Objective:

- It will help you understand the issues related to shallow copy and to get a grip on array of structures.

Task-1: MonthlyBudget
A student has established the following monthly budget:

Housing	500.00
Utilities	150.00
Household Expenses	65.00
Transportation	50.00
Food	250.00
Medical	30.00
Insurance	100.00
Entertainment	150.00
Clothing	75.00
Miscellaneous	50.00

Write a program that has a MonthlyBudget structure designed to hold each of these expense categories. The program should pass the structure to a function that asks the user to enter the amounts spent in each budget category during a month. The program should then pass the structure to a function that displays a report indicating the amount over or under in each category, as well as the amount over or under for the entire monthly budget.

Task-2: Set struct as we discussed in class today

➔ *Third version of set library connected with Lab-1:*

Struct and parameter passing

In lab-1, In both versions of set functions, you might have noticed that whenever we need to make a set we need to give different name to data, capacity and noOfElements identifier.
We should group them together in struct because they together give a meaning to a Set and will be easy to handle and more natural to code.

```
struct Set
{
        int * data;
        int capacity;
        int noOfElements;
};
```

So, we need to change the prototypes accordingly.

Operations

void createSet (Set & s, int size)
void insertElement (Set & s, int elem)
void removeElement (Set & s, int elem)

void displaySet(Set s)
Set intersection (Set s1, Set s2)
Set union (Set s1, Set s2)
Set difference (Set s1, Set s2)
int isSubSet (Set s1, Set s2)
It return 0 if both sets are equal, returns 1 if s1 is proper subset of s2, otherwise returns -1.

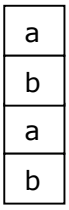
The 3rd version is quite easy to handle, easy to code, easy to understand.

But Soon You will see even more elegant solution using Object Orientation just a few steps away ☺.

Task-3: Focus on nested struct object layout:
Consider the following C/C++ structs:

```
struct A
{
        int a;
        int b;
};
```

```
struct B
{
        int a;
        int b
        A comp;
};
```



B obj;
Struct B object layout

One of your colleagues has following concepts/confusion in mind considering the above structs. Logically comment about his/her concept: whether she/he is right or wrong.

According to him/her, In the third column: object layout apparently looks ambiguous because there are two 'a' and two 'b'. When we shall do 'obj.a', which 'a' it will access?

Task-4:
You *must* implement yourself the Set Application that we discussed in lecture-3 today, it might help you in lab-2.

Task-5: You may look at this task after Lecture-4. **Drink Machine Simulator**
Write a program that simulates a soft drink machine. The program should use a structure that stores the following data:

Drink Name
Drink Cost
Number of Drinks in Machine

The program should create an array of five structures. The elements should be initialized with the following data:

Drink Name	Cost	Number in Machine
Cola	.75	20
Root Beer	.75	20
Lemon-Lime	.75	20
Grape Soda	.80	20
Cream Soda	.80	20

Each time the program runs, it should enter a loop that performs the following steps: A list of drinks is displayed on the screen. The user should be allowed to either quit the program or pick a drink. If the user selects a drink, he or she will next enter the amount of money that is to be inserted into the drink machine. The program should display the amount of change that would be returned and subtract one from the number of that drink left in the machine. If the user selects a drink that has sold out, a message should be displayed. The loop then repeats. When the user chooses to quit the program it should display the total amount of money the machine earned.

Input Validation: When the user enters an amount of money, do not accept negative values, or values greater than $1.00.