Topic 11 – Verifying valid input

Error Checking inputs

- How do we check for valid input?
  - Thus far we've been using do-while loops
  - Why do while?
    - Always has to run one time

For Example:

```
do
{
     invalid = false;
    cout << "How old are you? ";
    cin   >> age;
    if (age < 0)
    {
        cout << "You can't be less than zero – please try again";
        invalid = true;
    }
}
while (invalid);
```

Will that always work?

- What if the interaction is like this:

  How old are you? C

What if the input doesn't match the datatype?

Expecting #s for float and int

- The extraction operator expects #'s when extracting into a float or an int.

- When characters are entered two things happen

  First

  → The extraction operator returns the Boolean value false

  → This indicates that extraction failed (puts it in a 'fail state')

  Second

  → The invalid characters are NOT extracted from the input buffer

  → The extraction operator keeps trying to read from the input buffer

  → THUS the infinite loop

Fixing the problem

There are two problems we need to solve
→ thus is a two-step process

First → we need to reset the 'fail state' or prepare cin for normal operations

Second → we need to clear out all the characters left in the input buffer

Resetting the fail state

First, we need to "reset the fail state"

→This prepares the program for normal input operations

To do this we use cin.clear()

```
cin.clear();
```

→This prepares the input stream for normal use again

Now we must clear the offending characters…

Clearing the input buffer

- So far… we have been putting in an arbitrarily large # for cin.ignore (10000, '\n')

  10000 is an arbitrarily large

  - What if 10000 isn't large enough?
    ➔ we need to clear the entire stream

- How do we find what the largest stream is when it is system dependent?

  numeric_limits<streamsize>::*max()*

Numeric Limits and Streamsize

numeric_limits is a predefined template in C++ that determines the limits (max or min) for an implemenation-specific type

→ this works with any integral or floating point type

streamsize is an implementation-specific type that represents the size of the I/O buffer

(in this case we are concerned about the input buffer)

What we need is the maximum size of the input buffer

numeric_limits<streamsize>::*max()*

NOTE:
To use numeric_limits we need to #include <limits>
To use streamsize we need to #include <ios>

How to use it

```
#include <limits>
#include <ios>
...

if(!(cin >> command))
{
    cout << "\nPlease input an integer (0-5) ";
    cin.clear();

    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    invalid = true;
}
else if (command < 0 || command > 5)
{
    cout << "Command #: " << command;
    cout << " is an invalid entry - please try again\n\n";
    invalid = true;
}
```

DON'T use another cin just this one will suffice

This checks to see if our extraction is valid
→ If it is then it performs the exctraction
→ if not It returns false

Clears the 'fail state'

Clears all characters from the input buffer

Checks our other valid inputs

NOTE:
This should be within your do-while loop

Compiler specifics

- As you know… not all compilers adhere to c++ standards
  - Some don't include numeric_limits or streamsize
- In this case → use a very high arbitrary # for cin.ignore

  cin.ignore(10000, '\n');
- Eclipse uses numeric_limits and streamsize
  - So… in this class use them

Alternative Method

atoi() and atof()

- These functions convert c-strings into integers or floating point numbers → they ignore characters

  atoi () → converts c-strings to integers

  atof() → converts c-strings to double

- Strings allow us to read in any type of character
  - We can read all of our inputs as strings
  - Then convert them back to int or float

→ If atoi() or atof() read in only characters they return 0 or 0.0 respectively

→ We can error check for 0 or 0.0

Using atoi and atof

```
string commandStr;
int command;

getline(cin,commandstr);
command = atoi(commandStr.c_str());

if (command == 0 && commandstr[0] != '0')
{
    cout << "\nPlease input an integer (0-5) ";
    invalid = true;
}
else if (command < 0 || command > 5)
{
    cout << "Command #: " << command;
    cout << " is an invalid entry - please try again\n\n";
    invalid = true;
}
```

Reads input in as a string

Converts input into an int

This is iffy what if they type 0a?

Which approach to use

- The first approach has some advantages
  - It doesn't involve converting from one type to another which can be problematic
  - What if 0 is valid? Or is someone types in a leading 0 and then garbage
- The second approach works to an extent

→ the first approach is better form