freelance_Project available to buy contact on 8007592194

SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

freelance_Project available to buy contact on 8007592194

21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql
41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48		React+Springboot+MySql
49		React+Springboot+MySql
50		React+Springboot+MySql

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/cceesept2023

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project

C++ Programming

Basic concept of C program and C++ program

```
#include<stdio.h>
//main->It is a user-defined function
//calling main function is a job of OS. Hence main function is
called as callback function
// we cannot declare main function as static and constant
//Bcoz main ko OS call karta hai bahar se call hai same scope nhi
isliye static and const nhi kar sakte
//per project we  can define only one main

//if we try to build c++ project without main then linker generates
error
//In C++ executions starts from main, It is called calling
function.It is designated to call other functions
// we cannot define main function inside function/structure/class
void print( void  ); // Global function declaration
int main()
{
    //void print( void  ); // local function declaration
    print( ); // function call
    return 0;
}
// called function
void print( void  )// function defination / Implementation
{
    printf("Hello OM25 \n");
}
```

```
#include<stdio.h>
//This is Global function defination no need of declaration
void print( void  )// called function
{
    printf("Hello OM25 \n");
}
int main()  // calling function
{
    print( ); // function call       return 0; }
```

```
#include<stdio.h>
int sum( int a , int b ) //formal argument /  a , b -> function
parameters
{
        int result;
        result = a + b;
        return result;
}

int main()
{
    int x = 10;
    int y = 20;
    int result;

    result = sum ( x , y ); // actual arg / function arguments
    printf("Result  :   %d",result); //30
    return 0;
}
```

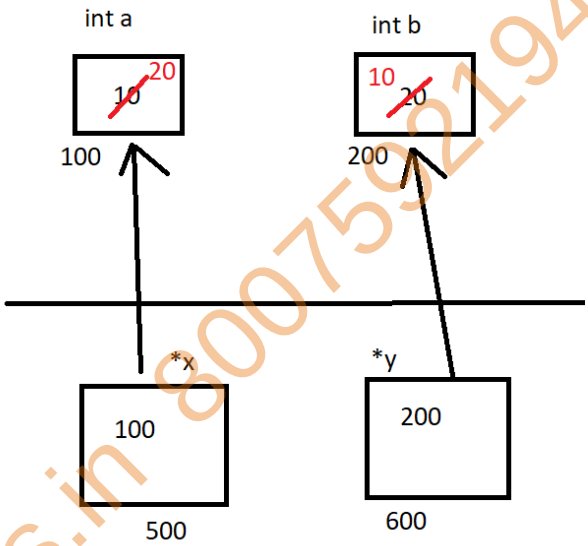
Swap cannot done

```
#include<stdio.h>
void swap ( int x, int y)
{
        int temp = x;
        x = y;
        y = temp;
}
int main()
{
    int a = 10;
    int b = 20;

    swap( a ,b );

//      swap( a ,b );  ==> we are calling swap function by value
//      we are passing function argument by value
//      Function call by value

    printf("a   :   %d\n",a); // 10
    printf("b   :   %d\n",b); //20
    //swap cannot done bcoz by value function cannot pass two
arguments.
    return 0;
```

```
}
```

What is the solution for above swap programm? Look at here
Pass by address is the solution

```
#include<stdio.h>
void swap ( int *x, int *y)
{
        int temp = *x;
        *x = *y;
        *y = temp;
}
int main()
{
    int a = 10;
    int b = 20;

    swap( &a ,&b );
    // function pass by address

    printf("a   :   %d\n",a); //20
    printf("b   :   %d\n",b); //10

    return 0;
}
```



Without Structure

```
#include<stdio.h>
int main()
{
    char name[32];
    int empid;
    float salary;

    printf("Name    :   ");
    scanf("%s",name); //name of array represent address
itself
                      //i.e we cannot use & address
operator.
    printf("Empid   :   ");
    scanf("%d",&empid);
```

```
    printf("Salary  :   ");
    scanf("%f",&salary);

    printf("Name    :   %s\n",name);
    printf("Empid   :   %d\n",empid);
    printf("Salary  :   %f\n",salary);

    return 0;
}
```

Output:-Name : Ashok
Empid :7
Salary : 10000

Now by using Local Structure

```
#include<stdio.h>
int main()
{
    struct Employee // local structure
    {
        char name[32];
        int empid;
        float salary;
    };
    struct Employee emp;
    // struct Employee  : Data type
    // emp : variable/object

    printf("Name    :   ");
    scanf("%s",emp.name);

    printf("Empid   :   ");
    scanf("%d",&emp.empid);

    printf("Salary  :   ");
    scanf("%f",&emp.salary);

    printf("Name    :   %s\n",emp.name);
    printf("Empid   :   %d\n",emp.empid);
    printf("Salary  :   %f\n",emp.salary);
    return 0;
}
```

Output:-Name : Ashok

```
        Empid :7
        Salary : 10000
```

Now by using Global Structure

```
#include<stdio.h>
struct Employee // Global structure
{
    char name[32];
    int empid;
    float salary;
};
//struct Employee *ptr  = &emp
void print ( struct Employee *ptr )
{
    printf("Name    :   %s\n",ptr->name);
    printf("Empid   :   %d\n",ptr->empid);
    printf("Salary  :   %f\n",ptr->salary);
}
int main()
{
    struct Employee emp = { "Ketan", 1 , 1000.00f};
    // struct Employee  : Data type
    // emp : variable/object
    print(&emp); // pass by address
    return 0; }       Output:- Ketan, 1 , 1000.00
```

Token

- Program is collection of statements.
- Statement is an instruction given to the computer.
- Every instruction is made from token.
- Token is basic unit of a program.
- Following are tokens in C:

1. Identifier 2. Keyword 3. Constant / Literal 4. Operator 5. Punctuator/Separator

Object Oriented Programming Structure

- Object-Oriented Programming` (OOP) was coined by Alan Kay.
- According to Alan Kay, the essential ingredients of OOP are:
    1. Message passing 2. Encapsulation 3. Dynamic binding
- Grady Booch is inventor of UML(Unified Modelling Language).
- According to Grady Booch, there are 4 major and 3 minor pillars of oops

Major Pillars Of OOPS

• Following are the four major pillars of oops:

1. Abstraction – To achieve simplicity
2. Encapsulation – To achieve data hiding
3. Modularity – To minimize module dependency
4. Hierarchy – To achieve reusability

• By major, we mean that a language without any one of these elements is not object oriented

Minor Pillars Of OOPS

• Following are the three minor pillars of oops:

1. Typing – To reduce maintenance of the system
2. Concurrency – To utilize hardware resources efficiently
3. Persistence – To maintain state of object on secondary storage.

• By minor, we mean that each of these elements is a useful, but not essential, to classify language object oriented.

C++ Introduction

• C++ is a general-purpose programming language created by Bjarne Stroustrup in 1979.
• It is a pure C programming language in addition with Classes.
• It is object oriented programming language which is derived from C and Simula.
• Extension of C++ source file should be .cpp.
• Initial name of the language was “C with Classes”.
• C++ is standardized by the International Organization for Standardization(ISO).
• C++98, C++03, C++11, C++14, C++17, C++20, C++23 are C++ standards.
• Reference Website : https://en.cppreference.com/w/cpp

C++ Programming: Language Keywords

• Keywords are the reserved words that we can not use as identifier
• Reference : https://en.cppreference.com/w/cpp/keyword
• According to C++ 98, there are 74 keywords in C++.
• C++98 : 74 keywords
• C++11 : 10 keywords

Data Type

• Data type of any variable decides 4 things:

1. Memory 2. Nature 3. Operation 4. Range

• Types of data type:

1. Fundamental data types 2. Derived data types 3. User Defined data types

Fundamental Data Types	Derived Data Types	User Defined Data Types
void(Not Mentioned)	Array	enum
bool(1 byte)	Function	union
char(1 byte)	Pointer	structure
wchar_t(2 bytes)	Reference	class
int(4 bytes)		
float(4 bytes)		
double(8 bytes)		

C++ Concept Start

```
struct Employee // Global structure
{
    char name[32];
    int empid;
    float salary;
};

// Global function
void accept_record( struct Employee *ptr)
{
    printf("Name    :   ");
    scanf("%s",ptr->name);

    printf("Empid   :   ");
    scanf("%d",&ptr->empid);

    printf("Salary  :   ");
    scanf("%f",&ptr->salary);
}
void print_record( struct Employee *ptr )
{
    printf("Name    :   %s\n",ptr->name);
    printf("Empid   :   %d\n",ptr->empid);
    printf("Salary  :   %f\n",ptr->salary);
}
```

```
int main() // calling function
{
    struct Employee emp;

    accept_record(  &emp  );
    print_record (  &emp );

    return 0;
}
```

- ➔ Here name,empid,salary are member of structure function
- ➔ And the accept_record and print_record are the global function
- ➔ Not member of structure still it can be access
- ➔ Employee ke jo member hai usse jagah pe access hone chahiye jo function
  is structure ke member hai
- ➔ name ,empid,salry jo is structure ko belong karte hai ussi ko access hone chahiye

---

```
#include<stdio.h>
struct Employee // Global structure
{
    private:
    char name[32];
    int empid;
    float salary;


    public:
    void accept_record( void )
    {
        printf("Name    :   ");
        scanf("%s",name);

        printf("Empid   :   ");
        scanf("%d",&empid);

        printf("Salary  :   ");
        scanf("%f",&salary);
    }
```

```
void print_record( void  )
    {
        printf("Name    :   %s\n",name);
        printf("Empid   :   %d\n",empid);
        printf("Salary  :   %f\n",salary);
    }
};
int main() // calling function
{
    struct Employee emp;

    emp.accept_record(  );//accept_record(  &emp  );
    emp.salary = 0;
    emp.print_record(   );//print_record (  &emp );

    return 0;
}
```

->So we can change something here, first the function is written in a structure that is why now they are members of structure and now they are not global functions this is global structure.

-> second change is how to access them so now emp.accept_record() & emp.print_record()

-> third change explicitly address/argument pass krne ki jrurat nhi hai accept_record or print_record ko.

->catch krte waqt bhi address operator use karne ki jarurat nhi internally implicitly perforem ho raha hai by this pointer.

->name salary empid update ya change karna hai tho sirf aur sirf structure member mai hi honi chahiye par aisa nhi ho raaha.when you change emp.salary=0;

->so I want to restrict them c++ mai there is feature called access specifier ya visibility lable => private

->private ka matlab ab structure ke member sirf structure mai hi change honge

->Now name empid salary private ho gaye ab lenik private ke niche ke sare ke sare private ho gaye accept_record aur print_record bhi so I cannot used them in main for printing and accept

->So simply I do them public now they can be access and my name,empid,salary are private

->Now bcoz of private salary cannot change

->By default in c++ structure member are public.

-> in C structure ke ander function nhi likh sakhte

->in c++ structure ke ander function likh sakhte hai bcoz of this pointer

---

Access specifier

1.Private 2. Public 3.Protected (we can see in inheritors)

Access Specifier	Same Class	Derived Class	Non Member Function
private	A	NA	NA
protected	A	A	NA
public	A	A	A

- If we want to control visibility of the members of a structure/class then we should use access specifier.
- Private, protected and public are access specifiers in C++.
- In C++, structure members are by default public and class members are by default private.

Class Concept C++:-

class members are by default private.

```
#include<stdio.h>
class Employee
{
    private:
    char name[32]; // 32 // Data members / field / attribute / property
    int empid; // 4
    float salary; // 4

    //member functions/method/operation/behaivour
    public:
    void accept_record( void )
    {
        printf("Name    :   ");
        scanf("%s",name);

        printf("Empid   :   ");
        scanf("%d",&empid);

        printf("Salary  :   ");
        scanf("%f",&salary);

    }
```

```
    void print_record( void  )
    {
        printf("Name    :   %s\n",name);
        printf("Empid   :   %d\n",empid);
        printf("Salary  :   %f\n",salary);
    }
};

int main() // calling function
{
     int  num;
     Employee emp; // Instantiation
    //classname Identifier
    // emp ==> variable
    //         Instance
    //         object
    // Ketan 1 1000
    emp.accept_record(  );// message passing
//accept_record(  &emp  );
    emp.print_record(   );//message passing // print_record (  &emp
);

    return 0;
}
```

->Class mai Data member ko private krenge member function ko nhi
->structure ka object banate waqt struct keyword dena padta tha
  Struct Employee emp;
->Class ka object banate waqt class keyword dena optional hai
   Employee emp; or class Employee emp;
->object ke ander sirf aur sirf data type ko space milti hai
->member function are behaviour

Data member

- A variable declared inside class/class scope is called data member.
- Data member is also called as field/attribute/property.
- Only data member get space once per object and according to order of their declaration inside class.
- Only by understanding problem statement, we can decide data members inside class/structure.

```
class Complex{
private:
   int real;  //Data Member
   int imag;  //Data Member
};
```

Member function

- A function defined/implemented inside class/class scope is called member function.
- Member function is also called as method/operation/behavior/operation.
- Member function do not get space inside object. All the objects of same class share single copy of member function

```
class Complex{
public:
   void acceptRecord( void ){ //Member Function
   }
   void printRecord( void ){ //Member Function
   }
};
```

Class

- class is a keyword in C++.
- It is a collection of data member and member function.
- It is a basic unit of encapsulation.
- Class can contain:
- Data members
  1. Non static data members 2. Static data members

- Member Functions
  1. Non static member functions 2. Static member functions
- Nested Types Class (not in syllabus)

- Some member functions are special: under certain circumstances they are defined by the compiler:
  1. Constructor 2. Destructor 3. Copy constructor 4. Assignment operator function

• A class from which, we can create object is called concrete class. In words, we can instantiate concrete class. (instantiate means creating object)
• A class from which, we can not create object is called abstract class. In words, we can not instantiate abstract class.

```
Example:- Clsss Test //abstract class
          { public:
              Virtual void f1()=0;   //abstract method
          };
```

• A member function of a class, which is having a body is called concrete method.
• A member function of a class, which do not have a body is called abstract method.

Instance and Instantiation

• Variable or instance of a class is called object.
  Ex. emp is a variable/instance/object
• Process of creating object from a class is called instantiation.
  (Object create krne ko hi instantiation bolte hai)
• Syntax : 1. class ClassName identifier; //or 2. ClassName identifier; • As shown above, during instantiation, use of class keyword is optional. • Example: • Complex c1;
• Here class Complex is instantiated and name of instance is c1.
• Global variable/Local Variable/Function Parameter / Member function do not get space inside object
•object ke ander sirf aur sirf data member ko hi space milti hai.

Message Passing

• Process of calling member function on object is called message passing
Consider following example:

```
Complex c1;
c1.acceptRecord( );  //Message Passing
c1.printRecord( );   //Message Passing
```

or

```
Complex c1;
c1.Complex::acceptRecord( );    //Message Passing
c1.Complex::printRecord( );     //Message Passing
```

Scope resolution Operator

->Now in this program member function definition written outside the class function and declaration of member function given in the class.

-> that's why we used here :: scope resolution operator for member function and it is denotes as(::)

```
#include<stdio.h>
class Employee
{
    // Data members / field / attribute / property
    private:
    char name[32]; // 32
    int empid; // 4
    float salary; // 4

    //member functions/method/operation/behaivour

    public:
    // declaration
    void accept_record( void );
    void print_record( void  );
};


// :: ==> scope resolution operator
// member function defined outside the class
// defination
void Employee :: accept_record( void )
{
    printf("Name    :   ");
    scanf("%s",name);

    printf("Empid   :   ");
    scanf("%d",&empid);

    printf("Salary  :   ");
    scanf("%f",&salary);

}
```

```
void Employee :: print_record( void  )
{
    printf("Name    :   %s\n",name);
    printf("Empid   :   %d\n",empid);
    printf("Salary  :   %f\n",salary);
}

int main() // calling function
{
     int  num;
     Employee emp; // Instantiation
    //classname Identifier
    // emp ==> variable
    //         Instance
    //         object

    // Ketan 1 1000
   emp.accept_record(  );// message passing
//accept_record(  &emp  );
    emp.print_record(   );//message passing // print_record (  &emp
);

    return 0;
}
```

Storage Classes

- Following are the storage classes in C/C++:
  1. auto 2. static 3. extern 4. Register
- Storage class decides scope of variable and function and lifetime of the **variable. Scope in C++**
- Scope decides region/area/boundary where we can access variable/function.
- Following are the scope in C++:

1. Block scope
2. Function scope
3. Prototype scope
4. Class Scope
5. Namespace Scope
6. File Scope
7. Program Scope

Scope represent in below program.

```
#include<stdio.h>
int num5 = 60; // program scope
static int num6 = 50; // file scope
namespace na
{
    int num4 = 40;  // namespace scope

    class test
    {
        int num3; // class scope
    };
}
int main()
{
    int sum( int num1 , int num2);// prototype scope
    int num2 = 20; // local variable // function scope
    {
        int num1 = 10; // local variable ==> block scope
    }
}
```

Lifetime in C++

- Lifetime represents existence of variable/object inside RAM.
- Following are the lifetimes in C++:

1. Automatic lifetime
2. Static lifetime
3. Dynamic lifetime

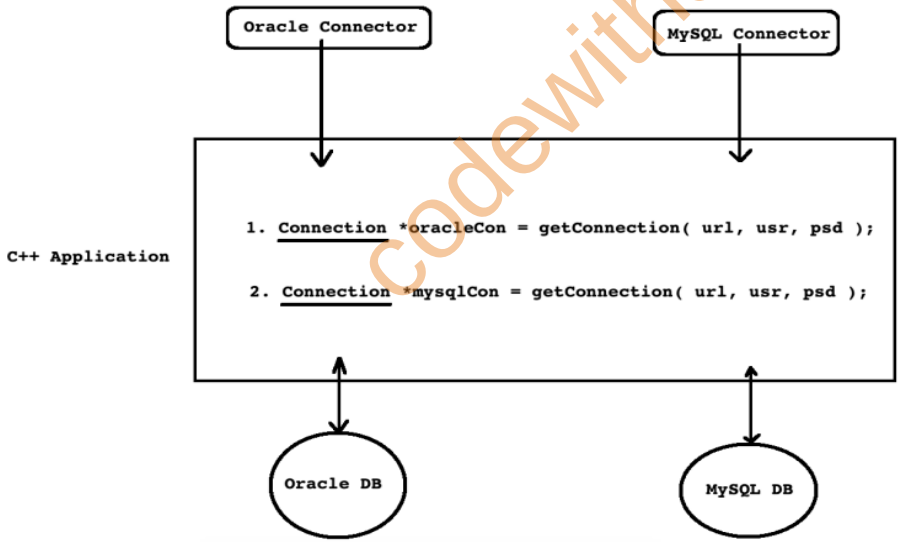
->I want to used global variable in main scope for print I need to used scope resolution operator(::).
->Scope resolution operator print global variable not local variable.

