CS171 – Computer Programming 1

File I/O

Objectives

1. Use `ifstream` objects for input from a file
2. Use `ofstream` objects for output to a file.
3. Create file stream objects with a user supplied file name.

Reading

- Chapter 9

Input from a File

- C++ has an object type for reading files called the `ifstream` class
  - This is defined in the `<fstream>` library
- You create an instance of `ifstream` to represent each file you want to read via:

```
ifstream variableName("filename");
```

- Alternatively you can first create the `ifstream` object and then use the `open()` method to open the file

```
ifstream variableName;
variableName.open("filename");
```

Input from a File

- Once you have an open file you can now use the `ifstream` object exactly the same way you would use `cin`!
- Note that if you don't put the absolute path of the file in the filename, the program will assume the file is relative to the current directory.
  - If you're at a command line, the current directory is the directory you're in when you run the program.
  - If you're in a IDE (like Visual Studio or Xcode), the IDE allows you to set the current directory.

Input From a File

- Recall what we could do with reading in from the command line:
  - To read in a single "thing" (separated by a space or end-line)

```
string data;
cin >> data;
```

  - To read in a line

```
string data;
getline(cout, data, '\n');
```

- But remember the unfortunately quirkiness of using `getline` after >> in the same program?
- We may need to ignore some left over characters from the inputs stream:

```
const unsigned int NUMTOIGNORE = 256;
const char CHARTOIGNORE = '\n';
cin.ignore(NUMTOIGNORE, CHARTOIGNORE);
```

Input From a File

- Likewise, if you write a program that uses both `getline()` and the extraction operator >>, you may need to make use of `fin.ignore(n,c)` to make your program behave properly.
- Also…
  - You can use the `is_open()` method to check that the file exists and that you have permission to read it.

```
#include <fstream>
///stuff

//--- create input object
ifstream fin( "input.txt" );

//--- use input object like cin
//--- read 3 pieces of data from the file
fin >> data1 >> data2 >> data3;

//--- for mixing >> with getline()
fin.ignore();

//--- read in a line of data using getline()
getline( fin, lineOfData, '\n');
```

Closing Files

- When we're doing (reading or writing) from a file you should always remember to close it!
  - Otherwise it might block other programs from opening it
- To do this we use the `close()` method

```
#include <fstream>
///stuff

//--- create input object
ifstream fin( "input.txt" );

//--- use input object like cin
//--- read 3 pieces of data from the file
fin >> data1 >> data2 >> data3;

//--- for mixing >> with getline()
fin.ignore();

//--- read in a line of data using getline()
getline( fin, lineOfData, '\n');

fin.close();
```

Output to a File

- To write to a file, we use an `ofstream` object.
- You declare it just like we did for `ifstream` objects:

```
ofstream variableName("filename");
```

- Now we can use the `ofstream` object just like we used the `cout` stream.
- Just like the `ifstream` object, by default a file is created in the current directory
  - To change this you can specify either an absolute or relative path.
- And just like with `ifstream` objects, we could first declare an `ofstream` object, then use the `open()` method.

Output to a File

- By default if the file already exists your program will overwrite it!
  - Be careful…..
- Sometimes we want to *append* to a file (add it to the end)
- To do that we need to provide an additional argument when opening the file

```
ofstream outFile;
outFile.open(filename, ofstream::app);
```

Output to a File

- Just like `ifstream`, we can use the `is_open()` method to check to see that the file exists and that it can be written to
- And we can use the `close()` method to close it.

```
#include <fstream>
//stuff

 //--- create output object
ofstream fout( "output.txt" );

if(fout.is_open())
{
     //--- writing stuff to file ---
     fout << "This text is saved in the file" << endl;

     //--- so is this data ---
     fout << data << data2 << data3 << endl;

     fout.close();
}
```

Reading to the End of a File

- Often we want to read everything from a file
- For example, maybe want to read all the numbers in a file to compute their average
  - But we don't know how many there actually are!
- We can use a loop to input the items over and over
- The >> operator returns `false` when it can't read from the stream
  - This will be our exit condition!

Reading to the End of a File

- Note: The following code will have problems if the >> operator encounters anything it can't interpret as numbers
  - We often must know the format of the input file

```
ifstream fin( "input.txt" );
double sum = 0.0;

// haven't read anything in yet
int count = 0; // number of values read
double value; // the value to be read in from file

while( fin >> value ) // try to read value
{
    // performed only when a value is read
    sum = sum + value; // add value to sum
    count++; // increment number of values read

}
fin.close() ; // we're done reading, so close the file.
double average = sum / count;
```

Reading to the End of File

- We can also ask the `ifstream` object directly if it reached the end of the file using it's `eof()` member.
  - `ifstream` objects keeps track of where in the file we currently are
  - **However**, for this to return true, the `ifstream` object must have tried, and failed, to read in more stuff (via >> or `getline`)

```
ifstream fin( "input.txt" );
double sum = 0.0;

// haven't read anything in yet
int count = 0; // number of values read
double value; // the value to be read in from file

while(true) // try to read value
{
    fin >> value;
    if(fin.eof())
        break;
    // performed only when a value is read
    sum = sum + value; // add value to sum
    count++; // increment number of values read

}
fin.close() ; // we're done reading, so close the file.
double average = sum / count;
```

User Supplied File Names

- Often we want to make custom file names
  - These may include information like our name, the date, etc..
- The problem is that the `stream` library pre-dates the `string` library so it doesn't know what strings are ☹
  - They only know the built-in (first-class) type of strings, `char *` (which we haven't seen yet. We'll see them in 172)
  - Therefore we can't send a `string` object to a streams constructor.
- Fortunately the `string` class foresaw this dilemma and provided a method, `c_str()`, that return an old-style string (`char *`) for us to use

User Supplied File Names

- The user supplies a name for the file

```
cout << “Please enter the name of the file: “;
string  filename;
cin >> filename;
```

- Open the file for input

```
ifstream fin(fileName.c_str());
```

- Or open the file for output

```
ofstream fout(fileName.c_str());
```

Input from Multiple Files

- Sometimes we may want to read from several files in the same program.
- Rather than create a new local file variable name for each file, as long as we don't need to read from both at the same time, we could use the same name.
- To do this we must:
  1. Close the previous file: `fin.close();`
  2. Clear all the internal information: `fin.clear();`
  3. And now we can open a new file into that stream: `fin.open(filename);`

Input from Multiple Files

```
ifstream fin("input1.txt");

double sum = 0.0; // haven't read anything in yet
int count = 0; // number of values read
double value; // the value to be read in from file

while( fin >> value ) {
    sum = sum + value; count++;
}

fin.close() ; // we're done reading, so close the
file.
double average1 = sum / count;

// Clear any internal flags associated with fin and
thus
// reset the file pointer to the beginning of the
file
fin.clear() ;
```

```
// Notice we do not re-declare fin,
// sum, count when we re-initialize
//them.

fin.open("input2.txt");

sum = 0.0;
int count = 0;

while( fin >> value ) {
    sum = sum + value; count++;
}
fin.close() ; // we're done reading,
          //so close the file.

double average2 = sum / count;
```

EOF from the console

- You may want to test by reading in from the console even though in the end you'll read in from a file.
  - Could be useful for development/debugging
- So we'll need a way to "simulate" the end-of-file
- To do this we type CTRL-D, which is called the "control character"

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

int main() {
    cout << "Start inputing integers (CTRL-D to stop):" << endl ;

    int n ;
    while( cin >> n ) {
        cout << "I read " << n << endl ;
    }

    cout << endl << "DONE!" << endl ;

    return 0 ;
}
```