**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ,
МОЛОДЕЖИ И СПОРТА УКРАИНИ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
“ХАРКОВСЬКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ”**

ГМКГ

Методические указания к выполнению лабораторных работ средствами пакета visual studio (Принципы создания программ на языке С++)

Харьков НТУ “ХПИ” 2013

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ,
МОЛОДЕЖИ И СПОРТА УКРАИНИ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
“ХАРКОВСЬКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ”**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ СРЕДСТВАМИ ПАКЕТА VISUAL STUDIO ПО КУРСУ ИНФОРМАТИКА

(ПРИНЦИПЫ СОЗДАНИЯ ПРОГРАММ НА ЯЗЫКЕ С++)

Затверджено
редакційно-видавничою
радою університету,
протокол № 2 від 07.12.2011

Харьков
НТУ «ХПИ»
2013

Методические указания к выполнению лабораторных работ средствами пакета visual studio по курсу «Информатика» (Принципы создания программ на языке С++) / О.Г. Симонова, Д.В. Воронцова. – Х.: НТУ „ХПИ”, 2012. – 30 с.

Составители: О.Г. Симонова
Д.В. Воронцова

Рецензент: . В.В. Таряник

Кафедра геометрического моделирования и компьютерной графики

ВСТУПЛЕНИЕ

В 1972 году 31-летний специалист по системному программированию из фирмы Bell Labs Деннис Ритчи разработал язык программирования Си.

Первое описание языка было дано в книге Б. Кернигана и Д. Ритчи, которая была переведена на русский язык. Долгое время это описание являлось стандартом, однако ряд моментов допускали неоднозначное толкование, которое породило множество трактовок языка С. Для исправления этой ситуации при Американском национальном институте стандартов (ANSI) был образован комитет по стандартизации языка С 1983 г. был утвержден стандарт языка С, получивший название ANSI C. В начале 80-х годов в той же Bell Laboratory Бьерном Страуструпом в результате дополнения и расширения языка С был создан, по сути, новый язык, получивший название "С с классами". В настоящее время С++ считается господствующим языком программирования программных продуктов, в том числе и программного обеспечения компьютерной графики. С++ является очень мощным объектно-ориентированным языком. В последнее время внимание программистов привлек язык Java и C#. Эти языки очень схожи с С++. Поэтому, освоив язык программирования С++, при необходимости, можно легко овладеть и языками Java и C#.

Язык С++ стандартизован Аккредитованным комитетом стандартов под эгидой Американского национального института стандартов (ANSI). Стандарт ANSI на язык С++ обеспечивает аппаратную независимость создаваемых программных продуктов, то есть программы, написанные на языке С++ в соответствии со стандартом ANSI будут без проблем компилироваться на компьютерах, работающих на различных платформах, например, Mac, Windows, UNIX и других. В данном курсе для практических и лабораторных занятий будет использована операционная система Windows, а в качестве среды разработки и компилятора – Microsoft Visual C++ .

Цель работы: приобрести навыки создания программ на языке С++

1. Теоретические сведения

Типичная среда разработки C++

Программы на C++ обычно проходят через шесть стадий: *редактирования*, *препроцессорной обработки, компиляции, компоновки, загрузки* и *исполнения.*

Стадия 1: Создание программы

С помощью *программы*-редактора вы вводите программу на C++ (*исходный код),* вносите необходимые исправления и сохраняете программу на жестком диске. Файлы исходного кода C++ имеют расширения .cpp. Среда Visual Studio имеет встроенный редактор.

Стадии 2 и 3: Препроцессорная обработка и компиляция программы

На второй стадии программист дает команду *компилировать* программу. В среде C++ - перед началом стадии компиляции автоматически исполняется *программа-препроцессор* (поэтому мы называем препроцессорную обработку стадией 2, а компиляцию стадией 3). Препроцессор C++ распознает команды, называемые *препроцессорными директивами,* которые указывают, что над программой перед компиляцией должны быть произведены определенные манипуляции. Эти манипуляции состоят обычно во включении в компиляцию заголовочных* и других текстовых файлов, а также в различных текстовых заменах. На 3-й стадии компилятор транслирует программу на C++ в код машинного языка называемый также *объектным кодом.*

Стадия 4: Компоновка

Четвертая стадия называется *компоновкой* (linking). Программы C++ обычно содержат ссылки на функции и данные, определяемые в другом месте, например, в стандартных библиотеках или в частных библиотеках группы программистов, работающих над конкретным проектом. Из-за отсутствия этих частей в программах имеются «дыры». *Компоновщик* (linker) присоединяет к объектному коду код отсутствующих функций, чтобы создать *исполняемый образ* (в котором нет отсутствующих частей). Если программа успешно компилируется и компонуется, образуется исполняемый образ.

---

* **Заголовочный файл** (или подключаемый файл) – файл, содержащий определения типов данных, структуры, прототипы функций, перечисления, макросы препроцессора. Имеет по умолчанию расширение .h. Заголовочный файл используется путём включения его текста в данный файл директивой препроцессора #include.

Стадия 5: Загрузка

Пятая стадия называется *загрузкой.* До того как программа сможет исполнять, ее нужно поместить в память. Это выполняется *загрузчиком,* который берет с диска исполняемый образ и переносит его в память. Загружаются также компоненты разделяемых библиотек, поддерживающие данную программу.

Стадия 6: Исполнение

Наконец, компьютер под управлением процессора *исполняет* программу одиночными инструкциями.

Среда разработки Visual Studio

Среда разработки Visual Studio позволяет разрабатывать и отлаживать программы, написанные на языках программирования C++, C#, Visual Basic .NET, F#, J# и других.

Проекты

Разрабатываемая программа представляется в Visual Studio в виде проекта, содержащего файлы исходных кодов на языке программирования, файлы ресурсов, например, изображений и различных вспомогательных файлов. Перечень файлов, входящих в проект, а также свойства проекта хранятся в файле проекта. Расширение файла проекта зависит от используемого языка программирования. Для проектов, написанных на языках программирования C и C++, это файл с расширением .vcproj.

Сборка проекта

Сборка проекта – процесс преобразования исходных файлов проекта в некоторое конечное представление, например, исполняемый файл (.EXE), динамическую (.DLL) или статическую (.LIB) библиотеку. В процессе сборки, как правило, выполняется компиляция***** файлов исходных кодов для получения объектных файлов 2 и их последующая компоновка 3 в приложение или библиотечный модуль.

Решение

Среда Visual Studio позволяет использовать несколько проектов совместно в составе решения (англ. Solution), что облегчает разработку целых программных комплексов, состоящих из нескольких программных модулей, использующихся совместно.

***** **Компиляция** — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера).

Между проектами, входящими в состав одного решения, можно установить зависимости, влияющие на последовательность сборки проектов решения, а также на использование одних проектов в составе других. Таким образом, становится возможным вынесение совместно используемого кода нескольких приложений в отдельный проект (например, статическую или динамическую библиотеку*) и установка зависимостей проектов приложений от проектов используемой ими библиотеки. При сборке приложения сначала будут собраны проекты используемых данным приложением библиотек, а затем будет собрано само приложение.

