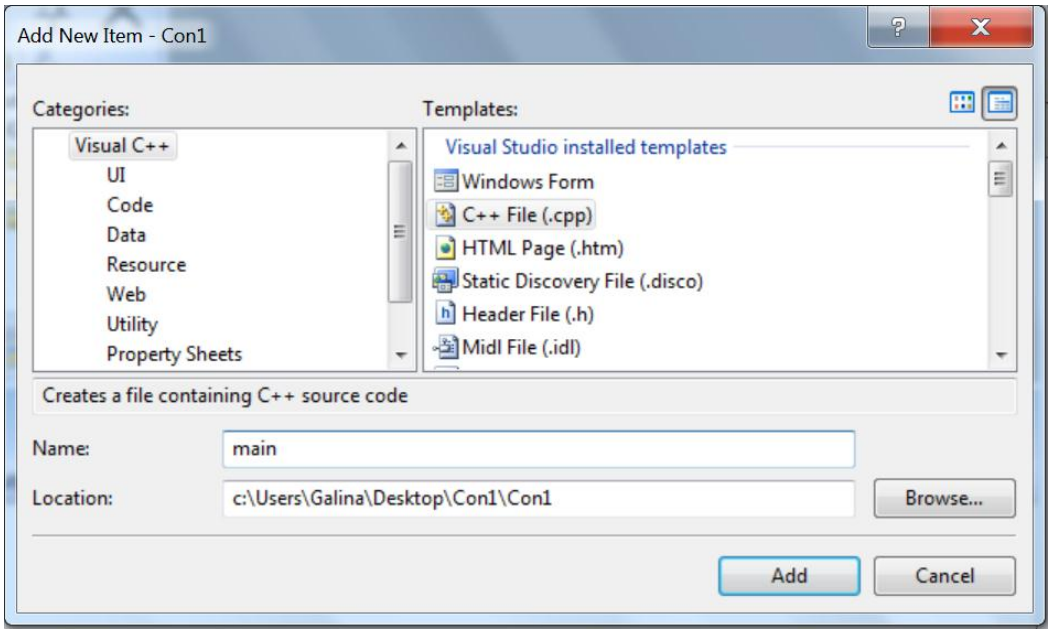

Рисунок 7 – Окно добавление файла к проекту

В качестве имени файла введем имя `main` или `con1`, как в проекте с прекомпилируемым заголовком.

2 Ввод программы

В качестве примера введем в среду программирования программу вычисления наибольшего общего делителя двух целых чисел. Причем само вычисление выделим в функцию nod, оставив в основной программе ввод данных и вывод результата.

```
#include "stdafx.h" // только при создании проекта с заголовком!!!
#include <locale.h> // для подключения русского языка
#include <stdio.h>  // подключение процедур ввода вывода
#include <conio.h>  // подключение процедур консольного режима
int nod(int x, int y)
{
    while (x!=y)
        if (x>y) x=x-y;
        else y=y-x;
    return y;
}
void main()
{
    int a,b;
    setlocale(0,"russian"); // подключение русского языка
    puts("Введите два натуральных числа:");
    scanf("%d %d",&a,&b);
    printf("НОД %d и %d = %d.\n",a,b,nod(a,b));
    puts("Нажмите любую клавишу для завершения...");
    getch();
}
```

Программа специально настроена на использование при выводе русского языка. Для этого в начале программы подключена библиотека, а в самой программе вызвана

процедура подключения необходимой кодировки. Консольная программа с такими настройками может выводить русские тексты.

Две последние строки программы – запрос нажатия клавиши и ввод – осуществляют задержку закрытия окна консольного приложения до чтения результатов пользователем. Эти операторы необходимо вставлять в каждое консольное приложение, создаваемое в рассматриваемой среде.

