Lec_10.md

Lecture 10 - C++ Input/Output (IO)

Nov.1/2020

The Fail flag

One thing to note:

main.cpp

```
int main(){
  int x = 9;
  string y;
  cin >> x;
  ...
  cin >> y;
}
```

If the input stream is:

> 'T' 'e' 'n' ' ' '2'

`cin` will **fail** at `cin >> x;`

- Attempting to place `characters` into `int`

However, `cin >> y` will run and place 'ten' into y

- Buffer has not been cleared
- **Fail** flag is *still* set to true

Checking for Valid Cin

main.cpp

```
#include <iostream>
using namespace std;
int main(){
  int anInteger;
  bool retry = true;
  while(retry){
    cin >> anInteger;
    if(cin.fail()){
      cout << "bad input" << endl;
    }else{
      cout << "read from cin~!" << endl;
    }
  }
  return 0;
}
```

If the input stream is:

> 'T' 'e' 'n' ' ' '2'

Notes:

- This is an *infinite loop*
  - On the first iteration of `while` loop:
    - `cin` attempts to place 'T' into variable `anInteger`
    - Fails to do so, and raises **Fail flag** because of type mismatch.
  - The `cin` stream is ***not modified*** between consecutive `cin` calls
    - The `cin` stream remains because the initial call raised the **Fail flag**
    - `cin.fail()` will always evaluate to true, since the stream ***has not changed***
  - `bool retry` never gets updated
- `anInteger` does not change if fail

So how do we deal with this issue?

- Flags are **persistent**
  - Flags remain until you reset them
  - No input until Flags are set

main.cpp

```
#include <iostream>
using namespace std;
int main(){
  int anInteger;
  bool retry = true;
  while(retry){
    cin >> anInteger;
    if(cin.fail()){
      cout << "bad input" << endl;
      cin.clear();
      cin.ignore(1000,'\n');
    }else{
      cout << "read from cin~!" << endl;
    }
  }
  return 0;
}
```

Notes:

- `cin.clear();`
  - This function *clears the flags*
    - Will turn off the **Fail flag**
  - Without calling `cin.clear();`, subsequent `cin` calls will not run
- `cin.ignore(1000,'\n');`
  - This function ignores characters in the stream
    - `cin.ignore();` takes two parameters. Either:
      a. Will *ignore 1000 characters*
        - 1000 is an arbitrary example, some systems have a limit of 256 characters per stream

      b. Will *ignore until '\n' is found*
- Handle errors in this order:
  - Clear flags with `cin.clear();`, ***and then*** ignore stream characters with `cin.ignore();`
    - The reverse order may not work, since `cin.ignore()` depends on flags

The Ignore Function

`cin.ignore()` is used to discard part of the stream. For the below example:

main.cpp

```
int main(){
  int x = 9;
  cin >> x;
}
```

If the input stream is:

> 'T' 'e' 'n' ' ' '2'

There are different ways to implement `cin.ignore();`

1. `cin.ignore(1000,'\n');`
   - Clears the first 1000 character of the stream, or until '\n'
2. `cin.ignore(1000,' ');`
   - Ignores spaces
   - More elegant way to clear problematic parts of the stream
   - Will read the '2' into x
3. `cin.ignore(3,'\n')`
   - Ignores 3 characters or until '\n' is reached
     - Can be used to ignore first part of a word/name
   - Will read the '2' into x

End of File

We would like to read inputs until the user has no more inputs to enter

- In APS105, we sometimes used '-1' to indicate the *last element* of sequence of integer inputs
  - Limits us to only *positive* integers

Enter the **End of File** (eof) error

- Indicates that there is no more input to be expected
- Encountering the **eof** when more input is expected raises **two** flags in `cin` :
  - **fail** flag is raised
  - **eof** flag is raised

On the terminal (when running executables), **eof** can be induced by using **ctrl+d**

- Special character, like '\0' or '\n'
- Is not included after '\n'
  - User must *specifically* enter **eof** using ctrl+d

main.cpp

```
#include <iostream>
using namespace std;
int main(){
  int x,sum=0;
  bool more = true;
  while(more){
```

```
      cin >> x;
      if(cin.eof()) more = false;
      else sum=sum+x;
    }
  }
```

Notes:

- Read characters through `cin` when `more==true`
- As soon as **eof** is provided, stop
- `cin.eof()` is an **accessor** method that returns the **eof** flag

This works for input:

> '2' '0' '4' ' ' '1' '1' '3' eof

`sum=317`

- `317=204+113`

However for this input:

> '2' '0' '4' ' ' ' ' 't' 'e' 'n'

`sum=408` . Why?

- Notice the two spaces
  - Upon reaching the second space character (second **delimiter**), `cin` will read from stream again
    - The issue is the **eof** flag is still `false` , so the `if` statement goes to `else` again, so `sum=sum+x;` runs again.

How do we fix this?

main.cpp

```
#include <iostream>
using namespace std;
int main(){
  int x,sum=0;
  bool more = true;
  while(more){
    cin >> x;
    if(cin.fail()){
      if(cin.eof()){
        more = false;
      }else{
        sum=sum+x;
      }

    }
  }
}
```