SSCM 1313
C++ COMPUTER PROGRAMMING

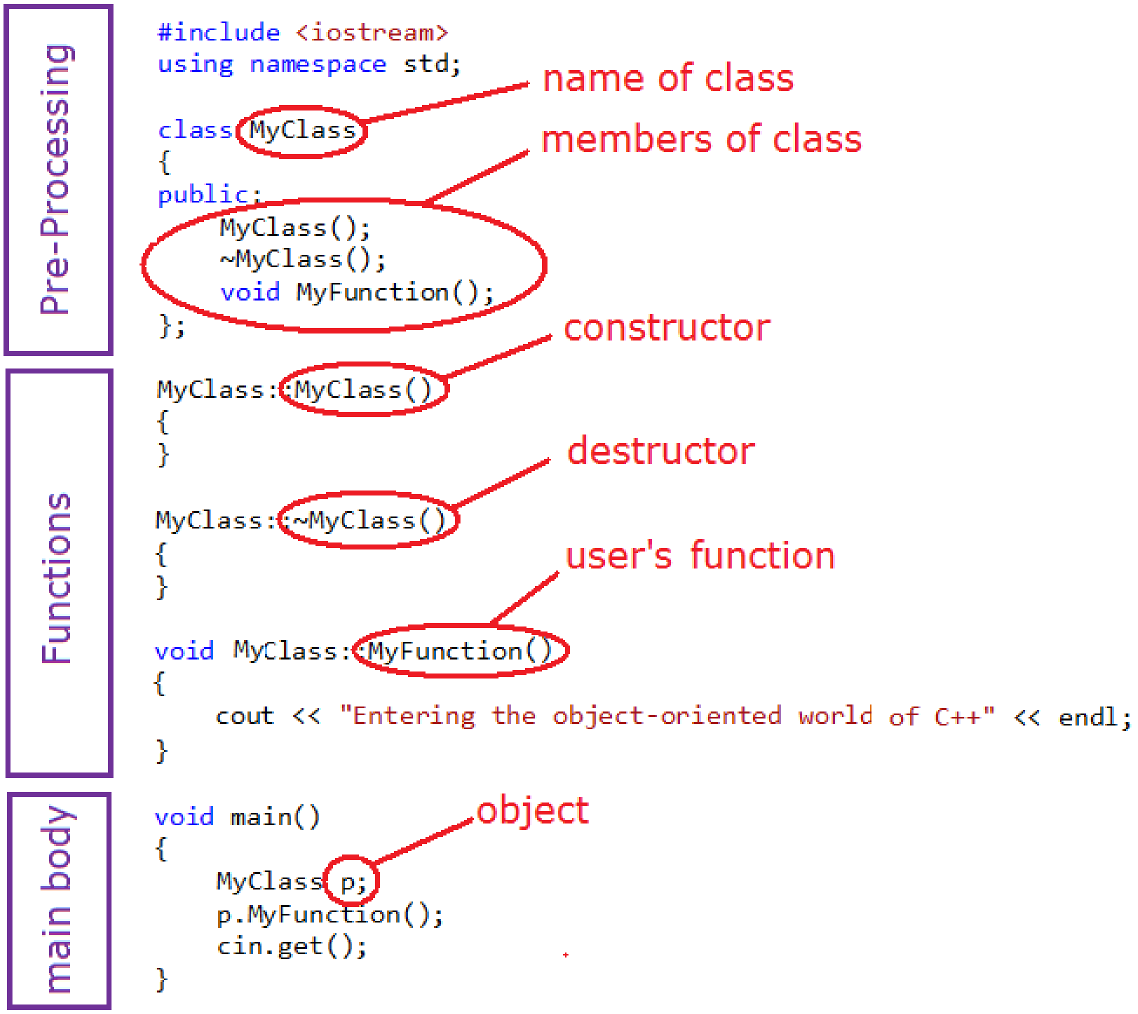
Chapter 6:
Class, Object and Function

Authors:
Farhana Johar
Professor Dr. Shaharuddin Salleh

Class, Object and Function

Object-oriented approach:
Function. Classes and their objects, Constructor/ destructor and function overloading. Inheritance, class collaboration and polymorphism. Dynamic memory allocation. Applications in graph theoretical and matrix problems.

C++ Language Organization



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

class MyClass
{
public:
    MyClass();
    ~MyClass();
    void MyFunction();
};

MyClass::MyClass()
{
}

MyClass::~MyClass()
{
}

void MyClass::MyFunction()
{
    cout << "Entering the object-oriented world of C++" << endl;
}

void main()
{
    MyClass p;
    p.MyFunction();
    cin.get();
}
```

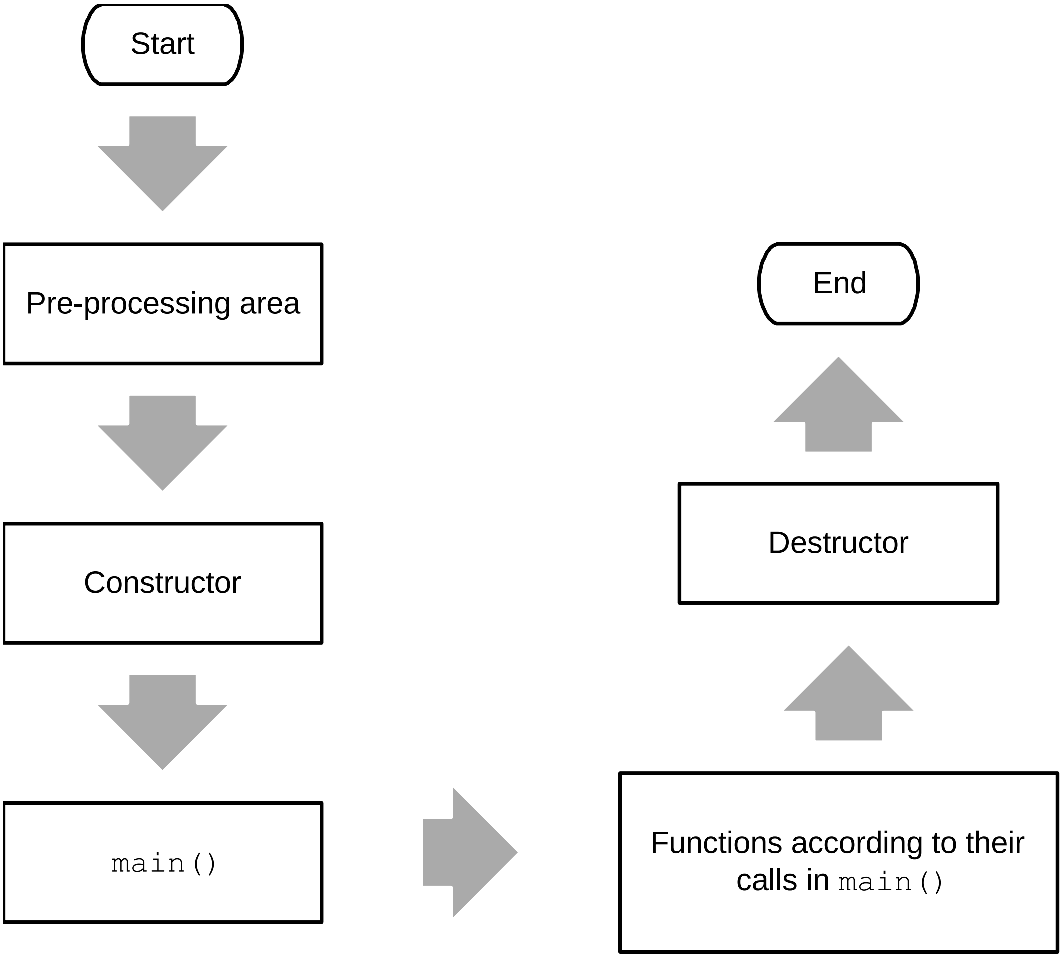


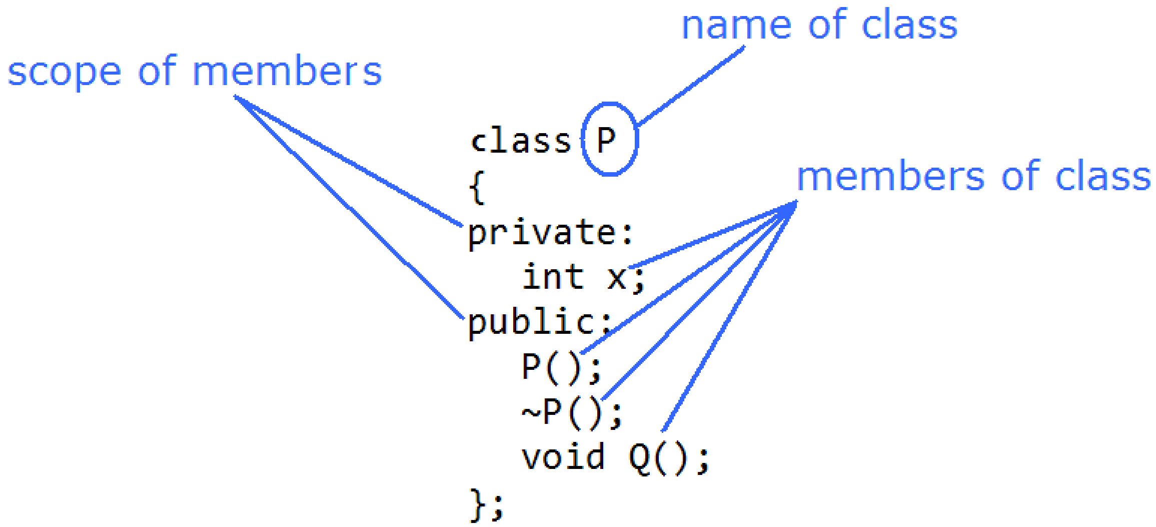
Figure 6.1. Order of execution in a typical C++ program.

Class

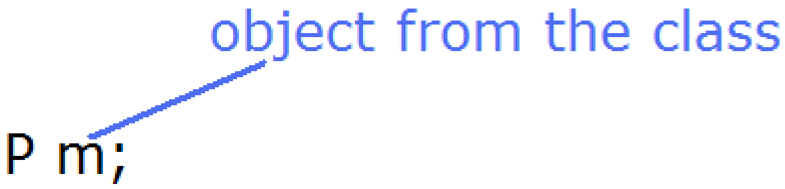
- a grouping of variables and functions that have a common ancestor
- a special form of structure
- *object* is an instance of a class
- members of a class consist of variables, functions and structures, which are global