3 Запуск программы на выполнение

Для запуска программы на выполнение используют команды подменю пункта главного меню **Debug** или клавиши «быстрого доступа», показанные в скобках:

- **Start Debugging (F5)** – компиляция, компоновка и выполнение с отладкой – позволяет в случае выдачи сообщения об ошибке выполнения просмотреть содержимое интересующих программиста переменных (см. раздел 6)**;**
- **Start Without Debugging (Ctrl +F5)** – компиляция, компоновка и выполнение программы без отладки**.**

В обоих случаях, если готовая программа не существует, среда выдаст соответствующих запрос на выполнение компиляции и компоновки программы перед запуском.

Если при компиляции или компоновке обнаружены ошибки, то среда запрашивает, продолжить или нет создание программы (рисунок 8).

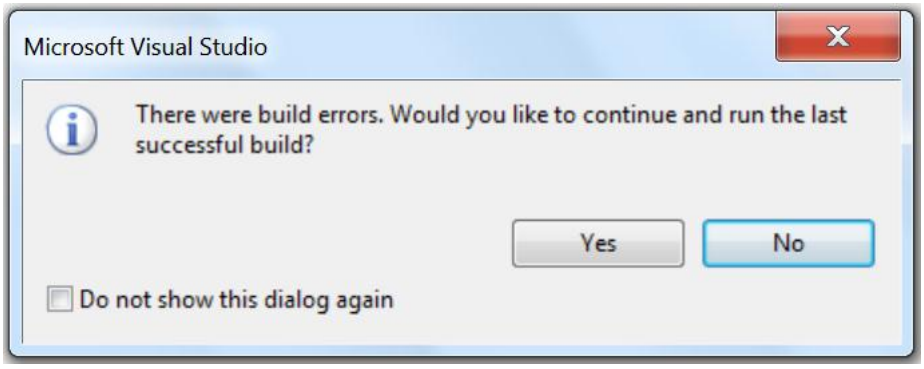

Рисунок 8 – Запрос на продолжение запуска программы при наличии ошибок

При отрицательном ответе на запрос среда высвечивает сообщения об ошибках в специальном окне **Output/Build**, которое появляется в нижней части окна текстового редактора. При положительном – пытается достроить программу с ошибками, что в большинстве случаев обречено на провал.

Для перехода на строку, при обработке которой зафиксирована ошибка, необходимо дважды щелкнуть мышью по соответствующему сообщению. После исправления ошибок программу необходимо вновь скомпилировать и скомпоновать.

Если компиляция и компоновка программы прошли без ошибок, то на экране появляется функционирующее в режиме консоли окно результатов, в которое выводятся сообщения программы и «эхо» вводимых символов (см. рисунок 9). Окно результата закрывается по нажатию любой клавиши или кнопки закрытия окна «×» в правом верхнем углу.

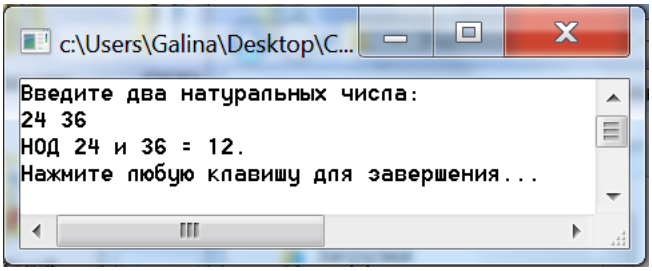

Рисунок 9 – Окно результата

Если ошибка обнаружена уже при выполнении программы, а именно:

- программа выдает сообщение об ошибке выполнения,
- результаты программы отличны от ожидаемых,
- программа «зациклилась»,

то для поиска ошибки следует использовать средства отладки среды Visual C++ (см. разделы 5 и 6).

При зацикливании программы ее выполнение можно завершить, используя кнопку закрытия окна «×» в правом верхнем углу окна результата.

4 Модульное программирование. Файлы заголовков

Среда Microsoft Visual Studio позволяет создавать и отлаживать программы, использующие не только стандартные, но и пользовательские библиотеки (модули).

