CE144 : OOPC Assignment

20DCS103 Rushik Rathod

Date Page 1

1. Define object-oriented programming and explain feature of object-oriented programming. How it is different from procedure-oriented programming

→ Object oriented programming is a programming paradigm based on the concept of objects, which can contain data and data-code in the form of attributes or properties and code in the form of procedure.

→ Object oriented programming emphasis on the data rather than procedure.
→ In this programming programs are divided into objects.
→ Object oriented programming is more secure as data is hidden and cannot be accessed by the external functions.
→ Objects may communicate with each other through functions, which is one of the crucial features of object-oriented programming.
→ The most important feature of this type of programming is the bottom-up approach. and new data and function can be added.

→ The difference between object oriented programming and procedure oriented programming

* Procedural oriented programming	Object oriented programming
→ Programs are divided into small parts called functions.	→ Program is divided into small parts called objects.
→ Procedural programming follows top-down approach.	→ Object-oriented programming follows bottom-up approach.
→ There is no access specifier in procedural programming.	→ Object oriented programming have access specifiers like private, public, protected, etc.
→ It does not have any proper way for hiding data so it is less secure.	→ Object oriented programming provides data hiding so it is more secure.
→ In procedural oriented programming, function overloading is not possible.	→ In object oriented programming, function overloading is possible.
→ Examples :- C, FORTRAN, etc.	→ Examples :- C++, Java, Python, etc.

2. Explain the following concepts of object oriented programming. 1). Encapsulation, 2). Abstraction, 3) Polymorphism, 4) Inheritance, 5) Dynamic Binding.

→ 1). Encapsulation

→ Hiding the implementation details of the class from the user through an object's methods is known as data encapsulation. In object oriented programming, it binds the code and the data together and keeps them safe from outside interference.

→ 2). Abstraction

→ Abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

→ 3). Polymorphism

→ Polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. Additionally, polymorphism means that a call to a member function will cause a different function to be executed depending on the

type of object that invokes the function.

→ 4). Inheritance

→ Basically, inheritance is the process of acquiring properties. In object oriented programming one object inherit the properties of another object. This also provides an opportunity to reuse the code functionality and fast implementation time.

→ 5). Dynamic Binding.

→ Dynamic Binding is the process of linking procedure call to a specific sequence of code at run time. It means that the code to be executed for a specific procedure call is not known until run-time.

3. What is class in object oriented programming? What is realistic need for making class? Explain class and object with any real life example. What kinds of things can become objects in OOP? What are the applications of object oriented programming?

→ Class :- A class is the building block, that leads to object oriented programming.

→ It is 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 class is a collection of objects of similar type. It is also called abstract data types.

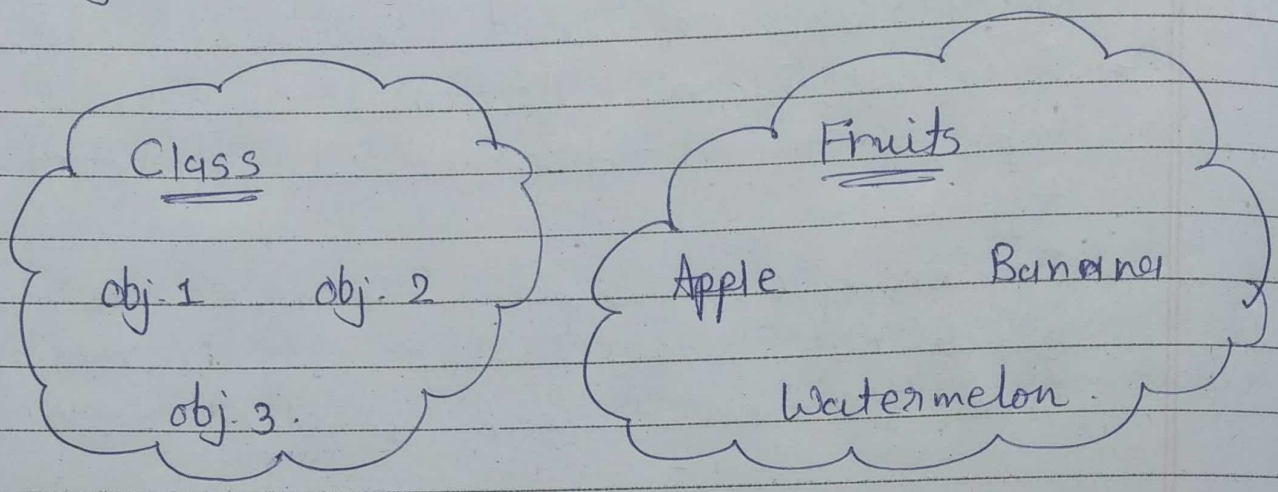
→ For example, suppose student is a class which has two members : roll number and name, then it can be defined as follows :

```
class student
{
    int roll_no;
    char name[20];
};
```

