Lecture 4-7

Classes and Objects

Contents

- Introduction to Classes and Objects
- Class declaration
- Creating Objects
- Accessing Object Members
- Defining Member Functions
- Nesting of Member Function
- Access Specifiers: Public , Private
- Nested Classes

Classes

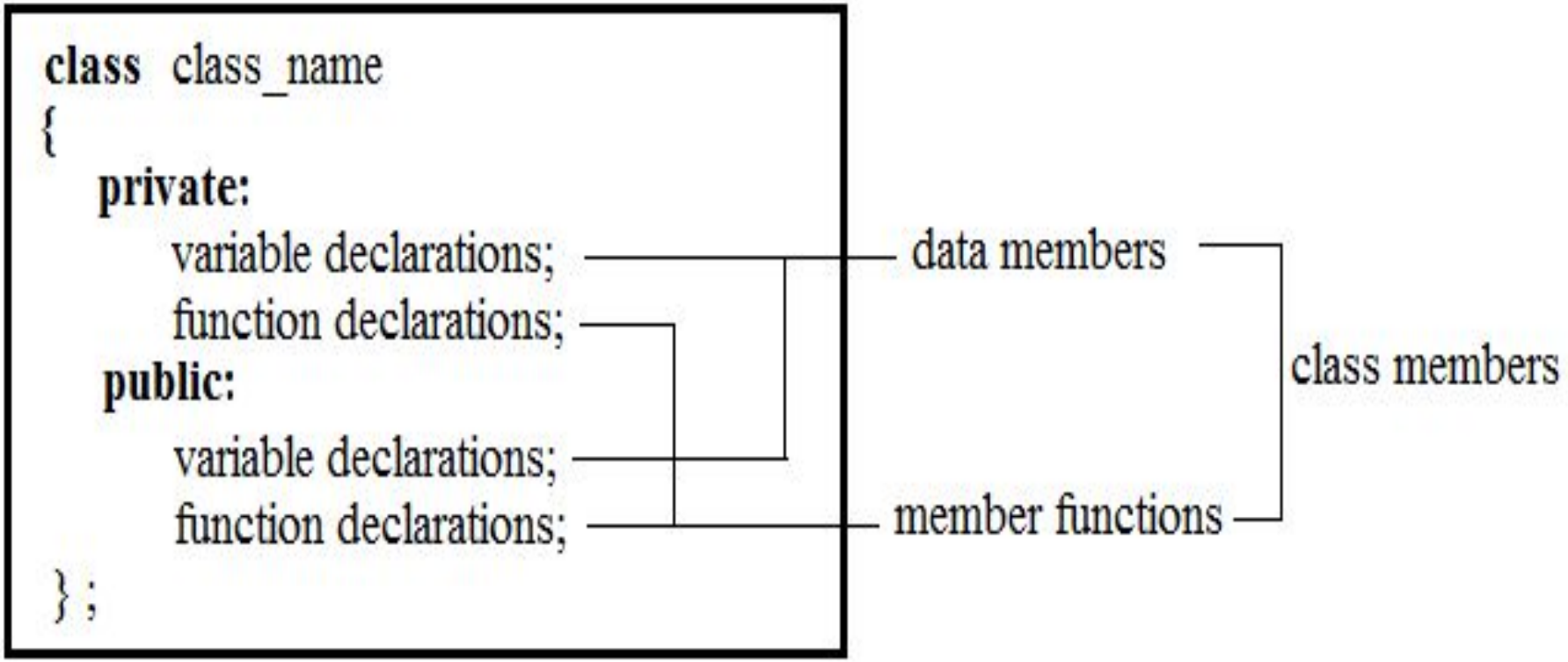
- Different from standard C structures
  - Adds member **functions**
  - Not just member data
- Integral to object-oriented programming
  - Focus on objects
    - Object: Contains data and operations
    - In C++, variables of class type are objects

Specifying a Class

- A class is a way to bind the data and its associated functions together.
- A class specification has 2 parts:
  - Class declaration
  - Class function definitions

- **Class declaration**: describes type and scope of its members.
- **Class function definitions**: describes how the class functions are implemented.

