[5. Написание и компиляция простых программ на C/C++.](#_Toc515769916)

[5.1. Запуск среды Visual C++.](#_Toc515769917)

[5.2. Написание вашей первой программы.](#_Toc515769918)

[5.2.1. Пример простой программы на С.](#_Toc515769919)

[5.2.2. Структура простой программы.](#_Toc515769920)

[5.2.3. Как сделать программу читаемой.](#_Toc515769921)

[5.2.4. Подготовка и компиляция простых программ на С/С++.](#_Toc515769922)

[5.3. Редактирование текста программы.](#_Toc515769923)

[5.4. Сохранение программ.](#_Toc515769924)

[5.4.1. Построение программы.](#_Toc515769925)

[5.4.2. Использование утилиты Project Workspace.](#_Toc515769926)

[5.4.2.1. Создание нового проекта.](#_Toc515769927)

[5.4.2.2. Добавление файлов к проекту.](#_Toc515769928)

[5.4.3. Запуск команд Build или Rebuild All.](#_Toc515769929)

[5.5. Отладка программы.](#_Toc515769930)

[5.5.1. Понимание сообщений об ошибках и предупреждений.](#_Toc515769931)

[5.5.2. Распространенная ошибка при использовании нового языка.](#_Toc515769932)

[5.5.3. Переключение между окном вывода сообщений и окном редактирования.](#_Toc515769933)

[5.5.4. Использование функций замены или быстрого поиска.](#_Toc515769934)

[5.5.4.1. Выбор опций замены.](#_Toc515769935)

[5.5.5. Переключение между окном вывода сообщений и окном редактирования. Быстрый способ.](#_Toc515769936)

[5.5.6. Значение сообщений об ошибках и предупреждений.](#_Toc515769937)

[5.5.7. Повторная сборка программы ERROR.С.](#_Toc515769938)

[5.6. Запуск программы.](#_Toc515769939)

[5.6.1. Использование встроенного отладчика.](#_Toc515769940)

[5.6.1.1. Использование команд пошагового выполнения (Step Into и Step Over).](#_Toc515769941)

[5.7. Определение точек останова (breakpoints).](#_Toc515769942)

[5.7.1. Запуск программы с точками останова.](#_Toc515769943)

[5.7.2. Использование быстрого просмотра (QuickWatch).](#_Toc515769944)

[5.8. Первая программа](#_Toc515769945)

1. Написание и компиляция простых программ на C/C++.

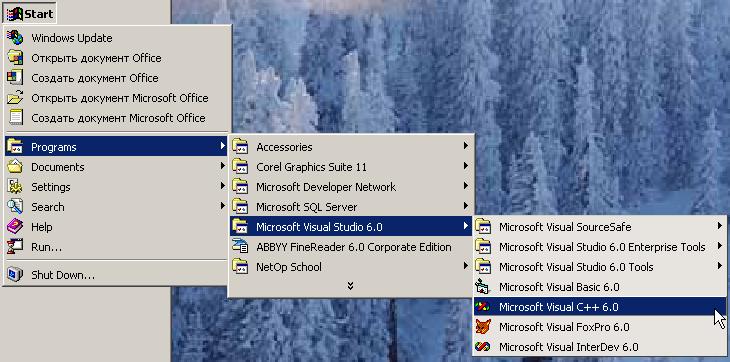
Здесь даются примеры практического использования команд и средств Интегрированной Среды Разработки (IDE) 32-разрядной версии 4.0 Visual C++, называемой иногда Developer Studio (мастерской разработчика), необходимых для создания, редактирования, сохранения, компиляции и отладки простых программ. Возможно, вам понадобится маркирующий карандаш (если еще не воспользовались им). Поскольку интегрированная среда предлагает очень много путей для осуществления каждой операции, вероятно, вы захотите выделить (подчеркнуть) текст, где описан предпочтительный для вас метод. К примеру, некоторые чаще используют команды клавиатуры, а не мышь и меню.

## Запуск среды Visual C++.

Ранее было показано, что запустить среду Visual C++ достаточно просто. Если вы используете мышь, щелкните дважды на значке Visual C++, который находится в группе Microsoft Visual C++. Другой вариант — найдите пункт меню Windows Run и введите следующую команду: (Вот первая возможность выбора наиболее предпочтительного для вас варианта!)

С:MSDEV.EXE

Воспользуйтесь любым способом и запустите Microsoft Visual C++.



## Написание вашей первой программы.

Первое, что нужно сделать для написания программы — это открыть новый файл.

В меню File укажите пункт New... . Эта опция открывает окно New, показанное на рис. 5.1. Окно диалога используется для выбора типа создаваемого файла. В нашем примере, сначала щелкните на опции Text File (текстовый файл), а затем — на кнопке ОК.

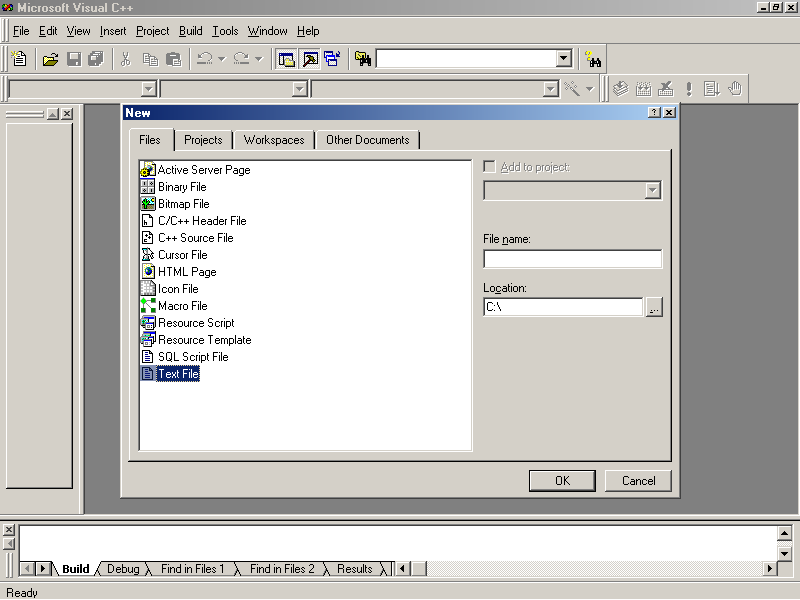


Рис. 5.1. Окно диалога New позволяет программисту создать новую программу

Имея пустое поле редактирования, вы готовы начать ввод программы. Наберите следующий пример:

/\* Простая демонстрационная программа \*/

/\* В программе имеются ошибки! \*/

#include <stdio.h>

#define SIZE 5

void print\_them(int index, char continue, int int\_aray[SIZE]);

main ()

int index;

int int\_aray[SIZE] ;

char continue=0;

print\_them (index, continue, int\_aray) ;

printf(\n\nWelcome to a trace demonstration!");

printf("\nWould you like to continue (Y/N) ");

scanf("%c",continue);

if(continue == 'Y')

for(index=0; index < SIZE; index++)

{

printf("\nPlease enter an integer: ");

scanf("%d",&int\_aray[index]);

}

print\_them (index, continue, int\_aray) ;

return (0);

void print\_them(int index, char continue, int int\_aray[SIZE])

printf("\n\n%d",index);

printf("\n\n%d",continue);

for(index=0; index < SIZE, index++)

printf("\n%d",int\_aray[index]);

Если вы знакомы с языком С, то заметите наличие ошибок в программе. Не исправляйте их. Ошибки сделаны преднамеренно для того, чтобы на практике показать различные возможности интегрированной среды разработки.

### Пример простой программы на С.

Давайте рассмотрим простую программу на языке Си. Следует сразу сказать, что такой пример нужен нам лишь для выявления некоторых основных черт любой программы, написанной на языке Си. Далее мы дадим пояснения к каждой строке, но, перед тем как вы с ними познакомитесь, просто взгляните на программу и попробуйте понять, если сможете, что она будет делать.

/\* X01.C \*/

#include <stdio.h>

#include<iostream>

void **main(**void) /\*Первая программа\*/

{

**setlocale(LC\_ALL,** "Rus");

int **n;**

**n = 1;**

**printf(**"Я простая ");

**printf(**"вычислительная машина\n");

**printf(**"Мое любимое число = %d, потому что оно самое первое\n",n);

// ################################################

**printf(**"Для продолжения нажмите любую клавишу >>");

**scanf(**"%d",&n);

// ################################################

}

Если вы считаете, что программа должна вывести нечто на экран дисплея, то вы совершенно правы! Несколько труднее понять, что же появится на экране на самом деле, поэтому давайте выполним программу на ЭВМ и посмотрим к чему это приведет. Первый шаг заключается в использовании имеющегося у вас текстового редактора для создания файла, содержащего текст программы. Этому файлу необходимо будет присвоить какое-то имя; если вам не приходит в голову ничего оригинального, то назовите его main.с. Выполните компиляцию вашей программы. (Для этого вы должны терпеливо ознакомиться с руководством по компилятору, имеющемуся в составе вашей вычислительной системы.) Теперь запустим программу. Если все пойдет хорошо, то результат должен выглядеть следующим образом:



В общем, этот результат не кажется особенно неожиданным. Но какую роль в программе выполняют символы \n и %d? И вообще некоторые строки выглядят немного странно. Здесь необходимы дополнительные пояснения.

ПОЯСНЕНИЯ. Мы выполним два просмотра текста программы: во время первого объясним смысл каждой строки, а во время второго — рассмотрим дополнительные вопросы и детали.

***Первый просмотр: краткий обзор.***

#include <stdio.h> —включение другого файла.

Эта строка указывает компилятору, что нужно включить информацию, содержащуюся в файле stdio.h. main() — имя функции

_03_01

Структура программы, написанной на языке Си.

Любая программа, написанная на языке Си, состоит из одной или более «функций», являющихся основными модулями, из которых она собирается. Наша программа состоит из одной функции main, и круглые скобки указывают именно на то, что main() — имя функции.

/\*простая программа\*/ — комментарий

Вы можете использовать пары символов /\* и \*/ в качестве открывающей и закрывающей скобок для комментария. Комментарии - это примечания, помогающие понять смысл программы. Они предназначены для читателя и игнорируются компилятором.

