3.3 C++ LANGUAGE CONSTRUCTS WITH OBJECTS AND CLASSES

1. NAMESPACE
2. FUNCTION OVERLOADING
3. INLINE FUNCTION
4. DEFAULT ARGUMENT
5. PASS/RETURN BY REFERENCE
6. INTRODUCTION TO CLASS AND OBJECT
7. ACCESS SPECIFIERS
8. OBJECTS AND THE MEMBER ACCESS
9. DEFINING MEMBER FUNCTION
10. CONSTURCTOR AND ITS TYPES, DESTRUCTOR
11. DYNAMIC MEMORY ALLOCATION FOR OBJECTS AND OBJECT ARRAY
12. this POINTER
13. STATIC DATA MEMBER AND STATIC FUNCTION
14. CONSTANT MEMBER FUNCTIONS AND CONSTANT OBJECTS
15. FRIEND FUNCTION AND FRIEND CLASSES

NAMESPACE

The namespace is used for the logical grouping of program elements like variables, classes, functions, etc. If some program elements are related to each other, they can be put into a single namespace. The namespace helps to localize the name of identifiers so that there is no naming conflict across different modules designed by different members of programming team.

Syntax for defining namespace is as follows:

namespace namespace_name {

//declaration of variables, functions,classes etc

} // no semicolon at the end

Example:

```
namespace myNamespace
{
int a, b;
void name();
}
```

NAMESPACE

Example:

```
namespace myNamespace
{
 int a, b;
 void name();
}
```

In this case, a and b are normal variables and name() is a user-defined function declared within a namespace called myNamespace. In order to access these variables and user-defined functions from outside the myNamespace namespace, we have to use the scope resolution operator (: :).For example, to access the previous variables and function from outside myNamespace we can write:

1. myNamespace::a
2. myNamespace::b
3. void myNamespace::name()

NAMESPACE

The functionality of namespaces is especially useful in the case that there is a possibility that a global object or function uses the same identifier as another one, causing ambiguity (redefinition) errors. For example:

```
// namespaces
#include <iostream>
using namespace std;
namespace first
{
int var = 5;
}
namespace second
{
double var = 3.1416;
}
int main () {
cout<< first::var<< endl;
cout << second::var << endl;
return 0;
}
```

Output:

```
5
3.1416
```

In this case, there are two global variables with the same name: var. One is defined within the namespace first and the other one in second. No redefinition errors happen thanks to namespaces.

NAMESPACE

Example:

```
#include<iostream>
using  namespace  std;
namespace  Rectangle
{
int width;
int height;
void area();
}
```

```
int main()  {
Rectangle::width = 10;
Rectangle::height = 2;
Rectangle::area();
return 0;
}
void Rectangle::area()
{ c
out << "\nThe Area is: " << width*height << endl;
}
```

Output: The Area is: 20

Multiple Choice questions on Namespace

1. What is the use of Namespace?

a) To encapsulate the data
b) To structure a program into logical units
c) Encapsulate the data & structure a program into logical units
d) It is used to mark the beginning of the program

2. What is the general syntax for accessing the namespace variable?

a) namespace,operator
b) namespace::operator
c) namespace#operator
d) namespace$operator

3. Which keyword is used to access the variable in the namespace?

a) dynamic
b) Const
c) using
d) static

4. What is namespace in C++?

a)It is a block of code that is used to group related variables and functions
b) It is a feature that allows a class to inherit from multiple base classes
c) It is a type of loop in C++
d) It is a mechanism for managing the scope of identifiers in large programs

5. What will be the output of the following C++ code?

```
#include <iostream>
   using namespace std;
   namespace first
   {
      int var = 5;
   }
   namespace second
   {
      double var = 3.1416;
   }
   int main ()
   {
      int a;
      a = first::var + second::var;
      cout << a;
      return 0;
  }
```

a) 8
b) 8.31416
c) 9
d) compile time error

6. What will be the output of the following C++ code?

```
   #include <iostream>
   using namespace std;
   namespace first{
      int x = 5;
      int y = 10;
}
   namespace second {
      double x = 3.1416;
      double y = 2.7183;
}
   int main (){
      using first::x;
      using second::y;
      bool a, b;
      a = x > y;
      b = first::y < second::x;
      cout << a << b;
      return 0;
   }
```

a) 11
b) 01
c) 00
d) 10

FUNCTIONS

Function Syntax

Function Declaration/Prototype:

```
return_type function_name(parameter list);
```

Function Definition:

```
return_type function_name(parameter list)
{
body_of _the_function
}
```

Function Call:

```
function_name(list of variables or values);
```

FUNCTIONS

Function Overloading

- C++ enables several functions of the same name to be defined as long as these functions have different sets of parameters. This capability is called function overloading/function polymorphism.
- The correct function to be invoked is determined by checking the number, types and order of the arguments in the call.

Function Overloading Example

```
#include <iostream>
using namespace std;
void sum(int a, int b)
{

    cout<<”Function 1: The sum is : ” << (a+b) << endl;
}
void sum(int a, double b, int c)
{
    cout<<”Function 2: The sum is : ” << double(a+b+c) << endl;
}
int main()
{
    sum(20, 30);
    sum(20, 30.10, 40);
    return 0;
}
```

OUTPUT:

```
Function 1 : The sum is : 50
Function 2 : The sum is : 90.1
```

FUNCTIONS

Inline Functions

- Inline Function is a function which expands to a line where it is invoked or called instead of jumping to the function itself. It is declared using the keyword ***inline*** in the function definition.

```
inline return_type function_name(list of arguments)
{
        //body of the function
}
```

- Inline functions increase the speed of execution of the program but it also increases the size of the program.
- So it is used only for small functions. A function cannot be made inline:
  - If static variable is declared inside the function
  - If a function returns a value where return type is specified
  - If function contains loop, switch or goto
  - If a function is recursive

FUNCTIONS

Inline Functions

- The inline keyword is just a request to the compiler to make a function inline function. The compiler may ignore the request if the function definition is too long or complicated and compile the function as a normal function.
- All the inline functions must be defined before they are called.

