Int main()  
{  
 Circle c(-2, 5, 10);  
 c.scale(2);  
 c.draw();  
 cout << area(c);  
}  
  
If you don’t start off with using “public” or “private”, using Class defaults to private and using Struct defaults to public. It’s easiest to just put public/private regardless

class Circle  
{  
 public:  
 Circle(double x, double y, double r);  
 bool scale(double factor);  
 void draw() const;  
 double radius const;  
 private:  
 double m\_x;   
 double m\_y;  
 double m\_r; //radius of the circle   
  
};  
  
Circle::Circle(double x, double y, double r)  
 : m\_x(x), m\_y(y), m\_r(r)  
{  
 if (r <= 0)  
 /\* error\*/  
}  
bool Circle::scale(double factor)  
{  
 if (factor <= 0)  
 return false;  
 m\_r \*= factor;  
 return true;  
}  
void Circle::draw() const  
{  
 …/\*draw the circle (depends on graphics package)\*/  
}  
double Circle::radius() const  
{  
 return m\_r;  
}  
double area(const Circle& a)  
{  
 const double PI = 4 \* std::atan(1.0);  
 return (PI) \* a.radius() \* a.radius();  
}  
  
Pass by *value*: copy the variables and operate on them, then return and discard the copy. Ex: area(Circle a)  
  
Pass by *reference*: operate on the original variables from their original addresses, then return and quit the function. Original values are changed permanently. Ex: area(Circle& a)  
  
If you pass by reference you can put *const* in front of it to promise not to modify the referenced variable. Ex: area(const Circle& a) You can do this to save system resources but have basically the same usage as a pass by value. If you do this to call an element of a class, you can declare a public member function of the class to be constant like this: double radius() *const;*.

In a constructor you can utilize what’s called a *member initialization list.* You follow the name of the constructor with a colon and how you want elements to be initialized. Ex:  
Circle::Circle(double x, double y, double r) : m\_x(x), m\_y(y), m\_r(r) { … }

The syntax is : element to be initialized(element to copy value from), element2(copy2), …  
  
A class *invariant* is the valid state for data members of a class. For the circle, x and y can be all real integers, and the radius must be greater than 0.