Object oriented Programming with C++

Classes and Objects

Class

- Class is a way to bind data and its associated functions together.
- It allows the data (and functions) to be hidden, if necessary, from external use.
- When defining a class, we creating a new abstract data type that can be treated like any other built-in data type.
- Class specification has two parts:
  - Class declaration
  - Class function definitions
- Class declaration describes the type and scope of its members.

Class

The general form of class declaration:

```
class class_name
{
  private:
     variable declaration;
     function declaration;
  public:
     variable declaration;
     function declaration;
};
```

Class

- Keyword **`class`** specifies, that what follows is an abstract data of type **`class_name`**.
- Body of class enclosed within braces and terminated by semicolon.
- Contains declaration of variables and functions, collectively called as class members.

Class

- Keyword **`private`** and **`public`** are known as visibility labels.
- If both keyword are missing then, by default, all members are **`private`** .
- **`public`** member function can have access to **`private`** data members and **`private`** functions.
- **`public`** members can be accessible from outside class.

Class

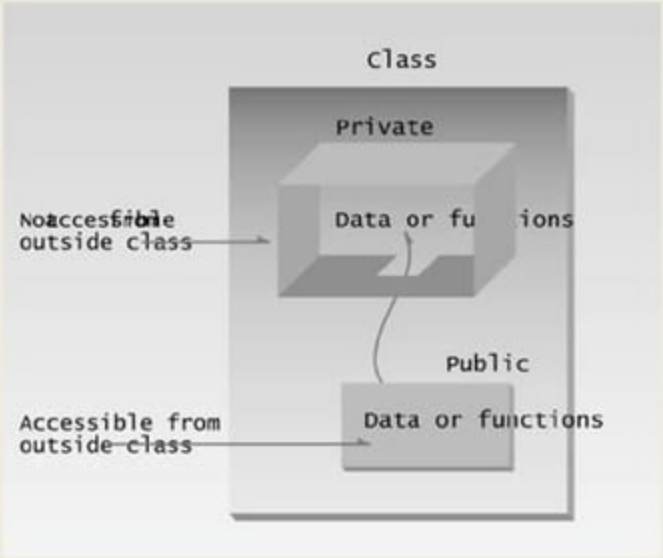


Fig. 1 Data hiding in classes

Class

A Simple class example:

```
class item
{
        int number;                          //variable declaration
        float cost;                          //private by default
   public:
        void getdata(int a,   float b);      //functions declaration
        void putdata(void);
};                                           //ends with semicolon
```

Creating Objects

- Once a class has been declared, we can create variables of that type by using the class name.
- For Example:

```
item x;      // memory for x is created.
item x, y, z;
```

- Another way,

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

Accessing Class Members

- Syntax:

```
object-name.function-name(argList);
```

- For example:

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

- Similarly,

```
x.putdata();
```

- The statement like,

```
getdata(100,75.5);
```

- has no meaning. Similarly,

```
x.number  = 100;
```

is illegal.

Defining Member Functions

- A member function of a class is a function that has its definition or its prototype within the class definition like any other variable.
- It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.
- Member functions can be defined in two parts:
  - Outside the class function
  - Inside the class function
- Both place definition should perform same task.

Outside class definition

- Member functions that are declared inside a class have to defined separately outside the class.
- The general form of definitions is:

```
return-type class-name :: function-name (argList)
{
        function body
}
```

- Membership label **`class-name ::`** tells the compiler that function **`function-name`** belongs to **`class-name`** specified in header line.
- Symbol :: is called scope resolution operator.

Outside class definition

- For instance, consider function **`getdata()`** and **`putdata()`**.

```
void item :: getdata(int a, float b)
{
    number = a;
    cost = b;
}
void item :: putdata(void)
{
    cout << "Number :" << number << "\n";
    cout << "Cost :" << cost << "\n";
}
```

- Function do not return any value, their return type is void.

Inside class definition

```
class item
{
            int number;
            int cost;
    public:
            void getdata(int a, float b);//declaration
            //inline function
            void putdata(void)
            {
                    cout << "Number :" << number << "\n";
                    cout << "Cost :" << cost << "\n";
            }
};
```

- When a function is defined inside a class, it is treated as **inline** function;

Inline function

- We must keep inline functions small, small inline functions have better efficiency.
- Inline functions do increase efficiency, but we should not make all the functions inline.
- Because if we make large functions inline, it may lead to code bloat, and might affect the speed too.
- Hence, it is advised to define large functions outside the class definition using scope resolution :: operator, because if we define such functions inside class definition, then they become inline automatically.