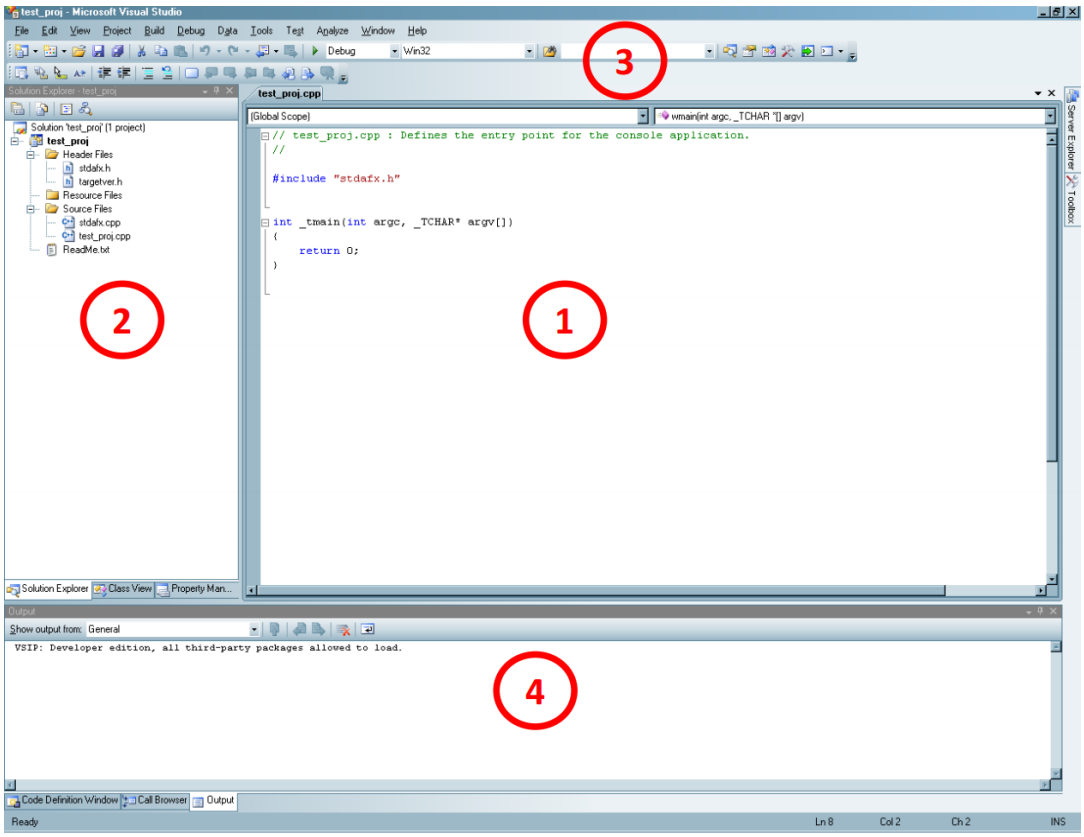

Рисунок 1 – Основное окно среды разработки

Основное окно среды разработки (рис. 1) :

- 1- окно редактирования исходного кода
- 2 - Solution Explorer (и другие панели)
- 3 - Панели инструментов, меню
- 4 - Панели вывода информации

* **Динамическая библиотека** - часть основной программы, которая загружается в ОС по запросу работающей программы в ходе её выполнения (Run-time), то есть динамически (Dynamic Link Library, DLL в Windows, SO в Linux).

* **Статическая библиотека** - исходный текст, подключаемый программистом к своей программе на этапе написания (например, для языка Fortran существует огромное количество библиотек для решения разных задач именно в исходных текстах)

Создание нового проекта

Меню ***File*** → ***New Project*** открывает мастер создания нового проекта. Для каждого используемого языка программирования доступен свой набор проектов (рис. 2).

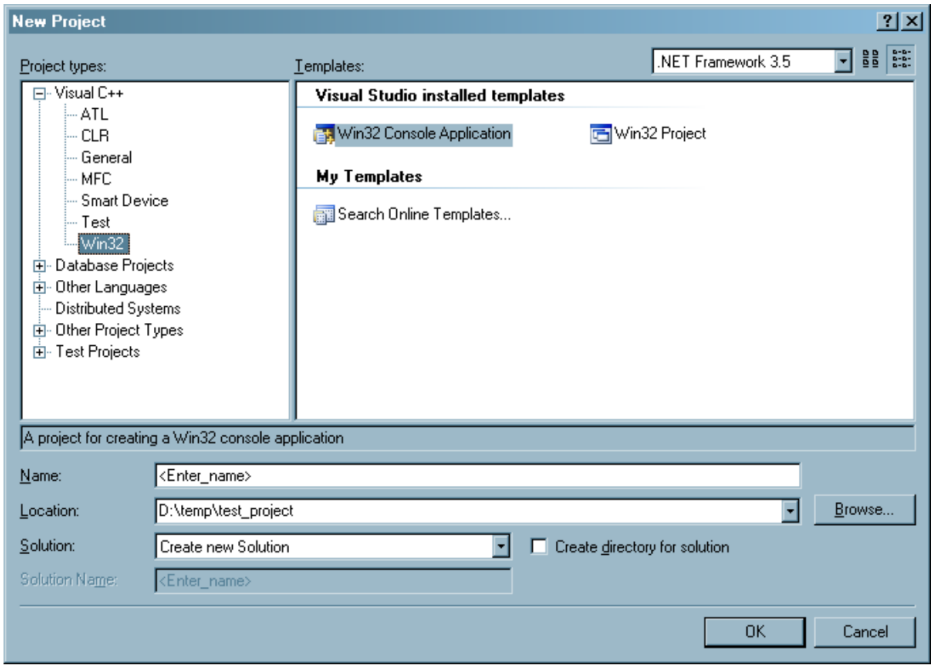

Рисунок 2 – Набор проектов

Редактирование свойств проекта

Окно редактирования свойств проекта вызывается при помощи выбора пункта Properties контекстного меню выбранного проекта в окне Solution Explorer (рис.3).