Note: Inline functions serve the same function as #define macro (generally used in C) but it provides better type checking and do not require special care for parentheses.

FUNCTIONS

Default Arguments

Default arguments are the default values provided to the arguments of the function. The default values are assigned during function declaration. The default value is used if the value is not passed during the function call. Otherwise, it uses the passed value.

Default arguments must be provided from the rightmost parameter in the argument list.

```
        float interest(float p; int time; int rate = 0.5); float
        interest(float p = 1000; int time = 1; int rate); //error
        float interest(float p; int time = 1; int rate);     //error
```

In the function call, any argument in a function cannot have a default value unless all arguments appearing on its right have their default values.

Default Arguments Example

```
#include <iostream>
using namespace std;
float interest(float p = 1000, int time = 1, int rate = 0.5)
{
 return (p * time * rate);
}
int main()
{
        cout<< “Interest =” << interest() <<endl;
        cout << “Interest =” << interest(1200) << endl;
        cout << “Interest =” << interest(1200, 10) << endl;
        cout << “Interest =” << interest(1500, 10, 0.7) << endl;
        return 0;
}
```

A function having N default arguments can be invoked in N+1 different ways as shown by the above example.

FUNCTIONS

Default Arguments

Note: Ambiguity arising when both function overloading and function with default arguments are used must be avoided.

Eg:

```
void function(int x, int y=0);
void function(int x);
```

Ambiguity arises when the function is called using only one integer as argument.

FUNCTIONS

Pass by Reference

Usually when a function is called, the arguments are copied into function and the function works on the copied value. Thus, the original value is not changed. If the original value must be changed then arguments must be passed by reference. It can be achieved in two ways:

- using reference variable
- using pointer

Passing by reference using reference variable vs using pointer

- using a reference variable saves memory as memory is not allocated for a pointer variable.
- using a reference variable is easier as no referencing / dereferencing is required like in pointers.
- reference variable references only a particular address it was first used to reference and hence works like a constant pointer. Using a pointer allows us to use the same pointer to point different addresses when required.

Pass By Reference Example

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

//pass by value
void swap1(int a, int b)
{
 int temp = b;
 b = a;
a = temp;
}

//pass by reference using reference variable
void swap2(int &a, int &b)
{
 int temp = b;
 b = a;
a = temp;
}
//pass by reference using pointer
void swap3(int *a, int *b)
{
int temp = *b;
*b = *a;
*a = temp;
}
int main()
{
 int a = 10, b = 20;
 swap1(a,b);
 cout << “a = ” << a << “, ” << “b = ” << b << endl;
 swap2(a,b);
 cout << “a = ” << a << “, ” << “b = ” << b << endl;
 swap3(&a,&b);
 cout << “a = ” << a << “, ” << “b = ” << b << endl;
 return 0;
}
```

Pass By Reference Example

OUTPUT

```
a = 10, b = 20
a = 20, b = 10
a = 10, b = 20
```

Note: only variables can be passed by reference. We cannot pass constants like swap(2,3). Compiler generates error message if constants are passed.

In the above example,

- swap1() uses pass by value, i.e., it copies the values to the arguments of function. So, ‘a’ and ‘b’ in the main() function and ‘a’ and ‘b’ inside swap1() are different variables. Thus, in main() ‘a’ remains 10 and ‘b’ remains 20.
- swap2() uses pass by reference using a reference variable. So swap2() uses alias names to reference the variables in main(). ‘a’ is referenced as ‘a’ and ‘b’ is referenced as ‘b’ (they can be referenced by different names of course). When ‘a’ and ‘b’ are swapped in swap2(),the change is reflected in main() too.
- swap3() uses pass by reference using pointers. swap3() uses pointers to directly access the address of the variables in main() and, hence, directly swaps the values stored in the two memory locations.

FUNCTIONS

Return By Reference

- In C++, a function can return a variable by reference.
- Return by reference means a function is returning an alias of the variable in the return statement so the variable which is being returned should have a scope where the function is being invoked.
- Normally, Global variables and a variable which is being passed by reference to the function have a scope where the function is being invoked.
- Return by reference allows value to be assigned to the variable returned i.e. the function call can be used on the left side of the assignment operator to assign any value.

***Note:** Scope of the variables should be carefully chosen. A local variable of a function cannot be returned by referenced and generates an error. This is because local variables has a scope within the function only and is destroyed when function execution is completed. Thus, returning by reference will try to reference a variable that does not exist causing the error.*

Return By Reference Example

```
#include <iostream>
using namespace std;
int &max(int&, int&);
int main()
{
 int a, b;
 cout << "Enter the numbers: " << endl;
cin >> a >> b;
cout << "The values are " << a << " and " << b <<
endl;
cout << "The maximum value is " << max(a,b)
<< endl;  max(a, b) = -1;
cout << "The values are " << a << " and " << b <<
endl;
return 0;
}
```

```
int &max(int& x, int& y)
{
return (x>y?x:y);
}
```

OUTPUT:

