

1. What is object slicing?

Ans: "Slicing" is where you assign an object of a derived class to an instance of a base class, thereby losing part of the information of derived class- some of it is "sliced" away.

```
class A{
    int a,b;
    public:
        A(){
            a=0;b=0;
        }
        A(int x, int y){
            a=x;b=y;
        }
        virtual void show(){ //virtual is not works here
            cout<<a<<b;
        }
};

class B : public A{
    int c;
    public:
        B(int x, int y,int z) : A(x , y){
            c =z;
        }
        void show(){
            cout<<c;
        }
};

int main(){
    B b(10,20,30);
    func(b);
}

void func(A a){
    a.show(); // c will sliced
//10 20
void func(A &a){ //or A *a
    a.show();
// 30
```

Q2). What is deep copy and shallow copy ?

Ans:-

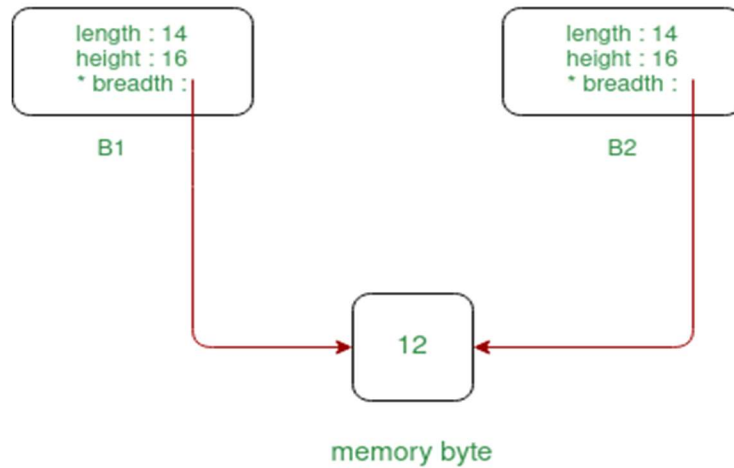
How we can create the copy of object?

- Copy Constructor
- Implicit Default assignment operator

Shallow copy:- Creating a copy of object by copying data of all member variables as it is. Is called shallow copy. if some variables are defined in heap memory, then:

If some variables are dynamically allocated memory from heap section, then the copied object variable will also **reference the same memory location**.

Shallow Copy



```
class box {
private:
    int length, breadth , height;
public:
    void set_dimensions(int length1, int breadth1, int height1) {
        length = length1;
        breadth = breadth1;
        height = height1;
    }
    void show_data(){
        cout << " Length = " << length<< "\n Breadth = " << breadth << "\n Height = " << height;
    }
};

int main(){
    box B1, B3;
    B1.set_dimensions(14, 12, 16);
    B1.show_data();
    // When copying the data of object at the time of initialization then copy is made through
    // COPY CONSTRUCTOR
```

box B2 = B1; // at time of initialization so copy constructor invoked

B2.show_data();

// When copying the data of object after initialization then the copy is done through DEFAULT

// **ASSIGNMENT OPERATOR**

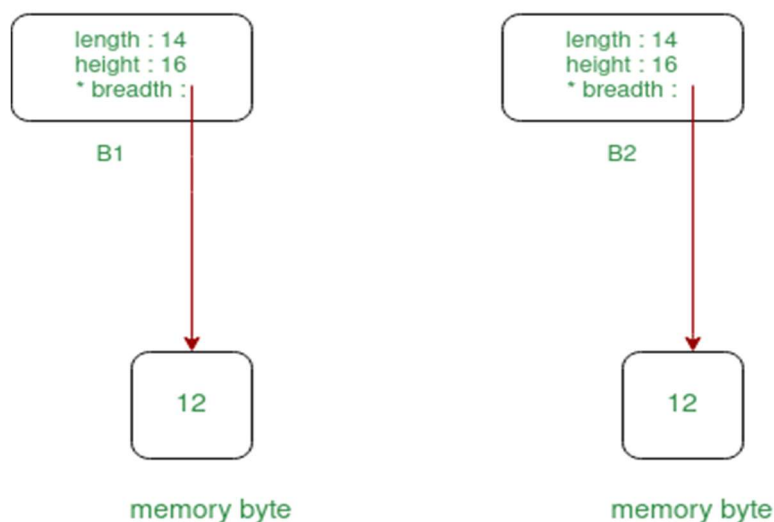
B3 = B1;

B3.show_data(); }

Deep Copy:- means creating an object by copying data of another object along with the values of memory resources **resides outside the object but handled by that object.**

In Deep copy, an object is created by copying data of all variables, and it also allocates similar memory resources with the same value to the object. In order to perform Deep copy, we need to **explicitly define the copy constructor** and assign dynamic memory as well, if required. Also, it is required to dynamically allocate memory to the variables in the other constructors, as well.

Deep Copy



```
class box {  
private:  
    int length;  
    int* breadth;  
    int height;  
public:  
    box()  
    {  
        breadth = new int;  
    }  
    void set_dimension(int len, int brea, int heig)
```

```

{
    length = len;
    *breadth = brea;
    height = heig;
}

void show_data()
{
    cout << " Length = " << length<< "\n Breadth = " << *breadth<< "\n Height = " << height;
}

```

// Parameterized Constructors for for implementing deep copy

```

box(box& sample)
{
    length = sample.length;
    breadth = new int;
    *breadth = *(sample.breadth);
    height = sample.height;
}

~box()
{
    delete breadth;
}

```

```

};

```

```

int main()

```

```

{

```

```

    box first;
    first.set_dimension(12, 14, 16);
    first.show_data();

```

// When the data will be copied then all the resources will also get allocated to the new object

```

    box second = first;
    second.show_data();}

```

Q 3). What is a Header File?

