OOP's

(class 1)

OOP's

01-04-2023.

Date.....

Basically, OOP's is type of programming techniaue, in which everything revolves around object, everything is done with the help of objects.

* what is an object?

In layman terms, object is a real world entity. For example, chair, car, pen, laptop, mouse, etc.

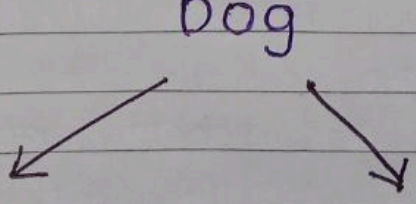
In other words, object is an entity that has a state & behaviour. Here, state means data & behaviour means functionality.

object is a runtime entity, it is created at runtime.

So, object is an entity, having 2 things:

- state / properties
- behaviour / methods.

For example:- Dog is an object.



state:
2 eyes,
4 feets,
2 ears

Behavious:
BarK,
Eat,
Sleep

* why object oriented programming is needed?

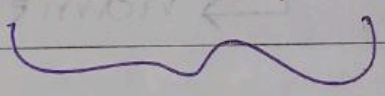
when we write the code using object oriented programming, then there are many benefits :-

i) It can related with real life problems.
ii) Easy to understand.
iii) code becomes more readable.
iv) we can also reuse the code.
v) Easy to maintain.

* what is a class?

Like we have some pre-defined datatypes -

```
int a;
string b;
bool c;
```



these are the pre-defined datatype.

suppose, we want to create our own datatype. In this condition, where we want to create our own custom datatype then we can use class.

→ A class is a blueprint or prototype that defines the variables and functions common to all object of a certain kind.
→ object is an instance of a class.

A class is a user defined data type declared with keyword 'class', it include data members & member functions, whose access is governed by the three access specifiers public, priva -te & protected.
By default, access to members of ctt class is private.

Syntax:- To create a class, the syntax is:

```
class car {
};
```

keyword

name of a class

currently the class is empty, but it still taking 1 byte of memory, because of its existance. If it was given 0 byte, then we won't be able to track its existance in the memory. 1 means, this class exists, 1 is minimum allocated space.

Date.....

```
class car{
};
```

Empty class will take 1 byte space because of its existence.

This class will take 8 bytes, because it contains 2 integers, i.e., 4 x 2 = 8 bytes.

```
class car{
    int price;
    int doors;
};
```

```
class car{
    int price;
    int doors;
    bool ignition;
};
```

Generally, int takes 4 bytes & bool takes 1 byte. So, this class should take 4 + 4 + 1, i.e., 9 byte But, it is taken 12 Bytes. Why??

Padding & Greedy Alignment.

In C++ class, there are member functi-ons, static data members. Do those have any contribution to the size of the class?
The answer is NO, only the non static data members contribute to the size of class. This is because static members have only one insta-nce which is shared among all objects, and normal member function are like executable code which does not have size like data members.

Date.....

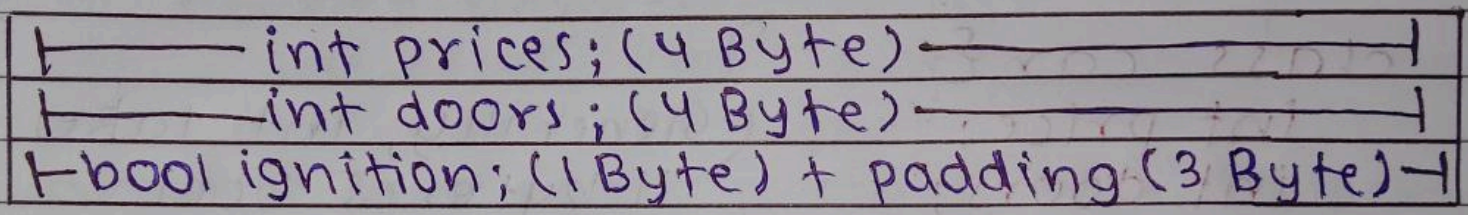
```
class car {
    int prices;
    int doors;
    bool ignition;
};
```

int prices; → 4 Byte

int doors; → 4 Byte

bool ignition; → 1 Byte + 3 Byte of Padding

In memory, it is represented like:-




```
class Animal {
    double c;
    double a;
    char b;
};
```

double c; → 8 Byte

double a; → 8 Byte

char b; → 1 Byte + 7 Byte of Padding.

```
class Animal {
    int a;
    char b;
    double c;
};
```

int a; → 4 Byte

char b; → 1 + 3 Byte of Padding

double c; → 8 Byte.