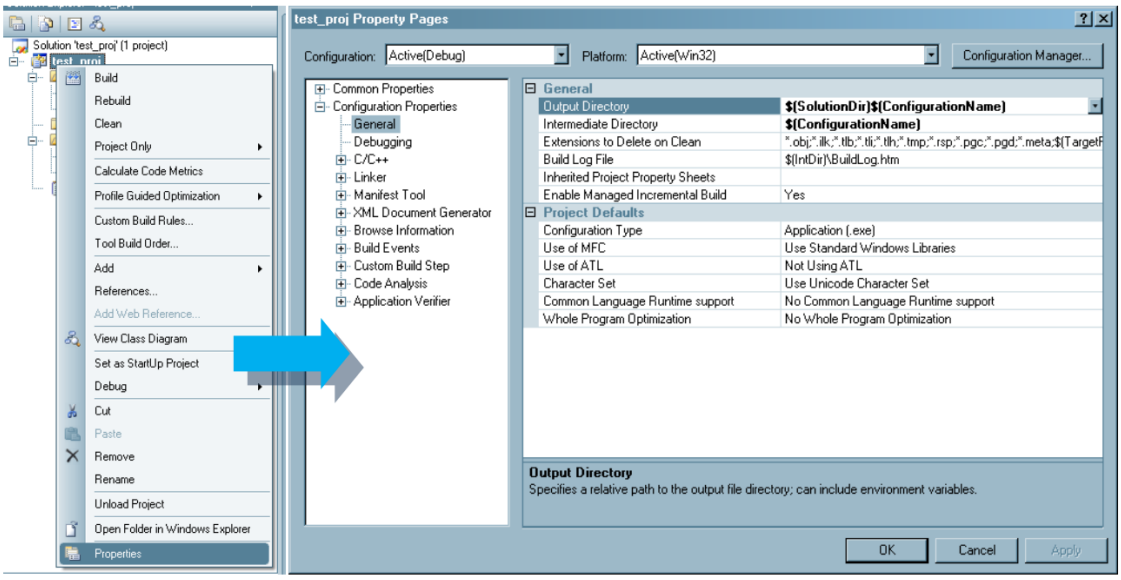

Рисунок 3 – Меню выбранного проекта в окне Solution Explorer

Установление зависимостей между проектами

При помощи меню Project Dependencies возможно установить зависимости одного проекта от других (рис.4). Данные зависимости будут учитываться при определении порядка сборки проектов в решении.

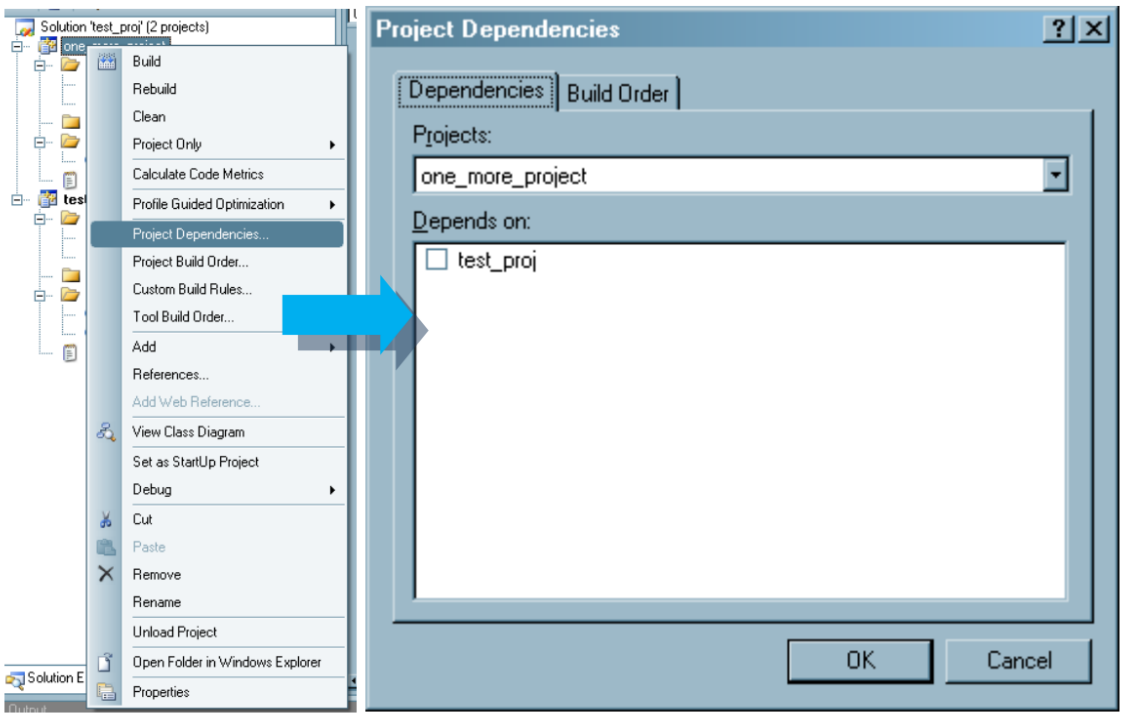

Рисунок 4 – Зависимость одного проекта от другого

Редактирование исходного кода

Встроенный редактор исходного кода Visual Studio предоставляет возможности по автоматическому завершению вводимых операторов, названий переменных и функций. Существуют также коммерческие плагины, расширяющие возможности стандартного редактора исходного кода. Один из лучших – Visual Assist компании Whole Tomato.

Компиляция и компоновка проекта

Собрать проекты в составе одного решения можно при помощи клавиши F7. Откомпилировать открытый файл исходного кода можно при помощи комбинации клавиш Ctrl+F7.

Конфигурации проекта

Конфигурация проекта – именованный набор настроек проекта. Изначально проект содержит 2 конфигурации:

• ***Debug*** – отладочная конфигурация. В процессе компиляции отключается оптимизация кода, в собираемый модуль помещается дополнительная информация для отладки кода, используются отладочные версии библиотек времени выполнения5. Собранный в отладочной конфигурации модуль, как

правило, обладает большим размером, и меньшим быстродействием, однако наиболее полно обеспечивает возможности отладки на уровне исходного кода.

• ***Release*** – финальная конфигурация. При компиляции в ***Release***-конфигурации включена оптимизация кода по быстродействию или размеру, и, как правило, исключается отладочная информация из собираемого модуля. Собранный в финальной конфигурации модуль зачастую обладает меньшим размером и большим быстродействием, однако возможность отладки на уровне исходного кода значительно затруднена.

Отладочная конфигурация должна использоваться только в процессе отладки приложения. Для предоставления конечному пользователю, напротив, следует использовать ***Release***-конфигурацию. Помимо низкого быстродействия и часто большего размера, приложение, собранное в отладочной конфигурации, может требовать для своей работы наличия отладочной версии библиотеки времени выполнения, которые не поставляются вместе с операционной системой.

Помимо стандартных конфигураций программист может задать дополнительные конфигурации для проектов, например, оптимизированные для уменьшения размера, или уменьшения зависимостей от внешних ***runtime***-библиотек. Пользователь может выбрать одну из доступных конфигураций при помощи выпадающего списка на панели инструментов. Во время следующей сборки проекты будут скомпилированы и скомпонованы в указанной конфигурации (рис.5).

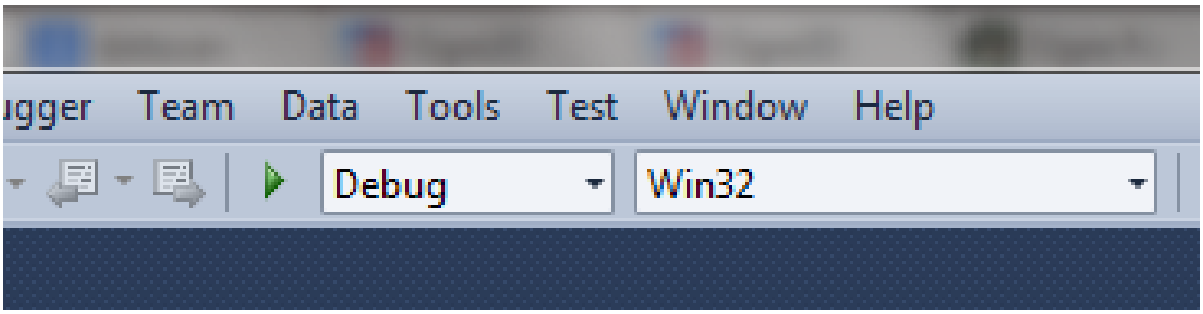

Рисунок 5 – Пример конфигурации ***Debug***

Для пакетной сборки проектов сразу в нескольких конфигурациях можно воспользоваться командой меню ***Build→Batch Build***.