Ans:- which contain only **declarations** of functions , classes etc. ie only prototype not definations.

A header file contains:

- Function definitions
- Data type definitions
- Macros

Example:- **stdio.h** printf(), scanf() etc.

Math.h sqrt(),pow() etc.

The actual definition of these files resides in **Library Files**.

Create your own Header File:

Sum.h

```
int sumOfTwoNumbers(int a, int b)
```

```
{  
    return (a + b); //providing declaration is not a good practice  
}
```

Main.cpp

```
#include "iostream"  
  
#include "sum.h"  
  
using namespace std;  
  
int main()  
{  
    int a = 13, b = 22;  
  
    // Function declared in header file to find the sum  
  
    cout << "Sum is: "<< sumOfTwoNumbers(a, b) << endl;  
}
```

Q4. Difference between header file and library file ?

Header Files

Library Files

They have the extension .h

They have the extension .lib

They contain function declaration and even macros.

They contain function definitions

They are available inside "include sub directory" which itself is in Turbo compiler.

They are available inside "lib sub directory" which itself is in Turbo compiler.

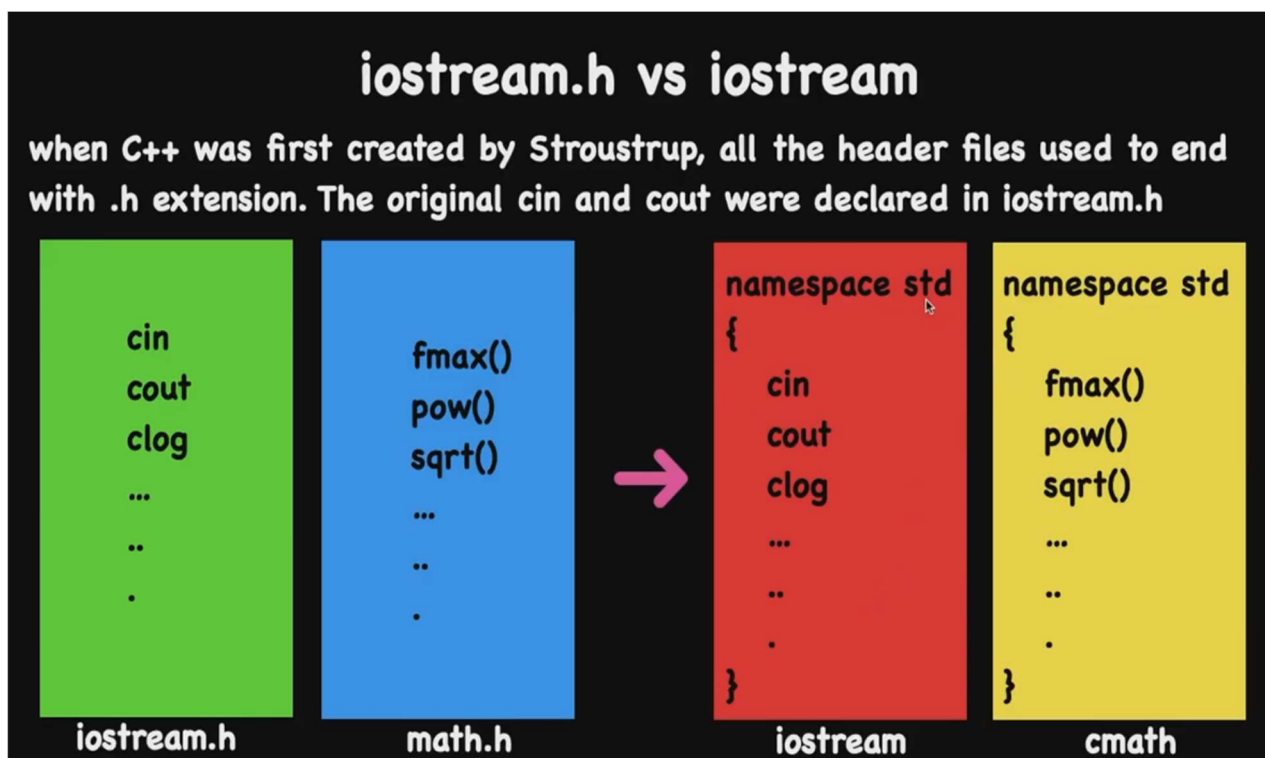
Header files are human-readable. Since they are in the form of source code.

Library files are non-human-readable. Since they are in the form of machine code.

Header files in our program are included by using a command `#include` which is internally handle by pre-processor.

Library files in our program are included in last stage by special software called as linker.

Q5. `iostream.h` vs `iostream`



Q6. Namespace:-

Problems Before namespace:-

- **Ambiguity** for compiler Example, you might be writing some code that has a **function called `xyz()`** and there is **another library** available which is also having **same function `xyz()`**. Now the

compiler has no way of knowing which **version of xyz()** function you are referring to within your code.

Namespace:-

Namespace is the container for identifiers.