Модуль языка C++ в отличие от модуля языка Pascal обычно включает два файла: заголовочный файл с расширением `.h` и файл реализации с расширением `.cpp`.

Заголовочный файл играет роль своеобразной интерфейсной секции модуля. В него помещают объявление экспортируемых ресурсов модуля: прототипы (заголовки)

процедур и функций, объявления переменных, типов и констант. Этот файл подключают директивой **`#include "<Имя модуля>.h"`** к файлу реализации модуля, программы или другого модуля, если они используют ресурсы описываемого модуля.

Файл реализации представляет собой аналог секции реализации модуля Pascal. Он должен содержать директивы подключения заголовочных файлов используемых модулей, описания экспортируемых процедур и функций, а также объявления и описания внутренних ресурсов модуля. Если используется проект с прекомпилируемым заголовком, то в начало каждого файла реализации необходимо поместить оператор подключения заголовочного файла **stdafx.h**: **`#include "stdafx.h"`**. Этот файл осуществляет подсоединение библиотек среды, и при его отсутствии компилятор выдает ошибку «Не найден конец файла».

Для создания файлов заголовка и реализации модуля и *добавления их к проекту* используют пункт меню **Project/Add New Item**. Выполнение этого пункта вызовет открытие окна добавления файлов различных типов к проекту (см. рисунок 10).

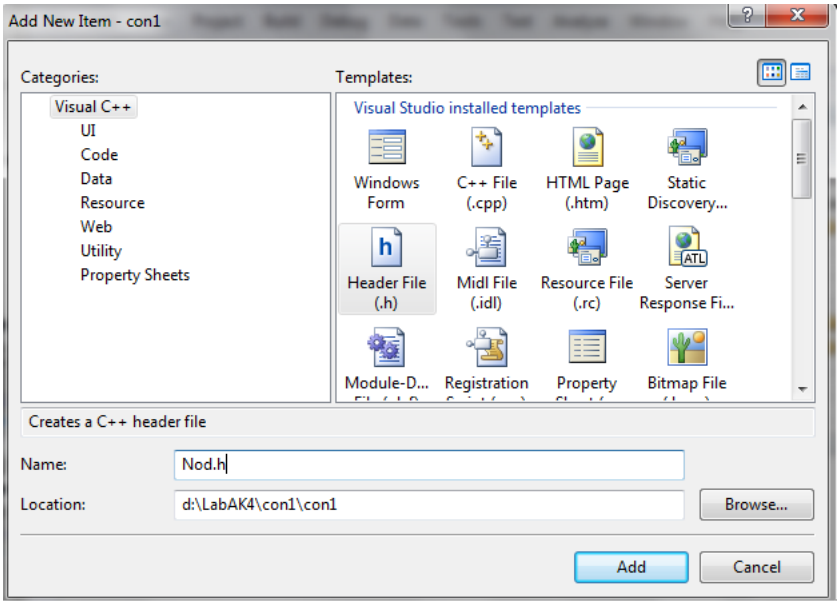

Рисунок 10 – Окно Add New Item

В этом окне необходимо выбрать и тип подключаемого файла и ввести его имя. По данной схеме создадим два файла, образующих модуль:

- файл заголовка Nod.h – **Header File (.h)**;
- файл реализации Nod.cpp – **C++ File (.cpp)**.

Если используется проект с прекомпилированным файлом, то в начало файла Nod.cpp добавим строку подключения файла `stdafx.h`**:**

`#include "stdafx.h"` .

Затем перенесем в этот файл текст функции **nod**, вырезав его из файла реализации программы Con1.cpp, и добавим строку подключения заголовочного файла модуля. В результате файл Nod.cpp должен содержать:

```
#include "stdafx.h"     // только для проекта с заготовкой !!!!
#include "Nod.h"
int nod(int x, int y)
{
     while (x!=y)
          if (x>y) x=x-y;
          else y=y-x;
          return y;
}
```

В заголовочном файле **Nod.h** объявим внешний ресурс – функцию `nod()`**:**

```
int nod(int x, int y);
```

Вместо функции **nod()** в файл основной программы **con1.cpp** добавим ссылку на заголовочный файл **Nod.h**:

`#include "Nod.h"`

Для навигации по многофайловому проекту используют вкладку **Solution** навигатора, на которой высвечен список всех исходных файлов проекта.