- Inside same scope, we can not give same name to the multiple variables. But name of local variable and global variable can be same.
- If name of local variable and global variable is same then preference is always given to the local variable.
- In C++, using scope resolution operator, we can access value of global variable inside function.

```
#include<stdio.h>
int num1 =10 ; // program scope
int main()
{
    int num1 = 20; // function scope
    printf("num1    :   %d\n",::num1); //10
    printf("num2    :   %d\n",num1);// 20
    {
            int num1 = 30;
            printf("num1    :   %d\n",::num1); //10
            printf("num1    :   %d\n",num1); //30
    }
    return 0;
}
```

Namespace Concept

What is the Need of Namespace?
->for example two compnies of oracle and mysql.i want to fetch data from database from this companies connector but the class name of this connector is same connection so,that why c++ complier may get confused which connection is either of oracle or mysql so to avoid fixed this type of problem in c++ there is concept of Namespace.
1. To avoid name clashing/collision/ambiguity. 2. To group functionally related / equivalent types together
Example diagram refer:-



Namespace concept by program

```
#include<stdio.h>

namespace na
{
    int num1 = 10;// OK // namespace scope
}
int num1=50;
int main()
{
    printf("Num1    :   %d\n",na::num1); //10
    printf("Num1    :   %d\n",num1); //50

    return 0;
}
```

->Main ko hum namespace main hi likh sakate linker error give.

If name of the namespaces are different then name of members of namespace may/may not be same

```
#include<stdio.h>
namespace na
{
    int num1 = 10;// OK // namespace scope
    int num3 = 30;
}
namespace nb
{
    int num2 = 20;// OK // namespace scope
    int num3 = 40;
}

int main()
{
    printf("Num1    :   %d\n",na::num1); //10
    printf("Num3    :   %d\n",na::num3); //30

    printf("Num2    :   %d\n",nb::num2); //20
    printf("Num3    :   %d\n",nb::num3); //40

    return 0;
}
```

• If name of the namespaces are same then name of members of namespace can not be same

```
#include<stdio.h>
namespace na
{
    int num1 = 10;// OK // namespace scope
    int num3 = 30;
}
namespace na
{
    int num2 = 20;// OK // namespace scope
    int num3 = 40;//not ok
}
int main()
{
    printf("Num1    :   %d\n",na::num1); //10
    printf("Num3    :   %d\n",na::num3); //error

    printf("Num2    :   %d\n",na::num2); //20
    printf("Num3    :   %d\n",na::num3); //error

    return 0;
}
```

->in this program error give bcoz
->Namespace ke name same hai tho chalata hai but it treated like below
thats why 2 same variable num3 give error.

```
namespace na
{
    int num1 = 10;// OK // namespace scope
    int num3 = 30; //error same variable
    int num2 = 20;// OK // namespace scope
    int num3 = 40; //error same variable
}
```

Now in this case same namespace but different variable and its work observe output in this program

```
#include<stdio.h>
namespace na
{
    int num1 = 10;// OK // namespace scope
    int num3 = 30;
}
```

```
namespace na
{
    int num2 = 20;// OK // namespace scope
    int num4 = 40;
}


int main()
{
    printf("Num1    :   %d\n",na::num1); //10
    printf("Num3    :   %d\n",na::num3); //30

    printf("Num2    :   %d\n",na::num2); //20
    printf("Num4    :   %d\n",na::num4); //40

    return 0;
}
```

• We can define namespace inside another namespace. It is called nested namespace.

```
#include<stdio.h>
int num1 = 100; // program scope
namespace na // top level namespace
{
    int num2 = 20; // namespace scope
    namespace nb // nested namespace
    {
        int num3 = 30;
    }
}
int main()
{
    printf("Num1    :   %d\n",na::nb::num3); //30
    printf("Num2    :   %d\n",na::num2); //20
    printf("Num1    :   %d\n",::num1); //100
    return 0;
}
```

Now my requirement is I want to used one variable many time so every time scope resolution operator is must but I want to avoid that so the thing is in c++ is using directive. Simply used using namespace na and after that you didn't need of scope resolution operator.Refer below program then.

• If we want to use members of namespace frequently then we should use using directive

```
#include<stdio.h>
namespace na
{
    int num1  = 10;
}
int main()
{
    using namespace na;
    printf("Num1    :   %d\n",num1); // 10
    return 0;
}
```

Now, But namespace aur main directive mai same variable honga tho using directive kam nhi karenga you have to give scope resolution operator.

Observe below program output.

```
#include<stdio.h>
namespace na
{
    int num1  = 10;
}
int main()
{
        int num1 = 20;
        using namespace na;
        printf("Num1    :   %d\n",num1); //20
        printf("Num1    :   %d\n",na::num1); //10
}
```

```
#include<stdio.h>

namespace na
{
    int num1 = 10;
}

using namespace na;
void show_record( void )
{
    //printf("Num1    :   %d\n",na::num1); without using directive
    //using namespace na;
    printf("Num1    :   %d\n",num1);//10 //using directive global
}
void print_record( void )
{
    //printf("Num1    :   %d\n",na::num1); without using directive

     //using namespace na;
     printf("Num1    :   %d\n",num1); //10 //using directive global

}
void display_record( void )
{
    //printf("Num1    :   %d\n",na::num1); without using directive
     //using namespace na;
     printf("Num1    :   %d\n",num1); //10 //using directive global
}

int main()
{
    ::show_record();
    ::print_record( );
    ::display_record( );

    return 0;
}
->Here all functions are global thts why when we call function that
time use :: scope resolution operator.
```

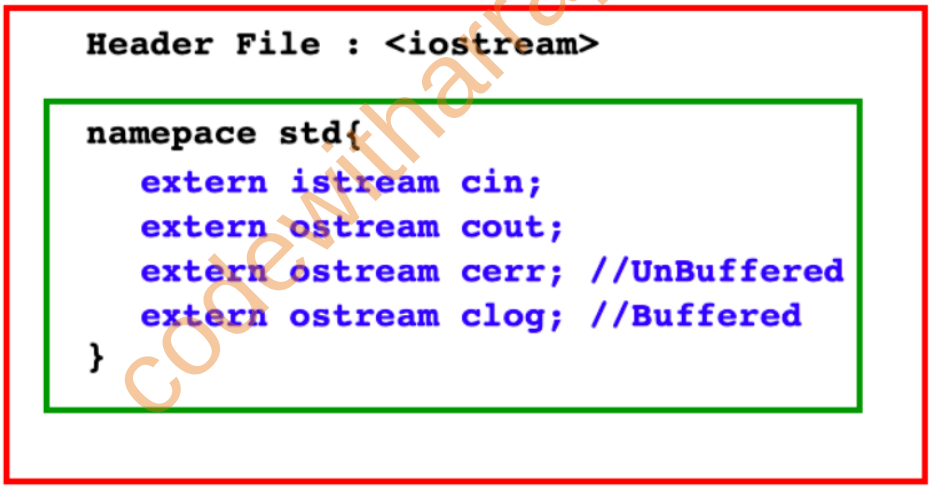
Points to remember about namespace

- We can give same /different name to the namespaces.
- To access members of namespace either we should use :: operator / using directive.
- We can not define namespace inside function/class. It should be global / it should be inside another namespace.
- We can not define main function inside namespace. It must be member of global namespace.
- We can not create object of namespace. It is used for grouping.
- std is standard namespace in C++.
- Generally name of the namespace should be in lower case.

Cin , cout , cerr , clog concept

Standard Streams Of C++

- Stream is a an abstraction(object) which is used to produce(write) and consume(read) data from source to destination.
- Console = Keyboard + Monitor.
- Following are the standard streams of C++ that is associated with console: 1. cin : input stream represents keyboard
2. cout : output stream represents monitor
3. cerr : Unbuffered standard error stream
4. clog : Buffered standard error stream



->In C++ there standard library for cin,cout,cerr,clog i.e <iostream> header file.
->Cin is used to take input from user like scanf.
->Cout is used to print data on moniter like printf.

COUT

• cout stands for character output.
• It is standard output stream of C++ which represents monitor. In other words, if we want to print state of object on console/monitor then we should use cout.
• cout is object of ostream class and ostream is nickname/alias of basic_ostream class. typedef basic_ostream ostream;
• cout is declared as extern object inside std namespace (hence std::cout) and std namespace is declared in header file.
• cout is not a function hence to read data from keyboard we must use insertion(<<)operator with it.

Print data by using cout

```
#include<iostream>

int main2()
{
    std::cout<<" Hello ";
}
```

Output:-Hello

->I don't want use std every time so i.e using namespace std; used Std is the name of namespace
-> << is insertion operator used for cout.
->Here in c++ endl is use for new line like \n in C lang.

```
#include<iostream>
int main()
{
    using namespace std;
    cout<<" Hello "<<endl;
    // endl ==> manipulator ( \n )
    return 0;
}
```

Output:-Hello

```
#include<iostream>
int main()
{
    using namespace std;
    int num1 = 10;
    cout<<num1<<endl;          //in c++
    //printf("%d",num1);       // in C

    cout<<"num1     :   "<<num1<<endl;
    //printf("num1  :   %d",num1);
    return 0;
}
Output:- 10
         Num1 : 10
```

```
#include<iostream>

int main()
{
    using namespace std;
    int num1 = 10;
    cout<<"num1     :   "<<num1<<endl;//10

    int num2 = 20;
    cout<<"num2     :   "<<num2<<endl;//20

    cout<<num1<<num2;   //10 20

    return 0;
}
```

CIN

- cin stands for character input.
- It is standard input stream of C++ which represents keyboard. In other words, if we want to read data from console/keyboard then we should use cin.

- cin is object of istream class and istream is nickname/alias of basic_istream class. typedef basic_istream istream;
- cin is declared as extern object inside std namespace (hence std::cin) and std namespace is declared in header file.
- cin is not a function hence to read data from keyboard we must use extraction(>>) operator with it.

```
-----------------------------------------------------------------
Take input from user by cin
-> >> extraction operator is used for cin to take input from user
#include<iostream>
int main()
{
    int num;
    std::cout<<"Enter the number";
    std::cin>>num;         //in c++
    //scanf("%d",&num);  //in c

    std::cout<<"Num1    :   "<<num<<std::endl;  //in c++
    //printf("Num1    :   %d",num);   //in c
    return 0;
}
-----------------------------------------------------------------
#include<iostream>
int main()
{
    using namespace std;
    int num;
    cout<<"Enter the number";
    cin>>num;
    //scanf("%d",&num);

    cout<<"Num1    :   "<<num<<endl;
    //printf("Num1    :   %d\n",num);
    return 0;
}
-----------------------------------------------------------------

#include<iostream>
int main()
{
    using namespace std;
    int num1,num2,num3;

    cout<<"Enter the num1 and num2";
    //printf("Enter the num1 and num2")
    cin>>num1>>num2;// 10 20
    //scanf("%d%d",&num1,&num2);

    cout<<num1<<num2<<endl;
    //printf("%d     %d",num1,num2);
```

```
    //    10          20

    cout<<"Enter the num3";
    //printf("Enter the num3")
    cin>>num3;
    //scanf("%d",&num3)
    cout<<num3<<endl;
    //printf("%d",num3)
    return 0;

}
```

Manipulator

- In C/C++, escape sequence is a character which is used to format the output.
- Example : '\n', '\t', '\b', '\a' etc.
- In C++, manipulator is a function which is used to format the output.
- Example : endl, setw, fixed, scientific, setprecision, dec, oct, hex etc.
- All the manipulators are declared in std namespace but header files are different
    1. To use endl include header file 2. To use remaining manipulators include header file
- Reference : https://en.cppreference.com/w/cpp/io/manip

Proper c++ program start

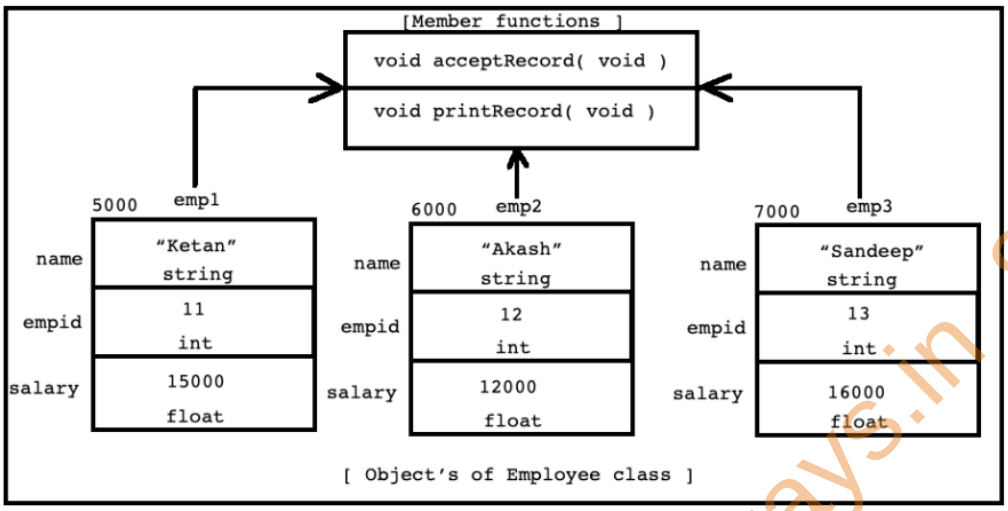
```
#include<iostream>
using namespace std;

int main()
{
    int num1;

    cout<<"Enter num1   :   ";
    cin>>num1;

    int num2;

    cout<<"Enter num2   :   ";
    cin>>num2;
```

```
    cout<<"Num1 :    "<<num1<<endl;
    cout<<"Num2 :    "<<num2<<endl;


    return 0;
}
```

Object Oriented Concepts.

• Data members of the class get space once per object according their order of declaration inside class.

• Member function do not get space inside object, rather all the objects of same class share single copy of it.



- **Characteristics of Object**
  1. **State**
     - Value/data stored inside object is called state of the object.
     - Value of the data member represent state of object.

     Example:- Ketan ,11 ,15000
  2. **Behavior**
     - Set of operations that we can perform on object is called behavior of the object.
     - Member functions defined inside class represents behavior of the object.

     Example:-value can be change by accept_record,print_record
  3. **Identity**
     - Identity is that property of an object which distinguishes it from all other objects.

     Example:-unique id here empid, college mai PRN no unique id, for mine adhar card no unique id.

- **Empty class**
- A class which do not contain any member is called empty class.
- Size of object of empty class is 1 byte.

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

};

int main()
{
    Test t;
    size_t size = sizeof(t);  //1 byte
    cout<<"Size :   "<<size<<endl;
    return 0;
}
```

Output:- Size= 1 byte

Class

Definition:

- A class is collection of data member and member function.
- Structure and behavior of the object depends on class hence class is considered as a template/model/blueprint for object.
- A class represents a set of objects that share a common structure and a common behavior.
- Class is logical or imaginary entity.

Example 1.Car 2.Book 3.Mobile Phone

- Class implementation represents encapsulation.

```
/* Example of class:-
        class book
        {
                private:
                char author[10];
                int price;
                char name[20];
```

```
                int pages;

                public:

                void accept_record()
                {
                        //TODO
                }
        }
        int main()
        {
            book b1,b2,b3,b4;
        }
    b1==> author , price ,name ,pages ( C programming)
    b2==>  author , price ,name ,pages ( C++ programming)
    b3=> author , price ,name ,pages ( java programming)
    b4=> author , price ,name ,pages ( c# programming)
*/
```

Object

Definition:

- Object is a variable / instance of class.
- Any entity which is having physical existence is called object.
- In other words, an entity which get space inside memory is called object. • An entity, which has sate, behavior and identity is called object.
- Object is physical or real time entity.

Example 1.Tata nano 2.C++ Complete reference 3.Nokia 1100

String Class

->in C++ there is string class to print string on terminal

->The size of string class is 24 bytes.

