File input and output

Anna Simon

September 16, 2015

- no class on Sep. 30th and Oct. 2th,
- instead two codes to write, due Oct. 7th, will be posted on Sakai soon.

File I/O

Three basic file I/O classes:

- ifstream ("in" stream, read in data from a file)
- ofstream ("out" stream write to a file)
- fstream ("in/out")

The streams have to be explicitly open by the programmer (The name of the file is a c-style string):

- `ostream fOut("test.dat");`
- ```
  ostream fOut;
  fOut.open("test.dat");
  ```

Then the insertion << and extraction >> operators can be used to read from / write to a file.
To close the file:

- use close() function: `fOut.close();`
- let the I/O variable go out of scope
  ```

File I/O

```
1  #include <fstream>
2  #include <iostream>
3
4  using namespace std;
5  int main()
6  {
7      // ofstream is used for writing files
8      ofstream fOut("test.dat");
9
10     // Always check if the file was open properly
11     if (!fOut)
12     {
13         cout << "test.dat could not be opened for writing!" << endl;
14         exit(1);
15     }
16
17     // Write a line into the file
18     fOut << "test line" << endl;
19
20     return 0;
21
22 }
```

Insertion and extraction operators

Write a code

Write a code that writes two numbers into a file and then read them back to the terminal. Use insertion $<<$ and extraction $>>$ operators.

getline()

`getline(istream& stream, string& str)` - read from the istream stream into the string str until the newline ('\n') character is found.
NOTE 1: this is not the same `getline()` function that was used to write into a C-style string!
NOTE 2: the stream is written into a string (we'll talk about them later!)

```
1  #include <fstream>
2  #include <iostream>
3  #include <string>
4
5  using namespace std;
6  int main(){
7      string str1, str2;
8      ifstream fIn("test.dat");
9
10     if (!fIn){
11         cout << "test.dat could not be opened!" << endl;
12         exit(1);
13     }
14
15     fIn >> str1; //read from file to str1, will stop at space
16     getline(fIn,str2); //read a line from file to string
17
18     return 0;
19 }
```

Buffer flush

- the output is buffered before it is flushed into the file (to improve performance)
- if the program exits unexpectedly, the buffer will not be flushed
- good practice: use flush() function to write the buffer to file or close all the files before calling exit()
- buffer is automatically flushed when the file is closed

```
1  #include <fstream>
2  using namespace std;
3  int main() {
4
5    ofstream outfile ("test.txt");
6
7    for (int n=0; n<100; ++n)
8    {
9      outfile << n;
10     outfile.flush();
11   }
12   outfile.close();
13
14   return 0;
15 }
```

Problem 1

Multiplication table

Write a code that calculates the results of multiplication of integers between 1 and 9.
Write the code's output into a file. Flush the output every time the multiplier is incremented.
Use <iomanip> header and
`cout << setfill('0') << setw(3) << nValue;`
to add leading zeros to the integers. The output should look like this:

001 002 003 004 005 006 007 008 009
002 004 006 008 010 012 014 016 018
003 006 009 012 015 018 021 024 027
004 008 012 016 020 024 028 032 036
005 010 015 020 025 030 035 040 045
006 012 018 024 030 036 042 048 054
007 014 021 028 035 042 049 056 063
008 016 024 032 040 048 056 064 072
009 018 027 036 045 054 063 072 081

File modes

```
ofstream outf("Sample.dat", ios::app);
fstream fs;
fs.open ("test.txt", fstream::in | fstream::out |
fstream::app);
```

File modes:

- app - open the file in append mode (stream will be appended to the end of the file)
- ate - seek to the end of the file before reading/writing (you're free to move the pointer)
- binary - open the file in binary mode (instead of text mode)
- in - open the file in read mode (default for ifstream)
- nocreate - open the file only if it already exists
- noreplace - open the file only if it does not already exist
- out - open the file in write mode (default for ofstream)
- trunc - erase the file if it already exists

Problem 2

Append output

Write a code that extends the table from Problem 1 to multiplications of numbers from 10 to 19 by 1 to 9. Print out the result by appending it to the file generated by the code from Problem 1.
The output should include:

001 002 003 004 005 006 007 008 009
002 004 006 008 010 012 014 016 018
...
008 016 024 032 040 048 056 064 072
009 018 027 036 045 054 063 072 081
010 020 030 040 050 060 070 080 090
011 022 033 044 055 066 077 088 099
...
017 034 051 068 085 102 119 136 153
018 036 054 072 090 108 126 144 162

Pointer position in the I/O file

The user can read/write starting from any location in the file. This is done by placing the pointer to the specific location in the file.

- `seekg(off, flag)` - for input
- `seekp(off, flag)` - for output

off is an offset in bytes from the reference point flag
flag: beg, cur, end - beginning of the file, current location of the pointer, and end of the file

```
1  // move forward 14 bytes
2  fln.seekg(14, ios::cur);
3   // move backwards 18 bytes
4  fln.seekg(-18, ios::cur);
5
6  // move to 22nd byte in file
7  fln.seekg(22, ios::beg);
8   // move to 24th byte in file
9  fln.seekg(24);
10
11  // move to the 28th byte before
        end of the file
12 fln.seekg(-28, ios::end);
13
14  // move to beginning of file
15 fln.seekg(0, ios::beg);
16
17  // move to end of file
18 fln.seekg(0, ios::end);
```

Problem 3

Scanning input file

Write a code that extracts from a file (generated by Problem 2) the product of multiplication of user specified integers. Use:

- `getline()` to skip the lines,
- `seekg()` to place the cursor before the value to be read in,
- the extraction operator to read it and print it out to the screen.