Date.....

The method of padding is however compiler dependent & kind of greedy. It aligns till the boundary of maximum memory allocated.

* **object creation** :-

```
Animal an;
```

name of the class. ↙ ↘ name of object

By default, the data members & member functions are marked as private. So, to access the data members & member functions we have to marked them as public :-

```
#include<bits/stdc++.h>
using namespace std;
class Animal {
  public:
    // state or properties
    int age;
    string name;
    // behaviour or functions
    void eat(){
      cout << "Eating";
    }
};
```

data members

member functions

Date.....

```
int main(){
    // object creation
    // static memory
    Animal an;
    cout << "age: " << an.age;
    cout << "name: " << an.age;
    return 0;
}
```

* **Access Modifiers :-**

```
class Animal {
    private:
        int a;
    public:
        int b;
};
```

private: } can be accessed within class

public: } can be accessed outside the class also.

we can't access private member of the class outside that class. If we try to do it, this will give an error. But we can do it with the help of getter and setter methods.

Date.....

```
#include<bits/stdc++.h>
using namespace std;
class Animal {
  private:
    int weight;
  public:
    void setweight(int w){
      weight = w;
    }
    void getweight(){
      cout << weight << endl;
    }
};
int main(){
  Animal an;
  an.setweight(154);
  an.getweight();
  return 0;
}
```

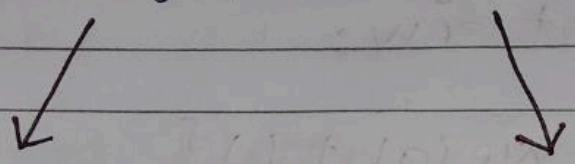
weight is a private data member in class Animal, but we can access it using public get & set methods. first we set the weight using setweight() method, then we access the weight using getweight() method.

Date.....

All the previous object creation is Static Memory Allocation.