Примечание. Модуль не всегда включает два файла. Возможно создание модулей, состоящих из одного файла заголовка или файла реализации. Если файл реализации пуст, то он не создается. В программе это никак не отражается. Если не используется файл заголовка, то в проекте или других модулях, обращающихся к ресурсам этого модуля, объявленным в файле реализации, указывается непосредственно подключение файла реализации. Однако в последнем случае нарушается принцип инкапсуляции модулей, что *нетехнологично*.

Если необходимо подключить модуль, файлы которого размещены в других каталогах и не могут быть скопированы в каталог текущего проекта, то в операторе `include` указывают полное имя файла, например:

```
#include "C:\primer.cpp\con2\filemod.h"
```

Файл реализации модуля с расширением `.cpp` при этом обязательно должен быть скопирован в папку проекта.

Для удаления файла *из проекта* необходимо выделить этот файл на вкладке **Solution Explorer** навигатора и нажать на клавиатуре **Delete**.

5 Отладка консольных приложений

Для входа в режим пошагового выполнения используют команды **Build/Step Info (F11)** или **Build/Step Over (F10)** (см. рисунок 11).

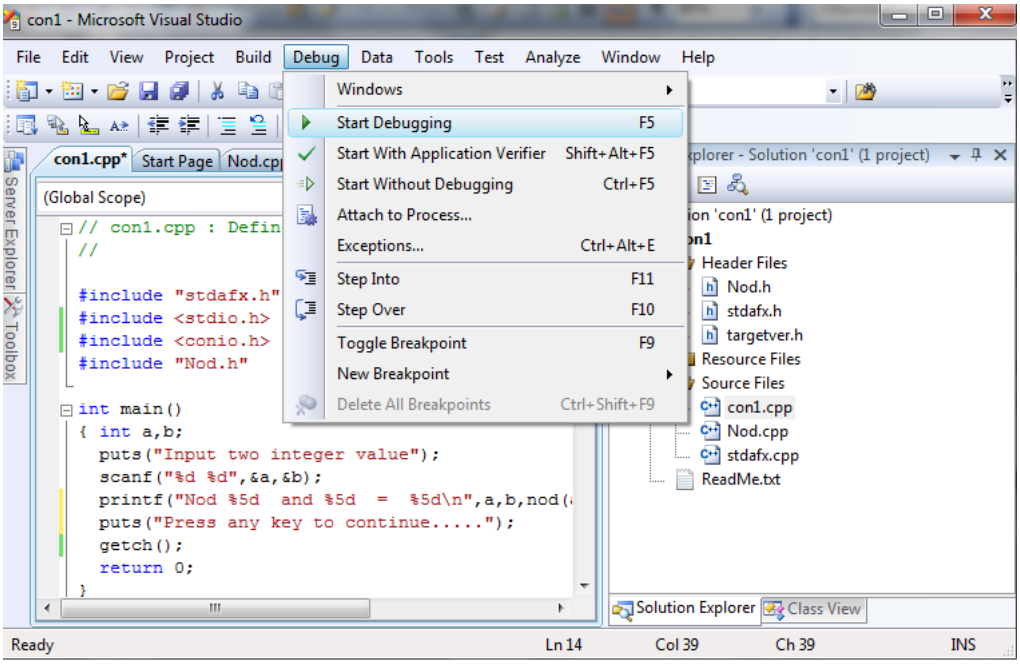

Рисунок 11 – Подменю отладки

После запуска программы в режиме отладки на экране появляется окно с текстом основной программы, в котором следующий выполняемый оператор отмечен специальной стрелкой слева (см. рисунок 12).

