В.Л. Тарасов

Лекции по программированию на C++

Лекция 14

Ввод и вывод

Ввод и вывод в языке C++ обеспечиваются стандартной библиотекой с заголовочным файлом `iostream`, все имена которой входят в пространство имен `std`. В C++ можно также применять все средства ввода и вывода языка Си, которые объявлены в заголовочном файле `stdio.h`.

Средства ввода и вывода языка C++ реализованы в виде иерархии *потоковых* классов. Поток – это последовательность байтов. С точки зрения программы поток не зависит от тех конкретных устройств, с которыми ведется обмен данными. Поток может быть связан со стандартным устройством ввода (клавиатурой), стандартным устройством вывода (дисплеем), с файлом на диске или с массивом в оперативной памяти. Потоки бывают *входными*, *выходными* и *двунаправленными*, допускающими как чтение, так и запись. В составе библиотеки ввода и вывода предусмотрены средства для включения данных в поток и извлечения данных из потока.

В стандартной библиотеке есть реализации потоковых классов, ориентированные на однобайтовые символы `char` и символы типа `wchar_t`, размер которых, согласно стандарту языка C++, составляет не менее 2 байтов. Далее рассматриваются потоковые классы, ориентированные на `char`.

Все потоковые классы имеют в качестве базового класс `ios`. Для непосредственной организации ввода и вывода используются классы:

`istream` – класс входных потоков;
`ostream` – класс выходных потоков;
`iostream` – класс двунаправленных потоков.

14.1. Вывод

В классе `ostream` есть функции-операторы `<<` для вывода стандартных типов. В качестве результата данные функции возвращают

ссылку на поток, в который делают вывод. Объявления функций-операторов << в классе ostream выглядят, примерно, так, как показано ниже:

```
class ostream : public ios
{
...
public:
  ostream& operator<< (const char*);  // Вывод строки
  ostream& operator<< (int);          // Вывод целого
  ostream& operator<< (long);         // Вывод длинного целого
  ostream& operator<< (double);       // Вывод double
  ostream& operator<< (long double);  // Вывод long double
  ostream& put(char);                 // Вывод символа
...
};
```

В файле iostream объявлен объект cout класса ostream, связанный по умолчанию со стандартным выходным устройством (дисплеем) – это стандартный выходной поток. Его можно перенаправить вместо экрана на файл средствами операционной системы.

Для ускорения выполнения операций ввода и вывода могут создаваться буфера в оперативной памяти, в которые предварительно помещаются выводимые или вводимые данные. В частности, поток cout имеет буфер.

Для вывода сообщений об ошибках определен объект cerr типа ostream, всегда связанный со стандартным выходным устройством. Он не имеет буфера, то есть сообщение, направленное в cerr, немедленно появляется на экране.

Благодаря тому, что функции operator<<() возвращают *ссылку на поток*, можно писать цепочки вызовов вида:

```
double x = 3.14;
cout << "\nx = " << x;
```

Последняя инструкции является сокращенной записью инструкции:

```
cout.operator<<("\nx = ").operator<<(x);
```

Здесь сначала вызывается функция cout.operator<<("\nx = "), которая выводит в поток cout строку "\nx = " и возвращает в качестве результата cout. Таким образом, перед второй точкой будет снова поток cout, в который функция cout.operator<<(x) выводит x.

14.2. Ввод

В классе входных потоков istream определены функции-операторы для ввода стандартных типов, объявления которых выглядят, примерно, так:

```
class istream : public ios
{
...
public:
  istream& operator>> (short& h);         // Ввод короткого целого h
  istream& operator>> (int& n);           // Ввод целого n
  istream& operator>> (long& ln);         // Ввод длинного целого ln
  istream& operator>> (float& f);         // Ввод f
  istream& operator>> (double& d);        // Ввод d
  istream& operator>> (long double& ld);  // Ввод ld
...
};
```

Аргументом этих функций является *ссылк*а на переменную, которой присваивается значение при вводе. Оператор >> ("взять из…") при чтении сначала пропускает все ведущие пробелы, затем извлекает символы для вводимого значения, пока не встретит разделитель или недопустимый для вводимой величины символ. Разделителями данных при вводе являются пробельные символы.

Для организации ввода со стандартного входного устройства в заголовке iostream объявлен объект cin класса istream, связанный по умолчанию с клавиатурой.

Функции-операторы >> возвращают ссылку на поток, для которого вызываются. Это позволяет строить цепочки вида:

```
cin >> i >> j >> k;
```

Данная инструкция является краткой записью инструкции:

```
cin.operator>>(i).operator>>(j).operator>>(k);
```

При наборе на клавиатуре 1 2 3 и нажатии клавиши Enter, переменные i, j, k получат соответствующие значения.

14.3. Ввод и вывод определяемых пользователем типов

Операторы вывода << и ввода >> можно переопределять для пользовательских типов. Эти операторы являются бинарными и требуют двух операндов. В стандартной библиотеке ввода и вывода предусмотрено, что их первым операндом является поток, поэтому,

чтобы сохранить привычный способ их применения для пользовательских типов (классов), нужно при их перегрузке в качестве первого операнда также указывать поток. Поэтому эти операторы не могут быть членами пользовательских классов.