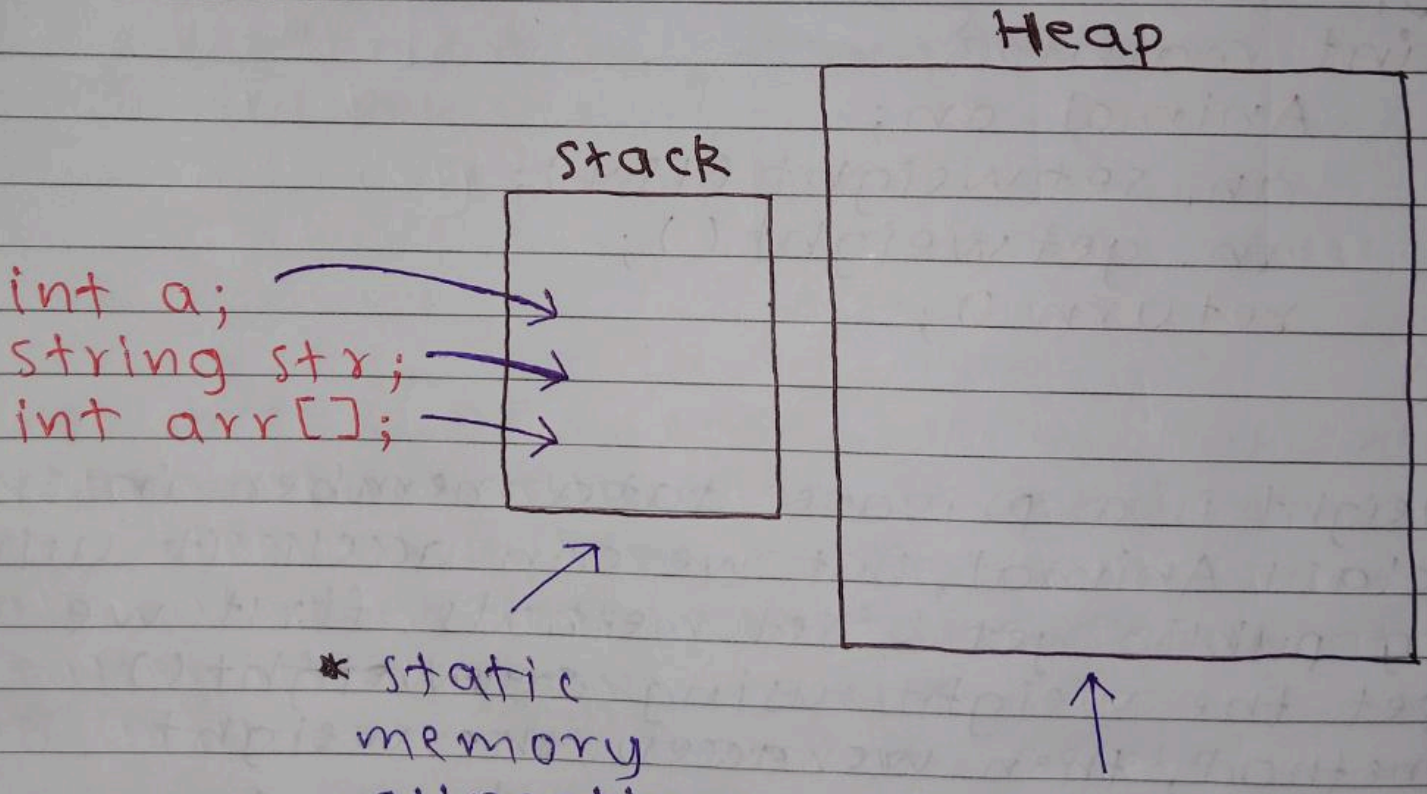
* **Dynamic Memory Allocation :-**

There are 2 types of memory availab -le in the system :-



Stack Memory

Heap Memory



* static memory allocation

* dynamic memory allocation

* Stack memory is small memory.
* Heap memory is very large memory

Date.....

* Using "new" Keyword, we can allocate memory dynamically.

```
int a;
```

static memory allocation

```
int *b = new int;
```

dynamic memo -ry allocation

"new int" means we've allocated a space of integer size from Heap memory.

"new int" return the address of the space in the memory. And as we know address is stored by pointer variables. That's why we write as:-

