CompSci 373 Tutorial

C++ Programming

C++ Language

- Is an extension of C -- so that pretty much any valid C program will compile and run on a C++ compiler.
- Unlike C, C++ is an OOP language
- Every program must have a **main** method, which serves as the entry point to the program

```
int main(int argc, char* argv[])
{

    return 0;
}
```

- There is no in-built memory management (clean up your own mess!)

C++ Data Type Review

- ***bool:*** Boolean in Java, indicating true or false
- ***int, unsigned int, short***: for integers .e.g. 8, 19, 2012
- ***float, double***: for floating point numbers .e.g. 7.2658
- ***char***: for a single letter .e.g. 'a' or 'm', .etc.
- ***int myArray[10]***: an array having 10 elements of type of integer
  - myArray[0] = 19;
- *: a pointer, an array or a string (a string is considered as an array of letters)
  - int* myArray = new int[10];
    - myArray[0] = 19;
  - char* strName = "Luffy";

C++ Program Structure

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

int main(int argc, char* argv[])
{
    cout << "Hello World" << endl;

    //Do the computation here ...
    /*
        Call to other methods
        Create object's intances
        Make calls
    */

    getchar();
    return 0;
}
```

External library to be used
- **iostream** provides input/output

C++ Program Structure

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

int main(int argc, char* argv[])
{
    cout << "Hello World" << endl;

    //Do the computation here ...
    /*
        Call to other methods
        Create object's intances
        Make calls
    */

    getchar();
    return 0;
}
```

C++ Program Structure

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

int main(int argc, char* argv[])
{
    cout << "Hello World" << endl;

    //Do the computation here ...
    /*
        Call to other methods
        Create object's intances
        Make calls
    */

    getchar();
    return 0;
}
```

Use output stream (display), instead of input stream

Display a message on the screen

Add newline character "\n" at the end of the message

C++ Program Structure

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

int main(int argc, char* argv[])
{
    cout << "Hello World" << endl;

    //Do the computation here ...
    /*
        Call to other methods
        Create object's intances
        Make calls
    */

    getchar();
    return 0;
}
```

Stop here, and wait for 1 char to be entered from the keyboard.

Effectively pausing the screen until the user presses any key

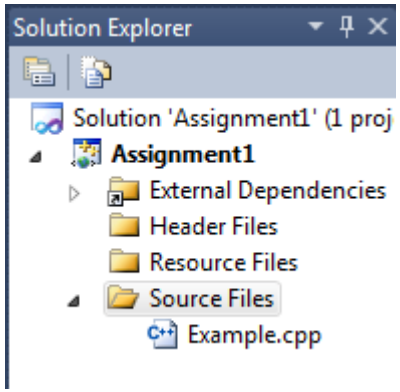
C++ Program Structure

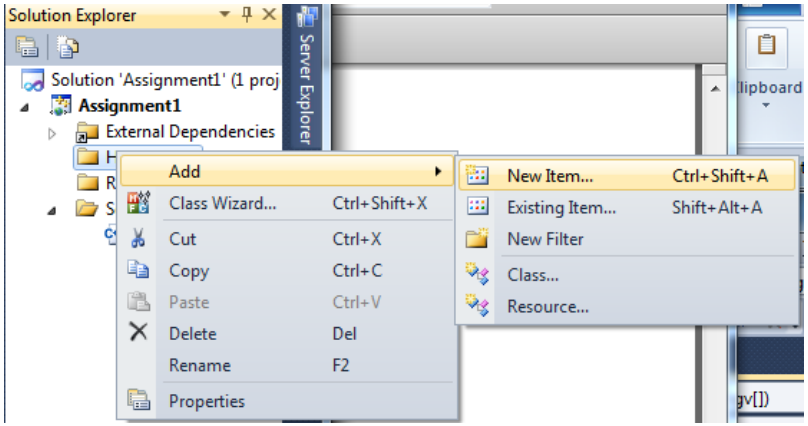
```
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
    cout << "Hello World" << endl;

    //Do the computation here ...
    /*
        Call to other methods
        Create object's intances
        Make calls
    */

    getchar();
    return 0;
}
```