Программа 14.1. Перегрузка операторов ввода/вывода

```
// Файл UserIO.cpp

#include <iostream>
#include <locale>
using namespace std;

class Point3{                                  // Точка в трехмерном пространстве
   float x, y, z;                              // Три координаты
   friend ostream& operator<<(ostream&, const Point3&);    // Оператор
                                                           // вывода
   friend istream&  operator>>(istream& , Point3&);        // Оператор ввода
};
```

Функции-операторы operator>>() и operator<<() объявлены друзьями класса Point3, чтобы они имели доступ к закрытым членам этого класса. Далее следуют определения функций-операторов ввода и вывода.

```
// Оператор вывода для Point3

ostream& operator<<(ostream& outs, const Point3& d)
{
  // Вывод данных в поток outs
   outs << "(" << d.x << ", " << d.y << ", " << d.z << ")";
   return outs;                                  // Возвращение потока
}

// Оператор ввода для Point3

istream& operator>>(istream& ins, Point3& d)
{
  // Чтение данных из потока ins и возвращение потока
  return ins >> d.x >> d.y >> d.z;
}
```

Здесь использован тот факт, что стандартный оператор ввода >> возвращает поток, для которого вызывается, то есть ins. После того как из потока ins будет произведено чтение, он возвращается из функции как результат ее работы. Теперь вводить и выводить данные типа Point3 можно так же, как данные стандартных типов:

```
void main()
{
```

```
  setlocale(LC_ALL, "Russian");
  Point3 F, G;           // Две точки с неопределенными координатами
  cout << "\n Координаты точек после их создания:\nF = "
       << F << "\nG = " << G << endl;
  cout << "Введите координаты двух точек: ";
  cin >> F >> G;
  cout<<"Теперь координаты точек: F= " << F << ", G = " << G << endl;
}
```

Так как в классе `Point3` нет конструктора, объекты `F` и `G` после создания будут иметь неопределенное значение. Программа выдает:

```
Координаты точек после их создания:
F = (-1.07374e+008, -1.07374e+008, -1.07374e+008)
G = (-1.07374e+008, -1.07374e+008, -1.07374e+008)
Введите координаты двух точек: 1 2 3 4 5 6
Теперь координаты точек: F= (1, 2, 3), G = (4, 5, 6)
```

14.4. Работа с файлами

Файлом называется поименованная часть памяти диска, содержащая некоторый набор записей. Непосредственную работу с файлами выполняет операционная система. В языках программирования обычно имеются высокоуровневые средства для доступа к файлам. В C++ они объявлены в заголовочном файле `fstream`.

Источник или приемник данных называется *поток*. Примерами потоков являются стандартные потоки для ввода `cin` и для вывода `cout`. Для чтения из файла или записи в файл создаются потоки, связанные с файлами. Существуют два вида потоков: *текстовые* и *бинарные*.

Текстовый поток – это последовательность символьных строк, заканчивающихся символом «новая строка» - `'\n'`. При выводе перед каждым символом `'\n'` помещается символ возврата каретки `'\r'`. При вводе происходит обратное преобразование – два символа `'\r'` и `'\n'` заменяются одним символом `'\n'`. Необходимость такого преобразования вызвана различиями в фиксации признака новой строки в C++ и в операционной системе. В языке C++ новая строка обозначается единственным символом `'\n'`, реально же на диске в файле хранятся оба символа `'\r'` и `'\n'`, генерируемые нажатием клавиши **Enter**, которая используется при завершении ввода очередной строки с помощью клавиатуры.

Бинарный поток – это последовательность байтов, которая не подвергается какому-либо преобразованию.

При каждом пуске программы открыты три *стандартных потока*:

`cin` – стандартный ввод,
`cout` – стандартный вывод,

cerr – стандартный приемник ошибок.

По умолчанию стандартный ввод связан с клавиатурой, стандартный вывод – с экраном дисплея. Стандартный ввод, и вывод можно перенаправить на файлы средствами операционной системы. Эти потоки имеют буфера, в которые предварительно направляются данные. Это позволяет ускорить ввод и вывод. Поток cerr не имеет буфера, его нельзя перенаправить на файл. Этот поток всегда связан со стандартным устройством вывода, обычно это экран монитора. -.

В заголовочном файле fstream объявлены три потоковых класса, обеспечивающие работу с файлами:

ofstream – класс выходных файловых потоков;
ifstream – класс входных файловых потоков;
fstream – класс двунаправленных файловых потоков.