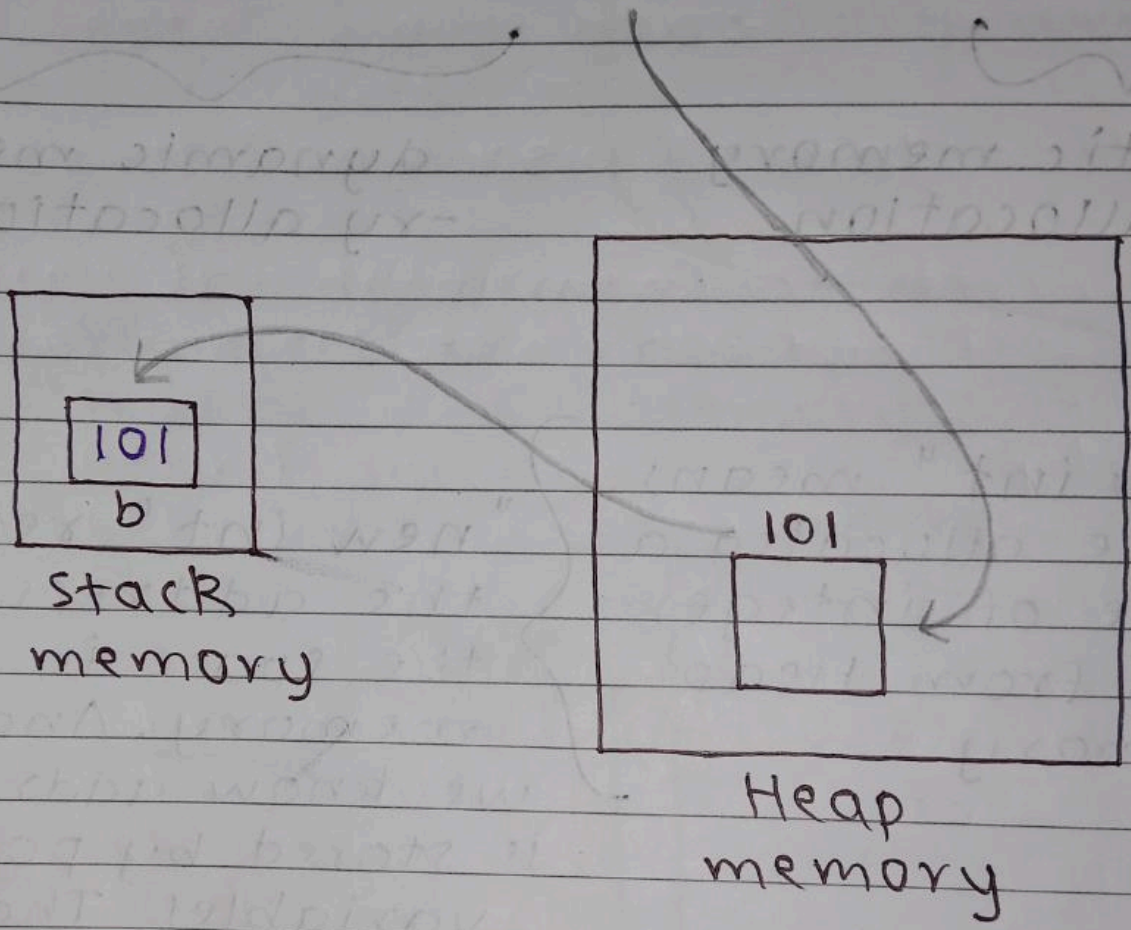
```
int *b = new int;
```

Here we created a new integer & this integer resides at Heap memory.

Date.....

Basically what happened is, the memory is created in Heap memory but the pointer is created in stack memory.

```
int *b = new int;
```



Note :- Space allocated in stack is cleaned automatically, but space allocated in Heap won't be deallocate automatically.
To deallocate the memory, 'delete' keyword is used, else heap memory will get full.

"new" keyword → for allocation
"delete" keyword → for deallocation

Date.....

Memory Leak & its prevention.

Memory Leak occurs in C++, when the programmer has allocated memory with the help of new keyword & then forget to delete the memory allocated with the delete keyword.
Memory Leak occurs when we use the wrong delete operator. The delete operator should be use to free a single allocated memory space whereas delete[] is used to free an array of data values.
To prevent memory leak :-
i) Always deallocate the memory after use.
ii) we should have few new/delete calls in program level - ideally none.

Garbage collection

Garbage collection is an automated process of deleting code that is no longer needed or used. This automa-tically frees up the memory space.
The garbage collector consider unrea-chable objects garbage & release the memory that is allocated by them.
During the collection, it examines

Date.....

the managed heap, looking for blocks of address space occupied by unreacha -ble objects.