General form of Class declaration

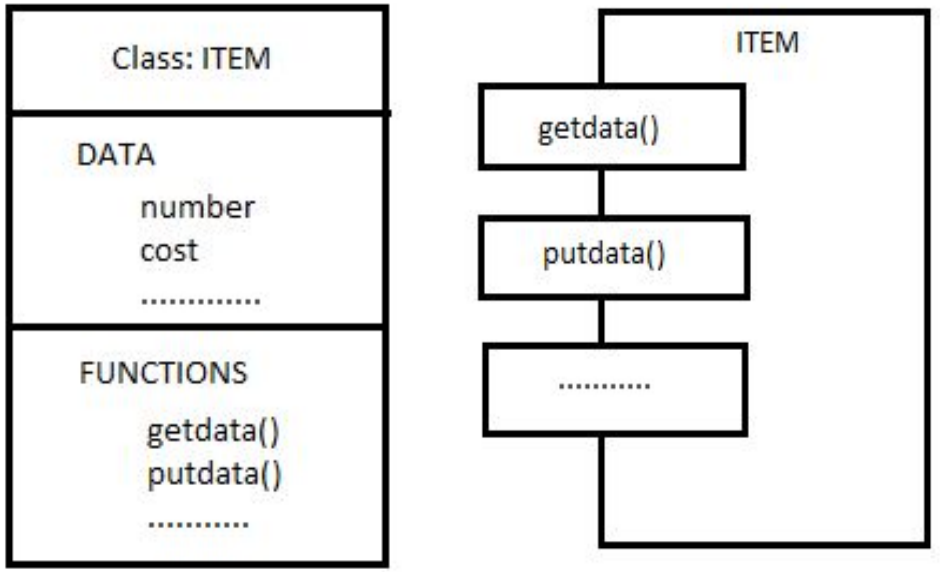


Difference between Structure and Class

- By default, members of class are private, while, by default, members of structure are public.
- Encapsulation
- The keywords public and private are known as **visibility labels**.

A Simple Class Example

```
class item
{
        int number;
        float cost;
    public :
        void getdata(int a, float b);
        void putdata (void);
};
```



Class Representation

Creating Objects:

item x;

Accessing Class Members:

Object_name.function-name (actual-arguments);

x.getdata(100, 75.5); x.putdata( );

Creating Objects

- Objects can also be created as follows:

```
class item
{
        .....
        .....
        .....
} x, y, z;
```

Complete Example of a Class - 1

```
#include <iostream>
using namespace std;
class myclass
{
      public:
            int i, j, k;       // accessible to entire program
};
int main()
{
      myclass a, b;
      a.i = 100;       // access to i, j, and k is OK
      a.j = 4;
      a.k = a.i * a.j;
      b.k = 12;        // remember, a.k and b.k are different
      cout << a.k << " " << b.k;
      return 0;
}
```