{ — начало тела функции

Открывающая фигурная скобка отмечает начало последовательности операторов, образующих тело (или определение) функции. Конец определения отмечается закрывающей фигурной скобкой – }.

int num; — оператор описания

С помощью такого оператора мы объявляем, что будем использовать в программе переменную num, которая принимает целые (int) значения.

num = 1; — оператор присваивания f

Этот оператор служит для присваивания переменной num значения 1.

printf (" Я простая"); — оператор вывода на печать

С его помощью выводится на печать фраза, заключенная в кавычки:

Я простая

printf ("вычислительная машина\n"); — еще один оператор вывода на печать. Этот оператор добавляет слова - вычислительная машина - в конец последней печатаемой фразы. Комбинация символов \n указывает компилятору на начало новой строки.

printf ("Мое любимое число %d, потому что оно самое первое. \n", num);

Этот оператор выводит на печать значение переменной num (равное 1), содержащееся во фразе в кавычках. Символы %d указывают компилятору, где и в какой форме печатать значение этой переменной num.

} — конец

Как уже упоминалось, программа завершается закрывающей фигурной скобкой.

Теперь рассмотрим нашу программу более внимательно.

***Второй просмотр: детали.***

#include < stdio.h>;

Файл с именем stdio.h является частью пакета, имеющегося в любом компиляторе языка Си и содержащего информацию о вводе-выводе (например, средства взаимодействия программы с вашим терминалом). В качестве имени файла используется аббревиатура английских слов (программисты называют набор данных, содержащийся в начале файла, заголовком):

Standard input/output header — стандартный заголовок ввода-вывода.

В некоторых случаях включение этой строки в начало программы обязательно, а в некоторых — нет. Мы не можем дать однозначную рекомендацию, поскольку ответ зависит как от программы, так и от используемой вами вычислительной системы. При работе на нашей системе вводить указанную строку в эту программу совсем не обязательно, но на системе, имеющейся у вас, она может быть необходимой. В любом случае ее использование не принесет никакого вреда. В дальнейшем мы будем указывать эту строку только тогда, когда действительно необходимо.

Возможно, вас удивляет, почему одно из основных средств языка — процедуры ввода-вывода — не включается компилятором в программу автоматически. Дело в том, что этот пакет используется далеко не всегда, а ведь одна из целей создания языка Си — получение компактного объектного кода. Между прочим, упомянутая строка не является даже оператором языка Си. Символ # указывает, что она должна быть обработана «препроцессором» языка Си. Как вы уже могли предположить из названия, препроцессор осуществляет некоторую предварительную обработку текста программы перед началом компиляции. В дальнейшем мы рассмотрим несколько примеров использования команд препроцессора.

main()

Выбор имени main в качестве названия нашей программы довольно очевиден; более того, назвать ее как-то по-другому и нельзя. Дело в том, что программа, написанная на языке Си, всегда начинает выполняться с функции, называемой main(), поэтому мы имеем возможность выбирать имена для всех используемых нами функций кроме той, с которой начинается выполнение программы. Зачем здесь скобки? Как уже упоминалось, они указывают на то, что main() — имя функции. Дополнительные вопросы, относящиеся к функциям, будут обсуждаться ниже. Здесь мы только повторим, что функции — это основные модули программы, написанной на языке Си.

В круглых скобках в общем случае содержится информация, передаваемая этой функции. В нашем простом примере передача информации отсутствует и, следовательно, в скобках ничего не содержится. Заканчивая обсуждение данного вопроса, дадим вам один совет: при написании программы старайтесь не пропускать скобок. Файл, содержащий программу, может иметь любое имя, правда, с тем ограничением, что оно должно удовлетворять системным соглашениям и оканчиваться символом .с. Например, вместо main.с мы могли бы выбрать имена mighty.с или silly.с.

/\*простая программа\*/

Использование комментариев облегчает процесс понимания вашей программы любым программистом (включая вас самих). Большим удобством при написании комментариев является возможность располагать их на той же строке, что и операции, которые они объясняют. Длинный комментарий может помещаться на отдельной строке или даже занимать несколько строк. Все, что находится между символом, указывающим на начало комментария /\*, и символом, указывающим на его конец \*/, игнорируется компилятором, поскольку он не в состоянии интерпретировать язык, отличающийся от Си. Фигурные скобки { } (и только они) отмечают начало и конец тела функции. Для этой цели не используются ни круглые (), ни квадратные [] скобки. Фигурные скобки применяются также для того, чтобы объединить несколько операторов программы в сегмент или блок». Если вы знакомы с такими языками, как Паскаль или Алгол, вы легко сообразите, что такие скобки аналогичны операторам begin и end в этих языках.

int num;

Оператор описания переменной — одно из важнейших средств языка Си. Как уже упоминалось выше, в нашем простом примере вводятся два понятия. Первое — использование в теле функции переменной, имеющей имя num; второе — с помощью слова int объявляется, что переменная num принимает целые значения. Точка с запятой в конце строки указывает на то, что в ней содержится оператор языка Си, причем этот символ является здесь частью оператора, а не разделителем операторов, как в Паскале.

Слово int служит ключевым словом, определяющим один из основных типов данных языка Си. Ключевыми словами называются специальные зарезервированные слова, используемые для построения фраз языка; список ключевых слов вы можете найти в приложении в конце книги.

В языке Си все переменные должны быть объявлены. Это означает, что, во-первых, в начале программы вы должны привести список всех используемых переменных, а во-вторых, необходимо указать тип каждой из них. Вообще объявление переменных считается «хорошим стилем» программирования.

Здесь вы можете задать три вопроса. Первый: каким образом надо выбирать имена? Второй: что такое типы данных? Третий: зачем вообще требуется объявлять переменные? Ответы на первый и третий вопросы приведены ниже и отмечены вертикальной линейкой голубого цвета.

Второй вопрос мы обсудим позднее, а здесь сделаем краткое замечание. Язык Си имеет дело с некоторыми классами (или типами) данных: целыми числами, символами и числами с плавающей точкой. Объявление переменной, имеющей целый или символьный тип, позволяет компилятору размещать данные в памяти, осуществлять их выборку и интерпретировать нужным образом.

Выбор имен. Мы предполагаем, что вы используете осмысленные обозначения переменных. Имя переменной может содержать от одного до восьми символов. (Фактически вы можете использовать и большее их число, но компилятор пропустит все символы, начиная с девятого, поэтому имена Shakespeare и shakespencil считались бы одинаковыми, поскольку первые восемь букв у них совпадают). Для образования имени переменной разрешается использовать строчные и прописные буквы, цифры и символ подчеркивания, считающийся буквой. Первым символом должна быть обязательно буква.

**Правильные имена Неправильные имена**

Wiggly $Z^\*\*

Cat1 1cat

Hot\_\_Tub Hot—Tub

\_kcaB don't

В библиотечных процедурах часто используются имена, начинающиеся с символа подчеркивания. Это делается в предположении, что пользователи вряд ли выберут имена, начинающиеся с этого символа, поэтому маловероятно, что одно из них будет случайно выбрано для обозначения другого понятия. Старайтесь не использовать имен, начинающихся с символа подчеркивания, и вам удастся избежать взаимопересечений с множеством библиотечных имен.

*Четыре довода в пользу объявления переменных.*

1. Сведение всех операторов объявления переменных в начало программы облегчает понимание ее смысла. Это особенно справедливо, если вы даете переменным осмысленные имена (например, taxrate [налоговый тариф] вместо r) и, кроме того, включаете в программу комментарии для объяснения того, что обозначают переменные. Документирование программы подобным образом является одним из основных признаков «хорошего стиля» программирования.
2. Размышление о том, что поместить в секцию объявления переменных, побуждает спланировать программу перед тем, как погрузиться в ее написание. Это эквивалентно получению ответов на вопросы: какая информация необходима программе при запуске? Какую выходную информацию хотелось бы получить?
3. Объявление переменных позволяет избежать одной из наиболее коварных и труднообнаруживаемых ошибок — неправильно написанных имен. Например, предположим, что, программируя на некотором языке, вы использовали оператор

B0ZO = 32.4;

а дальше в программе вы ошибочно написали ANS = 19.7\*BOZO - 2.0

случайно заменив цифру 0 буквой О. Вследствие этого в программе появится новая переменная с именем BOZO, и будет использовано какое-то ее значение (возможно нуль или какой-то «мусор»). В результате переменная ANS получит неправильное значение, и вы, возможно, потратите много времени, пытаясь найти причину. Это не может произойти при программировании на языке Си (если только вы не объявили две переменные со столь похожими именами), поскольку компилятор сразу выдаст сообщение об ошибке, как только встретит в программе необъявленную переменную с именем BOZO.

1. Любая программа, написанная на языке Си, не будет выполняться, если не описать все используемые переменные. Мы полагаем, что последний довод окажется решающим в том случае, если первые три вас не убедили.

num = 1;

Оператор присваивания ( “=“) является одним из основных средств языка. Приведенную выше строку программы можно интерпретировать так: «присвоить переменной num значение 1». Дело в том, что, согласно оператору в четвертой строке программы, переменной num была выделена ячейка памяти, и только теперь в результате выполнения оператора присваивания переменная получает свое значение. При желании мы могли бы присвоить ей другое значение — вот почему имя num обозначает переменную. Отметим, что этот оператор тоже заканчивается точкой с запятой.

printf (" Я простая");

printf (" вычислительная машина\n");

printf (" Мое любимое число %d, потому что оно самое первое. \n" , num);

Во всех этих строках используется стандартная функция языка Си, называемая printf(); скобки указывают на то, что мы, конечно же, имеем дело с функцией. Строка символов, заключенная в скобки, является информацией, передаваемой функции printf() из нашей главной функции [main()].

Такая информация называется аргументом; в первом случае аргументом является строка " Я простая". Возникает вопрос: что функция printf() делает с этим аргументом? Ответ довольно очевиден: она просматривает все символы, содержащиеся между кавычками, и выводит их на экран терминала.

Данная строка дает нам пример того, как мы «вызываем» функцию или «обращаемся» к ней, программируя на языке Си. Для этого требуется только указать имя функции и заключить требуемый аргумент (или аргументы) в скобки. Когда при выполнении ваша программа «достигнет» этой строки, управление будет передано указанной функции [в данном случае printf()]. Когда выполнение функции будет завершено, управление вернется обратно в исходную («вызывающую») программу.

Что можно сказать по поводу следующей строки программы? В ней имеются символы \n, которые не появились на экране. В чем дело? Эти символы служат директивой начать новую строку на устройстве вывода. Комбинация \n на самом деле представляет собой один символ, называемый новая строка. Его смысл кратко формулируется так: начать вывод новой строки с самой левой колонки. Другими словами, с помощью этого символа осуществляются те же функции, что и с помощью клавиши [ввод], имеющейся на обычном терминале. Но вы можете сказать, что комбинация \n выглядит, как два символа, а не как один. Вы, конечно же, правы, но просто по смыслу они представляют собой один символ, для которого не существует соответствующей клавиши на клавиатуре. Возникает вопрос: почему для этой цели нельзя использовать клавишу [ввод]? В ответ скажем, что это может быть интерпретировано как некоторая директива вашему текстовому редактору, а не как команда, которая должна быть помещена в память ЭВМ. Другими словами, когда вы нажимаете клавишу [ввод], редактор прекращает заполнение текущей строки, с которой вы в данный момент работаете, и начинает новую строку, оставляя старую неоконченной.

