Глава 10

Отладка программ на C++

В этой главе…

- Определение типа ошибки
- Использование отладочной печати
- Использование отладчика

Не часто случается (особенно с "чайниками"), что программа идеально работает с первого раза. Крайне редко удается написать нетривиальную программу и не допустить ни одной ошибки.

Чтобы избавиться от ошибок, можно пойти двумя путями. Первый — стереть программу и написать ее заново, а второй — найти и исправить ошибки. Освоение первого пути я оставляю читателю, а в этой главе расскажу о том, как выследить и исправить ошибку в программе.

Определение типа ошибки

Можно выделить два типа ошибок: те, которые компилятор может найти, и те, которые не может. Первый тип называют *ошибками компиляции* (compile-time error). Их довольно легко найти, поскольку компилятор сам указывает место в программе, где встретилась ошибка. Правда, иногда описание ошибки бывает не совсем точным (компьютер так легко сбить с толку!), однако, зная капризы своего компилятора, нетрудно разобраться в его жалобах.

Ошибки, которые компилятор не может найти, проявляются при запуске программы и называются *ошибками времени исполнения* (run-time error). Их найти намного труднее, поскольку, кроме сообщения об ошибке, нет и намека на то, какая именно ошибка возникла и где (сообщения, которые генерируются при возникновении ошибок выполнения, вполне достойны "звания" ошибочных).

Для выявления "жучков" в программе обычно используется два метода. Первый — добавить отладочные команды, выводящие ключевые значения в ключевых точках программы. Увидев значения переменных в месте возникновения ошибки, можно понять, что именно неправильно в данной программе. Второй метод заключается в использовании специальной программы — отладчика. Отладчик позволяет отслеживать процесс выполнения программы.

Использование отладочной печати

Добавление команд вывода в ключевых точках помогает понять, что происходит в программе, и называется методом отладочной печати (иногда именуемым WRITE). Метод WRITE появился во времена, когда программы писались на языке FORTRAN, в котором вывод осуществляется с помощью команды WRITE.

Приведенная ниже "дефектная" программа наглядно демонстрирует применение отладочных команд. Эта программа должна считывать последовательность чисел с клавиатуры и вы-

водить их среднее арифметическое значение. Однако она не делает этого, поскольку содержит две ошибки, одна из которых вызывает аварийный останов, а вторая приводит к неправильному результату.

Данная программа имеется на прилагаемом компакт-диске под именем `ErrorProgram1.cpp`.

```
// ErrorProgram — эта программа усредняла бы
//                ряд чисел, если бы не содержала
//                как минимум одну фатальную ошибку
#include <cstdio>
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int nNumberOfArgs, char* pszArgs[])
{
    cout << "Эта программа содержит ошибки!\n";

    int nSum;
    int nNums;

    // Суммируем ряд чисел, пока пользователь не введет
    // отрицательное число, после чего выводим среднее

    nNums = 0;
    while(true)
    {
        // Ввод очередного числа
        int nValue;
        cout << "Введите следующее число:";
        cin  >> nValue;
        cout << endl;

        // Если это число отрицательное...
        if (nValue < 0)
        {
            // ...то вывести среднее значение
            cout << "Среднее равно: "
                 << nSum/nNums
                 << endl;
            break;

        }

        // Введенное число не отрицательно -добавляем
        // его к накапливаемой сумме
        nSum += nValue;
    }

    // Пауза для того, чтобы посмотреть
    // на результат работы программы
    system("PAUSE");
    return 0;
}
```

После ввода этой программы создайте выполнимый файл (клавиша <F9>).

Запустите эту программу и введите числа 1, 2 и 3, а затем –1. Вы ожидаете увидеть, что их среднее равно двум? Вместо этого очевидного результата будет выдано довольно непривлекательное сообщение об ошибке, показанное на рис. 10.1.

Рис. 10.1 может отличаться в зависимости от используемых операционной системы и компилятора.