```
#include<iostream>
int main2()
{
    using namespace std;
    string str;
    size_t size  = sizeof(str); // 24 bytes
    cout<<"Size     :   "<<size<<endl;
    return 0; }
```

```
-----------------------------------------------------------
#include<iostream>
int main()
{
    using namespace std;
    string str = "Ketan";
    str = str + " " + "kore";
    cout<<"Name     :   "<<str<<endl;
    return 0;
}
Output:-Name : Ketan kore
-----------------------------------------------------------
Observe the input and output here
#include<iostream>
int main()
{
    using namespace std;

    string name;

    cout<<"Name :   ";
    cin>>name;
    cout<<"Name :   "<<name<<endl;

    return 0;
}
Input:-  Name :   ketan kore
Output:- Name :   ketan
-----------------------------------------------------------
->Here class written in namespace



#include<iostream>

#include<string>
//string is a class
using namespace std;
```

```
namespace ntest
{
        class Employee
        {
            private:
                string name;
                int empid;
                float salary;
            public:

            void accept_record( void  )
            {
                    cout<<"Name  :   ";
                    cin>>name;
                    cout<<"Empid    :   ";
                    cin>>empid;
                    cout<<"Salary   :   ";
                    cin>>salary;
            }
            void print_record( )
            {
                cout<<"Name :   "<<name<<endl;
                cout<<"Empid    :   "<<empid <<endl;
                cout<<"Salary   :   "<<salary<<endl;
            }
        };
}

int main()
{
        //ntest::Employee emp;
         using namespace ntest;
         Employee emp;
         emp.accept_record( );
         emp.print_record( );
        return 0;
}
```

```
/*
Input:- Name   :    ashok
        Empid     :    7
        Salary    :    10000
Output:-Name :    ashok
        Empid     :    7
        Salary    :    10000
*/-------------------------------------------------------
```

->Here class member function written outside the class and outside the namespace. Here only 1 object create you can create multiple object.
Refer next program for multiple object

```
#include<iostream>
#include<string>
//string is a class
using namespace std;

namespace ntest
{
        class Employee
        {
            private:
                string name;
                int empid;
                float salary;
            public:

            void accept_record( void  );
            void print_record( void );
        }; // end of class
}//end of namespace

using namespace ntest;
```

```
void Employee :: accept_record( void  )
{
    cout<<"Name  :   ";
    cin>>name;
    cout<<"Empid    :   ";
    cin>>empid;
    cout<<"Salary   :   ";
    cin>>salary;
}

void Employee :: print_record( void  )
{
    cout<<"Name :   "<<name<<endl;
    cout<<"Empid    :   "<<empid <<endl;
    cout<<"Salary   :   "<<salary<<endl;
}
int main()
{
        //ntest::Employee emp;
         using namespace ntest;
         Employee emp;

         emp.accept_record( );
         emp.print_record( );
        return 0;
}
/*
Input:- Name  :   ashok
        Empid    :   7
        Salary   :   10000
Output:-Name :   ashok
        Empid    :   7
        Salary   :   10000
*/
```

->Multiple objects program

```
#include<iostream>
#include<string>
//string is a class
using namespace std;

class Employee
{
    private:
        string name;
        int empid;
        float salary;
    public:

    void accept_record( void  );

    void print_record( void );

}; // end of class


void Employee :: accept_record( void  )
{
    cout<<"Name  :   ";
    cin>>name;
    cout<<"Empid    :  ";
    cin>>empid;
    cout<<"Salary :   ";
    cin>>salary;
}

void Employee :: print_record( void  )
{
    cout<<"Name :   "<<name<<endl;
    cout<<"Empid    :   "<<empid <<endl;
    cout<<"Salary   :   "<<salary<<endl;

}
```

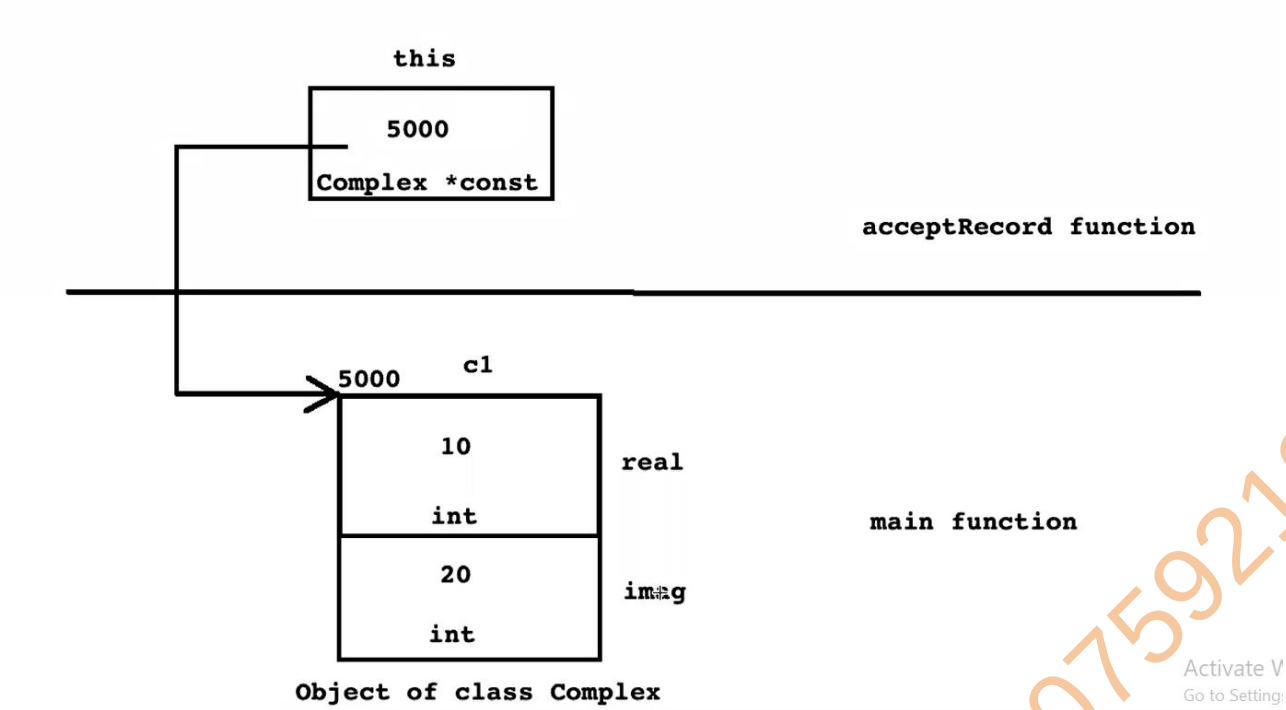
```
int main()
{
        //ntest::Employee emp;

         Employee e1,e2,e3;

        e1.accept_record( );
        e2.accept_record( );
        e3.accept_record ( );

        e1.print_record( );
        e2.print_record ( );
        e3.print_record( );
        return 0;
}
Input:-
Name   :  ashok    Name   :  vikas     Name:   niraj
Empid  :  7        Empid  :  8         Empid:  9
Salary:   1000     Salary:   2000      Salary: 3000

output:-
Name   :  ashok    Name   :  vikas     Name:   niraj
Empid  :  7        Empid  :  8         Empid:  9
Salary:   1000     Salary:   2000      Salary: 3000
*/
```

This Pointer Concept

```
#include<iostream>
using namespace std;
class Complex
{
     int real;
     int imag;
     public:
     //this = &c1
     //classname * const this;
     //Complex * const this = &c1
```

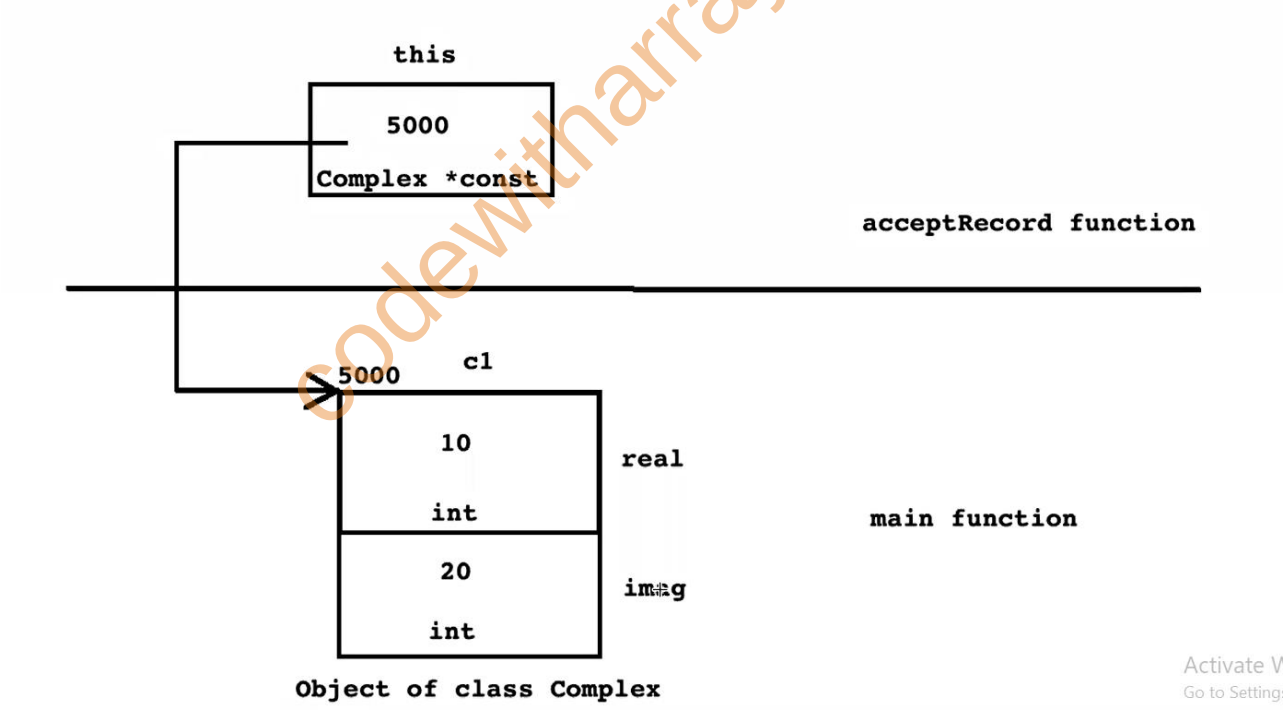
```
      void accept_record( )
      {
            cout<<"Real  :   " ;
            //cin>>real;
            cin>>this->real;  //10
            cout<<"Imag  :   ";
            cin>>this->imag;  //20
            //cin>>imag;
      }
      //this = &c1
      // this pointer is a constant pointer
      // classname * const this;
      // Complex * const this = &c1
      void print_record ( )
      {
            //cout<<"Real :   "<<real<<endl ;
            cout<<"Real :   "<<this->real<<endl ;
            //cout<<"Imag :   "<<imag<<endl;
            cout<<"Imag :   "<<this->imag<<endl;
    }
};

int main()
{
    Complex c1;

    c1.accept_record( ); //  c1.accept_record( &c1 );
    c1.print_record( ); // c1.print_record( &c1  );

}
```

->When you not used this pointer the complier explicity passed the address and explicitly catch in function observed both with and with out this pointer in above program



```
#include<iostream>
using namespace std;
class Complex
{
	int real;
	int imag;

	public:

	//classname * const this;
	// Complex * const this = &c1
	//                        10          20
	void initComplex( int real , int imag )
	{
		this->real = real;
		//10
		this->imag = imag;
		//20
	}

	//this = &c1
	// this pointer is a constant pointer
```

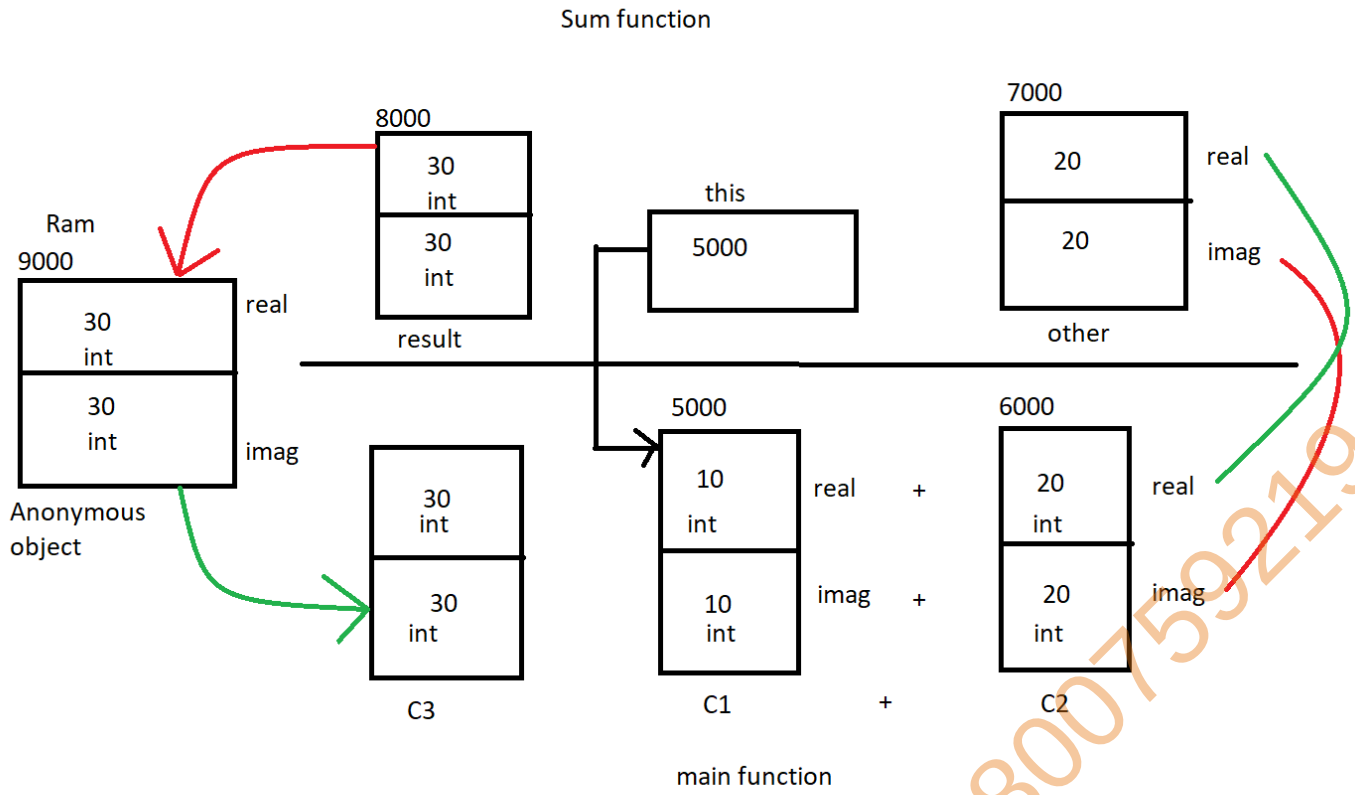
```
      // classname * const this;
      // Complex * const this = &c1
      void print_record ( )
      {
            //cout<<"Real :   "<<real<<endl ;
            cout<<"Real :   "<<this->real<<endl ;
            //cout<<"Imag :   "<<imag<<endl;
            cout<<"Imag :   "<<this->imag<<endl;
     }

};

int main()
{
       Complex c1;

       c1.initComplex(10 ,20);//c1.initComplex(&c1, 10
,20);

       c1.print_record( );//c1.print_record( &c1 );

}
```



->Here we do sum of two object by using this pointer.

```
#include<iostream>
using namespace std;
class Complex
{
      int real;
      int imag;

      public:
      //this = &c1
      //classname * const this;
      //Complex * const this = &c1
      void accept_record( )
      {
            cout<<"Real  :   " ;
            //cin>>real;
            cin>>this->real;
            cout<<"Imag  :   ";
            cin>>this->imag;
            //cin>>imag;
      }
      //this = &c1
      // this pointer is a constant pointer
      // classname * const this;
      // Complex * const this = &c3
      void print_record ( )
      {
            //cout<<"Real :   "<<real<<endl ;
            cout<<"Real :   "<<this->real<<endl ;
            //cout<<"Imag :   "<<imag<<endl;
            cout<<"Imag :   "<<this->imag<<endl;
     }

      //this = &c1;
      //other =c2;
      Complex sum( Complex other )
      {
            Complex result;
```

```
            result.real = this->real + other.real;
            result.imag = this->imag + other.imag;

            return result;
     }
};

int main()
{
    Complex c1;

    c1.accept_record( ); //  c1.accept_record( &c1 );

    Complex c2;
    c2.accept_record ( );//c2.accept_record ( &c2 );

    Complex c3;

    c3 = c1.sum( c2 );// c1.sum(&c1,c2)

    c3.print_record( ); // c3.print_record( &c3 );
}

Output:-
/* pass c1= 10,10    pass c2=20,20
        c1          +     c2          =      c3
        real(10)          real(20)    =     real(30)
        imag(10)          imag(20)    =     imag(30)
```



We can create object without name. it is called anoymous object.
If we return object from function by value then complier implicitly generates anonymous object in RAM

->Refer this diagram for above program.

->When we return object from by value then complier implicitly generates anonymous object in ram.

->Value first copy in anonymous object then after it copy in result.

->anonymous object means object created without name.

->for c2. Other object created and for result object result created.

---

this pointer:-

• Consider steps in program development

1. Understand clients requirement i.e. problem statement.

2. Depending on the requirement, define class and declare data member inside it.

3. Create object of the class.(Here data member will get space inside object)

4. To process state of the object, call member function on object and define member function inside class.

• If we call member function on object then compiler implicitly pass, address of current object as a argument to the function.
• To store address of argument(current object), compiler implicitly declare one pointer as function parameter inside member function. It is called this pointer.

• General type of this pointer is : ClassName *const this;

• this is a keyword in C++. this pointer

• Using this pointer, non static data member and non static member function can communicate with each other hence, it is considered as a link/connection between them.

• If name of the data member and name of local variable is same then preference is always given to the local variable. In this case to refer data member, we must use this keyword. Otherwise use of this keyword is optional.

• this pointer is implicit pointer, which is available in every non static member function of the class which is used to store address of current object or calling object.

• Following member function do not get this pointer:

1. Global function

2. Static member function

3. Friend function

• We can not declare this pointer explicitly. But compiler consider this pointer as a first parameter inside member function.

FUNCTION OVERLOADING

->Now I want to add two int number,add two float number,two double number so I create three different function.but the operation is same three of them but in c the function name cannot be same.It is not possible in C

->So in c++ this can be avoid by the concept of Function Overloading.

->If implementation of function is logically same/equivalent then we should give same name to the function.

1.Function overloading By number of arguments

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

// 1. number of arguments
void sum( int num1 , int num2  )// no of arguments = 2
{
        int result;
        result = num1 + num2;     //30
        cout<<"Result   :  "<<result;
}
void sum( int num1 , int num2 , int num3 )//no of argument
= 3
{

        int result;
        result = num1 + num2 + num3;  //60
        cout<<"Result   :   "<<result;
}

int main()
{
    ::sum(10,20);
```

```
    ::sum(10,20,30);
    return 0;
}
```

->Observe here two function of sum with same name and operation perform is also same.
->how to decide complier konsa function kiska hai.
->Yaha par complier arguments ke basis pe decide kr pa raha hai
->agar no of arguments bhi same rahenge isme tho fir ambiguity error aayenga.

2.Function overloading By Type of arguments

->Now in this program function overloading Type of arguments ke basis pe ho rahi hai.no arguments same hai par type of arguments are different.

```
#include<iostream>
using namespace std;
// 2. Type of arguments
void sum( int num1 , int num2  )// int int
{
        int result;
        result = num1 + num2;
        cout<<"Result   : "<<result<<endl;
}
void sum( int num1 , double num2  )// int double
{

        double result;
        result = num1 + num2 ;
        cout<<"Result   :   "<<result<<endl;
}
int main()
{
    ::sum(10,20);
    ::sum(10,20.5);
    return 0;
}
```

3.Function overloading By Order of arguments

->Now in this program function overloading order of arguments ke basis pe ho rahi hai.no arguments same hai par order of arguments are different.

```
#include<iostream>

using namespace std;
// 3. order of argument
void sum( int num1 , float num2  )
{
        float result;
        result = num1 + num2;
        cout<<"Result   :   "<<result<<endl;
}
void sum( float num1 , int num2  )
{

        float result;
        result = num1 + num2 ;
        cout<<"Result   :   "<<result<<endl;
}
int main()
{
    ::sum(10,20.5f);
    ::sum(10.5f,20);
    return 0;
}
  sum ==> 2 forms
  Function overloading
      Compile-time polymorphism     poly-many   morphism-forms
                    ( many forms )
     Ability to have more than one form
```

->Only on the basis of different return type, we can not give same name to function.