```
class  Name of Class
{
private:
        List of variables
public:
        List of variables and functions
};
```

Example:



```
class P
{
private:
   int x;
public:
   P();
   ~P();
   void Q();
};
```



```
P m;
```

Visibility of Members

- Members of a class are variables, structures and functions
- Each member are global, and its scope is either private or public

private

- for variables or structures only
- the scope is global but limited to member functions only

public

- for variables, structures and functions
- The scope covers the whole program including non-member functions

Function

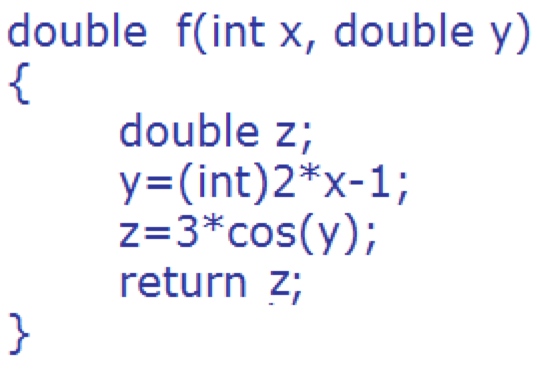
- a module or one unit of work in a program that can be called for solving a given problem
- Needed in order to make the program structured and modular
- A function must have a name.
- A function must belong and declared into a class.
- A function must be a prototype from one of the followings

```
void
int
float
double
char
bool
```

- A function of type other than `void` must return a value, and its return value is received by a variable from another function.
- A function may include argument(s) for passing or receiving data to another function.

```
type ClassName :: FunctionName(arguments)
{
        // body of statements
}
```

Example:



```
double  f(int x, double y)
{
        double z;
        y=(int)2*x-1;
        z=3*cos(y);
        return z;
}
```

Special Functions

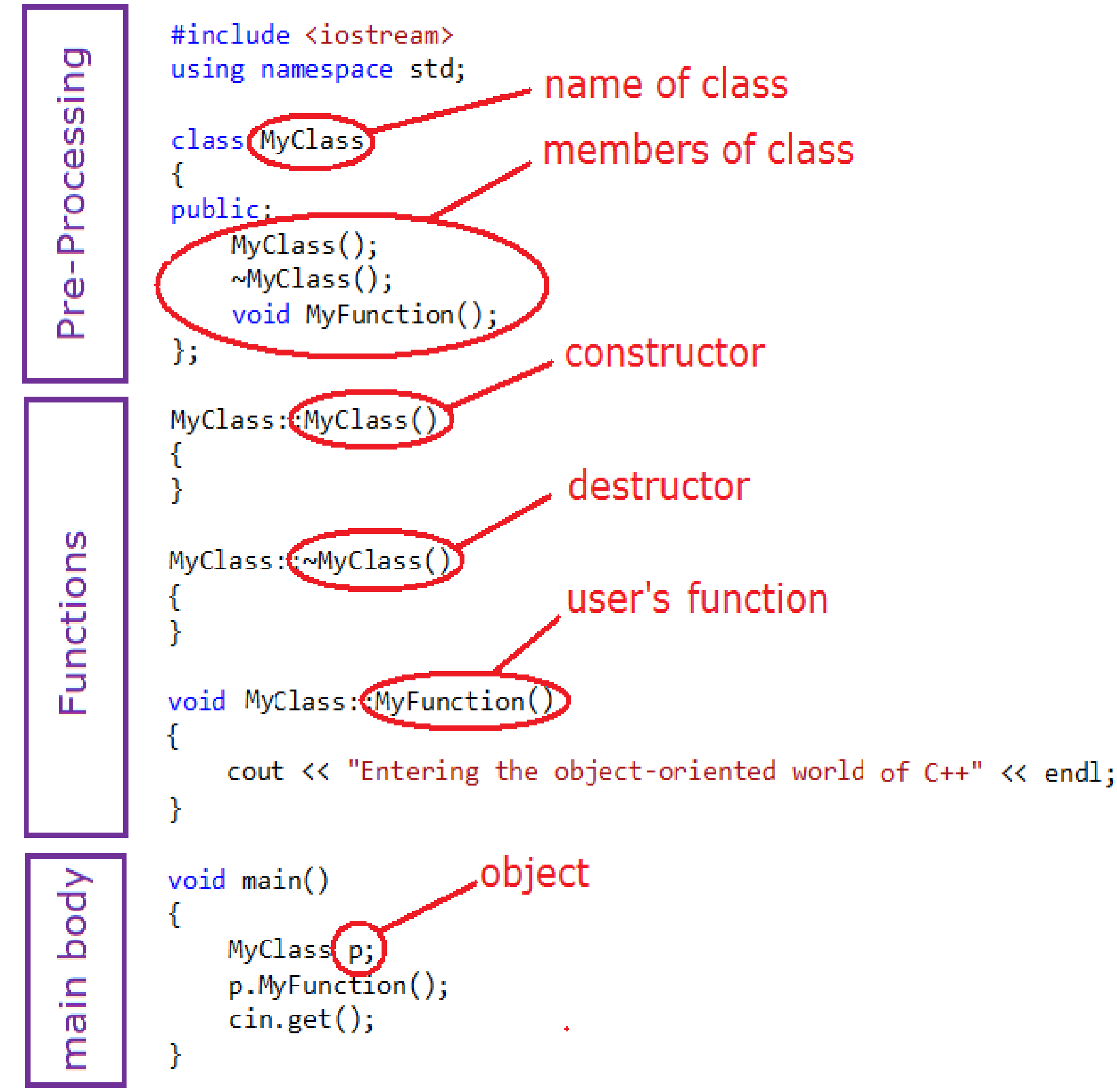
Constructor

- A special function which has the same name as the class
- a function with no specific class
- allocates memory for the class
- a suitable place for initializing global variables

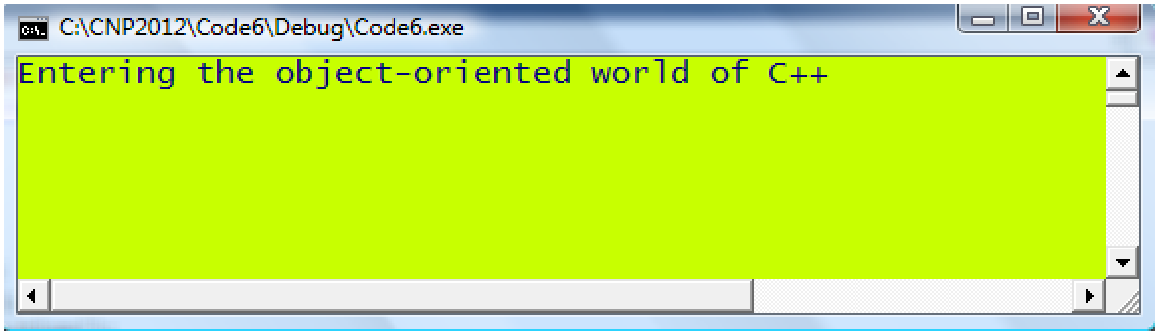
Destructor

- a special function which has the same name as the class preceded with a tilde
- a function with no specific class
- deallocates memory from the class
- a suitable place for destroying global variables and arrays

VC2010 **Code6A.cpp**: Class and object.



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

class MyClass
{
public:
        MyClass();
        ~MyClass();
        void MyFunction();
};

MyClass::MyClass()
{
}

MyClass::~MyClass()
{
}

void MyClass::MyFunction()
{
        cout << "Entering the object-oriented world of C++" << endl;
}

void main()
{
        MyClass p;
        p.MyFunction();
        cin.get();
}
```



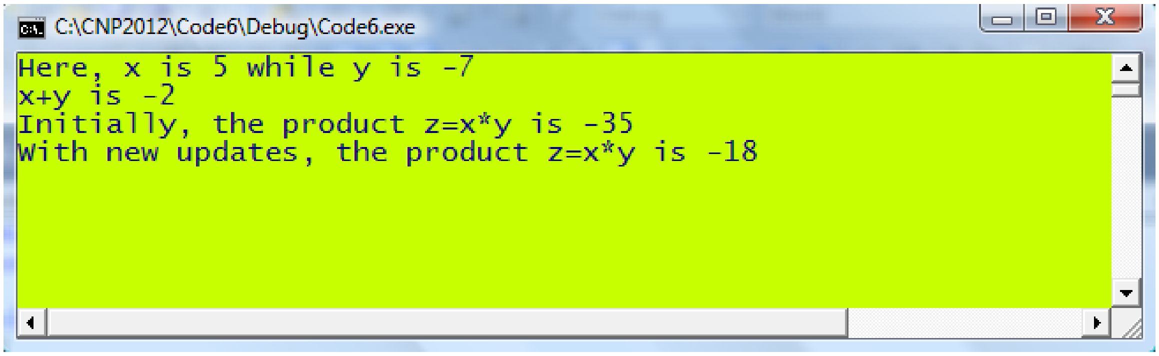
```
Entering the object-oriented world of C++
```