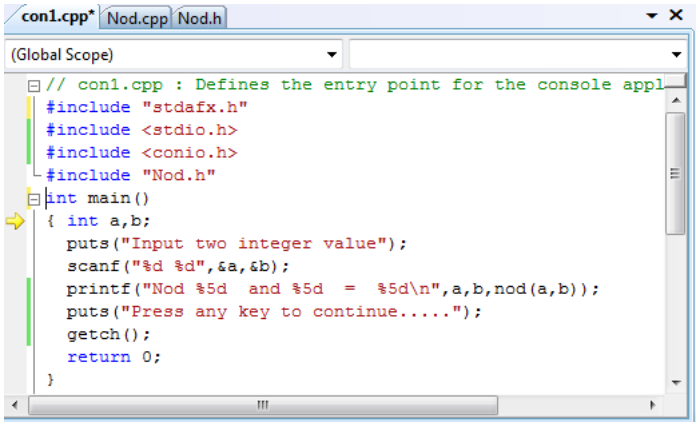

Рисунок 12 – Окно основной программы при запуске в режиме отладки

При входе в режим отладки содержимое пункта меню **Debug** изменяется (см. рисунок 13). В нем появляются следующие основные команды управления режимом:

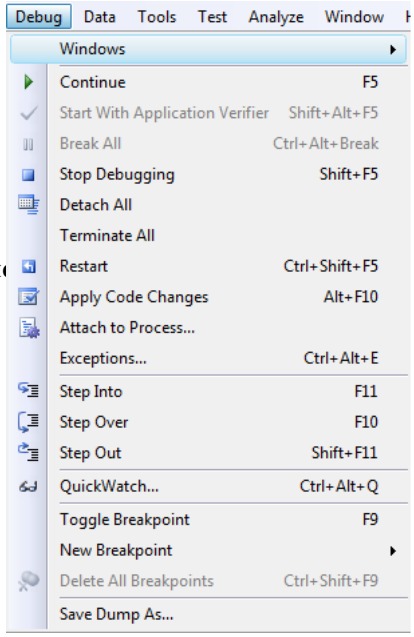

Рисунок 13 – Подпункты меню в режиме отладки

Stop Debugging (Shift+F5) – завершить отладку;

Restart (Ctrl+Shift+F5) – начать отладку с начала;

Step Info (F11) – выполнить операторы текущей строки – если на данном шаге осуществляется вызов процедуры или функции, то зайти в них (не использовать для процедур и функций стандартных библиотек!);

Step Over (F10) – выполнить операторы текущей строки, если на данном шаге осуществляется вызов процедуры или функции, то не заходить в них;

Step Out (Shift+F10) – выполнить процедуру или функцию до конца и вернуться в вызывающую процедуру или функцию.

После выхода из режима отладки меню среды принимает исходный вид.

Отладку программы можно выполнять в пошаговом режиме, просматривая изменяющиеся значения переменных, или устанавливая точки останова в ключевых точках программы.

В процессе пошагового выполнения многофайловых проектов переключение между файлами, содержащими исходные тексты программы и модулей, происходит автоматически.

6 Просмотр значений переменных в режиме отладки

Для просмотра значений переменных в процессе отладки открывают вкладки окна **Output**, появляющегося в нижней части окна среды при запуске программы на выполнение: вкладка Переменные **Locals,** вкладка Наблюдение **Watch 1** и вкладки текущих значений **Autos**.

На вкладке **Locals** (рисунок 14) высвечиваются значения всех переменных выполняемой функции. По мере выполнения операторов программы значения этих переменных меняются. Однако таких переменных в сложных функциях много, отслеживать их трудно, поэтому обычно используют вкладку **Watch 1**.