Рис. 10.1. Первоначальная версия программы содержит как минимум одну ошибку

Выявление “жучка” № 1

Очень сложно разобраться, в чем именно проблема, по столь неинформативному сообщению. Но в некоторых операционных системах попроще вы получите сообщение (поверьте на слово), в котором будет туманный намек относительно деления на нуль. Впрочем, мы и сами можем догадаться о том, что проблема где-то рядом, поскольку ошибка произошла после того, как мы ввели отрицательное число, но до того, как на экран был выведен результат.

Давайте еще раз посмотрим на так и не выполнившуюся инструкцию:

```
cout << "Среднее равно: "
     << nSum/nNums
     << endl;
```

Кстати говоря, хотя это и единственное деление, использованное нами в программе, это еще не означает, что ошибка именно в нем. Компилятор может сгенерировать команду деления в результате обработки некоторой иной инструкции, написанной программистом. Кроме того, делений хватает и в стандартной библиотеке C++.

Давайте посмотрим, чему равно значение `nNums` перед выполнением деления, изменив код следующим образом:

```
while(true)
{
    cout << "nNums = " << nNums << endl;

    // Остальная часть программы остается неизменной
```

Такое дополнение нашей программы приводит к следующему выводу на экран:

```
Эта программа содержит ошибки!
nNums = 0
Введите следующее число: 1
```

```
nNums = 0
Введите следующее число: 2

nNums = 0
Введите следующее число: 3

nNums = 0
Введите следующее число:
```

Как видите, `nNums` инициализировано нулевым значением, но оно не увеличивается в процессе ввода новых чисел. Это неверно, и именно в этом состоит ошибка в программе. Очевидно, что количество введенных чисел `nNums` должно увеличиваться, чего можно легко достичь, заменив цикл `while` циклом `for`:

```
for(int nNums = 0;;nNums++)
```

Выявление "жучка" № 2

Теперь, когда найдена и исправлена ошибка № 1, можно запустить программу, введя числа, заставившие ее в прошлый раз аварийно завершиться. На этот раз сообщение об ошибке не появится и программа вернет нулевой код выхода, но будет работать не так, как ожидалось. Вместо так горячо ожидаемой двойки будет выведено какое-то нелепое число.

```
Эта программа содержит ошибки!

Введите следующее число: 1
Введите следующее число: 2
Введите следующее число: 3
Введите следующее число: -1
Среднее равно: 229523
```

Очевидно, какая-то из переменных — `nNums` или `nSum` (а возможно, и обе) содержит неверное значение. Для того чтобы исправить ошибку, необходимо узнать, какая именно из этих переменных содержит неверную информацию. Не помешало бы также знать, что содержится в переменной `nValue`, поскольку она используется для подсчета суммы в `nSum`.

Для этого воспользуемся методом отладочной печати. Чтобы узнать значения `nValue`, `nSum` и `nNums`, перепишите тело цикла `for` так, как показано в следующем листинге (версия программы имеется на прилагаемом компакт-диске в файле с именем `ErrorProgram2.cpp`).

```
// ErrorProgram - эта программа усредняла бы
//                ряд чисел, если бы не содержала
//                как минимум одну фатальную ошибку

#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

int main(int nNumberofArgs, char* pszArgs[])
{
    cout << " Эта программа содержит ошибки!"
         << endl;

    // Суммируем ряд чисел, пока пользователь не введет
    // отрицательное число, после чего выводим среднее

    int nSum;

    for (int nNums = 0; ;nNums++)
```

```
    {
        // Ввод следующего числа
        int nValue;
        cout << "Введите следующее число:";
        cin  >> nValue;
        cout << endl;

        // Если введенное число отрицательно...
        if (nValue < 0)
        {
            // ...выводим результат усреднения
            cout << "\nСреднее равно: "
                 << nSum/nNums
                 << "\n";
            break;

        }

        // Вывод отладочной информации
        cout << "nSum = " << nSum   << "\n";
        cout << "nNums= " << nNums  << "\n";
        cout << "nValue= "<< nValue << "\n";
        cout << endl;

        // Введенное число не отрицательно, суммируем его
        nSum += nValue;
    }

    // Пауза для того, чтобы посмотреть
    // на результат работы программы
    system("PAUSE");
    return 0;
}
```

Обратите внимание на то, что информация о состоянии отслеживаемых переменных `nValue`, `nSum` и `nNums` выводится в каждом цикле.

