MODULE 19
C++ FILE I/O

MODULE 9
C FILE INPUT OUTPUT

My Training Period: hours

Note: The examples in this Module were compiled and run using **Win32 empty console application** without `.def` (definition) and `.rc` (resource) files (Borland®). All program examples have been tested using Borland C++ 5.xx **ONLY**. It should be OK if you use **Win32 empty console application** as a target for other compilers because header files used are from C++ Standard Library. For Linux/Unices, you have to concern about the path or directory access because of the different file system. You may need some code modification and been left for your assignments :o).
You may consider reading Section 23.3, Module 23 first, for using traditional, fully complied C++ or mixing the C and C++ codes.

Abilities

- Understand and use the `ifstream`, `ofstream` and `fstream` class objects.
- Understand and use a sequential access file – Read and Write member functions.
- Understand and use a random access file – Read and Write member functions.
- Be familiar with other file I/O member functions.

19.1 Introduction

- The idea about the stream has been explained extensively in Module 9 (C file I/O) and C++ formatted I/O also has been explained in Module 18.
- A file just a collection of related data that treated by C++ as a series of bytes.
- By the way, streams move in one way and in serial fashion from *receiver* and *sender*.
- Other than files on disk, devices such as magnetic tapes, primary or secondary storage devices, printers and networks that carrying data are also treated as files.
- File I/O is one of the interesting topics in C/C++ because of the wide applications. We need to write to disk when we do software installation, writing windows registry, read, write, delete, update file contents, sending and receiving data through networks (sockets) etc.
- The include files needed in order to use the **disk file** I/O class objects are: `iostream.h` and `fstream.h`. `iostream.h` contains **standard input output** objects such as `cin`, `cout`, `cerr` and `clog` (refer to Module 18).
- Keep in mind that new header files which are fully C++ standard compliance that using template based header files (ISO/IEC C++) don't have the `.h` anymore. You can read the story HERE.
- Furthermore same as in C File I/O (Module 9), all the program examples don't consider the file access permissions. Normally, file access permissions are combined with the user permissions and rights as security features of the Operating Systems. They are implementation dependant and the discussion can be found starting from HERE or Tutorial #2.
- For C++, all file objects belong to the one of the following classes:

Class	Brief description
`ifstream`	Objects belong to this class are associated with files opened for input purposes.
`ofstream`	Objects belong to this class are associated with files opened for output purposes.
`fstream`	Objects belong to this class are associated with files opened for input and output purposes.

Table 19.1: File input/output classes.

- These classes are defined in `fstream.h` header file.
- Data files are attached with files objects using the `open()` member function of the file object.
- The `open()` prototype is:

```
void open(const char *name, int mode, int access);
```

- Where:

- `name` is the filename.
- Open file `modes` (flags) are used to indicate what to do to the **file** (e.g. open, close) and the **data** (e.g. read, write). The flags can be logically `OR`ed. Mode must be one of the following:

Mode	Description
`ios::app`	Append data to the end of the output file.
`ios::ate`	Go to the end of the file when open.
`ios::in`	Open for input, with `open()` member function of the `ifstream` variable.
`ios::out`	Open for output, with `open()` member function of the `ofstream` variable.
`ios::binary`	Binary file, if not present, the file is opened as an `ASCII` file as default.
`ios::trunc`	Discard contents of existing file when opening for write.
`ios::nocreate`	Fail if the file does not exist. For output file only, opening an input file always fails if there is no fail.
`ios::noreplace`	Do not overwrite existing file. If a file exists, cause the opening file to fail.

Table 19.2: File open modes

- File protection `access` determines how the file can be accessed. It is Operating System dependent and there are others attributes that are implementation extensions. For DOS® example, it must be one of the following:

Attributes	Description
`0`	Normal file or Archive
`1`	Read-only file
`2`	Hidden file
`4`	System file

Table 19.3: File types

- To declare the input file stream i.e. for reading we would declare like this:

```
ifstream  theinputfile;
```

- To declare the output file stream i.e. for writing we would declare like this:

```
ofstream  theoutputfile;
```

- Then, to connect to a stream, let say opening file `testfileio.dat` for reading, the file which located in the same folder as the executable program we would write like this:

```
theinputfile.open("testfileio.dat");
```

- Or with the path we could write:

```
theinputfile.open("c:\\testfileio.dat");
```

- Next, opening a file for reading and binary type modes, we could write:

```
theinputfile.open("testfileio.dat", ios::in|ios::binary);
```

- Another example, opening a file for reading, with normal type access and go to the end of the file, we could write like this:

```
theinputfile.open("testfileio.dat", ios::in|ios::ate, 0);
```

- For writing, to connect to a stream, let say opening file `testfileio.dat` for writing, the file which located in the same folder as the running program. Previous content of the `testfileio.dat` will be overwritten, we could write like this:

```
theoutputfile.open("testfileio.dat");
```

