Headers

```
#include <iostream>
#using namespace std;
```

Structure Example

Declare Structure

```
struct Order {
    int modelNumber;
    string phone;
    float cost;
};
```

Main Function, define the structure variable

```
int main( ) {
    Order item1;
```

Give values to structure variables

```
    item1.modelNumber = 10;
    item1.phone = "Samsung Galaxy S";
    item1.cost = 1099.99;
```

Display/Output structure members

```
    cout << "Order, " << item1.phone << item1.modelNumber;
    cout << "This item costs: $" << part1.cost << endl;
}
```

Output:

```
Order, Samsung Galaxy S10
This item costs: $1099.99
```

Whenever finished with output or line,
place 'end line' indicator: endl

Class Example: point.h

Declare class with the class name

```
class Point
{
```

Provide the class specifiers, set as private

```
private:
    double x;
    double y;
```

Set x as the first variable, y as the second variable

```
void set (double first, double second)
{
    x = first;
    y = second;
}
```

Class Example: point.h (cont)

Provide the class properties and methods, set as public

No-parameter constructor, when an object for this class is created, the Point class values (x and y) will be set to 0

```
public:

    Point( ) : x(0.0), y(0.0)
    {
        cout << "New Point object created/instantiated." << endl;
    }
```

Alternatively, the above constructor can also be written as:

```
    Point( )
    {
        x = 0.0;
        y = 0.0;
    }
```

A two parameter constructor, set the x and y values

```
    Point (double first, double second) : x(first), y(second)
    {
        cout << "New object, values (x,y) are << x << "," << y << endl;
    }
```

For returning individual values of x and y

```
    double getX( ) { return x; }
    double getY( ) { return y; }
```

Display the current values of x and y

```
    void display ( ) {
        cout << x << "," << y << endl; }
```

Overloading the function with two new doubles as input parameters

```
void add(double first, double second) {
    x += first;
    y += second; }
```

Overloading the function with another Point object as the input

```
void minus(Point pp) {
    x = x - pp.x;
    y = y - pp.y ; }
```

Add the destructor

```
~Point ( ) {
    cout << "Point object for x,y values of " << x << "," << y << "des-
troyed." << endl; }
```

Overload the function with another Point object and return the result

Class Example: point.h (cont)

```
Point mult(Point pp1) {
    Point pp2;
    pp2.x = x * pp2.x;
    pp2.y = y * pp2.y;
    return p2;
}
```

Class Example: point.cpp

```
#include "point.h"
```

Main Function

```
int main( ) {
    Point ptA;
    Point ptB(1.0, 3.0), ptC (5.0, 8.0);
    ptB.add(6.0,2.0);
    ptC.minus(ptB);
    ptA = ptC;
    Point ptD = ptB.mult(ptA);
    ptA.display( );
    return 0;
}
```

Object Values:

ptA = (0.0,0.0); ptB(1.0, 3.0); ptC(5.0,8.0);
ptB.add(6.0,2.0)= (1.0+6.0, 3.0+2.0);
ptC.minus(ptB)= ((5.0-1.0),(8.0-3.0)) = (4.0,5.0);
ptA = new ptC value = (4.0,5.0);
ptD = ptB.mult(ptA) = ((4x7),(5x5)) = (28, 25);

Note:

About Properties and Classes

A **property** is a defined attribute of the object, for example properties for object Point include 'x' and 'y'
A **member**, often a function, is an action that is performed on the object and its properties.

```
class myClass {
    private: property = value;
    public: method { (action on value); }
}
main { myClass newObject; }
```

Internal and External Class Methods

If the Class methods are written externally, a prototype must be declared within the class first.

```
Point (double first, double second) : x(first), y(second);
double getX( );
double getY( );
void display ( );
```

External syntax for returning class:

```
className :: className(any parameters) { }
```

```
Point :: Point (double first, double second) : x(first), y(second) {
    cout << "Object made with values x,y of " << x << "," << y <<
endl;
}
```

External syntax for returning variable (or void):

```
returnDataType className :: methodName(any parameters) { }
```

```
double Point :: getX( ) { return x; }
double Point :: getY { return y; }
void Point :: display ( ) {
    cout << "point x,y : " << x << "," << y << endl; }
```

Class Operator Overloading: Vector.h

Declare class with private array variable as property

```
class Vector {
private:
    double num[3];
```

Set initialization values for class property

```
public:
    Vector ( ) : num{0.0, 0.0, 0.0 } {}
    Vector (double no1, double no2, double no3) {
        num[0] = no1; num[1] = no2; num[2] = no3; }
    void display ( ) {
        cout << "[" << num[0] << ", " << num[1] << ", " << num[2] << "]"
<< endl; }
```

Create prototypes for external class methods

```
    Vector operator++();
    Vector operator++(int);
    Vector operator--();
    Vector operator+(Vector);
    Vector operator-(Vector&);
    friend Vector operator *(Vector&, Vector&);
    double& operator[](int);
```

Class Operator Overloading: Vector.cpp

```
#include "Vector.h"

Vector Vector :: operator ++(){
   return Vector((num[0] + 1), (num[1] + 1), (num[2] + 1)); }
Vector Vector :: operator++(int) {

   num[0]++;
   num[1]++;
   num[2]++;
   return Vector(num[0], num[1], num[2]); }

Vector Vector :: operator--(){
   return Vector((num[0]-1), (num[1]-1), (num[2]-1)); }

Vector Vector :: operator + (Vector v) {
   return Vector((num[0] + v.num[0]), (num[1] + v.num[1]), (num[2] +
v.num[2])); }

Vector Vector :: operator - (Vector& v) {
   return Vector((num[0] - v.num[0]), (num[1] - v.num[1]), (num[2] -
v.num[2]));
}

Vector operator*(Vector& v1, Vector& v2) {
   return Vector((v1.num[1] * v2.num[2]) - (v1.num[2] * v2.num[1]),
(v1.num[2] * v2.num[0]) - (v1.num[0] * v2.num[2]), ((v1.num[0] *
v2.num[1]) - (v1.num[1] * v2.num[0]))); }

double& Vector::operator[](int index) //friend operator {
   if (index >= 3) {
      cout << "Array index is out of bounds, exiting the program" <<
endl;
      exit(1);
   }
   return num[index]; }
```

Class Operator Overloading: Main.cpp

Create Object and use class methods

Class Operator Overloading: Main.cpp (cont)

```
int main(void) {
   Vector a, b(3.0, 4.0, 5.0);

   a[0] = 11.0;
   a[1] = -15.0;
   a[2] = 3.0;

   Vector c(2.0,4.0,7.0);

   (++c).display();
   (c++).display();
   (--c).display();

   Vector d = a + b;
   d.display();
   d = a - b;
   d.display();
   d = a * b;
   d.display();
   d[5] = 5.0;

   return 0;
}
```

Inheritance