Using Dynamic Memory Allocation, we can create objects also:-

```
Animal *bn = new Animal;
```

To access the data members from the pointer object, they syntax is:-

```
(*bn).age = 5;
(*bn).name = "Dog";
```
} 1st way

```
bn -> age = 10;
bn -> name = "Cat";
```
} 2nd way

This Keyword

```
#include<bits/stdc++.h>
using namespace std;
class Animal {
   //data members
   private:
     int weight;
   public:
     int age;
     string name;
     //member functions
     void eat(){
        cout << "eating";
     }
     void sleep(){
        cout << "sleeping";
     }
     void setweight(int w){
        weight = w;
     }
     void getweight(){
        cout << "weight of animal
           is:" << weight;
     }
};
int main(){
   Animal *an = new Animal;
   an -> age = 10;
   an -> name = "cat";
   cout << "Age:" << an -> age;
   cout << "Nam:" << an -> name;
```

Date.....

```
an -> eat();
an -> sleep();
an -> setweight(20);
an -> getweight();
return 0;
}
```

In setweight() function, we've written that,

```
void setweight(int w){
    weight = w;
}
```

Here, the data member name & the incoming parameter variable name, both are different, so there is no issue.

But, what if the both names are same ??

```
void setweight(int weight){
    weight = weight;
}
```

in this situation, actual weight variable is set by garbage value.

Date.....

In this situation, compiler will confu -se that which weight variable to be refer. So, how can we differenti -ate both variables ??
This is done using "this" Keyword.

"this" Keyword is a pointer to current object -> jis bhi object Ke liye setweight() wala function call hua hai, us object KO current obje -ct bolte hai and "this" is a pointer to that current object.

so we can differentiate both the same name variable using "this" Keyword, like this :-

```
void setweight(int weight){
    this -> weight = weight;
}
```

Note:- "this" Keyword cannot be used outside the class.

Also we can write it like this :-

```
(*this).weight = weight;
```

constructor

When an object of a class is created, the very first thing is it will call the constructor.
The job of constructor is to initialize object, it is having a same name as that of the class & it doesn't have any return type.
If we don't create a constructor then the class will automatically create a default constructor, and call it.

```
#include<bits/stdc++.h>
using namespace std;
class Animal{
  public:
    int age;
    void eat(){
      cout << "eating";
    }
    // constructor
    Animal(){
      cout << "constructor called...";
    }
};
int main(){
  Animal *an = new Animal;
  an -> age = 10;
  cout << "Age:" << an -> age;
  an.eat();
  return 0;
}
```

Date.....

The first thing that will print while running the code is :-

"constructor called..."

* Default constructor :-

```
Animal(){
  cout << "default constructor
          called.....";
}
```

* Parameterized constructor :-

```
Animal(int age, int weight){
  this -> age = age;
  this -> weight = weight;
  cout << "parameterized
          constructor called...";
}
```

can we make a constructor private ?
Yes, we can declare a constructor as private.
If we declare a constructor as private we are not able to create an object of that class.

* copy constructor :-

A copy constructor are the member functions of a class that initialize the data members of the class using another object of a same class. It copies the values of the data varia-ble of one object of a class to the data members of another object of the same class.

```
class Animal {
  public:
    int age;
    int weight;
    Animal(int age, int weight) {
        this -> age = age;
        this -> weight = weight;
        cout << " parameterized
                 constructor called... ";
    }
    Animal(Animal obj) {
        this -> age = obj.age;
        this -> weight = obj.weight;
        cout << " copy constructor
                 called... ";
    }
};
int main() {
    Animal *ab = new Animal(10, 20);
    Animal *ac = ab;
    return 0;
}
```

parameterized constructor

copy constructor

Date.....

The copy constructor we've created will give an error, because when we create an object it will first call the constructor. constructor is called, in input para-meter we passed another object of same class & this parameter is pass by value, so it will create a copy of object, & when the copy is created it will again call the contructor & this loop is again & again repeated & converted into infinite loop.

To prevent this loop, we should pass the parameter by reference, like this :-

```
Animal(Animal &obj){
    this -> age = obj.age;
    this -> weight = obj.weight;
    cout << "copy constructor
            called...";
}
```

Date.....

```
class Animal{
  public:
    int age;
    int weight;
    Animal(int age, int weight){
        this -> age = age;
        this -> weight = weight;
        cout << "parameterized
                constructor called...";
    }
    Animal (Animal &obj){
        this -> age = obj.age;
        this -> weight = obj.weight;
        cout << "copy constructor
                called...";
    }
};
int main(){
    Animal ab(10, 20);
    Animal ac = ab;
    Animal ad(ac);
    return 0;
}
```

} both are the ways to create copy of an object & create copy constructor.