Платформа проекта

Операционные системы семейства Windows выпускаются в редакциях для 32-битных (x86) и 64-битных (x64) процессоров. Под управлением 64-битных версий операционных систем Windows могут выполняться 64-битные приложения, которым доступен больший объем оперативной памяти, способные выполнять операции над 64-битными целыми числами быстрее, чем 32-битные приложения.

Среда Visual Studio позволяет создавать как 32-битные, так и 64-битные приложения (при установке соответствующих версий компилятора). При сборке

приложения можно выбрать платформу (32-битную или 64-битную, если были установлены соответствующие компоненты), для которой будет осуществляться сборка указанной конфигурации проекта (рис.6).

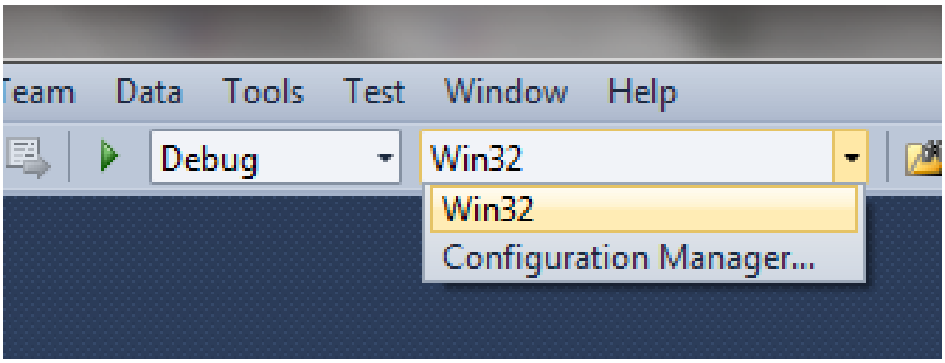

Рисунок 6 – Создание 32-битного приложения

Запуск приложения

Комбинация клавиш Ctrl+F5 позволяет запустить собранный проект (в выбранной конфигурации для указанной платформы) прямо из среды разработки (рис.7). В случае, когда решение содержит несколько проектов, один из проектов может быть помечен как запускаемый при помощи контекстного меню окна Solutions Explorer.

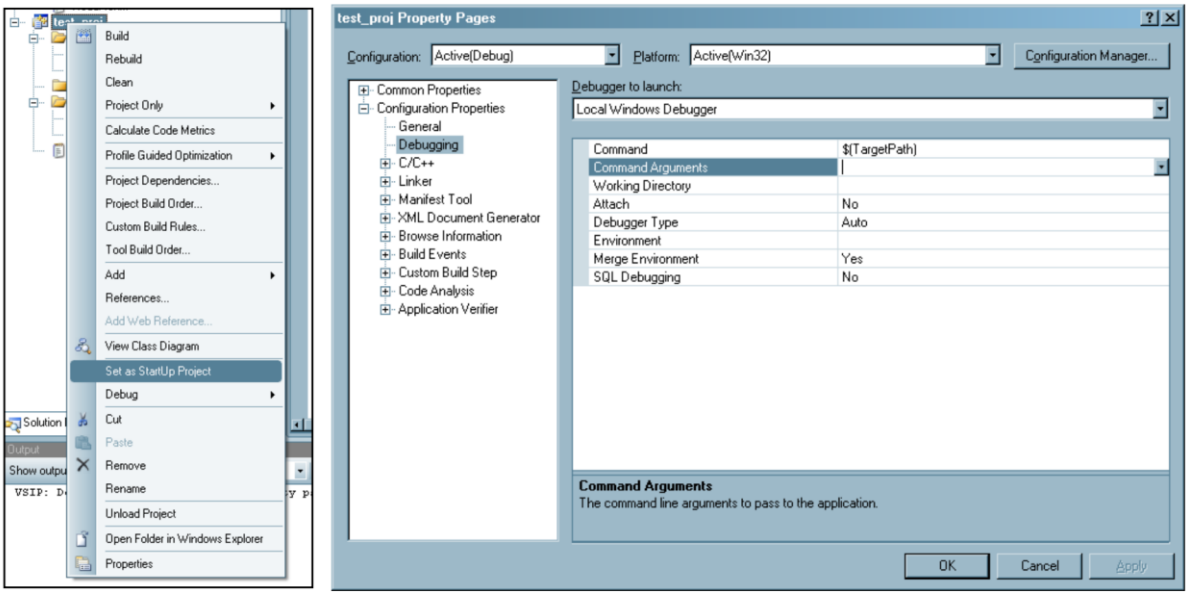

Рисунок 7 – Запуск приложения

В окне настройки проекта пользователь может указать аргументы командной строки, передаваемые приложению при его запуске, директорию по умолчанию, а также дополнительные динамически подключаемые библиотеки, загружаемые в адресное пространство приложения при его запуске.

Отладка приложения

Клавиша F5 осуществляет запуск активного проекта под управлением встроенного отладчика среды Visual Studio. В режиме отладки программисту доступны следующие возможности отладки приложений:

- Остановка выполнения программы в указанных точках останова
- Трассировка программы
- Просмотр и изменение значений переменных и содержимого ячеек памяти

Точки останова

Клавиша F9 позволяет установить или снять ранее установленную точку останова со строки, на которую установлен курсор в редакторе исходного кода. Строка программы, на которой установлена точка останова, помечается кружком (рис.8).

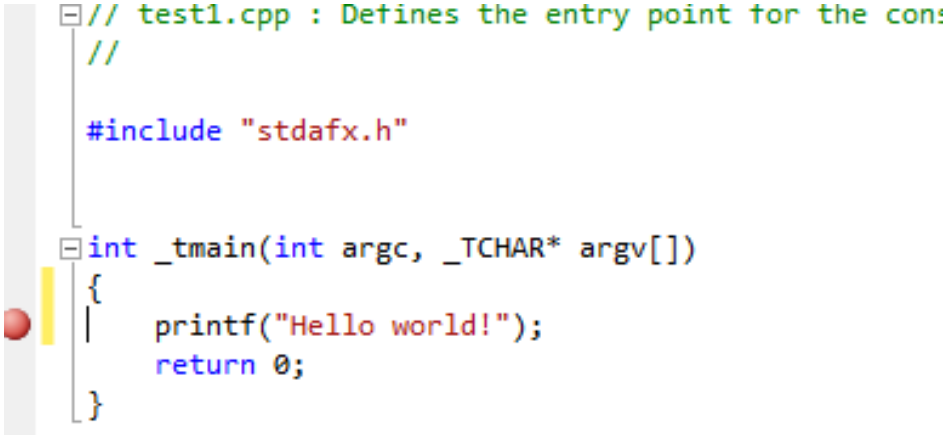

Рисунок 8 – Пример точки останова

Программист может установить произвольное количество точек останова в программе.