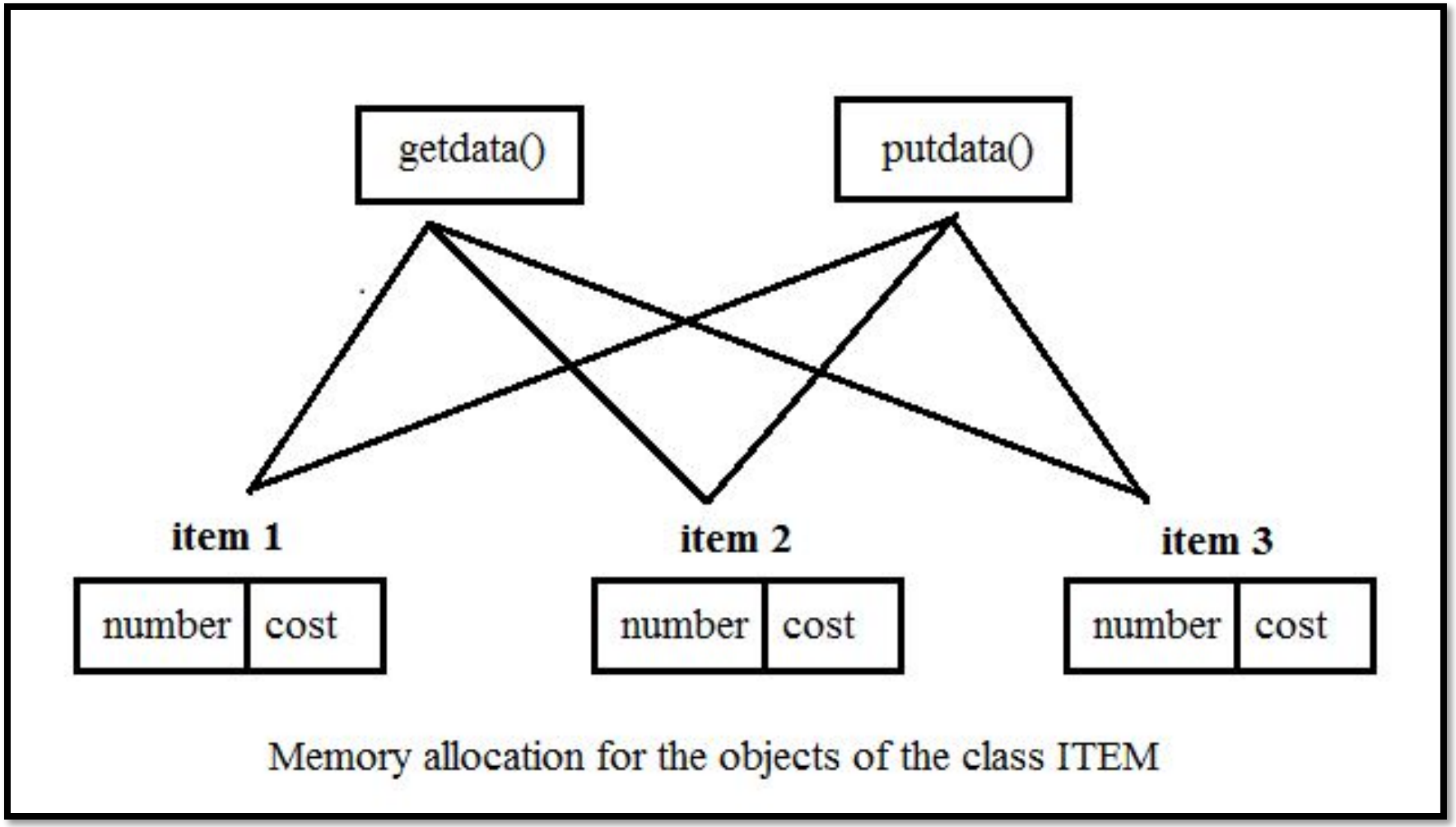
Complete Example of a Class - 2

```
#include <iostream>
using namespace std;
class myclass
{
	int a, b;
	public:
		void init(int i, int j) { a=i; b=j; }
		void show() { cout << a << " " << b << "\n"; }
};
int main()
{
	myclass x;
	x.init(10, 20);
	x.show();
	return 0;
}
```

Memory Allocation for Objects

- Memory space for objects is allocated when they are declared and not when the class is specified : **partially true.**
- Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created.
- Only space for member variables is allocated separately for each object. Because, member variables will hold different data values for different objects.

Example



Memory allocation for the objects of the class ITEM

Member Functions

Defining Member Functions

- **Outside the class definition**
  When function is defined outside class, it requires a prototype declaration in the class.

```
return-type class-name :: function-name (argument declaration)
{
    Function body
}
```

```
void item :: getdata (int a, int b)
{
    number = a;
    cost = b;
}
```

- **Inside the class definition**
  When function is defined inside class, it does not require a prototype declaration in the class.

```
class item
{
    int number;
    float cost;
  public:
    void getdata(int a, float b)
    {
        number=a;
        cost=b;
    }
};
```