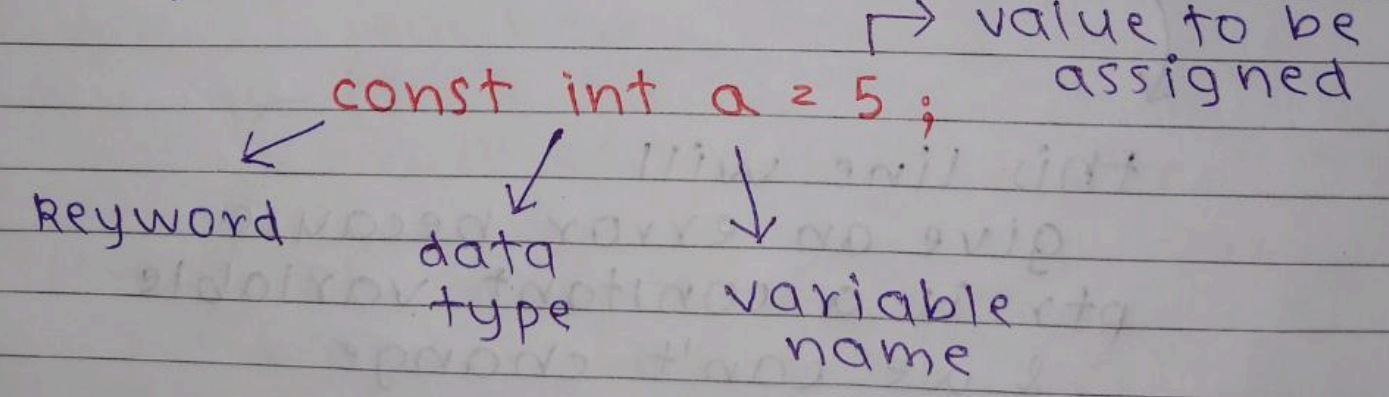
Date.....

* what is the use of copy constructor
By default, the constructor will do shallow copy, to avoid shallow copy, if we want to do deep copy by ourself, then we have to write our own copy constructor
Means,
To do deep copy by ourself we have to create our own copy constructor.

const Keyword

we use 'const' Keyword to define a constant value that can not be changed during the execution.

i) const variable = It defines variable values that can never be changed.
Syntax :-



Now, if we try to do any modifica -tions to a, we will get an error.

Date.....

ii) constant pointer :- A constant pointer cannot change the address of the variable to which it is pointing, i.e., the address will remain constant. Therefore, we can say that if a constant pointer is pointing to some variable, then it cannot point to any other variable.

Syntax :-

```
int *const ptr;
```

Example :-

```
int a = 1;
int b = 2;
int *const ptr = &a;
cout << a;
cout << *ptr;
ptr = &b;
```

this line will give an error because ptr is a constant variable & we can't change

Date.....

iii) Pointer to constant: A pointer to constant is a pointer through which the value of the variable that the pointer points cannot be changed. The address of these pointers can be changed, but the value of the variable that the pointer points cannot be chang-ed.

Syntax:-

```
const int *ptr;
```

→ First, we write the code where we are changing the value of a pointer:-

```
int a = 100;
int b = 200;
const int *ptr;
ptr = &a;
ptr = &b;
cout << "value of ptr is " <<
                        ptr;
```

this is possible, because we are changing the value of pointer

Date.....

→ Now, we write the code in which we are changing the value of the variable to which the pointer points:-

```
int a = 100;
int b = 200;
const int *ptr;
ptr = &b;
*ptr = 300;
cout << "value of ptr is: " <<
                        *ptr;
```

this is not possible, because we cannot change the value of the variable to which the pointer pointer.