```
Enter the numbers: 34 45
The values are 34 and 45
The maximum value is 45
The values are 34 and -1
```

Multiple Choice questions on Functions in C++

1. An inline function is expanded during ___________

a) compile-time
b) run-time
c) never expanded
d) end of the program

2. When we define the default values for a function?

a) When a function is defined
b) When a function is declared
c) When the scope of the function is over
d) When a function is called

3. Where should default parameters appear in a function prototype?

a) To the leftmost side of the parameter list
b) Anywhere inside the parameter list
c) To the rightmost side of the parameter list
d) Middle of the parameter list

4. Which is of the following is an example of function overloading in C++?

a)int add(int a, int b)
b) float add(float a, float b)
c) void add(int a, int b)
d) All of the above

5. Which functions of a class are called inline functions?

a) All the functions containing declared inside the class
b) All functions defined inside or with the inline keyword
c) All the functions accessing static members of the class
d) All the functions that are defined outside the class

6. Which of the following is an example of function overloading?

a) int add(int x, int y) and
   float add(int x, int y, int z)
b) int add(int x, int y) and
   int add(int x, int y, int z)
c) int add(int x, int y) and
   int subtract(int x, int y, int z)
d) All of the above

7. What is the advantage of using an inline function in C++?

a) It allows for recursion
b) It increases the performance of the program
c) It reduces the function call overhead
d) It reduces size of the program

8. In which of the following cases inline functions may not word?

i) If the function has static variables.
ii) If the function has global and register variables.
iii) If the function contains loops
iv) If the function is recursive
a) i, iv
b) iii, iv
c) ii, iii, iv
d) i, iii, iv

9. Which is more effective while calling the functions?

a) call by value
b) call by reference
c) call by pointer
d) call by object

10. Which value will it take when both user and default values are given?

a) user value
b) default value
c) custom value
d) defined value

11. What will be the output of the following C++ code?

```
#include <iostream>
using namespace std;
int func(int m = 10, int n)    {
    int c;
    c = m + n;
    return c;
}
int main()    {
    cout << func(5);
    return 0;
}
```

a) 15
b) 10
c) compile time error
d) 30

12. What will be the output of the following C++ code?

```
    #include <iostream>
    using namespace std;
    void copy (int& a, int& b, int& c) {
        a *= 2;
        b *= 2;
        c *= 2;
}
    int main ()  {
        int x = 1, y = 3, z = 7;
        copy (x, y, z);
        cout << "x =" << x << ", y =" << y << ", z =" << z;
        return 0;
    }
```

a) 2 5 10
b) 2 4 5
c) 2 6 14
d) 2 4 9

13. When should we use pass by reference in C++?

a) When we need to modify the original variable in the calling function
b) When we don't want the function to modify the original variable in the calling function
c) When the variable is too large to copy
d) When we want to make the code harder to read

14. Which of the following is true about pass-by-reference in C++?

a) It can only be used with primitive data types
b) It is less efficient than pass-by-value
c) The original value of the passed variable is not affected by the function
d) The reference must be explicitly dereferenced within the function

15. What is the syntax for returning a reference in C++?

a) int& func();
b) int func&()
c) int func();
d) &int func();

16. How a reference is different from a pointer?

a) A reference cannot be null
b) A reference once established cannot be changed
c) The reference doesn't need an explicit dereferencing mechanism
d) All of the mentioned

C++ CLASSES

A Class is a group of similar objects and describes both characteristics(data members) and behavior(member functions) of the object. Classes are user defined data types that bind together data types and functions.

```
class class_name
{
//members of the class are defined here
};
```

Declaration of a class involves four attributes:
Tag name/Class name: the name by which the objects of the class are created
Data Members: the data types which makes up the class.
Member Functions/Methods: the function which operate on the data of the class.
Program Access Levels(private/public/protected): defines where the members of class can be used

C++ CLASSES

A general class construct is show below:

```
class class_name
{
private: //private data members and member
functions
public: //public data members and member
functions

protected: // protected data members and
member functions
};
```

ACCESS SPECIFIERS

Access modifiers are used to implement an important aspect of Object-Oriented Programming known as Data Hiding. Access Modifiers or Access Specifiers in a class are used to assign the accessibility to the class members, i.e., they set some restrictions on the class members so that they can’t be directly accessed by the outside functions.
There are 3 types of access modifiers available in C++:

1. **Public:** All the class members declared under the public specifier will be available to everyone. The data members and member functions declared as public can be accessed by other classes and functions too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

ACCESS SPECIFIERS

2. **Private:** The class members declared as private can be accessed only by the member functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of the class.

3. **Protected:** The protected access modifier is similar to the private access modifier in the sense that it can't be accessed outside of its class unless with the help of a friend class. The difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.

OBJECTS AND THE MEMBER ACCESS

After the declaration of a class, memory is not allocated for the data member of the class. The declaration of class develops a template but data members cannot be manipulated unless an object of its types is created. Objects are the instances of the class just as variables are instances of basic data types. An object is declared like a normal variable, but using the class name as data type.

__class_name o1, o2;__ // o1 and o2 are objects of class object

The data members and member functions(in public section) can be accessed just like in structures using the dot(.) operator.

Eg: *__o1.data;__ //accessing data member*

__o1.function();__ //accessing member function

Note: if a pointer of the class is created -> operator is used instead of '.'

DEFINING MEMBER FUNCTIONS

Member functions are declared inside the class but may be defined inside a class or outside the class.

Defining Member Function inside a class

A member function can be defined inside a class just like a normal function. A member function defined inside a class is automatically inline.

Eg:
```
class X
{
            int a;
      public:
             void input(){
                        cout<< “Enter value of a: ”;
                         cin >> a;
             }
}   ;
```

DEFINING MEMBER FUNCTIONS

Defining Member Function outside a class

A member function can be defined outside a class using class name and scope resolution operator (::) before the function name as shown in example below:

```
class X {
        int a;
        void input(); // function declaration inside class
};
void X::input(){
        cout <<"Enter value of 'a':";
        cin >> a;
}
```

The scope resolution operator is used to specify the class to which the member function belongs.

Multiple Choice questions on Classes in C++

1. What does a class in C++ holds?

a) data
b) functions
c) both data & functions
d) arrays

2. How many specifiers are present in access specifiers in class?

a) 1
b) 2
c) 3
d) 4

3. Which of the following is a valid class declaration?

a) class A { int x; };
b) class B { }
c) public class A { }
d) object A { int x; };

4. The data members and functions of a class in C++ are by default __________

a) protected
b) private
c) public
d) public & protected

5. Which operator a pointer object of a class uses to access its data members and member functions?

a) .
b) ->
c) :
d) ::

6. Which category of data type a class belongs to?

a) Fundamental data type
b) Derived data type
c) User defined derived data type
d) Atomic data type

7. What is the syntax for calling a member function of an object in C++?

a) object.member_function();
b) object->member_function();
c) Both A and B
d) None of the above

8. Which of the following is a valid syntax for defining a member function outside of a class in C++?

a) void MyClass::myFunction(){}
b) MyClass::void myFunction(){}
c) void myFunction()::MyClass{}
d) None of the above

9. What is the dot operator used for in C++?

a) To declare objects
b) To access a member of an object
c) To access a function of a class
d) To access a class variable

10. Which of the following keywords is used to create an object of a class?

a) New
b) classname
c) Object
d) none

11. Can a member function be overloaded in C++?

a) Yes
b) No
c) It depends on the access specifier
d) None of the above

CONSTRUCTORS

A constructor is a special member function that can be used for the necessary initialization of the data members of an object.
Some major points about constructors:

- The constructor has the same as its class.
- The constructor is automatically called at the time of object creation.
- The constructor doesn’t have any return type not even “void” but it can take arguments.
- The constructors are always declared/defined inside the public section.

```
class class_name{
 ....
 public:
      class_name()
      {
                //constructor body definition
      }  }  ;
```

Default Constructor

Default constructor is the constructor with no parameter.

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

class Example
{
 int a;
 public:
Example()
{  a = 10;  }
void display()
{
 cout << “The value of a is ” << a <<endl;
}
};
```

```
int main()
{
 Example e1;
 e1.display();
 return 0;
}
```

Output:

```
The value of a is 10
```

Parameterized constructor

The constructor function that can take arguments is called a parameterized constructor.

```
#include<iostream>
using namespace std;
class Example
{
    int a;
    Example();
    public:
        Example(int c){
            a = c;
        }
     void display()
     {
          Example e1;
          cout << a <<endl;
       }
};
```

```
Example:: Example()
{
 a = 0;
}
int main()
{
        Example e1; //since the default constructor is not
defined, it cannot be called
      Example e2(10);     //implicit call
      Example  e3  =  Example(5);  //explicit  call
      e2.display();
      e3.display();
      return 0;
}
```

Output:

```
10
5
```

Copy constructor

It is a member function. It has the same name as the class name. It is invoked automatically when the object of that class is created. This constructor has an argument of an object of the same type or same class as a reference. It is used for initializing an object of a class through another object of the same class.

```
#include<iostream>
using namespace std;
class Example
{
 int r, m;
 public:
Example(){
cout << “Default constructor” << endl;
r = m = 0 ;
}
Example(int x, int y){
cout << “Parameterized constructor” << endl;
r = x;  m = y;
}
Example(Example &T1)
{
cout << “Copy constructor” << endl;
r = T1.r + 1;  m = T1.m + 1;
}
void display()
{
cout << r << “ + ” << m << “i” << endl;
}
};
```

Copy constructor

```
int main() {


 Example e1;
 Example e2(1, 2);
Example e4(e1);      //implicit
 Example  e5  =  e2;  //explicit
e4.display();
e5.display();
e4 = e2;  // this does not invoke the copy constructor


 e4.display();


 return 0;
}
```

Output:

```
Default constructor
Parameterized  constructor
copy constructor
copy constructor
1 + 1i
2 + 3i
1 + 2i
```

Initialization List

C++ supports another method of initializing the class objects. This method is known as the initialization list in the constructor function.

```
constructor(argument_list) : initialization section{
        //body of constructor
}
```

```
#include<iostream>
using namespace std;
class complex
{
    int imag;
    int real;
   public:
   complex(int x, int y): real(x), imag(y){}
   void display()
   {  cout << real << " + i" << imag << endl;}
   };
```

```
int main()
{
 complex c(10,20);
 c.display();
return 0;
}
```

Output:

```
10 + i20
```

Initialization List

When using an initialization list to initialize objects, the members are initialized in the order in which they are declared rather than in the order in which they are placed in the Initialization list. Thus, using the data member declared later cannot be used to initialize the members coming earlier. For example in previous example,

complex(int x):real(imag),imag(x) {} *// valid*
complex(int x):real(x),imag(real) {} *//invalid*

Note: In the initialization list, members are given value before the constructor even starts to execute.