- Or with the path:

```
theoutputfile.open("c:\\testfileio.dat");
```

- Then, opening a file for writing and appending at the end of the file modes, we could write like this:

```
theoutputfile.open("testfileio.dat", ios::out|ios::app);
```

- Or opening for writing, check the existing of the file with normal access modes, we could write like this:

```
theoutputfile.open("testfileio.dat", ios::out|ios::nocreate, 0);
```

- After we have completed the file processing, we have to close or disconnect the stream to free up the resources to be used by other processes or programs. Using the `close()` member function, for input stream we would write like this:

```
theinputfile.close();
```

- And for output stream:

```
theoutputfile.close();
```

- During the opening for reading or writing, we should provide **error handling routines** to make sure the file operations have completed successfully otherwise some error message should be displayed or error handling routines been executed.
- For example we can use `fail()` member function:

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>  //for exit()

void main(void)
{

      ifstream inputfile;

      inputfile.open("testfileio.dat");

      if(inputfile.fail())
      {
            cout<<"The file could not be opened!\n";
            exit(1); // 0 – normal exit, non zero – some error
      }
…
```

- Or `bad()` with `cerr`.

```
if(inputfile.bad())
{
      cerr<<"Unable to open testfileio.dat\n";
      exit(1); // 0 – normal exit, non zero – some error
}
```

- The following examples created for **Win32 empty console application** without `.def` (definition) and `.rc` (resource) files. All examples in this Module have been tested using Borland 5.xx **ONLY**.
- At the end of this Module, there are program examples compiled with `VC++`/`VC++ .Net` and `g++` for Linux.

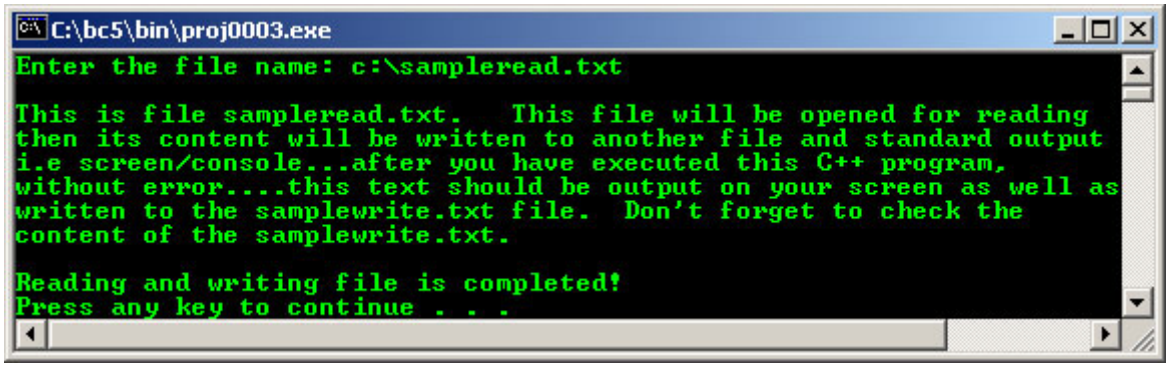
- Keep in mind that in Microsoft Windows operating system environment there are another layer of security for accessing Windows objects. The documentation of the Windows security features can be found in Win32 SDK documentation. This Module will only discuss file input/output in general from C++ programming perspective.
- Firstly, create a file named `sampleread.txt` at root on `C:` drive (or other location provided you explicitly state the full path in the program). Write some text as shown below in `sampleread.txt` file and save it.

```
This is file sampleread.txt.   This file will be opened for reading
then its content will be written to another file and standard output
i.e screen/console...after you have executed this C++ program,
without error....this text should be output on your screen as well as
written to the samplewrite.txt file.  Don't forget to check the
content of the samplewrite.txt.
```

`sampleread.txt` file content

```
//Reading from available file content
//then writing the content to another
//file.  Firstly, create file for reading (can include path)
//let says "C:\sampleread.txt", at root on C drive.
//Type some text as shown…
//Then executes this program.

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

//function definition, to open file for reading...
void openinfile(ifstream &infile)
{
	char filename[100];
	cout<<"Enter the file name: ";
	//Enter the filename that you have created
	//(can include path). From the comment above
	//you have to enter "C:\sampleread.txt"
	//without the double quotes.

	cin>>filename;
	infile.open(filename);
}

void main(void)
{
	//declare the input file stream
	ifstream inputfile;
	//declare the output file stream
	ofstream outputfile;

	char chs;

	//function call for opening file for reading...
	openinfile(inputfile);
	//create, if not exist and open it for writing
	outputfile.open("C:\\samplewrite.txt");