• Using above rules, Process of defining multiple functions with name is called function overloading. In other words, process of defining function with same name and different signature is called function overloading.

• Functions, which are taking part in function overloading are called overloaded function

• Function overloading is OOPS concept. It represents compile time polymorphism.

• If implementation of function is logically same/equivalent then we should overload function.

• For function overloading, functions must be exist inside same scope.

• In function overloading return type is not considered. ¬ Returning value from function and catching value which returned by function is optional hence we can not overload function.

• Except 2 functions, we can overload any global function as well as member function:

1. main function
2. destructor

->But what happened internally how the function overloading works

->The concept behind this is mangled name.

Mangled name

- If we define function in C++ then by looking toward name of the function and type of parameters passed to the function compiler implicitly generates unique name. It is called mangled name.

```
void sum( int num1, int num2 ){ //__Z3sumii
    //TODO
}
void sum( int num1, float num2 ){   //__Z3sumif
    //TODO
}
void sum( int num1, float num2, double num3 ){  //__Z3sumifd
    //TODO
}
```

- Mangled name may vary from compiler to compiler.

Name Mangling

- To generate mangled name, compiler implicitly use one algorithm. It is called name mangling.

Default Arguments concept

->My requirement is I want to add two numbers with 2 agruments
Add three numbers with 3 arguments 4 number with 4 arguments
->But in c I have to write different function for each but I want to write only one function for all of them it is not possible in C but in C++ it is possible By the concept of default arguments.

->default arguments right to left hi de sakte hai bich mai de nhi de sakte
->default arguments kuch bhi de sakte hai 0, 10,100 50 kuch bhi according to you.
->int hai tho all int lo, float hai tho flaot lo.

```
#include<iostream>
using namespace std;
void sum ( int num1,int num2=0,int num3=0,int num4=0,int num5=0)
{
    int result;
    result = num1 + num2 + num3 + num4 + num5;
    cout<<"Result   :   "<<result<<endl;
}
int main()
{

    sum(10);                    //10
    sum(10,20);                 //30
    sum(10,20,30);              //60
    sum(10,20,30,40);           //100
    sum(10,20,30,40,50);        //150
    return 0;
}
```

```
#include<iostream>
using namespace std;
void sum ( int num1,int num2=40,int num3=30,int num4=20,int
num5=10)
{
    int result;
    result = num1 + num2 + num3 + num4 + num5;
    cout<<"Result   :   "<<result<<endl;
}
int main()
{
    sum(10);
    sum(10,20);
    sum(10,20,30);
    sum(10,20,30,40);
    sum(10,20,30,40,50);
    return 0;
}
```

Output:- Observe output in both case when we change default arguments.

```
Result   :   110
Result   :   90
Result   :   90
Result   :   110
Result   :   150
```

- Default arguments are always assigned from right to left direction.
- Using default argument, we can reduce developers effort.
- We can assign default arguments to the parameters of any global function as well as member function.

Constructor Concept:-

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

class Complex
{
	int real;
	int imag;

	public:
	//this = &c1
	void initComplex(  void )
	{
		this->real = 0;
		this->imag = 0;
	}
	/*
		c1 ==> real and imag = 0
	*/

	//this = &c1
	void accept_record( )
	{
		cout<<"Real  : " ;
		cin>>this->real;  //10
		cout<<"Imag  :   ";
		cin>>this->imag;  //20
	}

	//this = &c1
	void print_record ( )
	{
		//cout<<"Real :   "<<real<<endl ;
		cout<<"Real :   "<<this->real<<endl ;
		//cout<<"Imag :   "<<imag<<endl;
		cout<<"Imag :   "<<this->imag<<endl;
    }
};
```

```
int main()
{
	Complex c1 ;

	c1.initComplex( );//c1.initComplex( &c1 )
	// real = 0  and imag = 0
	c1.print_record( ); // 0 and  0

	c1.accept_record( );//c1.accept_record(&c1);
	c1.print_record( );//c1.print_record( &c1 );

	c1.initComplex( ); //c1.initComplex(&c1 );
	c1.print_record( );//c1.print_record( &c1 );
}

/*
    int num1 = 10;
    int num1 = 20; // NOT OK initialization only once
*/
```

->By rule the initialization ek hi bar hona chahiye par above program mai initialization multiple time ho raha hai it's a big problem in this case so to avoid this problem there is one solution in C++ is called constructor.

```
#include<iostream>

using namespace std;
class Complex
{
     int real;
     int imag;

     public:
     //this = &c1
     Complex(  void ) // constructor
     {
          cout<<"Complex(  void )"<<this<<endl;
```

```
            this->real = 0;
            this->imag = 0;
      }
      /*
             c1 ==> real and imag = 0
      */

      //this = &c1
      void accept_record( )
      {
            cout<<"Real  :   " ;
            cin>>this->real;  //10
            cout<<"Imag  :   ";
            cin>>this->imag;  //20
      }

      //this = &c1
      void print_record ( )
      {
            //cout<<"Real :   "<<real<<endl ;
            cout<<"Real :   "<<this->real<<endl ;
            //cout<<"Imag :   "<<imag<<endl;
            cout<<"Imag :   "<<this->imag<<endl;
    }
};

int main()
{
        Complex c1 ; // instantation
        Complex c2;
        Complex c3;
        Complex c4;
        C1.complex(); //NOT OK
        c1.accept_record( );//c1.accept_record(&c1);
        c1.print_record( );//c1.print_record( &c1 );

}
```

```
//constructor is special member function.
//member function aur class name is same
//constructor ka koi return type nhi hota.
//use explicitly call krne ki jarurat nhi
//woh implicitly call hota hai jab aap class ka object
banaoge woh automatically call hota hai
//most imporatant constructor is called only once per
object.
//object ka initialization bhi ek hi bar hota hai.
//C1.complex(); is not okay
```

Constructor

- It is a member function of the class which is used to initialize the object.
- Due to following reasons, constructor is considered as special member function of a class:

1. Its name is same as class name.
2. It doesn't have any return type.
3. It gets called implicitly
4. It get called once per object.

- We can not call/invoke constructor on object/pointer/reference explicitly. It is designed to call implicitly.

- We can not declare constructor static, constant, volatile or virtual. We can declare constructor **inline only.**
- We can use any access specifier on constructor.

1. If constructor is public then we can create object of the class inside member function as well as non member function.
2. If constructor is private then we can create object of the class inside member function only.

- Constructor calling sequence depends on order of object declaration.

- Types of constructor

1. Parameterless constructor
2. Parameterized constructor
3. Default constructor

---

1. Parameterless constructor

- We can define constructor without parameter(parameterless). It is called parameterless constructor.

```
Complex( void ){
    this->real = 0;
    this->imag = 0;
}
```

- If we create object, without passing argument then parameterless constructor gets called.
- Complex c1; //Here on c1, parameterless constructor will call. • It is also called as zero argument constructor or user defined default constructor.

2. Parameterized constructor

- We can define constructor with parameter. It is called parameterized constructor.

```
Complex( int real, int imag ){
    this->real = real;
    this->imag = imag;
}
```

- If we create object, by passing argument then parameterized constructor gets called.
- Complex c1(10,20); //Here on c1, parameterized constructor will call

3.Default Constructor

- If we do not define constructor inside class then compiler generate one constructor for that class by default. It is called default constructor.

• Compiler generated default constructor is parameterless constructor. In other words, if we want to create object by passing arguments then we must define parameterized constructor inside class

```
class Complex{
public:
    /*
        Complex( void ){
            //Empty
        }
    */
};
```

->Example of parameterized and parameterless constructor by program

```
#include<iostream>
using namespace std;
class Complex
{
    int real;
    int imag;
    public:
    Complex( void )   //parameterless constructor
    {
            cout<<"Complex( void )"<<endl;
            this->real = 0;
            this->imag = 0;
    }

    //            this = &c2
    //             10        20
    Complex( int real, int imag)  //parameterized
constructor
    {

            cout<<"Complex( int real, int imag)"<<endl;
            this->real = real;
            this->imag = imag;
```

```
    }
    // this = &c1
    // this = &c2
    void print_record( void )
    {
            cout<<"Real :   "<<this->real<<endl;
            cout<<"Imag :   "<<this->imag<<endl;
    }
};

int main()
{
    Complex c1;
    c1.print_record( ); // c1.print_record( &c1 );

    Complex c2(10,20);
    c2.print_record( );//c2.print_record( &c2 );

    return 0;
}
/* output:-//parameterless constr   parametrized constru
             Complex( void )      Complex( int real, int
imag)
               Real :   0             Real :   10
               Imag :   0             Imag :   20*/
```

Sample MCQ questions:-

Q.1 C++ is developed by_____.
A. Alan Kay.
B. Bjarne Stroustrup.
C. James Gosling.
D. Brian Karnighan.

Q.2 C++ is invented in year _____.
A. 1972
B. 1979
C. 1983
D. 1998

Q.3 Which one of the following data type is introduced in C++.
A. void
B. char
C. wchar_t
D. wchar

Q.4 In C++, by default structure members are ____ and class members are _____.
A. private, private
B. private, public
C. public, public
D. public, private

Q.5 _____ is standard namespace in C++.
A. global namespace
B. std namespace
C. default namespace
D. system namespace

Q.6 Select the wrong statement about namespace.
A. We can define namespace as File scope.
B. We can define namespace at namespace scope.
C. we can define main function inside namespace.
D. we can not define namespace inside function.

Q.7 _____ operator is designed to use with cin.
A. <
B. <<
C. >>
D. >

Q.8 Which one of the header file is required to use manipulator.
A. iostream
B. manip
C. iomanip
D. limits

Q.9 Size of object of empty class is_____.
A. 0 byte
B. 1 byte
C. 2 bytes
D. 4 bytes.

Q.10 Select the correct statement about class.
A. State, bahavior and identity are characterictics of class.
B. Class represents structure and behavior of the object.
C. Class get space inside memory.
D. By default, access specifier of the class is private.

4.Constructor's Member Initializer List concept

```
#include<iostream>
using namespace std;
class Test
{
    private:
    int num1;
    int num2;
    int num3;
    public:
    Test ( void )
    {
            this->num3 = num2;
            this->num2 = num1;
            this->num1 = 10;
    }
    // this = &t1
    void print_record( )
    {
        cout<<"Num1 :   "<<this->num1<<endl;
        cout<<"Num2 :   "<<this->num2<<endl;
        cout<<"Num3 :   "<<this->num3<<endl;
    }
};
```

```
int main()
{
    Test t1;
    t1.print_record( );//t1.print_record(&t1);

    return 0;
}
```

->In this program the value is printed according to constructor member not data member of class i.e the output is get weird

Num1=10
Num2=garbage
Num3=garbage

So,how to avoid this problem the solution of this is Constructor's Member Initializer List

- If we want to initialize data members, according to order of data member declaration then we should use Constructor's member initializer list.
- Except array and string we can initialize any data member in Constructor member initializer list.
- In case of modular approach, Constructor member initializer list must appear in definition part.

```
class Test{
    int num1, num2, num3;
public:
    Test( void ) : num1( 10 ), num2( 20 ), num3( 30 ){
    }
    Test( int num1, int num2, int num3 ) : num1( num1 ), num2( num2 ), num3( num3 ){
    }
};
```

Constructor's Member Initializer by example program

```
#include<iostream>
using namespace std;
class Test
{
    private:
    int num1;
    int num2;
    int num3;

    public:
    // Test ( void ):num3(num2),num2(num1),num1(10)  //its
ok
    // { } // constructor member initilization list
    Test ( void ):num2(num1),num3(num2),num1(10)    //its
ok
    { } // constructor member initilization list
    Test ( void ): num1(10),num2(num1),num3(num2), //its ok
    { } // constructor member initilization list

    /*
            this->num1 = 10;
            this->num2 = num1
            this->num3 = num2
    */
    // this->num3 = num2;
    // this->num2 = num1;
    // this->num1 = 10;

    // this = &t1
    void print_record( )
    {
        cout<<"Num1 :   "<<this->num1<<endl;
        cout<<"Num2 :   "<<this->num2<<endl;
        cout<<"Num3 :   "<<this->num3<<endl;
    }
};
```

```
int main()
{
    Test t1;
    t1.print_record( );//t1.print_record(&t1);

    return 0;
}
```

Output:- 10 , 10 , 10

: ke bad hi initialized kr dena ab ye data member ke order mai print honge constructor member ki order main nhi.

->When data member is constant that time constructor member gives error. It is a rule when data member is constant we have to used constructor member initialization list.
->when data member is constant I don't change the state of the data member.
->if you willing want to change data member that time read only member error give you.

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

    private:
    const int num1;

    public:
    Test( void ):num1(10){
                                //Correct way in case of const
    }
    // Test ( void )
    // {
    //     this->num1=10;    //gives error in case of const
    // }
```

```
    //this = &t1
    void showRecord()
    {
        //++this->num1; //NOT OK
        cout<<"Num1    :   "<<this->num1<<endl;
    }

    //this = &t1
    void printRecord()
    {
        //++this->num1;// NOT OK
        cout<<"Num1    :   "<<this->num1<<endl;
    }
};

int main()
{
    Test t1;
    t1.showRecord( ); //t1.showRecord( &t1 );
    t1.printRecord( );//t1.printRecord(&t1 );
    return 0;
}
Output:-10 , 10

-------------------------------------------------------
#include<iostream>

using namespace std;
class Test
{
    int num1;
    int num2;
    mutable int count;

    public:

    Test( void ):num1(10),num2(10),count(0)
    { }
    //classname *const this
```

```
    void showRecord( void )
    {
        ++this->num1;
        ++this->num2;
        cout<<"Num1 :   "<<num1<<endl;
        cout<<"Num1 :   "<<num2<<endl;
    }
    //classname *const this = &callingobject
    //const Test * const this = &t1
    void printRecord( void ) const
    {
        //++this->num1; // NOT OK
        //++this->num2; // NOT OK
        ++this->count;
        cout<<"Num1 :   "<<num1<<endl;
        cout<<"Num1 :   "<<num2<<endl;

    }


    // this = &t1
    int getcount (  void )
    {
            return this->count;
    }

};

int main()
{
    Test t1;
    t1.printRecord( );
    t1.printRecord( );
    t1.printRecord( );

    cout<<"Count    :   "<<t1.getcount( )<<endl;

}
```

->in this program the print_record function is for printing purpose but I can easily modified the value of num1,num2,count ->To avoid this problem in C++ just give const after the function now they are constant and wont be modified.
e.g void printRecord(void) const
->but I want to modified count data so for this solution is simply use mutable keyword (e.g mutable int count) now you can easily modified the value of count but not other.
->simply give mutable jisko app modified karna chahite ho agar num1 honga tho numko mutable kro num2 honga tho usko mutable karo.

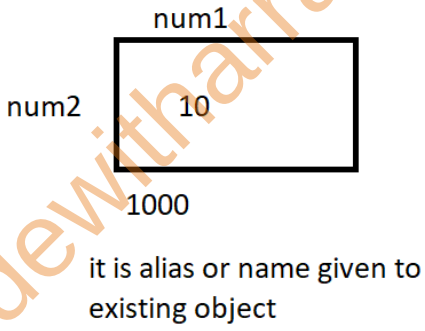
Reference concept

```
//Typedef
/*
    C
    If we want to give short or meaningfull name
    to existing datatype then we should use typedef
*/
/*
    If we want to create the alias in C++ for
    existing object we should use Reference in C++
    Reference is derived datatype  type in C++
*/
-----------------------------------------------------
#include<iostream>
using namespace std;
int main1()
{
    int num1 =10; // init
    int num2 = num1;// init
    cout<<"Num1 :   "<<num1<<"  "<<&num1<<endl;
    cout<<"Num2 :   "<<num2<<"  "<<&num2<<endl;
    return 0; }
```

```
Output:-
Num1 : 10     address in hexa decimal 0x8383bffa7c
Num2 : 10     address in hexa decimal 0x8383bffa78
```

The num1 is assigned to num2 but the address is not same in this case.