Символ «новая строка» служит одним из примеров того, что называется «управляющей последовательностью». Эта последовательность используется для представления символов, которые трудно или вообще невозможно вводить с обычной клавиатуры. Другими примерами служат \t для табуляции и \b для возврата на одну позицию. В любом случае управляющая последовательность начинается со знака \.

Теперь, мы думаем, стало понятно, почему три оператора печати вывели на экран только две строки: аргумент первого оператора не содержал символа «новая строка».

Вид второй строки, появившейся на экране, может вызвать недоуменный вопрос: почему отсутствуют символы %d, имеющиеся в операторе вывода? Напомним, что напечатанная строка имела следующий вид:

Мое любимое число 1, потому что оно самое первое.

Вы, наверное, уже догадались — при печати вместо символов %d было подставлено число 1, являющееся значением переменной num. По-видимому, комбинация символов %d служит своего рода указателем места в строке, куда необходимо вставить значение переменной num при печати. На языке Бейсик аналогичный оператор печати выглядел бы следующим образом:

PRINT "Мое любимое число"; num; "потому что оно самое -первое".

На самом деле в языке Си обсуждаемый оператор позволяет сделать несколько больше. Символ % сигнализирует программе, что, начиная с этой позиции, необходимо напечатать число, a d указывает, что переменную необходимо печатать в десятичном формате. Функция printf() предоставляет возможность выбора соответствующего формата для печати переменных. Буква f в имени функции printf() фактически служит напоминанием, что это оператор печати с заданным форматом.

### Структура простой программы.

Теперь, после того как мы привели конкретный пример, вы готовы к тому, чтобы познакомиться с несколькими общими правилами, касающимися программ, написанных на языке Си. Программа состоит из одной или более функций, причем какая-то из них обязательно должна называться main(). Описание функции состоит из заголовка и тела. Заголовок в свою очередь состоит из директив препроцессора типа #include и т. д. и имени функции. Отличительным признаком имени функции служат круглые скобки, причем аргумент, вообще говоря, может отсутствовать. Тело функции заключено в фигурные скобки и представляет собой набор операторов, каждый из которых оканчивается символом «точка с запятой». В нашем примере тело состояло из оператора описания, в котором объявлялись имя и тип используемой переменной, оператора присваивания, с помощью которого переменная получила некоторое значение, и, наконец, трех операторов печати, каждый из которых представляет собой вызов функции

printf().

03_02

### Как сделать программу читаемой.

03_03

Наша первая программа была довольно простой, и следующий пример будет ненамного сложнее. Он выглядит так:

/\* Переводит 2 морские сажени в футы\*/

#include<stdio.h>

void main()

{

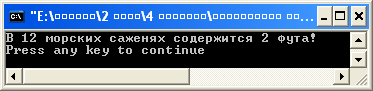
int feet, fathoms;

fathoms=2;

feet= 6\*fathoms;

printf ("В %d морских саженях содержится %d фута!\n", feet, fathoms);

}



Что здесь нового? Во-первых, мы описали две переменные вместо одной. Для этого потребовалось только разделить в операторе описания имена двух переменных запятой.

Во-вторых, мы выполнили вычисления — использовали громадную вычислительную мощность нашего компьютера для умножения 2 на 6. В Си, так же как и во многих других языках, символ \* обозначает умножение. Поэтому смысл оператора

feet = 6\*fathoms;

заключается в следующем: взять величину переменной fathoms, умножить ее на 6 и присвоить результат переменной feet. (Судя по этой парафразе, обычный английский язык менее лаконичен, чем простой язык Си; это одна из причин, лежащих в основе разработки языков программирования.)

И наконец, мы использовали функцию printf() более сложным образом. Если вы выполните эту программу на компьютере, то результат должен выглядеть так:

В 12 морских саженях содержится 2 фута!

Можно заметить, что было произведено две подстановки: первое вхождение символов %d в строку, заключенную в кавычки, было заменено значением первой переменной (fathoms) из списка, следующего за указанной строкой, а второе — значением второй переменной (feet) из этого же списка. Обратите внимание, что список печатаемых переменных расположен в конце оператора.

Область применения данной программы несколько ограниченна, но она может послужить прообразом программы перевода морских саженей в футы. Все, что нам потребуется — специальный способ присваивания произвольных значений переменной feet; о том, как это делается, вы узнаете несколько позже.

Дополнительный пример. Здесь мы приведем еще один пример. До сих пор в наших программах использовалась только стандартная функция printf(). В данном разделе мы хотим продемонстрировать, как включить в программу и использовать функцию, которую вы сами написали.

#include <stdio.h>

void butler (void);

void main()

{

printf("Я вызываю функцию butler.\n");

butler();

printf ("Да. Принесите мне чашку чая и гибкие диски.\n");

}

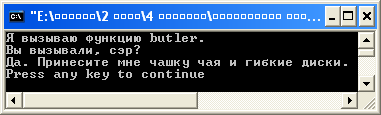
void butler()

{

printf("Вы вызывали, сэр?\n");

}

Результаты работы программы выглядят следующим образом:



Функция butler() определяется точно так же, как и функция main(); ее тело заключено в фигурные скобки. Вызов функции осуществляется путем простого указания ее имени, включая круглые скобки. Мы вернемся к этому важному вопросу позднее, а здесь хотели продемонстрировать ту легкость, с которой вы можете включать в программу свои собственные функции.

### Подготовка и компиляция простых программ на С/С++.

Здесь даются примеры практического использования команд и средств Интегрированной Среды Разработки (IDE) 32-разрядной версии Visual C++, называемой иногда Developer Studio (мастерской разработчика), необходимых для создания, редактирования, сохранения, компиляции и отладки простых программ. Возможно, вам понадобится маркирующий карандаш (если еще не воспользовались им). Поскольку интегрированная среда предлагает очень много путей для осуществления каждой операции, вероятно, вы захотите выделить (подчеркнуть) текст, где описан предпочтительный для вас метод. К примеру, некоторые чаще используют команды клавиатуры, а не мышь и меню.

## Редактирование текста программы.

Одной из основных причин успеха Microsoft Windows является использование графического интерфейса пользователя (Graphical User Interface -GUI), общего для Windows 3.x и Windows NT. Windows 95 использует модифицированный GUI, который, однако, интуитивно понятен опытному пользователю Windows. Если некая функция появляется в двух разных приложениях — например, в текстовом редакторе Windows и в Visual C++ IDE — то при согласованности пользовательских интерфейсов в обоих приложениях для вызова этой функции используются одинаковые команды меню и клавиатуры, и расположены команды аналогично. Это означает, что если даже вы никогда не пользовались редактором среды Visual C++, вы обнаружите, что операции исправления ошибок, перемещения к концу строки, в начало строки или в начало окна редактирования — также просты и знакомы, как в вашем любимом текстовом редакторе Windows. Далее предлагается несколько полезных советов по работе с редактором среды Visual C++. Для быстрого перемещения по строке нажмите клавишу <CTRL> одновременно с клавишей перемещения курсора вправо или влево. При этом курсор редактирования переместится на слово вправо или влево (в зависимости от нажатой клавиши). (Словом называется любой набор символов, разделенных пробелом или знаком пунктуации.) Для удаления слова целиком переместите курсор на пробел перед или после удаляемого слова и нажмите или <CTRL>+<DELETE> (для удаления слова, расположенного справа), или <CTRL>+<BACKSPACE> (для удаления слова слева от курсора).

Чтобы обеспечить максимум рабочего пространства для редактирования, можно отключить горизонтальную и вертикальную полосы прокрутки (ToolsОptions в папке Editor). В этом случае нельзя будет использовать мышь для горизонтальной или вертикальной прокрутки окна. Поэтому необходимо знать две комбинации клавиш: <CTRL>+<PAGE UP> — для перехода в начало программы, и <CTRL>+<END> — для перехода в конец программы. Как вам такая стремительность?

Может возникнуть вопрос: почему не упоминаются эквивалентные клавиши для горизонтального перемещения курсора?

Ответ: чаще всего, профессионально написанная программа умещается по ширине в 80 столбцов стандартного монитора. Это делается для удобства чтения и отладки текста программы: если каждая строка программы видна целиком, вряд ли возникнут незамеченные ошибки в какой-нибудь 95-й позиции.

## Сохранение программ.

Часто между программистом и компилятором возникает важное противоречие. Программист считает, что программа написана без ошибок, а компилятор придерживается другого мнения на этот счет. Если этого недостаточно, алгоритмические способности разработчика оценивает компоновщик. И, в конце концов, удар по самолюбию автора наносит сам микропроцессор, шокированный командами исполняемой программы, полученной в результате работы компилятора и компоновщика.

Хотя разногласия между программистом и компилятором или программистом и компоновщиком не ведут к катастрофе, таковая может возникнуть при отсутствии координации между разработчиком и процессором.

Мораль всей истории такова: сохраняйте файлы перед компиляцией, перед компоновкой и уж обязательно — перед запуском программы. Много печальных историй рассказано программистами, которые не сохраняли файл перед запуском приложения, что приводило к краху приложения или всей операционной системы, и что, в свою очередь, вело к необходимости заново набирать всю программу.

Если вы еще не сделали этого, сохраните пример программы, с которым вы работали. Для этого можно щелкнуть на третьей слева кнопке на панели инструментов (значок на этой кнопке изображает 3,5-дюймовую дискету), либо при помощи команды File|Save, либо нажав <CTRL>+<S>.

При первом сохранении файла среда вызовет окно диалога Save. Сохраните файл под именем ERROR.C.

На рисунке 5.2 показано окно редактирования непосредственно перед сохранением файла. После запоминания строка заголовка будет показывать имя сохраненного файла.

По мере понимания проблем проектирования приложений C/C++ и Windows, разработчик разбивает программу на несколько логически связанных файлов C/C++. К ним добавляются собственные заголовочные файлы (заголовочные файлы имеют расширение .Н). К моменту окончания чтения этой книги вы будете писать приложения в виде файлов исходного текста, заголовочных файлов, файлов ресурсов и других. Таким образом, хотя программный пример состоит только из одного файла, в дальнейшем рассматриваются шаги, необходимые для создания полноценного приложения C/C++ для Windows.

### Построение программы.

Большинство программ для Windows 3.x, Windows 95 и Windows NT состоит из множества файлов. Однако, на начальном этапе программы на C/C++ состоят только из одного файла C/C++, в котором находится функция **main().** По мере роста квалификации программиста такой упрощенный подход становится недостаточным подход становится недостаточным.