	//test until the end of file
	while (!inputfile.eof())
	{
		//read character until end of file
		inputfile.get(chs);
		if (!inputfile.eof())
		{
		//output character by character (byte) on screen, standard output
		cout<<chs;
		//write to output file, samplewrite.txt
		outputfile<<chs;
		}
	}
	cout<<"\nReading and writing file is completed!"<<endl;
	system("pause");
	//close the input file stream
	inputfile.close();
	//close the output file stream
	outputfile.close();
}
```

Output:

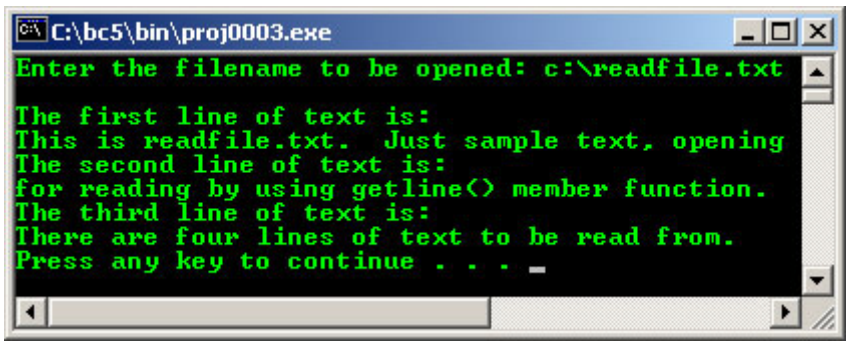


- In this program we do not provide error handlings, the existing file to be opened is not verified.
- Another example using `getline()` member function. Firstly create text file, named `readfile.txt`, put it on drive C: of the Windows platform, then type the following sample text and save it.

```
This is readfile.txt.  Just sample text, opening
for reading by using getline() member function.
There are four lines of text to be read from.
This is just plain simple reading text from a file.
```

```
//using getline() member function
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

void main(void)
{
	char filename[50];
	ifstream inputfile;
	char FirstLine[50];
	char SecondLine[50];
	char ThirdLine[50];

	//prompt user for file name to be opened...
	cout<<"Enter the filename to be opened: ";
	cin>>filename;

	//test open file for reading...
	inputfile.open(filename);
	//if not the end of file, do...
	if(!inputfile.eof())
	{
		cout<<"\nThe first line of text is: \n";
		inputfile.getline(FirstLine, 50);
		cout<<FirstLine<<'\n';
		cout<<"The second line of text is: \n";
		inputfile.getline(SecondLine, 50);
		cout<<SecondLine<<endl;
		cout<<"The third line of text is: \n";
		inputfile.getline(ThirdLine, 50);
		cout<<ThirdLine<<endl;
	}
	system("pause");
}
```

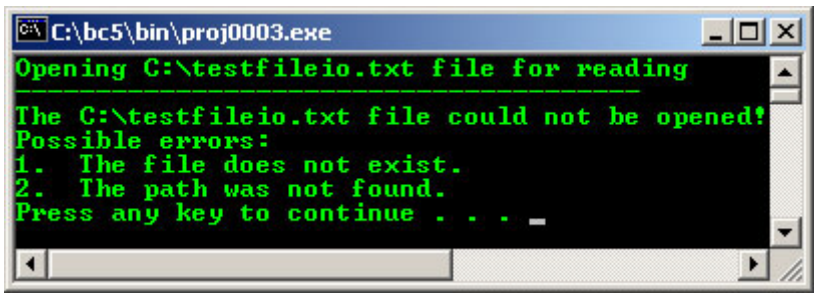
Output:



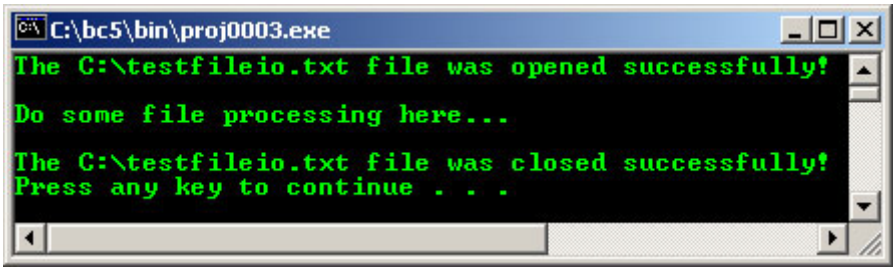
- Another program example with a simple exception handling.

```
//Simple file 'exception handling' when
//opening file for reading.
//There is no testfileio.txt at the root of
//drive C at the beginning.