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

int main()
{
   int num1 = 10; // num1 : referent variable
   int &num2 = num1;//num2 : reference variable

   cout<<"Num1 :   "<<num1<<"  "<<&num1<<endl;
   cout<<"Num2 :   "<<num2<<"  "<<&num2<<endl;

   return 0;
}
```



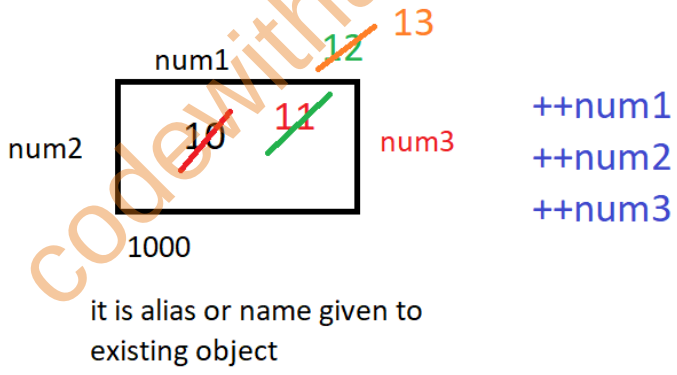
it is alias or name given to existing object


```
Output:-   Num1 : 10   0x716dfffba4
           Num2 : 10   0x716dfffba4
```

->Now,in C the address give at right side but in c++ the address operator is used at left side to assigned the value.
->This is nothing but alias created it is known as reference concept.

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

int main()
{
    int num1 = 10; // num1 : referent variable
    int &num2 = num1;//num2 : reference variable
    int &num3 = num1;

    ++num1;
    ++num2;
    ++num3;

    cout<<"Num1 :    "<<num1<<"  "<<&num1<<endl;
    cout<<"Num2 :    "<<num2<<"  "<<&num2<<endl;
    cout<<"Num2 :    "<<num3<<"  "<<&num2<<endl;

    return 0;
}
```

Output:- 13 , 13 , 13



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

int main()
{
   int num1 = 10; // num1 : referent variable
   int &num2 = num1;//num2 : reference variable
   int &num3 = num2; // OK  aloud
    // IS it reference to reference ==> NO


   ++num1;
   ++num2;
   ++num3;

   cout<<"Num1 :   "<<num1<<"  "<<&num1<<endl;
   cout<<"Num2 :   "<<num2<<"  "<<&num2<<endl;
   cout<<"Num2 :   "<<num3<<"  "<<&num2<<endl;

   return 0;
}
Output:- 13 ,13 ,13
-----------------------------------------------------------

#include<iostream>
using namespace std;

int main()
{
    //Reference is internally considered as Const pointer
    //int &num1; // NOT OK
    //int &num1 = 10 ;// NOT OK

      int num1 = 10;
      int &num2 = num1; // int const pointer

    //To minimize the complexity of pointer
    // reference is used
    // reference is automatically derefenced to
```

```
    // const pointer variable

     // C mai pass by value
     // C mai pass by address
     // C++ mai pass by reference

    return 0;
}
```

Memory allocation in C program.

1.Malloc

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

int main1()
{
    int *ptr;
    ptr = (int*)malloc(sizeof(int));

    if(ptr!=NULL)
    {
          *ptr = 125;
          cout<<"Value  :  "<<*ptr<<endl;   //125
          free(ptr); // to avoid memory leakage
          ptr = NULL; // to avoid dangling pointer
    }
    else
        cout<<"bad memory allocation"<<endl;
    return 0;
}
```

2.calloc

```
#include<cstdlib>
#include<iostream>

using namespace std;
```

```
int main2()
{
    int *ptr;

    ptr = (int*)calloc(1,sizeof(int));

    if(ptr!=NULL)
    {
          *ptr = 125;
          cout<<"Value  :   "<<*ptr<<endl; //125
          free(ptr); // to avoid mem leakage
          ptr = NULL; // to avoid dangling pointer
    }
    else
        cout<<"bad memory allocation"<<endl;
    return 0;
}
```

3.Realloc

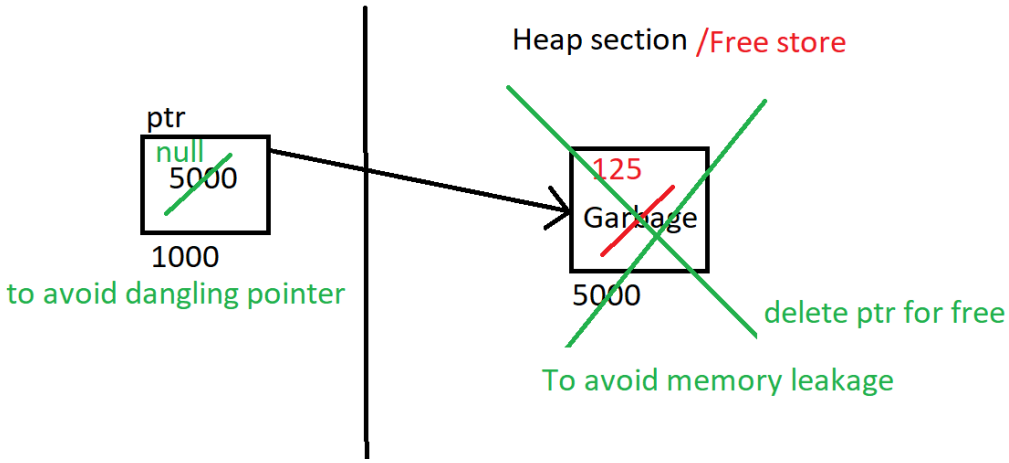
```
#include<cstdlib>
#include<iostream>

using namespace std;

int main()
{
    int *ptr;

    ptr = (int*)calloc(3,sizeof(int));

    ptr[0] = 10;
    ptr[1] = 20;
    ptr[2] = 30;

    ptr=(int*)realloc( ptr,5*sizeof(int));

    if(ptr!=NULL)
    {
          ptr[3]= 40;
```

```
            ptr[4]= 50;

            for(int index = 0;index<5;index++)
                 cout<<ptr[index]<<endl;

            free(ptr); // to avoid mem leakage
            ptr = NULL; // to avoid dangling pointer
    }
    else
        cout<<"bad memory allocation"<<endl;
    return 0;
}
```

Memory allocation in C++

```
#include<iostream>
using namespace std;
int main1()
{
    // new is an operator
    int *ptr = new int;// Memory allocation

    *ptr = 125;
    cout<<"Value    :   "<<*ptr<<endl;

    delete ptr; // Deallocation
    // TO avoid mem leakage

    ptr = NULL; // To avoid dangling pointer

    return 0;
}
```

->In C++ the new operator is used for memory allocation and for deallocation delete operator is used in c free is used.
->after ptr=null; kar do to avoid dangling pointer.



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

int main()
{
    // new is an operator
    int *ptr = new int[3];// Memory allocation

    ptr[0] = 10;
    ptr[1] = 20;
    ptr[2] = 30;
     for(int index = 0;index<3;index++)
               cout<<ptr[index]<<endl;

    delete[] ptr; // Deallocation
    // free(ptr);
    // TO avoid mem leakage

          ptr = NULL; // To avoid dangling pointer
    return 0;
}
```

->Now if you want more blocks simply do new int []; and for deallocation used delete[]=ptr; remain all is same.

->In malloc if we create object using malloc then constructor will not called.

Inline function

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

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

int main()
{
    int a = 10, b=20;
    int result = max ( a ,b );
    cout<<"Result   :   "<<result<<endl;
    return 0;
}
```

->in above program when user called function then function executed and the function activation record generate on stack after completion the activation record may get destroyed.
->it will take too much time for complier that's why in c++ there is a keyword called inline function.
->bcoz of inline function activation record not created the code is replace with function call at the time of compilation.
->But you just request for inline function this will perform or not its upto on complier you cannot forced to do this operation.

```
#include<iostream>
using namespace std;
inline int max( int x , int y)
{
    return x > y ? x : y;
}
```

```
int main()
{
    int a = 10, b=20;
    int result = max ( a ,b );
    //return x > y ? x : y;
    cout<<"Result   :   "<<result<<endl;
    return 0;
}
```

->observe now in this program you may get clearity.

• In simple words, if we call function then compiler generates function activation record.
• FAR get generated per function call.
• Due to FAR, giving call to the function is considered as overhead to the compiler.
• If we want to avoid this overhead then we should declare function inline.
• inline is keyword in C++.
• macro is command to the pre-processor but inline is request to the compiler.
• In following conditions, function can not be considered as inline.
1. Use of loop(for/while) inside function
2. Use of recursion.
3. Use of jump statements.
• Excessive use of inline function increases code size.
• Except main function, we can declare any global function inline. • We can not separate inline function code into multiple files.

Sample Question:-

1.What is the general syntax for accessing the namespace variable?
a) namespace::operator
b) namespace.operator
c) namespace#operator
d) namespace$operator

2.Which keyword is used to access the variable in the namespace?
a) using
b) dynamic
c) const
d) static

3.Which of the following is a properly defined structure?
a) struct {int a;}
b) struct a_struct {int a;}
c) struct a_struct int a;
d) struct a_struct {int a;};

4.Which is used to define the member of a class externally?
a) :
b) ::
c) #
d) !!$

5.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; };

6.The data members and functions of a class in C++ are by default _____
a) protected
b) private

c) public
d) public & protected

7.An inline function is expanded during _____
a) compile-time
b) run-time
c) never expanded
d) end of the program

8.
```
#include <iostream>
using namespace std;
namespace Box1
{ int a = 4;}
namespace Box2
{
    int a = 13;
}
int main ()
{
    int a = 16;
    Box1::a;
    Box2::a;
    cout << a;
    return 0;
}
```
A.4
B.13
C.16
D.error

Exception Handling

```
#include<iostream>
#include<string>

using namespace std;
void accept_record(string message,int &number)
```

```
{
        cout<<message;
        cin>>number;
}

int main()
{
    cout<<"Open the connection"<<endl;
    int num1;
    accept_record("Num1 :   ",num1);


    int num2;

    accept_record("Num2 :   ",num2);

    int result = num1 / num2;

    cout<<"Result   :   "<<result<<endl;


    cout<<"Close the connection"<<endl;

    return 0;
}
```

```
Input:- 10  then 2                  input:- 10 then 0
Output:-Open the connection         Open the connection
        Num1=10                     Num1=10
        Num2=5                      Num2=0
        Close the connection
```

->here problem is first function ko hum value passed kr rahe hai uske bad second function ko value pass kr rahe hai agar second functon mai 0 chodkr value pass ki tho result aa raha hai but 0 pass krne se program abnormaly terminate ho raha hai aur niche ka code execute hi nhi ho raha hai.

->think agar program 1000 line ka hai aur kisi jagah mistake ho gayi tho uske niche ka program execute nhi honga it's a big problem
->agar c lang hoti tho if else lagakr problem solve ho jata.
->in C++ there solution is Exception handling concept.

->we will see in next problem how to solve.

Exception Handling

- **Definition**

1. Runtime error is also called as exception.
2. Exception is an object which is used to send notification to the end user of the system if any exceptional situation occurs in the program.

- Need of exception handling

1. If we want to manage OS resources carefully in the program we should handle exception.
2. If we want to handle all the runtime errors at single place so that we can reduce maintenance of the application.

- How to handle exception

•In C++, we can handle exception using 3 keywords:
    1. Try 2. Catch 3. throw

- **try**

1. try is a keyword in C++.
2. If we want to keep watch on group of statements then we should use try block.
3. try block is also called as try handler.
4. We can not define try block after catch block/handler.
5. try block must have at least one catch block.

- **throw**

1. throw is keyword in C++.
2. Exception can be raised implicitly or we can generate it explicitly.

3. If we want to generate exception explicitly then we should use throw keyword.
4. throw statement is a jump statement
5. Jump statements in C/C++ : break, return, goto, continue, throw

- **catch**

1. catch is keyword in C++.
2. If we want to handle exception thrown from try block then we should use catch block.
3. Catch block is also called as catch handler.
4. Single try block may have multiple catch blocks.
5. We can not define catch block before try block/handler.
6. For thrown exception, if matching catch block is not available then C++ runtime environment implicitly give call to the std::terminate() function which implicitly calls std::abort( )function.
7. A catch block, which can handle all type of exception is called default/generic catch block. E.g. catch(. . .){ }
8. We must define generic catch block after all specific catch block

---

```
#include<iostream>
#include<string>
using namespace std;
void accept_record(string message,int &number)
{
        cout<<message;
        cin>>number;
}
int main()
{
    cout<<"Open the connection"<<endl;

    int num1;
    accept_record("Num1 :   ",num1);
    int num2;
    accept_record("Num2 :   ",num2);//0
```

```
    try
    {
            if(num2==0)
            {
                    throw 0;
            }
            else
            {
                     int result = num1 / num2;

                      cout<<"Result   :   "<<result<<endl;
            }
    }
    catch( int ex)
    {
            cout<<"Int exception"<<endl;
    }
    cout<<"Close the connection"<<endl;
    return 0;
}
-------------------------------------------------------------
--
/* output:-
Open the connection
Num1 :   10
Num2 :   0
Int exception
Close the connection
*/
-------------------------------------------------------------
```

->multiple Catch by throw

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

void accept_record(string message,int &number)
{
        cout<<message;
        cin>>number;
}
```

```
int main()
{

    cout<<"Open the connection"<<endl;

    int num1;
    accept_record("Num1 :   ",num1);

    int num2;
    accept_record("Num2 :   ",num2);//0

    try
    {
        if(num2==0)
            {
                throw "/ by zero exception";
                throw 0;
            }
        else
            {
                int result = num1 / num2;

                cout<<"Result   :   "<<result<<endl;
            }
    }
    catch( int ex)
    {
            cout<<"Int exception"<<endl;
    }
    catch( const char *ex)
    {
            cout<<ex<<endl;
    }
    cout<<"Close the connection"<<endl;
    return 0;
}
```

->If thrown exception is not available in catch block and you don't know ki konsa exception aayenga so generic catch/default catch block likh sakte hai
->generic catch block last mai likhte hai
-> syntax is catch(...)

```
#include<iostream>

#include<string>
using namespace std;


void accept_record(string message,int &number)
{
        cout<<message;
        cin>>number;
}
int main()
{
    cout<<"Open the connection"<<endl;

    int num1;
    accept_record("Num1 :    ",num1);
    int num2;
    accept_record("Num2 :    ",num2);//0

    try
    {
        if(num2==0)
            {
            throw "/ by zero exception";
           //throw 0;
            }
        else
            {
            int result = num1 / num2;
            cout<<"Result   :   "<<result<<endl;
            }
```

```
    }
    catch( int ex)
    {
        cout<<"Int exception"<<endl;
    }
    // catch( const char *ex)
    // {
    //          cout<<ex<<endl;     // }
    catch(...)// generic catch block / default
    {
            cout<<"Generic catch block";
    }
    cout<<"Close the connection"<<endl;
    return 0;
}
```

Destructor

```
#include<iostream>
#define size 3
using namespace std;

class Array
{
    private:
    int arr[size];

    public:
    // this = &a1
    void accept_record()
    {
        for(int index=0;index<size;index++)
        {
            cout<<"Enter the element";
            cin>>this->arr[index];
        }
    }
```

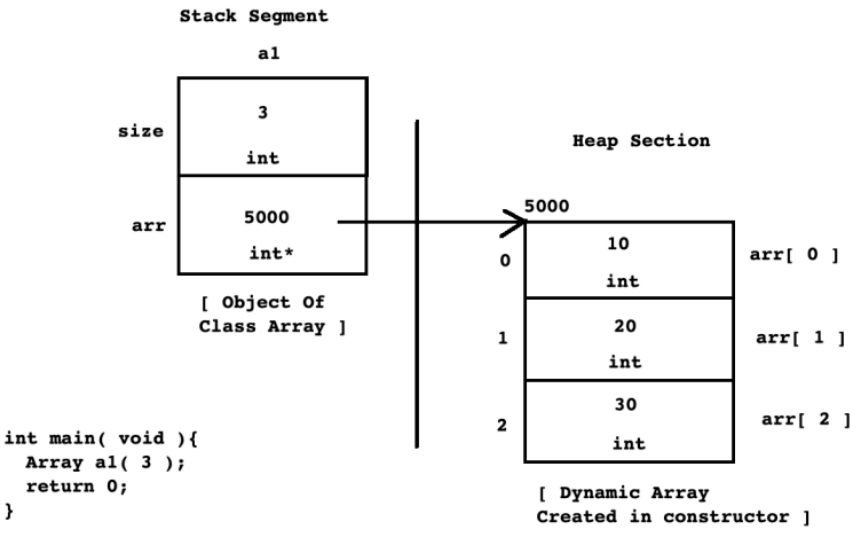
```
    void print_record()
    {
        for(int index=0;index<size;index++)
            {
            cout<<this->arr[index]<<endl;
            }
    }
};
int main()
{
    Array a1,a2,a3;
   // Array a1(2),a2(3); //not possible bcoz array size
fixed
    a1.accept_record( );//a1.accept_record( &a1 );
    a1.print_record( );//a1.print_record(&a1);
}
```

->In above program ek array create kiya hai aur mai use accept_record , print_record kr kr raha hu but here one problem is here the size of array is fixed in this case I cannot send to arrays different sizes so to solve this problem in c++ the concept is Destructor.

Destructor program:-

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

class Array
{
    private:
    int size;
    int *arr;


   public:
    // this = &a1
    Array( int size)
    {
```

```
                cout<<"Array( int size)"<<this<<endl;
                this->size = size;
                this->arr = new int [size];
        }
        // this = &a1
        void accept_record()
        {
                for(int index=0;index<size;index++)
                {
                        cout<<"Enter the element";
                        cin>>this->arr[index];
                }
        }
        void print_record()
        {
             for(int index=0;index<size;index++)
                {
                        cout<<this->arr[index]<<endl;

                }
        }
        ~Array( ) // Destructor
        {
                cout<<"~Array() "<<this<<endl;
                delete [ ] this->arr;
                this->arr = NULL;
        }
};

int main()
{
        Array a1(3),a2(5);

        a1.accept_record( );//a1.accept_record( &a1 );
        a1.print_record( );//a1.print_record(&a1);


}
```



->now I can send the arrays different size and to clear the arrays on heap section we can write on function and called that but the problem is if user may get called multiple time then program get abnormally terminate.
->that's why we used here Destructor.destructor called self at the last of program termination only once. Destructor has same name of class.
-> destructor syntax is ~ operator.

->array a1(2); array a1(3); array a1(4);
The constructor call is a1 , a2 , a3 But destructor call is exactly
Opposite Destructor call is a3, a2, a1.

- Destructor is a member function of a class which is used to release resources hold by the object.
- If we do not define destructor inside class then compiler provide one destructor for that class by default. It is called default destructor.
- Default destructor do not perform any clean up operation on data member declared by the programmer.
- Destructor calling sequence is exactly opposite of constructor calling sequence.
- We can not declare destructor static, constant or volatile but we can declare destructor inline and virtual.

• Due to following reasons, destructor is considered as special member function of the class:
1. Its name is same as class name and always precedes with ~ operator.
2. It doesn't have a return type.
3. It doesn't have any parameters.
4. We can call destructor on object/pointer/reference explicitly. But it is designed to call implicitly.

• We can overload constructor but we can not overload destructor. 1.To overload any function either number of parameters must be different or if number of parameters are same then type of parameters must be different or order of type of parameters must be different.
2.Since destructor do not take any parameter, we can not overload destructor.
• If constructor and destructor are public then we can create object of the class inside member function as well as non member function.
• If constructor or destructor is private then we can create object of the class inside member function only.

Copy Constructor

• It is parameterized constructor of a class which take single parameter of same type as a reference.
• Job of copy constructor is to initialize object from existing object.
• Since copy constructor take single parameter it is considered as parameterized constructor.
• If we do not define copy constructor inside class then compiler generates default copy constructor for the class. By default it creates shallow copy.
• Copy constructor gets called in five conditions
1. If we pass object as a argument to the function by value then its copy gets created in function parameter. On function parameter copy constructor gets called.

2. If we return object from function by value then its copy gets created inside in memory(anonymous object). On anonymous object, copy constructor gets called.
3. If we initialize object from another object of same class then on newly created object copy constructor gets called.
4. If we throw object then its copy gets created on environmental stack. Compiler invoke copy constructor on object created on stack.
5. If we catch object by value then on catching object copy constructor gets called.