Before namespace:

```
#include<math.h>
#include<iostream.h>
double pow(double, double);
double pow(double x, double y)
{
    return x;
}
int main()
{
    double a =2, b = 3, c;
    c = pow(a,b);
    cout<< c ;
}
```

You can't access pow()
of math.h

either change the
name of the function
or argument type
or number of arguments

But with the help of namespace
You can access pow()
of cmath

fmax()
pow()
sqrt()
...
..
.

math.h

2

Solution

```
1 #include<cmath>
2 #include<iostream>
3 namespace ok
4 {
5     double pow(double, double);
6 }
7 double ok::pow (double x, double y)
8 {
9     return x;
10 }
11 int main()
12 {
13     double a, b;
14     a = 2;
15     b = 3;
16     double c = ok::pow(a,b);
17     std::cout<<c;
18 }
```

Or use **std :: pow(a,b)** give 8 output

How to create namespace ?

- **namespace** MySpace{
 //Declaration
}
- Namespace definition doesn't terminate with a semicolon like in class definition.
- The namespace definition **must be done at global scope**, or nested inside another namespace.
- You can use **alias** name for your namespace name, for ease of use.
 namespace ms = Myspace;
- Namespace is not a class, you cannot create instance of namespace.
- **Unnamed namespaces**
 Namespace {};
- **Accessing members of namespace** any name (identifier) declare in a namespace can be explicitly specified using namespaces name and the scope resolution :: operator with the identifier.

Example:

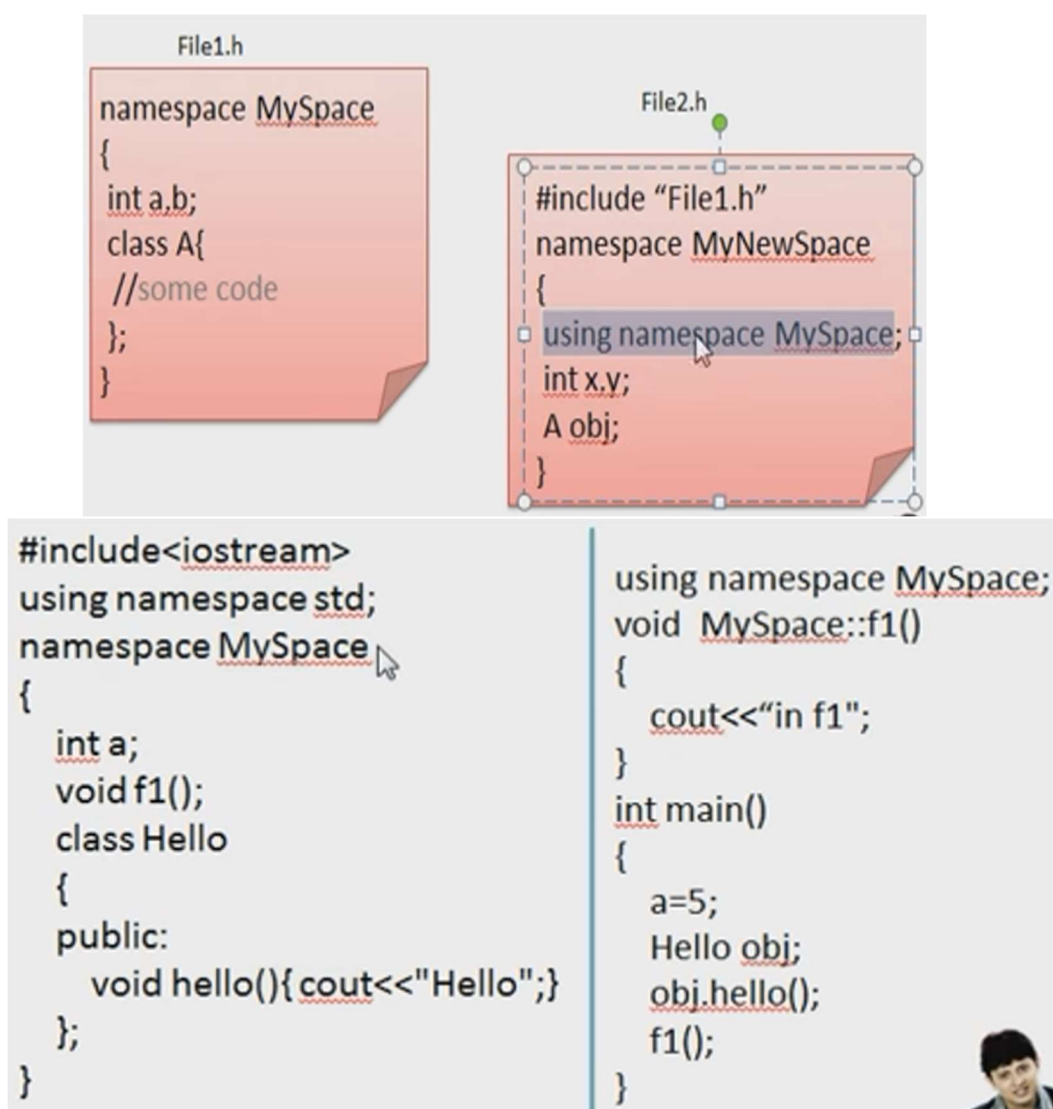
```
#include<iostream>
using namespace std;
namespace MySpace
{
    int a;
    void f1();
    class Hello
    {
    public:
        void hello(){ cout<<"Hello";}
    };
}

void MySpace::f1()
{
    cout<<"In f1";
}

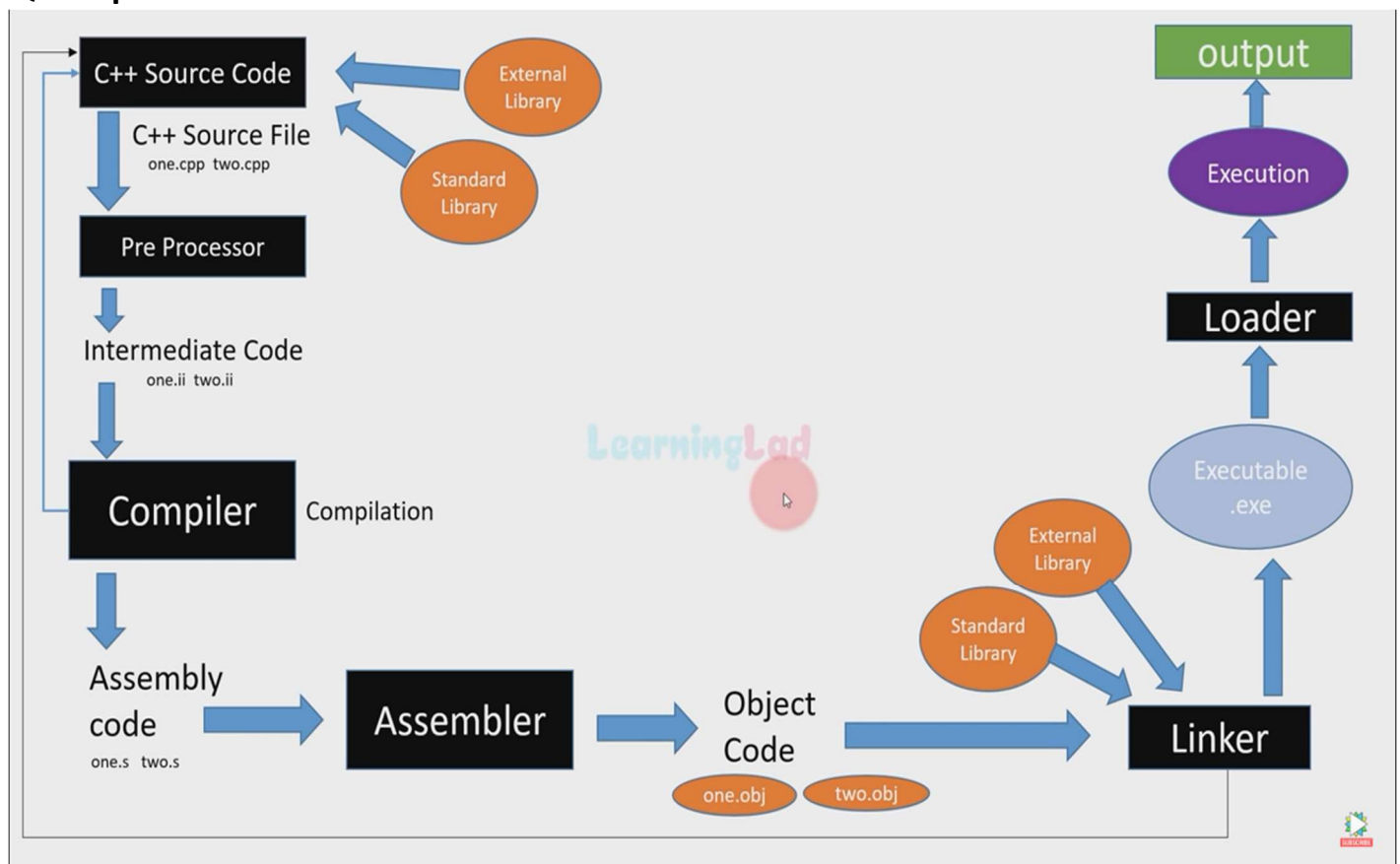
int main()
{
    MySpace::a=5;
    MySpace::Hello obj;
    obj.hello();
    MySpace::f1();
}
```

The **using** directive

- Using keyword allows you to import an entire namespace into your program with a global scope.
- It can be used to import a namespace into another namespace or any program.



Q6.Steps from source file to executable file:-



Source code: The code written in any programming language is called as the source code and the code written using the C++ programming language is called as the C++ source code. C++ source code is saved with .cpp Extension and .hpp extension.

you can write the c++ program by using the cpp standard library and external library(static library or dynamic library)

we directly can't execute the code that we have written in the C++ programming language because as a programmer we can look at the code and understand what exactly is going on but the computers can **only understand the code written in the machine language ie 0 or 1.**

the process of creating an executable file from the source code is **called as the build process.**

so the first step in the build process in order to create an executable file is preprocessing. we use a program called **as pre processor** and this preprocessor is a macro processing program.

Next we need to translate the **code to machine code** and for that purpose we use a program called as the **compiler**. example **GNU – GCC Compiler Collection.**

compiler is a computer program that transforms the code written in the **higher level** programming language **to lower level** programming language and this process is called as the compilation process.

The assembler will take the **Assembly code** and it will generate the **object code** or the **machine code.**

The linker will take the **object files** generated, it will **link** them together and it will generate the **executable file.**

The loader will load the executable file into the main memory which is the RAM so that the CPU can execute that program.