Ответ программы на ввод уже привычных 1, 2, 3 и –1 приведен ниже. При первом же проходе `nSum` принимает какое-то несуразное значение, хотя оно должно равняться нулю (поскольку к этой переменной пока что ничего не прибавлялось).

```
Эта программа содержит ошибки!

Введите следующее число:1
nSum = -858993460
nNums= 0
nValue= 1

Введите следующее число:2
nSum = -858993459
nNums= 1
nValue= 2

Введите следующее число:3
nSum = -858993457
nNums= 2
nValue= 3

Введите следующее число:
```

Внимательно присмотревшись к программе, можно заметить, что `nSum` была объявлена, но не проинициализирована. Для того чтобы исправить эту ошибку, объявление переменной необходимо изменить следующим образом:

```
int nSum = 0;
```

Примечание. Пока переменная не проинициализирована, ее значение непредсказуемо.

Теперь, когда вы нашли все ошибки, перепишите программу так, как показано в следующем листинге (эта программа имеется на прилагаемом компакт-диске в файле `ErrorProgram3.cpp`).

```
// ErrorProgram - эта программа усредняет
//                ряд чисел и не содержит
//                ошибок
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

int main(int nNumberofArgs, char* pszArgs[])
{
    // Суммируем ряд чисел, пока пользователь не введет
    // отрицательное число, после чего выводим среднее
    int nSum = 0;

    for (int nNums = 0; ;nNums++)
    {
        // Ввод следующего числа:
        int nValue;
        cout << "Введите следующее число:";
        cin  >> nValue;
        cout << endl;

        // Если введенное число отрицательно...
        if (nValue < 0)
        {
            // ...выводим усредненное значение
            cout << "\nСреднее равно: "
                 << nSum/nNums
                 << "\n";
            break;

        }
        // Введенное число не отрицательно, суммируем его
        nSum += nValue;
    }
    // Пауза для того, чтобы посмотреть
    // на результат работы программы
    system("PAUSE");
    return 0;
}
```

Теперь вывод программы будет правильным. Протестировав эту программу с другими наборами чисел, я убедился, что она работает без ошибок.

Использование отладчика

В небольших программах метод отладочной печати работает довольно неплохо. Добавление отладочных команд — достаточно простой и не влияющий на время компиляции способ нахождения ошибок, с помощью которого можно быстро отыскать ошибку, если программа невелика.

В больших программах зачастую программист даже не знает, куда нужно добавлять отладочные команды. Работа по добавлению отладочных команд, перезапуску программы, повторному добавлению отладочных команд и т.д. становится утомительной. Кроме того, после каждого переписывания программу нужно собирать заново. Не забывайте, что в большой программе один только процесс сборки может занять немало времени.

В конце концов, с помощью этого метода почти невозможно найти ошибку, связанную с указателями. Указатель, выведенный на экран в шестнадцатеричном виде, малоинформативен, и, пока программист поймет, что нужно сделать для исправления ошибки, программа успеет морально устареть.

Второй, более изощренный метод — использование отдельной утилиты, которая называется отладчиком. С помощью отладчика можно избежать трудностей, возникающих при использовании методики отладочной печати (однако, если вы хотите использовать отладчик, вам придется научиться с ним работать).

Что такое отладчик

Отладчик — это утилита, встроенная, например, в Dev-C++ или Microsoft Visual Studio .NET (в этих приложениях программы отладчиков отличаются, однако работают они по одному принципу).

Программист управляет отладчиком с помощью команд так же, как, например, при редактировании или компиляции программы. Команды отладчика можно выполнять с помощью контекстных меню или горячих клавиш.

Отладчик позволяет программисту контролировать работу программы по ходу ее выполнения. С помощью отладчика можно выполнять программу в пошаговом режиме, останавливать ее в любой точке и просматривать содержимое любой переменной. Чтобы оценить удобство отладчика, его нужно увидеть в действии.