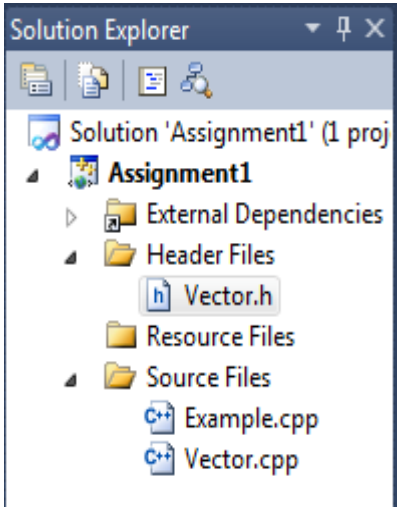
Finish the program and exit

Classes in C++

- Each class is often defined using 2 separate files.
  - Header file (.h) contains the definition of the class. Things like methods' signature, instance variables etc.
  - Class file (.cpp) contains the actual implementation of those methods







Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

class name

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Whatever declared in this block will be marked as **public**

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Constructor of this class

<Constructors have NO return type>

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Class methods

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Overloading operators

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Whatever declared in this block will be marked as **private**

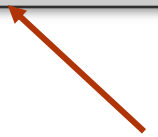
Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```



private instance variables

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Tell the compiler that we will use what have been declared in this file

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Implement the constructor

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

The following signature belongs to class Vector

Constructor's signature

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

this refers to the current class we are in

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Provide the implementation for **length** method

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Provide the implementation for **length** and **normalise** method

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Class file: vector.cpp

```
1  #include "Vector.h"
2
3  //provide the implementation
4  //for the Constructor
5  Vector::Vector(int x, int y)
6  {
7      this->x = x;
8      this->y = y;
9  }
10
11 //provide the implementation
12 //for the two methods
13 float Vector::length()
14 {
15     return 0;
16 }
17
18 Vector Vector::normalise()
19 {
20     return Vector(0, 0);
21 }
22
23 //provide the implementation
24 //for the two operators
25 Vector Vector::operator*(const double a) const
26 {
27     return Vector(0, 0);
28 }
29
30 Vector& Vector::operator+ (const Vector &vect)
31 {
32     return Vector(0, 0);
33 }
```

Provide the implementation for **length** and **normalise** method

Classes in C++

Header file: vector.h

```
1  class Vector
2  {
3  public:
4      Vector(int x, int y);
5
6      float length();
7      Vector normalise();
8
9      Vector operator*(const double a) const;
10     Vector& operator+ (const Vector &vect);
11
12 private:
13     int x, y;
14 };
```

Class file: vector.cpp

```
5  //provide the implementation
6  //for the Constructor
7  Vector::Vector(int x, int y)
8  {
9      this->x = x;
10     this->y = y;
11 }
12
13 //provide the implementation
14 //for the two methods
15 float Vector::length()
16 {
17     //length of a vector sqrt(x * x + y * y)
18
19     return sqrt((float) (x * x + y * y));
20 }
21
22 Vector Vector::normalise()
23 {
24     float len = length();
25
26     return Vector(x / len, y / len);
27 }
28
29 //provide the implementation
30 //for the two operators
31 Vector Vector::operator*(const double a) const
32 {
33     return Vector(x * a, y * a);
34 }
35
36 Vector& Vector::operator+(const Vector &vect)
37 {
38     Vector newVector(x + vect.x, y + vect.y);
39     return newVector;
40 }
41
```

Classes in C++

```
#include <iostream>

#include "Vector.h"

using namespace std;

int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    cout << "Length of Vector 1 is " << vector1.length() << endl;
    cout << "Length of Vector 2 is " << vector2.length() << endl;

    //normalise vector1
    vector1 = vector1.normalise();
    cout << "Length of Vector 1 is " << vector1.length() << endl;
    cout << "Length of Vector 2 is " << vector2.length() << endl;

    getchar();
    return 0;
}
```