Для работы с файлами в программе нужно определить переменные этих классов, например,

```
ifstream fin;        // fin  - входной файловый поток
ofstream fout;       // fout - выходной файловый поток
```

Связь потока с файлом можно установить с помощью функции-члена open(), с двумя аргументами. Первый аргумент обязателен, он задает имя открываемого файла. Второй аргумент не является обязательным, он задает режим работы с файлом. Если второй аргумент на задан, файлы считаются текстовыми. Например, следующие инструкции открывают два текстовых файла на чтение и запись:

```
fin.open("InpData.txt"); // Файл InpData.txt открывается для чтения
fout.open("ResData.txt");// Файл ResData.txt открывается для записи
```

В качестве второго аргумента функции open() можно использовать константы (называемые флагами), определенные в классе ios:

ios::out – файл используется для вывода;
ios::in – файл используется для ввода;
ios::app – файл открывается для добавления в конец файла;
ios::binary – файл открывается как бинарный.

Данные флаги представляют из себя целые числа, все разряды которых нули, кроме одного разряда, поэтому их можно комбинировать с помощью операции логического или (|).

Например, файлы InpData.txt и ResData.txt можно открыть соответственно на чтение и запись инструкциями:

```
fin.open("InpData.txt", ios::in);      // Файл открывается для чтения
```

```
fout.open("ResData.txt", ios::out );  // Файл открывается для записи
```

Для проверки успешности завершения функции open() к потоку можно применить оператор !, который возвращает ненулевое значение при наличии ошибки. Например, проверить, что файл InpData.txt удалось открыть, можно следующим образом:

```
if( !fin != 0 ){
  cerr << "Не могу открыть файл " << "InpData.txt";
  exit(1);                    // Завершение работы программы
}
```

По умолчанию потоки открываются на чтение и запись как текстовые. О работе с бинарными файлами говорится в следующем параграфе. Для текстовых файловых потоков применимы все рассмотренные ранее операторы и функции ввода и вывода.

Открытые файловые потоки закрываются автоматически при завершении программы или при выходе из функции, в которой они были созданы. Поток можно явно закрыть, вызвав для него функцию close(), например,

```
fin.close();
```

Программа 14.2. Копирование файлов

В приводимой далее программе открываются два файловых потока - один на чтение, другой на запись. Из входного файла производится построчное чтение, прочитанные строки выводятся на экран и в выходной файл. После чтения каждых 20 строк программа останавливается и продолжает выполняться после нажатия клавиши **Enter**.

```
// Файл CopyFile.cpp
// Файловый ввод и вывод

#include <fstream>
#include <iostream>
#include <string>
#include <cstdlib>                // Для exit()
using namespace std;

const int PAGE = 20;              // Максимальное число строк на странице

void main()
{
    setlocale(LC_ALL, "Russian");
    ifstream fin;                 // Файловый поток для ввода
    ofstream fout;                // Файловый поток для вывода
    string filename;              // Строка для имени файла
    string line;                  // Строка из файла
```

```
    cout << "Введите имя входного файла ";
    getline(cin, filename);      // Ввод имени входного файла
    fin.open(filename.c_str());
    if( !fin ){                  // Если не удалось открыть входной файл
        cerr << "Не удалось открыть файл " << filename;
        system("pause");         // Ожидает нажатия любой клавиши
        exit(1);                 // Завершение программы
    }
    cout << "Введите имя выходного файла ";
    getline(cin, filename);
    fout.open(filename.c_str());
    if(!fout){                   // Если не удалось открыть выходной файл
        cerr << "Не удалось открыть файл " << filename;
        system("pause");
        exit(1);
    }
    int i = 0;                   // Количество прочитанных строк
    while( !fin.eof() ){         // Пока не достигнут конец файла
        getline(fin, line);      // Чтение строки из файла.
        i++;                     // Увеличение счетчика
        cout << line << '\n';    // Вывод строки на экран
        fout << line << '\n';    // Вывод строки в файл
        if( i % PAGE == 0 )      // Если выведена очередная страница,
            cin.get();           // ждем нажатия клавиши Enter
    }
    fin.close();                 // Закрытие входного файла
    fout.close();                // Закрытие выходного файла
    system("pause");
}
```

Ввод имени файла выполняется вариантом функции `getline()`:

```
getline(cin, filename);
```

Первый аргумент - входной поток `cin`, из которого производится чтение, второй аргумент – строка `filename` типа `string`, в которую помещается введенное имя файла. Функция `getline()` может прочитать из потока произвольную строку, в том числе строку, состоящую из нескольких слов, разделенных пробельными символами. Чтение набранных на клавиатуре символов начинается после нажатия клавиши **`Enter`**. В строку включаются все прочитанные символы кроме символа новой строки, который извлекается из входного потока, но в строку не включается.

Затем файл открывается на чтение:

```
fin.open(filename.c_str());
```

В функцию `open()` нужно передавать имя файла в виде символьного массива, завершаемого нулевым байтом. Указатель на такой массив получается из строки типа `string` функцией `c_str()`.

При открытии файла возможна неудача, например, из-за неправильно указанного имени, в этом случае программа завершается вызовом `exit()`.

Если входной файл удалось открыть, запрашивается имя выходного файла, и он открывается на запись инструкцией

```
fout.open(filename.c_str());
```

Если открываемый на запись файл не существовал, он создается, если же файл существовал, его старое содержимое уничтожается.

