Functions
Chapter 6, 8

Review

- Functions
  - Prototype
  - Call
  - Definition
- Passing arguments
  - By value
  - By reference

Arrays as Function Arguments (8.8)

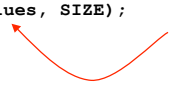
- You can pass an array as an argument to a function

```
void printIntArray(int arr[], int size);

int main()
{
 int values[] = {5, 6, 0, 1, 2, 3, 4};
 const int SIZE = 7;

 printIntArray(values, SIZE);

 return 0;
}
```



Arrays as Function Arguments

```
void printIntArray(int arr[], int size)
{
  for(int i = 0; i < size; i++)
  {
    cout << arr[i] << endl;
  }
}
```


Passing Single Array Elements

- Passing single array elements is like passing in a single variable

```
void showValue(int num);

int main()
{
 int values[] = {5, 6, 0, 1, 2, 3, 4};
 const int SIZE = 7;
 for(int i = 0; i < SIZE; i++)
 {
  showValue(values[i]);
 }
 return 0;
}
```


Passing Arrays into Functions

- Arrays are always passed by reference
- What does this mean?

Q.1. Practice

- Write a function that will accept an integer array and a size for that array, and return the sum of all the elements in the array
  - `int sumArray(int array[], int size);`

Q.2. Practice

- Write a function that will accept an integer array and a size for that array and return the highest value in the array
  - `int getHighest(int array[], int size);`

Variable Scope (6.10)

- 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;
}
```

Q.3. Practice: What is the result?

```
int number = 0;

int foo(int value)
{
  int number = value * 10;
  for(int value = 0; value < number; value++)
  {
    value += number;
  }
  return value;
}

int main()
{
  int number = 2;
  cout << foo(number);
  return 0;
}
```


Q.4. Static Local Variables (6.11)

- What happens here?

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

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


Q.6. Static Local Variables

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

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

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

Q.7. Practice: Static Local Variables

- Write a function that will count the number of times it has been called and print that 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 (6.12)

- "Default arguments are passed to the parameters automatically if no argument is provided in the function call" p343

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

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;
}
```

Q.8. 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 (6.14)

- "Two or more functions may have the same name as long as their parameter lists are different." p354
  - 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
```


Q.9. Practice: Overloaded Functions

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