Classes in C++

```
#include <iostream>

#include "Vector.h"

using namespace std;

int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    cout << "Length of Vector 1 is " << vector1.length() << endl;
    cout << "Length of Vector 2 is " << vector2.length() << endl;

    //normalise vector1
    vector1 = vector1.normalise();
    cout << "Length of Vector 1 is " << vector1.length() << endl;
    cout << "Length of Vector 2 is " << vector2.length() << endl;

    getchar();
    return 0;
}
```

Indicate that we are going to use Vector class

Classes in C++

```
#include <iostream>

#include "Vector.h"

using namespace std;

int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    cout << "Length of Vector 1 is " << vector1.length() << endl;
    cout << "Length of Vector 2 is " << vector2.length() << endl;

    //normalise vector1
    vector1 = vector1.normalise();
    cout << "Length of Vector 1 is " << vector1.length() << endl;
    cout << "Length of Vector 2 is " << vector2.length() << endl;

    getchar();
    return 0;
}
```

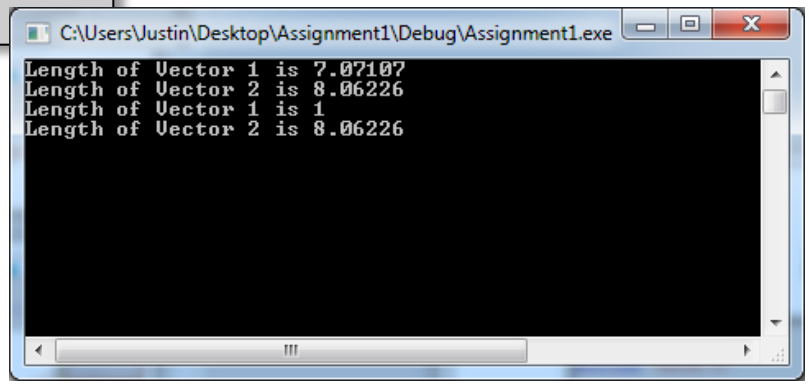
Create 2 vectors. The defined constructor is called

Classes in C++

```
#include <iostream>

#include "Vector.h"

using namespace std;

int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    cout << "Length of Vector 1 is " << vector1.length() << endl;
    cout << "Length of Vector 2 is " << vector2.length() << endl;

    //normalise vector1
    vector1 = vector1.normalise();
    cout << "Length of Vector 1 is " << vector1.length() << endl;
    cout << "Length of Vector 2 is " << vector2.length() << endl;

    getchar();
    return 0;
}
```

Call method length of the vector

Call method normalise of the vector



Operator Overloading

- **Goal**: provide a mean to add 2 vectors
  - Solution: implement a method that take 2 vectors and perform addition

```
class Vector
{
public:
    Vector(float x, float y);

    float length();
    Vector normalise();

    Vector addTwoVectors(const Vector &vect);

    /*Vector operator*(const double a) const;
    Vector& operator+(const Vector &vect);*/


    float x, y;
};
```

Declare and implement a method that performs the task

```
Vector Vector::addTwoVectors(const Vector &vect)
{
    return Vector(x + vect.x, y + vect.y);
}
```

Operator Overloading

```
int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    Vector vector3 = vector1.addTwoVectors(vector2);

    getchar();
    return 0;
}
```

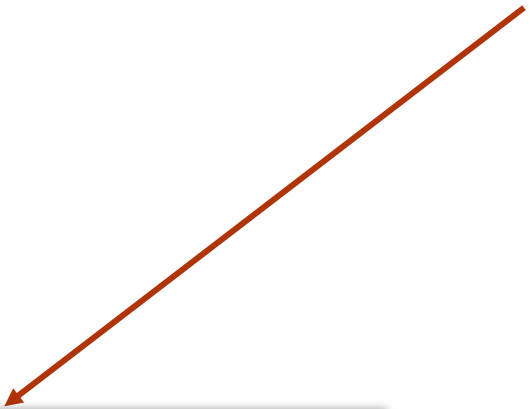
Work fine!
But isn't it more intuitive if we could have it this way?

```
int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    Vector vector3 = vector1 + vector2;

    getchar();
    return 0;
}
```

Operator Overloading

```
int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    Vector vector3 = vector1 + vector2;

    getchar();
    return 0;
}
```