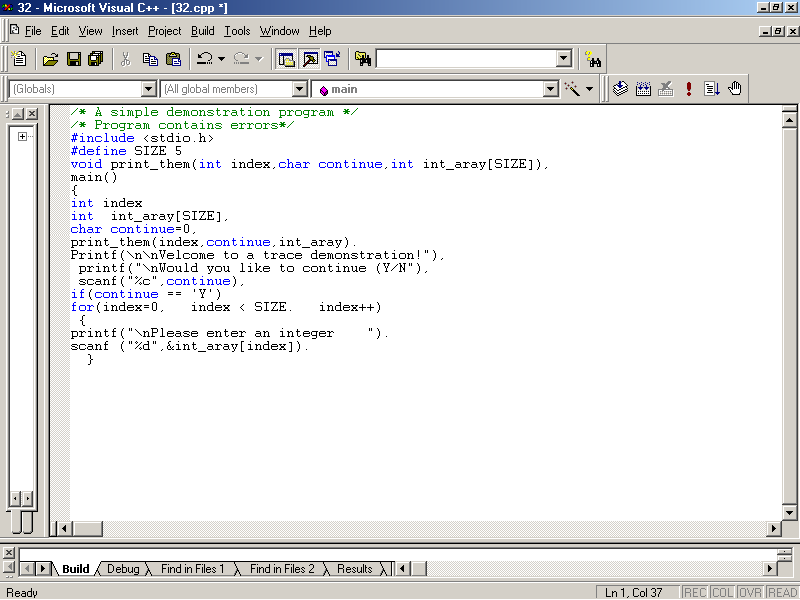


Рис. 5.2. Окно редактирования Visual C++ перед первым сохранением файла

По мере понимания проблем проектирования приложений C/C++ и Windows, разработчик разбивает программу на несколько логически связанных файлов C/C++. К ним добавляются собственные заголовочные файлы (заголовочные файлы имеют расширение .Н). К моменту окончания чтения этого курса вы будете писать приложения в виде файлов исходного текста, заголовочных файлов, файлов ресурсов и других. Таким образом, хотя программный пример состоит только из одного файла, в дальнейшем рассматриваются шаги, необходимые для создания полноценного приложения C/C++ для Windows.

### Использование утилиты Project Workspace.

Для того чтобы начать компиляцию обычной программы C/C++, необходимо предварительно сообщить среде Visual C++ имена всех файлов C/C++ и ресурсов, необходимых для создания исполняемого модуля. Ранее для этого обычно использовался make-файл (make-файлы имеют расширение .МАК).

Make-файлы представляют собой текстовые файлы с заданным синтаксисом, в которых описываются связи между модулями. Другими словами, синтаксис make-файла определяет то, какие файлы должны существовать и компилироваться до того, как результирующий файл может использоваться в следующей фазе процесса компиляции или компоновки. Кстати, из-за сложности этот процесс больше не называется "компиляция и компоновка". В настоящее время для описания этих шагов используется понятие "сборка". Традиционно, make-файлы исполнялись из командной строки при помощи автономной программной утилиты, называемой NMAKE. Microsoft упростил весь этот процесс, включив в компилятор Visual C++ эквивалентную утилиту Project (проект). Эта утилита запускается непосредственно из IDE.

В то время, как раньше программист вынужден был создавать отдельный текстовый файл \*.МАК и запускать утилиту NMAKE, теперь утилита Project позволяет получить тот же результат, не покидая Visual C++ IDE. Утилита Project создает, редактирует и использует make-файлы с расширением \*.МАК. А главное — облегчает процесс создания файла проекта.

Создание нового проекта.

Для создания нового проекта выберите из меню пункт File|New... . Эта команда открывает окно диалога New File. Затем выберите опцию ProjectWorkspace (рабочая область проекта), показанную ранее на рис. 5.1. После добора этого пункта появится окно диалога New Project Workspace (рис. 5.3), это становится первым шагом по созданию файла проекта.

_03_06

Рис. 5.3. Окно диалога New Project Workspace используется для создания нового файла проекта

После этого в работу вступает Мастер.

_03_07

_03_08

Первое, что нужно сделать, — это указать утилите Project имя файла проекта. Это существенно, так как оно в дальнейшем будет именем исполняемого файла.

Многие начинающие программисты C/C++ удивляются тому, что имя исполняемого файла программы не соответствует имени исходного файла, содержащего функции main() или WinMain(). Запомните, что все файлы рабочей области проекта имеют расширение \*.МАК, однако их реальное название может отличаться от названий файлов с исходным текстом. Для нашего примера программы используйте имя файла проекта ERROR. (IDE автоматически добавляет расширение .МАК).

Во-вторых, необходимо указать тип проекта. Среди возможностей: динамически подключаемые библиотеки (DLL) и различные форматы исполняемых файлов. Для нашего примера необходимо выбрать опцию Console Application (консольное приложение).

Если вы введете информацию в поле Location: (местоположение), среда Visual C++ будет использовать диск:\nуть\nодкаталог для размещения новой рабочей области проекта.

Четвертая используемая опция определяет целевую платформу данного проекта. Для 32-разрядной версии компилятора Visual C++ активной является опция Win32. На изображенном рисунке показана полная информация для файла проекта программного примера. Для ее сохранения щелкните на кнопке Create... (создать).

Добавление файлов к проекту.

После создания нового файла проекта необходимо выбрать в команде Project... опцию Insert Files (вставить файлы) для открытия окна диалога Insert Files into Project (добавление файлов в проект). На рис. 5.4 показано окно диалога Project Files для примера файла проекта ERROR. МАК.

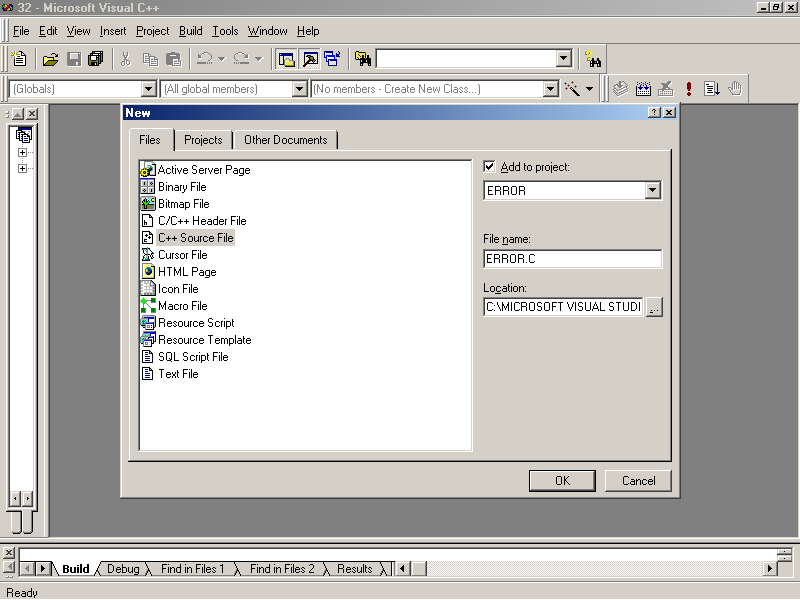
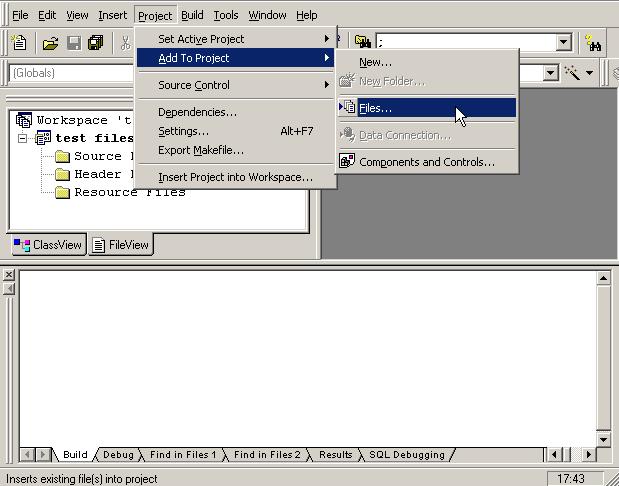
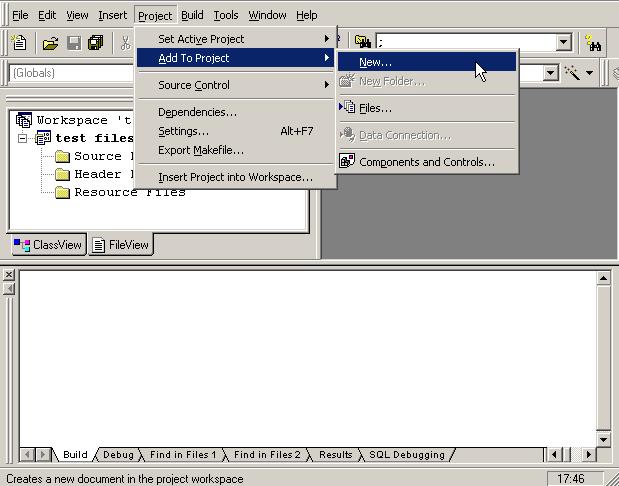


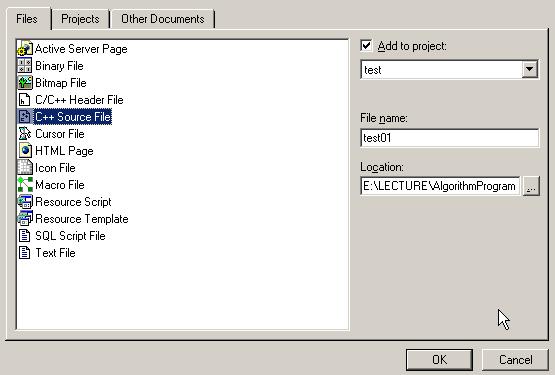
Рис. 5.4. Окно диалога Project Files

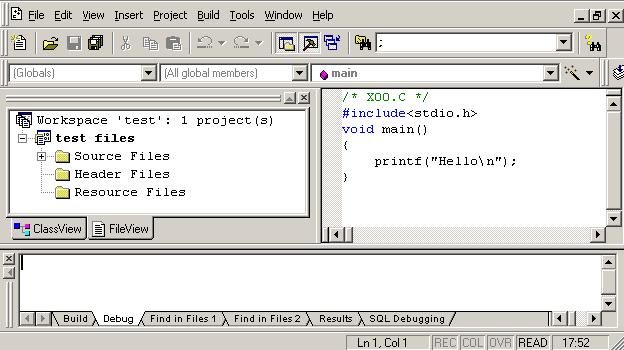
Данное окно используется для удобства нахождения и последующего включения всех файлов, необходимых для создания исполняемой программы. Одно замечание, связанное с типами включаемых в проект файлов: заголовочные файлы (с расширением Н) не добавляются в список файлов проекта. Заголовочные файлы подключаются к процессу сборки непосредственно при помощи команды препроцессора #include. Окно Insert Files into Project очень напоминает стандартное окно диалога Windows — File. Оно позволяет указывать устройство и путь по умолчанию и автоматически показывает список имен файлов в целевом каталоге. В нашем примере достаточно дважды щелкнуть на имени ERROR.C в списке File Name:. После этого имя файла автоматически добавляется в файл проекта. Если бы был более сложный файл проекта, такой как для создания Windows-приложений, далее нужно было бы продолжить добавление файлов, необходимых для проекта. В нашем примере, однако, достаточно одного файла. Теперь можно формально закончить описание файла проекта, щелкнув на кнопке Add (добавить).