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

 void main(void)
 {
       char filename[] = "C:\\testfileio.txt";
       ifstream inputfile;

       inputfile.open(filename, ios::in);
       //test if fail to open fail for reading, do…
       if(inputfile.fail())
       {
              cout<<"Opening "<<filename<<" file for reading\n";
              cout<<"---------------------------------------\n";
              cout<<"The "<<filename<<" file could not be opened!\n";
              cout<<"Possible errors:\n";
              cout<<"1.  The file does not exist.\n";
              cout<<"2.  The path was not found.\n";
              system("pause");
              exit(1);   //just exit
              //0-normal, non zero - some error
       }
       //if successful opening file for reading, do…
       else
       {
              cout<<"The "<<filename<<" file was opened successfully!\n";
              cout<<"\nDo some file processing here...\n";
       }
       inputfile.close();

       //test if fail to close the file, do…
       if(inputfile.fail())
       {
              cout<<"\nThe file "<<filename<<" could not be closed!\n";
              system("pause");
              exit(1);
       }
       //else, do…
       else
       cout<<"\nThe "<<filename<<" file was closed successfully!\n";
       system("pause");
}
```

Output:



- Then, create file named `testfileio.txt` on drive `C`. Re-run the program, the following should be output.

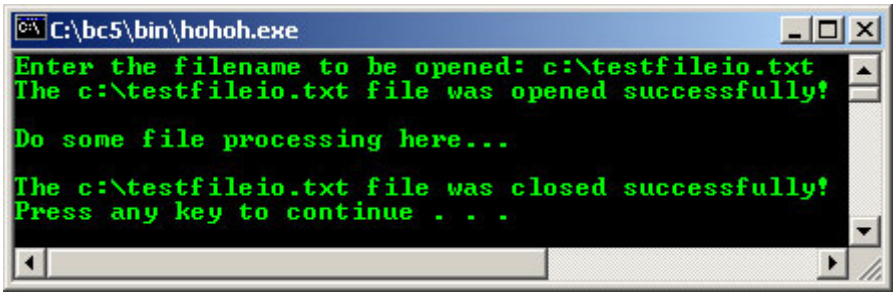


- Same routine can be use for file output `ofstream`, by replacing the `ifstream` objects.
- The following example shows you how to prompt user for the file name.

```
//Prompting user for filename
//to be opened….others should be
//same as the previous example.
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

void main(void)
{
      char filename[100];
      ifstream inputfile;

 //Prompting user for filename to be opened…
 //including the full path if necessary…
 //e.g. c:\testfileio.txt, c:\Windows\Temp\testfile.txt etc
 cout<<"Enter the file name to be opened: ";

 //store at an array filename...
 //Array without [] is a pointer to the
 //first array's element...
 cin>>filename;

  //opened the file for input...
  inputfile.open(filename, ios::in);

      //test if fail to open file for reading, do…
      if(inputfile.fail())
      {
            cout<<"Opening "<<filename<<" file for reading\n";
            cout<<"-------------------------------------\n";
            cout<<"The "<<filename<<" file could not be opened!\n";
            cout<<"Possible errors:\n";
            cout<<"1.  The file does not exist.\n";
            cout<<"2.  The path was not found.\n";
            system("pause");
            exit(1);   //just exit
            //0-normal, non zero - some error
      }
      //if successful opening file for reading, do…
      else
      {
            cout<<"The "<<filename<<" file was opened successfully!\n";
            cout<<"\nDo some file processing here...\n";
      }
      //close file for input…
      inputfile.close();
      //test if fail to close the file, do…
```

```
        if(inputfile.fail())
        {
               cout<<"\nThe file "<<filename<<" could not be closed!\n";
               system("pause");
               exit(1);
        }
        //else, do…
        else
        cout<<"\nThe "<<filename<<" file was closed successfully!\n";
        system("pause");
}
//tested using the win32 console mode........
//provided the file testfileio.txt exists on the C: drive…
```

Output:



19.2 Sequential File Processing

- Reading data and do some calculation, then display the data. Firstly, create a file named `testfileio1.txt` on drive `C`. Key in some data in this test file as shown below and save the file.

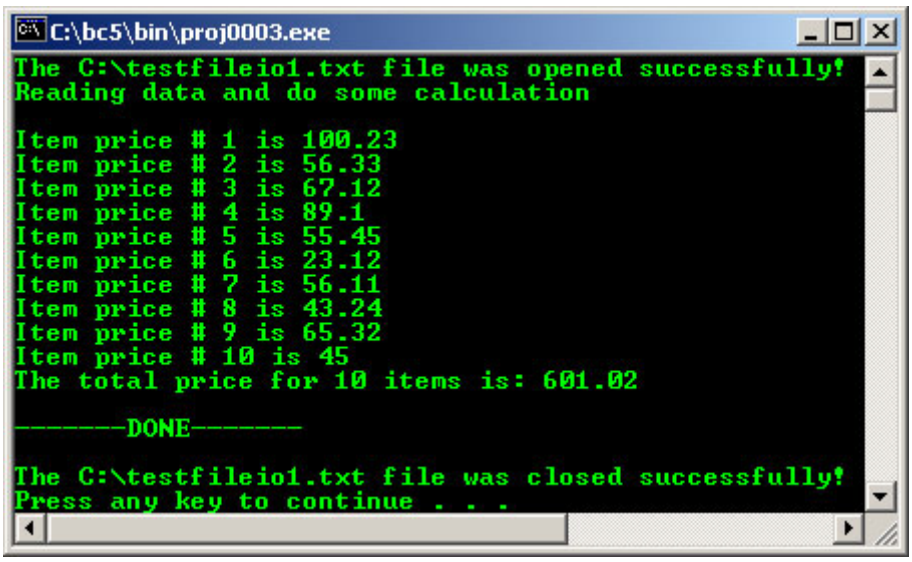
```
100.23  56.33  67.12 89.10 55.45
23.12  56.11  43.24  65.32  45.00
```

- Create and run the following program.