> **Внимание!** В момент запуска приложения в режиме отладки положение точек останова может измениться, например, если точка останова была установлена на строках, не содержащих операторов, или внутри комментариев.

Выполнение программы приостанавливается, как только выполнение доходит строки, помеченной точкой останова.

Просмотр и изменение значений переменных в процессе отладки

В процессе отладки приостановленного приложения программисту доступны окна просмотра локальных переменных (Local variables) и окна просмотра выражений (Watch). Окна отладчика показываются при помощи команд меню Debug->Windows (рис.9).

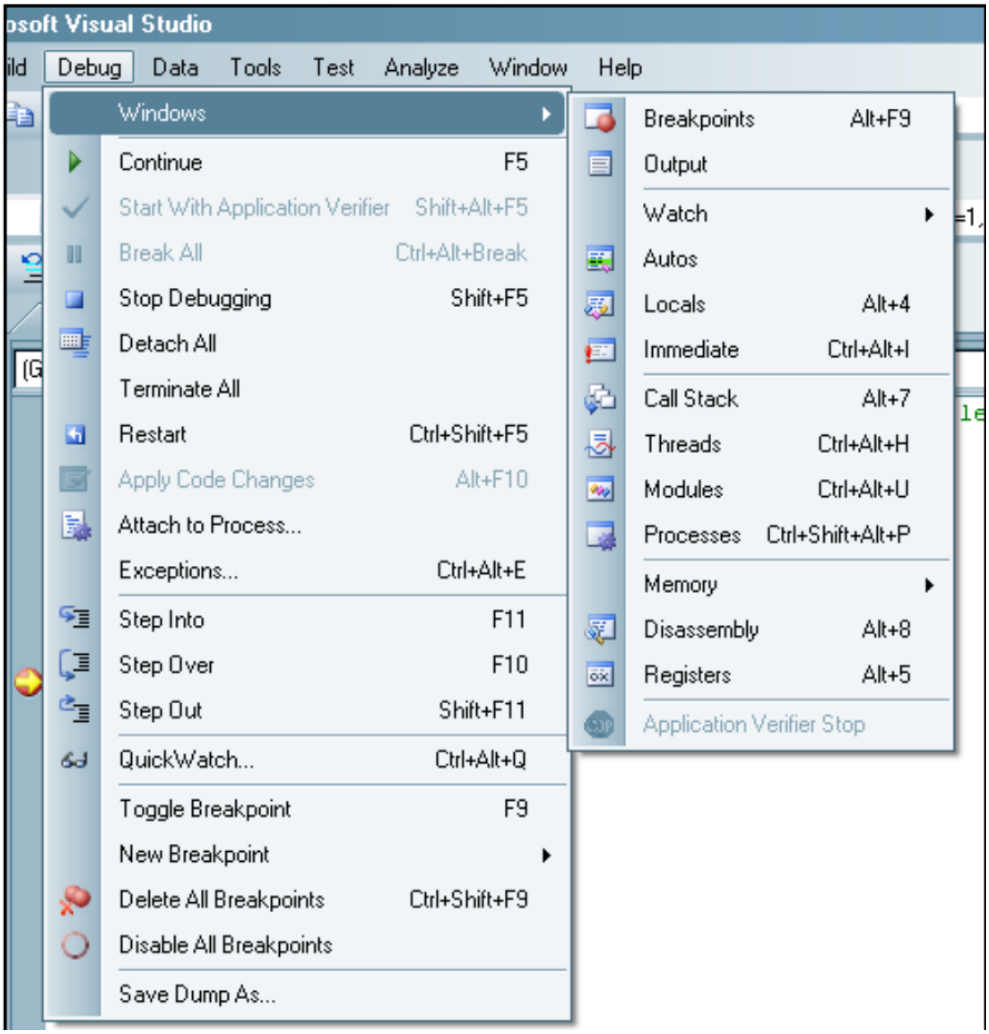

Рисунок 9 – Меню окна отладчика

Для просмотра переменных доступны следующие окна:

• ***Autos*** – отображение переменных, использующихся текущим и предыдущим операторами.

• ***Locals*** – отображение локальных переменных текущей функции или метода класса

В окне ***Watch*** выражений программист может просматривать значения переменных и выражений, а также изменять значения переменных (но не значения выражений).

Просмотр и изменение содержимого ячеек памяти

Помимо изменения значения отдельных переменных программисту также доступно содержимое адресного пространства отлаживаемого процесса. При помощи окна ***Memory*** программист может просматривать и изменять содержимое ячеек памяти. Значения ячеек памяти могут отображаться в виде 8- , 16-, и 32-битных значений (в шестнадцатеричной системе счисления). Окно ***Memory*** вызывается при помощи команды меню ***Debug->Memory***.

Пошаговое выполнение (трассировка) программы

Трассировка позволяет программисту проследить выполнение программы «по шагам», на каждом шаге отслеживая значения переменных и/или содержимое ячеек памяти, а также порядок выполнения операторов.

Следующие комбинации клавиш используются для пошаговой тассировки программы:

• ***F10*** – выполнить текущий оператор. Если текущим оператором является оператор вызова функции или метода класса, то он выполняется как единое целое.

• ***F11*** – выполнить текущий оператор. Если текущим оператором является оператор вызова функции, то происходит вход внутрь тела функции

• ***Shift + F11*** – продолжить выполнение программы до выхода из текущей функции или метода класса.

• ***Ctrl + F10*** – продолжить выполнение программы до достижения позиции курсора

• ***Alt + Num*** – установить курсор в позицию выполнения следующего оператора

• ***F5*** – возобновить выполнение программы.

Внесение изменений в код в процессе отладки программы

Среда ***Visual Studio*** предоставляет механизм ***Edit and Continue***, позволяющий внести в программу необходимые изменения в процессе отладки и в ряде случаев продолжить выполнение измененной программы без ее перезапуска. После внесения изменений достаточно нажать комбинацию клавиш ***Alt + F10***, либо выполнить одну из команд трассировки программы.

Прекращение выполнения приложения

Остановить работу приложения и выйти из режима отладки можно осуществить при помощи комбинации клавиш ***Shift + F5***.

Структура программы

В состав каждой программы на языке C++ должна входить главная функция ***main()***.Именно эта функция является начальной точкой входа в программу.

Кроме функции ***main()*** в программу могут также входить и другие функции. Каждая функция по отношению друг к другу является внешней, т.е. ни одна из функций не может находится в середине другой. Для того, что бы функция была доступна, необходимо, что бы до ее вызова про нее было известно компилятору.

Основную структуру программы на языке С++ приведено на рис.10. Ниже приведен пример1 простой программы на языке С++.