VC2010 **Code6B.cpp**: Class, object and method.

```
#include <iostream>
using namespace std;
class MyClass
{
public:
      int x,y;
      MyClass();
      ~MyClass();
      void MyFunction();
};

MyClass::MyClass()                          // constructor
{
      x=5; y=-7;
}

MyClass::~MyClass()                         // destructor
{
}

void MyClass::MyFunction()
{
      cout << "Here, x is " << x << " while y is " << y << endl;
      cout << "x+y is " << x+y << endl;
}

void main()
{
      int z;
      MyClass p;
      p.MyFunction();
      z=p.x*p.y;
      p.x=2; p.y=-9;
      z=p.x*p.y;
      cout << "With new updates, the product z=x*y is " << z << endl;
      cin.get();
 }
```

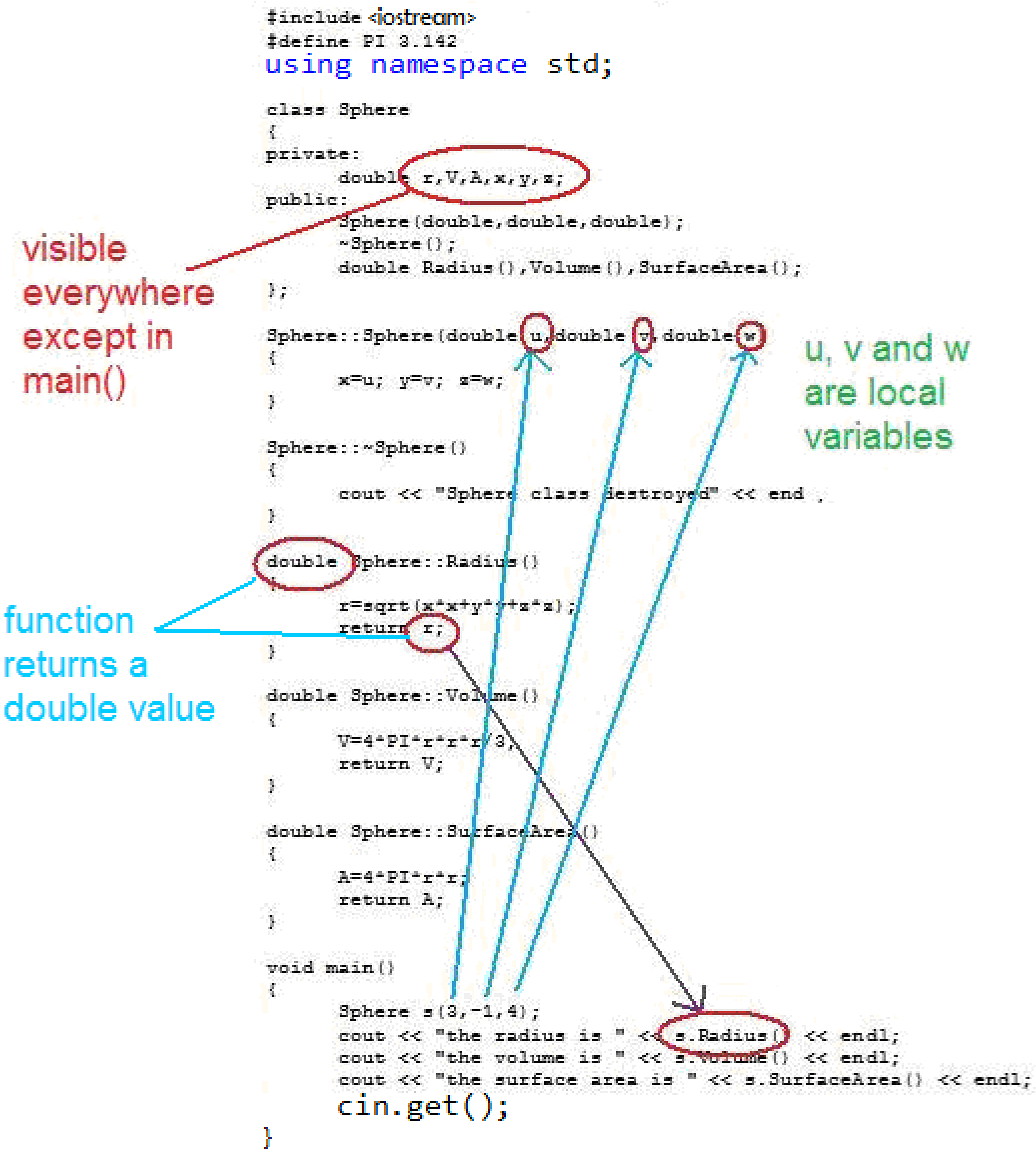
visible everywhere

z is a local variable, p.x and p.y are global



```
Here, x is 5 while y is -7
x+y is -2
Initially, the product z=x*y is -35
With new updates, the product z=x*y is -18
```

VC2010 **Code6C.cpp**: Class construction.

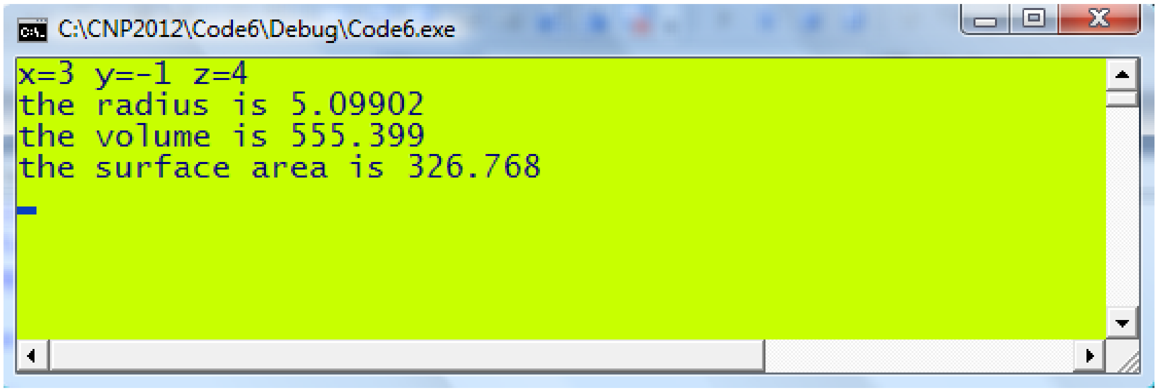


```
#include <iostream>
#define PI 3.142
using namespace std;

class Sphere
{
private:
        double r,V,A,x,y,z;
public:
        Sphere(double,double,double);
        ~Sphere();
        double Radius(),Volume(),SurfaceArea();
};

Sphere::Sphere(double u,double v,double w)
{
        x=u; y=v; z=w;
}

Sphere::~Sphere()
{
        cout << "Sphere class destroyed" << end ,
}

double Sphere::Radius()
{
        r=sqrt(x*x+y*y+z*z);
        return r;
}

double Sphere::Volume()
{
        V=4*PI*r*r*r/3;
        return V;
}

double Sphere::SurfaceArea()
{
        A=4*PI*r*r;
        return A;
}

void main()
{
        Sphere s(3,-1,4);
        cout << "the radius is " << s.Radius() << endl;
        cout << "the volume is " << s.Volume() << endl;
        cout << "the surface area is " << s.SurfaceArea() << endl;
        cin.get();
}
```

visible everywhere except in main()