Inline function

- The compiler is unable to perform inlining if the function is too complicated. So we must avoid big looping conditions inside such functions.
- In case of inline functions, entire function body is inserted in place of each call, so if the function is large it will affect speed and memory badly.

Making an outside function inline:

```
class item
{
            int number;                          //private by default
            float cost;
      public:
            void getdata(int a, float b);        //Declaration
};
inline void item :: getdata (int a, float b)     // use membership label
{
      number = a;                                //private variables
      cost = b;                                  //directly used
}
```

C++ program with class

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

class item
{
            int number;        //private by default
            float cost;
      public:
            void getdata(int a, float b); //Declaration
            //function defined inside class
            void putdata()
            {
                  cout << "Number :" << number << endl;
                  cout << "Cost :" << cost << endl;
            }
};
```

C++ program with class

```
//.............Member Function Definition.................
void item :: getdata (int a, float b)    // use membership label
{
      number = a;                        //private variables
      cost = b;                          //directly used
}
//.............Main program...................
int main()
{
      item x;                             //create object x
      x.getdata (200, 175.50);            //call member function
      x.putdata();                        //call member function
      return 0;
}
```

Programming Exercise:

Define a class to display triangle:

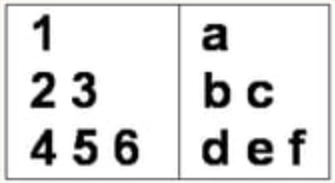
Include the following members:

Data member:

- Row
- Column

Member function:

- display_num_tri();
- Display_alph_tri();



Programming Exercise:

Define a class to interchange the values of X and Y:

Include the following members:

Data member:

- X
- Y

Member function:

- getdata();
- putdata();
- swapnum();

Programming Exercise:

Define a class to calculate area of rectangle:

Include the following members:

Data member:

- length
- breadth

Member function:

- getdata();
- putdata();
- area();

Nesting of member functions

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

class set
{
            int m,n;
      public:
            void input(void);
            int largest(void);
            void display(void)
            {
                  cout << "Largest value :" << largest() << endl;
            }
};
```

Nesting of member functions

```
//.....Member Function Definition.....
void set :: input (void)
{
      cout << "Enter the values of m and n: " << endl;
      cin >>m>>n;
}
int set :: largest()
{
      if (m >= n)
            return m;
      else
            return n;
}
//.....Main program.....
int main()
{
      set x;
      x.input();
      x.display();
      return 0;
}
```

Private member functions

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

If x is an object of sample.

```
x.read();  // won't work

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

Static data members

- Static data members hold global data that is common to all objects of the class.
- For example, such global data are,
  - Count of objects currently present,
  - Common data accessed by all objects, etc.
- Let us consider class Account. We want all objects of this class to calculate interest at the rate of say 4.5%.
- Therefore, this data should be globally available to all objects of this class.

Static data members

Characteristics:

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
- It is visible only within the class, but its lifetime is the entire program.

Static data members

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

class item
{
      static int count;
      int number;
  public:
      void getdata (int a)
      {
            number = a;
            count ++;
      }
      void getcount()
      {
            cout << "Count: " << count << "\n";
      }
};

int item :: count;
```

Static data members

```
int main()
{
      item a, b, c;              //count initialized to zero.

      cout << "Before reading data:" << "\n";

      a.getcount();              //display count.
      b.getcount();              //display count.
      c.getcount();              //display count.

      a.getdata(100);            //getting data.
      b.getdata(200);            //getting data.
      c.getdata(300);            //getting data.

      cout << "After reading data:" << "\n";

      a.getcount();              //display count.
      b.getcount();              //display count.
      c.getcount();              //display count.

      return 0;
}
```

Static member function

- `static` keyword makes the function free from the individual object of the class and its scope is global in the class without creating any side effect for the other part of the program.
- A member function that is declared as `static` has a following **properties**:
  - A static function can have access to only other static members (functions or variables) declared in the same class.
  - A static member function can be called using the class name (instead of its objects) as follows:

```
class-name :: function-name;
```

