C++ in depth

Constructor

Saurabh Shukla (MySirG)

Agenda

1. Constructer
2. Parameterized Constructer
3. Constructor Overloading
4. Default Constructer
5. Copy Constructer
6. shallow copy vs Deep Copy

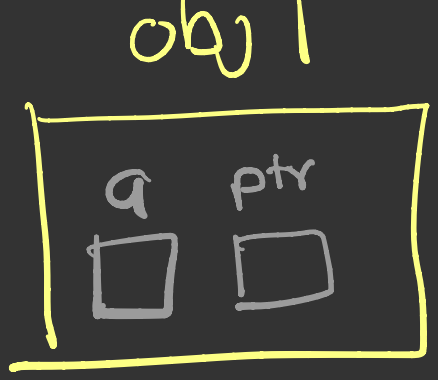
Constructor

- Constructor is a special member of the class whose name is same as the name of the class
- Constructor has no return type
- Constructor is invoked at the time of object creation (automatically)
- Constructor is an instance member
- Usually constructor is defined as public member but it can be private also.

- Programmer has to define Constructor, so he can write any code but it is useful to initialize properties of an object.

Why you need constructer ?

To initialize member variables of an object.




```
{
    ptr = NULL;

}
```

```
void setData(int x, int y)
{
    a = x;
    if ( ptr == NULL)
        ptr = (int*) malloc(4);
    *ptr = y;
}
```

Parameterized Constructor

- You can make a constructor with arguments.
- Constructor arguments are passed at the object creation.

Constructor Overloading

- Programmer can provide multiple constructors in the class with different signatures.

Default Constructor

- When programmer doesn't provide explicit Constructor in the class, compiler creates an empty body, no argument Constructor in the class

Copy Constructor

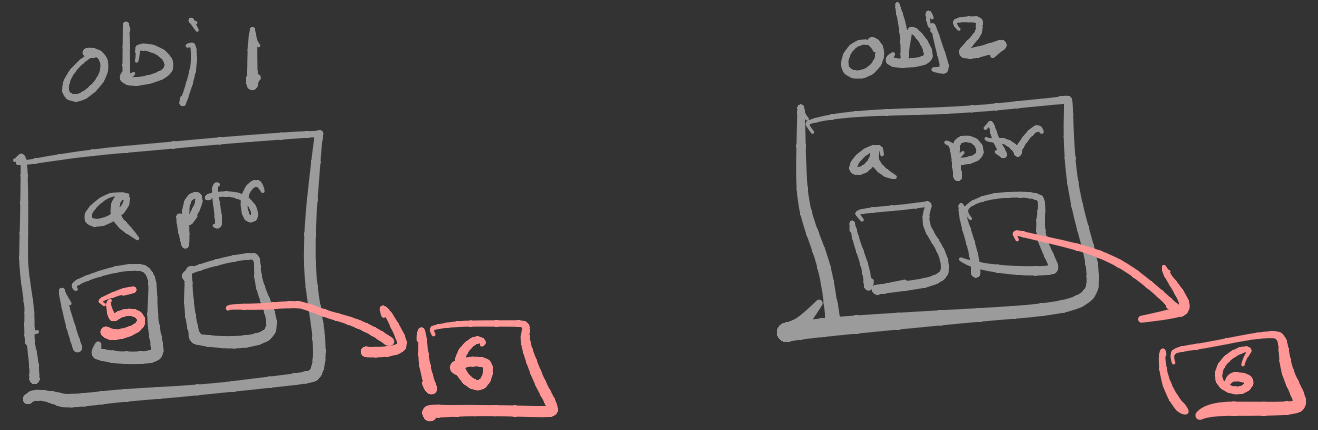
- Either programmer has to provide copy constructor in the class or compiler itself provides copy constructor.
- Copy constructor is invoked for newly created object which is initialized with the object of the same class
- Formal argument of copy constructor must be a reference variable of same class

① when there is no explicit constructor defined in the class Compiler defines two constructors
- default constructor
- Copy constructor

② when there is at least one explicit constructor of any type Compiler doesn't provide default constructor

③ when there is explicit copy constructor in the class compiler defines none

Shallow Copy vs Deep Copy




```
ClassName obj2 = obj1;
```

```
{
    a = obj.a;
    ptr = (int*) malloc(4);
    *ptr = *(obj.ptr);
}
```