Advanced File Operations

Adapted from Tony Gaddis and Barret Krupnow (2016), Starting out with C++: From Control Structures through Objects

Content

- File Operations
- Passing File Stream Objects to Functions
- Member Functions for Reading and Writing Files
- Binary Files
- Random-Access Files

FILE OPERATIONS

File Operations

- A file is a collection of data that is usually stored on a computer's disk. Data can be saved to files and then later reused

- Almost all real-world programs use files to store and retrieve data extensively:
  - Word processing
  - Databases
  - Spreadsheets
  - Compilers

This chapter covers more advanced file operations and focuses primarily on the fstream data type. Table below compares the `ifstream`, `ofstream`, and `fstream` data types. All of these data types require the fstream header file.

Data Type	Description
`ifstream`	Input File Stream. This data type can be used only to read data from files into memory.
`ofstream`	Output File Stream. This data type can be used to create files and write data to them.
`fstream`	File Stream. This data type can be used to create files, write data to them, and read data from them.

Using Files

- Requires fstream header file
  - use **`ifstream`** data type for input files
  - use **`ofstream`** data type for output files
  - use **`fstream`** data type for both input, output files
- Can use **>>**, **<<** to read from, write to a file
- Can use **`eof`** member function to test for the end of input file.

fstream Object

- The **`fstream`** object can be used for either input or output
- Must specify mode on the open statement
- Sample modes:
  - **`ios::in`** – for input files
  - **`ios::out`** – for output files
- Can be combined on open call:

```
dFile.open("class.txt", ios::in | ios::out);
```

File Access Flags

Table 12-2

File Access Flag	Meaning
`ios::app`	Append mode. If the file already exists, its contents are preserved and all output is written to the end of the file. By default, this flag causes the file to be created if it does not exist.
`ios::ate`	If the file already exists, the program goes directly to the end of it. Output may be written anywhere in the file.
`ios::binary`	Binary mode. When a file is opened in binary mode, data is written to or read from it in pure binary format. (The default mode is text.)
`ios::in`	Input mode. Data will be read from the file. If the file does not exist, it will not be created and the open function will fail.
`ios::out`	Output mode. Data will be written to the file. By default, the file's contents will be deleted if it already exists.
`ios::trunc`	If the file already exists, its contents will be deleted (truncated). This is the default mode used by `ios::out`.

Using Files - Example

```
// copy 10 numbers between files
// open the files

fstream infile("input.txt",ios::in);
fstream outfile("output.txt",ios::out);
int num;

for (int i = 1; i <= 10; i++){
    infile >> num;  // read from the input file
    outfile << num; // write into the output file
}

infile.close(); // close the files
outfile.close();
```

Default File Open Modes

- **ifstream:**
  - open for input only
  - file cannot be written to
  - open fails if file does not exist

- **ofstream:**
  - open for output only
  - file cannot be read from
  - file created if no file exists
  - file contents erased if file exists

- The filename used by `ifstream`, `ifstream` and `fstream` objects must be a c-string (i.e., array of char).
- Thus, if you pass a C++ string, you need to convert it first to a c-string by c_str() method. Example:

```
string filename ="input.txt";
ifstream fin(filename.c_str());
```

More File Open Details

- Can use filename, flags in definition:

```
ifstream gradeList("grades.txt");
```

- File stream object set to **NULL** or **0** (**false**) if open **failed**:

```
if (!gradeList)  ...   // This means that it is unable to open the file.
                       // It is the same meaning as if (gradeList==NULL)
```

- Can also check fail member function to detect file open error:

```
if (gradeList.fail()) ...
```

PASSING FILE STREAM OBJECTS TO FUNCTIONS

Passing File Stream Objects to Functions

- It is very useful to pass file stream objects to functions
- Be sure to always **pass file stream objects by reference**.