Inline Functions

- In C++, you can create short functions that are not actually called; rather, their code is expanded in line at the point of each invocation.
- This process is similar to using function-like macro.
- To cause a function to be expanded in line rather than called, precede its definition with the **inline keyword.**

Inline Functions

- We can define a member function outside the class definition and still make it inline by using the qualifier **inline** in the header line of the function definition.

```
class item
{
        .....
    public:
        void getdata(int a, float b);
};
inline void item :: getdata (int a, int b)
{
        number a;
        cost = b;
}
```

Nesting of Member Functions

Nesting of Member Functions

- An object of a class using dot operator generally calls a member function of the class.
- However, a member function may also be called by using its name inside another member function of the same class.
- This is known as “Nesting of Member Functions”

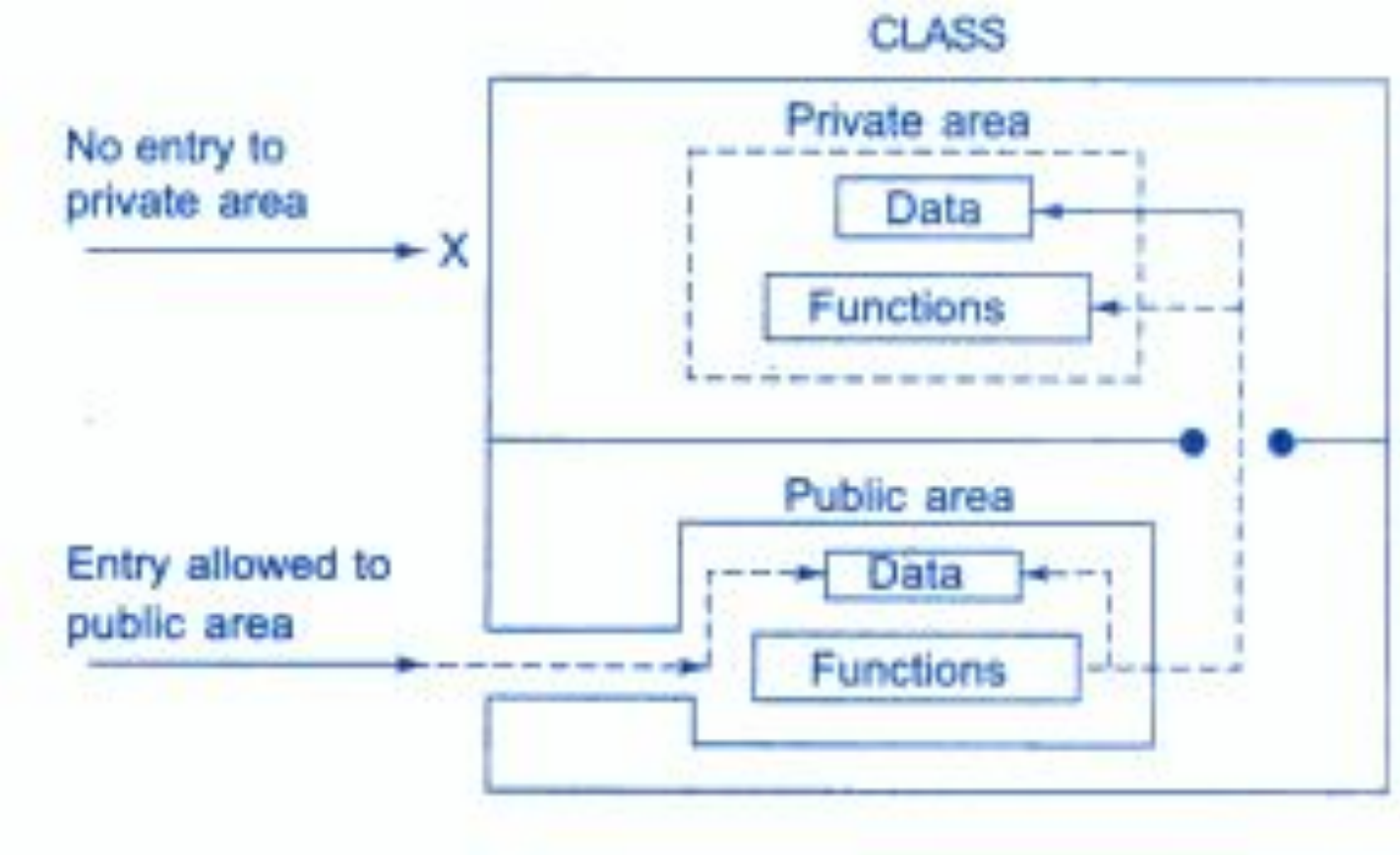
Nesting of Member Functions

```
class emp
{
        float basic;
public:
        void display()
                { cout<<basic; }
        void takedata()
          {    cin>>basic;
               display();
           }
};
 void main()
{
          emp e1;
          e1.takedata();
}
```