```
//Simple processing data from external file.
//Read the data in sequential mode, do some
//calculation and display to the standard output.
//Create file testfileio1.txt on drive C, and
//type some data as shown
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

void main(void)
{
      char filename[] = "C:\\testfileio1.txt";
      ifstream inputfile;

      //-------opening input file for reading--------
      inputfile.open(filename, ios::in);
      //test if fail to open the file, do…
       //error handling for file opening
      if(inputfile.fail())
      {
             cout<<"Opening file "<<filename<<" for reading\n";
             cout<<"----------------------------------------\n";
             cout<<"The file could not be opened!\n";
             cout<<"Possible errors:\n";
             cout<<"1.  The file does not exist.\n";
             cout<<"2.  The path was not found.\n";
             system("pause");
             exit(1);   //just exit
             //0-normal, non zero - some error
      }
       //if successful, do the following...
      else
      {
      cout<<"The "<<filename<<" file was opened successfully!\n";

     //declare some variables for simple calculation
      float  price, total = 0;
```

```
        int count = 0;

        cout<<"Reading data and do some calculation\n\n";
       //read data from input stream...
        inputfile>>price;

       //test, if end of file not found, do the following...
        while(!inputfile.eof())
        {
               //total = total + price
               total += price;
               count++;
               cout<<"Item price # "<<count<<" is "<<price<<endl;
               //re read the next item price within the loop
               inputfile>>price;
        }
        cout<<"The total price for "<<count<<" items is: "<<total<<endl;
        cout<<"\n-------DONE-------\n"<<endl;
        //close the input file
        inputfile.close();

        //test closing file, if fail to close the file, do...
        //error handling for file closing
        if(inputfile.fail())
        {
               cout<<"The "<<filename<<" file could not be closed!\n";
               system("pause");
                 //something wrong, just exit...
               exit(1);
        }

         //if successful closes the file, do....
        else
          cout<<"The "<<filename<<" file was closed successfully!\n";
        system("pause");
        }
}
```

Output:

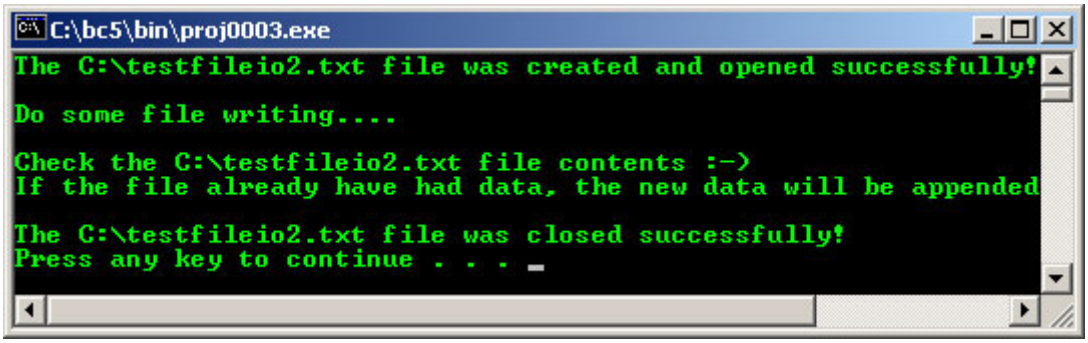


- Now let try using the `ostream` class object. This will create and open a file for writing. The file will be created if it does not exist yet.

```
//Simple processing data from external file.
//Creating, opening and writing some data in file
//and appending data at the end of file...
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

void main(void)
{
char filename[] = "C:\\testfileio2.txt";
ofstream outputfile;
```