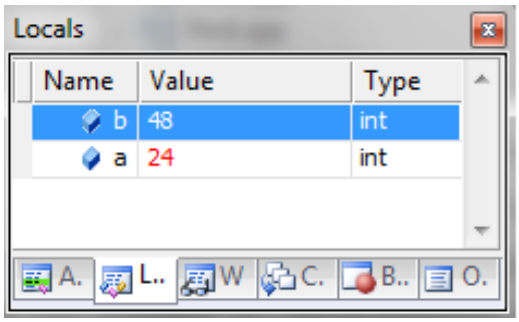

Рисунок 4 – Окно Locals

На вкладке **Watch 1** высвечиваются значения только переменных, указанных программистом (рисунок 15). Добавить переменную в окно **Watch 1** можно, введя ее имя в поле **Name**. Также непосредственно в окне, можно удалить идентификатор отслеживаемой переменной.

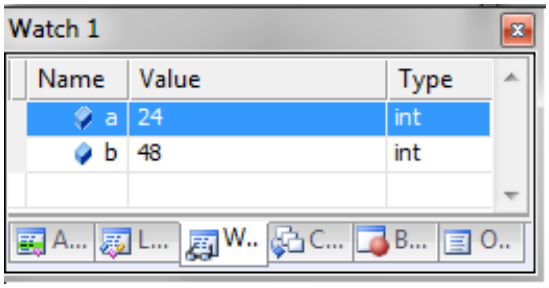

Рисунок 5 – Окно Watch 1

Очень удобно при пошаговом выполнении также окно **Autos** (рисунок 16). В нем высвечиваются значения переменных текущего и следующего оператора программы. Особенно это удобно, если отлаживаемый фрагмент сложен и использует большое количество переменных.

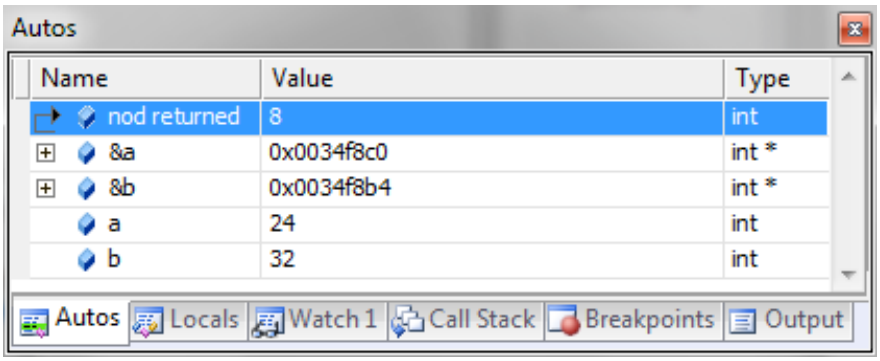

Рисунок 16 – Окно Autos

7 Установка и отмена точек останова в программе

Часто при отладке программу целесообразно выполнять пошагово не с самого начала, а с некоторого места, подозрительного с точки зрения наличия ошибки. Для останова программы в этом месте используют точки останова.

Задать точку останова в программе проще всего, щелкнув мышкой по серому полю текстового редактора перед интересующим нас оператором. Это можно сделать в процессе пошагового выполнения программы или перед ее запуском. При таком щелчке на сером поле появляется красный кружок, отмечающий точки останова (рисунок 17).

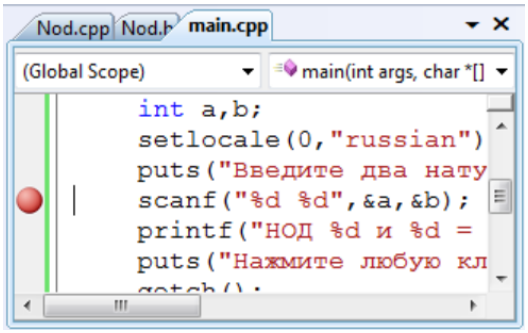

Рисунок 17 – Отметка точки останова в программе

Теперь, если программу запустить на выполнение, то она автоматически остановится перед выполнением отмеченного оператора.