При открытии выходного файла на запись также возможна неудача, например, потому, что файл с указанным именем существует и открыт другой программой. В этом случае также вызывается функция `exit()`, которая корректно завершит программу, закрыв ранее открытый входной файл.

Для тестирования программы создадим в папке `C:\Tmp` текстовый файл `numbabc.txt` с произвольным содержанием. Далее приведен диалог с программой:

```
Введите имя входного файла C:\Tmp\numbabc.txt
Введите имя выходного файла C:\Tmp\copynumbabc.txt
1
12
123
12345
123456
abcdef
abcde
abcd
ab
a
```

Убеждаемся, что в папке `C:\Tmp` появился файл `copynumbabc.txt`, содержащий то же, что и файл `numbabc.txt`. Их содержимое также выведено на экран.

Для файлов, находящихся в папке проекта, достаточно указывать только имя, путь можно не задавать.

14.5. Работа с бинарными файлами

Связать файловые переменные с конкретными файлами на диске можно, передавая конструктору имена файлов и режим работы с ними, например, в приводимой долее программе 14.3 инструкция

```
ofstream textout(NameTextFile, ios::out);
```

создает текстовый файловый поток textout, связывает его с файлом, имя которого находится в строке NameTextFile и открывает его на запись.

В этой же программе инструкция

```
ofstream binout(NameBinFile, ios::out | ios::binary);
```

создает бинарный выходной поток binout (флаги ios::binary и ios::out), связанный с файлом, имя которого указано в строке NameBinFile. Здесь использована возможность комбинировать флаги, задающие режимы работы с файлами, с помощью побитового логического оператора ИЛИ (|).

Для чтения из бинарного потока применяют функцию:

```
int read(char *pc, size_t count);
```

которая читает из потока count байтов и сохраняет их в массиве, начало которого указывает pc.

Для записи в бинарный поток служит функция:

```
int write(const char *pc, size_t count);
```

направляющая в поток count байтов из массива, на который указывает pc.

Позиции в файле нумеруются с нуля, поэтому бинарный поток подобен массиву байтов. С каждым файловым потоком связан текущий указатель на байт, который будет прочитан или записан при следующей операции ввода/вывода. Положением текущего указателя в *выходном* потоке можно управлять с помощью функции:

```
ostream& seekp(long offs, seek_dir dir);
```

Текущую позицию во *входном* потоке можно изменить функцией

```
istream& seekg(long offs, seek_dir dir);
```

Первый параметр offs функций seekp() и seekg() задает число позиций, на которое надо переместить текущий указатель; второй параметр dir назначает точку отсчета, от которой надо произвести смещение. Для указания точки отсчета можно использовать перечислимые константы из файла iostream:

```
enum seek_dir { beg, cur, end };
```

beg – смещаться от начала файла,
cur – смещаться от текущей позиции,
end – смещаться от конца файла.

Например, если файловый поток f открывается на чтение и запись, то текущий указатель перемещается в начало файла инструкцией:

```
f.seekg(0, ios::beg);
```