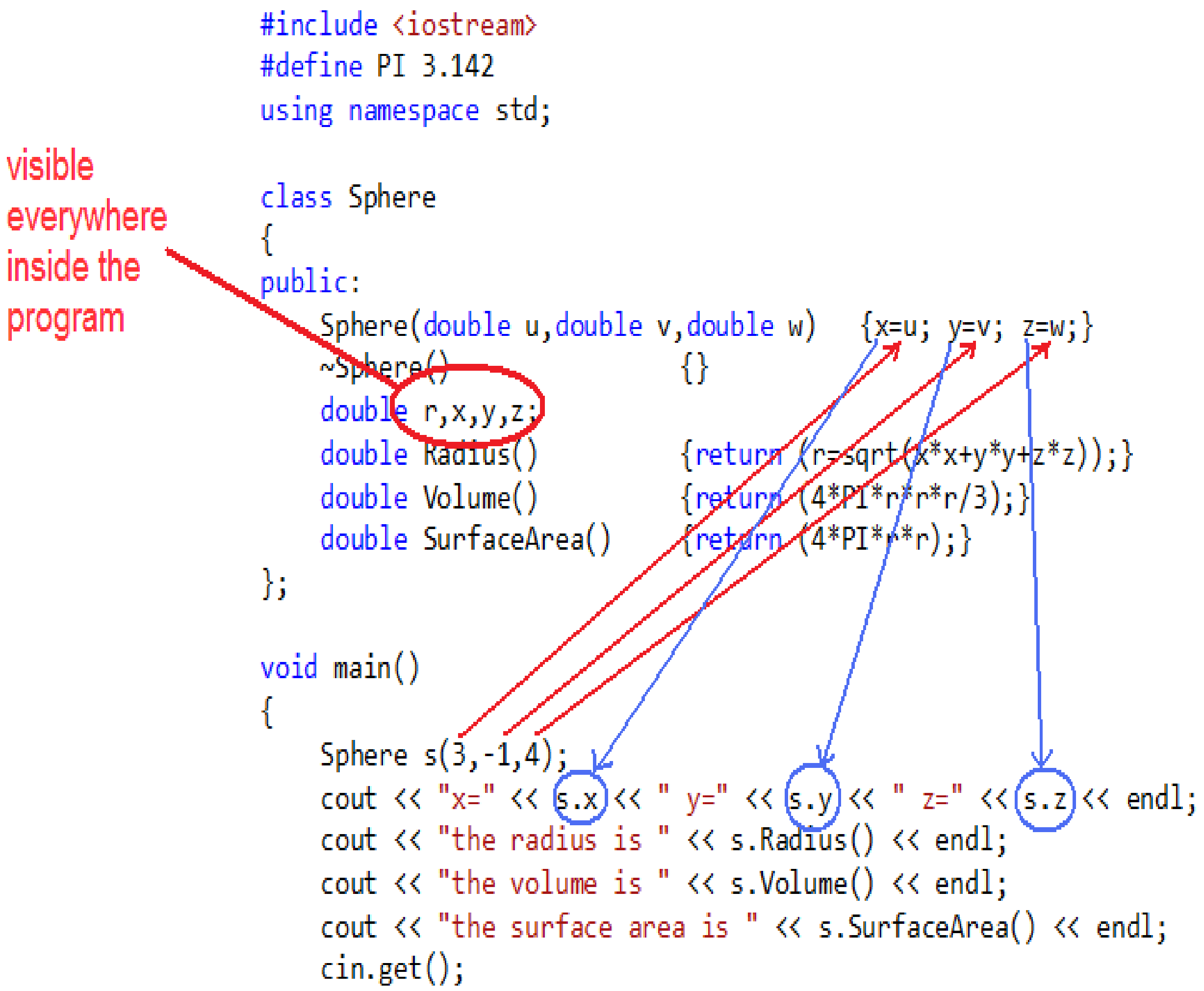
u, v and w are local variables

function returns a double value



```
x=3 y=-1 z=4
the radius is 5.09902
the volume is 555.399
the surface area is 326.768
```

VC2010 **Code6D.cpp**: Class construction in a compact form.



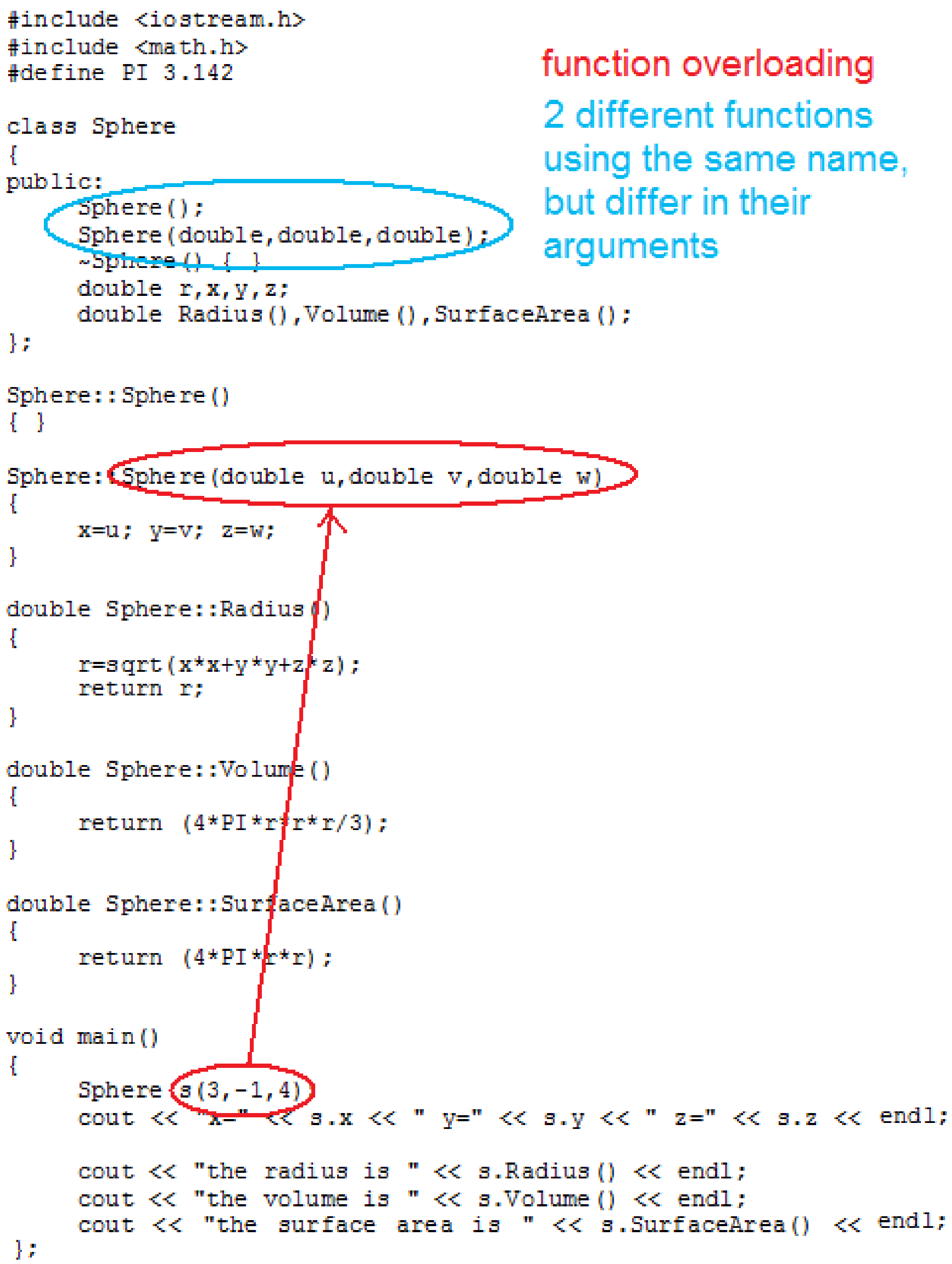
```
#include <iostream>
#define PI 3.142
using namespace std;

class Sphere
{
public:
    Sphere(double u,double v,double w)  {x=u; y=v; z=w;}
    ~Sphere()                           {}
    double r,x,y,z;
    double Radius()                     {return (r=sqrt(x*x+y*y+z*z));}
    double Volume()                     {return (4*PI*r*r*r/3);}
    double SurfaceArea()                {return (4*PI*r*r);}
};

void main()
{
    Sphere s(3,-1,4);
    cout << "x=" << s.x << " y=" << s.y << " z=" << s.z << endl;
    cout << "the radius is " << s.Radius() << endl;
    cout << "the volume is " << s.Volume() << endl;
    cout << "the surface area is " << s.SurfaceArea() << endl;
    cin.get();
```

visible everywhere inside the program

Function Overloading

- Different functions using the same name

VC2010 **Code6E.cpp**: Constructor overloading.



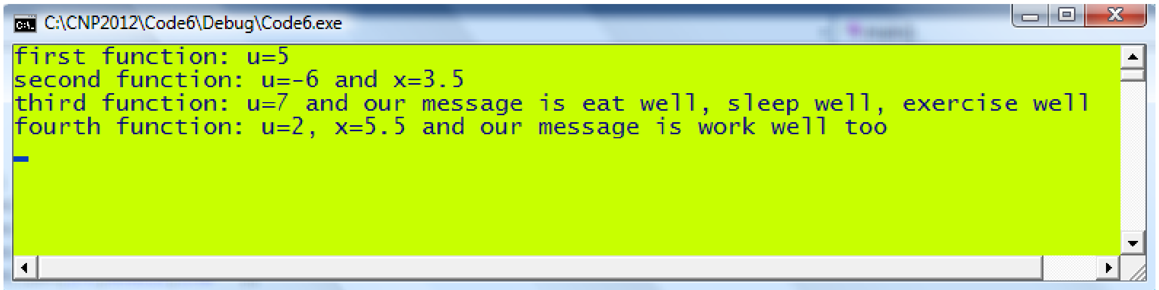
```
#include <iostream.h>
#include <math.h>
#define PI 3.142

class Sphere
{
public:
     Sphere();
     Sphere(double,double,double);
     ~Sphere() { }
     double r,x,y,z;
     double Radius(),Volume(),SurfaceArea();
};

Sphere::Sphere()
{ }

Sphere::Sphere(double u,double v,double w)
{
     x=u; y=v; z=w;
}

double Sphere::Radius()
{
     r=sqrt(x*x+y*y+z*z);
     return r;
}

double Sphere::Volume()
{
     return (4*PI*r*r*r/3);
}

double Sphere::SurfaceArea()
{
     return (4*PI*r*r);
}

void main()
{
     Sphere s(3,-1,4);
     cout << "x=" << s.x << " y=" << s.y << " z=" << s.z << endl;

     cout << "the radius is " << s.Radius() << endl;
     cout << "the volume is " << s.Volume() << endl;
     cout << "the surface area is " << s.SurfaceArea() << endl;
};
```

function overloading
2 different functions using the same name, but differ in their arguments

VC2010

Code6F.cpp: Overloading in functions.

```
#include <iostream.h>

class MyGraph
{
public:
	MyGraph()			{}
	~MyGraph()			{}
	void MyFunction(int);
	void MyFunction(int,double);
	void MyFunction(int,char *);
	void MyFunction(int,double,char *);
};

void MyGraph::MyFunction(int u)
{
	cout << "first function: u="<< u << endl;
}

void MyGraph::MyFunction(int u,double x)
{
	cout << "second function: u=" << u << " and x=" << x << endl;
}

void MyGraph::MyFunction(int u,char *s)
{
	cout << "third function: u=" << u << " and our message is "<< s << endl;
}

void MyGraph::MyFunction(int u,double x,char *s)
{
	cout << "fourth function: u=" << u
		<< ", x=" << x << " and our message is " << s << endl;
}

void main()
{
	MyGraph g;
	g.MyFunction(5);
	g.MyFunction(-6,3.5);
	g.MyFunction(7,"eat well, sleep well, exercise well");
	g.MyFunction(2,5.5,"work well too");
}
```