C++ allows us to define the behaviour for those operators such as +, -, ++, ==, !=, etc.



```
Vector& Vector::operator+(const Vector &vect)
{
    Vector newVector(this->x + vect.x, this->y + vect.y);
    return newVector;
}
```

Operator Overloading

```
int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    Vector vector3 = vector1 + vector2;

    getchar();
    return 0;
}
```

C++ allows us to define the behaviour for those operators such as +, -, ++, ==, !=, etc.

```
Vector& Vector::operator+(const Vector &vect)
{
    Vector newVector(this->x + vect.x, this->y + vect.y);
    return newVector;
}
```

Operator signature

Operator Overloading

```
int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    Vector vector3 = vector1 + vector2;

    getchar();
    return 0;
}
```

C++ allows us to define the behaviour for those operators such as +, -, ++, ==, !=, etc.

```
Vector& Vector::operator+(const Vector &vect)
{
    Vector newVector(this->x + vect.x, this->y + vect.y);
    return newVector;
}
```

Define the behaviour for “+” symbol

Operator Overloading

```
int main(int argc, char* argv[])
{
    //create 2 vectors
    Vector vector1(5, 5);
    Vector vector2(7, 4);

    Vector vector3 = vector1 + vector2;

    getchar();
    return 0;
}
```

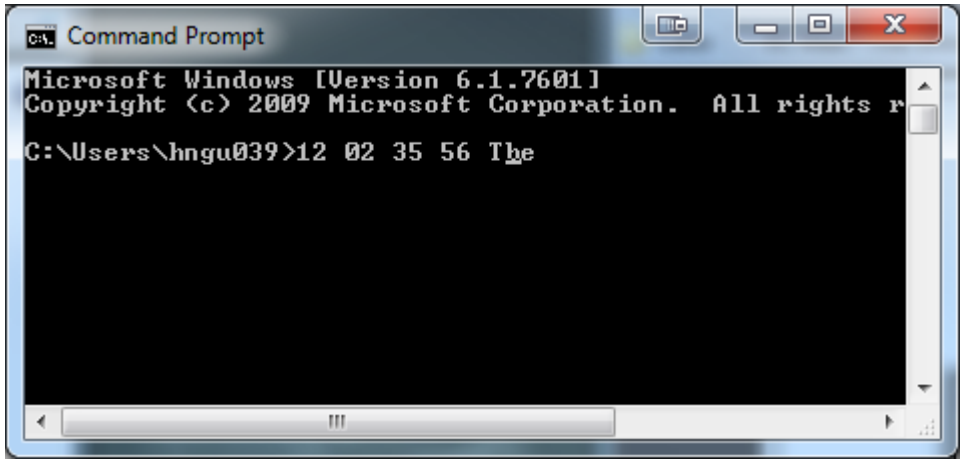
C++ allows us to define the behaviour for those operators such as +, -, ++, ==, !=, etc.

```
Vector& Vector::operator+(const Vector &vect)
{
    Vector newVector(this->x + vect.x, this->y + vect.y);
    return newVector;
}
```

Read inputs from Users

```
int main(int argc, char* argv[])
{
    int x, y;

    //create 2 vectors
    cout << "Enter the coordinates for vector 1: ";
    cin >> x;
    cin >> y;
    Vector vector1(x, y);

    cout << "Enter the coordinates for vector 2: ";
    cin >> x;
    cin >> y;
    Vector vector2(x, y);

    cout << "The Coordinate of vector 1 is " << vector1.getCoordinates() << endl;
    cout << "The Coordinate of vector 2 is " << vector2.getCoordinates() << endl;
```