Passing File Stream Objects to Functions

Program 12-5

```
1  // This program demonstrates how file stream objects may
2  // be passed by reference to functions.
3  #include <iostream>
4  #include <fstream>
5  using namespace std;
6
7  // Maximum amount to read from a line in the file
8  const int MAX_LINE_SIZE = 81;
9
10 // Function prototypes
11 bool openFileIn(fstream &, char *);
12 void showContents(fstream &);
13
14 int main()
15 {
16    fstream dataFile;
17
18    if (!openFileIn(dataFile,"demofile.txt"))
19    {
20       cout << "File open error!" << endl;
21       return 0;    // Exit the program on error.
22    }
```

Function call

Program 12-5 (Continued)

Program 12-5 *(continued)*

```
23      cout << "File opened successfully.\n";
24      cout << "Now reading data from the file.\n\n";
25      showContents(dataFile);
26      dataFile.close();
27      cout << "\nDone.\n";
28      return 0;
29   }
30
31   //*********************************************************
32   // Definition of function openFileIn. Accepts a reference   *
33   // to an fstream object as its argument. The file is opened *
34   // for input. The function returns true upon success, false *
35   // upon failure.                                            *
36   //*********************************************************
37
38   bool openFileIn(fstream &file, char *name)
39   {
40      file.open(name, ios::in);
41      if (file.fail())
42         return false;
43      else
44         return true;
45   }
46
```

Function call

Function definition

Program 12-5 (Continued)

```
47  //*************************************************************
48  // Definition of function showContents. Accepts an fstream  *
49  // reference as its argument. Uses a loop to read each name *
50  // from the file and displays it on the screen.             *
51  //*************************************************************
52
53  void showContents(fstream &file)
54  {
55     char line[MAX_LINE_SIZE];
56
57     while (file >> line)
58     {
59        cout << line << endl;
60     }
61  }
```

Function definition

Program Screen Output

```
File opened successfully.
Now reading data from the file.

Jones
Smith
Willis
Davis

Done.
```

Member Functions for Reading and Writing Files

Member Functions for Reading and Writing Files

- Functions that may be used for input with whitespace, to perform single character I/O, or to return to the beginning of an input file

- Member functions:
  - `getline`: reads input including whitespace
  - `get`: reads a single character
  - `put`: writes a single character

`getline` Method

For working with C-strings. Three arguments needed:

- A c-string (i.e., character array) to hold input
- Number of characters to read
- Terminator (or delimiter) to stop at if encountered before number of characters was read in
- `'\n'` is the default delimiter (i.e, for the third argument)
- Example:

```
ifstream myFile("input.txt");
char name[41], addr[41];
myFile.getline(name, 40);  //using default delimiter '\n'
myFile.getline(addr, 40, '\t');
```

getline Member Function

Program 12-8

```
1  // This program uses the file stream object's getline member
2  // function to read a line of data  from the file.
3  #include <iostream>
4  #include <fstream>
5  using namespace std;
6
7  int main()
8  {
9     const int SIZE = 81;  // Size of input array
10    char input[SIZE];     // To hold file input
11    fstream nameFile;     // File stream object
12
13    // Open the file in input mode.
14    nameFile.open("murphy.txt", ios::in);
15    if (!nameFile)
16    {
17       cout << "ERROR: Cannot open file.\n";
18       return 0;
19    }
20
```

```
21      // Read the file contents.
22      nameFile.getline(input, SIZE);  // Use \n as a delimiter.
23      while (!nameFile.eof())
24      {
25         cout << input << endl;
26         nameFile.getline(input, SIZE);  // Use \n as a delimiter.
27      }
28
29      // Close the file.
30      nameFile.close();
31      return 0;
32  }
```

Program Screen Output

```
Jayne Murphy
47 Jones Circle
Almond, NC 28702
```

getline function

For working with C++ strings

- Prototype:

```
getline(fileObject, stringBuffer, delimiter);
```

- `'\n'` is the default delimiter (i.e, for the third argument)

- Examples:

```
ifstream myFile("input.txt");
string name,addr;
getline(myFile, name);   // using default delimiter '\n'
getline(myFile, addr, '\t');
```

getline Function

```
1  // This program uses the file stream object's getline member
2  // function to read a line of data from a file.
3
4  #include<iostream>
5  #include<fstream>
6  using namespace std;
7
8  int main()
9  {
10     string input;         // To hold file input
11     fstream nameFile;     // File stream object
12
13     // Open the file in input mode
14     nameFile.open("murphy.txt",ios::in);
15
16     if (!nameFile){
17         cout << "ERROR: Cannot open file."  << endl;
18         return 0;
19     }
20
21     // Read the file contents
22     getline(nameFile, input);     // use default delimiter '\n'
23
24     while (!nameFile.eof()){
25         cout << input << endl;
26         getline(nameFile, input);
27     }
28
29     return 0;
30 } // end main
```