Initialization List

Initialization list is generally used to initialize constant and reference data members. Let us consider the following example

```
class ABC{
        const int x = 10;
        ...
}
;
```

AND

```
class ABC{
    const int x;
    int &y;
    public:
        ABC(int a, int &b){
            x = a;
            y = b;
}};
```

Both the above classes produce errors. Constants and Reference data members must be initialized when they are declared (their memory is allocated). The solution to this problem is an initialization list. They can be initialized in the initialization list as:

```
ABC(int a,int &b): x(a),y(b){}
```

DESTRUCTORS

It is a member function. Ii has same name as a class named preceded by tilda(~). It does not have any arguments and it does not have any return type(not even void). It is invoked automatically when the object of that class goes out of the scope or flushed from the memory.

```
#include<iostream>
using namespace std;
class demo
{
 int id;
 static int count;
public:
demo()
{
count++;
id = count;
cout << “\nID” << id << “ object created.”;
}
 ~demo()
  {
 cout << “\nID” << id << “ object destroyed.”;
  }
} ;

int demo::count=0;

int main()
{
 demo d1, d2;
 return 0;
}
```

Output:

```
ID1 object created.
ID2 object created.
ID2  object  destroyed.
ID1  object destroyed.
```

Multiple Choice questions on Constructor and Destructor

1. What is the role of a constructor in classes?

a) To modify the data whenever required
b) To destroy an object
c) To initialize the data members of an object when it is created
d) To call private functions from the outer world

6. What is a copy constructor?

a) A constructor that allows a user to move data from one object to another
b) A constructor to initialize an object with the values of another object
c) A constructor to check the whether to objects are equal or not
d) A constructor to kill other copies of a given object.

7. What happens if a user forgets to define a constructor inside a class?

a) Error occurs
b) Segmentation fault
c) Objects are not created properly
d) Compiler provides a default constructor to avoid faults/errors

8. How many parameters does a default constructor require?
a) 1
b) 2
c) 0
d) 3

5. How constructors are different from other member functions of the class?

a) Constructor has the same name as the class itself
b) Constructors do not return anything
c) Constructors are automatically called when an object is created
d) All of the mentioned

6. What is the role of destructors in Classes?

a) To modify the data whenever required
b) To destroy an object when the lifetime of an object ends
c) To initialize the data members of an object when it is created
d) To call private functions from the outer world

7. What is syntax of defining a destructor of class A?
a) A(){}
b) ~A(){}
c) A::A(){}
d) ~A(){};

8. What is the difference between constructors and destructors?

a) They have a different function name
b) Constructors does not have return type whereas destructors do have
c) Constructors allow function parameters whereas destructors do not
d) Constructors does not function parameters

9. Which of the following represents the correct explicit call to a constructor of class A?

```
class A{
        int a;
       public:
        A(int i)
      {
            a = i;
        }
   }
```

a) A a(5);
b) A a;
c) A a = A(5);
d) A a = A();

10. Can a constructor be defined in the private section in the class?

a) Yes
b) No
c) Depends on type of constructor
d) None of the above

11. How many destructors can a class have in C++?

a) 1
b) 2
c) 0
d) 3

OBJECT AS FUNCTION ARGUMENT AND RETURN TYPE

Objects can also be passed as function argument or be returned by a function like normal data type.

```
#include <iostream>
using namespace std;
class complex
{
    public:  int r, i;
    void input()
    {
    cout << "Enter real part: ";
    cin >> r;
    cout << "Enter imaginary part: ";
    cin >> i;
    }
    void display() {
    cout << r << "+i" << i << endl;
    }
    complex add(complex x, complex y) {
    complex t;
    t.r = x.r + y.r; t.i = x.i + y.i;
    return t;
    }
};
int main() {
    complex c1,c2,c3;
    c1.input();
    c2.input();
    c3 = add(c1,c2);
    cout << "The sum is: ";
    c3.display();
    return 0;
}
```

ARRAY OF OBJECTS

We know that an array can be of any data type including struct. Similarly, we can also have an array of variables that are of type class. Such variables are called array of objects.

classname arrayname[arraysize];

The members can be accessed using the following syntax:

arrayname[index].datamember; //for data member
arrayname[index].function(); //for member functions

Consider the following class definition:

```
class employee{
    char name[20];
    float age;
    public:
         void getdata();
         void putdata();
};
```

```
employee e[4]; // create array of objects of size 4
```

The member functions can be accessed as:

```
e[i].name;
e[i].putdata ();
```

POINTERS TO OBJECTS AND MEMBER ACCESS

Similar to pointer of other data type, we can also create pointer type of object of class. This pointer holds the address of an object of the class. The general form of declaring the pointer type of object is:

classname *Pointername;

Similar to the pointer type variable of structure, the pointer object to class uses the arrow operator(->) to access the members of the class. The general form is

Pointerobject->member

Or,

(*Pointerobject).member

POINTERS TO OBJECTS AND MEMBER ACCESS

For instance, let us consider a class named kantipur having input() and display() as its public member function, then.

```
kantipur k; //here k is a object of class kantipur
kantipur *p = &k; // p is a pointer that points to the object k;
```

Now, we can access input() and display() through pointer p as:

```
 p->input();
 p->display();
Or
(*p). input();
(*p).display();
```

DYNAMIC MEMORY ALLOCATION WITH NEW AND DELETE

Allocation of memory during runtime is known as dynamic memory allocation(DMA). C++ provides two operators for DMA new and delete.

New Operator:
C++ provides a new approach to obtaining blocks of memory using the new operator. This operator obtains memory from the operating system and returns a pointer to its starting point.

Delete Operator:
If our program reserves many chunks of memory using new, eventually all the available memory will be reserved and the system will crash. To ensure safe and efficient use of memory, the delete deallocates the memory pointed by the given pointer.