→ Private members of a class are accessible only from within other members of the same class. Further, protected members are accessible from other members of the same class, but also from members of their derived class. Hence, the data is more secure in the concept of making classes which results in the need for creating class in the program.

→ Real life example of class and object is as follows :

→ Here, class is the collection of objects. let Fruits be a class of objects, where objects are banana, Apple, Grapes, etc...



→ Everything in the world can be represented as an object.
For example :- A university, A classroom, A student, A building, A tree, An animal, A city, A country, etc.

→ Applications of object oriented programming:

1) Real - Time System
2) simulation and Modeling
3) Object oriented Database
4) Hypertext and hypermedia.
5). AI and expert system.
6). Neural network and parallel programming
7) Decision support and office automation system
8) CIM / CAM / CAD system.

Date
Page 7

4. What is the basic structure of c++ program ? Explain client - server model.

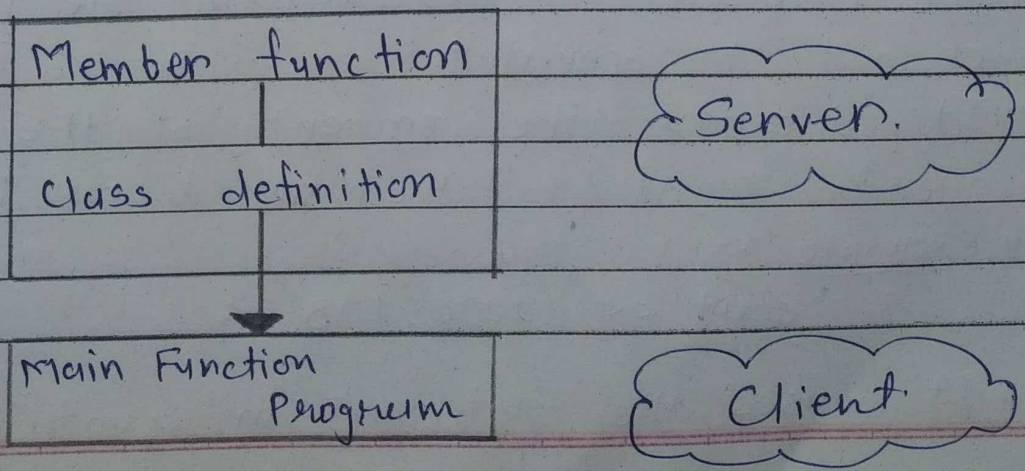
→ A typical c++ program would contain four sections. These sections may be placed in separate code files and then compiled independently or jointly.

Include Files
Class Declaration
Member Function Definition
Main Function Program

→ The class declarations are placed in a header file and the definitions of member functions are placed into another file.

→ Finally, the main program that uses the class is placed in a third file which includes the previous two files as well as any other files required.

→ The client - server model :



→ The client server model is a distributed application framework dividing task between servers and clients, which either reside in the same system or communicate through a computer network or the Internet.

→ The client relies on sending a request to another program in order to access a service made available by a server.

→ The server runs one or more programs that share resources with and distribute work among clients.

5. Explain in detail Insertion Operator [Put to Operator or Output Operator] and Extraction Operator [Get from Operator or Input Operator] with diagram.