Q7. What is encapsulation ? explain with example.

Ans. It is one the important feature of oops. That used to wrapping the **related** data and functions into single unit. The data of class is **not accessible** to outside the class , only function **access** data which are wrapped in the class.

Encapsulation = Data hiding + abstraction.

Example : - class is an example of encapsulation, which talk about binding the data and function together that manipulate the data.

```
class Encapsulation
{
    private:
        // data hidden from outside world
        int x;
    public:
        // function to set value of variable x
        void set(int a)
        {
            x =a;
```

```

    }

    // function to return value of variable x
    int get()
    {
        return x;
    }
};

int main()
{
    Encapsulation obj;
    obj.set(5);
    cout<<obj.get();
    return 0;
}

```

Q8. What is abstraction ? explain with example?

Ans:- is one of the important feature of oops which is showing the essential information to the outside world and hiding the internal details.

Exapmle:- Class, Headerfile(sqrt()) don't show details, ATM GUI screen, Abstract datatype.

```

Class A{
    Private:
        Int a=10;
    Public
        Void show(){ cout<<a;}
};

Main(){ A obj; obj.show();} // a is not accessible from outside the fn

```

Qus 9. What is reference variable in c++?

- A variable which are provide alternate name for the previously defined or existing variable are called reference variable.
- Reference variable is an **internal pointer**.
- **It** must be initialized during declaration.
- **It** can only be initialized with already declared variables only.
- **Reference variable cannot be updated.**

Example:-

```

int a =10;           12345[ ] <-----a
Int &b =a;           |-----b and b both point to same memory location

b++;

cout<<a<<b; // 11 11

int &b=a;

int c=6;

&b=c; error

```

Q10. Difference between c and c++?

C by Dennis Ritchie	C++ Bjarne Stroustrup
1.C supports function programming.	Supports both procedural and object oriented style of programming.
2. C is a middle level language.	C++ is the high level language.
C is subset of c++	It is superset
Modifiers can be used for class members to make it inaccessible for outside users.	Data is less secured.
Exception handling can only be performed using other functions.	Provides exception handling using Try and Catch block.
Does not support the feature of namespace. 32 keyword	Supports the feature of namespace. 63 keyword

Q. class vs structure:

BASIS FOR COMPARISON	STRUCTURE	CLASS
Basic	If access specifier is not declared, by default all member are 'public'.	If access specifier is not declared, by default all members are 'private'.
Declaration	<pre>struct structure_name{ type struct_element 1; type struct_element 2; type struct_element 3; . };</pre>	<pre>class class_name{ data member; member function; };</pre>
Instance	Instance of 'structure' is called 'structure variable'.	Instance of a 'class' is called 'object'.

BASIS FOR COMPARISON	STRUCTURE	CLASS
Polymorphism and inheritance	Not supported	Supports polymorphism and a class can also be inherited.
Nature	Members of a structure are public by default.	Members of a class are private by default.
Memory is allocated on	Stack	Heap
Null values	Not possible	Can have null values
Requires constructor and destructor	No	Yes

Q12. What is template?

Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type. **In simple terms, you can create a single function or a class to work with different data types using templates.** Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs.

The concept of templates can be used in two different ways:

- 2. Function Templates
- 3. Class Templates
- **Function Template:-**

Function Overloading vs Function Template

- **Function overloading –**

```
int add(int x, int y){}  
float add(float x, float y){}  
double add(double x, double y){}
```

```
int main ()  
{  
    add(5,4);  
    add(2.3f, 4.2f);  
    add(5.3232, 42324.453);  
}
```

- **Function Template –**

```
template <typename T>  
T add(T x, T y)  
{}
```

```
int main()  
{  
    add<int>(3, 7);  
    add<float>(3.3, 7.5);  
    add<double>(3.55, 7.66);  
}
```

Template <typename T,typename U>

U add(T x , U y){}

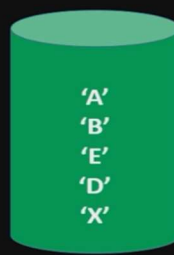
Add<int,float>(3,5);

- **Class Template:-**

Example



```
class Stack  
{  
    public:  
        int arr[5]  
    private:  
        push();  
        pop();  
}
```



```
class Stack  
{  
    public:  
        char arr[5]  
    private:  
        push();  
        pop();  
}
```

```
template <typename T>  
class Weight{  
    private:  
        T kg;  
  
    public:  
        void setData(T x)  
        {  
            kg=x;  
        }  
        T getData()  
        {  
            return kg;  
        }  
};  
  
int main() {  
  
    Weight <int>obj1;  
    obj1.setData(5);  
    cout<<"Value is: "<<obj1.getData();  
}
```

Q13. What is inline function?

Inline is a request to the compiler (not a command) to make a function as an inline function to reduce the overhead (time) of func calling. **If compiler treat a function as a inline function It substitute the code of function in a single line.** It must contain a single instruction.

```
inline return-type function-name(parameters)
{
    // function code
}

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

// Main function for the program
int main( ) {
    cout << "Max (20,10): " << Max(20,10) << endl;
    cout << "Max (0,200): " << Max(0,200) << endl;
    cout << "Max (100,1010): " << Max(100,1010) << endl;

    return 0;
}
```

Q Types of pointers:-

1. Void Pointer:-

Use:- malloc and calloc function returns a void pointer due to this reason , they can allocate a memory for any data type.