Read the first input (separated by whitespace) and store in x



Buffer

12
02
35
56
The

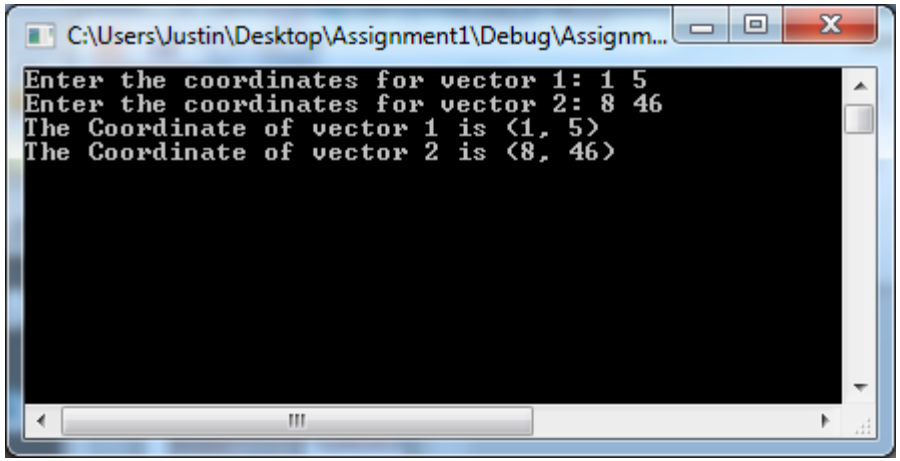
Read inputs from Users

```
int main(int argc, char* argv[])
{
    int x, y;

    //create 2 vectors
    cout << "Enter the coordinates for vector 1: ";
    cin >> x;
    cin >> y;
    Vector vector1(x, y);

    cout << "Enter the coordinates for vector 2: ";
    cin >> x;
    cin >> y;
    Vector vector2(x, y);

    cout << "The Coordinate of vector 1 is " << vector1.getCoordinates() << endl;
    cout << "The Coordinate of vector 2 is " << vector2.getCoordinates() << endl;
```

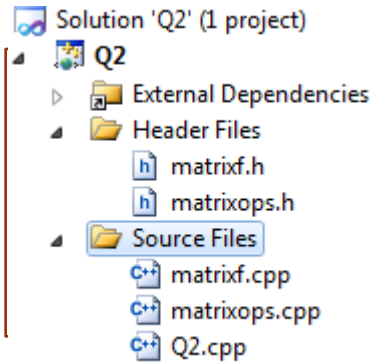
Read the second input (separated by whitespace) and store in y

Read inputs from Users

```
int main(int argc, char* argv[])
{
    int x, y;

    //create 2 vectors
    cout << "Enter the coordinates for vector 1: ";
    cin >> x;
    cin >> y;
    Vector vector1(x, y);

    cout << "Enter the coordinates for vector 2: ";
    cin >> x;
    cin >> y;
    Vector vector2(x, y);

    cout << "The Coordinate of vector 1 is " << vector1.getCoordinates() << endl;
    cout << "The Coordinate of vector 2 is " << vector2.getCoordinates() << endl;
```



Assignment Skeleton Code Explained



Project for question 2, modify codes in here

- **matrixf:** contains the definition of a matrix or a vector
- **matrixops:** contains a number of operations we can perform on a matrix or a vector .e.g: transpose, multiple, dot, cross product, etc.

Assignment Skeleton Code Explained

```
class Matrixf
{

public:
    // Constructors and Destructor
    // Creates a matrix of size rows and cols with all values set to zero
    Matrixf(unsigned int rows, unsigned int cols) ;
    virtual ~Matrixf() ;

    // copy and assign
    // performs deep copy of data
    Matrixf(Matrixf const& other) ;
    // replaces existing data with data from another matrix (can be of different size)
    virtual Matrixf& operator = (Matrixf const& other) ;

    // accessors and mutators
    // row index and column index are 0-based
    virtual float get(unsigned int row, unsigned int col) const ;
    virtual void set(unsigned int row, unsigned int col, float val) ;

    virtual float& operator()(unsigned int row, unsigned int col) ;
    virtual float operator()(unsigned int row, unsigned int col) const ;

    // gets the size of the matrix
    virtual unsigned int nrows() const ;
    virtual unsigned int ncols() const ;

    virtual bool isVector() const ;

protected: // Member variables
    unsigned int rows_;
    unsigned int cols_;
    float* data_;

public: // helper static methods
    static Matrixf eye(unsigned int size) ;

public: // iostream functions
    friend std::ostream& operator << (std::ostream& os, Matrixf const& matrix) ;

}; // class Matrixf
```