В проект входит и исходный текст нашей первой программы.

****

****

****

### Запуск команд Build или Rebuild All.

После того как создан файл проекта, среда компилятора готова к построению исполняемого файла. Помните, что в Visual C++ этот процесс называется сборкой.

На рис. 5.5 показано меню Project с выделенной курсором командой Rebuild All

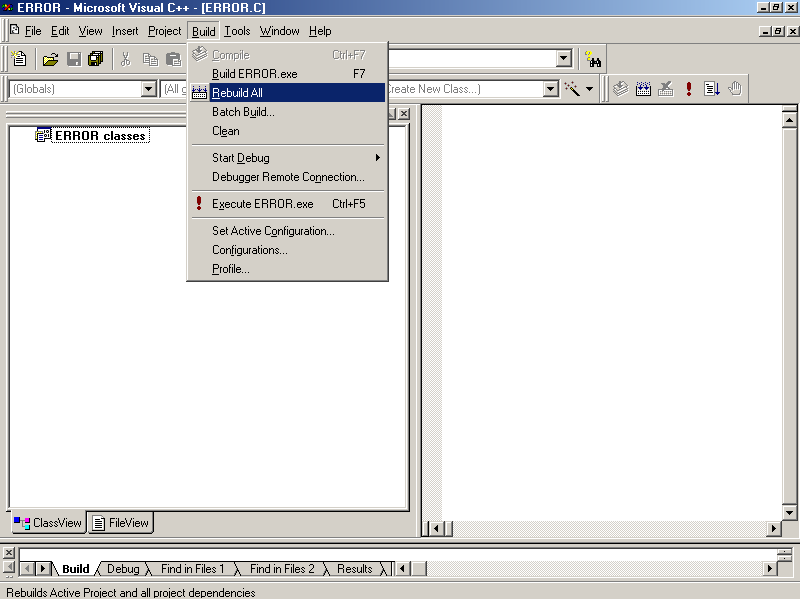
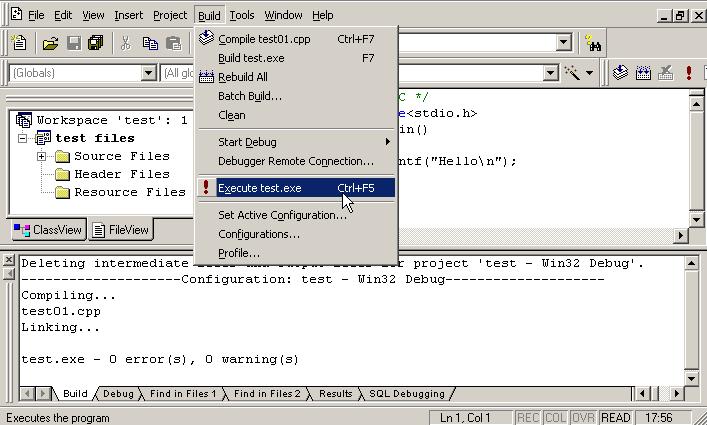


Рис. 5.5. Команда меню Rebuild All компилирует и компонует выбранное новое приложение

Единственным отличием команд Build и Rebuild All является то, что команда Rebuild All не проверяет даты создания файлов, использующихся в конкретном проекте. Эта команда заново компилирует и компонует все файлы проекта.

Поскольку неправильно эксплуатируемая система может иметь неточно установленный внутренний таймер, для небольших приложений наиболее надежный путь — выбрать опцию Rebuild All. Теперь можно запустить процесс сборки, щелкнув на команде Rebuild All или нажав <ENTER> при выделенной курсором команде.

****

****

## Отладка программы.

Если программа содержит синтаксические ошибки, при выполнении команд Build или Rebuild All автоматически открывается окно сообщений, выдаваемых компилятором, что видно в нижней части экрана на рис. 5.6.

Каждое сообщение начинается с имени исходного файла, который в нашем примере называется D:\VISUALC\ERROR\ERROR.C. Это имя — существенно, поскольку в обычном Windows-приложении содержится множество исходных файлов.

Сразу же за именем исходного файла следует номер строки (в скобках), где обнаружена ошибка или предупреждение. В нашем примере первое сообщение об ошибке получено для строки 9. Следом за номером строки — двоеточие, затем слово "error" (ошибка) или "warning" (предупреждение) и соответствующий номер ошибки.

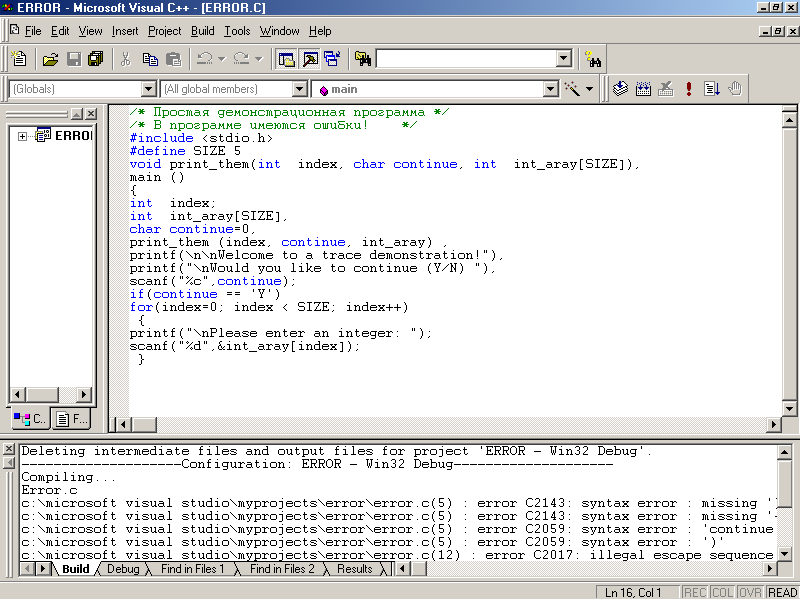


Рис. 5.6. Окно компиляции Visual C++ для просмотра синтаксических ошибок программы.

Программу с одними предупреждениями запустить можно, но при наличии сообщений об ошибках — нет. И, наконец, в каждой строке сообщений содержится краткое описание обнаруженной синтаксической ошибки.

### Понимание сообщений об ошибках и предупреждений.

Предупреждения могут свидетельствовать об автоматическом выполнении стандартных правил преобразования C/C++. Примером такого правила может служить автоматическое округление вещественного числа при присваивании его целой переменной. Это не означает наличие ошибки в программе, просто данный оператор использует некоторые неявные возможности C/C++.

Например, все функции, имеющие прототипы в файле МАТН.Н, имеют формальные параметры типа double и возвращают значение типа double. Если ваша программа вызовет какую-либо из этих функций с аргументом типа float, компилятор выдаст предупреждение, извещающее вас о том, что при помещении аргумента в стек вызова выполняется его преобразование из типа float в тип double.

Вы можете избежать многих предупреждений, обойдя действующие по умолчанию автоматические преобразования языка. Это можно сделать, выполнив непосредственно в программе операции или функции, предназначенные для выполнения неявных действий. К примеру, предупреждение, описанное в предыдущем разделе, можно устранить, если сделать явное преобразование аргумента из типа float в тип double.

### Распространенная ошибка при использовании нового языка.

Например, сообщение об ошибке "...missing ')'..." говорит о том, что может случиться, если вы впервые используете новый язык. Здесь программист попытался дать переменной имя зарезервированного в языке ключевого слова. Если бы вы работали со знакомым вам языком программирования, скорее всего этой ошибки не произошло бы.

В C/C++ слово continue является ключевым. В примере программу имя переменной было выбрано из соображений читабельности и самодокументирования программы; однако, оно пересеклось с ограничениями языка.

### Переключение между окном вывода сообщений и окном редактирования.

После просмотра списка сообщений об ошибках и предупреждений нужно вернуться в окно редактирования для выполнения необходимых изменений в программе. Вы можете выбрать окно редактирования либо щелкнув мышью внутри этого окна, либо перейдя в меню Window и щелкнув на имени файла ERROR.C. Наиболее простой и доступный путь — дважды щелкнуть на том сообщении об ошибке компилятора, которое вы хотите исправить. При этом IDE сама найдет ошибочную строку исходной программы. При использовании любого способа окно редактирования становится самым верхним.

### Использование функций замены или быстрого поиска.

Иногда возникает необходимость быстрого поиска какого-либо слова в программе. Для этого можно вызвать окно диалога SearchReplace... (поиск|замена); однако, Visual C++ IDE предлагает более быстрый способ. Если вы внимательно посмотрите на панель инструментов на рис. 5.7, то увидите слово "continue" в поле списка Quick Find (быстрый поиск).

Для использования Quick Find достаточно щелкнуть левой клавишей мыши в любой части этого окна и напечатать слово, которое вы хотите найти. Потом можно запустить поиск, нажав клавишу <ENTER>.

На рис. 5.7 показан результат таких действий. Первая найденная переменная continue подсвечивается.

Такой подход удобен для определения первого местоположения слова continue, однако, в нашем случае этого недостаточно, так как мы должны найти все его вхождения. Для этого лучше воспользоваться окном диалога Edit|Replace... (правка|замена), показанным на рис. 5.8.

Проще всего для использования Replace... поместить курсор на искомое слово перед вызовом опции Edit|Replace... . В этом случае искомое слово автоматически вводится в поле списка Find what: (что искать) при вызове команды.