```
//----creating, opening and writing/appending data to file-----
outputfile.open(filename, ios::out|ios::app);
//simple error handling for file creating/opening for writing
//test if fail to open the file, do…
if(outputfile.fail())
{
	cout<<"Creating and opening file "<<filename<<" for writing\n";
	cout<<"-----------------------------------------\n";
	cout<<"The "<<filename<<" file could not be created/opened!\n";
	cout<<"Possible errors:\n";
	cout<<"1.  The file does not exist.\n";
	cout<<"2.  The path was not found.\n";
	system("pause");
	exit(1);   //just exit
	//0-normal, non zero - some error
}
//else, if the file can be opened, do…
else
{
	cout<<"The "<<filename<<" file was created and opened successfully!\n";
	cout<<"\nDo some file writing....\n\n";

	outputfile<<"Writing some data in this file\n";
	outputfile<<"-----------------------------\n";
	cout<<"Check the "<<filename<<" file contents :-)"<<endl;
	cout<<"If the file already have had data, the new data will be appended\n";

	int sampledata;
	//write some integers to the file...
	for(sampledata=0; sampledata<=10; sampledata++)
	outputfile<<sampledata<<"  ";
	outputfile<<endl;
	//close the output file
	outputfile.close();
	//test if fail to close the file, do the following...
	//simple error handling for output files closing
	if(outputfile.fail())
	{
		cout<<"The "<<filename<<" file could not be closed!\n";
		system("pause");
		exit(1);
	}
	//test if successful to close the file, do the following...
	else
	cout<<"\nThe "<<filename<<" file was closed successfully!\n";
	system("pause");
}
}
```

Output:



- The content of the `testfileio2.txt` is as follows:

```
Writing some data in this file
------------------------------
0  1  2  3  4  5  6  7  8  9  10
```

- When you re run this program second times, the data will be appended at the end of the pervious data.

```
Writing some data in this file
------------------------------
```

```
0  1  2  3  4  5  6  7  8  9  10
Writing some data in this file
------------------------------
0  1  2  3  4  5  6  7  8  9  10
```

19.3 Random File Processing

- We can set position of the `get` pointer by using `seekg()`. The prototype are:

```
istream& seekg(streampos pos);
istream& seekg(streamoff off, ios_base::seekdir dir);
```

- These commands will set the position of the `get` pointer. The `get` pointer determines the next location to be read in the buffer associated to the input stream. The parameters:

seekg() parameter	Brief description
`pos`	The new position in the stream buffer of type `streampos` object.
`off`	Value of type `streamoff` indicating the offset in the stream's buffer. It is relative to `dir` parameter.
`dir`	Seeking direction. An object of type `seekdir` that may take any of the following values: ▪ **`ios_base::beg`** (seek from the **beginning** of the stream's buffer). ▪ **`ios_base::cur`** (seek from the **current** position in the stream's buffer). ▪ **`ios_base::end`** (seek from the **end** of the stream's buffer).

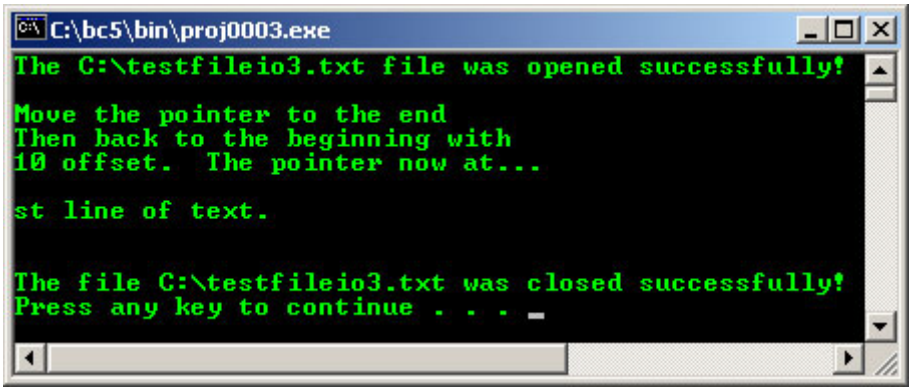
Table 19.4: `seekg()` parameter.

- Try the following `seekg()` member function example:

```
//using seekg()
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

void main(void)
{
	char filename[] = "C:\\testfileio3.txt";
	fstream inputfile, outputfile;

	//--------create, open and write data to file--------
	outputfile.open(filename, ios::out);
	//----write some text-------
	outputfile<<"This is just line of text."<<endl;
	//--------close the output file------------
	outputfile.close();

	//----opening and reading data from file-----
	inputfile.open(filename, ios::in);
	//simple error handling for files creating/opening for writing
	if(inputfile.fail())
	{
		cout<<"Opening "<<filename<<" file for reading\n";
		cout<<"----------------------------------------\n";
		cerr<<"The file "<<filename<<" could not be created/opened!\n";
		cerr<<"Possible errors:\n";
		cerr<<"1.  The file does not exist.\n";
		cerr<<"2.  The path was not found.\n";
		system("pause");
		exit(1);   //just exit
		//0-normal, non zero - some error
	}
	else
	{

		cout<<"The "<<filename<<" file was opened successfully!\n";
		cout<<"\nMove the pointer to the end\n"
			<<"Then back to the beginning with\n"
			<<"10 offset.  The pointer now at...\n"<<endl;
		//flush the stream buffer explicitly...
		cout<<flush;
```