C:\CNP2012\Code6\Debug\Code6.exe

```
first function: u=5
second function: u=-6 and x=3.5
third function: u=7 and our message is eat well, sleep well, exercise well
fourth function: u=2, x=5.5 and our message is work well too
```

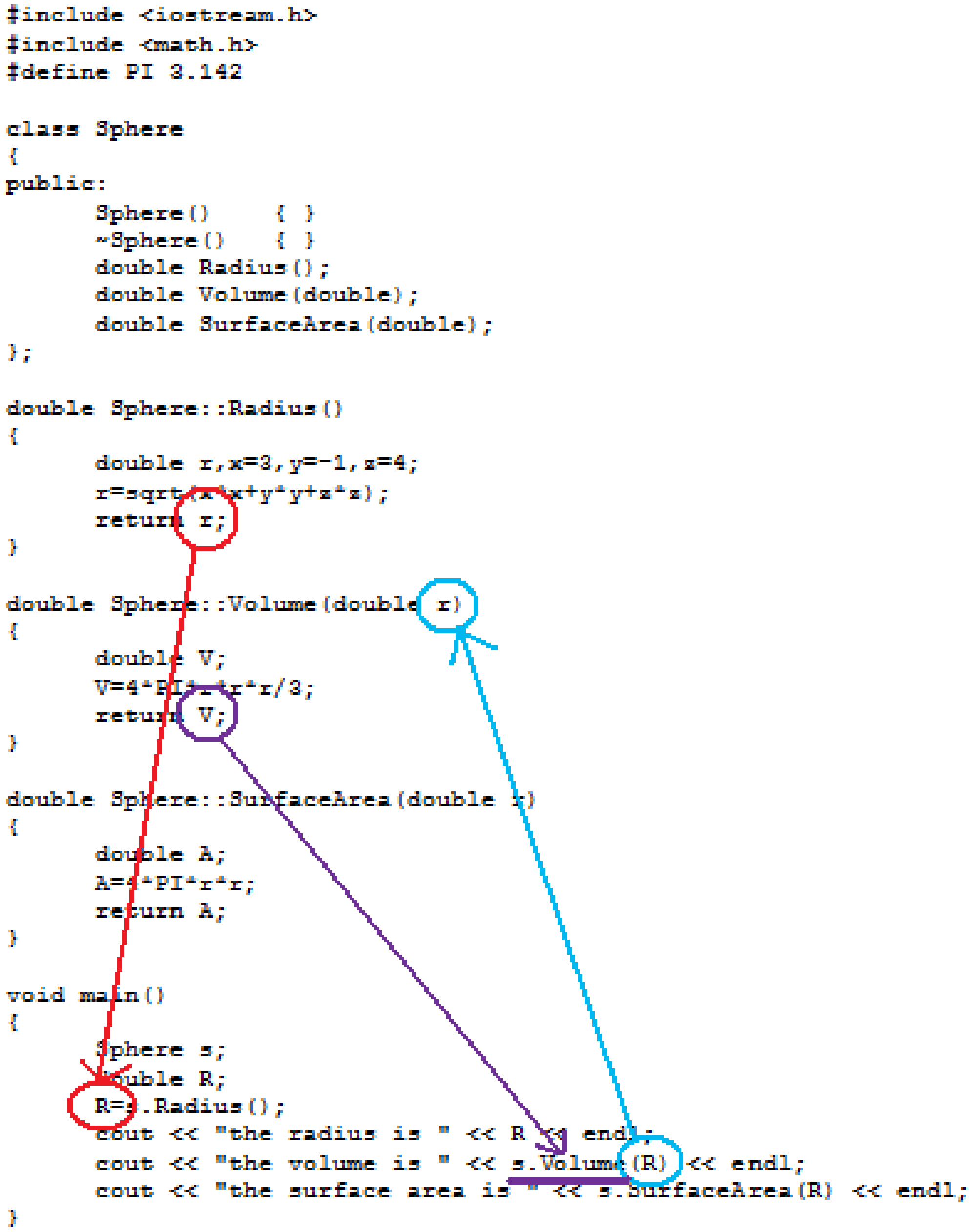
Data Passing

- The transfer of data from one function to another

Rules for data passing between functions

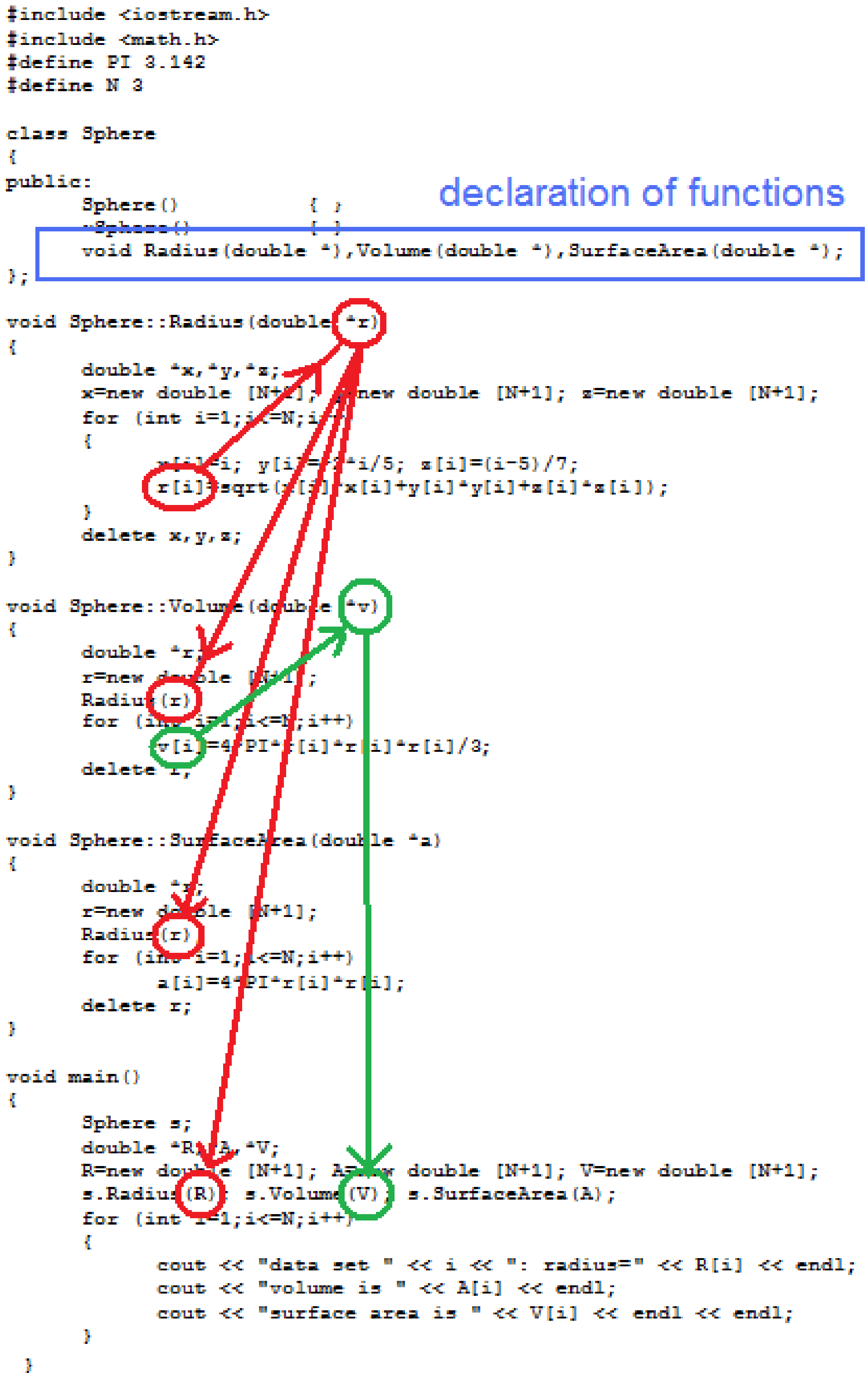
- both the passing and receiving functions must be declared with the same prototypes
- both the passing and receiving functions must match in their argument types and prototypes

VC2010 **Code6G.cpp**: Data passing between functions.