Работа с отладчиком

В отличие от стандартизированного языка C++, набор команд, поддерживаемый отладчиком, варьируется от производителя к производителю. К счастью, большинство отладчиков поддерживают некоторый базовый набор команд. Необходимые нам команды есть как в Dev-C++, так и в Microsoft Visual C++ .NET; в них также имеется возможность вызова этих команд с помощью меню и функциональных клавиш. В табл. 10.1 приведен список основных команд и клавиш их вызова.

Таблица 10.1. Команды отладчиков Microsoft Visual C++ .NET и Dev-C++

Команда	Visual C++	GNU C++ (`rhide`)
Отладка	<F5>	<F8>
Шаг внутрь (Step In)	<F11>	<Shift+F7>
Следующий шаг (Step Over)	<F10>	<F7>
Продолжить выполнения	<F5>	<Ctrl+F7>

Окончание табл. 10.1

Команда	Visual C++	GNU C++ (`rhide`)
Просмотр переменной (View Variable)	Только в меню	Только в меню
Установка точки останова (Set Breakpoint)*	<Ctrl+B>	<Ctrl+F5>
Добавить в наблюдаемые (Add watch)	Только в меню	<F4>
Перезагрузка программы (Program Reset)	<Shift+F5>	<Ctrl+Alt+F2>

* *Щелчок слева от строки исходного текста C++ в окне редактора представляет собой альтернативный путь установки точек останова.*

Запуск тестовой программы

Лучший способ исправить ошибки в программе — пройти ее пошагово. Приведенная ниже программа содержит несколько ошибок, которые надо найти и исправить. Эта программа имеется на прилагаемом компакт-диске в файле `Concatenate1.cpp`.

```
// Concatenate - конкатенация двух строк со вставкой " - "
//               между ними. В этой версии имеются ошибки.

#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <string.h>

using namespace std;
void stringEmUp(char* szTarget,
                char* szSource1,
                char* szSource2,
                int nLength);

int main(int nNumberofArgs, char* pszArgs[])
{
    cout << "Конкатенация двух строк со вставкой \" - \"\n"
         << "(В этой версии имеются ошибки.)" << endl;
    char szStrBuffer[256];

    // Создание двух строк одинаковой длины...
    char szString1[16];
    strncpy(szString1, "This is a string", 16);
    char szString2[16];
    strncpy(szString2, "THIS IS A STRING", 16);

    // ...и объединение их в одну
    stringEmUp(szStrBuffer,
               szString1,
               szString2,
               16);

    // Вывод результата
    cout << "<" << szStrBuffer << ">" << endl;

    // Пауза для того, чтобы посмотреть
    // на результат работы программы
```

```
    system("PAUSE");
    return 0;
}

void stringEmUp(char* szTarget,
                char* szSource1,
                char* szSource2,
                int nLength)
{
    strcpy(szTarget, szSource1);
    strcat(szTarget, " - ");
    strcat(szTarget, szSource2);
}
```

Соберите и запустите программу. Вместо объединения двух строк программа может вернуть все, что угодно. Нам надо обратиться к отладчику, чтобы разобраться, что же в этой программе не так.

Пошаговое выполнение программы

Первое, что стоит сделать при поиске ошибки с помощью отладчика, — это выполнить программу в отладочном режиме. Попытка выполнить эту программу в отладочном режиме в Dev-C++ (с помощью клавиши <F8>) приводит к появлению диалогового окна с сообщением об ошибке "Ваша программа вызвала нарушение доступа". Этой информации слишком мало, чтобы разобраться, в чем проблема.

Подобное сообщение об ошибке обычно говорит о некорректной работе с указателями того или иного типа.

Команда Остановить выполнение заставляет отладчик заново начать работу с программой (а не с того места, где вы находитесь). Никогда не вредно перезагрузить отладчик перед началом работы.

Для того, чтобы увидеть, где именно таится проблема, выполните только часть программы. Отладчик позволяет сделать это посредством так называемых *точек останова* (breakpoints). Отладчик всякий раз прекращает выполнение программы при прохождении через точку останова, и передает управление программисту.

Установим точку останова на первой выполнимой инструкции, щелкнув слева от строки вывода в `cout` или воспользовавшись клавишами <Ctrl+F5>, как сказано в табл. 10.1. При этом вы увидите появившийся маленький красный кружок, говорящий об установленной точке останова (рис. 10.2).