Date.....

iv) constant pointer to a constant :
A constant pointer to a constant is a pointer, which is a combination of the previous two pointers. It can neither change the address of the variable to which it is pointing nor it can change the value placed at this address.
Syntax :-

```
const int * const ptr;
```

v) constant function argument : If the value of function argument is constant, then function can't change its value.
Syntax :-

```
int test ( const int x ) {

}
```

test function can't change the value of x.

vi) const member function of a class :
A const member function of a class can never changes the value of any class data member & also never calls any non-const function.
Syntax :-

Date.....

```
class Animal{
  public:
    int a = 10;
    void eat(){
      cout << "eating";
    }
    void sleep() const {
      cout << "sleeping";
      cout << a;
      a = 100;
      cout << a;
      eat();
    }
};
int main(){
    Animal an;
    an.sleep();
    return 0;
}
```

this is not possible because we cannot change the value of any class data member in const member function.

this is also not possible because we cannot call a non const function in a const function.

vii) constant objects = The value of data members can never change till the life of the object in the program. They are also known as read-only objects.

Syntax:-

```
const Animal an;
```

static Keyword

when "static" Keyword is used, variables or data members or functions cannot be modified again. It is allocated for the lifetime of program. Static functions can be called directly by using class name. Static variables are initialized only once.

Syntax:-

```
void demo(){
    static int count = 10;
    cout << count;
    count++;
}
int main(){
    for(int i=0; i<5; i++){
        demo();
    }
    return 0;
}
```

Date.....

The previous program will print, 10, 11, 12, 13, 14. It is incrementing value each time because of static keyword.
If we didn't use static keyword, then output will be 10, 10, 10, 10, 10.

* static member function :-

Static member function in a class is the function that is declared as static.

i) A static member function is indepen -dent of any object of the class.
ii) A static member function can be called even if no object of the class exist.
iii) A static member function can also be accessed using the class name through the scope resolution opera -tor.
iv) A static member function can access static data member & static member functions inside or outside of the class.
v) static member function have a scope inside the class & cannot access the current object pointer.

Date.....

The reason we need static member function :-

i) static members are frequently used to store information that is shared by all objects in a class.

ii) For instance, we may keep track of the quantity of newly generated objects of a specific class type using a static data member as a counter. The static data member can be increased each time an object is generated to keep track of the overall number of objects.

static member function are allowed to access only static data member or static member functions of the class.

```
class Animal{
  public:
    int a;
    static void sleep(){
      cout<<"sleeping";
    }
};
int main(){
    Animal an;
    an.sleep();
    Animal::sleep();
    return 0;
}
```

we can access static member functions like this also.

Spiral

Date.....

Initializer List

Initializer List is used in initializing the data members of a class. The list of members to be initialized is indica-ted with constructor as a comma separated list followed by a colon.
For example :-

```
class Point {
    private:
        int x;
        int y;
    public:
        Point(int i, int j) : x(i), y(j){
        }
        void getX(){
            cout << x;
        }
        void getY(){
            cout << y;
        }
};
int main(){
    Point pn(10, 15);
    pn.getX();
    pn.getY();
    return 0;
}
```

initializer list

Date.....

Destructor

Destructor is used to free the memory.

→ In static memory allocation, destr-uctor is called automatically.

→ In dynamic memory allocation, we need to call the destructor manually, using delete keyword.

i) destructor has the same name as the class.
ii) no return type.
iii) It has a '~' tild sign in front.

* static memory allocation :-

```
#include<bits/stdc++.h>
using namespace std;
class Animal{
  public:
    ~Animal(){
      cout << "Destructor
                called...";
    }
    Animal(){
      cout << "constructor
                called...";
    }
};
int main(){
    Animal an;
    return 0;
}
```

→ destructor will automatically called.

Date.....

* dynamic memory allocation :-

```
#include<bits/stdc++.h>
using namespace std;
class Animal {
  public:
    ~Animal(){
        cout << "Destructor...";
    }
    Animal(){
        cout << "constructor...";
    }
};
int main(){
    Animal *an = new Animal();
    // manually call destructor
    delete an;
    return 0;
}
```