→ Insertion Operator :-

→ The operator << is called the insertion or put to operator.
→ It inserts the contents of the variable on its right to the object on its left.
→ Example :-

```
cout << " 20DCS103 " ;
```

→ It causes the statement in quotation marks

to be displayed on the screen.

→ This statement introduces two new C++ features, cout and <<.

→ The identifier cout is a predefined object that represents the standard output stream.

→ It is also possible to redirect the output to other output devices.

→ If name represents a string variable, then the following statement will display its contents:

```
cout << name;
```

→ Extraction Operator :-

→ The operator >> is called the extraction or get from operator.

→ It extracts or takes the value from the keyboard and assigns it to the variable on its right.

→ The statement: cin >>number; is an input statement and causes the program to wait for the user to type in a number. The number keyed in is placed in the variable number.

→ The identifier cin is a predefined object that represents the standard input stream. For example: keyboard.

6. Explain cin and cout objects with example.

→ cin :-

→ cin is a predefined object that represents the standard input stream and is used to take input from the user.

→ cin uses the ~~insertion~~ extraction operator >> which is also known as input operator.

→ Example :-

```
#include <iostream>
int main()
{
    int a;
    cout <<" Enter a number : " <<endl;
    cin >> a;
    cout << a;    //will display the value
    return 0;
}
```

Output :-

```
Enter a number :
103
103
```

→ cout :-

→ cout is a predefined object that represents the standard output stream and is used to show output.

→ cout uses the insertion operator << which is also known as output operator.

→ Example :-

```
#include <iostream>
int main()
{
  int x;
  cout << " Enter an integer : " << endl;
  cin >> x;
  cout << " Entered value is : " << x << endl;
  return 0;
}
```

Output :-

```
Enter an integer :
10
Entered value is : 10
```

7. Why do we need the preprocessor directive #include <iostream> ?

→ In #include <iostream>, #include is a compiler or preprocessor directive.

→ This directive causes the preprocessor to add the contents of the iostream file to the program.

#include <iostream>

- #include : preprocessor directive
- <iostream> : Contains function prototypes for the standard input and standard output functions.

→ This preprocessor ~~causes~~ contains the declarations for the identifier cout and the operator <<. Further, it also contains the declarations for the identifier cin and the operator >>.

8. What is namespace? What is nested namespace? Explain with an example. Why we need to include the directive 'using namespace std'? Where 'std' namespace in actual reside?

→ A namespace is a section of a file that is given a name.

→ In each scope, a name can only represent one entity. So, there cannot be two variables with the same name in the same scope. However, using namespace, one can create two variables or member functions having same name.

→ Example :-

```
#include <iostream>
using namespace std;
namespace first
{   int val = 500; }
int val = 100;
int main()
{   int val = 200;
    cout << first::val << endl;
    return 0;
}
```

Output :- 500.

→ A namespace is a declarative region that provides a scope to the identifiers such as names of the types, function, variables etc. inside it.

→ One can also create nested namespace by using scope resolution operator (::).

→ Example :-

```
#include <iostream>
using namespace std;
namespace first
{   void fun()
    {  cout<<" Inside first "<<endl; }
    namespace second
    {   void fun()
        { cout<<" Inside second "<<endl;}
}
using namespace first :: second;
int main()
{
  fun();  // it will call the function
          // from second namespace.
  return 0;
}
```

→ Output :- Inside second

→ We need to include the directive 'using namespace std' as it tells the compiler that the subsequent code is making use of names in the specified namespace.

→ The 'using' directive can also be used to refer to a particular item within a namespace.
For example: if the only part of the std namespace that one intends to use is cout, one can refer it as follows:

```
using std::cout;
```

9. Differentiate the followings:

A) #define		const
→ #define is pre-processor directive.	→	const is a keyword.
→ #define is not scope controlled.	→	const is a scope controlled.
→ #define can be redefined by un-defining and then defining.	→	const can not be redeclared or redefined even one can not re-assign the value in constant.