Input file

```
Jayne Murphy
47 Jones Circle
Almond, NC 28702
```

Screen output

```
Jayne Murphy
47 Jones Circle
Almond, NC 28702
```

Example of using different delimiters

```
1  // This program uses the file stream object's getline member
2  // function to read a line of data from a file.
3
4  #include<iostream>
5  #include<fstream>
6  using namespace std;
7
8  int main()
9  {
10     string input;         // To hold file input
11     fstream nameFile;     // File stream object
12
13     // Open the file in input mode
14     nameFile.open("murphy.txt",ios::in);
15
16     if (!nameFile){
17         cout << "ERROR: Cannot open file."  << endl;
18         return 0;
19     }
20
21     // Read the file contents
22     getline(nameFile, input,' ');     // use space as delimiter
23
24     while (!nameFile.eof()){
25         cout << input << endl;
26         getline(nameFile, input, ' ');
27     }
28
29     return 0;
30 } // end main
```

Input file

```
Jayne Murphy
47 Jones Circle
Almond, NC 28702
```

Screen output

```
Jayne
Murphy
47
Jones
Circle
Almond,
NC
```

Single Character I/O

- **get**: to read a single character from an input file.

  Example:

```
ifstream gradeFile("grade.txt");
char letterGrade;
gradeFile.get(letterGrade);
```

- Will read any character, **including whitespace**

- **put**: to write a single character to an output file.

  Example:

```
ofstream reportFile("report.out");
char letterGrade;
…..
…..
reportFile.put(letterGrade);
```

BINARY FILES

Binary Files

- Binary files contain unformatted and non-human-readable data.

- Specified by using a binary flag on open:

  Example:

```
inFile.open("nums.dat", ios::in|ios::binary);
```

Binary Files

- Use **read** instead of <<, to read data from files.

Example: To read a letter from an input file

```
fstream fin("input.dat",ios::in|ios::binary);
char ch;
fin.read(&ch, sizeof(ch));
```

First parameter: the address of where to put the data being read in.

The function expects the data variable as `char` .

Thus, if other than `char`, it must be type casted.

Second parameter: how many bytes to read from the file.

Binary Files

- Use **write** instead of >> to write data into files.

Example: To write a letter into an output file

```
fstream fout("report.out",ios::out|ios::binary);
char ch;
….
….
fout.write(&ch, sizeof(ch));
```

First parameter: the address of where to get the data from.

The function expects the data variable as `char` .

Thus, if other than `char`, it must be type casted.

Second parameter: how many bytes to write to the file.

Binary Files

- To `read` and `write` non-character data, must use a typecast operator to treat the address of the data as a character address

Example: To read an integer from a file

```
fstream fin("number.dat",ios::in|ios::binary);
int num;
fin.read(reinterpret_cast<char *>&num, sizeof(num));
```

treat the address of `num` as the address of a `char`

- Or simply use C-style casting

```
fin.read((char *)&num, sizeof(num));
```

Binary Files

Example: To write an integer into a file

```
fstream fout("number.dat",ios::out|ios::binary);
int num;
cin >> num;   // getting a number from the keyboard
fout.write(reinterpret_cast<char *>&num, sizeof(num));
```

