EXPERIMENT NO. 05

DATE OF PERFORMANCE: **GRADE:**

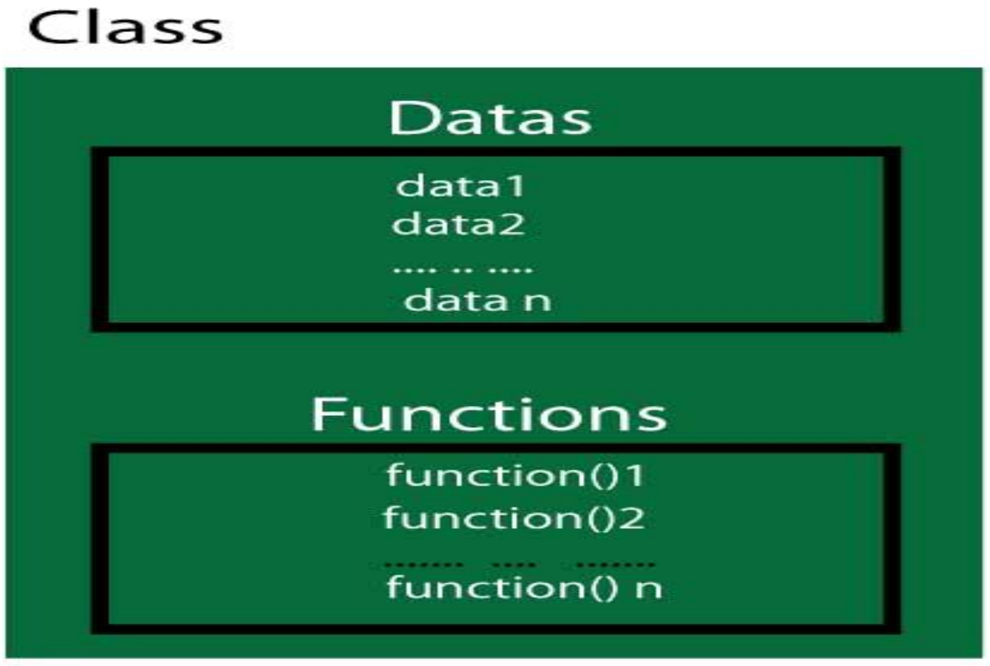
DATE OF ASSESSMENT: **SIGNATURE OF LECTURER/ TTA:**

AIM: To Study Classes and Objects in C++.

THEORY:

C++ Class:

A class is the collection of related data and function under a single name. A C++ program can have any number of classes. When related data and functions are kept under a class, it helps to visualize the complex problem efficiently and effectively.



A Class is a blueprint for objects. When a class is defined, no memory is allocated. You can imagine like a datatype.

```
int var;
```

The above code specifies *var* is a variable of type integer; int is used for specifying variable *var* is of integer type. Similarly, class are also just the specification for objects and object bears the property of that class.

Defining the Class in C++

Class is defined in C++ programming using keyword class followed by identifier(name of class). Body of class is defined inside curly brackets an terminated by semicolon at the end in similar way as structure.

```
class class_name
  {
  // some data
  // some functions
  };
```

Example of Class in C++

```
class temp
  {
    private:
      int data1;
      float data2;
    public:
      void func1()
        {
          data1=2;

           }

         float func2()

         {

             data2=3.5;

             retrun data;

          }

   };
```

Explanation:

Keywords: private and public

Keyword private makes data and functions private and keyword public makes data and functions public. Private data and functions are accessible inside that class only whereas, public data and functions are accessible both inside and outside the class. This feature in OOP is known as data hiding. If programmer mistakenly tries to access private data outside the class, compiler shows error which prevents the misuse of data.

Data member and Member functions:

The data within the class is known as data member. The function defined within the class is known as member function. These two technical terms are frequently used in explaining OOP. In the above class *temp, data1* and *data2* are data members and *func1()* and *func2()* are member functions.

