The command-line arguments are typed by the user and are delimited by a space. The first argument is always the filename (command name) and contains the program to be executed. How do these arguments get into the program?

The **main()** functions which we have been using up to now without any arguments can take two arguments as shown below:

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

The first argument **argc** (known as *argument counter*) represents the number of arguments in the command line. The second argument **argv** (known as *argument vector*) is an array of **char** type pointers that points to the command line arguments. The size of this array will be equal to the value of **argc**. For instance, for the command line

```
C > exam data results
```

the value of **argc** would be 3 and the **argv** would be an array of three pointers to strings as shown below:

```
argv[0] ---> exam
argv[1] ---> data
argv[2] ---> results
```

Note that **argv[0]** always represents the command name that invokes the program. The character pointers **argv[1]** and **argv[2]** can be used as file names in the file opening statements as shown below:

```
.....
.....
infile.open(argv[1]);   // open data file for reading
.....
.....
outfile.open(argv[2]);  // open results file for writing
.....
.....
```

Program 11.8 illustrates the use of the command-line arguments for supplying the file names. The command line is

```
test ODD EVEN
```

The program creates two files called **ODD** and **EVEN** using the command-line arguments, and a set of numbers stored in an array are written to these files. Note that the odd numbers are written to the file **ODD** and the even numbers are written to the file **EVEN**. The program then displays the contents of the files.

COMMAND-LINE ARGUMENTS

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{
        int number[9] = {11,22,33,44,55,66,77,88,99};

        if(argc != 3)
        {
                cout << "argc = " << argc << "\n";
                cout << "Error in arguments \n";
                exit(1);
        }
        ofstream fout1, fout2;

        fout1.open(argv[1]);

        if(fout1.fail())
        {
                cout << "could not open the file"
                     << argv[1] << "\n";
                exit(1);
        }

        fout2.open(argv[2]);

        if(fout2.fail())
        {
                cout << "could not open the file "
                     << argv[2]  << "\n";
                exit(1);
        }

for(int i=0; i<9; i++)
{
        if(number[i] % 2 == 0)
        fout2 << number[i] << " ";          // write to EVEN file
        else
        fout1 << number[i] << " ";          // write to ODD file
}
```

(Contd)

```
    fout1.close();
    fout2.close();

    ifstream  fin;
    char  ch;
    for(i=1; i<argc; i++)
    {
           fin.open(argv[i]);
           cout << "Contents of " << argv[i] << "\n";
           do
           {
                  fin.get(ch);  // read a value
                  cout << ch;   // display it
           }
           while(fin);
           cout << "\n\n";
           fin.close();
    }
    return 0;
}
```

PROGRAM 11.8

The output of Program 11.8 would be:

```
Contents of ODD
11 33 55 77 99

Contents of EVEN
22 44 66 88
```

SUMMARY

- ⇔ The C++ **I/O** system contains classes such as **ifstream, ofstream** and **fstream** to deal with file handling. These classes are derived from **fstreambase** class and are declared in a header file *iostream.*
- ⇔ A file can be opened in two ways by using the constructor function of the class and using the member function **open()** of the class.
- ⇔ While opening the file using constructor, we need to pass the desired filename as a parameter to the constructor.
- ⇔ The **open()** function can be used to open multiple files that use the same stream object. The second argument of the **open()** function called file mode, specifies the purpose for which the file is opened.

11.8 *How many file objects would you need to create to manage the following situations?*

(a) *To process four files sequentially.*

(b) *To merge two sorted files into a third file.*

Explain.

11.9 *Both* ***ios::ate*** *and* ***ios::app*** *place the file pointer at the end of the file (when it is opened). What then, is the difference between them?*

11.10 *What does the "current position" mean when applied to files?*

11.11 *Write statements using* ***seekg()*** *to achieve the following:*

(a) *To move the pointer by 15 positions backward from current position.*

(b) *To go to the beginning after an operation is over.*

(c) *To go backward by 20 bytes from the end.*

(d) *To go to byte number 50 in the file.*

11.12 *What are the advantages of saving data in binary form?*

11.13 *Describe how would you determine number of objects in a file. When do you need such information?*

11.14 *Describe the various approaches by which we can detect the end-of-file condition successfully.*

11.15 *State whether the following statements are* ***TRUE*** *or* ***FALSE***.

(a) *A stream may be connected to more than one file at a time.*

(b) *A file pointer always contains the address of the file.*

(c) *The statement*

```
outfile.write((char *) & obj,sizeof(obj));
```

writes only data in **obj** *to* **outfile.**

(d) *The* ***ios::ate*** *mode allows us to write data anywhere in the file.*

(e) *We can add data to an existing file by opening in write mode.*

(f) *The parameter* ***ios::app*** *can be used only with the files capable of output.*

(g) *The data written to a file with* ***write()*** *function can be read with the get() function.*

(h) *We can use the functions* ***tellp()*** *and* ***tellg()*** *interchangeably for any file.*

(i) *Binary files store floating point values more accurately and compactly than the text files.*

(j) *The* ***fin.fail()*** *call returns non-zero when an operation on the file has failed.*

Debugging Exercises

11.1 Identify the error in the following program.

```
#include <iostream.h>
#include <fstream.h>

void main()
```