2. C++ Destructor.

A Destructor works opposite to constructor ; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically

A destructor. is defined like a constructor. It must have same name as a class. But it is prefixed with a tilde (~) sign.

→ C++ destructor cannot have parameters. Moreover, modifier can't be applied on destructors.

ex:.

```
# include <iostream>
using namespace std:
class Employee
{
 public:
     Employee ()
     {
      cout << "constructor invoked" << endl;
     }
     ~ Employee ()
     {
       cout << "Destructor invoked" << endl;
     }
};
```

```
int    main () {

        Employee  e1;

        Employee  e2;

        return 0;

    }
```

<u>output</u>

```
            constructor   invoked
            constructor   invoked
            destructor    invoked
            destructor    invoked.
```

→ Destructor clears the memory occupied by the object.

→ Destructor is called upon when the scope of the object is exiked.

C++ This pointer

In C++, this is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.

(i) It can be used to pass current object as a parameter to another method.

(ii) It can be used to refer current class instance variable.

(iii) It can be used to refer indexers.

example → where this refers to fields of current class

```
class Employee {
  public:
        int id;  string name;  float salary;
      Employee (int id, string name, float salary) {
          this → id = id;
          this → name = name;
          this → salary = salary;
          }.
      void display () {
          cout << id << " " << name << " " << salary;
        }
  };
```

```
int main () {
    Employee e1 = Employee (101, "Sonoo", 8900);
    Employee e2 = Employee ( 102, "Nahul", 5900);
    e1 . display ();
    e2 . display ();
}
```

Output

```
101.   Sonoo   8900
102    Nahul   5900.
```

this is basically a pointer of the object to itself.

C++ static

In C++, static is a keyword or a modifier that belongs to the type not instance. So instance is not required to access the static members.

In C++, static can be field, method, constructor, class, properties, operator and event.

→ Advantages of C++ static

- Memory efficient - Now, we don't need to create instance for accessing the static members, so it saves memory. Moreover, it belongs to the

};

type, so it will not get memory each time when instance is created.

⇒ C++ static field

A field which is declared as static is called static field. Unlike instance field which gets memory each time whenever you create object, there is only 1 copy of static field created in memory. It is shared to all the objects.

It is used to refer the common property of all objects such as rate of interest in case of Account, company name in case of Employees etc.

example

```
class  Account {
public :
        int  accno ;   string name;
        static float rateofInterest ;
        Account ( int accno,  string name) {
             this → accno  =  accno;
             this → name   :  name;
        }
      void display() {.
         cout  <<  accno  << "  "<<name<< rateofInterest
                                                  <<
                                                  endl.
    }
};
```

```
float   Account :: rateofInterest  = 6.5;

int  main () {

     Account  a1  =  Account (201, "Sanjay") ;

     Account  a2  =  Account ( 202, "Nakul") ;

     a1 . display () ;
     a2 . display () ;
```

Output

```
  201   sanjay    6.5
  202   Nakul     6.5
```

⇒ static example → counting objects.

```
class  Account {
   public:
          int accno ;  string name;
          static  int  count ;

          Account ( int accno,  string  name ) {

               this → accno   = accno ;
               this →  name   = name ;
               count ++
          }

     void display () {
          cout  <<  accno .  <<  "\n" ;
          cout  <<  name     <<  "\n" ;

   }; }
```

```
int   Account :: count = 0 ;

int  main () {

      Account   a1  =  Account ( 201, "Sanjay" ) ;

      Account   a2  =  Account ( 202, "Nqhul" ) ;

      Account   a3  =  Account ( 203,  " manan" ) ;


      cout  <<  " Total objects "  <<  count << endl;

  }
```

Output :

```
        3
```