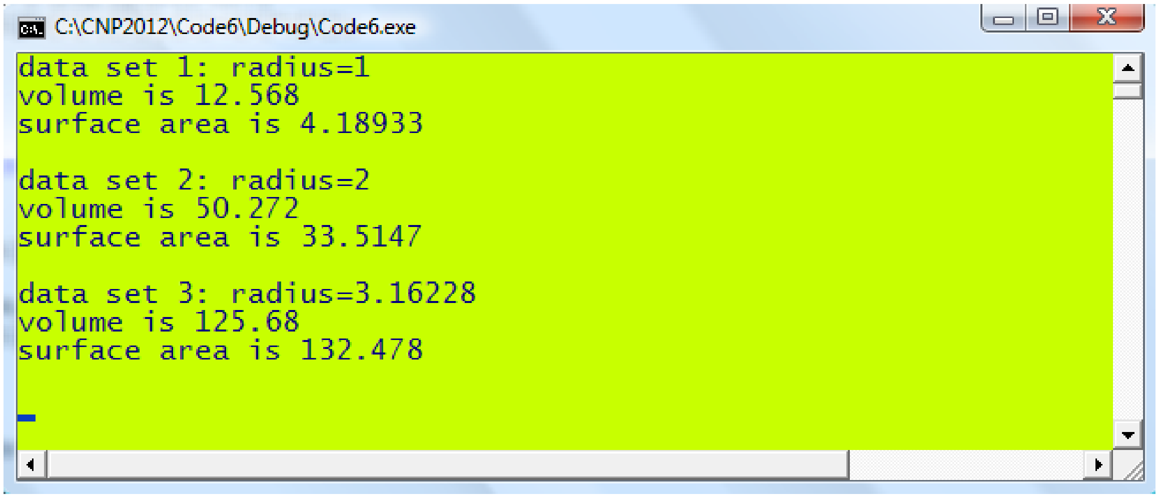


```
#include <iostream.h>
#include <math.h>
#define PI 3.142

class Sphere
{
public:
        Sphere()      { }
        ~Sphere()     { }
        double Radius();
        double Volume(double);
        double SurfaceArea(double);
};

double Sphere::Radius()
{
        double r,x=3,y=-1,z=4;
        r=sqrt(x*x+y*y+z*z);
        return r;
}

double Sphere::Volume(double r)
{
        double V;
        V=4*PI*r*r*r/3;
        return V;
}

double Sphere::SurfaceArea(double r)
{
        double A;
        A=4*PI*r*r;
        return A;
}

void main()
{
        Sphere s;
        double R;
        R=s.Radius();
        cout << "the radius is " << R << endl;
        cout << "the volume is " << s.Volume(R) << endl;
        cout << "the surface area is " << s.SurfaceArea(R) << endl;
}
```

VC2010 **Code6H.cpp**: Array passing between functions.



```
#include <iostream.h>
#include <math.h>
#define PI 3.142
#define N 3

class Sphere
{
public:
        Sphere()            { }
        ~Sphere()           { }
        void Radius(double *),Volume(double *),SurfaceArea(double *);
};

void Sphere::Radius(double *r)
{
        double *x,*y,*z;
        x=new double [N+1]; y=new double [N+1]; z=new double [N+1];
        for (int i=1;i<=N;i++)
        {
                x[i]=i; y[i]=2*i/5; z[i]=(i-5)/7;
                r[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);
        }
        delete x,y,z;
}

void Sphere::Volume(double *v)
{
        double *r;
        r=new double [N+1];
        Radius(r)
        for (int i=1;i<=N;i++)
                v[i]=4*PI*r[i]*r[i]*r[i]/3;
        delete r;
}

void Sphere::SurfaceArea(double *a)
{
        double *r;
        r=new double [N+1];
        Radius(r)
        for (int i=1;i<=N;i++)
                a[i]=4*PI*r[i]*r[i];
        delete r;
}

void main()
{
        Sphere s;
        double *R,*A,*V;
        R=new double [N+1]; A=new double [N+1]; V=new double [N+1];
        s.Radius(R); s.Volume(V); s.SurfaceArea(A);
        for (int i=1;i<=N;i++)
        {
                cout << "data set " << i << ": radius=" << R[i] << endl;
                cout << "volume is " << A[i] << endl;
                cout << "surface area is " << V[i] << endl << endl;
        }
  }
```



```
data set 1: radius=1
volume is 12.568
surface area is 4.18933

data set 2: radius=2
volume is 50.272
surface area is 33.5147

data set 3: radius=3.16228
volume is 125.68
surface area is 132.478
```

6.7 Data Passing involving Structure

Rules for data passing

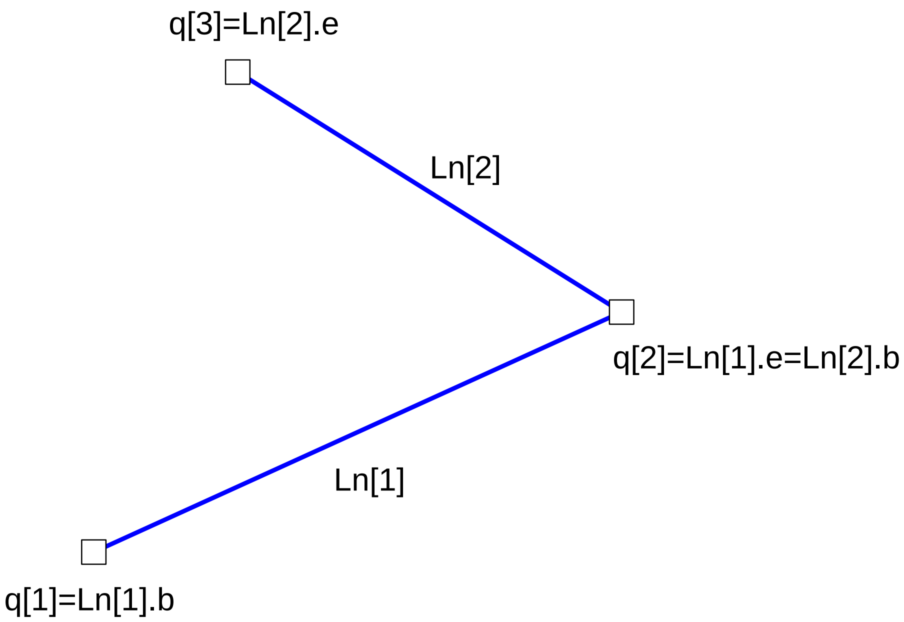
- both the passing and receiving functions must be declared with the same prototypes
- both the passing and receiving functions must match in their argument types and prototypes

`POINT` and `LINE` are structures.

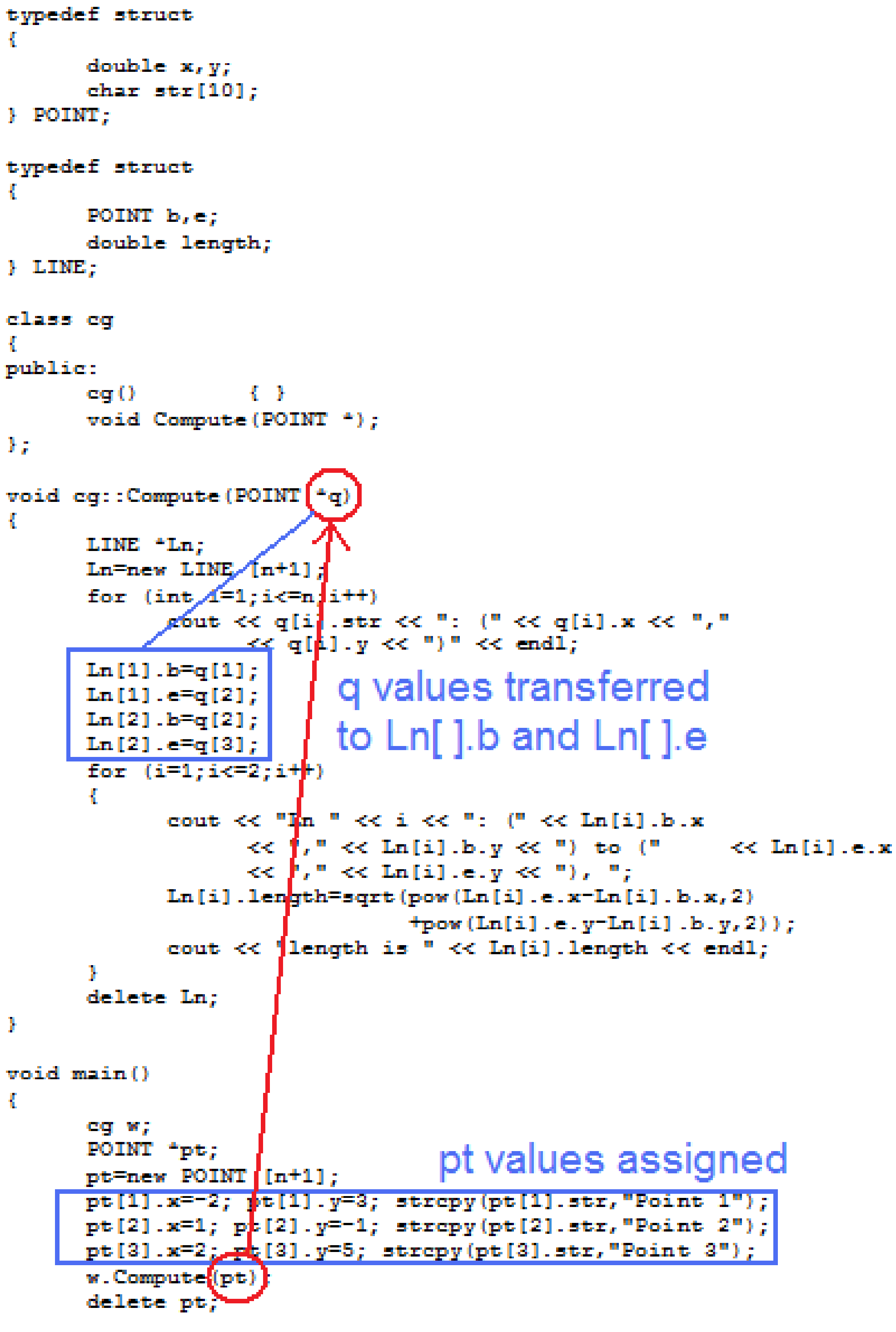
`Ln` is a local object of `LINE`.

`p`, `q`, `b` and `e` are the objects of `POINT`.

Who are the members of `POINT` and `LINE`?