Syntax	Example

//ClassName &other = source object;
//ClassName *const this = address of destination object;
ClassName(const ClassName &other){
//TODO : Shallow / Deep Copy
}
	//Complex &other = c1;
	//Complex *const this = &c2;
	Complex(const Complex &other){
	//Shallow Copy
	this->real = other.real;
	this->imag = other.imag;
	}

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

class Complex
{
    private:
    int real;
    int imag;

    public:

    Complex( void )
    {
        cout<<"Complex( void )"<<endl;
        this->real = 0;
        this->imag = 0;
    }
    // const complex &rhs = c1
```

```
    // Complex * const this = &c2
    Complex( const Complex &rhs )  //copy constructor
    {
        cout<<"Complex( const Complex &rhs )"<<endl;
        this->real = rhs.real;
        this->imag = rhs.imag;
    }note:-rhs must passed with reference in case of copy
constr

    // this = &c1
    Complex ( int real , int imag )
    {
        cout<<"Complex ( int real , int imag )"<<endl;
        this->real = real;
        this->imag = imag;
    }
    void print_record( void )
    {
            cout<<"Real :   "<<this->real<<endl;
            cout<<"Imag :   "<<this->imag<<endl;
    }
};

int main()
{
    Complex c1(10,20);
    Complex c2  = c1 ; // init

    c2.print_record( );
}

->above program is Copy constructor.
```

Shallow Copy

• Process of copying contents of object into another object as it is, is called shallow copy.
• Shallow copy is also called as bitwise copy / bit-by-bit copy. • Compiler by default creates shallow copy

```
int num1 = 10;
int num2 = num1;     //Shallow Copy

Complex c1( 10, 20 );
Complex c2 = c1;        //Shallow Copy
```

Deep Copy

• Conditions to create deep copy

1. Class must contain at least one pointer type data member.

2. Class must contain user defined destructor.

3. We must create copy of the object.

• Steps to create deep copy

1. Copy the required size(length, row/col/ array size etc) from source object into destination object.

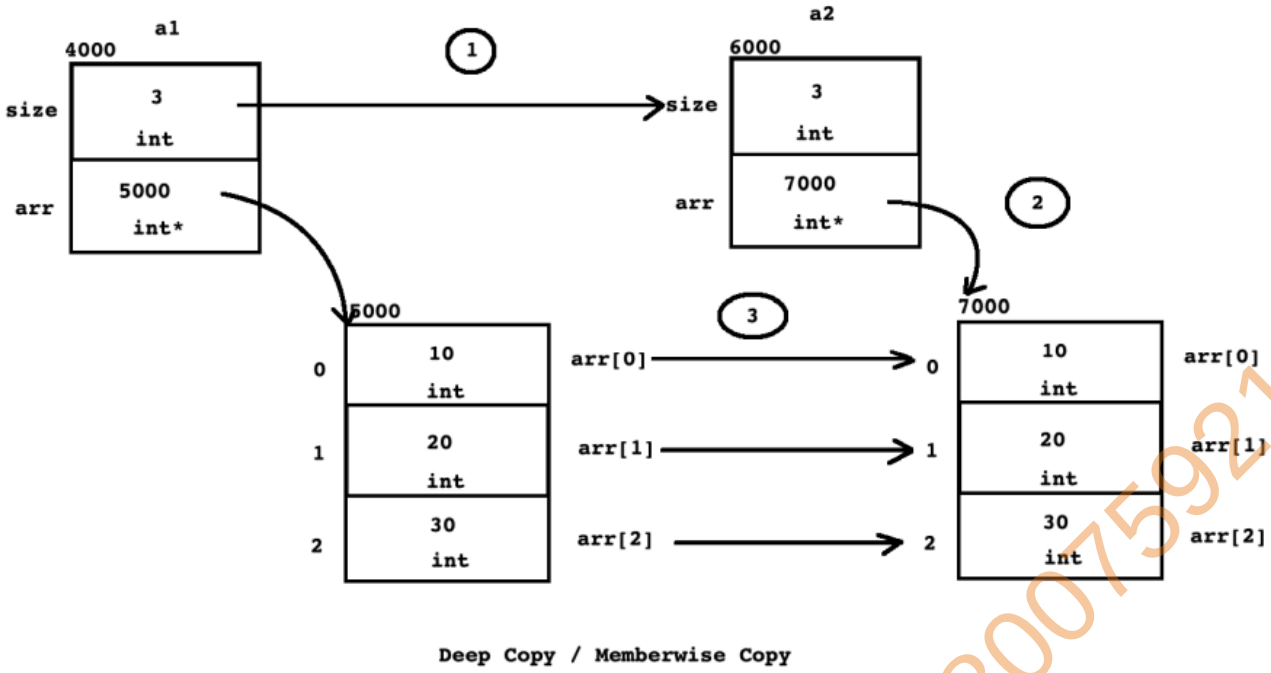
2. Allocate new resource for destination object.

3. Copy the contents from resource of source object into resource of destination object.

• Location to create deep copy

1. In case of assignment, we should create deep copy inside assignment operator function.

2. In rest of the conditions, we should create deep copy inside copy constructor.



Deep Copy / Memberwise Copy


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

class Array{
private:
    int size;
    int *arr;
public:
    Array( int size ){
        this->size = size;
        this->arr = new int[ size ];
    }
    //const Array &other = a1
    //Array *const this = &a2
    Array( const Array &other ){
        this->size = other.size;//step1;-copy the size
        this->arr = new int[ this->size ];//step2 allocate memory
        for( int index = 0; index < this->size; ++ index )//step3
copy the value
            this->arr[ index ] = other.arr[ index ];
    }
    //Array *const this = &a1
    void acceptRecord( void ){
```

```
        for( int index = 0; index < size; ++ index ){
            cout<<"Enter element    :   ";
            cin>>this->arr[ index ];
        }
    }
    //Array *const this = &a1
    void printRecord( void ){
        for( int index = 0; index < size; ++ index )
            cout<<this->arr[ index ]<<endl;
    }
    //Array *const this = &a1
    ~Array( void ){ //Destructor
        delete[] this->arr;
        this->arr = NULL;
    }
};
int main( void ){
    Array a1( 3 );

    a1.acceptRecord( ); //a1.acceptRecord( &a1 );
    Array a2 = a1;
    a2.printRecord( );  //a1.printRecord( &a1 );
    return 0;
}
->Theory hi ati hai exam mai code nhi aya tho bhi chalenga.
-----------------------------------------------------------

#include<iostream >
using namespace std;
class Test
{
    private:
    int num1;
    protected:
    int num2;
    public:
    Test( )
    {
        this->num1 = 10;
        this->num2 = 20;
    }
};
```

```
int main()
{
    Test t1;
    cout<<"Num1 :   "<<t1.num1<<endl;
    cout<<"Num2 :   "<<t1.num2<<endl;
    return 0;
} //Output:-Error protected.
```

->here num1 and num2 is protected and private so I cannot used in main function.but I want to used them then there is a solution in c++ is Friend function.friend is keyword in c++.

```
#include<iostream >

using namespace std;
class Test
{
    private:
    int num1;
    protected:
    int num2;
    public:
    Test( )
    {
        this->num1 = 10;
        this->num2 = 20;
    }
    friend int main();
};

int main()
{
    Test t1;
    cout<<"Num1 :   "<<t1.num1<<endl;
    cout<<"Num2 :   "<<t1.num2<<endl;
    return 0;
}
Output:-
Num1 :   10
Num2 :   20
```

->Now when I used friend keyword then my num1 and num2 is accessible.
->Main() ko friend bana sakte hai.
->below another example of friend program.

```
#include<iostream >
using namespace std;
class Test
{
    private:
    int num1;
    protected:
    int num2;
    public:
    Test( )
    {
        this->num1 = 10;
        this->num2 = 20;
    }

    friend void print( void );
};

void print( void )
{
    Test t1;  //object create karne ke bad hi num1 num2 access honge
    cout<<"Num1 :   "<<t1.num1<<endl;
    cout<<"Num2 :   "<<t1.num2<<endl;
}

int main()
{
    Test t1;
    ::print( );

    return 0;
}
```

->Sirf friend karne se value access nhi honge usme class ka object create karna padta hai uske bad hi value access honge.

```
#include<iostream>
using namespace std;
class A
{
    private:
    int num1;
    public:
    A( void  )
    {
        this->num1 = 10;
    }
    friend void sum( );
};
class B
{
    private:
    int num2;
    public:
    B( void  )
    {
        this->num2 = 20;
    }
    friend void sum( );
};
// Global function
void sum( )
{
        A a;  // parameterless constructor of A class is getting called
        B b;  //// parameterless constructor of B class is getting called
        int result;
        result = a.num1 + b.num2;
        cout<<"result   :   "<<result<<endl;
}
```

```
int main()
{
    ::sum( );
    return 0;
}
```

Output:-Result : 30

->Class A ke ander num1 hai and Class B ke ander num2 hai
->ab muze ek sum function maid dono chahiye addition karne ke liye. Isliye class A mai jakr sum function ko friend kr dunga.
Aur class B mai bhi friend kr dunga.
->uske bad sum function mai object create krna padenga class A aur class B ka.
->sum global function hai tho use object pr call nhi kar sakta isliye scope resolution operator used honga.

```
#include<iostream>
using namespace std;
class A
{
    public:
    void sum(  void );
};
class B
{
    int num1;
    int num2;
    public:
    B(void); //constructor
    friend void A::sum( void);
};
B::B(  void )//constructor inti...
{
    this->num1 = 10;
    this->num2 = 20;
}
```

```
void A::sum( void )
{
        B obj; //create object
        int result;
        result = obj.num1 + obj.num2;
        cout<<"Result   :   "<<result<<endl;
}

int main()
{
    A obj;
    obj.sum( );
    return 0;
}
```

->Yaha par Class A mai hi sum ka member function banaya hai
->aur class B mai ek constructor banaya hai parameterless aur uska definition class b ke bahar likhaa hai.
->now requirement is class B ke num1,num2 ki addition muze class A ke sum member function mai karni hai
->isliye Class B mai jakar class A ke sum member function ko friend bana diya friend void A::sum(void); this is syntax
->ab sum function mai class b ka object create karna padenga.object create hote hi who constructor ko called karena aur operation perform honga.
->Ab main mai sum function ko called karna hai member function hai isliye object pe call honga class A ke.

```
#include<iostream>
using namespace std;
class A
{
    public:
    void sum( void  );
    void sub( void  );
    void multiplication( void  );
```

```
};

class B
{
    int num1;
    int num2;
    public:

    B( void );
    friend class A;
};
B::B(  void ) // ctor
{
    this->num1 = 10;
    this->num2 = 20;
}

void A::sum( void )
{
        B obj;
        int result;
        result = obj.num1 + obj.num2;
        cout<<"Result   :   "<<result<<endl;
}

void A::sub( void )
{
        B obj;
        int result;
        result = obj.num1 - obj.num2;
        cout<<"Result   :   "<<result<<endl;
}
void A::multiplication( void )
{
        B obj;
        int result;
        result = obj.num1 * obj.num2;
        cout<<"Result   :   "<<result<<endl;
}
```

```
int main()
{
    A obj;
    obj.sum( );
    obj.sub( );
    obj.multiplication( );
    return 0;
}
```

->Now here class A mai 3 member function hai sum,sub, Multiplication. Aur class B mai num1 ,num2 hai aur uspar operation perform karna hai.
->ab sabko jakar friend karna paadenga isase accha directly class A ko hi class B mai jakar friend kar do.
->and remaining thing is same as above program explanation.

Sample Questions:-

Q.1 Which of the following function get this pointer
A. Global function
B. Static member function
C. Constant member function
D. Friend function

Q.2 Select the correct statement about constructor
A. We can call constructor on pointer/reference and object explicitly.
B. We can declare constructor static, constant, volatile and virtual.
C. We can declare constructor private/protected/public.
D. Default return type of constructor is integer.

Q.3 Which one of the following function compiler do not generate?
A. Parameterless constructor.

B. Parameterized constructor
C. Copy constructor
D. Assignment operator function.

Q.4 Choose Correct statement about function overloading
A. For function overloading function must be exist in different scope.
B. We can not overload constructor but we can overload destructor.
C. We can overload static and constant member functions.
D. Function overloading represents runtime polymorphism.

Q.5 Inside constant member function, if we want to modify state of non constant data member then we should use _____ keyword?
A. static
B. immutable
C. mutable
D. mutator

Q.6. Select correct statement about reference
A. We can store null value inside reference.
B. We can create reference to reference.
D. We can create reference to array.

Q.7. If we want to generate new exception then we shuld use _____ keyword?
A. try
B. catch
C. throw
D. throws

Q.8 For thrown exception, if we do not define catch block then_____
A. Compiler generates error.
B. C++ runtime invoke unexpected function.

C. C++ runtime invoke terminate function.
D. C++ runtime invoke abort function.

Q.9 Choose correct statement.
A. Macro is request to the preprocessor whereas inline is command to the compiler.
B. In case of separation, we can specify default arguments in definition part only.
C. We can not initialize array using member intializer list.
D. Return type is considered in function overloading.

Q.10 In Which one of the following condition, copy constructor do not call?
A. If we pass object do the function by value.
B. If we return object from function by value.
C. If we initialize object from same object.
D. If we assign object of same class to the another.

Operator Overloading

```
#include<iostream>

using namespace std;
int main()
{
    struct point
    {
        int x,y;

    };
    struct point p1 = { 10, 20 };
    struct point p2 = { 10, 20 };
    struct point p3;
    // p3 = p1 + p2;   //not ok
    // c = a + b // a,b,c ==> int
    p3.x = p1.x + p2.x;
    p3.y = p1.y + p2.y;
```

```
    return 0;
}
```

->in this program we cannot directly add p3=p1+p2. We have to do this like p3.x=p1.x+p2.x but I want solve this problem so there is concept in c++ called Operator overloading.

- **Operator overloading using member function:**

```
#include<iostream>

using namespace std;
class Complex
{
    int real;
    int imag;

    public:
    Complex( void )
    {
        this->real = 0;
        this->imag = 0;
    }
    Complex (int real , int imag)
    {
        this->real = real;
        this->imag = imag;
    }

    void print_record( void )
    {
        cout<<"Real :   "<<this->real<<endl;
        cout<<"Imag :   "<<this->imag<<endl;
    }

    // this = &c1;
    // other = c2;
    Complex operator+( Complex other  )
    {
```

```
            Complex result;
            result.real = this->real + other.real;
            result.imag = this->imag + other.imag;
            return result;
    }


    Complex operator-( Complex other  )
    {
            Complex result;
            result.real = this->real - other.real;
            result.imag = this->imag - other.imag;
            return result;
    }
};


int main()
{
    Complex c1(10,20);
    Complex c2(20,30);
    Complex c3;

    c3 = c1 + c2;// c3 = c1.operator+( c2 )
     c3 = c1 - c2;//      c1.operator-( c2 )
    c3.print_record();

      // c = a + b  // a and b were int

    return 0;
}
```

->now I can easily add like c3=c1+c2.
-> operator overloading member function mai internally passed like c1.operator+(c2), c1.operator-(c2) aise hota hai
 ->member function ka declaration aise hota hai operator overloading krte samay

- Complex operator+(complex other)
- Complex operator-(complex other)

- **Operator overloading using non-member function:**

```
#include<iostream>
using namespace std;
class Complex
{
    int real;
    int imag;

    public:
    Complex( void )
    {
        this->real = 0;
        this->imag = 0;
    }
    Complex (int real , int imag)
    {
        this->real = real;
        this->imag = imag;
    }

    void print_record( void )
    {
        cout<<"Real :   "<<this->real<<endl;
        cout<<"Imag :  "<<this->imag<<endl;
    }

    friend Complex operator+( Complex c1 , Complex c2  );
};
Complex operator+( Complex c1 , Complex c2  )
{
        Complex result;
        result.real = c1.real + c2.real;
        result.imag = c1.imag + c2.imag;
        return result;
}
```

```
int main()
{
    Complex c1(10,20);
    Complex c2(20,30);
    Complex c3;
    //c3 = c1 + c2; // c1.operator+(c2); //As a member
    c3 = c1 + c2;  // c3 = operator+(c1,c2);//As a non
member
    return 0;
}
```

->ab isme non member function ko operator overload karna hai sirf class mai jakar usse friend kardo.
->non member function mai internally call c3 = operator+(c1,c2);
aise jar aha hai.
-> non member function ka declaration aise dete hai
Complex operator+(Complex c1 , Complex c2)

Operator Overloading:-

- In C/C++, we can use operator with the variable fundamental type directly.
- In C language, we can not use operator with objects of user defined type directly/indirectly.
- In C++ language, we can not use operator with objects of user defined type(struct/class) directly.
- In C++, if we want to use operator with objects of user defined type then we should overload operator.
- Overloading operator means defining operator function.
- operator is keyword in C++.
- We can define operator function using 2 ways:
  1.Member function 2. Non member function.
- By defining operator function, it is possible to use operator with the objects of user defined type. This process of giving extension to the meaning of the operator is called operator overloading

Limitations Of Operator Overloading

• We can not overload following operators using **member function** as well as non member function:

1. Dot / Member selection operator(.)
2. Pointer to member selcetion operator(.*)
3. Scope resolution operator(::)
4. Ternary / Conditional operator(? :)
5. sizeof operator
6. typeid operator
7. static_cast operator
8. dynamic_cast operator
9. const_cast operator
10. reinterpret_cast operator

Limitations Of Operator Overloading

• We can not overloading following operators using **non member** function.

1. Assignment operator(=)
2. Index / subscript operator([ ])
3. Call / function call operator[( )]
4. Arrow operator(->)

• Using operator overloading, we can change meaning of the operator.

• We can not change number of parameters passed to the operator function.

• Operator overloading using member function:

```
class Complex{
    int real, imag;
public:
    Complex operator+( Complex other ){
        Complex result;
        result.real = this->real + other.real;
        result.imag = this->imag + other.imag;
        return result;
    }
}
c3 = c1 + c2;    //c3 = c1.operator+( c2 );
```

• Operator overloading using non member function:

```
class Complex{
    int real, imag;
public:
    friend Complex operator+( Complex c1, Complex c2 );
}
Complex operator+( Complex c1, Complex c2 ){
    Complex result;
    result.real = c1.real + c2.real;
    result.imag = c1.imag + c2.imag;
    return result;
}
c3 = c1 + c2;    //c3 = operator+( c1, c2 );
```

Assignment operator concept:-

```
#include<iostream>

using namespace std;
class Complex
{
    int real;
    int imag;

    public:
    Complex( void )
    {
        cout<<"Complex( void )"<<endl;
        this->real = 0;
        this->imag = 0;
    }
    Complex (int real , int imag)
    {
        cout<<"Complex (int real , int imag)"<<endl;
        this->real = real;
        this->imag = imag;
    }
    void print_record( void )
    {
        cout<<"Real :  "<<this->real<<endl;
        cout<<"Imag :  "<<this->imag<<endl;
    }
    //&other=c1
    //this = &c2
void operator=(const Complex &other)
    {
        cout<<"void operator=(const Complex &other)"<<endl;
        this->real = other.real;
        this->imag = other.imag;
    }
};
```

```
int main()
{

    Complex c1(10,20);
    Complex c2;
    c2 = c1;  // c2.operator=(c1)  it is assignment.
    c2.print_record( );
    return 0;
}
```

->in this program when you c2=c1 you will assign c1 value to c2
->but when we complie program error not get. And we not written any operator overloading declaration.
->so in this case complier give default assignment operator like copy constructor,destructor,constructor.
->aur hum chahe tho khud assignment operator declaration likh sakte hai.
->above program mai declaration diya hai observe the program carefully.
-> internally c2.operator=(c1) aise call ho raha hai.
-> declaration void operator=(const Complex &other) Aise dete hai.