DYNAMIC MEMORY ALLOCATION WITH NEW AND DELETE

The general form of using new and delete is as follows:

datatype *pointervariable;

pointervariable = new datatype; //allocates single variable

pointervariable = new datatype[size]; //allocates an array of size elements

delete pointervariable; //if memory was allocated for a single variable

delete [ ] pointervariable; //alternatively when memory is allocated for an array

We can also initialize the the memory value using the new operator. This can be done as follows:

pointer_var = new data_type(value);
For eg:

int *p = new int(25); //allocates the memory for int and initialises it to the value 25.
float *q = new float(7.5);

DYNAMIC MEMORY ALLOCATION WITH NEW AND DELETE

Example:

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
 {
 char *str = “Kantipur Engineering College”;
 int len = strlen(str);                  //get length of str
char *ptr;                               // make a pointer that points to char
ptr = new char[len+1];                   // set aside memory: string + ‘\0’
strcpy(ptr,str);                         // copy str to new memory area ptr
 cout<<endl<<”ptr =”<<ptr; // show that str is now in ptr
delete ptr;
return 0;
}
```

OUTPUT:

```
ptr = kantipur engineering college
```

ARRAY OF DYNAMIC MEMORY ALLOCATION FOR OBJECTS AND OBJECT ARRAY

Similar to DMA of other datatypes, we can dynamically allocate memory for an object or an array of objects.

classname *pointerobject;

pointerobject = new classname;

pointerobject = new classname[size];

For deallocation of memory:

delete [ ] pointerobject;

Let us consider we have class named “college”

college * ptr;

ptr = new college; *//allocates memory dynamically for an object of class college*

ptr = new college[5]; *//allocates memory dynamically for 5 objects of class college*

delete [ ] college;

THIS POINTER

“this” is a C++ keyword. “this” always refers to an object that has called the member function currently. We can say that “this'' is a pointer that points to the object for which this function was called. For example, the function call A.max() will set the pointer “this” to the address of the object A.

```
#include<iostream>
using namespace std;
class test
{
        int x;
        public:
                test(int value)
                {
                        x= value;
                }
                void print();
};
```

```
void test::print()
{
        cout<<”X =”<<x<<endl;
        cout<<this<<endl;
        cout<<”(*this).x = ”<<(*this).x<<endl;
        cout<<”this -> x = ”<< this -> x<<endl;
}
int main()
{
        test t(12);
        t.print();
}
```

Output:

```
X = 12;
0x16b31728c
*this.x = 12
this->x = 12
```

STATIC DATA MEMBER AND STATIC FUNCTION

- Since it is known that every object has its own copy of data members defined by the class but that is not always true in case of static data members.
- When a static data member is defined then only such an item is created for the entire class regardless of number of objects and static data are shared by all the objects.
- These kinds of variables are declared inside class but initialized outside class.
- Static data members are useful when sharing information that is common to all objects of a class such as number of objects created etc.

STATIC DATA MEMBER AND STATIC FUNCTION

```
#include <iostream>
using namespace std;
class Counter
{
static int c;
public: void
display()
{
 c++;
 cout << "The call to display function " << c << endl;
 }
 };
 int Counter::c=0;
```

```
int main()
{
        counter  C1,  C2,  C3;
        C1.display();
        C2.display();
        C3.display();  return
        0;
 }
```

Output:

```
The call to display function1
The call to display function2
The call to display function3
```

Static Member Function

- A member function which is defined as a static can access only static data member and it can be invoked or called using name or object of that class.

```
#include<iostream>
using namespace std;
class Counter
{
   int a;
   static int c;
   public: void
   input()
  {
      cout<<"\n Enter the value of a:";
      cin>>a;
      cout<<"\n Enter the value of c:";
      cin>>c;
   }
 static void display()
{
      cout<<"\nThe value of c is:"<<c<<endl;
} };
int Counter::c=0;
int main() {
   Counter C1,C2,C3;
   C1.input();
   C2.input();
   C3.input();
   cout<<"\nThe value of variable c is:";
   C1.display();
   Counter::display();
   C3.display();
}
```

Output

```
Enter the value of a:2
Enter the value of c:3

Enter the value of a:4
Enter the value of c:5

Enter the value of a:5
Enter the value of c:54


The value of variable c is:

The value of c is:54


The value of c is:54


The value of c is:54
```

CONSTANT MEMBER FUNCTIONS AND CONSTANT OBJECTS

Constant Member Functions

Constant member function is a function that can’t modify the data member of a class but can use that data member. A member function is declared as a constant member function using keyword ***const***.

return_type function_name(parameters) const;

For example, void
large(int, int) const;

The qualifier ***const*** appears both in member function declaration and definitions. Once a member function is declared as const, it cannot alter the data values of the class. The compiler will generate an error message if such functions try to alter the data values.

Constant Member Functions

```
#include<iostream>
#include<cmath>
using namespace std;
class Coordinate
{
        int x; int y;
        public:
        void input()
        {
               cout << "Enter X and Y: ";
               cin >> x >> y;
         }
         void display() const;
};
```

```
void Coordinate::display() const
{
      float sum;
      cout << "\nX Coordinate : " << x;
      cout << "\nY Coordinate : " << y;
      sum = sqrt(pow(x,2) + pow(y,2));
      cout << "\nMagnitude : " << sum;
      // x++ ; Error : Cannot change the value of
variable inside the constant member function
}
int main()
{
    Coordinate C1;
    C1.input();
    C1.display();
    return 0;
}
```

Note: A constant member function of a class can only invoke other constant member functions of the same class i.e. In above program, we cannot call input() from display().

CONSTANT MEMBER FUNCTIONS AND CONSTANT OBJECTS

Constant Objects

Just like constant variables, a constant object is an object of a class that cannot be modified. A constant object can call only const member functions because they are the only ones that guarantee not to modify the object.

const class_name object_name(parameter);

or,

class_name const object_name(parameter);

The member of a constant object can be initialized only by a constructor, as a part of the object creation procedure.

Constant Objects