VC2010 **Code6I.cpp**: Class with structure**.**



```
typedef struct
{
        double x,y;
        char str[10];
} POINT;

typedef struct
{
        POINT b,e;
        double length;
} LINE;

class cg
{
public:
        cg()            { }
        void Compute(POINT *);
};

void cg::Compute(POINT *q)
{
        LINE *Ln;
        Ln=new LINE [n+1];
        for (int i=1;i<=n;i++)
                cout << q[i].str << ": (" << q[i].x << ","
                        << q[i].y << ")" << endl;
        Ln[1].b=q[1];
        Ln[1].e=q[2];
        Ln[2].b=q[2];
        Ln[2].e=q[3];
        for (i=1;i<=2;i++)
        {
                cout << "Ln " << i << ": (" << Ln[i].b.x
                        << "," << Ln[i].b.y << ") to ("       << Ln[i].e.x
                        << "," << Ln[i].e.y << "), ";
                Ln[i].length=sqrt(pow(Ln[i].e.x-Ln[i].b.x,2)
                                    +pow(Ln[i].e.y-Ln[i].b.y,2));
                cout << "length is " << Ln[i].length << endl;
        }
        delete Ln;
}

void main()
{
        cg w;
        POINT *pt;
        pt=new POINT [n+1];
        pt[1].x=-2; pt[1].y=3; strcpy(pt[1].str,"Point 1");
        pt[2].x=1; pt[2].y=-1; strcpy(pt[2].str,"Point 2");
        pt[3].x=2; pt[3].y=5; strcpy(pt[3].str,"Point 3");
        w.Compute(pt);
        delete pt;
```

6.8 Class Inheritance

- *Inheritance* refers to resources such as variables belonging to an earlier class which can be shared by newer classes.
- The earlier class is called the *base class*
- The newer classes are called *inherited classes*

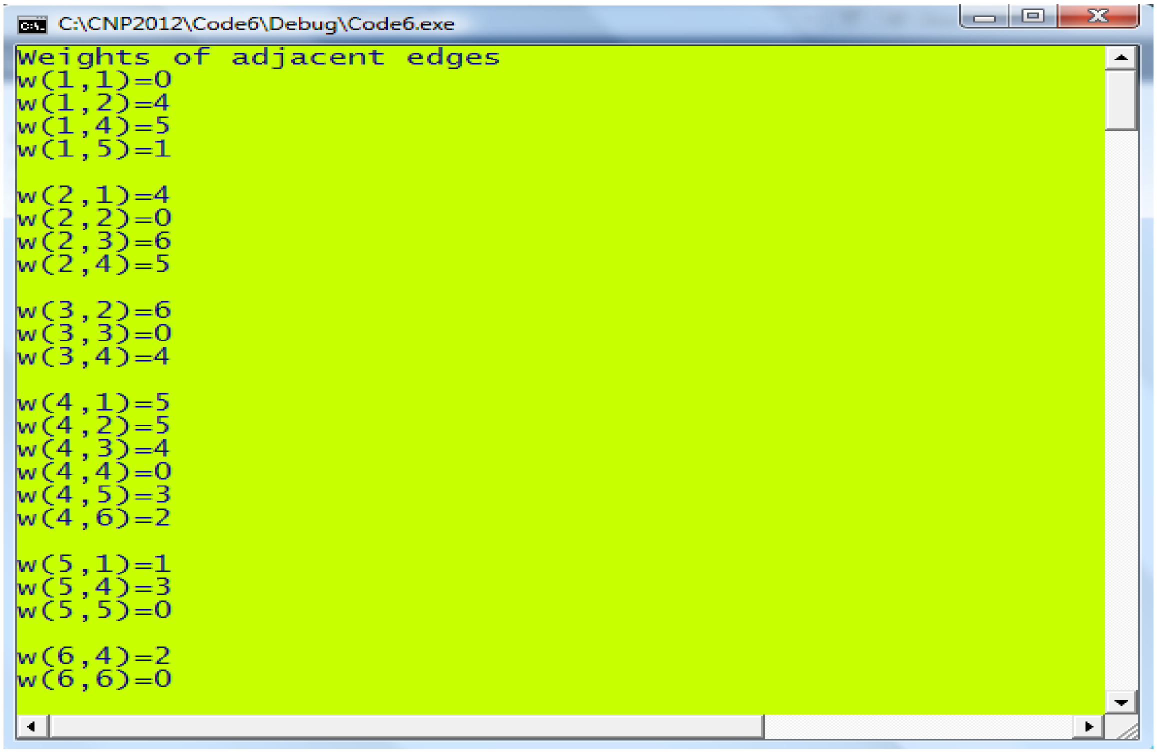
Inheritance rules

- the inherited class can access any member of the base class, but not the other way around
- several levels of inheritance are supported in C++

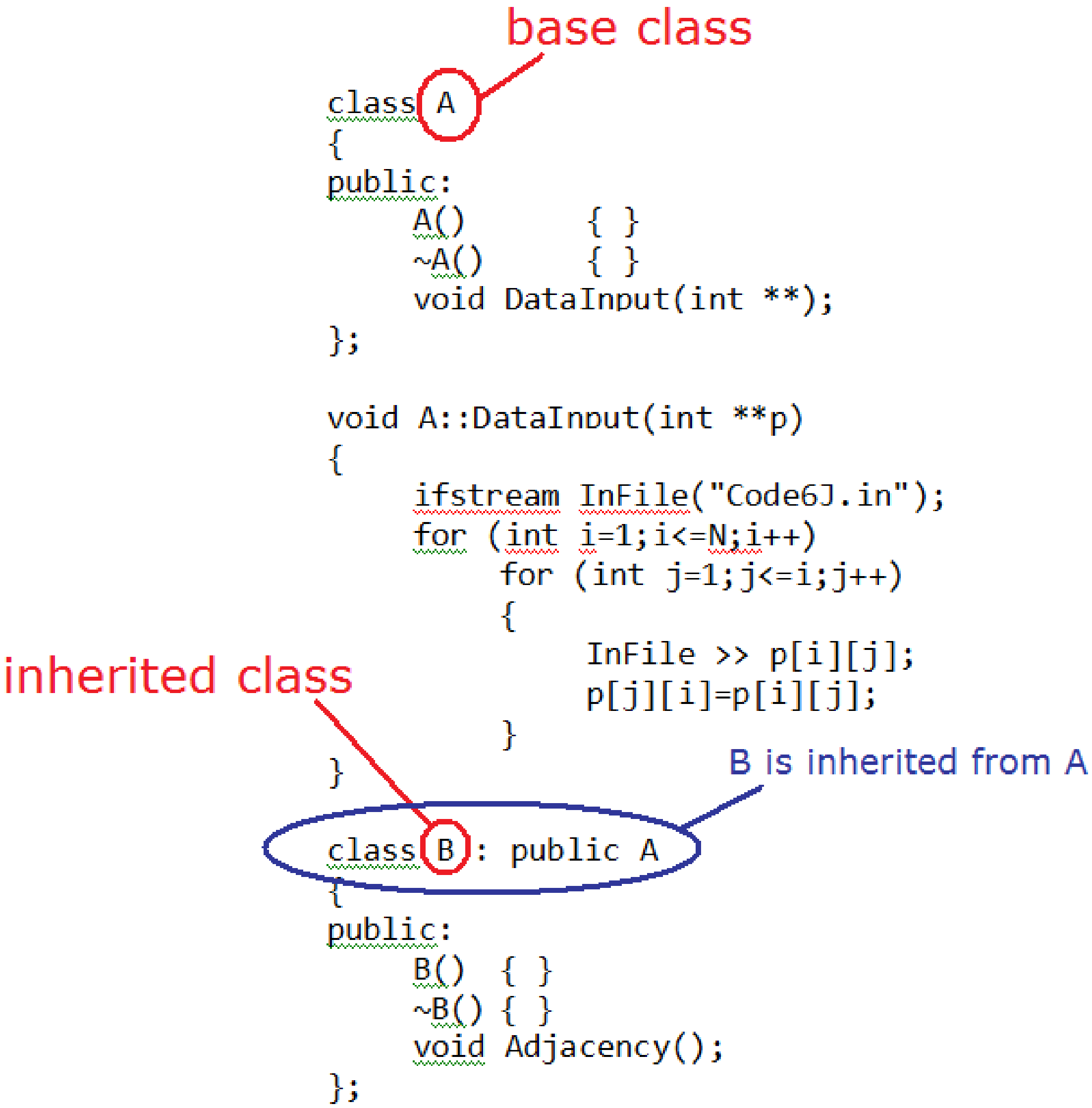
VC2010 **Code6J.cpp**: Two classes collaborating.

```
#include <fstream>
#include <iostream>
#define N 6
using namespace std;

class A
{
public:
       A()                  {}
       ~A()                 {}
       void DataInput(int **);
};

void A::DataInput(int **p)
{
       ifstream InFile("Code6J.in");
       for (int i=1;i<=N;i++)
              for (int j=1;j<=i;j++)
              {
                     InFile >> p[i][j];
                     p[j][i]=p[i][j];
              }
}

class B
{
public:
       B()           {}
       ~B()          {}
       void Adjacency();
};
```

```
void B::Adjacency()
{
        A g;
        int **q,i,j;
        q=new int *[N+1];
        for (i=1;i<=N;i++)
                q[i]=new int [N+1];
        g.DataInput(q);
        cout << "Weights of adjacent edges" << endl;
        for (i=1;i<=N;i++)
        {
                for (j=1;j<=N;j++)
                        if (q[i][j]!=99)
                                cout << "w(" << i << "," << j << ")="
                                        << q[i][j] << endl;
                cout << endl;
        }
        delete q;
}

void main()
{
        B m;
        m.Adjacency();
        cin.get();
}
```



```
Weights of adjacent edges
w(1,1)=0
w(1,2)=4
w(1,4)=5
w(1,5)=1

w(2,1)=4
w(2,2)=0
w(2,3)=6
w(2,4)=5

w(3,2)=6
w(3,3)=0
w(3,4)=4

w(4,1)=5
w(4,2)=5
w(4,3)=4
w(4,4)=0
w(4,5)=3
w(4,6)=2

w(5,1)=1
w(5,4)=3
w(5,5)=0

w(6,4)=2
w(6,6)=0
```