Describe a matrix or a vector

Keeping the number of rows and columns of the matrix

Assignment Skeleton Code Explained

```
class Matrixf
{

public:
    // Constructors and Destructor
    // Creates a matrix of size rows and cols with all values set to zero
    Matrixf(unsigned int rows, unsigned int cols) ;
    virtual ~Matrixf() ;

    // copy and assign
    // performs deep copy of data
    Matrixf(Matrixf const& other) ;
    // replaces existing data with data from another matrix (can be of different size)
    virtual Matrixf& operator = (Matrixf const& other) ;

    // accessors and mutators
    // row index and column index are 0-based
    virtual float get(unsigned int row, unsigned int col) const ;
    virtual void set(unsigned int row, unsigned int col, float val) ;

    virtual float& operator()(unsigned int row, unsigned int col) ;
    virtual float operator()(unsigned int row, unsigned int col) const ;

    // gets the size of the matrix
    virtual unsigned int nrows() const ;
    virtual unsigned int ncols() const ;

    virtual bool isVector() const ;

protected: // Member variables
    unsigned int rows_;
    unsigned int cols_;
    float* data_;

public: // helper static methods
    static Matrixf eye(unsigned int size) ;

public: // iostream functions
    friend std::ostream& operator << (std::ostream& os, Matrixf const& matrix) ;

}; // class Matrixf
```

Describe a matrix or a vector

Keeping the number of rows and columns of the matrix

```
8  int main(int argc, char* argv[])
9  {
10     //Create a vector with 3 rows and 1 col
11     Matrixf vect(3, 1);
12     vect(0, 0) = -1;   //assign values to row 1
13     vect(1, 0) = 3;    //assign values to row 2
14     vect(2, 0) = 1;    //assign values to row 3
15
16     //display the values of the vector
17     std::cout << vect << std::endl;
18
19     //Creat a matrix
20     Matrixf matrix(4, 4);
21     //assign values to row 1
22     matrix(0, 0) = 1;
23     matrix(0, 1) = 0;
24     matrix(0, 2) = 0;
25     matrix(0, 3) = 0;
26
27     //assign values to row 2
28     matrix(1, 0) = 0;
29     matrix(1, 1) = 1;
30     matrix(1, 2) = 0;
31     matrix(1, 3) = 0;
32
33     //assign values to row 3
34     matrix(2, 0) = 0;
35     matrix(2, 1) = 0;
36     matrix(2, 2) = 1;
37     matrix(2, 3) = 0;
38
39     //assign values to row 4
40     matrix(3, 0) = 0;
41     matrix(3, 1) = 0;
42     matrix(3, 2) = 0;
43     matrix(3, 3) = 1;
44
45     //display the values of the matrix
46     std::cout << matrix << std::endl;
47
48     getchar();
49     return 0;
50 }
```

Create a vector with 3 rows and 1 column

```
8  int main(int argc, char* argv[])
9  {
10     //Create a vector with 3 rows and 1 col
11     Matrixf vect(3, 1);
12     vect(0, 0) = -1;    //assign values to row 1
13     vect(1, 0) = 3;     //assign values to row 2
14     vect(2, 0) = 1;     //assign values to row 3
15
16     //display the values of the vector
17     std::cout << vect << std::endl;
18
19     //Creat a matrix
20     Matrixf matrix(4, 4);
21     //assign values to row 1
22     matrix(0, 0) = 1;
23     matrix(0, 1) = 0;
24     matrix(0, 2) = 0;
25     matrix(0, 3) = 0;
26
27     //assign values to row 2
28     matrix(1, 0) = 0;
29     matrix(1, 1) = 1;
30     matrix(1, 2) = 0;
31     matrix(1, 3) = 0;
32
33     //assign values to row 3
34     matrix(2, 0) = 0;
35     matrix(2, 1) = 0;
36     matrix(2, 2) = 1;
37     matrix(2, 3) = 0;
38
39     //assign values to row 4
40     matrix(3, 0) = 0;
41     matrix(3, 1) = 0;
42     matrix(3, 2) = 0;
43     matrix(3, 3) = 1;
44
45     //display the values of the matrix
46     std::cout << matrix << std::endl;
47
48     getchar();
49     return 0;
50 }
```

