Constants
Part 1

Written by Rachel J. Morris, last updated 2015-01-14

Const Variables

The **const** keyword allows us to declare variables that cannot be changed after they are initially assigned.

A variable that is declared as **const** must be initialized to some value.

```
const string version = "v2.0";
```

```
const int screenWidth = 640;
const int screenHeight = 480;
```

Const Variables

Using **const** is a good way to avoid hard-coding numeric values in your program.

By declaring a **const** variable, you give that number a label – a name – so that your code is more clear.

```
int main()
{
    const int screenWidth = 20;
    const int screenHeight = 10;

    for ( int y = 0; y < screenHeight; y++ )
    {
        for ( int x = 0; x < screenWidth; x++ )
        {
            cout << "*";
        }
        cout << endl;
    }

    return 0;
}
```

Const with Pass-By-Reference

Const is also useful with pass-by-reference parameters.

Sometimes, you want to pass an argument **by reference** not because you want to change its value, but because it would be **costly** to copy it, such as when passing-by-value.

```
struct User
{
    string firstName, lastName;
    string city, state;
    int zipCode;
};
```

```
void UserFunction( list<User> lst );
```

Non-primitive objects, such as **strings** or classes / structures defined by the user, **are potentially costly to copy.**

Const with Pass-By-Reference

Const is also useful when pass-by-reference parameters.

When making a copy of a big list of Users, every internal member of the **User** class/struct must also be copied.

```
struct User
{
    string firstName, lastName;
    string city, state;
    int zipCode;
};
```

A **string** is not a primitive type – it is also a class with its own members and functions.

```
void UserFunction( list<User> lst );
```

Const with Pass-By-Reference

```
void UserFunction( const list<User> & lst );
```

Instead of passing the argument as a pass-by-value, instead we can pass-by-reference, but add the **const** keyword to ensure that the argument will not be changed.

```
void DisplayName( const string& first, const string& last );
```

It is **good practice** to make a habit of passing any objects as const references
(when you need the value to not change).

Other uses of Const

We will revisit **const** when we cover classes, as it can also help protect the **members** of a class, by marking an object's functions **const**.

Const can also come in handy when dealing with pointers and functions.

More on that later, too...