B)

endl	\n
→ endl is a manipulator.	→ \n is a character.
→ endl doesn't occupy any memory.	→ \n occupy 1 byte memory as it is a character
→ endl would not specify any meaning when stored in a string.	→ \n being a character can be stored in a string.
→ One can not write endl in between double quotation.	→ One can write \n in between double quotation
→ endl is only supported in c++.	→ \n is supported in c and c++ as well.
→ endl insert a new line and flushes the stream.	→ \n only inserts a new line.

C) 1) Constant pointer

→ Syntax : datatype * const pointername = &var;

→ Ex : char *const ptr = &a ;

Here, we can not change the pointer ptr but we can change the value pointed by ptr.

→ Example :-
```
#include <iostream>
using namespace std;
int main()
{
    char a='A', b='B';
    char *const ptr = &a;
    cout<<"Value pointed to by ptr : "<<*ptr;
    *ptr = b;
    cout<<"\nValue pointed to by ptr : "<<*ptr;
    return 0;
}
```

Output :- Value pointed to by ptr : A
Value pointed to by ptr : B

2). Pointer to Constant

→ Syntax : ~~datatype~~ const datatype *pointername = &var;

→ Ex :- const char *ptr = &a;

Here, we can not change the value pointed by ptr, but we can change the pointer itself.

→ Example :-
```
#include <iostream>
using namespace std;
int main()
{ char a='A', b='B';
  const char *ptr = &a;
  cout<<"Value pointed to by ptr : "<<*ptr;
  ptr = &b;
  cout<<"\nValue pointed to by ptr : "<<*ptr;
  return 0;
}
```

Output :- Value pointed to by ptr : A
Value pointed to by ptr : B.

3). Constant pointer to constant.

→ Syntax : const datatype *const pointername = &var;

→ Ex :- const char *const ptr = &a ;

Here, we can neither change the value pointed by ptr not the pointer ptr.

→ Example :-

```
#include <iostream>
using namespace std;
int main()
{
  char a='A', b='B';
  const char *const ptr = &a ;
  cout<<" Value pointed to by ptr : "<<*ptr;
  //  *ptr = b ;    illegal
  //  ptr = &b ;    illegal
  return 0;
}
```

Output :- Value pointed to by ptr : A

D) Structure in C , Structure in C++ , Class in C++.

→ Here, first we will consider the difference between structure in C and structure in C++.

Date
Page 18

** Structure in C	Structure in C++
→ Structures in C, can not have member functions inside structures.	→ Structures in C++, can hold member functions with member variables.
→ We can not initialize the structure data directly in C.	→ We can directly initialize structure data in C++.
→ In C, we have to write 'struct' keyword to declare structure type variables.	→ In C++, we do not need to write 'struct' to declare variables.
→ C structures can not have static members.	→ C++ structures can have static members.
→ The sizeof operator will generate 0 for empty structure in C.	→ The sizeof operator will generate 1 for empty structure in C++.
→ The data hiding feature is not available in C structures.	→ The data hiding feature is present in C++ structure.
→ C structures does not have access modifiers.	→ C++ structures have access specifiers.

→ In c++, the structure and class are basically same. However, there are some of the differences which are mentioned here...

1) The class members are private by default, but members of structures are public

2) When we derive a structure from a class or structure, the default access specifier of that base class is public, but when we are deriving a class the default access specifier is private.

E). Call by value	Call by reference
→ When we pass values of variable to a calling function, it is called a call by value.	→ When we pass address of varivable to a calling function, it is known as call by reference.
→ In this method, the value of each variable in calling function is copied into corresponding dummy variables of the called function.	→ In this method, the address of actual variables in the calling functions are copied into the dummy variables of the called function

→ The changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function.

→ Example :-

```
#include <iostream>
using namespace std;
void swap(int, int);
int main()
{
  int a,b,c;
  a=10;
  b=20;
  swap(a,b);
  return 0;
}
void swap(int a,int b)
{
  int temp;
  temp = a;
  a = b;
  b = temp;
  cout<<"After swaping"<<endl;
  cout<<"a : "<<a<<endl;
  cout<<"b : "<<b<<endl;
}
```

→ In this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them.