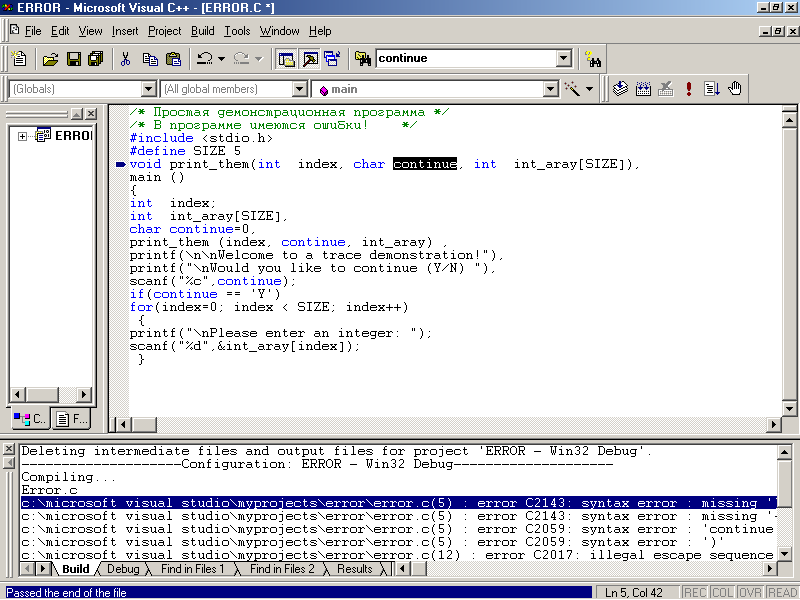


Рис. 5.7. Использование поля списка Quick Find

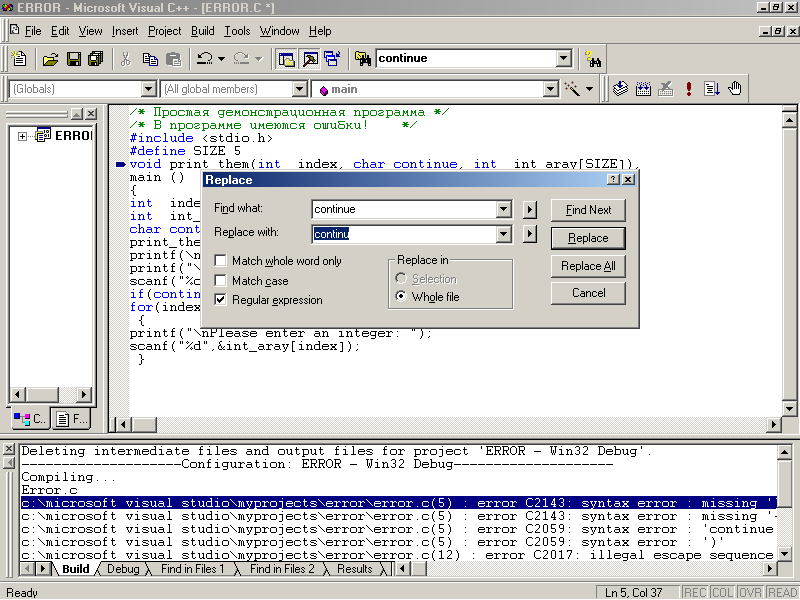


Рис.5.8. Использование опции редактора Edit|Replace

На рис. 5.8 показан данный подход, при этом первоначально курсор был помещен на переменную continue, которая на предыдущем рисунке изображена подсвеченной. Попробуйте воспроизвести эту последовательность действий и посмотрите, удалось ли получить тот же вид экрана, который показан на рис. 5.8.

В нашем примере мы хотим, чтобы переменная, названная continue, сохраняла свой смысл, но писалась отлично от ключевого слова. Для этого нужно вручную ввести слово "continu" в окно списка Replace with: (заменить на) в окне диалога Replace.

Обратите внимание на то, что это окно диалога содержит многие опции функции "поиск-и-замена" стандартного текстового процессора, например, возможность сравнения отдельных слов и учета регистра букв. Если вы не сталкивались с языком C/C++, то, возможно, удивитесь тому, что в C/C++ различаются прописные и строчные буквы. Поэтому переменные, названные TOTAL и total, рассматриваются как различные имена. Один совет: сохраните файл перед тем, как выполнять какие-либо операции поиска и замены. Это позволит вам легко восстановить текст при ошибочных совпадениях шаблонов. Другой вариант — использовать команду Edit|Undo. Однако, если буфер Undo недостаточно велик для того, чтобы запомнить все сделанные операции поиска и замены, то функция Undo может не восстановить вашу программу целиком. Теперь, когда вы ввели нужную информацию в окне Replace, все готово для выполнения замены. Есть, однако, одна проблема. В программе содержится оператор вывода с сообщением "\nWould you like to continue (Y/N)." ("Хотите ли вы продолжить?"). Если вы выберите в окне Replace опцию Replace All, в вашем выходном сообщении будет синтаксическая ошибка, так как выполнение этой опции приведет к появлению ошибки в слове "continue" при выводе сообщения на экран. Поэтому щелкните теперь на кнопке Find Next (найти следующий).

Выбор опций замены.

Окно диалога Replace предоставляет вам несколько опций поиска. Опция Find Next осуществляет поиск следующего вхождения строки. Replace выполняет подстановки новой строки. Опция Replace All непрерывно просматривает весь текст и осуществляет поиск и замену требуемой строки. В нашем примере необходимо раз за разом выбирать Replace и Find Next до тех пор, пока вы не замените каждое вхождение переменной continue на ее новое название continu. Помните о том, что не нужно менять слово "continue" в операторе printf().

### Переключение между окном вывода сообщений и окном редактирования. Быстрый способ.

Ранее мы видели, что переключение между окном вывода сообщений и окном редактирования требует некоторых манипуляций с клавиатурой или мышью. Есть более простой способ переключения между этими окнами. Однако сначала — если строго следовать циклу проектирования — вы должны остановиться и собрать вашу программу. Если сделаны все описанные выше подстановки слова continu, окно вывода сообщений должно выглядеть аналогично показанному на рис5.9.

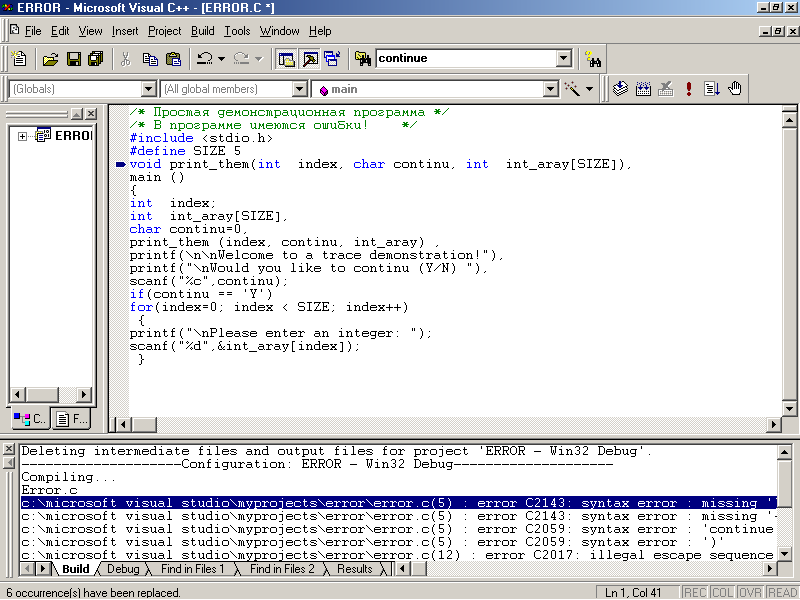


Рис. 5.9. Окно вывода сообщений

Быстрый способ переключения между окнами очень прост. Сначала поместите курсор на интересующее вас сообщение об ошибке или предупреждение. В нашем примере выберите первое сообщение в новом окне вывода сообщений:

warning C4013: 'Printf' undefined; ...

Затем нажмите <ENTER>. И все! Интегрированная среда автоматически переключится в окно редактирования и сама высветит ошибочный фрагмент программы, как показано на рис. 5.10.

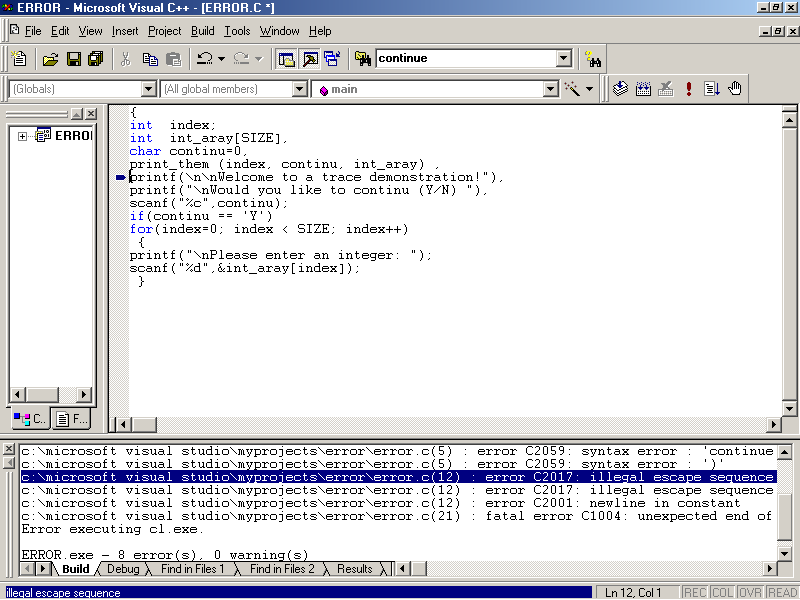


Рис. 5.10. Код, содержащий ошибку, автоматически высвечивается в окне редактирования

### Значение сообщений об ошибках и предупреждений.

При работе с новым языком фактически существует два основных направления его изучения. Во-первых, требуется время на изучение синтаксиса и нюансов самого языка. А во-вторых, более глубокое знание включает в себя понимание подсказок новой среды, сообщений об ошибках и предупреждений. Другими словами, вам необходимо понять, как данный компилятор обрабатывает исходный текст.

Приятно узнать, что компилятор Visual C++ генерирует наиболее точные сообщения по сравнению с любыми другими языковыми средами. В нашем примере компилятор с легкостью распознал случай ошибочного использования ключевого слова continue.

Как вам теперь известно, в языке C/C++ различаются строчные и прописные буквы. И опять-таки, компилятор правильно обнаружил ошибку. Название функции printf(), поставляемой с компилятором, пишется только строчными буквами. Поскольку в названии функции случайно была использована заглавная буква "Р", компилятор не смог найти соответствующую библиотечную функцию Printf(). Когда это слово высвечено в окне редактирования, отредактируйте его и замените заглавную "Р" на строчную. Не забудьте сохранить файл.

