First Year First Semester Course
M.Tech. (CS) [Batch 2021-23]

Lecture #13

Introduction to Programming

Classes and Objects in C++

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

Structure and Class

- C structures do not permit data hiding
- C and C++ structure members are *by default public* members
- C++ structure can have functions as member
- C++ structure can explicitly make some of the members private
- C++ structure names are stand alone

- *By default* the members of a class are *private*

Class Specification

- Class specification has two parts:
  - Class declaration
  - Class function definitions

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

Data members

Member functions

```
class item
{
        int number;
        float cost;

    public:
        void getdata(int a, float b);
        void putdata(void);
};
```

Defining Member Functions

- Can be defined in two places
  - Outside the class definition
  - Inside the class definition

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

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

```
void item :: putdata(void)
{
   cout<< "Number: "<< number <<"\n";
   cout<< "Cost: "<< cost <<"\n";
}
```

Defining Member Functions [contd.]

- Inside the class definition
  - Treated as inline function

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

        }
};
```

Defining Member Functions [contd.]

- Making outside function inline

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

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

Creating Objects

- Once a class has been declared, we can create variables of that type by using class name (like any built-in type variable)

```
item x;
```

- In C++, class variables are known as *objects*
- **X** is *object* of type **item**
- Objects can also be created when a class is declared.

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

```
class employee
{
  private:
    char name[20];
    int age,sal;
  public:
    void getdata();
    void putdata();
};
```

Array of objects → employee emp[5];

Accessing Members

- Private data of a class can be accessed only through the member functions of that class.

```
object-name.function-name(actual-arguments);
```

```
class item
{
        int number;
        float cost;

   public:
        void getdata(int a, float b);
        void putdata(void);
};
```

```
item x;

x.getdata(5,10.5);
x.putdata();

x.number = 100;     //illegal
```

C++ Program with Class


```
#include<iostream>
using namespace std;
class item
{
          int number;
          float cost;
   public:
          void getdata(int a, float b);
          void putdata(void)
          {
                 cout<< "Number: "<< number <<"\n";
                 cout<< "Cost: "<< cost <<"\n";

          }
};

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

C++ Program with Class [contd.]


```
int main()
{
        item x;

        cout<< "\n object x" << "\n";

        x.getdata(25,10.5);
        x.putdata();

        item y;

        cout<< "\n object x" << "\n";

        y.getdata(100,12.5);
        y.putdata();

        return 0;
}
```

Nesting of Member Function

- A member function can be called by using its name inside another member function of the same class.
- This is known as nesting of member functions.

```
int main(){
    set A;
    A.input( );
    A.display( );
    return 0;
}
```

```
#include <iostream>
class set {
    int m,n;
    public:
        void input(void);
        void display (void);
        void largest(void);
};
int set::largest (void) {
        if(m>n) return m;
        else return n;
}
```

```
void set::input(void){
    cout<<"Input values of m and n:";
    cin>>m>>n;
}
void set::display(void) {
    cout<<"The largest values is:"
    <<largest()<<"\n";
}
```

Private member functions

- A private member function can only be called by another function that is a member of its class.
- Even an object cannot invoke a private function using the dot operator

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

sample s1;
s1.read(); //illegal

```
void sample :: update(void)
{
        read();
}
```

Static Data Members

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

```
#include<iostream.h>
class item
{
    static int count; //count is static
    int number;
  public:
    void getdata(int a){
       number=a;
       count++;
    }
    void getcount(void)
    {
       cout<<"count:";
       cout<<count<<"\n";
    }
};
int item :: count ;
```

Static Data Members


```
#include<iostream.h>
class item
{
    static int count; //count is static
    int number;
  public:
    void getdata(int a){
       number=a;
       count++;
    }
    void getcount(void)
    {
       cout<<”count: ”;
       cout<<count<<“\n”;
    }
};
int item :: count;
```

```
int main()
{
   item a,b,c;
   a.getcount();
   b.getcount();
   c.getcount();

   a.getdata(100);
   b.getdata(200);
   c.getdata(300);
   cout<<"after reading data : \n";

   a.getcount();
   b.getcount();
   c.getcount();

   return(0);
}
```

Static Member Functions

- A static function can have access to only other static members declared in the same class.