Void* malloc(size_t size);

WHAT IS A VOID POINTER?

Void pointer is a pointer which has no associated data type with it.

It can point to any data of any data type and can be typecasted to any type.

EXAMPLE:

```
int main()
{
    int n = 10;
    void *ptr = &n;

    printf("%d", *(int*)ptr);
    return 0;
}
```

OUTPUT: 10

WHAT IS A VOID POINTER?

Void pointer is a pointer which has no associated data type with it.

It can point to any data of any data type and can be **typecasted** to any type.

EXAMPLE:

```
int main()
{
    int n = 10;
    void *ptr = &n;

    printf("%d", *ptr);
    return 0;
}
```

We cannot dereference a void pointer.

OUTPUT: Error



2. **NULL pointer** :- when it initialize with NULL value. It uses for initialization error control.

3. **Dangling Pointer** is a pointer which points to some non-existing memory location.

```
Int * ptr= (int*) malloc(sizeof(int));
```

```
Free(ptr); // ptr is now dangling
```

Soln reinitialize with NULL value

4. **Wild Pointer**:- are also known as **uninitialized pointer**.

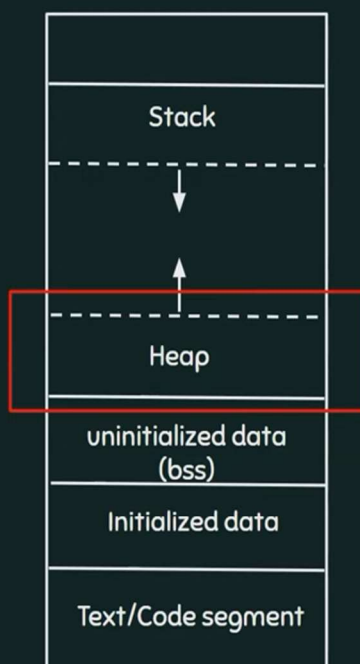
These pointers usually point to some arbitrary memory location and may cause a program to crash or misbehave.

```
Int *p;
```

```
*p=10
```

Static memory allocation: memory allocated at compile time and have fixed size like array.

Dynamic memory allocation: malloc memory alc , calc clear allocation



Heap is the segment of memory where dynamic memory allocation takes place

Unlike stack where memory is allocated or deallocated in a defined order, heap is an area of memory where memory is allocated or deallocated without any order or randomly.

There are certain built-in functions that can help in allocating or deallocating some memory space at run time.



WHAT IS MALLOC()

malloc is a built-in function declared in the header file <stdlib.h>

malloc is the short name for "memory allocation" and is used to dynamically allocate a single large block of contiguous memory according to the size specified.

SYNTAX: (void*)malloc(size_t size)

malloc function simply allocates a memory block according to the size specified in the heap and on success it returns a pointer pointing to the first byte of the allocated memory else returns NULL.

size_t is defined in <stdlib.h> as unsigned int.



```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int i, n;
    printf("Enter the number of integers: ");
    scanf("%d", &n);
    int *ptr = (int *)malloc(n*sizeof(int));

    if(ptr == NULL) {
        printf("Memory not available.");
        exit(1);
    }
    for(i=0; i<n; i++) {
        printf("Enter an integer: ");
        scanf("%d", ptr + i);
    }
    for(i=0; i<n; i++)
        printf("%d ", *(ptr+i));
    return 0;
}
```



WHAT IS CALLOC()?

calloc() function is used to dynamically allocate multiple blocks of memory.

It is different from malloc in two ways:



calloc() needs two arguments instead of just one

Syntax: void *calloc(size_t n, size_t size);

Number of blocks

Size of each block



EXAMPLE:

```
int *ptr = (int *)calloc(10, sizeof(int));
```

An equivalent malloc call:

```
int *ptr = (int *)malloc(10*sizeof(int));
```



DIFFERENCE #2



Memory allocated by calloc is initialized to zero

It is not the case with malloc. Memory allocated by malloc is initialized with some garbage value

NOTE:

malloc and calloc both return NULL when sufficient memory is not available in the heap.



WHAT IS REALLOC()?

realloc() function is used to change the size of the memory block without losing the old data.

SYNTAX: `void *realloc(void *ptr, size_t newSize)`

Pointer to the previously allocated memory.

New Size

On failure, realloc returns NULL.



EXAMPLE

```
int *ptr = (int *)malloc(sizeof(int));  
ptr = (int *)realloc(ptr, 2*sizeof(int));
```



This will allocate memory space of $2 \times \text{sizeof}(\text{int})$.



Also, this function moves the contents of the old block to a new block and the data of the old block is not lost.



We may lose the data when the new size is smaller than the old size.