### Повторная сборка программы ERROR.С.

Теперь программа готова к следующей попытке создать исполняемый файл. Вернитесь в меню Project и выберите опцию Rebuild All. На рис. 5.11 показаны новые выходные сообщения.

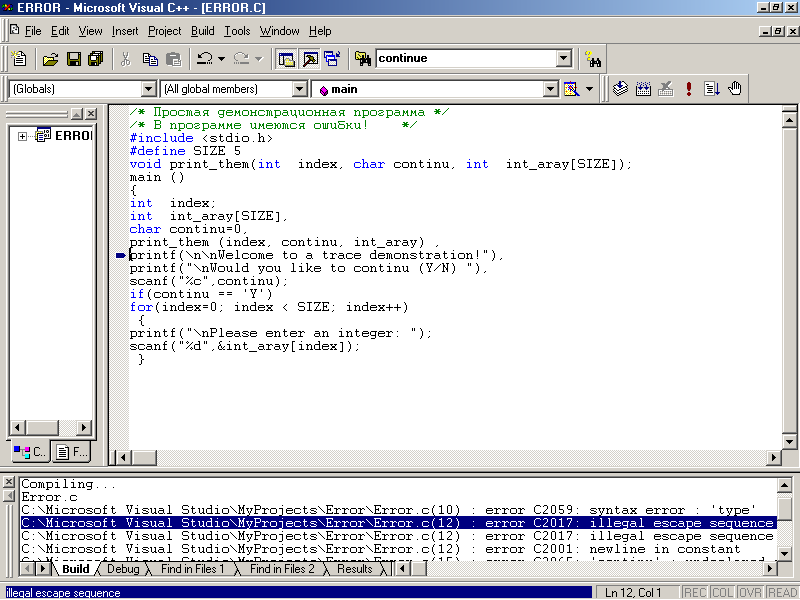


Рис.5.11.Новые выходные сообщения после повторной обработки программы ERROR.C

Вы помните, как быстро переключиться в окно редактирования и автоматически найти неправильную управляющую (escape) последовательность, отмеченную в сообщении об ошибке? (Достаточно поместить курсор на сообщение и нажать <ENTER>.)

Как оказалось, в операторе, содержащем неправильно написанную функцию printf(), имеется еще одна ошибка. В C/C++ все строки выходного формата должны начинаться двойными кавычками. Исправьте строку, поместив двойные кавычки (") после открывающей скобки в функции printf() — то есть, после printf(.

Убедитесь, что теперь первый оператор **printf()** соответствует рис. 5.12. После сохранения файла повторите сборку программы. На рис. 5.12 показан третий вариант обновленного окна вывода сообщений.

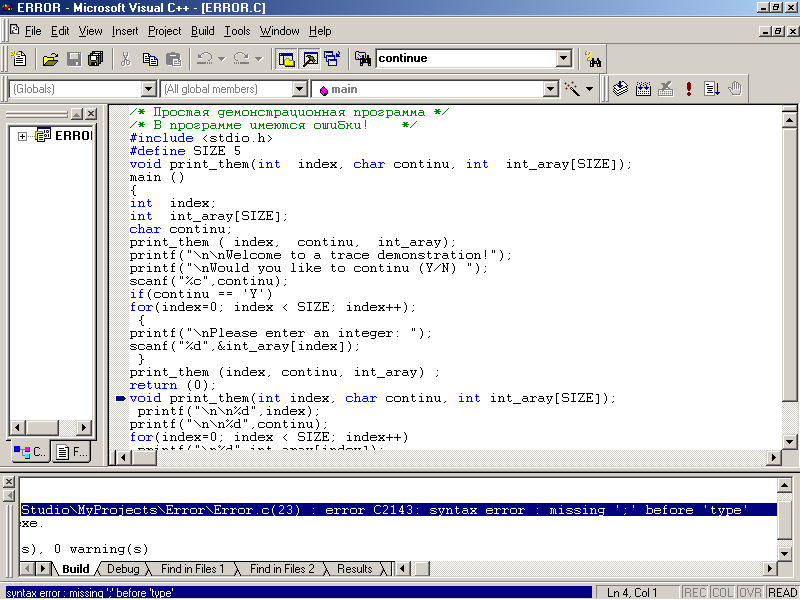


Рис. 5.12. Обновленные выходные сообщения после третьей сборки программы ERROR.C

Последняя наша ошибка была следующей: syntax error : missing ';' before ' ) Поместите курсор на это сообщение и нажмите <ENTER>.

На рис. 5.13 показан оператор, имеющий неправильный синтаксис.

В языке C/C++, в отличие от Паскаля, точка с запятой рассматривается как завершение оператора, а не как разделитель операторов. Поэтому для второго оператора в выражении для цикла for требуется в конце точка с запятой, а не запятая. Замените запятую после константы SIZE на точку с запятой, сохраните файл, и снова запустите Rebuild All

Успех? Как видно в окне вывода сообщений, больше нет ошибок и предупреждений, и команда Rebuild All успешно создала исполняемый файл DEBUG.EXE.

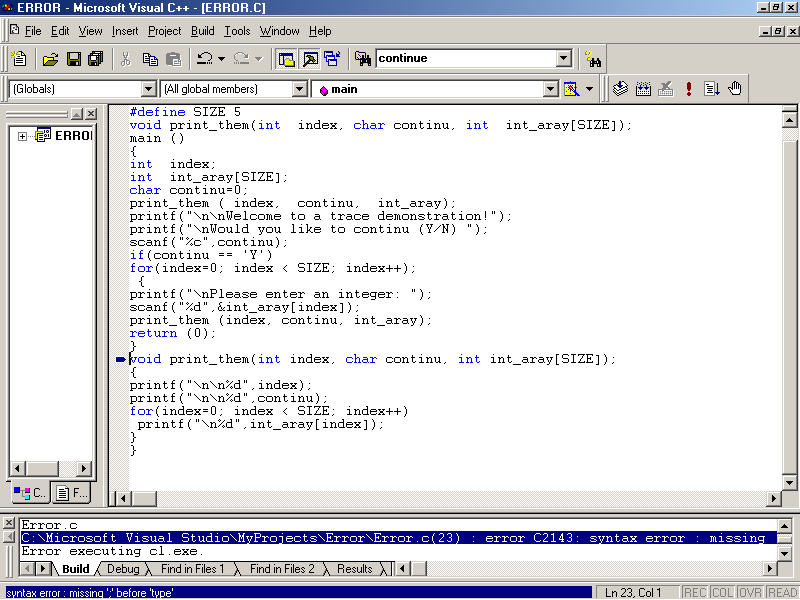
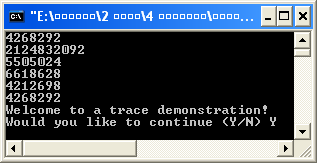


Рис. 5.13. Еще одно сообщение об ошибке

## Запуск программы.

Для запуска программы после успешного завершения операций Build или Rebuild All достаточно щелкнуть на команде Execute (выполнить) в меню Project. Если вы проделаете это с примером программы и нажмете Y после запроса на продолжение, то получите на экране нечто подобное следующему:



На рис. 5.14 показано, что получится, когда вы наберете "Y" и нажмете <ENTER>:

### Использование встроенного отладчика.

В начале примера программы на экран выводится значения неинициализированных элементов массива. Затем выдается запрос на продолжение.

Ответ "Y" (да) подразумевает, что теперь вы хотели бы заполнить массив вашими значениями, а затем распечатать его на экране.

При выполнении примера вы нажали "Y", однако программа выдала ошибку.

Другими словами, хотя программа выглядит синтаксически правильной, то есть, синтаксических ошибок нет, приложение работает не так, как ожидалось. Такие ошибки называются логическими. К счастью, в интегрированном отладчике Visual C++ IDE имеются различные средства, готовые прийти вам на помощь.

Хотя интегрированный отладчик имеет много возможностей, обычно пользуются лишь небольшим набором команд. По сути, отладчик имеет две мощные возможности. Во-первых, он позволяет выполнять программу не целиком, а построчно. Во-вторых, он позволяет просматривать значения любых переменных в любой точке программы.

При правильном использовании эти возможности позволят вам быстро найти ошибочную строку программы. К сожалению, отладчик не исправляет автоматически текст программы. (Поэтому в настоящий момент вашей работе программиста ничто не угрожает!)

Использование команд пошагового выполнения (Step Into и Step Over).

На рис. 5.15 показано меню Debug (отладка). В этом меню чаще всего используются две опции пошагового выполнения: Go (перейти) и Step Into (шаг внутрь).

При использовании любой из этих команд окно редактирования выглядит иначе, чем обычно. При отладке программы в режиме Step Into интегрированный отладчик высвечивает строку программы, которая будет выполняться следующей. Повторите команду Step Into три раза.

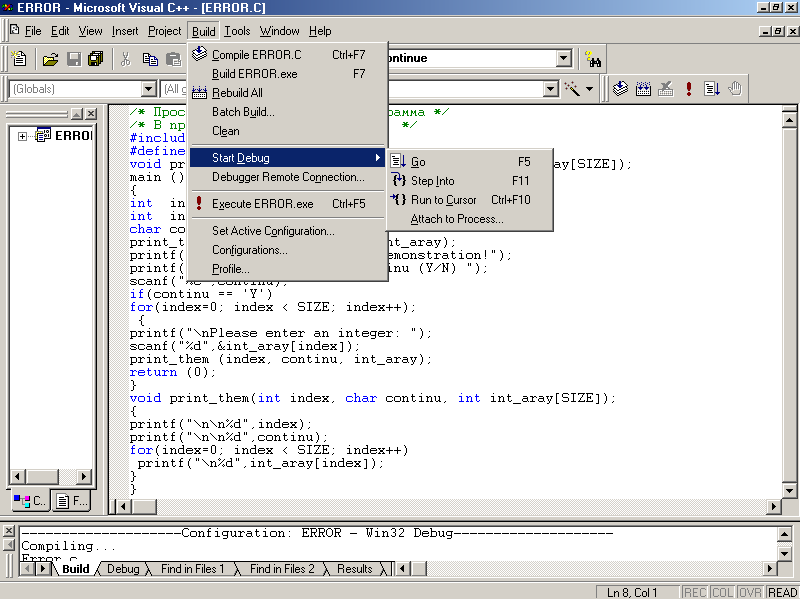


Рис. 5.14. Меню Debug

Стрелка пошагового выполнения (называемая также стрелкой трассировки) располагается рядом с вызовом функции print\_them(). Пусть теперь мы будем выполнять функцию print\_them() с нормальной скоростью. Кстати, как только вы начинаете отладку программы, основное меню Build изменяется на меню Debug, предоставляющее более широкие возможности отладки.

