More Functions!

Variable Scope

- Scope: where can a variable be used?
- Local Scope: variable is only available locally (within a function, loop, etc.)

```
int foo(int x)
{
  int value = x * 2;
  for(int k = 0; k < value; k++)
  {
    value += (k % 3);
  }
  value += k; // ERROR
  return value;
}
```

Variable Scope

- Global Scope: variable is available everywhere in the source code
  - often a bad idea!

```
int lowervalue = 0;

int foo(int x)
{
  int value = x * 2;
  for(int k = lowervalue; k < value; k++)
  {
    value += (k % 3);
  }
  return value;
}
```

Variable Scope

- Local variables can hide other variables

```
int lowervalue = 0;

int foo(int lowervalue)
{
  int value = lowervalue * 20;
  for(int k = lowervalue; k < value; k++)
  {
    value += (k % 3);
  }
  return value;
}
```

Variable Scope

```
int value = 99;
int foo(int lowervalue)
{
  int lowervalue = 20; // ERROR
  for(int k = lowervalue; k < value; k++)
  {
    value += (k % 3);
  }
  return value;
}
```

Practice: What is the result?

```
int number = 0;

int foo(int value)
{
  int number = value * 20;
  for(int value = 0; value < number; value++)
  {
    value += (k % 3);
  }
  return value;
}

int main()
{
  int number = 10;
  cout << foo(number);
  return 0;
}
```

Static Local Variables

- What happens here?

```
void foo()
{
  int value = 20;
  cout << " value: " << value << endl;
  value *= 22;
}

int main()
{
  foo();
  foo();
}
```

Static Local Variables

- Sometimes we want a function to retain a value between uses
  - static local variables

```
void foo()
{
   static int value = 20; // This only happens once
   cout << " value: " << value << endl;
   value *= 2;
}
```

What does this do?

```
int main()
{
   foo();
   foo();
}
```

Practice: Static Local Variables

- Write a function that will count the number of times it has been called and print that count to the screen.

- Write a function that will take one integer as a parameter and produce a running sum and running average of the values used as arguments when it is called.

Default Arguments

- "Default arguments are passed to the parameters automatically if no argument is provided in the function call" p353

```
void stars(int numberOfStars = 5)
{
   for(int i = 0; i < numberOfStars; i++)
   {
       cout << "*";
   }
   cout << endl;
}
```

What does this do?

```
int main()
{
stars(10);
stars();
}
```

Default Arguments

```
// specify the default arguments the first time
// you define the function
void stars(int numberOfStars = 5);

int main()
{
  stars(10);
  stars();
}

// do not redefine the default arguments here
void stars(int numberOfStars)
{
   for(int i = 0 ;i < numberOfStars; i++)
   {
       cout << "*";
   }
   cout << endl;
}
```

Practice: Default Arguments

- Write a function that will accept either one or two integers as parameters and return the area of a square (if one parameter is specified) or a rectangle (if two parameters are specified)

Overloading Functions

- "Two or more functions may have the same name as long as their parameter lists are different." p365
  - return data type is *not* considered

```
int area(int length);
int area(int length, int width);

int square(int value);
double square(double value);

int increment(int value); // ERROR
double increment(int value); // ERROR
```

Practice: Overloaded Functions

- Write two overloaded functions that will produce the sum and average of three integers or three doubles.