Теперь продолжим выполнение программы под отладчиком, либо выбирая команду меню Debug⇨Debug (Отладка⇨Отладка), либо щелчком на соответствующей пиктограмме в панели отладки, либо при помощи клавиши <F8>. Выполнение программы немедленно прекращается на первой же строке, а подсветка строки из красной делается синей, указывая, что выполнение программы заморожено на данной строке.

Теперь вы можете выбрать в меню команду Debug⇨Next Step (Отладка⇨Следующий шаг) либо нажать клавишу <F7> для выполнения одной строки программы. Синяя подсветка перемещается к следующей выполнимой инструкции, пропуская два объявления переменных. (Объявления не являются выполнимыми командами; они всего лишь выделяют память для объявляемых переменных.) Такое выполнение одной инструкции C++ называется *пошаговым выполнением программы*. Вы можете переключиться в окно консоли программы и посмотреть, что именно вывела программа при выполнении этой инструкции (рис. 10.3).

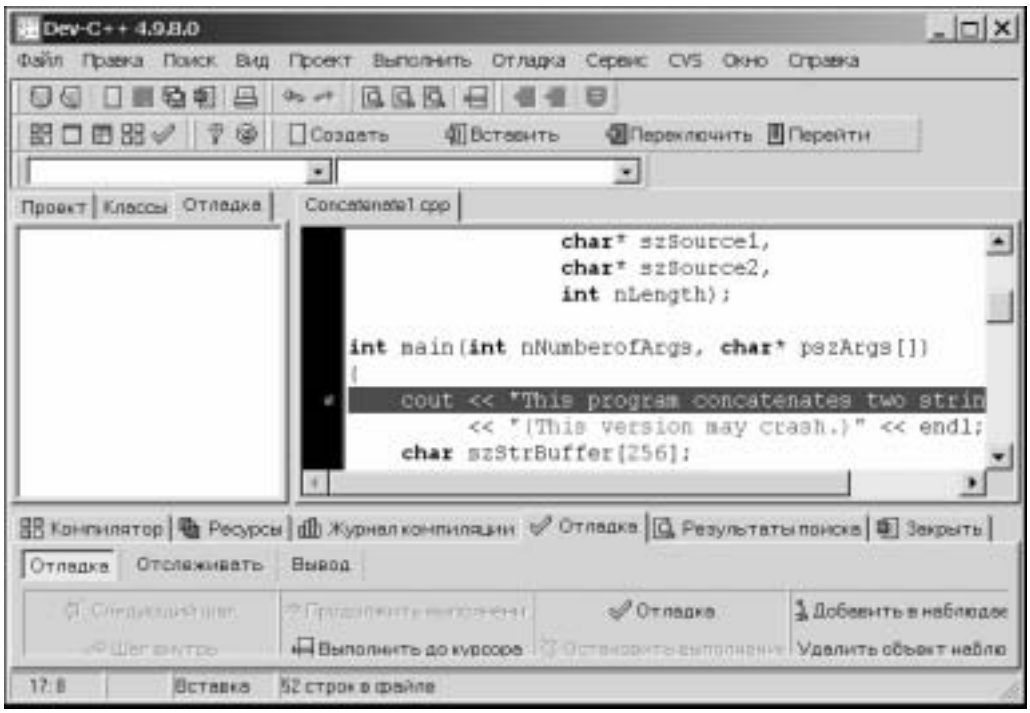

Рис. 10.2. Точку останова легко опознать по маленькому красному кружку

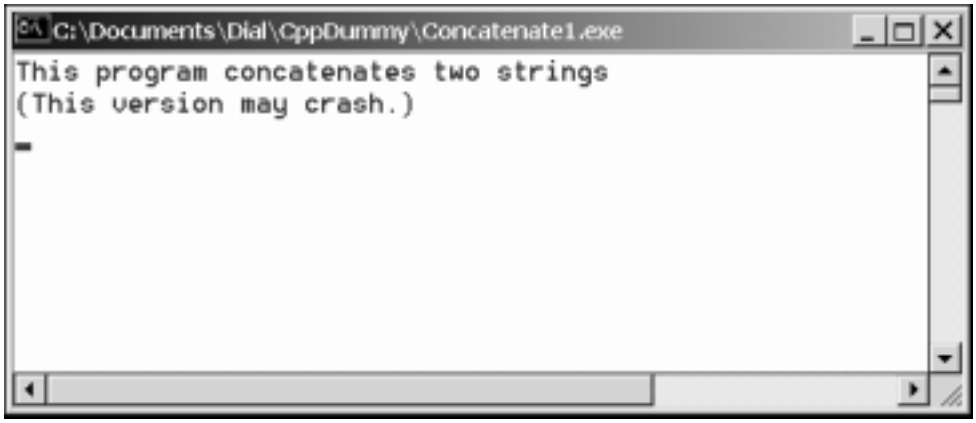

Рис. 10.3. В любой момент вы можете переключиться на окно выполняемой программы

