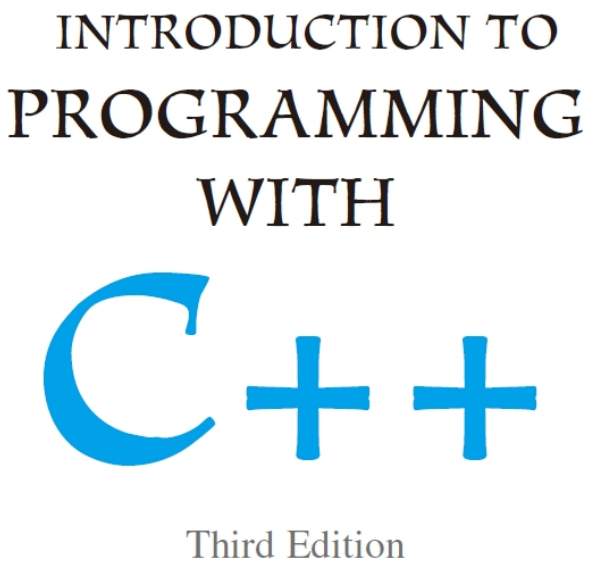
Introduction to Programming with C++

Chieh-Sen (Jason) Huang

Department of Applied Mathematics

National Sun Yat-sen University



Contents are based on book by Y. Daniel Liang

Operator Functions

- Let us revisit the Vector that we defined. We could define functions for adding two vectors as following

```
void Vector::Add(double x1, double y1){
    x += x1; \\ List14_1a.cpp
    y += y1;}
void Vector::Add(Vector AnotherV){
    x += AnotherV.x;
    y += AnotherV.y;}
Vector Vector::RetAdd(Vector AnotherV){
  return Vector(x + AnotherV.x, y + AnotherV.y);}

int main(){
    Vector Vec1(1,2), Vec2(3,4);
    Vec1.Add(Vec2);
    Vec1.Show();
    Vector Vec3 = Vec1.RetAdd(Vec2);
    Vec3.Show();
-------------------------------------------------
[4,6]
[7,10]
```

- Note that the first two add functions change the values of the the Vector object that calling Add functions.

Operator Functions

- The operators are actually **functions** defined in a class. These functions are named with keyword **operator** followed by the actual operator.

- We could defining functions for operators if they are not defined and which are called **operator overloading**.

```
Vector Vector::operator+(const Vector& v2)
{
   return (Vector(x + v2.x, y + v2.y));
}
int main(){
    Vector Vec3 = Vec1.RetAdd(Vec2);
    Vec3.Show();
    Vector Vec5 = Vec1 + Vec2;
    Vec5.Show();
-----------------------------------------
[4,6]
[4,6]
```

Friend Functions and Classes

- Private members of a class cannot be accessed from outside the class.
- Occasionally, it is convenient to allow some trusted functions and classes to access a class's private members.
- C++ enables you to use the **friend keyword** to define **friend functions** and **friend classes** so that these trusted functions and classes can access another class's private members.
- Friend function **operator+** is not a member of the Vector class but can access the private data in Vector.

Friend Functions and Classes

```
class Vector
{
 ....... \\ List14_3a.cpp
 friend Vector operator+(const Vector& v1, const Vector& v2);
}
 Vector operator+(const Vector& v1, const Vector& v2)
{
   return (Vector(v1.x + v2.x, v1.y + v2.y));
}
int main()
{
   Vector Vec1(1,2), Vec2(3,4);
   (Vec1 + Vec2).Show();
   Vector Vec5 = Vec1 + Vec2;
   Vec5.Show();
}
-----------------------------------------
[4,6]
[4,6]
```

friend Functions

- Friend function `xx(const Vector& v1), yy(const Vector& v1)` are not members of the Vector class but can access the private data in Vector.

- Using **stream insertion operator (<<)**.

```
class Vector
{
 ..... \\ List14_4a.cpp
  friend double xx(const Vector& v1);
  friend double yy(const Vector& v1);
}
inline ostream& operator<<(ostream& s, Vector& v)
{
  return (s << '(' << xx(v) << ',' << yy(v) << ')');
}
int main()
{
   Vector Vec1(1,2), Vec2(3,4);
   Vector Vec5 = Vec1 + Vec2;
   cout << Vec5 << endl;
-----------------------------------------------
[4,6]
(4,6)
```

friend Functions and Classes

- In Chapter 10 (List10_3a.cpp) we define the Quad class from Vector class. And Quad can not access private members of Vector.

- A friend class could be defined to circumvent this.

```
class Vector
{                  \\ List14_5a.cpp
 private:
  double x;
  double y;
 public:
  void Set(double x1,double y1);
  void Show();
  void Add(double x1,double y1);
  void Add(Vector AnotherV);
  Vector RetAdd(Vector AnotherV);
  void Multiple(double Constant);
  Vector(double i = 0, double j = 0);
  friend class Quad; <-- Friend class
                     <-- is declared.
};
```

```
class Quad
{
 private:
  Vector Vec1;
  Vector Vec2;
 public:
  void Add(Quad OtherQuad)
  {
    Vec1 = Vec1.RetAdd(OtherQuad.Vec1);
    Vec2 = Vec2.RetAdd(OtherQuad.Vec2);
  }
  void Add(Quad OtherQuad)
  {
    Vec1.x += OtherQuad.Vec1.x; <-- We could access the
    Vec1.y += OtherQuad.Vec1.y; <-- private members of
    Vec2.x += OtherQuad.Vec2.x; <-- Vector
    Vec2.y += OtherQuad.Vec2.y;
  }
.....
}
```

- A complete code is at List14_6a.cpp