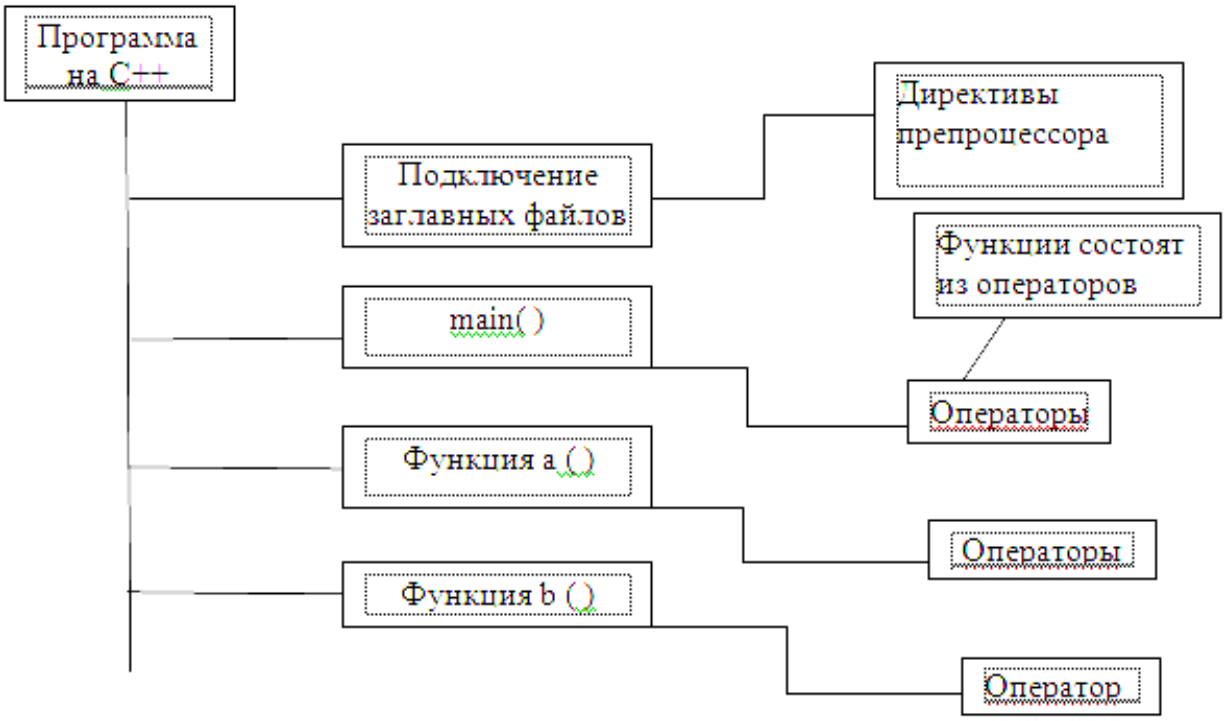

Рисунок 10 – Структура программы на языке С++

Пример 1 Программа вывода на экран «Учимся программировать на С++»

```
#include "stdafx.h"
#include <iostream> // директива препроцесора
using namespace std; // включает в программу определение стнадартного пространства имен
void main(void) // заголовок функции main
{ // начало тела функции main
   cout << "U4imcya programmirovat na C++!\n"; // \n – символ нового радка
} // конец тела функции main
```

Для запуска программы нажмите ***Ctrl+F5***. После этого программа должна выдать окошко (рис.11):

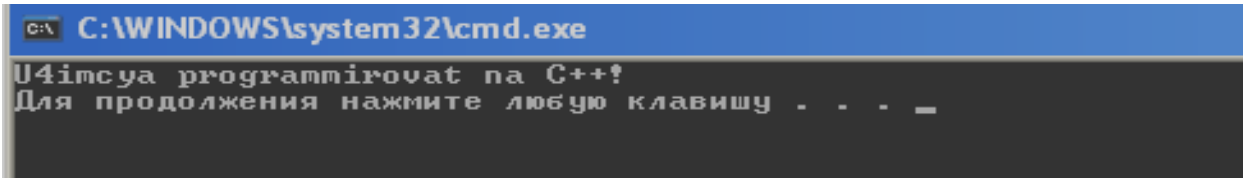

Рисунок 11 – Вывод результата на экран

Переменные и константы

Переменная – область оперативной памяти, обладающая собственным именем и предназначенная для хранения данных, которые могут быть изменены.

Константа – область оперативной памяти, обладающая собственным именем и предназначенная для хранения постоянных данных.

Переменные в C++ используются для временного хранения информации при выполнении какой-либо программы. Хранение этой информации осуществляется в оперативной памяти компьютера. Для выделения места хранения переменных компилятору нужно знать сколько памяти необходимо для их размещения. Поэтому при определении переменной нужно указать тип хранимых данных.

Типы данных

Основные типы переменных:

Тип	**Размер (байт)**	**Диапазон значений**
bool	1	true, false
unsigned short int	2	0..65535
short int	2	–32768...32767
unsigned long int	4	0...4294967295
long int	4	–2147483648...2147483647
int (16 разрядные системы)	2	–32768...32767
int (32 разрядные системы)	4	–2147483648...2147483647
unsigned int (16 разрядные системы)	2	0..65535
unsigned int (32 разрядные системы)	4	0...4294967295
char	1	256 значений символов (a,b,...z ...)
float	4	3.4E ± 38 (7 десятичных знаков)
double	8	1.7E ± 308 (15 десятичных знаков)

Тип ***bool*** – логический. Переменные такого типа могут хранить только идно из двух значений – ***true*** (истина) или ***false*** (ложь). Группа целочисленных типов позволяет создать переменные, хранящие целые значения различного размера. Типы ***float*** и ***double*** относятся к вещественным (с плавающей точкой). Переменные такого типа могут хранить значения дробного характера. Символьный тип данных ***char*** занимает один байт. В область памяти распределенной для переменной типа ***char*** помещается число в диапазоне от 0 до 255, которое условно определяет символ. Так, например, символ "a" латинского алфавита кодируется значением 97, а символ "5" – значением 53.

В стандарте C++ имеются специальные символы, которые предназначены для форматирования текста. Управляющие символы:

Символ	Значение
\a	Звуковой сигнал
\b	Забой
\f	Перевод страницы
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\'	Одиночная кавычка
\"	Двойная кавычка
\0	Нулевой символ (NUL)

Пример 2 Программа, которая выводит размер типов данных в байтах.

```
#include <iostream.h>

void main(void) {
        cout << " (unsigned)int = " << sizeof(int) << endl;
        cout << " (unsigned)short = " << sizeof(short) << endl;
        cout << " (unsigned)char = " << sizeof(char) << endl;
        cout << " (unsigned)float = " << sizeof(float) << endl;
        cout << " (unsigned)double = " << sizeof(double) << endl;
        cout << " (unsigned)long = " << sizeof(long) << endl;
        cout << " (unsigned)long double = " << sizeof(long double) << endl;
        }
```

Чтобы воспользоваться функциями преобразования типов необходимо подключить заголовочный файл **cstdlib**.

```
#include <cstdlib.h>
```

Можно одновременно объявлять несколько переменных одного типа.

```
unsigned int myVar1, myVar2; // объявлены две переменные беззнакового целого типа
```

Пример 4

Программа PRECISE. CPP присваивает значение чуть меньше 0.5 переменным *типа float и double.* К сожалению, поскольку компьютер обладает ограниченной способностью в представлении чисел, переменные реально содержат не присваиваемые им значения, а число 0.5:

```
#include <iostream.h>
void main(void)
{
  float f_not_half = 0.49999990;
  double d_not_half = 0.49999990;
  cout << "Значение типа float 0.49999990 равно " << f_not_half << endl;
  cout << "Значение типа double 0.49999990 равно " << d_not_half << endl;
}
```

Когда вы откомпилируете и запустите эту программу, на вашем экране появится следующий вывод:

```
Значение типа float 0.49999990 равно 0.5
Значение типа double 0.49999990 равно 0.5
```

Объявление определяет имя внутри области видимости. Имя, объявляемое внутри блока, класса, функции или пространства имен, является локальным. Блок в языке C++ заключается в фигурные скобки. Если вне блока существуют глобальные имена, обозначаемые теми же идентификаторами, то они становятся скрыты внутри блока и к ним следует обращаться, используя оператор разрешения области видимости ::.