Выполнение двух последующих инструкций приводит нас к вызову функции `StringEmUp()`. Если опять выбрать команду Debug⇨Next Step (Отладка⇨Следующий шаг), программа аварийно завершится. Теперь мы знаем, что проблема кроется в этой функции.

Если сбой происходит при вызове некоторой функции, то либо ошибка содержится в коде функции, либо ей передаются некорректные аргументы.

Команда Debug⇨Next Step (Отладка⇨Следующий шаг) рассматривает вызов функции как единую команду. Однако на самом деле функция состоит из ряда отдельных инструкций C++, и для отладки нам надо пройти их пошагово. Такая функциональность обеспечивается командой Debug⇨Step Into (Отладка⇨Шаг внутрь).

Перегрузите программу при помощи пункта меню Debug⇨Program Reset (Отладка⇨Остановить выполнение) либо соответствующей пиктограммы на панели отладки или клавиш <Ctrl+Alt+F2>. Теперь, чтобы сэкономить время, отключите установленную точку останова, щелкнув на красном кружке, и установите новую на вызове функции, как показано на рис. 10.4.

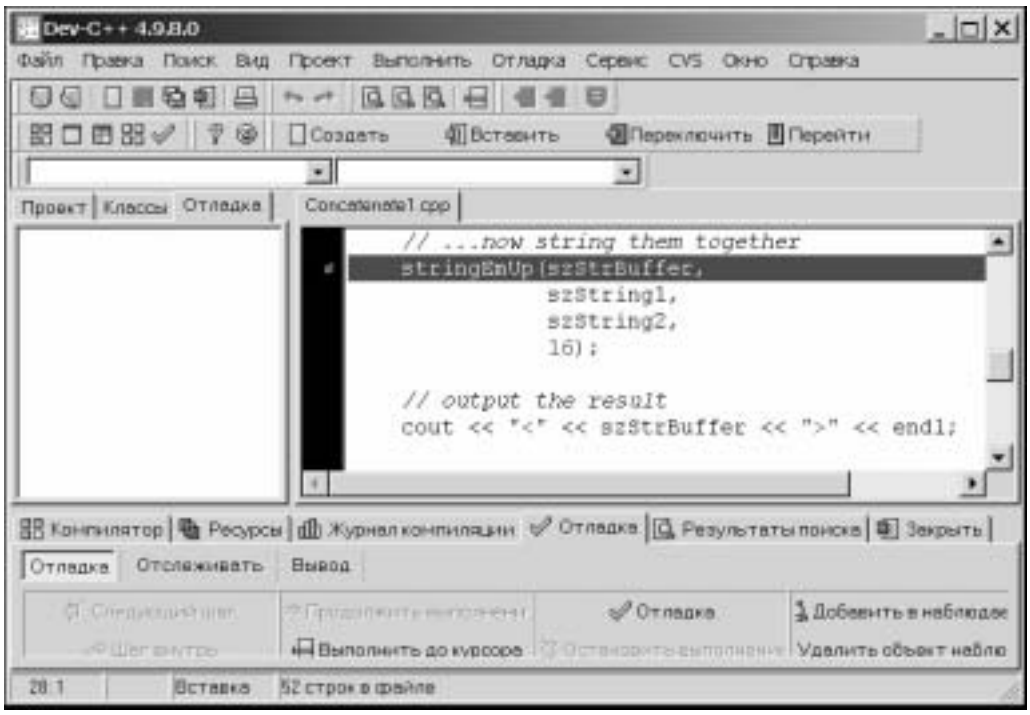

Рис. 10.4. Установка точки останова на вызове функции `StringEmUp()`

Вы можете установить в программе столько точек останова, сколько вам требуется.