```
#include    <iostream>
using namespace std;
class Coordinate  {

       int x;
       int y;
       public:
             Coordinate(int a, int b) {
                      x = a;
                      y = b; }
             void get_coordinate() {
                      cout << "Enter the xcoordinate:";
                      cin >> x;
                      cout << "Enter the ycoordinate:";
                      cin >> y;
              }
           void show_coordinate() const {
                      cout << "\nxcoordinate:" << x;
                      cout << "\nycoordinate:" << y;
              }
void display() {
cout << "\nxcoordinate:" << x;
cout << "\nycoordinate:" << y;
}
};
 int main() {
      Coordinate c1(0,0);
      c1.get_coordinate();
      c1.show_coordinate();
      const Coordinate c2(10,20);
      //c2.get_coordinate()
           //Invalid:get_coordinate() modifies the member of
Coordinate class
      c2.show_coordinate();
              //Valid: show_coordinate() is constant member
function
    //c2.display();
      //Invalid: constant object can only invoke constant
member function
     return 0;
}
```

CONSTANT MEMBER FUNCTIONS AND CONSTANT OBJECTS

mutable keyword

As discussed above, const objects can only invoke const member functions and this const member functions cannot change the data member defined in a class. However, a situation may arise when we want to create const object but we would like to modify a particular data item only. In such situation, we can make it possible by defining the data item as ***mutable***.

Note: the const function changing the value of mutable data cannot have statements that try to change the value of other ordinary data.

mutable keyword

```
#include <iostream>
using namespace std;
class Student
{
char *name;
mutable char *address;
   public:
      Student(char *n, char *ad) {
            name = n;
            address = ad
      }
     void change_name(char *new_name)
     {
           name = new_name;
     }
void change_address(char *new_address) const {
address = new_address;
}
void display() const {
cout << "Name:" << name << endl; cout <<
"Address:" << address << endl; }
};

int main()
{
const Student s1("ABC","PlanetEarth");
//s1.change_name("XYZ");
 s1.change_address("Nepal");
s1.display();
return 0;
}
```

Multiple Choice questions on This pointer, const and static member

1. Which is the pointer which denotes the object calling the member function?

a) Variable pointer
b) This pointer
c) Null pointer
d) Zero pointer

2. The this pointer is accessible ________________

a) Within all the member functions of the class
b) Only within functions returning void
c) Only within non-static functions
d) Within the member functions with zero arguments

3. What is the this pointer in C++?

a) A pointer to the current object
b) b) A pointer to the base class
c) c) A pointer to the derived class
d) d) A pointer to the first data member of the class

4. What is the type of the this pointer in a class named MyClass?

a) MyClass*
b) b) MyClass&
c) c) MyClass**
d) d) MyClass

5. In which of the following scenarios is the this pointer most useful?

a) When overloading the + operator
b) When using static member functions
c) When differentiating between member variables and parameters with the same name
d) When creating a new instance of a class

6. Which of the following statements about the this pointer is incorrect?
a) The this pointer can be used to return the current object from a member function.
b) The this pointer can be NULL.
c) The this pointer is a constant pointer.
d) The this pointer is automatically passed to all non-static member functions.

7. What is a static data member in C++?
a) A data member that can only be accessed by the first object of a class
b) A data member that is shared among all objects of a class
c) A data member that is unique to each object of a class
d) A data member that cannot be modified

8. **How can a static data member be accessed?**

a) Only through an object of the class
b) Only through a pointer to the class
c) Through both the class name and an object of the class
d) Only through the this pointer

9. What is the correct way to define a static data member outside the class?

a) static int MyClass::count = 0;
b) int MyClass::count = 0;
c) MyClass::count = 0;
d) static MyClass::count = 0;

10. When is the memory for a static data member allocated?

a) When an object of the class is created
b) When the first static member function is called
c) When the class is first loaded
d) When the static data member is first accessed

11. Where is a static data member of a class typically stored?

a) In the stack
b) In the heap
c) In the global data segment
d) In the code segment

12. What is the default value of a static data member if it is not explicitly initialized?

a) 0
b) NULL
c) Undefined
d) Compiler-dependent

13. What is a const member function in C++?

a) A function that can only be called on const objects
b) A function that cannot modify any member variables of the class
c) A function that cannot be inherited
d) A function that cannot return a value

14. How do you declare a member function as const in C++?
a) By placing the keyword const before the function name
b) By placing the keyword const after the function name and parameter list
c) By placing the keyword const before the return type
d) By placing the keyword const after the class name

15. Which of the following correctly declares a const member function in a class named Example?
a) void Example() const;
b) void const Example();
c) void Example() const
d) const void Example();

16. Which of the following is true about const member functions?

a) They can call non-const member functions of the same class
b) They cannot modify global variables
c) They can modify mutable member variables
d) They cannot return a valueAnswer:

FRIEND FUNCTION AND FRIEND CLASSES

Friend Function

- As we know, the private members of a class can be accessed only through the public section of the same class. But, if we want to give access to the private member to the function outside the class, we can use the concept of friend function in such circumstances.
- A friend function is a function that is not a member of a class but has access to the class's private and protected members.
- Some important points about the friend functions are:
    - A friend function cannot be called using the object of the class. They are called like a normal function.
    - A friend function can access the resources of a class using the object of the same class.
    - Usually, a friend function has an object as its argument.
    - Friend declaration can be placed anywhere in the class and the access specifier does not matter.

FRIEND FUNCTION AND FRIEND CLASSES

Friend Function

Syntax: class class_name

{...

friend return_type function_name(arguments);

};

return_type function_name(arguments) {

// body of the function

}

Example:

```
class demo
{
   int a;
   float b;
public:
void input() {
   cout<<"Enter the value of a and b";
   cin>> a >> b ;
}
friend  void  output(demo);
};
void output(demo d)
{
      cout<<"The values of a and b are : "<< d.a <<
endl<< d.b;
}
int main()
{
    demo d1;
    d1.input();
    output(d1);
    return 0;
}
```

FRIEND FUNCTION AND FRIEND CLASSES

Friend Function

WAP to make global function which returns the average of 10 number list stored at a member class

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

const unsigned int SIZE = 10;
class Num

{
        int n[SIZE];

        public:
              void input()

              {
                 cout << "Enter the elements:";

                 for(int i=0; i<SIZE; i++)
                 cin >> n[i];
              }
              friend float average(Num);

};
```

```
float average(Num n1)
{

    float sum=0.0;
    for(int i=0; i<SIZE; i++)

    {
            sum += n1.n[i];
     }

    return sum/SIZE;

}
```

```
int main()
{

     Num n1;
     float avg;

     n1.input();
     avg = average(n1);

     cout << avg ;
     return 0;

 }
```

FRIEND FUNCTION AND FRIEND CLASSES

Friend as a bridge function

- Friend functions can be used as a bridge between two classes. It can be used to
- access the private and protected members of two or more classes. This can be accomplished by declaring the same function as the friend of the classes it is
- required to link

Friend as a bridge function

WAP to swap the private data of two different class

```
#include <iostream>
using namespace std;
class ObjB;
class ObjA {
    int x;
    public:
        void input(){
            cout << "Enter value of x: ";
            cin >> x;
        }
        void display(){
            cout << "X : " << x << endl;
        }
    friend void swap(ObjA&, ObjB&);
};
```

```
class ObjB {
    int a;
    public:
        void input(){
            cout << "Enter value of a: ";
            cin >> a;
        }
        void display()
        { cout << "a : " << a << endl; }
friend void swap(ObjA&, ObjB&);
};

void swap(ObjA& a, ObjB& b){
    int tmp = a.x;
    a.x = b.a;
    b.a = tmp;
}
```

```
main() {
ObjA a1;
ObjB b1;
a1.input();
b1.input();
swap(a1, b1);
a1.display();
b1.display();
return 0;
}
```

Output: Enter value of x: 1
Enter value of a: 2
X : 2 a : 1

FRIEND FUNCTION AND FRIEND CLASSES

Friend function of a class as a friend of another

The member functions of a class can be friend functions of another class. In such case, their declaration as a friend in another class uses their qualified name (full name).

Friend function of a class as a friend of another

WAP to make a mult() function of class A as a friend of B and display the proper output

```
#include <iostream>
 using namespace std;
class Beta;
class Alpha
{
int x;
public:
void input()
{
cout << "Enter the value of x :";
cin >> x ;
}
void mult(Beta);
 };
```

```
class Beta {
int y;
public:
void input(){
     cout << "Enter the value of y:";
     cin >> y;
}
friend void Alpha::mult(Beta);
};
void Alpha::mult(Beta t)
{
cout << x << "*" << t.y << "=" << x*t.y << endl;
}
```

```
int main()
{
Alpha a;
Beta b;
a.input();
 b.input();
 a.mult(b);
 return 0;
  }
```

Output: Enter the value of x :1 Enter the value of y: 2 1*2=2

FRIEND FUNCTION AND FRIEND CLASSES

Friend Class

There may be a situation when all the member functions of a class have to be declared as friend of another class. In such situations, instead of making the functions friend separately, we can make the whole class a friend of another class. A friend class is a class whose member functions can access another class's private and protected members. This can be specified as follows:

```
class x;
class z
{
    ........
    friend class x;        //all member functions of class x are friends to z.
};
```

Friend Class

```
#include <iostream>
using namespace std;
class ABC;
class XYZ
{
int x;
public:
friend class ABC;
 };
```

```
class ABC
{
int a;
public:
 void getdata(XYZ &o1)
{
cout << "Enter the value of a:";
cin >> a;
cout << "Enter the value of x:";
cin >> o1.x;
}
void sum(XYZ &o1)
{
   cout << "The sum is:" << a+o1.x << endl;
}
void product(XYZ &o1)
{
cout << "The product is:" << a * o1.x << endl;
}
};
```

```
int main()
{
ABC obj1; XYZ obj2;
obj1.getdata(obj2);
obj1.sum(obj2);
obj1.product(obj2);
return 0;
}
```

Output:

```
Enter the value of a:2
Enter the value of x:1
The sum is:3
The product is:2
```

Multiple Choice questions on Friend Function and Friend Class

1. What is a friend function in C++?

a) A function which can access all the private, protected and public members of a class
b) A function which is not allowed to access any member of any class
c) A function which is allowed to access public and protected members of a class
d) A function which is allowed to access only public members of a class

2. How many member functions are there in this C++ class excluding constructors and destructors?

```
class Box
{
        int capacity;
   public:
        void print();
        friend void show();
        bool compare();
        friend bool lost();
};
```

a) 1 c) 3
b) 2 d) 4

3. Pick the correct statement.

a) Friend functions are in the scope of a class
b) Friend functions can be called using class objects
c) Friend functions can be invoked as a normal function
d) Friend functions can access only protected members not the private members

4. How is a friend function declared in C++?

a) Using the keyword friend followed by function prototype
b) Using the keyword friend before the function prototype
c) Using the keyword friend after the function prototype
d) None

5. What will be the output of the following C++ code?

```
#include <iostream>
using namespace std;
class sample
{
    private:
    int a, b;
    public:
    void test()
    {
        a = 100;
        b = 200;
    }
    friend int compute(sample e1);
};
int compute(sample e1)
{
    return int(e1.a + e1.b) - 5;
}
int main()
{
    sample e;
    e.test();
    cout  << compute(e);
    return 0;
}
```

a) 100
b) 200
c) 300
d) 295