```
        //get length of file
               int length;
               char * buffer;

        //move the get pointer to the end of the stream
        inputfile.seekg(0, ios::end);
        //The tellg() member function returns
        //the current stream position.
        //there is 27 characters including white space
        length = inputfile.tellg();
        //move back the pointer to the beginning with
        //offset of 10
        inputfile.seekg(10, ios::beg);

        //dynamically allocate some memory storage
        //for type char...
        buffer = new char [length];

        //read data as block from input file...
        inputfile.read(buffer, length);
        cout<<buffer<<endl;
        //free up the allocated memory storage...
        delete[] buffer;
        //close the input file
        inputfile.close();
        //simple error handling for output files closing
        //test if fail to close the file, do...
        if(inputfile.fail())
        {
               cerr<<"The "<<filename<<" file could not be closed!\n";
               system("pause");
               exit(1);
        }
        //test if successful to close the file, do...
        else
        cout<<"\nThe file "<<filename<<" was closed successfully!\n";
        system("pause");
        }
}
```

Output:



- We can set position of the `put` pointer by using `seekp()`. The prototype are:

```
ostream& seekp(streampos pos);
ostream& seekp(streamoff off, ios_base::seekdir dir);
```

- These will set the position of the `put` pointer. The `put` pointer determines the next location where to write in the buffer associated to the output stream. The parameters:

seekp() parameter	Brief description
`pos`	The new position in the stream buffer of type `streampos` object.
`off`	Value of type `streamoff` indicating the offset in the stream's buffer. It is relative to `dir` parameter.
`dir`	Seeking direction. An object of type `seekdir` that may take any of the following values: **`ios_base::beg`** (seek from the **beginning** of the stream's buffer). **`ios_base::cur`** (seek from the **current** position in the stream's buffer).

	ios_base::end (seek from the **end** of the stream's buffer).

Table 19.5: `seekp()` parameter.

- Try the following `seekp()` program example:

```
//using seekp()
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>

void main(void)
{
       char filename[] = "C:\\testfileio4.txt";
       ofstream outputfile;

       //----creating, opening and writing data to file-----
       outputfile.open(filename, ios::out);
       //simple error handling for file creating/opening
       //test if fail to open, do...
       if(outputfile.fail())
       {
              cout<<"Creating and opening "<<filename<<" file for writing\n";
              cout<<"-----------------------------------------------------\n";
              cout<<"The "<<filename<<" file could not be created/opened!\n";
              cout<<"Possible errors:\n";
              cout<<"1.  The file does not exist.\n";
              cout<<"2.  The path was not found.\n";
              system("pause");
              exit(1);   //just exit
              //0-normal, non zero - some error
       }
       //test if successful creating/opening the file, do...
       else
       {
              cout<<"The "<<filename<<" file was created and opened successfully!\n";
              cout<<"\nDo some file writing....\n\n";

              int locate;

              outputfile.write("Testing: Just simple example.", 29);
      //tell the pointer position...
       locate = outputfile.tellp();
      //seek the pointer position with offset...
      outputfile.seekp(locate-16);
      outputfile.write(" rumble", 7);
      //close the output file
       outputfile.close();
       //simple error handling for output files closing
       //test if fail to close the file, do...
       if(outputfile.fail())
       {
              cout<<"The "<<filename<<" file could not be closed!\n";
              system("pause");
              exit(1);
       }
       //if successful to close the file, do...
       else
       cout<<"\nThe "<<filename<<" file was closed successfully!\n";
       system("pause");
       }

 }
```

- The content of the `testfileio4.txt` should be:

```
Testing: Just rumble example.
```

- We can use the `ignore()` to extracts characters from input stream and discards them. The prototype:

```
istream& ignore(streamsize n = 1, int delimter = EOF);
```

- Here, the extraction ends when *`n`* characters have been discarded **or** when *`delimter`* is found, whichever comes first. In this last case *`delimter`* is also extracted. For example:

```
//istream ignore()
#include <iostream>
#include <stdlib.h>

int main ()
{
  char firstword, secondword;

  cout<<"Enter your first and last names: ";

  firstword = cin.get();
  cin.ignore(30,' ');
  secondword = cin.get();

  cout<<"The initials letters are: "<<firstword<<secondword<<endl;
  system("pause");

  return 0;
}
```

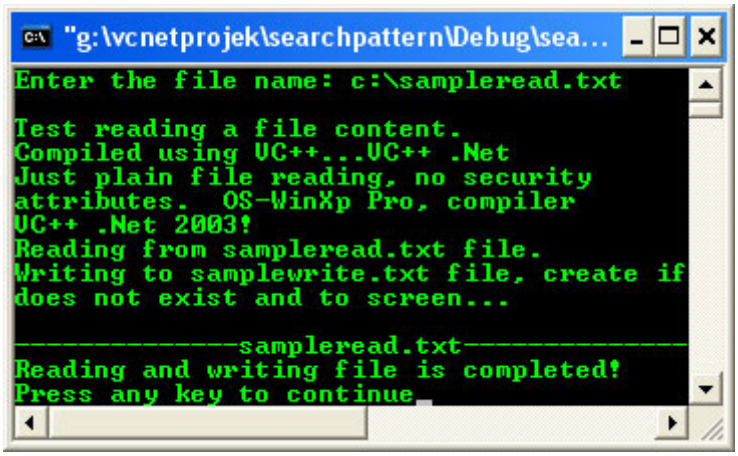
Output:

- Example compiled using VC++/VC++ .Net.