Теперь перезапустите программу на выполнение, и она остановится на вызове функции `StringEmUp()`. Войдите в функцию, воспользовавшись командой Debug⇨Step Into (Отладка⇨Шаг внутрь), как показано на рис. 10.5.

Допустим, вы обнаружили, что ваша программа время от времени работает некорректно. Чтобы лучше понять, почему это происходит, желательно знать, какие аргументы передаются в рассматриваемую функцию. Для этого нужна функция наблюдения за переменными, предоставляемая отладчиком, которая позволяет ознакомиться с содержимым всех переменных при каждом останове выполнения. Проще всего установить наблюдение за переменной, выбрав ее в окне редактора и нажав клавишу <F4>. На рис. 10.6 показано, как выглядят четыре аргумента рассматриваемой функции.

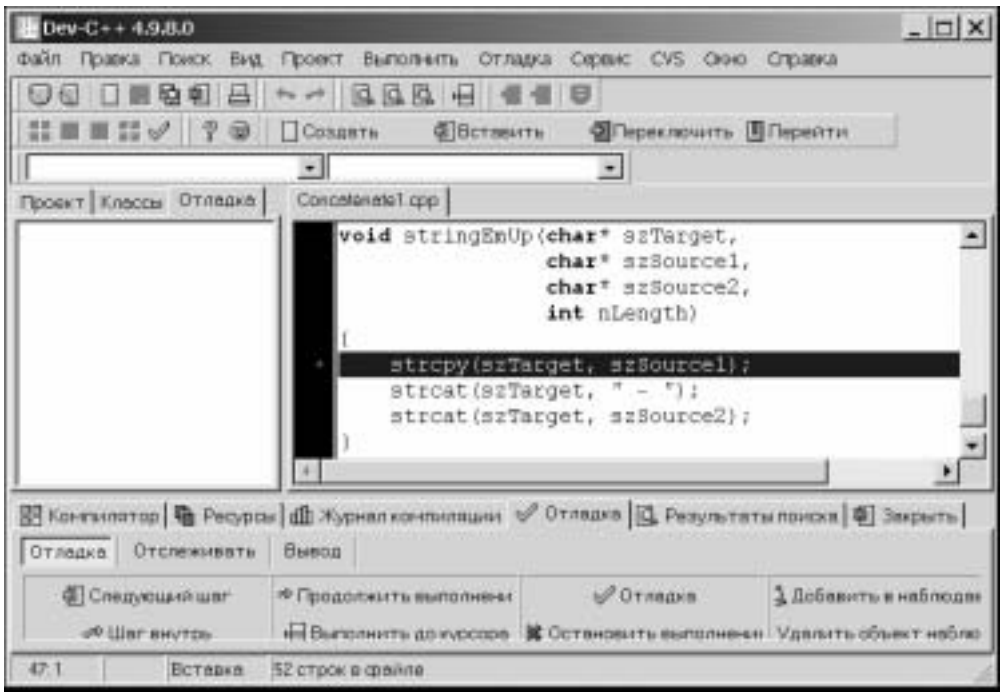

Рис. 10.5. Команда Debug⇨Step Into (Отладка⇨Шаг внутрь) позволяет выполнить вызов функции пошагово