Static member function:

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

class test
{
            static int c;
      public:
            static void count()
            {
                  c++;
            }
            static void display()
            {
                  cout << "Value of count:" << c << endl;
            }
};
```

Static member function:

```
int test :: c;

int main()
{
      test :: display();
      test :: count();
      test :: count();
      test :: display();
      return 0;
}
```

Friend function

- The central idea of encapsulation and data hiding concept is that any non-member function has no access permission to the private data of the class.
- The private members of the class are accessed only from member functions of that class.
- Any non-member function cannot access the private data of the class.
- C++ allows a mechanism, in which a non-member function has access permission to the private members of the class.
- This can be done by declaring a non-member function friend to the class whose private data is to be accessed. Here **friend** is a keyword.

Friend function

- For example, Consider a case where two classes, manager and scientist, have been defined.
- We would like to use a function income_tax() to operate on the objects of both these classes.
- Here we declare income_tax() as friend.
- Such function need not be a member of any of these classes.

Friend function

Properties:

- There is no scope restriction for the friend function; hence they can called without using objects.
- Unlike member functions of a class, friend function can not access the members directly. On the other hand, it uses objects and dot operator to access private and public member variable of class.
- Use of friend function is rarely done, because it violates the rule of encapsulation and data hiding.
- Function can be declared in public or private sections without changing its meaning.
- Usually, it has objects as arguments.

Friend function

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

class account
{
        char name [20];
        int acno;
        float balance;
    public:
        void read (void)
        {
            cout << "Enter Name: " << endl;
            cin >> name;
            cout << "Enter A/c no: " << endl;
            cin >> acno;
            cout << "Enter Balance: " << endl;
            cin >> balance;
        }
        friend void showbal (account); // declaration
};
```

```
void showbal (account ac)
{
    cout << "Balance of a/c no " << ac.acno << "is Rs. "<<ac.balance;
}

int main()
{
    account k;
    k.read();
    showbal(k); // call friend function.
    return 0;
}
```

Friend function

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

class ABC;  //forward declaration
class XYZ
{
        int data;
    public:
        void setvalue (int value)
        {
            data = value;
        }
        friend void add (XYZ, ABC); // declaration
};

class ABC
{
        int data;
    public:
        void setvalue (int value)
        {
            data = value;
        }
        friend void add (XYZ, ABC); // declaration
};
```

Friend function

```
void add (XYZ x, ABC a)
{
    cout << "Sum of data values: " << x.data + a.data <<endl;
}

int main()
{
    XYZ y;
    ABC b;
    y.setvalue(5);
    b.setvalue(50);
    add(y, b); // call friend function.
    return 0;
}
```

Friend function

"A friend function is a non-member function that has special rights to access private data members of any object of the class of whom it is a friend".

Friend classes

- A class can be a friend of another class. Member functions of a friend class can access private data members of objects of the class of which it is a friend.
- If a class B is to be made a friend of class A, then the statement

```
friend class B;
```

- Should be written within the definition of class A.

Friend classes

```
class manager;     //forward declaration
class emp
{
      int eid;
      char empname [10];
public:
      void getdata()
      {
            cout << "Enter Emp ID:" << endl;
            cin >> eid;
            cout << "Enter Emp Name: " <<endl;
            cin >> empname;
      }
      friend class manager;
};
```

Friend classes

```
class manager
{
public:
      void putdata (emp e)
      {
            cout << "Emp ID: " << e.eid << endl;
            cout << "Emp Name : " << e.empname << endl;
      }
};
int main()
{
      emp e;
      e.getdata ();
      manager m;
      m.putdata (e);
      return 0;
}
```

Friend classes

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

class Square;      //forward declaration.

class Rectangle
{
      int width, height;
public:
      void getdata(int w, int h)
      {
            width = w;
            height = h;
      }
      void display()
      {
            cout << "Rectangle: " << width * height << endl;
      }
      void morph(Square);
};
```

Friend classes

```
class Square
{
      int side;
public:
      void getdata(int s)
      {
            side = s;
      }
      void display()
      {
            cout << "Square: " << side * side << endl;
      }
      friend class Rectangle;
};

void Rectangle::morph(Square s)
{
      width = s.side;
      height = s.side;
}
```