C-style:

```
fout.write((char *)&num, sizeof(num));
```

Example Application of Binary Files

Copying a file

```
1  // This program makes a copy of a file into another.
2  // The source file can be any format, e.g. pdf, word file, ppt file or even a plain text file
3
4  #include<iostream>
5  #include<fstream>
6  using namespace std;
7
8  int main()
9  {
10     string src, dest;
11     char c;
12
13     // Ask the user to enter names for input and output files
14     cout << "Enter the source file =>";
15     cin >> src;
16
17     cout << "Enter the destination file =>";
18     cin >> dest;
19
20     fstream fin, fout;
21
22     // Open two files for reading and writing, respectively.
23     fin.open(src.c_str(), ios::in|ios::binary);   // note: use method c_str() to convert to c-string
24     fout.open(dest.c_str(), ios::out|ios::binary);
25
26     // Read and write a character at a time.
27     while (!fin.eof()){
28         fin.read(&c, 1);
29         fout.write(&c,1);
30     }
31
32     fin.close();
33     fout.close();
34
35     return 0;
36 }
```

Example Application of Binary Files

Writing and reading an object into/from a file

```
1  // This program writes and read an object data into/from a file.
2
3  #include<iostream>
4  #include<fstream>
5  using namespace std;
6
7  class Point{
8      private:
9          int x, y;
10
11     public:
12         Point(int _x=0, int _y=0){ x=_x; y=_y;}
13
14         // To print coordinates x and y onto the screen
15         void print() const{
16             cout << "The coordinates: (x=" << x << ", y=" << y << ")" <<endl;
17         }
18 };
19
20 int main()
21 {   Point p1(10,20), p2;
22
23     // Open an output file  and write the point into it
24     fstream fout("point.txt", ios::out|ios::binary);
25     fout.write( (char*)&p1, sizeof(p1));  // note: must perform type casting as
26                                           //       the method only accepts address of char
27     fout.close();
28
29     // Open back the file but now as an input file and read the point coordinates from it
30     fstream fin("point.txt", ios::in|ios::binary);
31     fin.read( (char*)&p2, sizeof(p2));
32     fin.close();
33
34     // Print the point onto the screen
35     p2.print();
36
37     return 0;
38 }
```

Example Application of Binary Files

Writing an array of objects into a file (1)

```
1  // This program writes an array of objects into a file by writing
2  //   an object at a time
3
4  #include<iostream>
5  #include<fstream>
6  using namespace std;
7
8  class Point{
9      private:
10         int x, y;
11
12     public:
13         Point(int _x=0, int _y=0){ x=_x; y=_y;}
14
15         // To print coordinates x and y onto the screen
16         void print() const{
17             cout << "The coordinates: (x=" << x << ", y=" << y << ")" <<endl;
18         }
19 };
20
21 int main()
22 {   Point points[3] = {Point(), Point(1,2), Point(55,88) };
23
24     // Open an output file  and write the points into it
25     fstream fout("points.txt", ios::out|ios::binary);
26
27     for (int i=0; i<3; i++) // use a loop as we want to write an object at a time
28         fout.write( (char*)(points+i), sizeof(Point));  // note: (points + i) is already an address.
29                                                         //       Thus no need to use &
30     fout.close();
31
32     return 0;
33 }
```

Example Application of Binary Files

Writing an array of objects into a file (2)

```
1  // This program writes an array of objects into a file by writing
2  //   all objects at once.
3
4  #include<iostream>
5  #include<fstream>
6  using namespace std;
7
8  class Point{
9      private:
10         int x, y;
11
12     public:
13         Point(int _x=0, int _y=0){ x=_x; y=_y;}
14
15         // To print coordinates x and y onto the screen
16         void print() const{
17             cout << "The coordinates: (x=" << x << ", y=" << y << ")" <<endl;
18         }
19 };
20
21 int main()
22 {   Point points[3] = {Point(), Point(1,2), Point(55,88) };
23
24     // Open an output file  and write the points into it
25     fstream fout("points.txt", ios::out|ios::binary);
26
27     fout.write( (char*)points, 3 * sizeof(Point));   // write 3 objects at once. Thus no more loop
28                                                      // Also, we dont use & for points as points is an array
29                                                      // and an array in C is represented by an address(i.e. a pointer)
30     return 0;
31 }
```

