Class 11

Class 4 c++ OOP

this Pointer & Copy Constructor

---

1. this pointer concept

the `this` pointer is a special implicit pointer present inside every non-static member function by default.

it always points to the current object that invoked the function.

syntax → `this` (keyword).

used to differentiate between class attributes and function parameters having the same name.

used for chained function calls and returning the current object reference.

2. parameterized constructor using this pointer

```
A(int id,string name,double salary)
{
    cout<<"Parameterized called"<<endl;
    this->id = id;
    this->name = name;
    this->salary = salary;
}
```

inside the constructor, parameters and data members have the same names.

the `this` keyword clarifies that `this->id` refers to the class attribute, while `id` refers to the parameter.

so the assignment becomes attribute = parameter.

it connects the current object's variables with the function arguments.

3. using this pointer inside member functions

```
void show()
{
    cout<<"Id :"<< this->id<<endl;
    cout<<"name :"<< this->name<<endl;
    this->show_sal();
}
```

the `this` pointer can call another member function using `this->functionname()`.
here, `this->show_sal()` is used to call a function of the same class.
it's optional, but it clearly shows that the call belongs to the current object.

4. complete demonstration

```
A o2(1087 , "Othoy" , 100000000);
o2.show();
```

when the object `o2` is created, the parameterized constructor runs.
the constructor assigns the values of parameters to the object's data members through the `this` pointer.
the `show()` function prints data using `this->id`, `this->name`, and `this->salary`.
the output displays the initialized attributes of the same object.

5. copy constructor syntax

```
A(const A &o)
{
    cout<<"Copy constructor called"<<endl;
    id = o.id;
    name = o.name;
    salary = o.salary;
}
```

syntax → `classname(const classname &obj){...}`

the copy constructor is used to create a new object using an existing one.

the parameter must be passed by reference, otherwise it will cause infinite recursion.

it copies all data members of one object into another.

6. rules of copy constructor calling

the copy constructor is called automatically in three cases:

1. when an object is passed by value to a function
2. when an object is returned by value from a function
3. when a new object is initialized with an existing object

7. examples of copy constructor calls

```
A o1(1087, "Othoy", 100000000);
A o3 = o1; // calls copy constructor
```

when `o3` is initialized with `o1`, the copy constructor is invoked automatically.

it prints “Copy constructor called” and copies all attributes.

`o3.show()` displays the same data as `o1`.

8. pass by value call

```
void f(A ob)
{
    cout<<"f called"<<endl;
}
```

when `f(o3)` is called, the object `o3` is passed **by value**, creating a new temporary copy.

this triggers the copy constructor again before entering the function.

9. return by value

```
A ff()
{
    A t;
    return t;
}
```

when a function returns an object by value, a temporary object is created to hold the return value. this again invokes the copy constructor internally.

in `A n = ff();`, the constructor of `t` is called first, and then the copy constructor copies it to `n`.

10. object assignment vs copy construction

object assignment (using `=`) does not call the copy constructor; it uses the **assignment operator**.

the copy constructor only works when a new object is being initialized at creation time.

for example,

`A o3 = o1;` → copy constructor

`o3 = o1;` → assignment operator