Good things to know

- If a function, struct, or class is defined in Main.cc, then it must be above int main().
- Two good online resources for C++ are [www.cplusplus.com](http://www.cplusplus.com) and search engine hits on stackoverflow.com

Suggested Coding Practice

- Write self documenting code
- Use very detailed variable names
- Have variable names start with lower case letters
- Class member names should start with an underscore (_)
- Indent your code 2–3 spaces within functions and loops

Input/Output From/To Terminal

cin

Example:

```cpp
double usersFavoriteNumber = 0;
std::cin >> usersFavoriteNumber;
```

cout

Example:

```cpp
double randomNumber = 1;
std::cout << "random number = " << randomNumber << endl;
```

printf

Example:

```cpp
double randomNumber = 1;
printf("randomNumber = %.2f", randomNumber);
```
Functions

General function setup:

```cpp
returnType
functionName(input1Type input1)
{
    // write some code here to carry out function details
    return answerWithFunctionsReturnType;
}
```

Example:

```cpp
double
calculateTheAreaOfAnEllipse(double minorRadius, double majorRadius)
{
    double area = M_PI * minorRadius * majorRadius;
    return area;
}
```

Example to try: write a function to return n terms of the Fibonacci Sequence.

Pass by Value or Reference

Functions can take in arguments as pass by value or by reference.

- pass by value = copy the variable from the original memory and I’ll work on the copy
- pass by reference = provide the memory address and I’ll use the original variable

Pass by Value

Example:

```cpp
double
sumAStlVector(vector<double> vectorOfDoubles)
{
    double sum = 0;
    for(size_t vectorIndex = 0; vectorIndex < vectorOfDoubles.size(); vectorIndex++)
    {
        sum += vectorOfDoubles[vectorIndex];
    }
    return sum;
}
```

```cpp
int main(int argc, char** argv)
{
    vector<double> vectorOfDoubles;
    for (unsigned int numToPushBack = 0; numToPushBack < 10; numToPushBack++)
    {
        vectorOfDoubles.push_back(numToPushBack);
    }
    double totalSum = sumAStlVector(vectorOfDoubles);
}
```
Pass by Reference

Example 1:

```cpp
double
sumAStlVector(vector<double> & vectorOfDoubles){
    double sum = 0;
    for (size_t vectorIndex = 0; vectorIndex < vectorOfDoubles.size(); vectorIndex++){
        sum += vectorOfDoubles[vectorIndex];
    }
    return sum;
}

int main(int argc, char** argv){
    vector<double> vectorOfDoubles;
    for (unsigned int numToPushBack = 0; numToPushBack < 10; numToPushBack++){
        vectorOfDoubles.push_back(numToPushBack);
    }
    double totalSum = sumAStlVector(vectorOfDoubles);
}
```

Example 2:

```cpp
double
sumAStlVector(vector<double> * vectorOfDoubles){
    double sum = 0;
    for (size_t vectorIndex = 0; vectorIndex < vectorOfDoubles->size(); vectorIndex++){
        sum += (*vectorOfDoubles)[vectorIndex];
    }
    return sum;
}

int main(int argc, char** argv){
    vector<double> vectorOfDoubles;
    for (unsigned int numToPushBack = 0; numToPushBack < 10; numToPushBack++){
        vectorOfDoubles.push_back(numToPushBack);
    }
    double totalSum = sumAStlVector(&vectorOfDoubles);
}
```

**Const**

Passing by value can be computationally expensive, so it is sometimes preferential to pass by reference. However, with great power comes great responsibility. If a function receives an argument as pass by reference it
can now change the value. To restrict this ability to change the value we can use const. Hence const means that you can safely hand over the address to the input data and rest assured the function will not change any values.

Example 1:

```cpp
void evilFunction(double & pristineData){
    pristineData += 7;
}
```

Example 2:

```cpp
void evilFunction(const double & pristineData){
    pristineData += 7;
}
```

Note that Example 1 will compile without error. However, Example 2 will throw an error while compiling, letting you know you have made a logic error.

## Structs and Classes

Structs and classes are public and and private respectively, by default.

**Struct Setup:**

```cpp
struct NameOfStruct {
    // constructor
    NameOfStruct(inputType1 input1, inputType2 input2) :
        _input1(input1),
        _input2(input2){
        // do stuff when the object is created
    }
    // specifying the type and variables within the struct
    inputType1 _input1;
    inputType2 _input2;
}
```

**Class Setup:**

```cpp
class NameOfClass {
    public:
    }
```
// constructor
NameOfClass(inputType1 input1, inputType2 input2) :
    _input1(input1),
    _input2(input2) {
    // do stuff when the object is created
}

// methods of the class (functions built into the class)
methodReturnType
method1(inputType3 input3) {
    // do stuff
    return methodReturnType;
}

// specifying the type and variables within the class
private:
    inputType1 _input1;
    inputType2 _input2;
};

Example using struct and classes:

struct Puppy{
    Puppy(double lengthOfFur):
        _lengthOfFur(lengthOfFur){
    }
    double _lengthOfFur;
};

class TrimmerPassByValue{
public:
    TrimmerPassByValue (Puppy puppy):
        _puppy(puppy){
            _numberOfTrimmingsPerformed = 0;
        }
        void trimPuppyFur(double trimLength){
            _puppy._lengthOfFur -= trimLength;
            _numberOfTrimmingsPerformed++;
        }

    private:
        Puppy _puppy;
        unsigned int _numberOfTrimmingsPerformed;
};
```cpp
class TrimmerPassByReference {
public:
    TrimmerPassByReference (Puppy * puppy):
        _puppy(puppy){
            _numberOfTrimmingsPerformed = 0;
        }

    void trimPuppyFur(double trimLength){
        _puppy->_lengthOfFur -= trimLength;
        _numberOfTrimmingsPerformed++;
    }

private:
    Puppy * _puppy;
    unsigned int _numberOfTrimmingsPerformed;
};

int main(int argc, char** argv){
    Puppy lucky(10);
    printf("Lucky's fur length before trimming is %f inches long\n",lucky._lengthOfFur);
    TrimmerPassByValue trimmerForLucky(lucky);
    trimmerForLucky.trimPuppyFur(5);
    printf("Lucky's fur length after trimming is %f inches long\n",lucky._lengthOfFur);
    // note that lucky's fur length has not changed because we "made a clone"
    // and trimmed the fur of the clone/copied puppy

    Puppy furBall(7);
    printf("FurBall's fur length before trimming is %f inches long\n",furBall._lengthOfFur);
    TrimmerPassByReference trimmerForFurBall(&furBall);
    trimmerForFurBall.trimPuppyFur(3);
    printf("FurBall's fur length after trimming is %f inches long\n",furBall._lengthOfFur);
    // note that furBall's fur length has changed because we always passed along the
    // original puppy using pass by reference

    return 0;
}
```