C++

Point.h

```
#include <math.h>
#ifndef POINT_H
#define POINT_H
class Point
{
    Point();
    Point ( float x, float y, float z );
    ~Point ();
    void normalize();
    float x, y, z;
private:
    float w;
};
#endif
```

mathstuff.h

```
float getAverage ( float values[], int numValues );
```

Point.cpp

```
#include "Point.h"      #include <math.h>
Point::Point()
{ x = y = z = 0; }

void Point::Normalize()
{
    float sum = x*x + y*y + z*z
    return sqrt(sum);
}
```

main.cpp

```
#include "Point.h"   #include "mathstuff.h"
#include <stdio.h>

void swap (int& a, int& b);

int main()
{
    int a = 1;
    int b = 2;
    swap(a,b);
    printf("a is %i, b is %i", a, b);

    Point p(1,2,3);
    p.normalize();

    float array[50];
    for (int i=0; i<50; i++)
        array[i] = ((float)i) / 100.0f;
    printf("average is %1.3f \n", getAverage(array, 50));
    return 0;
```

Files

- 2 types: .h & .cpp (.c indicates C not C++)
- class and function declarations go in headers
- implementation goes in .cpp

Preprocessor

- directives for the compiler
→ #include < x.h > → includes a library, " " local, < > standard
→ #define MY_CONST 4 → kind of a constant var.
→ #ifdef → conditional inclusion
#endif

Functions

→ can exist outside of a class
→ int main() is the entry point
→ pass parameters by default or value

```
float doStuff ( int& byRef, int byValue)
{
    byRef++;
    byValue++;
    return 2.5f;
}
```

→ put void as the type if not returning a value

Data Types

- bool (Boolean) 0 = false, 1 = true
- int & double are same
- long (8 bytes), short (2 bytes)
- float
- unsigned

Variables

- C++ does not check that you've initialized
- global variables
- const instead of final - const int BLAH = 30;

Classes

- default public, can create protected, private with
  private:
  protected:
- only contains definition, implementation is elsewhere
- semicolon at end of class
- when implementing `Classname :: functionName ();`

Objects

- no official "object" type
  → a variables holds a value, it does not refer to an object
- don't have to use "new"
  `Point p(1,2,3);`
- if you don't supply parameters, default constructor used
  `Point q;`
- assignment copies values
  `Point r = q;`
  `r.x = 3;`

Pointers

- a variable that refers to an object (contain a memory address)
- indicated by a * so
  `int * a;` → pointer to an int
  `a = 5;` → bad, a refers to memory address 5
  `*a = 5;` → good (maybe)
  `int y;`
  `a = &y;` → think of & as address of
  `*a = 5;` → better!
  `Point * s;`
  `s = new Point(0,0,0);`
  `s -> x = 5;`
  `delete s;` → no garbage collection
- pointers cause trouble, don't use them if you don't have to.
  `char ** argv;` pointer to a pointer to a char → pros an array;

I/O

```
#include <iostream.h>

cout << myVar << " some string \t " << endl;
cin >> x;
```

STL Vectors

- standard template library
- data structures that handle any type

```
#include <vector.h>
using namespace std;

vector <float> numbers;
vector < vector <float> > grid;
  :
  {
      numbers.resize (10);
      numbers.clear ();
      numbers.push_back (5);
      numbers [0] = 6;
      numbers.pop_back();
      if (numbers.size() > 5)
          return;
```

Compiling multiple files:

```
g++ -c Point.cpp
g++ -c main.cpp
g++ -o a.out Point.o main.o
```