# Lecture 9: Getting Loopy

PIC 10A

Todd Wittman

## Sec 3.10: Input Validation

- So far we've assumed the user gives us proper input. But what if they don't?

  Enter a number: four

- When you do a cin statement, it actually returns a boolean: *true* if you were able to read the value correctly, *false* if it failed to read the specified type.
- Returns the boolean *true* for a success, so we can use cin in an if statement.
- Alternatively you could use the member function cin.fail( ) described in Sec 4.4.

## Example: Input Validation

```
int main ( ) {
    int x;
    cout << "Enter a number: ";
    if ( cin >> x )  {
        cout << "Your number is " << x << ".\n";
        return 0;
    }
    else  {
        cout << "Supposed to be a number.";
        return 1;
    }
}
```

This line does 2 things:
1.) Gets input *x* from user.
2.) Checks if read successful.

This will end the program.
Use the return command with caution.

## Sec 3.6: The while loop

- Sometimes it's useful to repeat a block of code, like HW 3.
- We call a block of code that is repeated a loop.
- Each repetition of the code is called an iteration.
- The while loop acts like a repeating if statement, repeating the code below it as long as the boolean statement is true. Also called a stopping condition.

```
while ( statement is true )  {
        ** STATEMENTS **
}
```

- The braces { } enclose the code to repeat.
- Like with if, indenting makes it easier to read.

## Example of a while loop

```
int x = -1;
while ( x <= 0 )  {
        cout << “Enter a postive number: “;
        cin >> x;
}
cout << “Your number is “ << x << “.\n”;
```

- What does this code do?
- How many times will it repeat?
- Why was it necessary to initialize x=-1?

## Looping Until the User Says Stop

- Mulitply two numbers entered by the user.
- Ask the user if she wants to continue.
- Stop when the user says so.

```
double a, b;
string response = “y”;
while ( response == “y” || response == “Y” )  {
        cout << “Enter two numbers: “;
        cin >> a >> b;
        cout << “Their product is “ << a*b << “.\n”;
        cout << “Do you want to do it again?  (y/n)”;
        cin >> response;
}
```

## Looping a Fixed # of Times

- Suppose we want to say hello to Gandalf 10 times.

```
int counter = 1;
while ( counter <= 10 )  {
        cout << “Hello Gandalf!\n”;
        counter ++;
}
```

- What would happen if we deleted the counter++ line?

## Looping While Input is Valid

- Add up a list of numbers typed by the user. Stop when they enter something that is not an int.

```
int num;
int sum = 0;
cout << "Enter a number: ";
while (cin >> num) {
        sum += num;
        cout << "Enter a number: ";
}
cout << "Your sum is " << sum << ".\n\n";
```

```
Enter a number: 3
Enter a number: 1
Enter a number: 6
Enter a number: Gandalf
Your sum is 10.
```

- We'll see this type of loop again when we talk about reading files.

## Infinite Loops

- An infinite loop is a loop that never stops.
- An infinite loop is *very bad* programming style.
- Make sure your while loop terminates.

Case 1: Update the variable that controls the loop.

```
int counter = 1;
while ( counter <= 10 ) {
    cout << "Hello!\n";
}
```

Case 2: Make sure your boolean condition will eventually be false.

```
int counter = 1;
while ( counter >= 0 ) {
        cout << "Hello!\n";
        counter ++;
}
```

## Vegas Baby!

- The roulette wheel at Caesar's is broken and always lands on red. Winning a bet on red doubles your money. Starting with $1, how many bets will you have to make to earn $1,000,000?

```
int money = 1;
int numberBets = 0;
while ( money < 1e6 ) {
        money = 2*money;
        numberBets ++;
}
cout << "You need to make "
     << numberBets << " bets.\n";
```

Scientific notation is often easier than typing 6 zeros.

Recall ++ adds one to the number. Could also use numberBets = numberBets+1

## The Factorial: Loopy Math

- The factorial N! of a number N is defined as

  N! = N(N-1)(N-2)...(3)(2)(1)
- For example: 4! = (4)(3)(2)(1) = 24
- Not to be confused with the C++ *negation* !
- For our purposes, the factorial operator ! is only defined for positive integers.
- Write a program that gets a number N from the user and computes N!

  Enter your number: 4

  4! = 24

## The Factorial: First Attempt

```
int N;
cout << "Enter your number: ";
cin >> N;
int M = N;
int factorial = M;
while (M > = 1)  {
        M --;
        factorial = factorial * M;
}
cout << N << "! = " << factorial;
```

Sets M=N and then computes factorial = M*(M-1)*(M-2)...

- There's a very serious error in the code above...

## The Factorial: Second Attempt

- The previous code would output factorial = 0 every time.
- Iterated one too many times, so it multiplied by 0 in last step.

```
int N;
cout << "Enter your number: ";
cin >> N;
int M = N;
int factorial = M;
while (M >  1)  {
        M --;
        factorial = factorial * M;
}
cout << N << "! = " << factorial;
```

Small change in the stopping condition from (M>=1) to (M>1) makes a huge difference.

## Sec 3.5: Boolean Variables

- Recall that a boolean (bool) takes only true/false values. (Could also give it 1/0, or any positive # / zero.)
- We can use a boolean as a loop condition.

```
int x;
bool isPositive = true;
while ( isPositive )   {
        cin >> x;
        if ( x < 0 )   {
                isPositive = false;
        }
        cout << "Your number was " << x << "./n";
}
```

## Nested Loops

- A nested loop is a loop within a loop.
- How many times does this code say hello?

```
int outer = 1;
int inner;
while ( outer <= 5 )  {
        inner = 1;
        while ( inner <= 10 )  {
                cout << “Hello Gandalf!\n”;
                inner++;
        }
        outer++;
}
```

Notice inner is reset to 1 every iteration of the outer loop. Without it, how many times would we say hello?

## Another Nesting Example

- This program repeatedly doubles a number until it reaches 100.

```
string response = “y”;
int num;
while ( response == “y” || response == “Y” )   {
        cout << “Enter a number: “;
        cin >> num;
        while ( num <= 100 )   {
                cout << num << “  “;
                num *= 2;
        }
        cout << “Do you want to continue? (y/n)  “;
        cin >> response;
}
```

# Looking Into the Future...

- Friday we'll learn about other types of loops.
- The do-while loop always run at least once.

```
do {
    **STATEMENTS**
}  while (statement is true);
```

Note the final semi-colon on the while statement.

- The for loop runs for a fixed number of iterations.

```
for (int i = 1; i <= 10; i++) {
    **STATEMENTS**
}
```

The for statement takes 3 arguments (initialization; stopping condition; update)