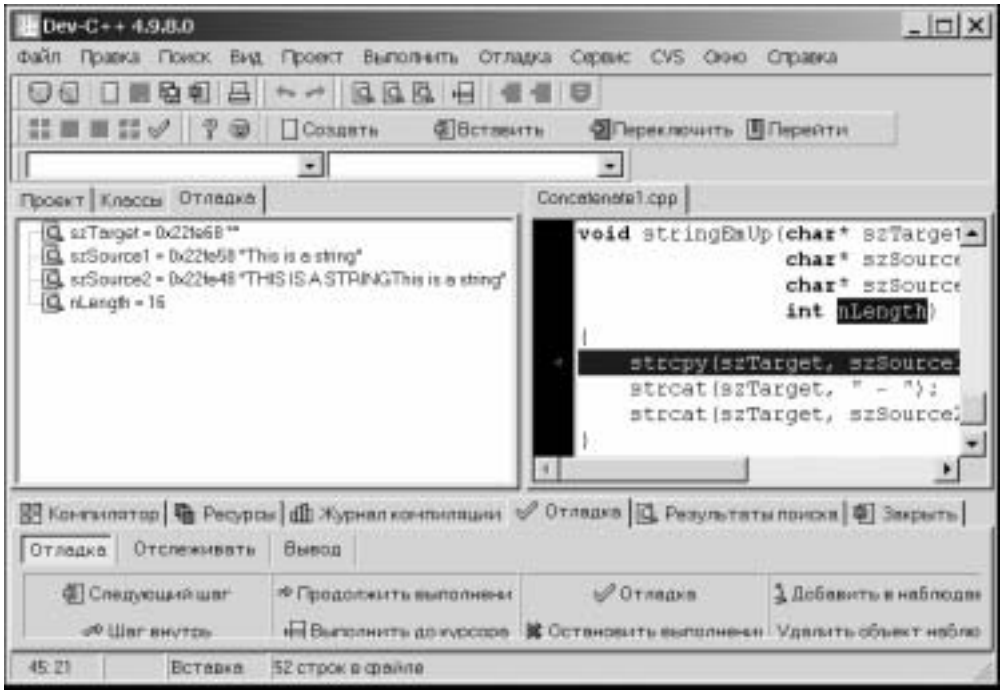

Рис. 10.6. Отладчик позволяет следить за значениями переменных

Числа возле имен переменных в окне отладки — адреса, которые в данном случае малоинформативны. Строка `szTarget` пока что пуста, что вполне закономерно, так как мы еще ничего в нее не скопировали. Значение строки `szString1` также выглядит вполне корректно, но вот значение `szString2` содержит сразу две строки — и “This is a string”, и “THIS IS A STRING”, чего вроде бы быть не должно.

Ответ находится в четвертой переменной. Дело в том, что длина этих двух строк не 16 символов, а 17! Программа не выделила память для завершающего нуля, что и приводит к сбою при выполнении функции `StringEmUp()`.

Длина строки всегда включает завершающий нулевой символ.

Давайте изменим программу, исправив ошибку. Пусть теперь C++ сам рассчитывает размер строк. Получившаяся в результате программа `Concatenate2.cpp`, приведенная ниже, работает вполне корректно.

```
// Concatenate - конкатенация двух строк со вставкой " - "
//               между ними.

#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <string.h>

using namespace std;
void stringEmUp(char* szTarget,
                char* szSource1,
                char* szSource2);

int main(int nNumberofArgs, char* pszArgs[])
{
    cout << "Конкатенация двух строк со вставкой \" - \"\n"
         << "(В этой версии нет ошибки.)" << endl;
    char szStrBuffer[256];

    // Определение двух строк...
    char szString1[] = "This is a string";
    char szString2[] = "THIS IS A STRING";

    // ...и объединение их в одну
    stringEmUp(szStrBuffer,
               szString1,
               szString2);

    // Вывод результата
    cout << "<" << szStrBuffer << ">" << endl;

    // Пауза для того, чтобы посмотреть
    // на результат работы программы
    system("PAUSE");
    return 0;
}
```

```
void stringEmUp(char* szTarget,
                char* szSource1,
                char* szSource2)
{
    strcpy(szTarget, szSource1);
    strcat(szTarget, " - ");
    strcat(szTarget, szSource2);
}
```

Вот вывод этой программы — именно такой, какой мы и ожидали:

```
Конкатенация двух строк со вставкой " - "
(В этой версии нет ошибки.)
<This is a string - THIS IS A STRING>
Press any key to continue...
```

Поздравляю! Вам удалось отладить программу.