DEPARTMENT OF COMMERCE (CA)
OBJECT ORIENTED PROGRAMMING WITH C++ (SEMESTER-IV)
II-B.COM (CA) SUBJECT CODE - 18BCA42C
UNIT III

Classes and objects – introduction – specifying a class – defining Member function – nesting of member functions – private member functions – Arrays within a class – static data members – array of objects as function Arguments- friend functions.

Class:

A class in C++ is the building block, that leads to Object-Oriented programming. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A C++ class is like a blueprint for an object.

For Example: Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

- A Class is a user defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
- In the above example of class Car, the data member will be speed limit, mileage etc and member functions can be apply brakes, increase speed etc.

Object:

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Defining Class and Declaring Objects:

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.classes-and-objects-in-C.

Declaring Objects:

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

ClassName ObjectName;

Accessing data members and member functions: The data members and member functions of class can be accessed using the dot('.') operator with the object. For example if the name of

object is obj and you want to access the member function with the name printName() then you will have to write obj.printName() .

Accessing Data Members

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by Access modifiers in C++. There are three access modifiers : public, private and protected.

```
// C++ program to demonstrate
// accessing of data members
#include <bits/stdc++.h>
using namespace std;
class Geeks
{
   // Access specifier
   public:
   // Data Members
   string geekname;
   // Member Functions()
   void printname()
   {
      cout << "Geekname is: " << geekname;
   }
};

int main() {

   // Declare an object of class geeks
   Geeks obj1;

   // accessing data member
   obj1.geekname = "Abhi";

   // accessing member function
   obj1.printname();
   return 0;
}
```

Output:

```
Geekname is: Abhi
```

Member Functions in Classes:

There are 2 ways to define a member function:

Inside class definition

Outside class definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.

```
// C++ program to demonstrate function
// declaration outside class

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
   public:
   string geekname;
   int id;

   // printname is not defined inside class definition
   void printname();

   // printid is defined inside class definition
   void printid()
   {
      cout << "Geek id is: " << id;
   }
};

// Definition of printname using scope resolution operator ::
void Geeks::printname()
{
   cout << "Geekname is: " << geekname;
}
int main() {

   Geeks obj1;
   obj1.geekname = "xyz";
   obj1.id=15;

   // call printname()
   obj1.printname();
   cout << endl;

   // call printid()
```

```
    obj1.printid();
    return 0;
}
```

Output:

```
Geekname is: xyz
Geek id is: 15
```

Note that all the member functions defined inside the class definition are by default inline, but you can also make any non-class function inline by using keyword inline with them. Inline functions are actual functions, which are copied everywhere during compilation, like pre-processor macro, so the overhead of function calling is reduced.

Note: Declaring a friend function is a way to give private access to a non-member function.

Nesting Of Member Functions

A member function of a class can be called only by an object of that class using a dot operator. However, there is an exception to this. 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.

Nesting of Member Function

```
#include
using namespace std;
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 value of m and n"<<"\n";
cin >> m>>n;
}
```

```
void set :: display(void)
{
cout << "largest value=" << largest() <<"\n";
}

int main()
{
set A;
A.input();
A.display();

return 0;
}
```

The output of program would be:

```
Input value of m and n
25 18
Largest value=25
```

The private Members Functions:

A **private** member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.

By default all the members of a class would be private, for example in the following class **width** is a private member, which means until you label a member, it will be assumed a private member −

```
class Box {
   double width;

   public:
      double length;
      void setWidth( double wid );
      double getWidth( void );
};
```

Practically, we define data in private section and related functions in public section so that they can be called from outside of the class as shown in the following program.

```
#include <iostream>

using namespace std;

class Box {
   public:
      double length;
      void setWidth( double wid );
```

```
    double getWidth( void );

  private:
    double width;
};

// Member functions definitions
double Box::getWidth(void) {
  return width ;
}

void Box::setWidth( double wid ) {
  width = wid;
}

// Main function for the program
int main() {
  Box box;

  // set box length without member function
  box.length = 10.0; // OK: because length is public
  cout << "Length of box : " << box.length <<endl;

  // set box width without member function
  // box.width = 10.0; // Error: because width is private
  box.setWidth(10.0);  // Use member function to set it.
  cout << "Width of box : " << box.getWidth() <<endl;

  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Length of box : 10
Width of box : 10
```

Arrays within a Class

- Arrays can be declared as the members of a class.
- The arrays can be declared as private, public or protected members of the class.
- To understand the concept of arrays as members of a class, consider this example.