Newly allocated bytes are uninitialized.



```
#include<stdlib.h>  
  
int main()  
{  
    int i;  
    int *ptr = (int *)malloc(2*sizeof(int));  
  
    if(ptr == NULL)  
    {  
        printf("Memory not available!");  
        exit(1);  
    }  
  
    printf("Enter the two numbers: \n");  
    for(i=0; i<2; i++) {  
        scanf("%d", ptr+i);  
    }  
  
    //Memory allocation for 2 more integers  
    ptr = (int *)realloc(ptr, 4*sizeof(int));  
    if(ptr == NULL)  
    {  
        printf("Memory not available!");  
        exit(1);  
    }  
  
    printf("Enter 2 more integers: \n");  
    for(i=2; i<4; i++)  
        scanf("%d", ptr+i);  
  
    //Printing the values on the screen  
    for(i=0; i<4; i++)
```



Q Goto:- The goto statement is a jump statement which is sometimes also referred to as **unconditional jump statement**. The goto statement can be used to jump from anywhere to anywhere within a function.

```
Cout<<"one"<<endl;  
    goto a; // a is label  
Cout<<"two"<<endl;  
Cout<<"three"<<endl;
```

a:

```
Cout<<"four"<<endl; // one four
```

C	C++	Java
1> It is <u>Procedural Language</u>	1> It is <u>object-oriented programming (oop) Language</u>	1> It is <u>pure object-oriented Language</u> .
2> It is <u>Compiler based Language</u>	2> It is <u>Compiler based Language</u>	2> It is <u>Compiler and Interpreter base</u> .
3> It <u>does not</u> Support <u>Inheritance</u> .	3> <u>Single, multilevel & multiple</u> Inheritance Support	3> <u>Single, multilevel & Hierarchical Inheritance</u> Support <u>but multiple Inheritance</u> Not Support
4> It is <u>platform Dependent</u>	4> It is <u>platform Dependent</u>	4> It is <u>platform Independent</u> .
5> It <u>does not</u> Support <u>multithreading and Interface</u> .	5> It <u>does not</u> Support <u>multithreading and Interface</u>	5> It Support <u>multithreading and Interface</u>
6> Not Support <u>database Connectivity</u>	6> Not Support <u>database Connectivity</u>	6> It Support <u>database Connectivity</u>
7> It Support <u>pointers</u>	7> It Support <u>pointers</u>	7> It <u>does not</u> Support

Destructor, pre-processor, c++ haven't default access specifier, support operator overloading, semi oop language.

What is Union ?

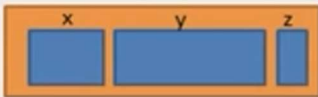
Union is similar to structure, except it allows you to define variables that share storage space.

What is union?

struct Item

```
{  
    int x;  
    float y;  
    char z;  
};  
struct Item i1;
```


i1



union Item

```
{  
    int x;  
    float y;  
    char z;  
};  
union Item i1;
```