Fill in values

```
8  int main(int argc, char* argv[])
9  {
10     //Create a vector with 3 rows and 1 col
11     Matrixf vect(3, 1);
12     vect(0, 0) = -1;   //assign values to row 1
13     vect(1, 0) = 3;    //assign values to row 2
14     vect(2, 0) = 1;    //assign values to row 3
15
16     //display the values of the vector
17     std::cout << vect << std::endl;
18
19     //Creat a matrix
20     Matrixf matrix(4, 4);
21     //assign values to row 1
22     matrix(0, 0) = 1;
23     matrix(0, 1) = 0;
24     matrix(0, 2) = 0;
25     matrix(0, 3) = 0;
26
27     //assign values to row 2
28     matrix(1, 0) = 0;
29     matrix(1, 1) = 1;
30     matrix(1, 2) = 0;
31     matrix(1, 3) = 0;
32
33     //assign values to row 3
34     matrix(2, 0) = 0;
35     matrix(2, 1) = 0;
36     matrix(2, 2) = 1;
37     matrix(2, 3) = 0;
38
39     //assign values to row 4
40     matrix(3, 0) = 0;
41     matrix(3, 1) = 0;
42     matrix(3, 2) = 0;
43     matrix(3, 3) = 1;
44
45     //display the values of the matrix
46     std::cout << matrix << std::endl;
47
48     getchar();
49     return 0;
50 }
```

Create a 4x4 matrix

Header file: matrixops.h

```
Matrixf multiply(Matrixf const& left, Matrixf const& right) ;

// Scales all elements of a matrix by a scalar.
Matrixf multiply(Matrixf const& mat, float scalar) ;
Matrixf multiply(float scalar, Matrixf const& mat) ;

// Finds the dot product of two vectors of the same length.
float dot(Matrixf const& vec1, Matrixf const& vec2) ;

// Computes the cross product of two 3x3 vectors.
Matrixf cross(Matrixf const& vec1, Matrixf const& vec2) ;

// Computes the sum of two matrices of the same size.
Matrixf add(Matrixf const& mat1, Matrixf const& mat2) ;

// Compute the transpose of a matrix.
Matrixf transpose(Matrixf const& mat) ;

// Computes the length (L2 norm) of a vector.
float length(Matrixf const& v) ;
```

Class file: matrixops.cpp

```
20  Matrixf multiply(Matrixf const& mat, float scalar){ ... }
28
29  Matrixf multiply(float scalar, Matrixf const& mat) {
30      return multiply(mat, scalar);
31  }
32
33  float dot(Matrixf const& vec1, Matrixf const& vec2) {
34
35      // error check
36      if (!vec1.isVector() || !vec2.isVector()) {
37          throw std::runtime_error("Unable to do dot product: not column vectors.");
38      }
39      if (vec1.nrows() != vec2.nrows()) {
40          throw std::runtime_error("Unable to do dot product: vector lengths not equal.");
41      }
42
43      /** implement dot product **********************************/
44      float ret = -1;
45
46      return ret;
47  }
48
49  Matrixf cross(Matrixf const& vec1, Matrixf const& vec2) {
50
51      // error check
52      if (!vec1.isVector() || !vec2.isVector()) {
53          throw std::runtime_error("Unable to do dot product: not column vectors.");
54      }
55      if (vec1.nrows() != 3 || vec2.nrows() != 3) {
56          throw std::runtime_error("Unable to do cross product: vector lengths not 3.");
57      }
58
59      /** implement cross product ********************************/
60      Matrixf ret(1, 1);
61
62      return ret;
63  }
```