- A static member function can be called using the class name as follows:-

class - name :: function - name;

```
include<iostream.h>
class test
{
    int code;
    static int count; // static member variable
  public:
    void set(void)
    {
        code=++count;
    }
    void showcode(void)
    {
        cout<<"object member : "<<code<<end;
    }
    static void showcount(void)
    {
        cout<<"count="<<count<<endl;
    }
};
```

Static Member Functions


```
include<iostream.h>
class test
{
    int code;
    static int count; // static member variable
  public:
    void setcode(void){
       code=++count;
    }
    void showcode(void){
       cout<<"object member : "<<code<<"\n";
    }
    static void showcount(void){
       cout<<"count="<<count<<"\n";
    }
};
```

```
int test:: count;

int main()
{
    test t1,t2;
    t1.setcode( );
    t2.setcode( );
    test :: showcount( );
    test t3;
    t3.setcode( );
    test:: showcount( );
    t1.showcode( );
    t2.showcode( );
    t3.showcode( );
    return(0);
}
```

Objects As Function Arguments


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

```
void time :: sum (time t1,time t2)
{
   minutes=t1.minutes + t2.minutes;
   hours=minutes%60;
   minutes=minutes%60;
   hours=hours+t 1.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);
}
```

Friendly Functions

- It is not in the scope of the class to which it has been declared as friend.
- It cannot be called using the object of that class.
- It can be invoked like a member function without the help of any object
- Unlike member functions it cannot access the member names directly.
- Needs to use object name and dot operator.

```
#include<iostream>
class sample
{
   int a;
   int b;
 public:
   void setvalue(){a=25;b=40;}
   friend float mean( sample s);
}
```

```
float mean (sample s)
{
return (float(s.a+s.b)/2.0);
}
int main ( )
 {
    sample x;
    x.setvalue( );
    cout<<"mean value="<<mean(x)<<"\n";
    return(0);
}
```

A function friendly to two classes


```
#include<iostream.h>
class abc;
class xyz
{
   int x;
  public:
   void setvalue(int i) { x = i; }
   friend void max (xyz,abc);
};
class abc
{
   int a;
 public:
   void setvalue( int i) {a=i;}
   friend void max(xyz,abc);
};
```

```
void max( xyz m, abc n)
{
    if(m.x >= n.a)
       cout<< m.x;
    else
       cout<< n.a;
}
int main( )
{
   abc j;
   j . setvalue( 10);
   xyz s;
   s.setvalue(20);
   max( s , j );
   return(0);
}
```

Returning Objects


```
#include<iostream>
using namespace std;
class complex
{
                    float x;
                    float y;
          public:
                    void input( float real , float imag){
                              x=real;
                              y=imag;
                    }
                    friend complex sum( complex , complex);
                    void show ( complex );
};
complex sum ( complex c1, complex c2){
          complex c3;
          c3.x=c1.x+c2.x;
          c3.y=c1.y+c2.y;
          return c3;
}
```

Returning Objects


```
#include<iostream>
using namespace std;
class complex
{
          float x;
          float y;
     public:
          void input( float real , float imag){
                    x=real;
                    y=imag;
          }
          friend complex sum( complex , complex);
          void show ( complex );
};
complex sum ( complex c1, complex c2){
          complex c3;
          c3.x=c1.x+c2.x;
          c3.y=c1.y+c2.y;
          return c3;
}
```

```
void complex :: show ( complex c)
{
          cout<<c.x<<" + i"<<c.y<<"\n";
}
int main( )
{
          complex a, b,c;
          a.input(3.1,5.65);
          b.input(2.75,1.2);
          c=sum(a,b);
          cout <<" a= "; a.show(a);
          cout <<" b= "; b.show(b);
          cout <<" c= " ; c.show(c);
          return(0);
}
```

Local Class

- Defined inside a function or a block
- Can use global variables and static variables declared inside the function
- Cannot have static data members
- Member functions must be defined inside the local class

Do it now

- Use friend function to swap the private data of two classes

```
#include<iostream>
using namespace std;
class class2;
class class1{
              int value1;
          public:
              void indata( int a) { value1=a; }
              void display(void) { cout<<value1<<endl; }
              friend void exchange ( class1 &, class2 &);
};
class class2{
              int value2;
          public:
              void indata( int a) { value2=a; }
              void display(void) { cout<<value2<<"\n"; }
              friend void exchange(class1 & , class2 &);
};
```

Do it now [contd.]


```
void exchange ( class1 &x, class2 &y)
{
          int temp=x. value1;
          x. value1=y.value2;
          y.value2=temp;
}
int main( )
{
          class1 c1;
          class2 c2;
          c1.indata(100);
          c2.indata(200);
          cout<<"values before exchange:"<<"\n";
          c1.display( );
          c2.display( );
          exchange (c1,c2);
          cout<<"values after exchange :"<<"\n";
          c1. display ( );
          c2. display ( );
          return(0);
}
```

Questions?