→ Example :

```
#include<iostream>
using namespace std;
void swap(int &, int &);
int main()
{
  int a,b,c;
  a=10;
  b=20;
  swap(a,b);
  cout<<"After swaping"<<endl;
cout<<"a : "<<a<<endl;
cout<<"b : "<<b<<endl;
return 0;
}
void swap(int &x,int &y)
{
  int temp;
  temp = x;
  x=y;
  y=temp;
}
```

→ Output :-

After swaping
a : 20
b : 10.

→ Output :-

After swaping
a : 20
b : 10.

F) Normal function vs Member function of class

→ A normal function always appears outside of a class, while the member function can appear outside of class body.

→ To declare member function outside the class, we must qualified the member function by the name of it's class, which will identify that the function is a member of a particular class.

10. Explain memory management operator new and delete in c++. Why memory management operator are known as free store operator? What are the advantages of using new operator compared to the function malloc()?

** new operator :-

→ The new operator can be used to create objects of any type as follows:

pointer-variable = new data-type;

→ pointer-variable is a pointer of type data-type. It holds the address of the memory space allocated. The new operator allocates the memory to hold the data object of type data-type and returns the address of the object.

```
p = new int;
q = new float;
```

where p is a pointer of type int and q is a pointer of type float. Here p and q must have been declared as pointers of appropriate type.

** delete operator :-

→ When the data object is no longer needed, it is destroyed to release the memory for reuse. The general form of using delete operator is :

```
delete pointer-variable;
```

→ The pointer-variable is the pointer that points to a data object.

For example :-

```
delete p;
delete q;
```

→ If we want to free a dynamically allocated array then it can be done as :

delete [size] pointe-variable;

→ Why it is called free store operator ?

→ The new operator allocates memory from an area called the free store which is also known as dynamic memory and heap.
→ Objects allocated on the free store operator are created and "live" until they are destroyed using the delete operator.
→ That's why memory management operators are known as free store operator.

→ Advantages of using new operator ove malloc() function :-

1) new does not need the sizeof() operator where as malloc() needs to know the size before memory allocation.

2). Operator new can make a call to a constructor where as malloc() can not.

3). new can be overloaded, malloc() can not be overloaded.
4). new would initialize object while allocating memory to it whereas malloc() can not.

Date
Page 24

11. Explain Scope Resolution :: in C++ with example and its applications.

→ The scope resolution operator is used to get hidden names due to variable scopes so that one can still use them.

→ The scope resolution operator can be used as both unary and binary. One can use the unary scope operator if a namespace scope or global scope name is hidden by a particular declaration of an equivalent name during a block or class.

→ Example :-

```
#include <iostream>
using namespace std;
int a = 10;
int main()
{   int a = 20
    {
        int a = 30;
        cout << "Value of a = " << a << endl;
        cout << "Value of ::a = " << ::a << endl;
    }
    cout << "Value of a = " << a << endl;
    cout << "Value of ::a = " << ::a << endl;
    return 0;
}
```

→ Output :-

Value of a = 30
Value of ::a = 10
Value of a = 20
Value of ::a = 10.

12. Explain Reference variables with example. What are the advantages of Call by Reference as compared to call by address? Can we declare an array of references? Can we assign NULL value to reference variable? Is reference variable a pointer variable? Can we declare a reference variable without initializing it?

*→ Reference variable :-

When a variable is declared as a reference, it becomes an alternative name for an existing variable.

→ A variable can be declared as a reference by putting '&' in the declaration.

→ Example :-

```
#include <iostream>
using namespace std;
int main()
```

```
{   int x = 10
    int& ref = x;
    ref = 20;
    cout << " x = " << x << endl;
    x = 30;
    cout << " ref = " << ref << endl;
    return 0;
}
```

→ Output :-

x = 20

ref = 30.

*) Advantages of call by reference as compared to call by address :-

1). In the call by reference method, the function can change the value of the argument, which is quite useful.

2). It does not create duplicate data for holding only one value which helps the programmer to save memory space.

3). In this method, there is no copy of the argument made. Therefore it is processed very fast.

4). A person reading the code never knows that the value can be modified in the function.