Accessing Data Members and Member functions:

Data members and member functions can be accessed in similar way the member of structure is accessed using member operator(.). For the class and object defined above, *func1()* for object *obj2*can be called using code:

obj2.func1();

Similary, the data member can be accessed as:

object_name.data_memeber;

C++ Objects:

When class is defined, only specification for the object is defined. Object has same relationship to class as variable has with the data type. Objects can be defined in similary way as structure is defined.

Syntax to Define Object in C++

class_name variable name;

For the above defined class *temp*, objects for that class can be defined as:

Temp t1,t2;

Defining Member Function Outside the Class:

A large program may contain many member functions. For the clarity of the code, member

functions can be defined outside the class. To do so, member function should be declared inside the class(function prototype should be inside the class). Then, the function definition can be defined using scope resolution operator ::. Learn more about defining member function outside the class.

```
double Box::getVolume(void)

{

   return length * breadth * height;

}
```

Static Keyword:

Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in static storage area. And they have a scope till the program lifetime. Static Keyword can be used with following,
Static variable in functions

Static Class Objects

Static member Variable in class

Static Methods in class

Static data member in class:

Static data members of class are those members which are shared by all the objects. Static data member has a single piece of storage, and is not available as separate copy with each object, like other non-static data members.

Static member variables (data members) are not initialied using constructor, because these are not dependent on object initialization.

Also, it must be initialized explicitly, always outside the class. If not initialized, Linker will give error.

```
class X
{
 static int i;
 public:
 X(){};
};

int X::i=1;

int main()
{
 X obj;
 cout << obj.i;   // prints value of i
}
```

Once the definition for static data member is made, user cannot redefine it. Though, arithmetic operations can be performed on it.

Static Member Functions:

These functions work for the class as whole rather than for a particular object of a class.

It can be called using an object and the direct member access . operator. But, its more typical to call a static member function by itself, using class name and scope resolution :: operator.

Example :

```
class X
{
 public:
 static void f(){};
};

int main()
{
 X::f();   // calling member function directly with class name
}
```

These functions cannot access ordinary data members and member functions, but only static data members and static member functions.

It doesn't have any "this" keyword which is the reason it cannot access ordinary members. We will study about "this" keyword later.

C++ Friend Functions:

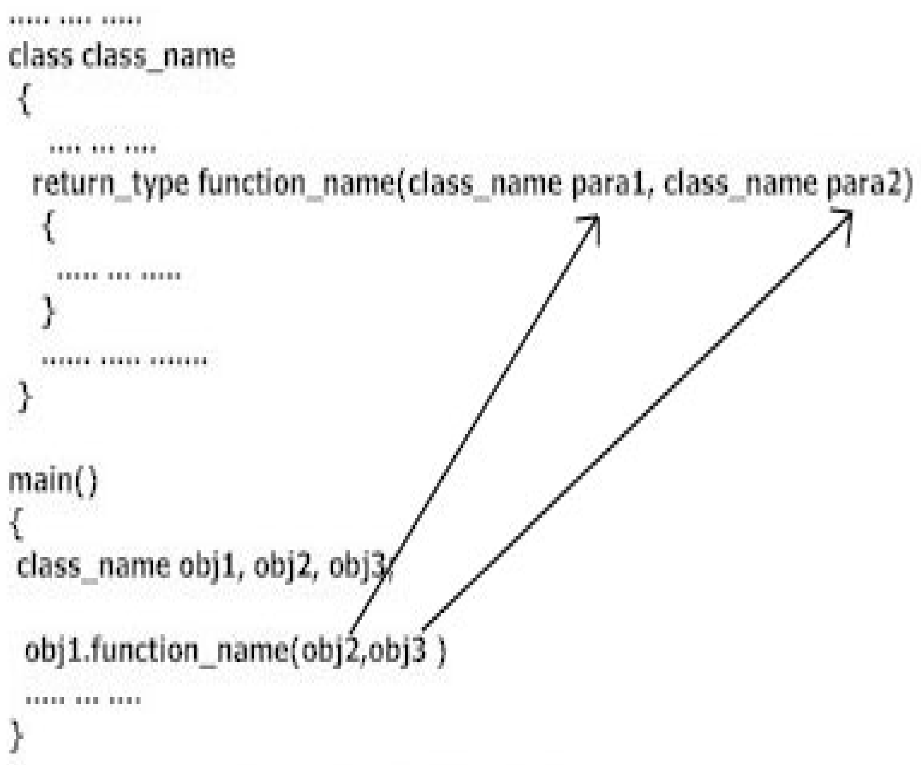
A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.