A member function of a class can be called by another member function Of the same class. takedata() is calling display()

Access Specifiers

Data Hiding Revisited



Access Specifier: Public

- Public members are accessible outside the class.
- Public members are accessible to both member Functions and non-member Functions.
- Objects can access the members directly using dot operator. E.g.
  - object name.public data member
  - object name.public member function

Access Specifier: Public

```
class emp
{   int empno;
    char empname[20];
    float basic;
public:
   char address[20];
        void readdata()
        {       cin>>empno;
                gets(empname);
                gets(address);
                cin>>basic;
        }
        void displaydata()
        {       cout<<empno;
                cout<<empname;
                cout<<basic<<address;
        }
};// end of class
```

emp e1;
Outside the class we can access

e1.address //OK
Object name.public data member

e1.readdata() //OK
Object name.public member function

e1.displaydata()//OK
Object name.public member function

Access Specifier: Private

- Private Members can only be accessed by the member functions of that class.
- They cannot be accessed by any non member functions (data is hidden from outside world).
- Object of a class cannot access private members using dot operators.
- All data at the beginning of a class is by default private, until any specifier is mentioned.

Access Specifier: Private

```
class emp
{   int empno;
    char empname[20];
    float basic;
public:
        void readdata()
        {       cin>>empno;
                gets(empname);
                cin>>basic;
        }
        void displaydata()
        {       cout<<empno;
                cout<<empname;
                cout<<basic;
        }
};// end of class
```

```
void main()
{
   emp e1;
   //To input details
   cin>>e1.empno; //NOT OK

   e1.readdata(); //OK
   //To Display details
   e1.displaydata();// OK
}
```

Private Member Functions

```
class sample
{
        int m;
        void read (void);
    public:
        void update (void);
        void write (void);
};

if s1 is an object of sample, then

        s1.read();              //won't work; objects cannot access
                                //private members
```

Private Member Functions

s1.read()
is illegal. However, the function **read()** can be called by the function **update()** to update the value of **m.**

```
void sample : : update ( void )
{
        read();         //simple call; no object used
}
```

Nested Classes

Nested Classes

- A class is declared within another class.
- The outer class is called enclosing class and inner class is called the nested class.
- A nested class is a member and as such has the same access rights as any other member.
- The members of an enclosing class have no special access to members of a nested class; the usual access rules shall be obeyed.

Nested Classes Example

```
#include<iostream>
using namespace std;
class Enclosing                    /* start of Enclosing class declaration */
{
      int x;
      class Nested
      {                            /* start of Nested class declaration */
            int y;
             void NestedFun(Enclosing e)
             {
                  cout<<e.x;     // works fine: nested class can access
                                 // private members of Enclosing class
             }
   };                            // declaration Nested class ends here
};                               // declaration Enclosing class ends here
int main(){}
```

Nested Classes Example

```
#include<iostream>
using namespace std;
class Enclosing                 /* start of Enclosing class declaration */
{
      int x;
      class Nested
      {                         /* start of Nested class declaration */
            int y;
      };                        // declaration Nested class ends here
      void EnclosingFun(Nested n)
      {
                  cout<<n.y;      // Compiler Error: y is private in Nested        }

};                              // declaration Enclosing class ends here
int main(){}
```

Thank You