** Can we declare an array of references ?
→ No, we cannot declare an array of references because if we apply any operation on a reference, it will directly affect the original value.

** Can we assign NULL value to reference variable ?
→ No, we cannot assign a NULL value to reference variable.

** Is reference variable a pointer variable ?
→ No, reference variable is not a pointer variable.

** Can we declare a reference variable without initializing it ?
→ No, we cannot declare a reference variable without initializing it. We must initilize reference variable at the time of declaration.

13. Explain bool datatype and wchar_t datatype with example.

** bool datatype :-

→ A boolean data type is declared with the bool keyword and can only take the values true or false.

→ When the value is returned, true = 1 and false = 0.

→ Example :-

```
#include <iostream>
using namespace std;
int main()
{
    bool a = true, b = false;
    cout << "Value of a : " << a << endl;
    cout << "value of b : " << b << endl;
    return 0;
}
```

→ Output :- Value of a : 1
value of b : 0.

*→ Wide char :-

→ Wide char is similar to char datatype, except that wide char take up twice the space and can take on much larger values as a result. char can take 256 values which corresponds to enthies in the ASCII table.

→

→ On the other hand, wide char can take on 65536 values which corresponds to UNICODE values which is a recent international standard.

→ wchar_t datatype occupies 2 or 4 bytes depending on the compiler being used.

→ Example :-

```
#include <iostream>
using namespace std;
int main()
{
    wchar_t w = L'A';
    cout << "Wide character value = " << w;
    return 0;
}
```

→ Output :- Wide character value = 65

→ Here, L is the prefix for wide character literals and wide-character string literals which tells the compiler that the char or string is of type wide-char.

14. What is inline function ? What is the use of inline function ? Explain with an example. How does an inline function differ from a preprocessor macro ? Write a situation where an inline function may not work.

→ Everytimes a function is called, it takes a lot of extra time in executing a series of instructions for tasks such as jumping to the function, saving registers, pushing arguments into stack and returning to the

calling function.

→ To eliminate the cost of calls to small functions, c++ provides a new feature called function.

→ Inline keyword sends a request, not a command, to the compiler. The compiler may ignore a request if the function definition is too large or too complicated and compile the function as normal function

→ Example :-

```
#include <iostream>
using namespace std;
inline int square(int s)
{
   return s*s;
}
int main()
{
   cout << "The square of 3 is : " << square(3);
   return 0;
}
```

→ Output :- The square of 3 is : 9

** The situations where inline function may not work:

1) If the function contains a loop.

2) If a function contains static variables.
3) If a function is recursive.
4) If a function contains switch or goto statement.
5) If a function return type is other than void and the return statement doesn't exist in function body.

15. Explain Return by Reference with example.

→ A function can return a reference. However, we should never return local variables as a reference, because local variable will be erased as soon as the function returns.

→ Example :-

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

int& retbyref()
{
   return x;
}
int main()
{
   retbyref() = 10;
   cout << "Value is : " << x;
   return 0;
}
```

→ Output :- Value is : 10

16. What is class ? Explain how does a class accomplish information hiding (data hiding), data abstraction and encapsulation ? Explain real life example information hiding, data abstraction and encapsulation.

→ Class :- A class is a way to bind the data and its associated functions together.

→ It allows the data and functions to be hidden, if necessary, from external use.

→ Private member functions and data members can be created for making them hidden and a private member function can only be called by another function is a member of its class.

→ The binding of data and functions together into a single class-type variable is referred to as encapsulation.

→ Real life example of information hiding, data abstraction and encapsulation is ATM machine, which takes only important and required data from the user and another

unnecessary but important data will not be visible to the user.

17. Describe the mechanism of accessing data members and member function of a class in the different cases.

A) Inside the main program.

→ When class is ~~defined~~ created inside the main program, the accessibility of data members and member function will depend upon the nature of them.

→ Private member-function and data members can only be called by the object and using dot operator while public member-function and data-members are accessible to the entire class.

B) Inside a member function of the same class.

→ In this situation, all the data members and memeber functions can be accessed by using dot operator while ~~not~~ considering all public.

C) Inside a member function of another class

→ When a progeammer wants to access the

data members and member function, it can be possible by using friend function and also can be accessed by using dot operator.

18. What is friend function? Explain in detail with example. What are the features of friend function? What are the advantages and disadvantages of using friend function?

→ A friend function is the function, which can access all private and protected members of the class

→ Example :-