Пример 4

```
int i1;     // Объявление глобальной переменной
void metod1() {
  int i1;   // Объявление локальной переменной
  i1=22;    // Доступ к локальной переменной метода
  ::i1=44;  // Доступ к глобальной переменной
  { int i1; // Объявление локальной переменной
    i1=33;  // Доступ к локальной переменной блока
  }
i1=44;      // Доступ к глобальной переменной
}
```

Средства ввода/вывода

При запуске программы на языке С++ автоматически создаются сразу несколько стандартных потоков: ***cin*** (стандартный поток ввода с клавиатуры) и ***cout*** (стандартный поток вывода на экран).

Существует еще два стандартных потока ***cer*** и ***clog***, которые предназначены для вывода ошибок. Для того чтобы использовать эти потоки, достаточно подключить заглавный файл ***<iostream>*** и указать стандартное пространство имен, как это показано в примере 2.

Пример 4 Программа позволяет ввести число, а после этого она его выводит на экран.

```
#include <iostream> // директива препроцесора
using namespace std; // включает в программу определение стнадартного пространства имен
int main() // заголовок функции main
{ // начало тела функции main
   int i;// объявление переменной і целого типа
   cout << "Input i:"; // программа выводит на экран "Input i:"
   cin >>i; // ввод с клавиатури значения і
   cout << "i= :" << i; // вівод на єкран значения і
} // конец тела функции main
```

Основные математические операции

Операция	Назначение	Пример
+	Сложение	total = cost + tax;
–	Вычитание	change = payment – total;
*.	Умножение	tax = cost * tax_rate;
/	Деление	average = total / count;

Операции С++, которые вы можете встретить в программах.

Операция	Функция
%	Взятие по модулю или остаток; возвращает остаток целочисленного деления
~	Дополнение; инвертирует биты значений
&	Побитовое И
|	Побитовое включающее ИЛИ
^	Побитовое исключающее ИЛИ
<<	Сдвиг влево; сдвигает биты значения влево
>>	Сдвиг вправо; сдвигает биты значения вправо

Старшинство операций в C++.

Операция	Имя	Пример
:: Разрешение области видимости classname::classmember_name		
::	Глобальное разрешение	::variable_name
.	Выбор элемента	object.member_name
–>	Выбор элемента	pointer->membername
[]	Индексация	pointer[element]
()	Вызов функции	expression(parameters)
()	Построение значения	type(parameters)
sizeof	Размер объекта	sizeof expression
sizeof	Размер типа	sizeof(type)
++	Приращение после	variable++
++	Приращение до	++variable
– –	Уменьшение после	variable– –
– –	Уменьшение до	– –variable
&	Адрес объекта	&variable
*	Разыменование	*pointer
new	Создание (размещение)	new type
delete	Уничтожение (освобождение) delete pointer	
delete[]	Уничтожение массива	delete pointer
~	Дополнение	~expression
!	Логическое НЕ	! expression
+	Унарный плюс	+1
–	Унарный минус	–1
()	Приведение	(type) expression
.*	Выбор элемента	object.*pointer
–>	Выбор элемента	object–>*pointer
*	Умножение	expression * expression
/	Деление	expression / expression
%	Взятие по модулю	expression % expression
+	Сложение (плюс)	expression + expression
–	Вычитание (минус)	expression expression

Увеличение переменной на один

Обычной операцией, которую вы будете выполнять при программировании, является прибавление 1 к значению целой переменной. Например, предположим, что ваша программа использует переменную с именем ***count***, чтобы сохранить данные о количестве напечатанных файлов. Каждый раз, когда программа печатает файл, 1 будет добавляться к текущему значению ***count***. Используя оператор присваивания C++, ваша программа может увеличивать значение ***count***, как показано ниже:

```
count = count + 1;
```

В данном случае программа сначала выбирает значение ***count***, а затем добавляет к нему единицу. Далее программа записывает результат сложения обратно в переменную ***count***. Следующая программа INTCOUNT.CPP использует оператор присваивания для увеличения переменной ***count*** (которая первоначально содержит значение 1000) на единицу (присваивая переменной результат 1001):

```
#include <iostream.h>
void main(void)
{
  int count = 1000;
  cout << "начальное значение count равно" << count << endl;
  count = count + 1;
  cout << "конечное значение count равно" << count << endl;
}
```

Когда вы откомпилируете и запустите эту программу, на вашем экране появится следующий вывод:

```
начальное значение count равно 1000
конечное значение count равно 1001
```

Так как увеличение значения переменной представляет собой обычную операцию в программах, в C++ есть *операция увеличения* — двойной знак плюс (++). Операция *увеличения* обеспечивает быстрый способ прибавления единицы к значению переменной. Следующие операторы, например, увеличивают значение переменной *count на* 1:

```
count = count + 1; count++;
```

Следующая программа INC_OP.CPP использует операцию увеличения для наращивания значения переменной ***count*** на 1:

```
#include <iostream.h>
void main(void)
```

```
{
  int count = 1000;
  cout << "начальное значение count равно " << count << endl;
  count++;
  cout << "конечное значение count равно " << count << endl;
}
```

Эта программа работает так же, как INCCOUNT.CPP, которая использовала оператор присваивания для увеличения значения переменной. Когда C++ встречает операцию увеличения, он сначала выбирает значение переменной, добавляет к этому значению единицу, а затем записывает результат обратно в переменную.

Представление о префиксной (до) и постфиксной (после) операциях увеличения

При использовании операций увеличения ваши программы могут размещать оператор увеличения до или после переменной, как показано ниже:

```
++variable; variable++;
```

Так как первый оператор появляется до переменной, он называется *префиксным оператором увеличения.* Аналогично этому второй оператор появляется после переменной и называется *постфиксным оператором увеличения.* Вам необходимо знать, что C++ трактует эти два оператора по-разному. Рассмотрим, например, следующий оператор присваивания:

```
current_count = count++;
```

Этот оператор присваивания указывает C++ присвоить текущее значение *count* переменной *current_count.* В дополнение к этому постфиксный оператор увеличения заставляет C++ увеличить текущее значение *count.* Использование постфиксного оператора в этом случае делает показанный выше оператор эквивалентным следующим двум операторам:

```
current_count = count;
count = count + 1;
```

Теперь рассмотрим следующий оператор присваивания, который использует префиксный оператор увеличения:

```
current_count = ++count;
```

В этом случае оператор присваивания указывает C++ сначала увеличить значение ***count***, а затем присвоить результат переменной ***current_count***. Использование префиксного оператора увеличения делает показанный выше оператор эквивалентным следующим двум операторам:

```
count = count + 1;
current_count = count;
```

