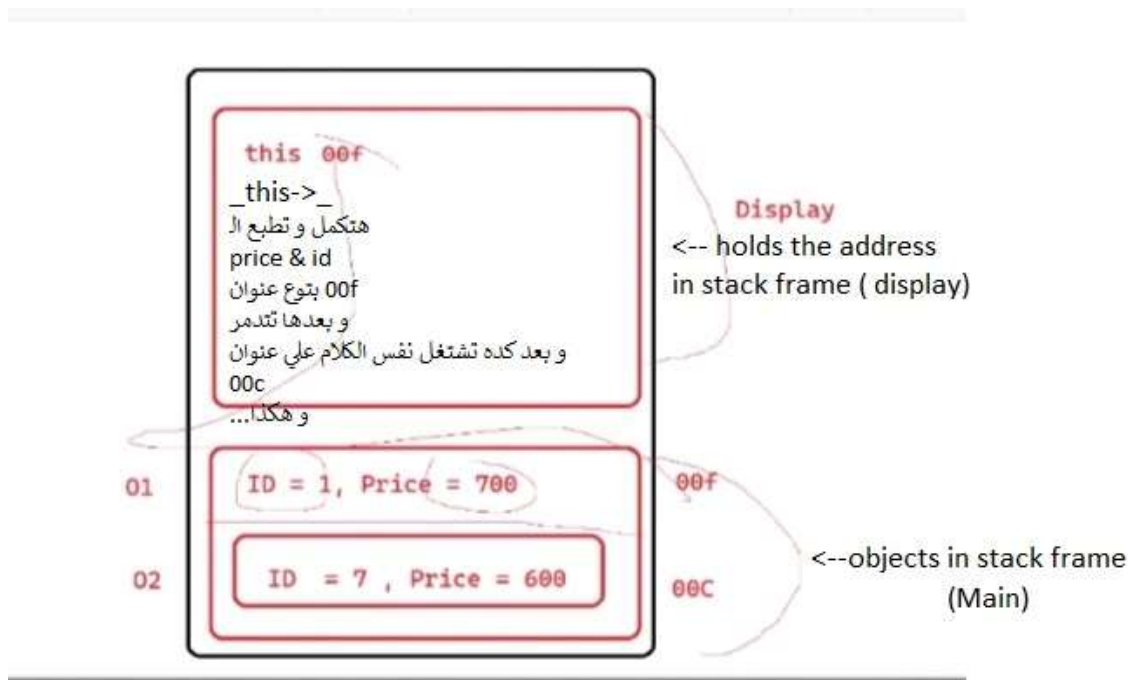


Day 3

Will continue **Encapsulation** & start with **Polymorphism**



Constructor Function

هنعمل function انده عليها مرة واحدة بس (افتح حساب مثلا) و هبقي محتاج ابدأ بقيمة ابتدائية initial value

فمش هينفع تتكرر ثاني الا مع object جديد _نسخة من الـ object

و دي **special function** و كمان الـ **compiler** هو اللي هينده عليها (مش انا) .. و هتبقى باسم يجبرني عليه (باسم الـ data type اللي عملته ف البداية)

And it **will not return** value...

مش هترجع لي اي حاجة .. لان اصلا هدفها تحط قيمة ابتدائية..