```
#include <iostream>
using namespace std;
class Box
{  private :
       int length;
   public :
       Box() : length(0) { }
       friend int printLength(Box);
};
int printLength (Box b)
{    b.length += 10;
     return b.length
}
int main()
```

```
{ Box b;
  cout << "Length of box : " << printLength(b) << endl;
  return 0;
}
```

** Advantages :-

1) While defining the friend function, there is no need to use the scope resolution operator as friend keyword.

2) The friend can be defined anywhere in the program similar to normal function.

3) It can be invoked like a normal function without the help of any object.

4) A function may be friend of more than one class.

** Disadvantages :-

1) The friend function is declared in the class but it, not a member function of class. To access the private data member of class it is necessary to create objects.

2) When a function is friend of more than one class then forward declaration is needed.

19 Explain how member function of one class can become friend of another class with an example. Also explain friend class with example.

→ The member function of one class can become the friend of another class by just following syntax of declaration.

→ When we write friend returntype class name::fun(); in another B class then it will be the friend function of the A class.

→ Example :-

```
#include <iostream>
using namespace std;
class A
{
    void max();
};
class B
{
    friend void A :: max();
};
```

→ Here, max() of class A is a friend function of class B.

→ We can also declare all the member functions of one class as the friend functions of another

class. In this case, the class is called a friend class.

→ Example :-

```
#include <iostream>
using namespace std;
class A
{
    int x = 5;
    friend class B;
};
class B
{
    public:
        void display (A &a)
        {
            cout << " Value of x is : " << a.x;
        }
};

int main()
{
    A a;
    B b;
    b.display(a);
    return 0;
}
```

→ Output :- Value of x is : 5

20. What is function overloading ? Can we use same member function name for a member function of a class and an outside function in the same program file ? If yes, how are they distinguished ? If no, give reasons.

→ Overloading refers to use of the same thing for different purposes. C++ permits overloading of functions.

→ This means that we can use the same function name to create functions that perform a variety of tasks. This is known as function polymorphism in OOP.

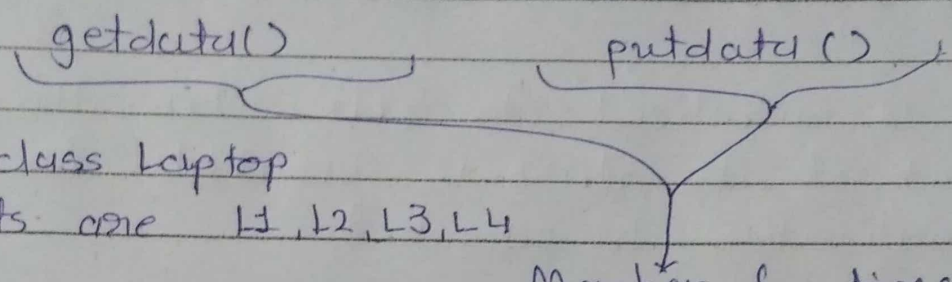
→ Example :-

