Lec_11.md

Lecture 11 - C++ Input/Output (IO)

Oct.6/2020

The getline Function

Last lecture an interesting property of strings and **delimiters** was brought up:

- Trying to `cin` "hi there" will only place "hi" into the variable, as the " " character is a **delimiter**

The `getline` function

- Possible to get the entire line, *spaces included*, into a string
- Takes the ***entire*** input stream and places it into the string argument

main.cpp

```
#include <string>
int main(){
  string fullName;
  getline(cin,fullName); //or cin.getline(fullName,256)
  cout << fullName;
}
```

Notes:

1. You need to `#include <string>` to use `getline`
   - `getline` defined in string header
2. `getline(cin,fullName);`
   - Notice that we pass `cin` to `getline`
3. `cin.getline(fullName,256);`
   - Another way to call getline is `cin.getline(fullName,256)`
     - Equivalent funciton call to point 2
     - Reads 256 characters

User-Created Streams

The notion of streams in c++ is incredibly powerful, so what if we want to create our own input streams?

- **String Streams**
  - Read in and out of a string
- **File Streams**
  - Read in and out of a file on the hard drive

String Streams

We want to make **streams** out of strings

- This way we can read in/out of strings as **streams** easily

main.cpp

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

int main(){
  int anInteger;
  string inputLine;
  inputLine="204 113";

  stringstream myOwnStream(inputLine);
  myOwnStream >> anInteger;
  myOwnStream >> anInteger;

  return 0;
}
```

Notes:

1. Must include `<sstream>` header to use **string stream**
   - **String stream** defined in `<sstream>` header
2. `stringstream myOwnStream(inputLine);`
   - Creates a new **stream** called "myOwnStream"
   - Initializes this stream with `string` "inputLine"
3. `myOwnStream >> anInteger;`
   - Reads an integer from stream "myOwnStream" into "anInteger"
   - *Exactly* the same functionality as `cin`
     - General functionality: read from input stream into variable
       - `sstream` reads from **string stream** into variable
       - `cin` reads from **input stream** into variable

Properties of **String Stream**

- Just like `cin`, it ***fails*** silently
  - `sstream` has its own **fail flag**

Uses of **String Stream**

- Very useful when you have "line-oriented input"
  - Better able to deal with incorrect inputs
- Use `getline` to grab entire line
  - *Then* build `sstream` out of it and read values

main.cpp

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

int main(){
  int anInteger;
  string inputLine;

  getline(cin,inputLine);//OR cin.getline(inputLine,256);
```

```
    stringstream myOwnStream(inputLine);
    myOwnStream >> anInteger;
    myOwnStream >> anInteger;

    return 0;
  }
```

Notes:

1. `getline` into a string variable, and then `sstream` the variable to make a **string stream**
2. Cannot use `>>` **insertion operator** on a `string`
   - Call `>>` on **streams**, like **string stream**

File Streams

What about if we wanted to read from a file on a hard drive?

- Different from `cin` or `sstream`, as reading from a file has no directly observable UI elements
- Use `ifstream` and `ofstream`

Input File Streams

Reading from file

main.cpp

```
  #include <iostream>
  #include <fstream>

  using namespace std;

  int main(){
    ifstream inFile;
    int a;

    inFile.open("inputfile");

    inFile >> a;

    cout << a << endl;

    inFile >> a;

    cout << a << endl;

    inFile.close();

    return 0;
  }
```

If the file contains:

> 604 233 1233

First `cout` will print "604" Second `cout` will print "233"

Notes:

1. `#include <fstream>`
   - `<fstream>` includes methods for **file stream** input and output

    - Contains `ifstream` (**input file stream**) and `ofstream` (**output file stream**)
2. `ifstream inFile;`
    - Defines an object of **input file stream** type with name "inFile"
3. `inFile.open("inputfile");`
    - Opens a file with name "inputfile"
        - Usually "inputfile" includes the **extension**
            - For example, "inputfile.txt"
    - File "inputfile" ***must*** be in the same directory as the ***executable*** for the program
4. `inFile >> a;`
    - Read from the **input file stream** into variable a
        - Same usage as `cin`
            - Can also check for **fail flags**, same as `cin`
5. `inFile.close()`
    - close the **file stream** object
        - Just remember to call for **file streams**
            - May not noticeable break anything, but could be an issue for **output streams**

Properties of `ifstream`

- One `ifstream` object can open *1* file at once
    - Having multiple **file streams** open requires multiple `ifstream` objects
    - That being said, you can reuse `ifstream` objects to open multiple files
        - You just need to `.close()` a file before `.open()` a new file
- What happens if the file you are trying to open ("inputfile") has higher permissions (e.g. is not readable)
    - `ifstream` will throw an exception
    - If there is an issue with *opening the file* (e.g. `inFile.open();`)
        - For example, file could not be found, file is not readable, etc..
        - OS will ***throw exception***
        - Program will ***stop running***
    - If there is an issue with *reading from the file* (e.g. `inFile >> a;`)
        - `inFile` will raise **fail flag** silently
            - Same behaviour as `cin`
- What does `inFile >> a;` actually do?
    - Given file with first line: "604 233 1233"
        - `inFile >> a;` will put the first integer (since a is an integer) into a
            - In this case, "604"
            - It does not put the entire line into `a`
                - Remember, the **stream** is attached to the file
            - It puts the first variable (detecting type and noticing **delimiters**) into `a`
        - The next `inFile >> a;` call will put the second integer into a
            - In this case, "233"
- `inFile.close();`
    - Unattaches the stream from the file
        - *Cannot* use this stream any more

Output File Streams

Reading to (or printing to) file

main.cpp

```
#include <iostream>
#include <fstream>

using namespace std;

int main(){
  ofstream outFile;
  int a=8;

  outFile.open("outputfile");

  outFile << a;

  outFile.close();

  return 0;
}
```

After execution, the file will contain

> 8

Notes:

1. `#include <fstream>`
   - `<fstream>` includes methods for **file stream** input and output
     - Contains `ifstream` (**input file stream**) and `ofstream` (**output file stream**)
2. `ofstream outFile;`
   - Defines an object of **output file stream** type with name "outFile"
3. `outFile.open("outputfile");`
   - Opens a file with name "outputfile"
     - Usually "outputfile" includes the **extension**
       - For example, "file.txt"
   - File written to disk is sent to same directory as executable by default
4. `outFile << a;`
   - Writes the variable a from the **output file stream** into defined file
   - Uses the `<<` operator
     - Similar semantics to `cout`
5. `outFile.close()`
   - close the **file stream** object
     - Just remember to call for **file streams**
       - May not noticeable break anything, but could be an issue for **output streams**

Properties of `ofstream`

- Truncate vs Append
  - By default, `ofstream` truncates by default
    - `ofstream` places output variables at *start* of file
  - Can define how file is open
    - Can indicate "append" option to `ofstream.open();`
- `ofstream << a;` writes variable to file, on a single line
  - Add a `\n` character to add multiple lines to file
  - Can also use `endl` to end line and carriage return

beep boop~