Example Application of Binary Files

Reading an array of objects from a file (1)

```
1  // This program reads an array of objects from a file by
2  //   reading an object at a time.
3
4  #include<iostream>
5  #include<fstream>
6  using namespace std;
7
8  class Point{
9      private:
10         int x, y;
11
12     public:
13         Point(int _x=0, int _y=0){ x=_x; y=_y;}
14
15         // To print coordinates x and y onto the screen
16         void print() const{
17             cout << "The coordinates: (x=" << x << ", y=" << y << ")" <<endl;
18         }
19 };
20
21 int main()
22 {   Point points[3];  // Prepare the array hold the data
23
24     // Open an input file  and read the points' coordinates from it
25     fstream fin("points.txt", ios::in|ios::binary);
26
27     for (int i=0; i<3; i++) // use a loop as we want to read an object at a time
28         fin.read( (char*)(points+i), sizeof(Point));  // note: (points + i) is already an address.
29                                                       //       Thus no need to use &
30     fin.close();
31
32     // display the points onto the screen
33     for (int i=0; i<3; i++) points[i].print();
34
35     return 0;
36 }
```

Screen output

```
The coordinates: (x=0, y=0)
The coordinates: (x=1, y=2)
The coordinates: (x=55, y=88)
```

Example Application of Binary Files

Reading an array of objects from a file (2)

```
1  // This program reads an array of objects from a file by
2  //   reading all objects at once.
3
4  #include<iostream>
5  #include<fstream>
6  using namespace std;
7
8  class Point{
9      private:
10         int x, y;
11     public:
12         Point(int _x=0, int _y=0){ x=_x; y=_y;}
13
14         // To print coordinates x and y onto the screen
15         void print() const{
16             cout << "The coordinates: (x=" << x << ", y=" << y << ")" <<endl;
17         }
18 };
19
20 int main()
21 {   Point points[3];  // Prepare the array hold the data
22
23     // Open an input file  and read the points' coordinates from it
24     fstream fin("points.txt", ios::in|ios::binary);
25
26     fin.read( (char*)points, 3 * sizeof(Point)); // Read 3 objects at once.
27
28     fin.close();
29
30     // display the points onto the screen
31     for (int i=0; i<3; i++) points[i].print();
32     return 0;
33 }
```

Screen output

```
The coordinates: (x=0, y=0)
The coordinates: (x=1, y=2)
The coordinates: (x=55, y=88)
```

RANDOM-ACCESS FILES

Random-Access Files

- Sequential access: start at beginning of file and go through data in file, in order, to end
  - to access 100th entry in file, go through 99 preceding entries first
- Random access: access data in a file in any order
  - can access 100th entry directly

Random Access Member Functions

- `seekg` (seek get): used with input files.
- `seekp` (seek put): used with output files.
- Used to go to a specific position in a file

Random Access Member Functions

- `seekg`,`seekp` arguments:
  - *offset*: number of bytes, as a long
  - *mode* flag: starting point to compute offset

Examples:

```
inData.seekg(25L, ios::beg);
// set read position at 26th byte
// from beginning of file

outData.seekp(-10L, ios::cur);
// set write position 10 bytes
// before current position
```

Important Note on Random Access

- If `eof` is true, it must be cleared before `seekg` or `seekp`:

Example:

```
gradeFile.clear();

gradeFile.seekg(0L, ios::beg);
// go to the beginning of the file
```

Random Access Information

- **tellg** member function: return current byte position in input file

  Example:

```
long int whereAmI;
whereAmI = inData.tellg();
```

- **tellp** member function: return current byte position in output file

  Example:

```
whereAmI = outData.tellp();
```

Example Application of Random-Access

Determining the size of a file

```
1  // This program determines the size of a file in bytes.
2
3  #include<iostream>
4  #include<fstream>
5  using namespace std;
6
7
8  int main()
9  {
10     fstream fin;
11     string fileName;
12     int size;
13
14     // Read the file name from the keyboard
15     cout << "Enter the file name =>";
16     cin >> fileName;
17
18     fin.open(fileName.c_str(), ios::in|ios::binary); // open the file as an input file
19
20     fin.seekg(0,ios::end);  // Move the file cursor to the end of the file
21
22     size = fin.tellg();     // Read the cursor location. As it is currently
23                             //  at the end of file, thus it can be used
24                             //  as indicator of the size of the file
25
26     cout << size <<" Bytes" << endl;  // Print the size in Bytes
27
28     return 0;
29 }
```