```
#include <iostream>
using namespace std;
class simple
{
   int a, b;
  public:
      void getdata()
      {
        a=0
        b=0
      }
      void getdata(int x)
      {     a = x;
            b = x;      }
```

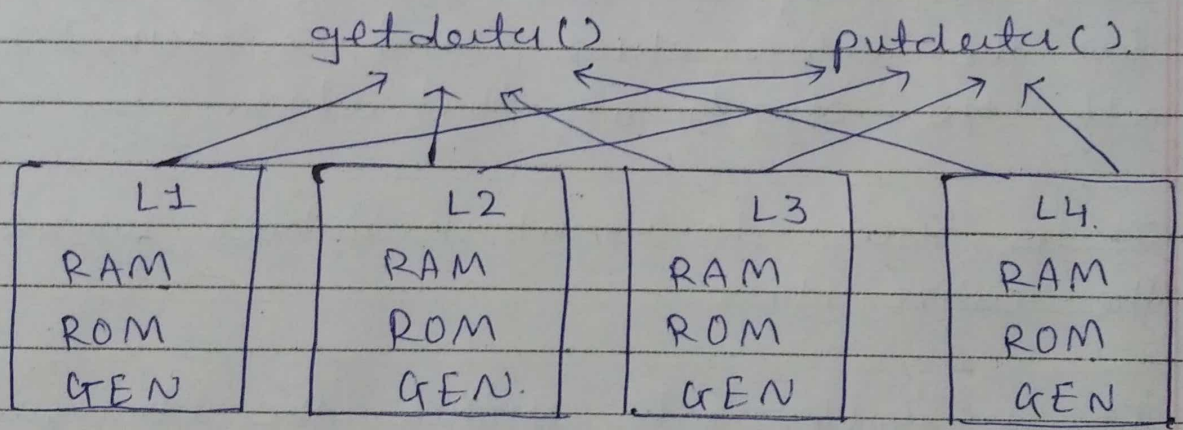
```
    void getdata (int x, int y)
    {
        a = x;
        b = y;
    }
    void putdata()
    {
        cout << "\n a = " << a;
        cout << "\n b = " << b;
    }
};
int main()
{
    simple s1, s2, s3;
    s1.getdata();
    s2.getdata(40);
    s3.getdata(50, 60);
    s1.putdata();
    s2.putdata();
    s3.putdata();
    return 0;
}
```

21 Draw the memory allocation of class having 3 data members and two member functions of the same 4 objects of the same class and explain it.

→ Here, let us assume a class name Laptop which has data members RAM, ROM, GEN. for ram, internal memory and generation respectively.



→ for class Laptop objects are L1, L2, L3, L4



→ Every single data member can have different values for different respective objects, every object declared for the class Laptop has an individual copy of all the data members.

→ However, for member functions, memory is allocated once when the class is defined. There is only single copy of each member function.

22. What is static data member and static member function? Where do we declare member of class static?

** The static member variable/static data member:

→ 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.

→ It is used to maintain values common to the entire class.

→ It is visible only within the class, but its lifetime is the entire program.

** Static member function :-

→ A static member function can have access to only static members declared in the same class.

→ A static member function can be called using the class name.

23. Explain arrays of object with example.

→ The array of type class contains the objects of the class as its individual elements.

→ Thus, an array of a class type is also known as array of object.

→ Syntax :- class_name array_name [size];

→ Example :-

```
#include <iostream>
using namespace std;
class Complex
{
   int x;
   public:
      void setx (int i) { x = i; }
      int getx ( ) { return x; }
};
void main()
{
   Complex c[4];
   int i;
   for (i=0; i<4; i++)
   {   c[i].setx[i);   }
```

```
for(i=0 ; i<4 ; i++)
{ cout << " c[" << i << "].getx(): "
   << c[i].getx() << "\n" ;
}
}
```

→ Output :-
```
c[1].getx(): 0
c[2].getx(): 1
c[3].getx(): 2
c[4].getx(): 3
```

24. List the drawbacks of classes with all public data members.

→ When we create all data members public, there will be no meaning of the data hiding feature of c++ language.

→ By creating all data membens public, any function can have an access to the member function which might end up in data in-segyority.

→ One can not do the encapsulation and abstraction when all the data-membens and data functions are created public, which is a major disadvantage of creating them public.

25. Explain Local Classes with an example.

→ A class declared inside a function becomes local to that function and is called Local class in c++.

→ Local classes can be used global variables and static variables but can not use automatic variables. The global variables should be used with the scope operator (::).

→ Local classes can not have data members and member functions must be defined inside the local classes.

→ Example :-

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

void fun()
{
    class Complex
    { public :
        void show()
        {
            cout << "You are in Complex." <<endl;
        }
    };
    Complex C;
    C.show();
}
```

```
int main()
{
    fun();
    return 0;
}
```

→ Output :-

You are in Complex.
