Lecture 6 Notes

Classes

There are generally two types of data types: primitive data types and composite data types.

The primitive data types you already know: bool, char, int, double. These data types are primitive because they are built-in to the C++ programming language. You do not need to include a header like iostream or vector to use them.

Composite data types are composed of other data types. For example, an array is a type of composite data type which can store multiple instances of the same data type. Examples of array data types include: int array, double array, char array.

Now you will learn about the class: a type of composite data type. A class can be composed of functions and data types; unlike an array, these data types do not have to be the same. Two types of classes that you have been using include vector and string. Any variable with a class-based data type is known as an object. So cin and cout are objects, their class-based data types are istream and ostream respectively.

Here is an example of a class:

```
class Vector2 {
public:
    double x;
    double y;
};
```

This is a Vector2 class. It can be used to represent a position, velocity, acceleration, or direction in 2 dimensional space. It contains two variables: x and y. A variable inside a class is known as an attribute. We can access these attributes using the member access operator:

```
Vector2 v;
v.x = 20.0;
v.y = 100.0;
```

The real power of classes comes from methods. A method is a function inside a class, let us expand the Vector2 class with a few methods:

```
#include <cmath>

class Vector2 {
public:
    double x;
    double y;

    void zero()
    {
        x = 0.0;
        y = 0.0;
    }
```

```
    double magnitude() const
    {
        return sqrt(x*x + y*y);
    }
};
```

The first method, zero, will assign the value zero to the x and y attributes. The magnitude method will calculate and return the magnitude of our Vector. This method uses the sqrt function to perform a square root calculation, sqrt is provided by the cmath library.

Notice that the magnitude method has a const suffix whereas the zero method does not. Using const as a suffix to your method specifies that the method will not modify any attributes inside the class. The zero method modifies the attributes of Vector2 by assigning zero to x and y. The magnitude method does not modify x or y; it only reads their values to perform a calculation.

Scoping

You might be, and should be, wondering about the public keyword used inside our class declaration. public means that the proceeding members of our class, attributes and methods, can be accessed with the member access operator from outside the class. By default, all class members are private meaning that only methods inside the class can access the other members.

Here is a modification of the Vector2 class using public and private scoping:

```
class Vector2 {
private:
    double x, y;
public:
    double magnitude() const
    {
        return sqrt(x*x + y*y);
    }

    double direction() const
    {
        return atan(y / x);
    }

    void set(double mag, double dir)
    {
        x = cos(dir) * mag;
        y = sin(dir) * mag;
    }
};
```

In this version of Vector2, the components of the vector are hidden from outside access; instead, we have methods for assigning and retrieving the vector's direction and magnitude.

In general there are two types of methods: accessor methods and mutator methods. Accessor methods query information about the class without modifying any attributes. In C++, accessor methods have a return value and a const suffix. In our Vector2 class, the magnitude and direction methods are accessor methods because they do not modify the state of the class. Mutator methods modify the state of the class, usually by modifying the attributes. Our Vector2 class contains the set method, which assigns a magnitude and direction to our vector, hence the set method is a mutator method.