```
//Reading from available file content
//then writing the content to another
//file.  Firstly, create file for reading (can include path)
//let says "C:\sampleread.txt", at root on C drive.
//Type some text as shown…
//Then executes this program.

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

//function definition, to open file for reading...
void openinfile(ifstream &infile)
{
     char filename[100];
     cout<<"Enter the file name: ";
     //Enter the filename that you have created
     //(can include path). From the comment above
     //you have to enter "C:\sampleread.txt"
     //without the double quotes.

     cin>>filename;
     infile.open(filename);
}

void main(void)
{
     //declare the input file stream
     ifstream inputfile;
     //declare the output file stream
     ofstream outputfile;

     char chs;

     //function call for opening file for reading...
     openinfile(inputfile);
     //create, if does not exist and open it for writing
     outputfile.open("C:\\samplewrite.txt");

     //test until the end of file
     while (!inputfile.eof())
     {
            //read character until end of file
            inputfile.get(chs);
            if (!inputfile.eof())
```

```
                {
                //output character by character (byte) on screen, standard output
                cout<<chs;
                //write to output file, samplewrite.txt
                outputfile<<chs;
                }
        }
        cout<<"\nReading and writing file is completed!"<<endl;
        //close the input file stream
        inputfile.close();
        //close the output file stream
        outputfile.close();
}
```

Output:



- Previous example compiled using **g++**.

```
//Simple processing data from external file.
//Read the data in sequential mode, do some
//calculation and display to the standard output.
//Create file testfileio.txt in the current
//working directory and type some int data in it...
#include <iostream>
#include <fstream>
using namespace std;

int main(void)
{
        char filename[] = "testfileio.txt";
        ifstream inputfile;

        //-------opening input file for reading--------
        inputfile.open(filename, ios::in);
        //test if fail to open the file, do...
        //error handling for file opening
        if(inputfile.fail())
        {
                cout<<"Opening file "<<filename<<" for reading\n";
                cout<<"----------------------------------------\n";
                cout<<"The file could not be opened!\n";
                cout<<"Possible errors:\n";
                cout<<"1.  The file does not exist.\n";
                cout<<"2.  The path was not found.\n";
                exit(1);   //just exit
                //0-normal, non zero - some error
        }
        //if successful, do the following...
        else
        {
        cout<<"The "<<filename<<" file was opened successfully!\n";

        //declare some variables for simple calculation
        float price, total = 0;
        int count = 0;
        cout<<"Reading data and do some calculation\n\n";
        //read data from input stream...
        inputfile>>price;
```

```
        //test, if end of file not found, do the following...
        while(!inputfile.eof())
        {
               //total = total + price
               total += price;
               count++;
              cout<<"Item price # "<<count<<" is "<<price<<endl;
              //re read the next item price within the loop
              inputfile>>price;
        }
        cout<<"The total price for "<<count<<" items is: "<<total<<endl;
        cout<<"\n-------DONE-------\n"<<endl;
        //close the input file
        inputfile.close();

        //test closing file, if fail to close the file, do
        //simple error handling for file closing
        if(inputfile.fail())
        {
        cout<<"The "<<filename<<" couldn't be closed!\n";
        exit(1);
        }

        //if fail, do....
        else
        cout<<"The "<<filename<<" file closed successfully!\n";
        return 0;
        }
}
```

[bodo@bakawali ~]$ g++ fileio.cpp -o fileio
[bodo@bakawali ~]$ ./fileio

```
The testfileio.txt file was opened successfully!
Reading data and do some calculation

Item price # 1 is 1.22
Item price # 2 is 4.5
Item price # 3 is 12
Item price # 4 is 10.56
Item price # 5 is 23.11
Item price # 6 is 7.8
Item price # 7 is 54.2
Item price # 8 is 30
Item price # 9 is 9.5
Item price # 10 is 45.45
The total price for 10 items is: 198.34

-------DONE-------

The testfileio.txt couldn't be closed!
```

- From the output, the `inputfile.fail()` is true. Can you find the reason for me?

----------o0o----------

Further reading and digging:

1. Check the best selling C/C++, Object Oriented and pattern analysis books at Amazon.com.