We declared **Rectangle** as a friend of **Square** so that **Rectangle** member functions could have access to the private member, **Square::side**

Friend classes

```
int main ()
{
      Rectangle rec;
      rec.getdata (5, 10);

      Square sq;
      sq.getdata (5);

      cout << "Before:" << endl;
      rec.display();
      sq.display();

      rec.morph(sq);

      cout << "\nAfter:" << endl;
      rec.display();
      sq.display();

      return 0;
}
```

- In our example, **Rectangle** is considered as a friend class by **Square** but **Rectangle** does not consider **Square** to be a friend
- So **Rectangle** can access the private members of **Square** but not the other way around.

Array of objects:

- We can create many objects at a time.
- For example, if the name of the class is Student then,

```
Student s1,s2,s3;
```

- Here s1,s2,s3 are three objects .
- Consider there are so many students and therefore if we declare object for each student as s1,s2,s3,.sn it is very complicated for writing.
- For that we use array of objects

Array of objects:

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

class student
{
        int rno;
        char name [20];
        int m1,m2;
    public:
        void getinfo(void)
        {
            cout << "Enter details of student:";
            cin >> rno;
            cin >>name;
            cin >>m1>>m2;
        }
        void display(void)
        {
            cout << "\n" << rno<< "\t"<< name<<"\t";
            cout << m1 << "\t" << m2 << "\t" << m1 + m2;
        }
};
```

Array of objects:

```
int main()
{
    int num;
    cout <<"How many student records u want to enter?" << endl;
    cin >> num;

    student s[num]; // creating array of objects..

    cout << "Enter the information of "<< num << "students: " << endl;

    for (int i = 0; i < num; i++)
    {
        s[i].getinfo();
    }

    cout << "Students details: " << endl;
    cout << "Roll No \t" ;
    cout << "Name \t";
    cout << "Sub1 \t Sub2 \t Total";

    for (int i = 0; i < num; i++)
    {
        s[i].display();
    }
    return 0;
}
```

Objects as function arguments:

```
#include<iostream>
using namespace std;
class time
{
            int hours,minutes;
       public:
            void gettime(int h,int m)
            {
                  hours=h;
                  minutes=m;
            }
            void puttime(void)
            {
                  cout<<hours<<"hours and";
                cout<<minutes<<"minutes "<<endl;
            }
            void sum(time,time);
};
```

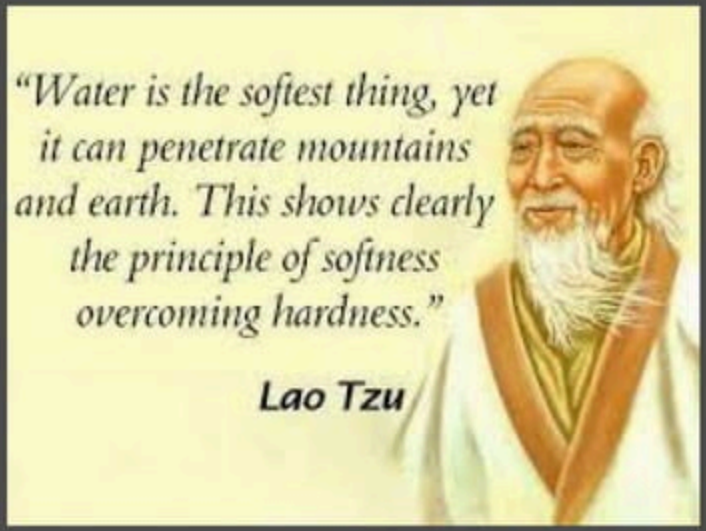
Objects as function arguments:

```
void time::sum(time t1,time t2)
{
      minutes = t1.minutes + t2.minutes;
      hours = minutes / 60;
      minutes = minutes % 60;
      hours = hours + t1.hours + t2.hours;
}


int main( )
{
      time t1,t2,t3;
      t1.gettime(2,45);
      t2.gettime(3,30);
      t3.sum(t1,t2);

      cout <<"T1 = " << t1.puttime();
      cout <<"T2 = " << t2.puttime();
      cout <<"T3 = " << t3.puttime();

      return 0;
}
```



To be continued.....