Программа 14.3. Сравнение текстового и бинарного файлов

В программе создаются два выходных файловых потока: текстовый и бинарный. Потоки связываются с файлами конструктором, которому в качестве аргумента передаются имена файлов. В файлы записываются случайные целые числа соответственно в текстовом и бинарном виде. Созданные файлы закрываются и открываются вновь как бинарные, для передачи функции `FileSize()`, определяющей их размеры. Чтобы прочитать содержимое файла, хранящего текстовое представление чисел, он сначала закрывается как бинарный, а затем открывается на ввод как текстовый. Файл с бинарным представлением чисел подготавливается к чтению перемещением текущего указателя к началу файла. Содержимое файлов выводится на экран.

```
// Файл CompareTextAndBnaryFiles.cpp

#include <cstdlib>
#include <fstream>
#include <iostream>
#include <ctime>
using namespace std;

long FileSize(istream& f)     // Возвращает размер файла в байтах
{                             // f – ссылка на файловый поток
    long k;                   // Размер файла в байтах
    f.seekg(0, ios::end);     // Поместить текущий указатель в конец файла
                              // 0 - величина смещения
                              // ios::end - позиция, относительно которой
                              // делается смещение
    k = f.tellg();            // Функция tellg() возвращает
                              // текущую позицию в файле
    return k;
}

int main()
{
    setlocale(LC_ALL, "Russian");
    const char* NameTextFile = "TxtFile.txt";     // Имя текстового файла
    const char* NameBinFile  = "BinFile.bin";     // Имя бинарного файла
    const int N  = 20;                // Количество чисел в файле
    const int NL = 12;                // Количество чисел на одной строке
    // Создание выходного текстового потока
    ofstream textout(NameTextFile, ios::out);
    if(!textout){                                 // Проверка
        cout<< "Не могу создать файл" << NameTextFile;
        cin.get();                    // Ждем нажатия Enter
        exit(1);
    }
```

```
    // Создание выходного бинарного потока
    ofstream binout(NameBinFile, ios::out | ios::binary);
    if(!binout){                                    // Проверка
        cout<< "Не могу создать файл" << NameBinFile;
        cin.get();
        exit(1);
    }
    int i, nmb;
    srand((unsigned)time(0));    // Инициализация датчика случайных чисел
    for(i = 0; i < N; i++){
        nmb = rand();                               // Генерируем числа
        textout << ' '  << nmb;                     // и заносим в текстовый
        binout.write((char*)&nmb, sizeof(int));     // и бинарный файлы
    }
    textout.close();                                // Закрываем
    binout.close();                                 // потоки
// Открываем оба файла на чтение как бинарные
    ifstream tin(NameTextFile, ios::binary);        // Бинарный поток
    ifstream bin(NameBinFile,  ios::binary);
    cout << "Размер текстового файла " << NameTextFile
        << " = " << FileSize(tin);
    cout << "\nРазмер бинарного файла " << NameBinFile
        << " = " <<  FileSize(bin);
    tin.close(); // Закрываем бинарный поток, связанный с текстовым файлом
    tin.open(NameTextFile);       // и окрываем файл как текстовый поток
    if(!tin){
        cout << "\nНе могу открыть файл " << NameTextFile;
        cin.get();
        exit(1);
    }
    // Вывод чисел из обоих файлов
    cout << "\nЧисла из файла " << NameTextFile << endl;
    i = 0;
    while(!tin.eof()){                  // Пока не достигнут конец файла,
        tin >> nmb;                     // читаем число из файла
        i++;                            // Счетчик чисел
        cout << nmb << ' ';             // выводим на экран
        if(i % NL == 0)                 // Если строка заполнена,
            cout << endl;               // переходим на новую строку
    }
    bin.seekg(0, ios::beg);             // Перемещаем текущий указатель файла
                                        // к его началу
    cout << "\nЧисла из файла " << NameBinFile << endl;
    i = 0;
    while(!bin.eof()){                  // Пока не достугнут конец файла,
        bin.read((char*)&nmb, sizeof(int));  // Читаем байты числа
        i++;
        cout << nmb << ' ';             // Выводим число
        if(i % NL == 0)
            cout << endl;
    }
    tin.close();                        // Закрываем
    bin.close();                        // потоки
```

```
    cin.get();
    return 0;
}
```

Для записи в текстовый файл и чтения из него использованы обычные операторы вывода (<<) и ввода (>>). Для записи в бинарный файл использована функция `write()`, для чтения – `read()`.

Далее приводятся результаты, выводимые программой. В тестовом файле числа хранятся в виде последовательностей десятичных цифр, разделенных пробелами. Если открыть файл `TxtFile.txt`, то увидим в нем числа в виде одной длинной строки. Если открыть бинарный файл в текстовом редакторе, то увидим непрерывную последовательность непонятных значков.

Размер бинарного файла равен 80 байт, так как в него записано 20 целых чисел, каждое их которых занимает 4 байта. Размер тестового файла оказался больше, так как многие числа пятизначные, кроме того между числами есть пробелы.

```
Размер текстового файла TxtFile.txt = 112
Размер бинарного файла BinFile.bin = 80
Числа из файла TxtFile.txt
17102 18167 7088 9561 11921 10043 11132 4722 1810 24216 25096 31524
16702 1940 17362 5570 11945 4035 8499 26135
Числа из файла BinFile.bin
17102 18167 7088 9561 11921 10043 11132 4722 1810 24216 25096 31524
16702 1940 17362 5570 11945 4035 8499 26135
```

14.6. Командная строка

Командная строка позволяет взаимодействовать с операционной системой путем ввода различных команд.

В Windows окно командной строки запускается приложением `cmd.exe` (рис.14.1). Данное приложение является командным интерпретатором, выполняющим команды, вводимые в командной строке.

Командная строка содержит приглашение, содержащее название текущего каталога (рис.14.1):

```
C:\Windows\System32>
```

после которого вводятся команды, выполняемые командным интерпретатором.

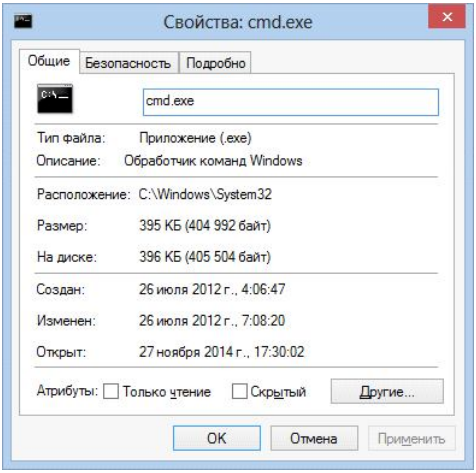

Рис. 14.1. Расположение и свойства командного интерпретатора

На рис.14.2 показано окно командной строки после выполнения команды, `cd`, которая изменяет текущий каталог:

```
cd /d C:\MyPrograms\CopyFiles\Debug
```

Параметрами команды является ключ `/d`, который необходим, если новый текущий каталог находится на другом диске, и полное имя каталога, который делается текущим.