i1



Satish Shukla Sir

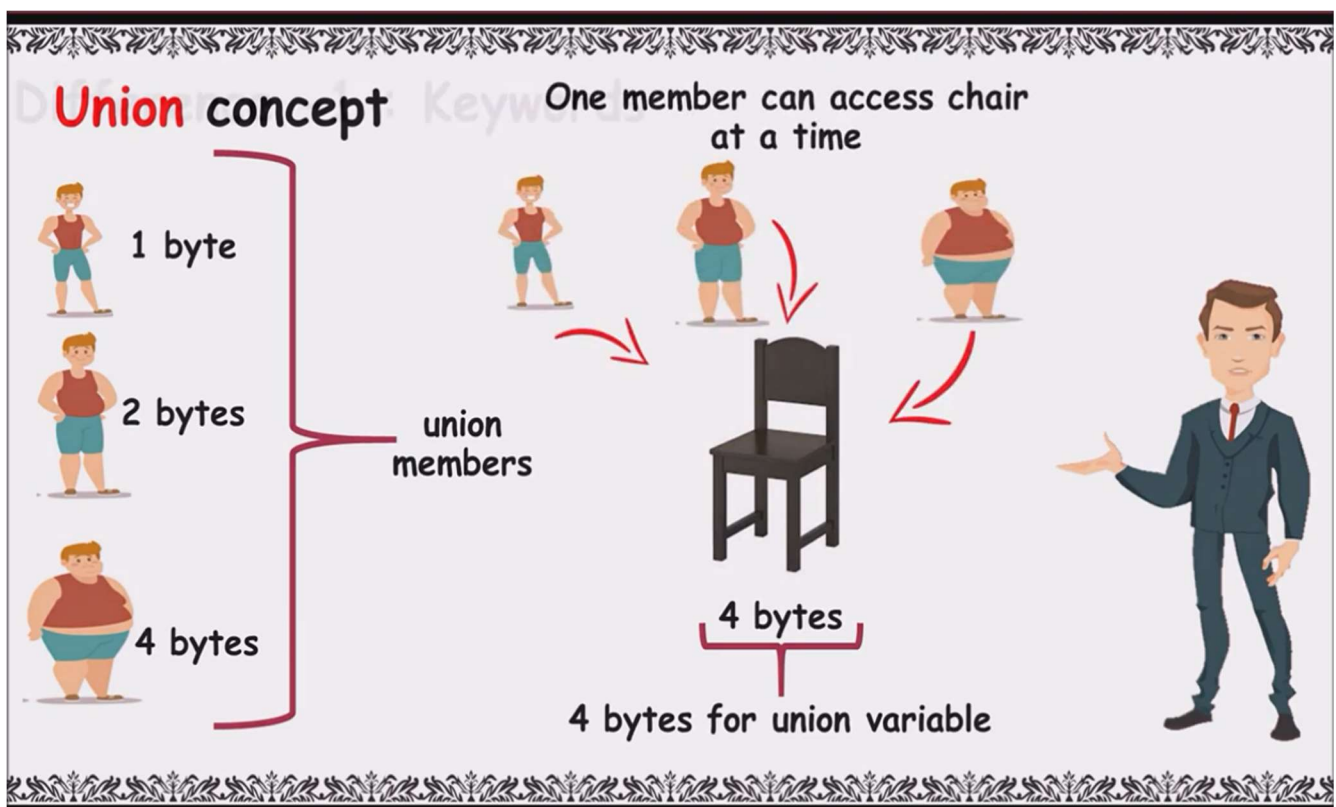
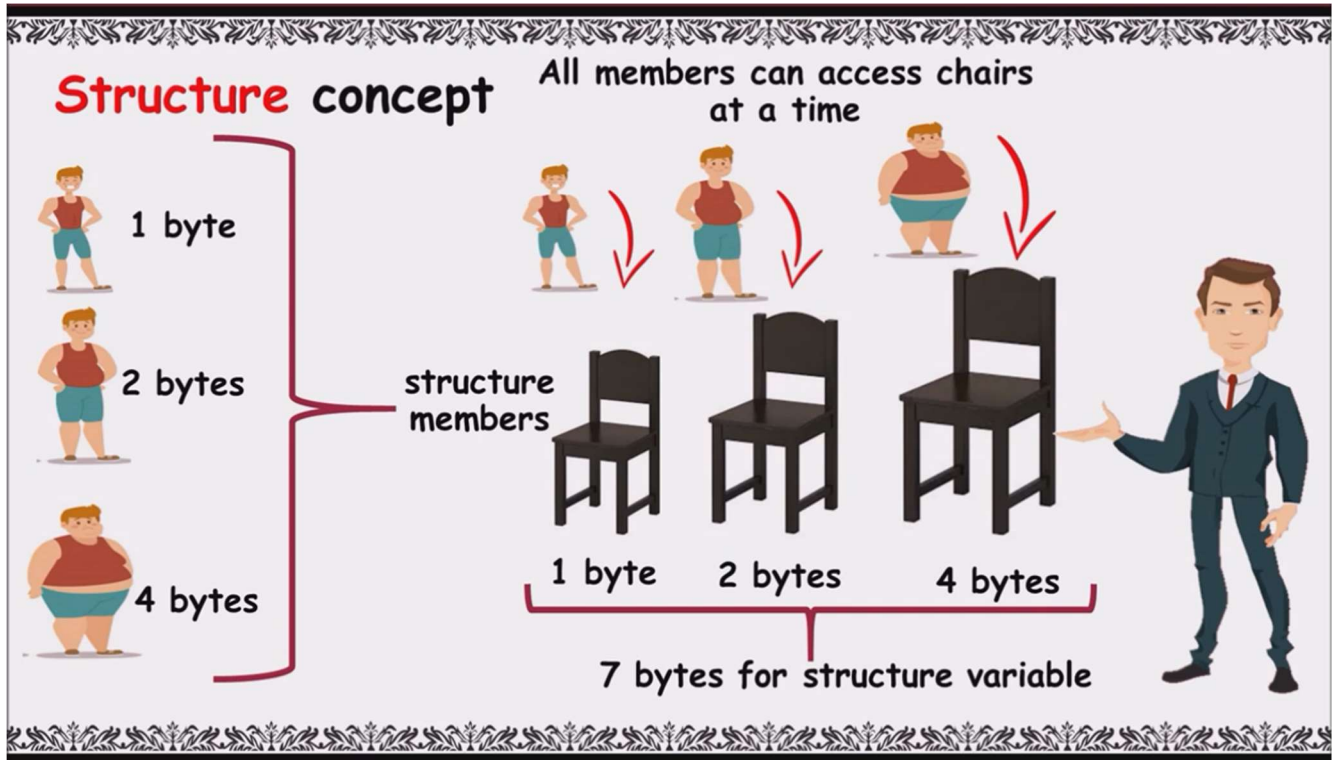
```
union Item{  
    int x ; float y ; char z;  
};
```



```
int main(){
```

```
    union Item i1;  
    //default value is 0 if we don't initialize  
    i1.x = 5;  
    cout<<i1.x<<endl; // 5  
    i1.y = 3.5;  
    cout<<i1.x<<endl; //garbage value  
    i1.z='a';  
    cout<<i1.z; //a
```

The member which occupy the larger memory among all variable is the size of union.
int a ; float b; char c; // size float



In union only one member are active at a time but in structure all member are active.

What is **Storage class** ? and its types?

A storage class defines the scope and lifetime of variables and functions.

Types:-

1. Auto
2. Static
3. Extern
4. Register

Storage Class	Keyword	Default Value	Storage	Scope	Life
Automatic	auto	Garbage	RAM	Limited to the block in which it is declared	Till the execution of the block in which it is declared
Register	register	Garbage	register	Same	same
Static	static	0 (zero)	RAM	Same	Till the end of program
External	extern	0 (zero)	RAM	Global	Same

Print hello world without semicolon

```
if(printf("Hello World")) {}
```

Swap two numbers without using third variable?

$A = a + b$

$B = a - b$

$A = a - b$

Print natural number without any loop?

```
void print(int n){//recursion
    if(n==0) return;
    print(n-1);
    cout<<n;
}

int i = 0;//using goto
```

label:

```
i = i + 1;
```

```
cout << i << " ";
```

```
if (i < 100) {
```

```
    goto label;
```

```
}
```

Actual parameter fun(5)

Formal parameter fun(int a)