Header file: matrixops.h

```
Matrixf multiply(Matrixf const& left, Matrixf const& right) ;

// Scales all elements of a matrix by a scalar.
Matrixf multiply(Matrixf const& mat, float scalar) ;
Matrixf multiply(float scalar, Matrixf const& mat) ;

// Finds the dot product of two vectors of the same length.
float dot(Matrixf const& vec1, Matrixf const& vec2) ;

// Computes the cross product of two 3x3 vectors.
Matrixf cross(Matrixf const& vec1, Matrixf const& vec2) ;

// Computes the sum of two matrices of the same size.
Matrixf add(Matrixf const& mat1, Matrixf const& mat2) ;

// Compute the transpose of a matrix.
Matrixf transpose(Matrixf const& mat) ;

// Computes the length (L2 norm) of a vector.
float length(Matrixf const& v) ;
```

```
int main(int argc, char* argv[])
{
    Matrixf vect1(3, 1);

    vect1(0, 0) = -1;
    vect1(1, 0) = 3;
    vect1(2, 0) = 1;

    std::cout << multiply(2, vect1) << std::endl;

    getchar();
    return 0;
}
```

Class file: matrixops.cpp

```
20  Matrixf multiply(Matrixf const& mat, float scalar) { ... }
28
29  Matrixf multiply(float scalar, Matrixf const& mat) {
30      return multiply(mat, scalar);
31  }
32
33  float dot(Matrixf const& vec1, Matrixf const& vec2) {
34
35      // error check
36      if (!vec1.isVector() || !vec2.isVector()) {
37          throw std::runtime_error("Unable to do dot product: not column vectors.");
38      }
39      if (vec1.nrows() != vec2.nrows()) {
40          throw std::runtime_error("Unable to do dot product: vector lengths not equal.");
41      }
42
43      /** implement dot product **********************************/
44      float ret = -1;
45
46      return ret;
47  }
48
49  Matrixf cross(Matrixf const& vec1, Matrixf const& vec2) {
50
51      // error check
52      if (!vec1.isVector() || !vec2.isVector()) {
53          throw std::runtime_error("Unable to do dot product: not column vectors.");
54      }
55      if (vec1.nrows() != 3 || vec2.nrows() != 3) {
56          throw std::runtime_error("Unable to do cross product: vector lengths not 3.");
57      }
58
59      /** implement cross product ********************************/
60      Matrixf ret(1, 1);
61
62      return ret;
63  }
```

Header file: matrixops.h

```
Matrixf multiply(Matrixf const& left, Matrixf const& right) ;

// Scales all elements of a matrix by a scalar.
Matrixf multiply(Matrixf const& mat, float scalar) ;
Matrixf multiply(float scalar, Matrixf const& mat) ;

// Finds the dot product of two vectors of the same length.
float dot(Matrixf const& vec1, Matrixf const& vec2) ;

// Computes the cross product of two 3x3 vectors.
Matrixf cross(Matrixf const& vec1, Matrixf const& vec2) ;

// Computes the sum of two matrices of the same size.
Matrixf add(Matrixf const& mat1, Matrixf const& mat2) ;

// Compute the transpose of a matrix.
Matrixf transpose(Matrixf const& mat) ;

// Computes the length (L2 norm) of a vector.
float length(Matrixf const& v) ;
```

Class file: matrixops.cpp

```
20  Matrixf multiply(Matrixf const& mat, float scalar) { ... }
28
29  Matrixf multiply(float scalar, Matrixf const& mat) {
30      return multiply(mat, scalar);
31  }
32
33  float dot(Matrixf const& vec1, Matrixf const& vec2) {
34
35      // error check
36      if (!vec1.isVector() || !vec2.isVector()) {
37          throw std::runtime_error("Unable to do dot product: not column vectors.");
38      }
39      if (vec1.nrows() != vec2.nrows()) {
40          throw std::runtime_error("Unable to do dot product: vector lengths not equal.");
41      }
42
43      /** implement dot product **********************************/
44      float ret = -1;
45
46      return ret;
47  }
48
49  Matrixf cross(Matrixf const& vec1, Matrixf const& vec2) {
50
51      // error check
52      if (!vec1.isVector() || !vec2.isVector()) {
53          throw std::runtime_error("Unable to do dot product: not column vectors.");
54      }
55      if (vec1.nrows() != 3 || vec2.nrows() != 3) {
56          throw std::runtime_error("Unable to do cross product: vector lengths not 3.");
57      }
58
59      /** implement cross product ********************************/
60      Matrixf ret(1, 1);
61
62      return ret;
63  }
```

NOT YET IMPLEMENTED

YOUR TASK