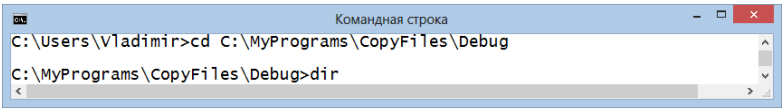

Рис. 14.2. Команда перехода в новый каталог

Команда `dir` выводит на экран содержимое текущего каталога (рис.14.3).

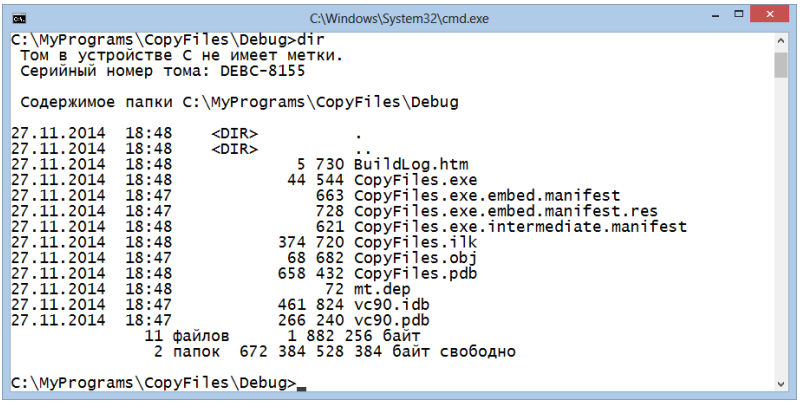

Рис. 14.3. Вывод команды `dir`

Для запуска программы надо набрать ее имя в командной строке (рис.14.4) и нажать **Enter**. Расширение файла `.exe` вводить необязательно. В рассматриваемом случае для входного файла указано полное имя, содержащее путь к нему, а для выходного файла задано только имя, поэтому выходной файл `copy_1_numbabc.txt` создается в текущей папке.

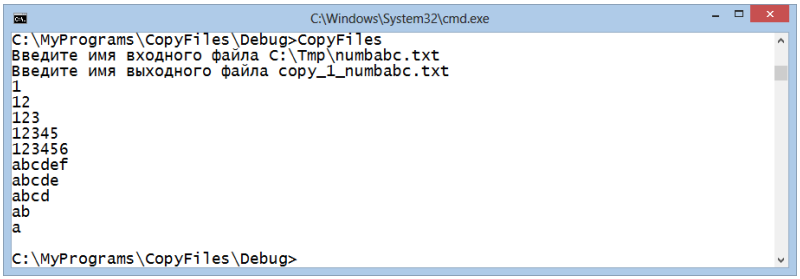

Рис. 14.4. Запуск программы из командной строки

Убедимся, что файл `copy_1_numbabc.txt` создан в текущей папке, выполнив команду:

```
dir *.txt
```

Здесь звездочка (*) заменяет любое имя файла, поэтому команда выведет имена всех файлов с расширением `.txt` (рис.14.5)

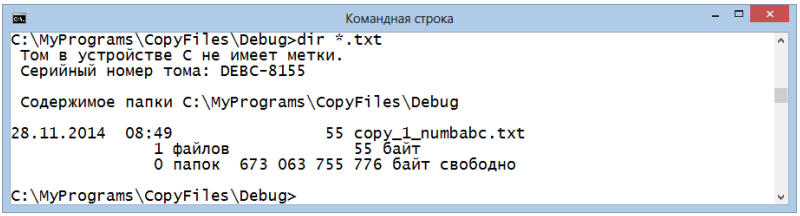

Рис. 14.5. Вывод имен текстовых файлов

Содержимое любого файла можно вывести на экран командой `type`, например,

```
type copy_1_numbabc.txt
```

Список команд, доступных в командной строке, выводится командой `help`.

14.7. Аргументы командной строки

При запуске программы на выполнение функция `main()` может получить из операционной среды два аргумента. Для доступа к ним `main()` должна быть объявлена с двумя формальными параметрами в виде:

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

Первый аргумент `argc` имеет целый тип и равен общему числу аргументов, заданных в командной строке при запуске программы. Аргументами командной строки являются имя программы и другие слова, набранные в командной строке, отделяемыми друг от друга пробелами или табуляциями.

Второй аргумент `argv` является массивом указателей на `char`. Эти указатели содержат адреса строк, в которые помещаются слова из командной строки.