Итак, мы хотим выполнить функцию с нормальной скоростью. Для этого выберите команду Step Over (шаг через). При внимательном наблюдении вы заметите, что функция выполнилась, и стрелка трассировки остановилась на первом операторе printf(). Если так, то хорошо. Теперь нажмите <F10> четыре раза (при нажатии <F8> вы могли бы трассировать исходный код внутри функции printf()) до тех пор, пока стрелка трассировки не остановится на операторе scanf().

В этой точке вам необходимо переключиться в окно выполнения программы. Для этого можно использовать комбинацию клавиш <ALT>+<TAB>. (Возможно, вам понадобится повторить эту комбинацию несколько раз в зависимости от того, сколько задач у вас загружено). Когда вы находитесь в окне ERROR.EXE, показанном на рис. 5.16, ответьте "Y" на вопрос "Would you like to continue (Y/N)" и нажмите <ENTER>.

Интегрированный отладчик немедленно ответит сообщением об ошибке, показанном на рис. 5.18.

Это сообщение относится к только что выполненному оператору scanf(). Посмотрим, достаточно ли вы знаете язык С, чтобы определить, в чем проблема.

Ошибка заключается в неправильном использовании функции scanf(). Функция должна получить адрес ячейки памяти, в которую заносится значение. Посмотрите на этот оператор:

scanf("%c", continu);

Как вы видите, в этом операторе не задействован адрес. Решение — поставить операцию определения адреса (&) перед переменной continu. Исправьте оператор, чтобы он выглядел следующим образом:

scanf("%c", &continu);

Сохраните изменения и запустите Rebuild All.

## Определение точек останова (breakpoints).

Точку останова можно рассматривать как стоп знак для интегрированного отладчика. Логически, точки останова говорят о правильности всех операторов до некоторой точки останова, и отладчик не должен тратить время на их пошаговое прохождение.

В нашем примере мы знаем, что все операторы до функции scanf() — правильные. Вы только что отредактировали эту строку и теперь хотите видеть, будет ли новый оператор нормально работать. Для эффективности отладки устанавливается точка останова в строке 21 на функции scanf(). На рис. 5.19 показан другой способ задания точек останова: при помощи команды Edit|Breakpoints... (правка|точки останова).

Эта опция меню открывает окно диалога Breakpoints. По умолчанию, тип точки останова — Break at Location (останов на строке). Вам необходимо лишь напечатать номер строки в окне Break at: или использовать раскрывающийся список. В нашем примере это строка 21. (Если в вашем тексте оператор scanf() имеет другой номер строки — это возможно из-за наличия пустых строк в исходной программе, — введите номер строки, которую оператор scanf() занимает в вашем файле программы.) Затем выберите кнопку ОК.

### Запуск программы с точками останова.

Для отладки программы в режиме быстрого прогона между точками останова можно использовать опцию меню DebuglGo.

Предположим, что описанная выше точка останова установлена, и запустим команду Go. (Выберите команду мышью, используйте клавиатуру для доступа к команде через меню или нажмите <F5>.) Обратите внимание, что стрелка трассировки быстро пробегает до оператора с функцией scanf(), а затем останавливается. Как только отладчик остановился в точке останова, вы можете вернуться к пошаговому прохождению программы или просмотреть содержимое переменных. В настоящий момент нас интересует, будут ли работать изменения, сделанные в операторе scanf(). Нажмите <F10> (Step Over), переключитесь в окно выполнения программы, напечатайте заглавную букву Y и нажмете <ENTER>.

Успех! Интегрированный отладчик больше не выдает окна с предупреждениями. Но означает ли это, что проблема в программе действительно устранена? Простейший способ ответить на этот вопрос — увидеть текущее значение переменной continu.

### Использование быстрого просмотра (QuickWatch).

Команда QuickWatch... открывает окно диалога, позволяющее вам мгновенно просматривать и менять содержимое переменных. Быстрее всего можно поместить некоторую переменную в окно QuickWatch, если переместить на нее курсор в исходном тексте программы и нажать <SHIFT>+<F9>. Если вы проделаете это в примере программы, то увидите окно диалога QuickWatch, аналогичное показанному на рис. 5.15.

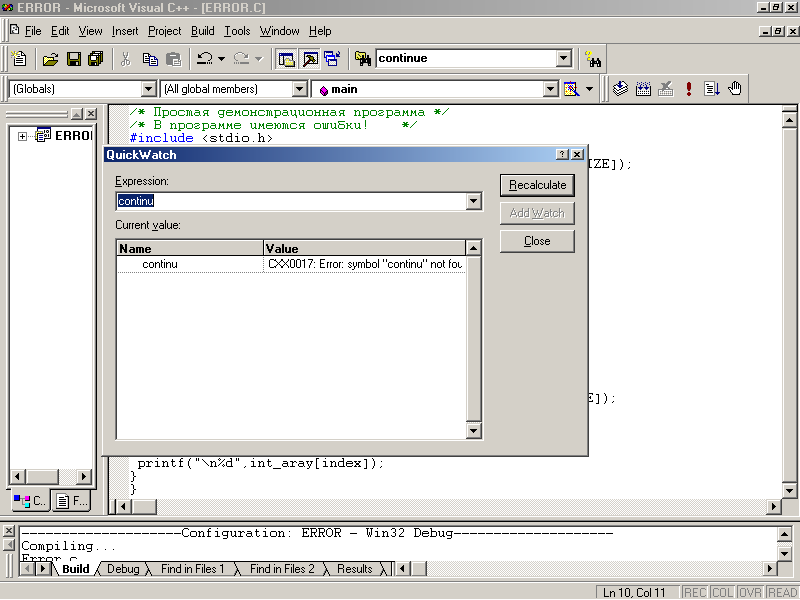


Рис.5.15 Окно QuickWatch

Теперь, когда вы удостоверились, что переменная continu имеет правильное значение, вы можете запустить быстрый прогон программы до конца при помощи команды Debug|Go.

## Первая программа

В следующем примере выполняются те же действия, что и в описанном выше, однако в нем используются уникальные достоинства C++.

//Ваша первая программа на C

#include<stdio.h>

#include<iostream>

int **main()**

{

**setlocale(LC\_ALL,** "Rus");

**printf(**"HELLO World!\n");

**printf(**"Здравствуй мир!\n");

**system(**"PAUSE");

return(0);

}



//Ваша первая программа на C++

#include<stdio.h>

#include<iostream>

usingnamespace **std;**

int **main()**

{

**setlocale(LC\_ALL,** "Rus");

**cout <<** "HELLO World!\n";

**cout <<** "Здравствуй мир!\n";

**system(**"PAUSE");

return(0);

}



Имеется три основных различия между этим примером и примером, описанным выше.

Во-первых, заменен разделитель комментариев: вместо пары /\* \*/ используется //.

Во-вторых, имя файла в #include заменено на iostream.h.

В-третьих, используется другой вызов оператора вывода — cout.

Далее будут отмечаться иногда неуловимые, а иногда очевидные различия между С и C++.

**Что вы должны были узнать.** Ниже приведена краткая сводка строгих правил (но не чрезмерно жестких), которые, мы надеемся, вы усвоили. Мы включили сюда же краткие примеры.

Как назвать файл, содержащий вашу программу: eye.с, или black.с, или infan.c и т. п.

Какое имя можно использовать в качестве названия программы, состоящей из одной функции: main()

Структура простой программы: заголовок, тело, фигурные скобки, операторы

Как описать целую переменную: int varname; Как присвоить значение переменной: varname = 1024; Как напечатать фразу printf ("Хотите купить утку?"); Как напечатать значение переменной: printf ("%d", varname); Символ новая строка: \n

Как включать комментарии в программу: /\*анализ движения наличных денег\*/

**Вопросы и ответы.** Ниже приведено несколько вопросов, которые помогут вам проверить и расширить свое понимание материала.

**Вопросы/**

1. Икабод Боуди Марфут (ИБМ) подготовил программу, приведенную ниже, и принес ее вам для проверки. Пожалуйста, помогите ему найти в ней ошибки.

include studio.h

main() /\*эта программа печатает число недель в году/\*

(

int s

s: = 56;

print (В году s недель.);

2. Что будет напечатано в каждом из примеров, приведенных ниже, в предположении, что они являются частями некоторых полных программ?

a. printf(" Б-э-э Б-э-э, Черная Овца.");

а. printf("У тебя есть шерсть?\n");

б. рrintf("Убирайся!\n Мешок сала!");

в. printf("Что?\n Нет/n Кларнет?\n");

г. int num; num = 2;

printf("%d + %d = %d", num, num, num + num);

**Ответы.**

1. Строка 1: данная строка должна начинаться с символа #; правильное написание имени файла — stdio.h; имя файла должно быть заключено в угловые скобки. Строка 2: вместо фигурных скобок () необходимо использовать круглые (); комментарий должен оканчиваться символами \*/, а не /\* Строка 3: вместо круглой скобки ( должна стоять фигурная {. Строка 4: оператор должен оканчиваться символом «точка с запятой». Строка 5: эту строку (пустую) м-р ИБМ написал совершенно правильно! Строка 6: в операторе присваивания необходимо использовать символ =, а не : =. (К сожалению, м-р ИБМ имеет представление о языке Паскаль.) В году 52 недели, а не 56

Строка 7: оператор должен выглядеть так printf ("В году %d недель. \n\*,.s); Строка 8: отсутствует, но она обязательно должна быть и содержать закрывающую фигурную скобку — ).

2 а Б-э-э Б-э-э, Черная Овца.У тебя есть шерсть?

(Заметим, что пробел после точки отсутствует. Для того чтобы поместить в это место пробел, необходимо было вместо "У тебя писать" У тебя)

б. Убирайся!

Мешок сала!

(Отметим, что курсор теперь находится в конце второй строчки.)

в. Что?

Нет/n Кларнет?

Заметим, что символ (/) производит не тот же эффект, как символ (\)

г. 2+2= 4

(Отметим, что каждое вхождение комбинации символов %d в строку заменяется значением соответствующей переменной из списка. Заметим также, что символ + означает сложение и что таким образом вычисления могут быть проведены «внутри» оператора printf().)

**Упражнения**. Чтобы изучить язык Си, одного только чтения недостаточно. Вы должны попробовать сами написать одну или две простые программы и посмотреть, пройдет ли все так же гладко, как это может показаться в результате этого материала. Мы хотим предложить вам несколько идей, но, если желаете, вы можете воспользоваться своими собственными соображениями на этот счет.

1. Напишите программу, печатающую ваше имя.

2. Напишите программу, печатающую ваши имя и адрес, используя три или более строк.

3. Напишите программу, которая укажет ваш возраст, данный в годах, в днях. Не усложняйте ее, учитывая високосные и невисокосные годы.