Passing and Returning Object from Function in C++:

In C++ programming, objects can be passed to function in similar way as variables and structures.

Procedure to Pass Object to Function:

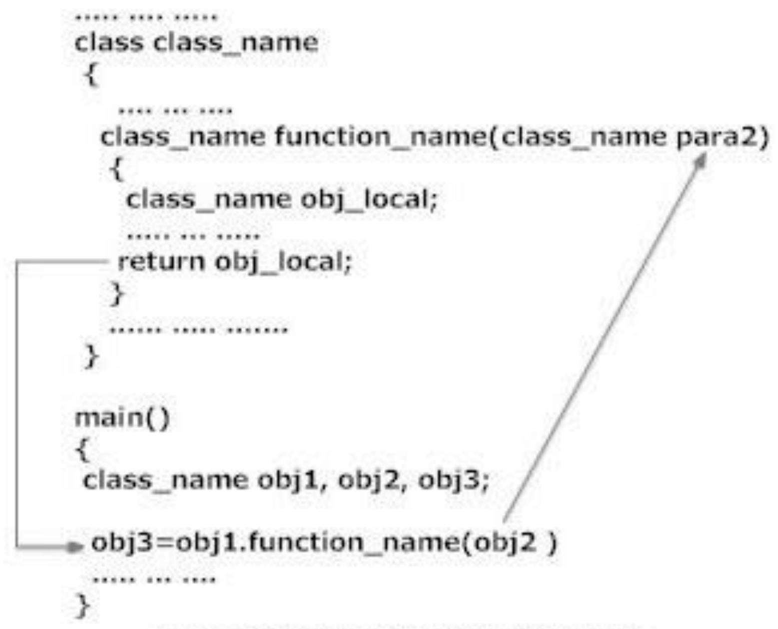


```
..... .... .....
class class_name
 {
    .... ... ....
   return_type function_name(class_name para1, class_name para2)
    {
     ..... ... .....
    }
   ...... ..... .......
 }

main()
{
 class_name obj1, obj2, obj3;

  obj1.function_name(obj2,obj3 )
  ..... ... ....
}
```

Figure: Passing Object to Function

Returning Object from Function:

The syntax and procedure to return object is similar to that of returning structure from function.



```
...... .... ......
class class_name
 {
   .... ... ....
  class_name function_name(class_name para2)
   {
    class_name obj_local;
    ...... ... ......
   return obj_local;
   }
  ....... ..... ........
 }

main()
{
 class_name obj1, obj2, obj3;

 obj3=obj1.function_name(obj2 )
 ...... ... ....
}
```

Figure: Returning Object from Function

PROGRAM 1: Find Volume of a BOX.

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

class Box
{
```

```
public:
   double length;   // Length of a box
   double breadth;  // Breadth of a box
   double height;   // Height of a box

   // Member functions declaration
   double getVolume(void);
   void setLength(double len);
   void setBreadth(double bre);
   void setHeight(double hei);
};

// Member functions definitions
double Box::getVolume(void)
{
   return length * breadth * height;
}

void Box::setLength(double len)
{
   length = len;
}

void Box::setBreadth(double bre)
{
   breadth = bre;
}

void Box::setHeight(double hei)
{
   height = hei;
}

// Main function for the program
int main()
{
   Box Box1;            // Declare Box1 of type Box
   Box Box2;            // Declare Box2 of type Box
   double volume = 0.0;  // Store the volume of a box here
   // Box 1 specification
   Box1.setLength(6.0);
   Box1.setBreadth(7.0);
   Box1.setHeight(5.0);

   // Box 2 specification
   Box2.setLength(12.0);
   Box2.setBreadth(13.0);
   Box2.setHeight(10.0);

   // Volume of Box 1
   volume = Box1.getVolume();
   cout << "Volume of Box1: " << volume << endl;

   // Volume of Box 2
   volume = Box2.getVolume();
   cout << "Volume of Box2: " << volume << endl;

   return 0;
}
```

OUTPUT:

```
Volume of Box1: 210
Volume of Box2: 1560
```

PROGRAM 2: Use of static Function.