Template Concept

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

void swap( int &x, int &y ){
    int temp = x;
    x = y;
    y = temp;
}
```

```
void swap( double &x, double &y ){
    double temp = x;
    x = y;
    y = temp;
}
void swap( string &x, string &y ){
    string temp = x;
    x = y;
    y = temp;
}
int main( void ){
    int a = 10;
    int b = 20;
    swap( a, b );
    cout<<"a    :   "<<a<<endl;
    cout<<"b    :   "<<b<<endl;

    double c = 10.5;
    double d = 20.5;
    swap( c, d );
    cout<<"c    :   "<<c<<endl;
    cout<<"d    :   "<<d<<endl;


    string s1 = "Karad";
    string s2 = "Pune";
    swap( s1, s2 );
    cout<<"s1   :   "<<s1<<endl;
    cout<<"s2   :   "<<s2<<endl;
    return 0;
}
```

->In above program I write three function for swap the data.
Function name same hai par type of argument different hai yaha pr function overload kiya hai.
->but agar muze aur kisi argument ko swap karna hai tho aur ek function likhana padena code size bhi badh gayi aur kam bhi difficult so isko avoid kaise kare.

->The solution in c++ is Template concept.

```
#include<iostream>
#include<string>
#include<stack>

using namespace std;
template<class T> //T : Type Parameter name   ye bhi syntax
ok
//template<typename T>   Ye bhi syntax ok
void swap_object( T  &x ,T &y)
{
    T temp = x;
    x = y;
    y = temp;
}

int main()
 {
    int a = 10;
    int b = 20;

    swap_object<int>(a,b); //int : type argument   Ye bhi
swap karenga
    // swap_object(a,b);
    // type inference (complier khud identify kr raha hai
type of argument)Ye bhi swap karenga
    cout<<"a    :   "<<a<<endl;
    cout<<"b    :   "<<b<<endl;


    double c = 10.5;
    double d = 20.5;
    swap_object<double>(c,d);
    cout<<"c    :   "<<c<<endl;
    cout<<"d    :   "<<d<<endl;
```

```
    string s1 = "Karad";
    string s2 = "Pune";
    swap_object<string>( s1, s2 );
    cout<<"s1   :   "<<s1<<endl;
    cout<<"s2   :   "<<s2<<endl;

}
```

->In above program,template use kiya to swap.
->template ko declare kro
template<class T> or template<typename T> yahi syntax hai dono hi okay hai
-> T : Type Parameter name
-> ab swap ka logic function likho aur T pass kro like this
void swap_object(T &x ,T &y)

->swap function ko call swap_object<int>(a,b); aise karte hai
Argument diya tho bhi thik hai agar nahi diya tho complier khud identify karta hai us case mai swap_object(a,b);
->ab ye koi bhi argument ke liye used kr sakte ho
->template ek generic code hai in c++

• If we want to write generic code in C++, then we should use template.
• Objective
1. Not to reduce code size
2. Not to reduce execution time
3. To reduce developers effort.
• It is possible by passing data type as a argument.
• Types of template: 1. Function template 2. Class Template
• An ability of compiler to detect and pass type of argument implicitly to the function is called type inference.
• Template is mainly designed for data structure and algorithm. • By passing data type as a argument, we can define generic code in C++. Hence parametrized type is called template

```
#include<iostream>
#include<stack>
#include<queue>
#include<list>
using namespace std;
int main()
{
    stack<int> stk;  //template in c++
    stk.push(10);
    stk.push(20);
    stk.push(30);
    // STL ==> standard template library
    return 0;
}
```

->This all we can used directly stack,linked list,queue.this is based used of template.

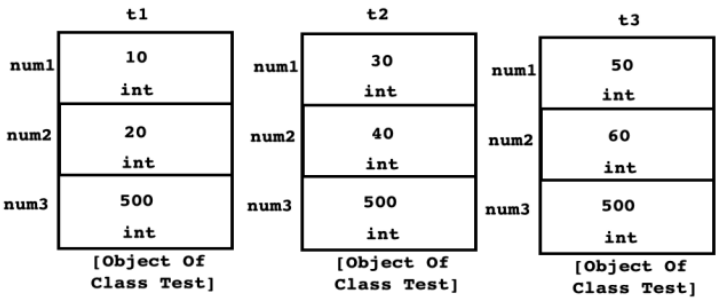
Static Concept

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

class Test
{
    private:
    int num1; // instance variable
    int num2; // instance variable
    int num3; // instance  variable

    public:
    Test(  ) //parameterless constructor
    {
        this->num1 = 10;
        this->num2 = 20;
        num3 = 500;
    }
```

```
    Test( int num1 , int num2 ) //parametrized constructor
    {
        this->num1 = num1;
        this->num2 = num2;
        num3 = 500;
    }
    void print_record( )
    {
        cout<<"Num1 :   "<<this->num1<<endl;
        cout<<"Num2 :   "<<this->num2<<endl;
        cout<<"Num3 :   "<<this->num3<<endl;

    }
};

int main()
{
    Test t1(10,20);
    t1.print_record( );

    Test t2(30,40);
    t2.print_record( );

    Test t3( 50,60);
    t3.print_record( );

    return 0;
}
```

->Now in this program in test t1 2 arguments pass ,then test t2 ,test t3 mai bhi 2 argument pass kiya.
->Test ke declaration mai muze num3=500 chahiye saabme tho har is mai likh liye.
->par sab maim era num3 common hai tho har bar used krne se meri memory waste ho rahi har bar 4 bytes waste ho rahi hai to avoid is problem the concept in c++ is called static.
->har bar num3 ko space mil rahi hai observe in diagram.

```
class Test{
private:
   int num1;
   int num2;
   int num3;
public:
   Test( void ){
      this->num1 = 0;
      this->num2 = 0;
      num3 = 500;
   }
   Test( int num1, int num2 ){
      this->num1 = num1;
      this->num2 = num2;
      num3 = 500;
   }
};
int main( void ){
   Test t1( 10, 20 );
   Test t2( 30, 40 );
   Test t3( 50, 60 );
    return 0;
```



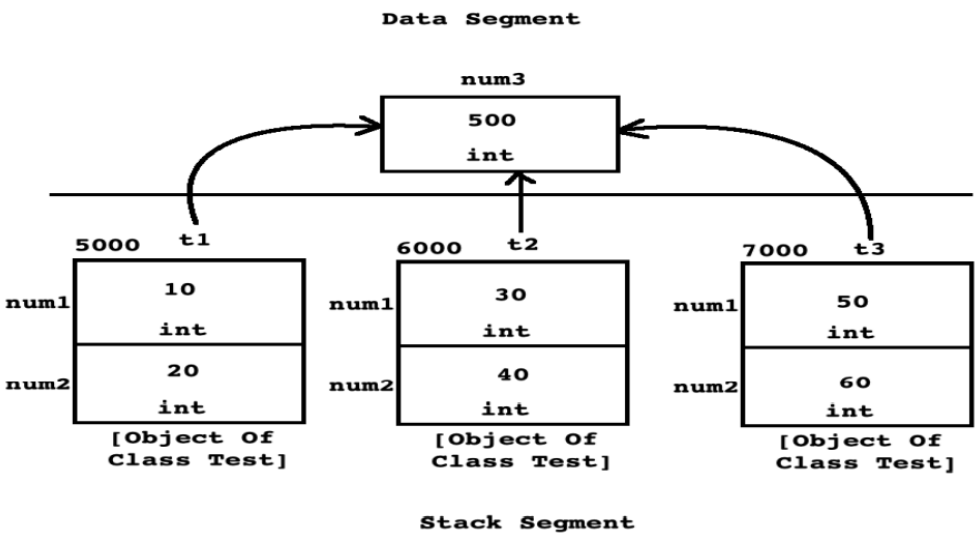
```
#include<iostream>
using namespace std;

class Test
{
    private:
    int num1; // instance variable
    int num2; // instance variable
    static int num3;// class level variable

    public:
    Test(  )
    {
        this->num1 = 10;
        this->num2 = 20;

    }
    Test( int num1 , int num2 )
    {
        this->num1 = num1;
        this->num2 = num2;

    }
```

```
    void print_record( )
    {
        cout<<"Num1 :   "<<this->num1<<endl;
        cout<<"Num2 :   "<<this->num2<<endl;
        cout<<"Num3 :   "<<Test::num3<<endl;
    //ab num3 part of object nhi hai isliye this pointer se
print nhi kr sakte isliye so used class name
    }
};
int Test :: num3 = 500; // Global defination


int main()
{
    Test t1(10,20);
    t1.print_record( );

    Test t2(30,40);
    t2.print_record( );

    Test t3( 50,60);
    t3.print_record( );

    return 0;
}
```

->Ab problem solve ho gayi num3 ko static karne se aur uska definition global dena padta hai.

->num3 ko ab space nhi milengi object pr usko data segment pe ek hi bar space mil jayengi

->//ab num3 part of object nhi hai isliye this pointer se print nhi kr sakte isliye so used class name.



Static Data Member:-

• Member function(static/non static) do not get space inside object. Hence size of object depends on size of all the data members declared inside class.
• Instance Variable : A data member of a class which get space inside object is called instance variable. In short, non static data member declared inside class is called instance variable. • Instance Method : A member function of a class, which is designed to call on object is called instance method. In short, non static member function defined inside class is called instance method. • Concrete Method : A member function of a class, which is having body is called concrete method.
• Concrete Class : A class from which we can create object is called concrete class. In other words, we can instantiate concrete class.
• Revision of some oops concepts:
1. Data member declared inside class get space once per object according to their declaration inside class.
2. If we call non static member function on object then non static member function get this pointer.
3. With the help of this pointer, all the objects of same class share single copy of member function.

- If we want to share value of any data member inside all the objects of same class then we should declare data member static. Static Data Member
- Static data member do not get space inside object. Rather all the objects of same class share single copy of it. Hence size of object is depends on size of all the non static data members declared inside class.
- A data member of a class which get space inside object is called instance variable.
- Instance variable is designed to access using object or pointer/reference of object.
- A data member of a class which do not get space inside object is called class level variable.
- Class level variable is designed to access using class name and scope resolution operator.
- If we declare data member static then we must provide global definition for it. Otherwise linker will generate error.
- Non static data member get space once per object. But static data member get space once per class.
- Only non static data member get space inside object. To initialize object we define constructor inside class. In other words, we should use constructor to initialize non static data member which get space inside object.
- Since static data member do not get space inside object, we should not initialize static data member inside constructor.


```
#include<iostream>
using namespace std;
class  Test
{
    private:
    int num1; // instance variable
    int num2; // instance variable
    static int num3; // class level variable
```

```
    public:
    Test ( void )
    {
        this->num1 = 0;
        this->num2 = 0;
    }

    void setNum1(int num1)
    {
        this->num1 = num1;
    }
    void setNum2( int num2 )
    {
        this->num2 = num2;
    }
    static void setNum3( int num3 )
    {
           Test::num3 = num3;
    }

    void printRecord( void)
    {
           cout<<"Num1   :   "<<this->num1<<endl;
           cout<<"Num2   :   "<<this->num2<<endl;
    }
};

int Test::num3 = 0; // Global def
int main()
{
        Test t1;
        t1.setNum1(10);//t1.setNum1(&t1,10);
        t1.setNum2(20);////t1.setNum2(&t1,20);
        Test::setNum3(30);
        t1.printRecord( );

        return 0;
}
```

• In C++, we can declare global function as well as member function static.
• If we want to access non static members of the class then member function should be non static and if we want to access static members of the class then member function should be static.
• Non static member function get this pointer. Hence inside non static member function, we can access static as well as non static members.
• Why static member function do not get this pointer?
1.If we call non static member function on object then non static member function get this pointer. In other words, non static member functions are designed to call on object.
2.Static member function is designed to call on class name.
3.Since static member function is not designed to call on object, it doesn't get this pointer.
• Since static member function do not get this pointer, we can not access non static members inside static member function. In other words, static member function, can access only static members of the class directly.
• Using object, we can use non static members inside static member function.
• Inside member function, if we are going to use this pointer then member function should be non static otherwise it should be static.
• We can not declare static member function:
  1. constant 2. volatile 3. Virtual

Advantages of OOPS

1. To achieve simplicity
2. To achieve data hiding and data security.
3. To minimize the module dependency so that failure in single part should not stop complete system.
4. To achieve reusability so that we can reduce development time/cost/efforts.
5. To reduce maintenance of the system.

6. To fully utilize hardware resources.
7. To maintain state of object on secondary storage so that failure in system should not impact on data.

Major and Minor pillars of oops:-

- **4 Major pillars**

1. Abstraction 2. Encapsulation 3. Modularity 4. Hierarchy

- **3 Minor Pillars**

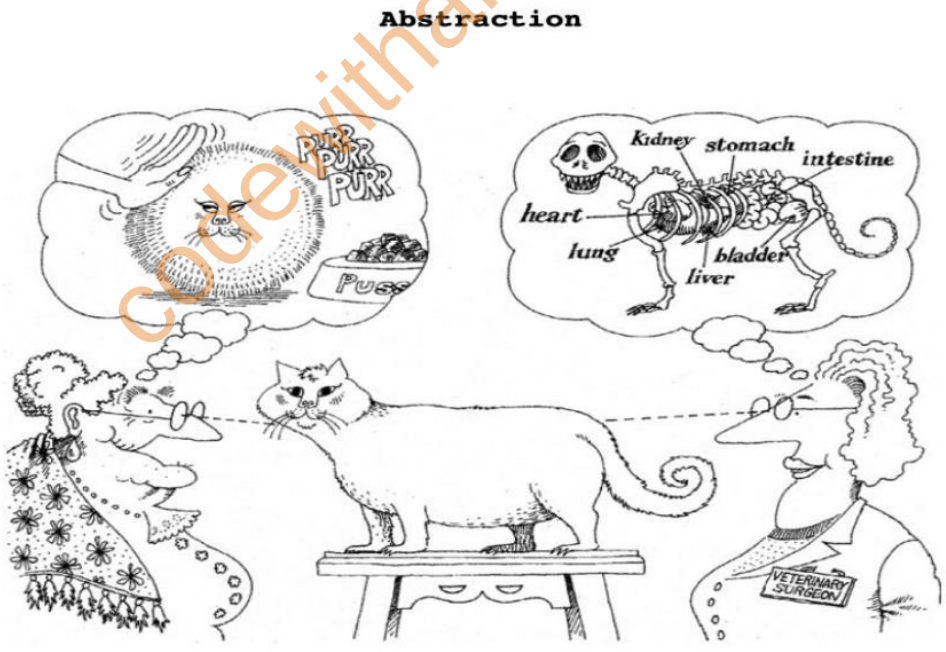
1. Typing 2. Concurrency 3. Persistence

1.Abstraction:-

- It is a major pillar of oops.
- It is a process of getting essential things from object.
- It describes outer behaviour of the object.
- Abstraction focuses on some essential characteristics of object relative to the perspective of viewer.

In other words, abstraction changes from user to user.

- Using abstraction, we can achieve simplicity.
- Abstraction in C++ Complex c1; c1.acceptRecord( ); c1.printRecord( );



Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

2.Encapsulation

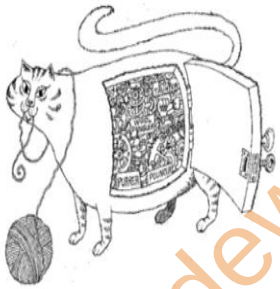
- It is a major pillar of oops.
- Definition:

1. Binding of data and code together is called encapsulation.

2. To achieve abstraction, we should provide some implementation. It is called encapsulation.

- Encapsulation represents, internal behaviour of the object.
- Using encapsulation we can achieve data hiding.
- Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behaviour of an object, whereas encapsulation focuses on the implementation that gives rise to this behaviour.

Encapsulation



Encapsulation hides the details of the implementation of an object.

Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.

Encapsulation using C++.

```
class Complex{
private:
    int real;
    int imag;
public:
    Complex( void ){
    }
    void acceptRecord( void ){
    }
    void printRecord( void ){
    }
};
```

3.Modularity

• It is a major pillar of oops.
• It is the process of developing complex system using small parts.
• Using modularity, we can reduce module dependency.
• We can implement modularity by creating library files. o .lib/.a, .dll / .so files o .jar/.war/.ear in java

---

4.Hierarchy

• It is a major pillar of oops.
• Level / order / ranking of abstraction is called hierarchy.
• Main purpose of hierarchy is to achieve reusability.
• Advantages of code reusability
1. We can reduce development time.
2. We can reduce development cost.
3. We can reduce developers effort.

• Types of hierarchy:
1. Has-a / Part-of => Association
2. Is-a / Kind-of => Inheritance / Generalization
3. Use-a => Dependency
4. Creates-a => Instantiation

1.Typing

• It is a minor pillar of oops.
• Typing is also called as polymorphism.
• Polymorphism is a Greek word. Polymorphism = Poly(many) + morphism(forms).
• An ability of object to take multiple forms is called polymorphism.
• Using polymorphism, we can reduce maintenance of the system. • Types of polymorphism: o Compile time polymorphism
• It is also calling static polymorphism / Early binding / Weak Typing / False polymorphism.

• We can achieve it using:
    1.Function Overloading
    2. Operator Overloading
    3. Template o Run time polymorphism
• It is also calling dynamic polymorphism / Late binding / Strong Typing / True polymorphism.
• We can achieve it using:
  1. Function Overriding.