base class

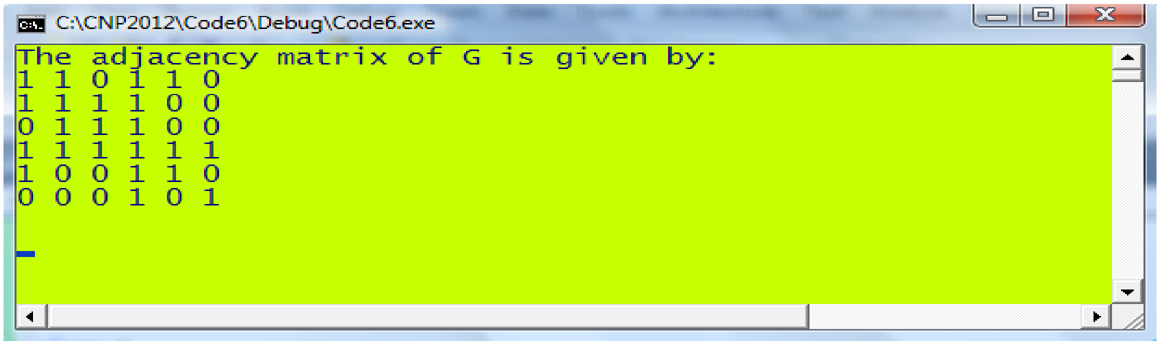
```
class A
{
public:
     A()         { }
     ~A()        { }
     void DataInput(int **);
};

void A::DataInput(int **p)
{
     ifstream InFile("Code6J.in");
     for (int i=1;i<=N;i++)
          for (int j=1;j<=i;j++)
          {
               InFile >> p[i][j];
               p[j][i]=p[i][j];
          }
}

class B : public A
{
public:
     B()  { }
     ~B() { }
     void Adjacency();
};
```

inherited class

B is inherited from A

VC2010 **Code6K.cpp**: Illustrating inheritance.

```
#include <fstream>
#include <iostream>
#define N 6
using namespace std;

class A
{
public:
       A()           { }
       ~A()          { }
       void DataInput(int **);
};

void A::DataInput(int **p)
{
       ifstream InFile("Code6J.in");
       for (int i=1;i<=N;i++)
              for (int j=1;j<=i;j++)
              {
                     InFile >> p[i][j];
                     p[j][i]=p[i][j];
              }
}

class B : public A
{
public:
       B()    { }
       ~B()   { }
       void Adjacency();
};

void B::Adjacency()
{
       int **q,i,j;
       q=new int *[N+1];
       for (i=1;i<=N;i++)
              q[i]=new int [N+1];
       DataInput(q);
       cout << "The adjacency matrix of G is given by: " <<endl;
       for (i=1;i<=N;i++)
       {
              for (j=1;j<=N;j++)
                     cout << ((q[i][j]==99)?0:1) << " ";
              cout << endl;
       }
       cout <<endl;
       delete q;
}

void main()
{
       B m;
       m.Adjacency();
       cin.get();
}
```

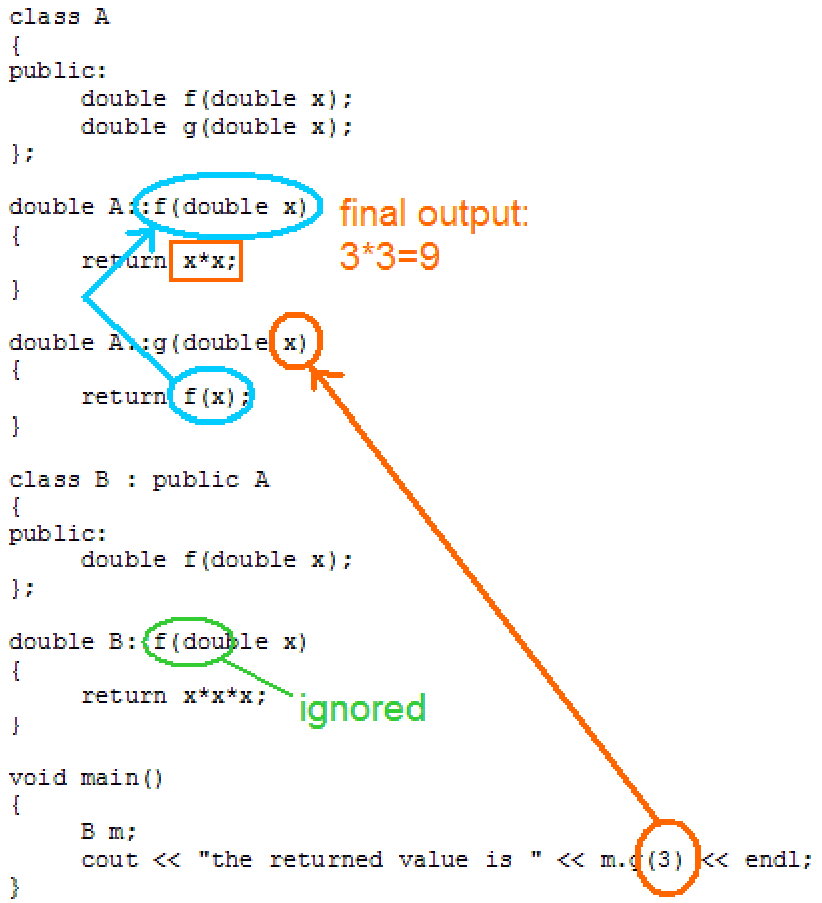


```
The adjacency matrix of G is given by:
1 1 0 1 1 0
1 1 1 1 0 0
0 1 1 1 0 0
1 1 1 1 1 1
1 0 0 1 1 0
0 0 0 1 0 1
```

Polymorphism

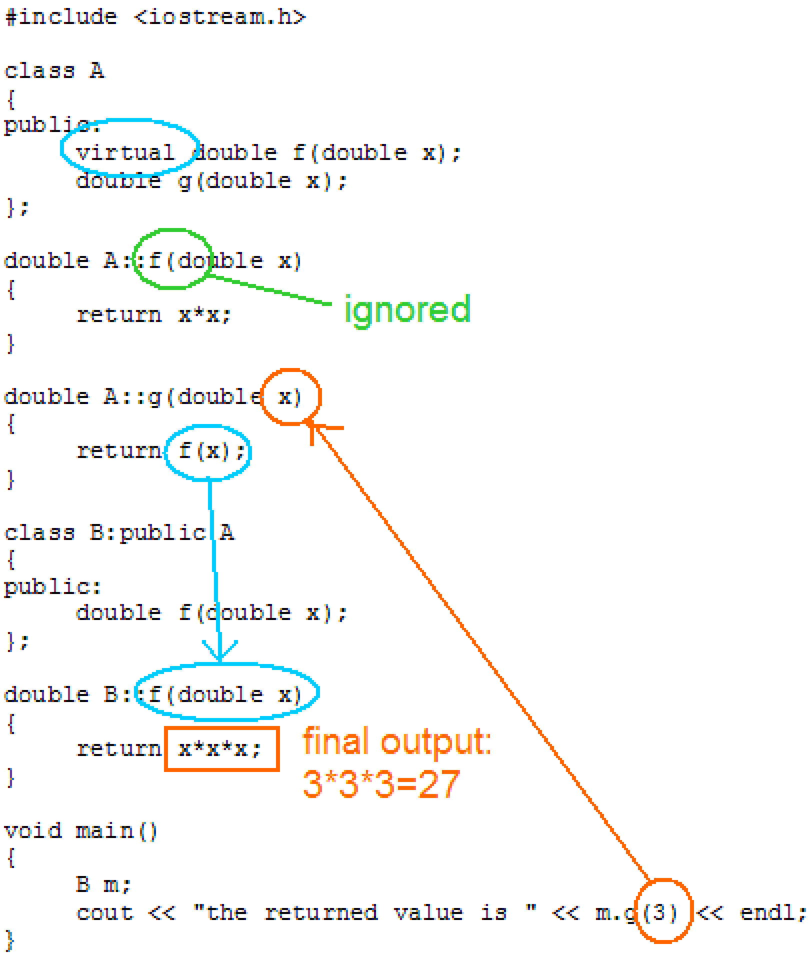
- 2 or more functions using the same name which exist in two or more classes, including the base and derived classes
- For these 2 functions, a function from the base class is called by default whenever it is called (refer to Code6L.cpp)
- To override the function from the base class, always declare the function in the base class as **virtual** (refer to Code6M.cpp)

Code6L.cpp: without virtual functions



```
class A
{
public:
     double f(double x);
     double g(double x);
};

double A::f(double x)
{
     return x*x;
}

double A::g(double x)
{
     return f(x);
}

class B : public A
{
public:
     double f(double x);
};

double B::f(double x)
{
     return x*x*x;
}

void main()
{
     B m;
     cout << "the returned value is " << m.g(3) << endl;
}
```

Code6M.cpp: with virtual functions



```
#include <iostream.h>

class A
{
public:
     virtual double f(double x);
     double g(double x);
};

double A::f(double x)
{
     return x*x;
}

double A::g(double x)
{
     return f(x);
}

class B:public A
{
public:
     double f(double x);
};

double B::f(double x)
{
     return x*x*x;
}

void main()
{
     B m;
     cout << "the returned value is " << m.g(3) << endl;
}
```

ignored

final output:
3*3*3=27