```
#include <iostream>

using namespace std;



class Box

{

public:

   static int objectCount;
   // Constructor definition
   Box(double l = 2.0, double b = 2.0, double h = 2.0)
   {
      cout << "Constructor called." << endl;
      length = l;
      breadth = b;
      height = h;
      // Increase every time object is created
      objectCount++;
   }
   double Volume()
   {
      return length * breadth * height;
   }
   static int getCount()
   {
      return objectCount;
   }

private:
   double length;   // Length of a box
   double breadth;  // Breadth of a box
   double height;   // Height of a box
};
```

```
// Initialize static member of class Box

int Box::objectCount = 0;


int main(void)

{

   // Print total number of objects before creating objects

   cout << "Initial Stage Count: " << Box::getCount() << endl;


   Box Box1(3.3, 1.2, 1.5); // Declare Box1

   Box Box2(8.5, 6.0, 2.0); // Declare Box2


   // Print total number of objects after creating objects

   cout << "Final Stage Count: " << Box::getCount() << endl;

   return 0;

}
```

OUTPUT:

```
Initial Stage Count: 0

Constructor called.

Constructor called.

Final Stage Count: 2
```

PROGRAM 3: Use of friend Function.

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

```
class Box
{
   double width;

public:
   friend void printWidth(Box box);
   void setWidth(double wid);
};

// Member function definition
void Box::setWidth(double wid)
{
   width = wid;
}

// Note: printWidth() is not a member function of any class.
void printWidth(Box box)
{
   // Because printWidth() is a friend of Box, it can directly access any member of this class
   cout << "Width of box: " << box.width << endl;
}

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

   // Set box width without member function
   box.setWidth(10.0);

   // Use friend function to print the width.
   printWidth(box);

   return 0;
}
```

OUTPUT:

```
Width of box: 10
```

PROGRAM 4: Pass Object to Function.

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

class Complex
{
private:
   int real;
   int imag;

public:
   Complex() : real(0), imag(0) { }
   void Read()
   {
      cout << "Enter real and imaginary numbers respectively:" << endl;
      cin >> real >> imag;
   }
   void Add(Complex comp1, Complex comp2)
   {
      real = comp1.real + comp2.real;
      imag = comp1.imag + comp2.imag;
   }
   void Display()
   {
      cout << "Sum = " << real << "+" << imag << "i";
   }
};

int main()
{
   Complex c1, c2, c3;
   c1.Read();
   c2.Read();
   c3.Add(c1, c2);
   c3.Display();
   return 0;
}
```

OUTPUT:

```
Enter real and imaginary numbers respectively:
2 3
Enter real and imaginary numbers respectively:
4 5
Sum = 6+8i
```

PROGRAM 5: Return Object from Function.

```
#include <iostream>

using namespace std;

class Complex
{
private:
   int real;
   int imag;

public:
   Complex() : real(0), imag(0) {}

   void Read()
   {
       cout << "Enter real and imaginary numbers respectively:" << endl;
       cin >> real >> imag;
   }

   Complex Add(Complex comp2)
   {
       Complex temp;
       temp.real = real + comp2.real;
       temp.imag = imag + comp2.imag;
       return temp;
   }

   void Display()
   {
       cout << "Sum = " << real << "+" << imag << "i" << endl;
   }
};

int main()
{
   Complex c1, c2, c3;
   c1.Read();
   c2.Read();
   c3 = c1.Add(c2);
   c3.Display();

   return 0;
}
```

OUTPUT:

```
Enter real and imaginary numbers respectively:
2 3
Enter real and imaginary numbers respectively:
-1 4
Sum = 1+7i
```