URBAN EDGE

Chapter 9 - Structures

Arrays and strings ⇒ Similar data (int, float, char)

Structures can hold ⇒ dissimilar data

Syntax for creating Structures
A C structure can be created as follows:

```
struct employee {
    int code;
    float salary;
    char name[10];
};
```

⇒ This declares a new user defined data-type!

→ Semicolon is important

We can use this user defined data type as follows:

```
Struct employee e1;
Strcpy(e1.name, "Harry");
e1.code = 100;
e1.salary = 71.22;
```

⇒ Creating a structure variable

So a structure in C is a collection of variables of different types under a single name.

Quick Quiz: Write a program to store the details of 3 employees from user defined data. Use the structure declared above.

Why use structures?
We can create the data types in the employee structure separately but when the number of properties in a structure increases, it becomes difficult for us to create data variables without structures. In a nut shell:

(a) Structures keep the data organized.
(b) Structures make data mangement easy for the programmer.

Array of Structures
Just like an array of integers, an array of floats and an array of characters, we can create an array of structures.

```
struct employee facebook[100]; ⇒ An array of structures
```

We can access the data using:

```
facebook[0].code = 100;
facebook[1].code = 101;
... & so on
```

Initializing Structures
Structures can also be initialized as follows:

```
struct employee harry = {100, 71.22, "Harry"};
struct employee shubh = {0}; ⇒ All elements set to 0
```

URBAN EDGE

Structures in memory

Structures are stored in contiguous memory locations
For the structure e1 of type struct employee, memory layout looks like this:

100	71.22	"Harry"
Address → 78810	78814	78818

In an array of structures, these employee instances are stored adjacent to each other.

Pointer to structures

A pointer to structure can be created as follows:

```
struct employee *ptr;
ptr = &e1;
```

Now we can print structure elements using:

```
printf("%d", *(ptr).code);
```

(*ptr).code

Arrow Operator

Instead of writing *(ptr).code, we can use arrow operator to access structure properties as follows

*(ptr).code Or ptr -> code

(*ptr).code

Here -> is known as the arrow operator.

URBAN EDGE

Passing Structure to a function

A structure can be passed to a function just like any other data type.

```
Void show (struct employee e);
```
⇒ function prototype

Quick Quiz : Complete this show function to display the content of employee.

Typedef keyword

We can use the typedef keyword to create an alias name for data types in C. typedef is more commonly used with structures.

```
struct complex {
    float real;
    float img;
};
```
⇒ struct complex C_1, C_2;
for defining Complex numbers

```
typedef struct complex {
    float real;
    float img;
} complexNo;
```
⇒ complexNo C_1, C_2;
for defining Complex numbers