A program to demonstrate the concept of arrays as class members

Example:

```
#include<iostream>
const int size=5;
class student
{
```

```
int roll_no;
int marks[size];
public:
void getdata ();
void tot_marks ();
} ;

void student :: getdata ()
{
cout<<"\nEnter roll no: ";
Cin>>roll_no;
for(int i=0; i<size; i++)
{
cout<<"Enter marks in subject"<<(i+1)<<": ";
cin>>marks[i] ;
}

void student :: tot_marks() //calculating total marks
{
int total=0;
for(int i=0; i<size; i++)
total+ = marks[i];
cout<<"\n\nTotal marks "<<total;
}

void main()
{
student stu;
stu.getdata() ;
stu.tot_marks() ;
getch();
}
```

Output:

Enter roll no: 101
Enter marks in subject 1: 67
Enter marks in subject 2 : 54
Enter marks in subject 3 : 68
Enter marks in subject 4 : 72
Enter marks in subject 5 : 82
Total marks = 343

Static Data Members:

Static data members are class members that are declared using the static keyword. There is only one copy of the static data member in the class, even if there are many class objects. This is because all the objects share the static data member. The static data member is always initialized to zero when the first class object is created.
The syntax of the static data members is given as follows −
static data_type data_member_name;
In the above syntax, static keyword is used. The data_type is the C++ data type such as int, float etc. The data_member_name is the name provided to the data member.

Array of objects:

We can easily create array of objects of type class.
Example :

```
Class student
{
        int rollno,marks[5];
        char name [30];
        public:
                void getdata( );
                void putdata( );
};

        -------
        -------
        -------
student s[10];          //  array of students
```

The array Student contains 10 students(objects) namely s[0],s[1],s[2],….s[9].

- Array element s[i] can be used like a ordinary array variable.
- s[i].getdata( ) indicates the object s[i] calls the member function getdata( ).
- Example: student average mark calculation using array of objects.

Objects as function arguments:

- An object may be used as a function argument.
- A copy of the entire object may be passed to the function (pass –by-value) or
- We can pass only the address of the object to the function (Pass-by reference)
- In pass-by-value, a copy of object is sent to the function, any changes to the object inside the function do not affect the object used.
- In pass-by-reference, an address of the object is passed, the function operates directly on the actual object used, if the function does any change, that directly affects the object.

Friend functions:

A friend is a special mechanism for allowing non-member functions to access private data values.

- A friend fuction need not be a member of any of classes.

- Declaration : Class class_name
```
{
        ---
        ----
public:
----
----
friend return_type   friend_fn_name (arguments);
----
----
};
```
- The function declaration should be prefixed by the keyword friend.
- The function can be defined anywhere in the program like a ordinary function.
- The function definition does not have either the keyword friend or the scope resolution operator : :

Example:

```
//…… C++ program for finding mean of two values using friend function ….
# include <iostream.h>
class sample
{
    int a,b;
    public:
        void getvalue( );
        friend  float  mean ( sample  s);
};
void sample : : getvalue ( )
{
        cout <<"Enter two values "<< endl;
        cin >> a;
        cin >> b;
}
//….  Defining a friend function …..
float mean (sample s)
{
    return  float(s.a+s.b)/2.0
}

//  …  main function …..
int main ( )
{
    sample x;
    x. getvalue( );
    cout <<"Mean value  =" <<mean (x)<< endl;
    return 0;
}
```

Rules for friend function:

A friend function can access the private data.

- A friend function is invoked like regular function
- A friend function is not the scope of the class when it is declared as friend, because the keyword friend informs the compiler that it is not a member function of that class.
- A friend function usually has objects as arguments.
- A friend function has to use an object name and dot operator with each member name

so it can not access member names directly like other member functions.

REFERENCE:

1. E. Balaguruswamy , " Object oriented programming with C++", TataMcGraw Hill publishing company Limited, 1998.
2. K.R, Venugopal, Rajkumar, T. Ravishankar, "Mastering C++", tata mc graw – Hill publishing company Limited, 1998.
3. D. Ravichandran, Programming with C++", Tata McGraw – Hill published Company Limited.

Prepared By Dr.N.Shanmugavadivu