Важно освоить префиксную и постфиксную операции увеличения, так как они будут встречаться вам в большинстве программ на C++. Следующая программа PRE_POST.CPP иллюстрирует использование префиксной и постфиксной операций увеличения:

```
#include <iostream.h>
void main(void)
{
  int small_count = 0;
  int big_count = 1000;
  cout << "small_count равно " << small_count << endl;
  cout << "small_count++ производит " << small_count++ << endl;
  cout << "конечное значение small_count равно " << sniall_count << endl;
  cout << "big_count равно " << big_count << endl;
  cout << "++big_count производит " << ++big_count << endl;
  cout << "конечное значение big_count равно " << big_count << endl;
}
```

Когда вы откомпилируете и запустите эту программу, на вашем экране появится следующий вывод:

```
small_count равно 0
small_count++ производит 0
конечное значение small_count равно 1
big_count равно 1000
++big_count производит 1001
конечное значение big_count равно 1001
```

С переменной ***small_count*** программа использует постфиксную операцию увеличения. В результате программа выводит текущее значение переменной (0), а затем увеличивает его на 1. С переменной ***big_count*** программа использует префиксную операцию увеличения. В результате программа сначала увеличивает значение переменной (1000 + 1), а затем выводит результат (1001). Найдите время, чтобы отредактировать эту программу, и сначала измените постфиксную операцию на префиксную, а затем префиксную на постфиксную. Откомпилируйте и запустите программу, обращая внимание на то, как изменение операции изменяет вывод.

C++ обеспечивает также операции уменьшения

Как вы уже знаете, двойной знак плюс (++) представляет собой оператор увеличения C++. Подобным образом двойной знак минус (– –) соответствует *оператору уменьшения* C++, который уменьшает значение переменной на 1. Как и в случае с операцией увеличения, C++ поддерживает префиксный и постфиксный операторы уменьшения. Следующая программа DECCOUNT.CPP иллюстрирует использование оператора уменьшения C++:

```
#include <iostream.h>
void main(void)
{
  int small_count = 0;
  int big_count = 1000;
  cout << "small_count равно " << small_count << endl;
  cout << "small_count-- производит " << small_count-- << endl;
  cout << "конечное значение small_count равно " “ small_count << endl;
  cout << "big_count равно " << big_count << endl;
  cout << "--big_count производит " << --big_count << endl;
  cout << "конечное значение big_count равно " << big_count << endl;
}
```

Когда вы откомпилируете и запустите эту программу, на вашем экране появится следующий вывод:

```
small_count равно 0
small_count-- производит 0
конечное значение small_count равно -1
big_count равно 1000
—big_count производит 999
конечное значение big_count равно 999
```

Как видите, префиксный и постфиксный операторы уменьшения C++ работают так же, как и соответствующие операторы увеличения, с той лишь разницей, что они уменьшают значение переменной на 1.

2. Порядок выполнения работы

1.Проанализировать условие задачи.

2.Разработать алгоритм и создать программы разрешения задач, которые приведены в пункте 3 (задания).

3.Результаты работы оформить протоколом.

3. Задания

1. Создать программу, выводящую на экран ваше имя.
2. Создать программу, вычисляющую и выводящую на экран значение выражения $20x^{1/3}+15^2$.
3. Создать программу, вычисляющую и выводящую на экран среднее арифметическое двух вещественных чисел, сохранённых в переменных ***x*** и ***y***.
4. Создать программу, вычисляющую и выводящую на экран куб числа, сохраненного в целой переменной ***num***.
5. Создать программу, вычисляющую и выводящую на экран сумму цифр натурального двухзначного числа, сохраненного в целой переменной ***s***.
6. Создать программу, вычисляющую и выводящую на экран сумму цифр натурального трёхзначного числа, сохраненного в целой переменной ***chis***.

4. Контрольные вопросы

1. Какая структура программы на языке С++ ?
2. В чем заключаются этапы перестроения программного кода на языкуе С++ в исполнительный файл?
3. Что такое препроцессор? Как получить единицу трансляции?.
4. Объясните смысл понятия оператор.
5. Что понимают под типом данных?
6. Какие типы данных используют в языке С++?
7. Укажите операции с наивысшим и низшим приоритетом.
8. Какие потоки языка С++ предназначены для ввода и вывода данных?

5. Пример выполнения задания

Решение задачи 5. Создать программу, вычисляющую и выводящую на экран сумму цифр натурального двухзначного числа, сохраненного в целочисленной переменной ***s***.

Чтобы решить данную задачу, необходимо знать и понимать:

а) что такое натуральное число? Натуральное число – это положительное целое число;

б) что такое двухзначное число? У двухзначного числа первая цифра - это количество десятков, вторая – количество единиц;

в) как найти сумму цифр двухзначного числа? Чтобы найти сумму цифр двухзначного числа необходимо количество десятков в этом числе сложить с количеством единиц.

Листинг программы:

```
/* Подключаем библиотеку, чтобы работали функции вывода на экран и ввода с клавиатуры – стандартные потоки ввода-вывода*/
#include < iostream >

// Подключаем библиотеку, чтобы работала функция getch();
#include < conio.h >

// Используем стандартную библиотеку имен  std
using namespace std;

int main() {
	int s,	// s – переменная, в которую записывается исходное число
	    sum; /* sum – переменная, в которую записывается сумма цифр исходного числа*/
	printf ("Vvedite chislo:");	// Вывод фразы на экран
	cin >> s;		/* Вводим значение числа с клавиатуры */
	sum =  s/10;	/* Вычисляем результат деления исходного числа s на 10 */
	sum += s % 10;/*Суммируем остаток от деления исходного числа s на 10 */
	cout << sum;	/* Выводим результат на экран*/
	getch();	/*Используем, чтоб окно не закрывалось сразу же */
	return 0;	/*Возвращает нулевое значение */
}
```

Список литературы

1. Культин Н. Основы програмирования в Visual C++ 2010 / Н. Культин. – СПб.: БХВ–Петербург, 2010. – 384 с.
2. Мартынов Н.Н. C# для начинающих 2007 / Н.М. Мартынов. – КУДЕЦ-ПРЕСС, 2007. – 272 с.
3. Пахомов Б. C/C++ и MS Visual C++ 2008 для начинающих / Б.Пахомов – БХВ-Петербург, 2008. – 624 с.

Навчальне видання

Методичні вказівки до виконання лабораторних робот засобами пакета visual studio (Принципи побудови програм на мові С++)

Укладачі: СИМОНОВА Ольга Геннадіївна
ВОРОНЦОВА Дар'я Володимирівна

Відповідальний за випуск О.В. Шоман

Роботу до видання рекомендував А.М. Краснокутський

В авторській редакції

План 2012 р., поз.35

Підп. до друку 10.04.2013 Формат 60х84 1/8. Папір офісний.
RISO- друк. Гарнітура Таймс. Ум. друк. арк. 3,3. Наклад 50 пр.
Зам. № 92 Ціна договірна.

Видавець ф виготовлювач
Видавничий центр НТУ “ ХПІ “.
вул. Фрунзе, 21 м. Харків-2, 61002

Свідоцтво суб’єкта видавничої справи ДК № 3657 від 24.12.2009 р.