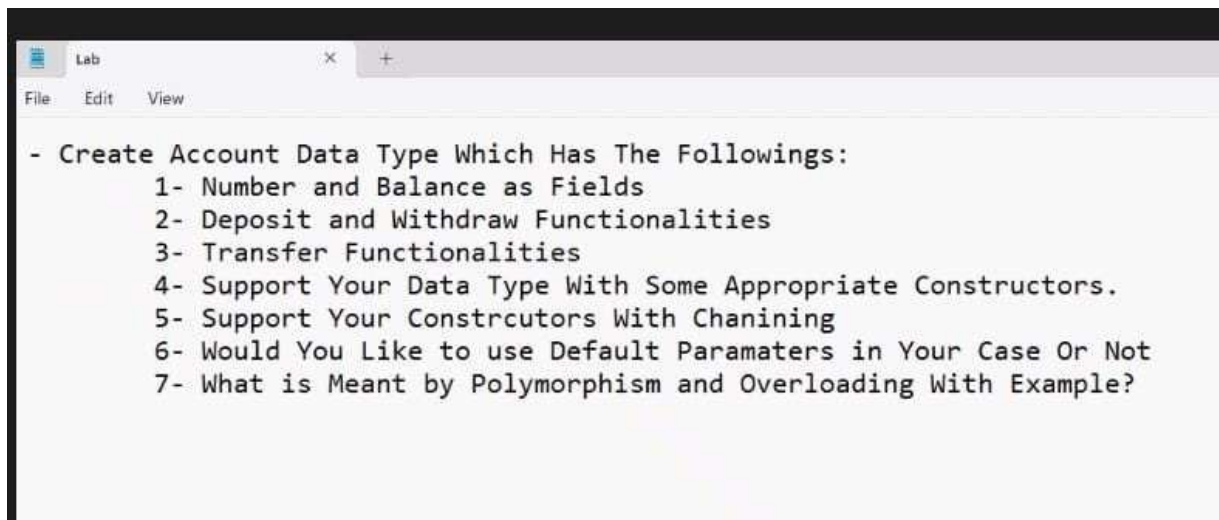
Polymorphism

دي بقي لما يكون عندي نفس الـ function بس بتعمل اكر من شغلانة (تعدد الواجهه)
الـ **overloading** دي شكل من اشكال الـ Polymorphism .. (تقدر تاخذ اكر من شكل)
فانكشنز ليهم نفس الاسم بس مختلفين ف الـ **signature** ...
-> الـ signature دي كده زي العلامات اللي بتفرق لي بينهم من حيث
(النوع _ العدد _ الترتيب).
يبقي كده الـ parameter هي اللي بتفرق بين الـ function لما اجي انده عليها ..

نكمل بكرة بقي معلىش ☹

بس نعمل تاسك النهارده علشان مهم..

Lab3



```
Lab
File Edit View
- Create Account Data Type Which Has The Followings:
  1- Number and Balance as Fields
  2- Deposit and Withdraw Functionalities
  3- Transfer Functionalities
  4- Support Your Data Type With Some Appropriate Constructors.
  5- Support Your Constructors With Chanining
  6- Would You Like to use Default Paramaters in Your Case Or Not
  7- What is Meant by Polymorphism and Overloading With Example?
```

Answer

```
4 using namespace std;
5 struct account
6 {
7     private:
8         int number;
9         float balance;
10    public:
11        //there are 3 functions(different tasks) but have the same name - different signature
12        account(): account(0,0){
13        }
14        account(int number): account (number,1500){
15        }
16        account(int number,float balance){
17            this->number=number;
18            this->balance=balance;
19        }
20
21        void deposit(float _amount)
22        {
23            balance+= _amount;
24        }
25        void withdraw(float _amount)
26        {
27            balance= balance- _amount;
28        }
29        void transfer(account &b, float _amount)
30        {
31            b.deposit(_amount);
32        }
33        void display()
34        {
35            cout << "number :" << number << endl
36              << "balance:" << balance;
37        }
38    }
```

```
34    {
35        cout << "number :" << number << endl
36          << "balance:" << balance;
37    }
38    };
39
40    //standalone function (outside the scope)
41
42    /* void display()
43    {
44        cout << "number :" << number << endl
45          << "balance:" << balance;
46    }*/
47
48
49    int main()
50    {
51        account a= account();           // initial value - constructor function
52        account b= account(6);
53        account c= account(3,1000);
54        a.display();
55        b.display();
56        c.display();
57
58        // account a = account();
59
60        // display(a); standalone
61
62
63
64
65        return 0;
66    }
67
```