2.Concurrency

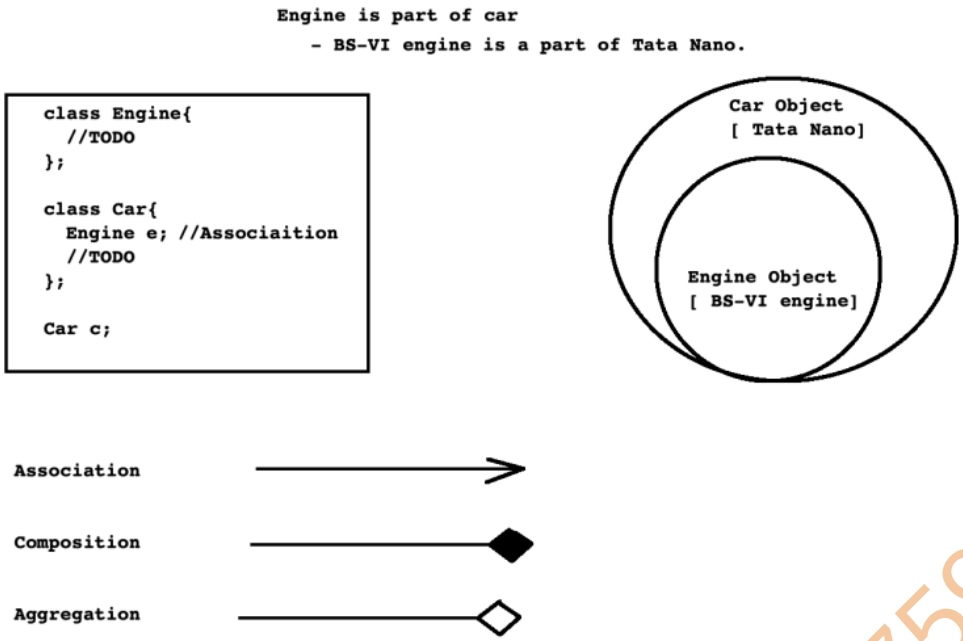
• It is a minor pillar of oops.
• In context of operating system, it is called as multitasking. • It is the process of executing multiple task simultaneously. • Main purpose of concurrency is to utilise CPU efficiently.
• In C++, we can achieve concurrency using thread.

3.Persistence

• It is a minor pillar of oops.
• It is process of maintaining state of object on secondary storage.
• In C++, we can achieve Persistence using file and database

Association:-

• Consider following examples:
1. Car has-a engine 2. Room has-a wall 3. Room has-a chair 4. Employee has-a join date.
• If "has-a" relationship is exist between the types then we should use association.
• Relation between car and engine can be considered as follows: 1. Car has-a engine 2. Engine is part of car




```
#include<iostream>

using namespace std;

class Date
{
        int day;
        int month;
        int year;
        public:
        Date(void):day(0),month(0),year(0) //constr member inti..
        { }
        Date(int day,int month,int
year):day(day),month(month),year(year){

        }
        void accept_record()
        {
             cout<<"Day :    ";
             cin>>this->day;

             cout<<"Month :    ";
             cin>>this->month;

             cout<<"Year :    ";
```

```
            cin>>this->year;
        }
        void print_record( void  )
        {
            cout<<this->day<<" / "<<this->month<<" / "<<this-
>year<<endl;
        }
};

class Employee
{
        private:
        string name; //24
        int empid; // 4
        float salary; // 4
        Date joinDate; //12  association

        public:
        Employee(void):name(""),empid(0),salary(0){

        }
        Employee(string name, int empid, float salary,Date
joinDate):
        name(name),empid(empid),salary(salary),joinDate(joinDate){
        }
         void accept_record()
         {
             cout<<"Name :   ";
             cin>>this->name;
             cout<<"Empid    :";
             cin>>this->empid;
             cout<<"Salary   :";
             cin>>this->salary;
          }
        void print_record( )
        {
            cout<<"Name :   "<<this->name<<endl;
            cout<<"Empid :   "<<this->empid<<endl;
            cout<<"Salary :   "<<this->salary<<endl;
            cout<<"join date    :   ";
            this->joinDate.print_record( );
        }
};
```

```
int main()
{
    Date joinDate(1,1,2007 );
    Employee emp("Ketan",1,10000,joinDate);
    emp.print_record( );
    return 0;
}
```

Association

- There are two special forms of Association:

1. Composition 2. Aggregation

- **Aggregation**
  - Let us consider example of department and faculty

```
class Faculty{
};
class Department{
    Faculty faculty;    //Associaition  :  Aggregation
};
//Dependant Object   :   Department object
//Dependancy Object :   Faculty Object
```

  - In case of association, if dependency object exist without dependant object then it is called aggregation.
  - In other words, aggregation represents loose coupling.

2. Aggregation

- **Composition**

```
class Heart{
    //TODO
};
class Human{
    Heart heart;    //Association   =>  Composition
};
//Dependant Object   :   Human object
//Dependancy Object :   Heart Object
```

  - In case of association, if dependency object do not exist without Dependant object then it is called composition.
  - In other words, composition represents tight coupling.

Inheritance

- If "is-a" relationship is exist between the types then we should use inheritance.
- Inheritance is also called as generalization.
- Example:

1. Employee is a Person
2. Manager is a Employee
3. Book is a product
4. Circle is a Shape
5. SavingAccount is a account.
6. Car is a vehicle

- Let us consider example of employee and person:

➢ Employee is a Person

```
class Person{    //Parent class
    //TODO
};
class Employee : public Person{ //Child class
    //TODO
};
```

- Employee is a Person
- In C++, parent class is called base class and child class is called derived class.
- In above statement, "public" is called as mode of inheritance. It can be private/protected/public.
- In C++, default mode of inheritance is private

```
#include<iostream>
using namespace std;
class Person
{
    private:
    string name; //4
    int age; // 4
```

```
    public:
    Person( void ):name(""),age(0){
    }
    Person(string name , int age):name(name),age(age){
    }
    void showRecord( )
    {
        cout<<"Name :    "<<this->name<<endl;
        cout<<"Age :    "<<this->age<<endl;
    }
};

class Employee : public Person  //inheritance
{
    //  string name; // 24
    //  int age;  // 4
     int empid; // 4
     float salary; // 4
     public:
     Employee( void ): empid( 0 ),salary(0){
     }
     Employee( int empid , float
salary):empid(empid),salary(salary){
     }
     void displayRecord( )
     {
         cout<<"Empid    :    "<<this->empid<<endl;
         cout<<"Salary    :    "<<this->salary <<endl;
     }
};

int main()
{
     Employee e;
     cout<<sizeof(e); // 36
}
```

• During inheritance, members(data member / member function / nested type) of derived class, do not inherit into base class. Rather members of base class, inherit into derived class.

- All the non static data members of base class get space inside object of derived class. In other words, non static data members of base class, inherit into derived class.
- Using derived class, we can access static members of base class. In other words, static data members of base class, inherit into derived class.
- All the data members(static/non static) of base class of any access specifier(private/protected/public), inherit into derived class but only non static data member get space inside object.
- Size of object of Base class = size of of all the non static data members declared in base class.
- Size of object of Derived class = size of of all the non static data members declared in base class + size of of all the non static data members declared in derived class.

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

class Person
{
    private:
    string name; // 24
    int age; // 4
    public:

    Person( void ):name(""),age(0){
            cout<<"Person( void )"<<endl;
    }
    Person( string name, int age  ):name(name),age(age){
            cout<<"Person( string name, int age  )"<<endl;
    }
```

```
    void showRecord(  void )
    {
            cout<<"Name    :   "<<this->name<<endl;
            cout<<"Age     :   "<<this->age<<endl;
    }

    ~Person( void )
    {
        cout<<"~Person( void )"<<endl;
    }
};


class Employee : public Person
{
    //string name; // 24
    //int age; // 4
    int empid; // 4
    float salary; // 4

    public:

    Employee( void ): empid(0),salary(0){
             cout<<"Employee( void )"<<endl;
    }
                        //constr base initilization list.
     Employee(string name,int age,int empid,float salary
):Person(name,age),empid(empid),salary(salary)
     {
         cout<<"Employee(  int empid,float salary )"<<endl;
    }
    // this = &e
    void displayRecord( void )
    {
            //this->showRecord( );
            Person::showRecord( );
            cout<<"Empid     :   "<<this->empid<<endl;
            cout<<"Salary     :   "<<this->salary<<endl;
    }
```

```
    ~Employee( void )
    {
        cout<<"~Employee( void )"<<endl;
    }
};
int main()
{
    Employee emp("Ketan",30,1,1000);
    emp.displayRecord ( );
    return 0;
}
```

Sample Question:-

1.Which of the following feature is used in function overloading and function with default argument?

a) Encapsulation
b) Polymorphism
c) Abstraction
d) Modularity

2.Which of the following gets called when an object is being created?

A. Constuctor
B. Virtual Function
C. Destructor
D. Main

3.Like constructors, can there be more than one destructors in a class?

A. Yes
B. No
C. May Be
D. Can't Say

4.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

5.Where can the default parameter be placed by the user?
a) leftmost
b) rightmost
c) both leftmost & rightmost
d) topmost

6.How many types of constructors are there in C++?
a) 1
b) 2
c) 3
d) 4

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

8.What are the constant member functions?
a) Functions which doesn’t change value of calling object
b) Functions which doesn’t change value of any object inside definition
c) Functions which doesn’t allow modification of any object of class
d) Functions which doesn’t allow modification of argument objects

9.Can a constructor function be constant?
a) Yes, always
b) Yes, only if permissions are given

c) No, because objects are not involved
d) No, never

10.Which one of the following is correct syntax of copy constructor?
A. const ClassName(ClassName &other)
B. ClassName const (ClassName &other)
C. ClassName (const ClassName &other)
D. ClassName (ClassName & const other)

11. Which one of the following operator we can not overload.
A. =
B. []
C. ()
D. ? :

12. Which one of the following is not a conversion function?
A. Single parameter constructor.
B. Copy constructor.
C. Assignment operator function.
D. Type converion operator function.

13. Which one of the following do not represent compile time polymorphism?
A. Function Overloading
B. Operator Overloading
C. Function Overriding
D. Template

14.If new operator fail to allocate memory then_____.
A. It return NULL.
B. It throws bad_cast exception.
C. It throws bad_typeid exception.
D. It throws bad_alloc exception.

15. Select correct syntax of template.
A. template class<T>
B. <template class T>
C. <template> class T
D. template< class T>

16. Which one of the following function can be static.
A. constant member function.
B. virtual member function.
C. volatile member function.
D. global function.

17. Select the incorrect statement about static.
A. Size of object do not depend on size of static data member.
B. We can call static member function on object.
C. Static member function get this pointer.
D. Using instance, we can access non static members inside static member function.

18. If we want to access private members in different scope then we should use___
A. scope resolution operator.
B. friend function / class
C. extern keyword
D. None of the above.

19. Which one of the following keyword is allowed to use with destructor
A. static
B. const
C. volatile
D. virtual.

20. What will be the output of the following program?

```
#include <iostream>
using namespace std;
class Program{
    int id;
    static int count;
public:
    Program() {
        count++;
        id = count;
        cout << "constructor for id " << id << endl;
    }
    ~Program() {
        cout << "destructor for id " << id << endl;
    }
};
int Program::count = 0; //Global Definition
int main() {
    Program a[3];
    return 0;
}
```

A. constructor for id 1 constructor for id 2 constructor for id 3 destructor for id 3 destructor for id 2 destructor for id 1

B. constructor for id 1 constructor for id 2 constructor for id 3 destructor for id 1 destructor for id 2 destructor for id 3

C. Compiler Dependent

D. constructor for id 1

Q.21 What will be the output of the following program?

```
#include <iostream>
using namespace std;
class Program{
    static int x;
public:
```

```
    static void Set(int xx){
        x = xx;
    }
    void Display() {
        cout<< x ;
    }
};
int Program::x = 0;
int main()
{
    Program::Set(33);
    Program::Display();
    return 0;
}
```

A. The program will print the output 0.
B. The program will print the output 33.
C. The program will print the output Garbage.
D. The program will report compile time error.

22.Which of the following statements are true about Catch handler?
i) It must be placed immediately after try block T.
ii) It can have multiple parameters.
iii) There must be only one catch handler for every try block.
iv) There can be multiple catch handler for a try block T.
v) Generic catch handler can be placed anywhere after try block.

A. Only i, iv, v
B. Only i, ii, iii
C. Only i, iv

Inheritance

• We can call, non static member function of base class on object of derived class. In other words, non static member function inherit into derived class.
• We can call static member function of base class on derived class. In other words, static member function inherit into derived class.
• Following function do not inherit into derived class:
1. Constructor
2. Destructor
3. Copy constructor
4. Assignment operator function
5. Friend function
• Except above 5 functions, all the member functions(static/non static) of base class inherit into derived class.

• If we create object of base class then only base class constructor and destructor gets called.
• If we create object of derived class then first base class constructor gets called and then derived class constructor gets called. Destructor calling sequence is exactly opposite.
• From any constructor of derived class, by default, base class's parameterless constructor gets called.
• If we want to call, any constructor of base class from constructor of derived class then we should use constructor's base initializer list.

Inheritance

• If we use private/protected/public keyword to control visibility of members of class then it is called access specifier. • If we use private/protected/public keyword to extend the class / to create derived class then it is called mode of inheritance.
1. private mode(default mode)
→ If has-a relationship is exist between the types then we should use either association or private mode of inheritance.

2. public mode
→If is a relationship is exist between the types then we should use public mode of inheritance.
3. protected mode.
→ Firmly not say in which scenario to used.

public mode of inheritance					
Access Specifier	Same Class	Derived Class	Indirect Derived Class	Friend Function	Non Member Function
private	A	NA	NA	A	NA
protected	A	A	A	A	NA
public	A	A	A	A	A

private mode of inheritance					
Access Specifier	Same Class	Derived Class	Indirect Derived Class	Friend Function	Non Member Function
private	A	NA	NA	A	NA
protected	A	A	NA	A	NA
public	A	A	NA	A	A Using Base class Object NA Using Derived class Object

protected mode of inheritance					
Access Specifier	Same Class	Derived Class	Indirect Derived Class	Friend Function	Non Member Function
private	A	NA	NA	A	NA
protected	A	A	A	A	NA
public	A	A	A	A	A Using Base class Object NA Using Derived class Object

---

Inheritance
• During inheritance, if base type and derived type is interface then it is called interface inheritance.
• During inheritance, if base type and derived type is class then it is called implementation inheritance.
• Types of inheritance
o Interface Inheritance
1.Single Inheritance
2.Multiple Inheritance
3.Hierarchical Inheritance
4. Multilevel Inheritance

o Implementation Inheritance

1. Single Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multilevel Inheritance

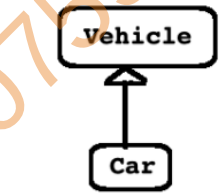
->In our case we learn Implementation Inheritance

---

1. Implementation Single Inheritance:-

- Consider example:
  - Car is a Vehicle.

```
class Vehicle{  };
class Car : public Vehicle{ };
```



Single Inheritance

- If single base class is having single derived class then it is called single inheritance.

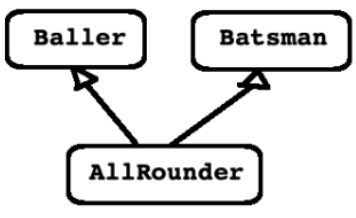
---

2. Implementation Multiple Inheritance:-

- Consider example:
  - Allrounder is Baller and Batsman.

```
class Baller{   };
class Batsman{  };
class AllRounder : public Baller, public Batsman{   };
```

- If multiple base classes are having single derived class then it is called multiple inheritance.



Multiple Inheritance

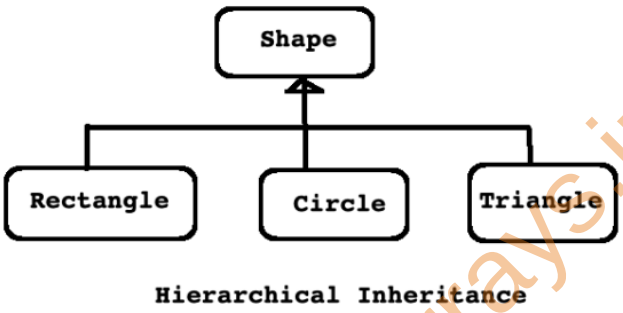
---

3. Implementation Hierarchical Inheritance:-

- Consider example:
  - Rectangle is a Shape
  - Circle is a Shape
  - Triangle is a Shape

```
class Shape{    };
class Rectangle : public Shape{ };
class Circle : public Shape{ };
class Triangle : public Shape{ };
```

- If single base class is having multiple derived classes then it is called hierarchical inheritance.



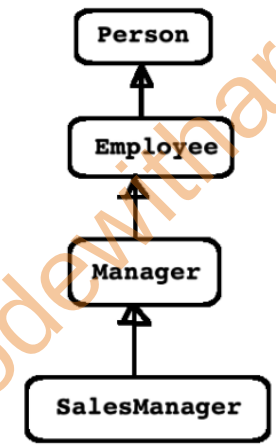
Hierarchical Inheritance

4.Implementation Multilevel Inheritance

- Consider example:
  - Employee is a Person
  - Manager is a Employee
  - SalesManager is a Manager

```
class Person{    };
class Employee : public Person{ };
class Manager : public Employee{ };
class SalesManager : public Manager{ };
```

- If single inheritance is having multiple levels then it is called multilevel inheritance.

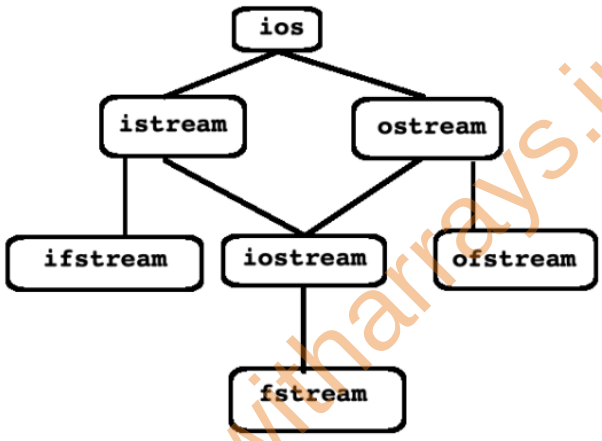


Multilevel Inheritance

5.Implementation Hybrid Inheritance

- Combination of any two or more than two types of inheritance is called hybrid inheritance.

```
class ios{  };
class istream : public ios{ };
class ifstream : public istream{    };
class ostream : public ios{ };
class ofstream : public ostream{    };
class iostream : public istream, public ostream{    };
class fstream : public iostream{    };
```



Hybrid Inheritance

• If implementation of base class member function is logically incomplete then we should redefine member function inside derived class. In other words, we should give same name to the member function in derived class.

• If name of base class and derived class member function is same and if we try to call such member function on object of derived class then preference is given to the derived class member function. Here derived class member

function hides implementation of inherited function. This process is called shadowing.

- Without changing implementation of existing class, if we want to extend meaning of that class then we should use inheritance.