Программа 14.4. Эхо аргументов командной строки

Программа выводит строки, указатели на которые хранятся в массиве `argv`.

```
// Файл Echo.cpp

#include <iostream>
using namespace std;

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

```
   for(int i = 1; i < argc; i++)      // Перебор аргументов командной строки
      cout << argv[i] <<  (i < argc - 1 ? " " : "\n");
   _getch();
   return 0;
}
```

Выражение `i < argc - 1 ? " " : "\n"` равно строке из одного пробела `" "`, либо строке из одного символа новая строка `"\n"`, когда `i` равен номеру последнего аргумента `argc - 1`. Это обеспечивает вывод пробела между аргументами и переход на новую строку после вывода последнего аргумента.

Проект был назван `Progr_14_04_Echo`, в результате его построения была создана программа `Progr_14_04_Echo.exe` в папке `Debug`, расположенной в папке проекта.

Запустим режим командной строки, как описано в предыдущем параграфе, перейдем в папку `Debug` и запустим программу командной строкой:

```
…>Progr_14_04_Echo.exe Hello,  World!
```

программа выведет:

```
Hello,  World!
```

Рассмотрим работу программы. Первым аргументом `argv[0]` является имя программы, поэтому всегда `argc >= 1`. Если `argc == 1`, то в командной строке после имени программы никаких аргументов нет. В нашем примере будет `argc == 3` и, соответственно, `argv[0]`, `argv[1]`, `argv[2]` – это указатели на строки: `"Progr_14_04_Echo.exe"`, `"Hello,"`, `"World!"`. Согласно стандарту `argv[argc] == 0`, то есть является пустым указателем.

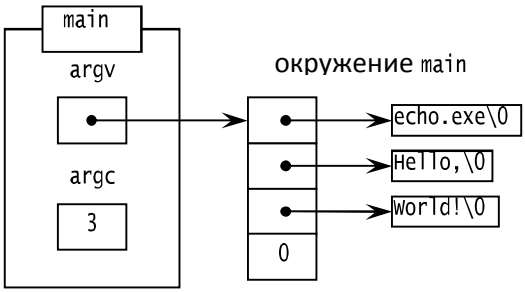

Рис. 14.6. Размещение в памяти аргументов командной строки

Схема памяти показана на рис.14.6. Внутри `main()` имеется только указатель `argv`, в котором хранится адрес нулевого элемента из массива

указателей, расположенного вне main(). А уже в этом внешнем по отношению к main() массиве указателей хранятся адреса строк с аргументами командной строки.

Программа 14.5. Печать строк, содержащих образец

Программа определяет вводимые строки, которые содержат образец, заданный в командной строке и выводит их. Ввод производится из файла, имя которого задается в командной строке, строки, содержащие образец, выводятся на экран.

Строки читаются в переменную line типа string. Для поиска в строке line строки-образца sample вызывается функция find():

```
line.find(sample)
```

Если образец sample не входит в строку line, функция find() возвращает -1, иначе возвращается номер символа line, начиная с которого в line входит sample.

```
// Файл PrnSample.cpp

// Печать строк из файла, содержащих образец
// Файл и образец задаются в командной строке

#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
using namespace std;

int main(int argc, char* argv[])
{
    setlocale(LC_ALL, "Russian");
    string line;                // Для читаемых строк
    ifstream fin;               // Входной файловый поток
    char* progrname = argv[0];  // Указатель на строку с именем программы
    if( argc < 3 ){      // Если аргументов в командной строке не хватает
        cout << "Используйте командную строку: \n"
             << progrname << " file образец\n";
        system("pause");
        exit(1);
    }
    char* filename  = argv[1];       // Указатель на имя файла
    char* sample    = argv[2];       // Указатель на строку - образец
    fin.open(filename);              // Файл  открывается на чтение
    if( ! fin ){                     // Если не удалось открыть файл
        cerr << "Файл " << filename << " не найден \n";
        system("pause");
        exit(1);
    }
    int pos_in_line;                 // Позиция образца в строке
    while( ! fin.eof() ){            // Пока не достигнут конец файла,
```

```
        getline(fin, line);          // читается строка
        pos_in_line = line.find(sample);
        if(pos_in_line != -1)        // Если образец есть
            cout << line << '\n';    // Печать строки
    }
    system("pause");
    return 0;
}
```

Исходный текст данной программы находится в файле PrnSample.cpp, проект был назван так же, поэтому рабочая программа получила название, совпадающее с именем проекта PrnSample.exe. Выполнение командной строки:

```
…> PrnSample.exe
```

приводит к появлению на экране следующих строк:

```
Используйте командную строку:
…\ PrnSample.exe file образец
```

Зададим в качестве исходного файл PrnSample.cpp, содержащий исходный текст рассматриваемой программы, а в качестве образца строку "cout". Результат приведен на рис. 14.7.

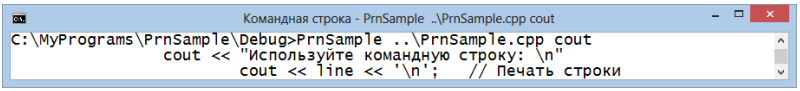

Рис. 14.7. Результат выполнения программы

Программа запускается из папки Debug. Для доступа к исходному файлу перед его именем указан путь в виде двух точек и обратной наклонной черты: (..\). Две точки обозначают каталог, охватывающий текущий каталог. Программа вывела на экран те строки файла PnSample.cpp, которые содержат в качестве подстроки образец "cout".

14.8. Задание аргументов командной строки в среде разработки

При отладке программ удобно задавать аргументы командной строки непосредственно в среде разработки. В Visual Studio для этого надо открыть окно свойств проекта командой **Проект, Свойства** (или **Alt+F7**), в разделе **Свойства конфигурации** выбрать ветку **Отладка** и в свойстве **Командные аргументы** ввести нужные параметры. В их число не надо включать имя самой программы. При запуске программы из среды разработки текущим каталогом считается каталог проекта, поэтому имя файла для чтения указывается без пути к нему (рис.14.8).

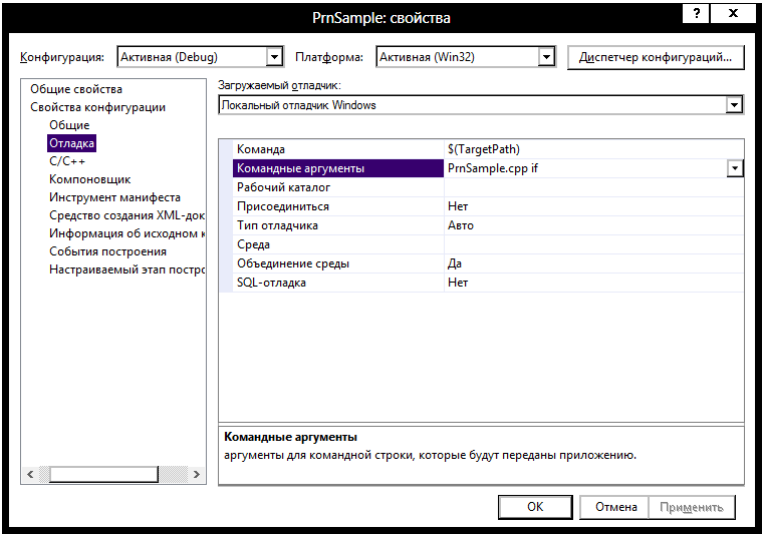

Рис. 14.8. Задание аргументов командной строки в среде Visual Studio

Результаты работы программы с аргументами командной строки `PrnSample.cpp` и `if` приведены на рис.14.9.

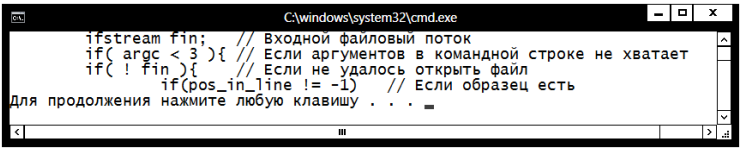

Рис. 14.9. Вывод строк файла, содержащих образец `if`

14.9. Варианты конфигурации проекта

Существуют две основные конфигурации проекта: **`Отладка`** (**`Debug`**) и **`Выпуск`** (**`Release`**). Выбор конфигурации проекта делается в окне свойств проекта. Например, на рис. 14.8 используется конфигурация **`Debug`** – отладочная. При построении такой конфигурации в исполняемый файл включается отладочная информация, благодаря которой становится возможным использование режима отладки, в том числе построчное выполнение программы и просмотр текущих значений переменных программы. Для такой конфигурации создается папка **`Debug`**, в которую помещаются результаты построения проекта, в том числе и исполняемый exe-файл.

Для выбора другой конфигурации проекта нужно в окне свойств (рис. 14.8) нажать кнопку **Диспетчер конфигураций** и выбрать желаемую конфигурацию (рис.14.10).

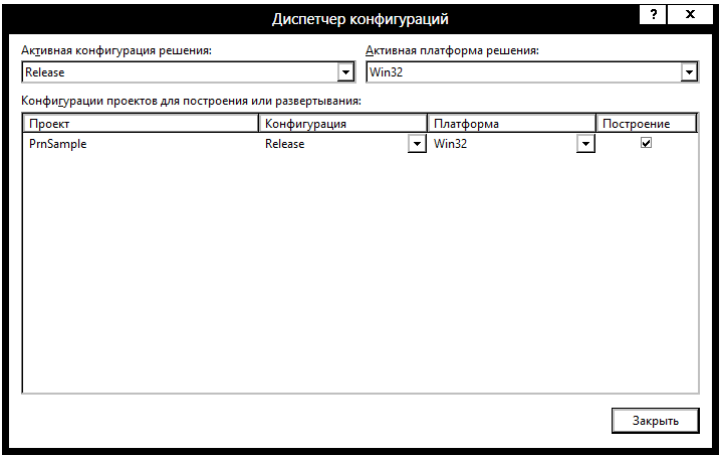

Рис. 14.10. Выбор конфигурации для проекта

При построении проекта в режиме **Release** создается папка с таким же названием, в которую помещаются результаты построения. В исполняемый exe-файл отладочная информация не вставляется, поэтому он получается существенно меньше, чем в режиме **Debug**.