Chapter 9 - Structures

Array and strings → Similar data (int, float, char).
Structures can hold → Dissimilar data
A C structure can be created a follows

```
Struct Employee
{
    int code; // This code declares a new user
    float salary;          defined data type!
    char name[10];
};  // Semicolon is important
```

We can use this user defined data type as follows:

```
Struct Employee e1; // Creating a structure variable
Strcpy(e1.name, "Harry");
e1.code = 100;
e1.salary = 71.22;
```

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

RANKA
DATE / /
PAGE

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 Structure in a nutshell

(a) Structures Keep the data Organized
(b) Structures make data managment Easy for the programmers

Array of Structures

Just like an array of floats and an array of characters

We can create an array of structures

```
Struct Employee facebook [100];

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

Initializing Structures

Structures can also be initialized as follows:

```
Struct Employee harry = {100, 71.92, "harry"};
Struct Employee shubh {0};
```

Structures in Memory

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

100	71.92	"Harry"
78810	78814	78818

Address ⟶

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

Pointer to Structures

A pointer to Structures can be created as follows:

```
Struct employee *ptr;
ptr = &e1;
// now we can print Structure Elements
   using:
printf("%d", (*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
```

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

RANKA
DATE / /
PAGE

7)

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;
};
typedef Struct Complex
{
    float real;
    float img;
} Complex;
```

Example Usage

Using the typedef alias, you can declare complex number variable more succinctly

Complex C1, C2;

Chapter 9 - Practice Set

Q1) Create a two-dimensional structures in C.

```
#include <stdio.h>
    struct vector{
        int i;
        int j;
    };
    int main(){
        struct vector v={1,2};
        printf("The value of vector is %di+%dj",
                                    v.i, v.j);
        return 0;
    }
```

Q2) write a function 'SumVector' which returns the sum of two vectors passed to it. the vector must be two-dimensional.

```
#include <stdio.h>
    typedef struct vector{
        int i;
        int j;
    } V;
V SumVector(V v1, V v2){
    V v3 = {v1.i + v2.i, v1.j + v2.j};
    return v3;
}
```

```
int main(){
    V v1 = {1,2};
    V v2 = {5,6};
    V v3 = SumVector(v1, v2);
    printf("The value of vector V3 is %di + %dj", v3.i,
                                                  v3.j);
    return 0;
}
```

Q3 Twenty integers are to be stored in memory. what will you prefer - Array or Structure?

```
#include <stdio.h>
typedef struct Emp{
    int Salary;
    float Score;
}Employee;

int main(){
    Employee e1;
    Employee *ptr = &e1;
    (*ptr).Salary = 56;
    (*ptr).Score = 45.3;
// ptr -> salary = 56;
// ptr -> Score = 45.3;

    printf("The value of Salary is %d and the value
            of Score is %.2f\n", ptr->Salary
                                 ptr->Score);
    return 0;
}
```

Q4) write a program to illustrate the use of arrow operator in C.

```
#include <stdio.h>
typedef struct c{
    int real;
    int imaginary;
} Complex;

int main(){
    Complex c = {1,2};
printf("The value of Complex number is
        %d + %di", c.real, c.imaginary);
return 0;
}
```

Q5) Write a program with a structure representing a Complex number

```
#include <stdio.h>
typedef struct c
{
    int real;
    int imaginary;
} Complex;
void display(Complex c)
{
printf("The value of Complex number is %d+%di
                c.real, c.imaginary);
}
int main(){
```

```
Complex carr[5];
for (int i=0; i<5; i++)
{
    printf("Enter real part \n");
    scanf("%d", &carr[i].real);
    printf("Enter imaginary part \n");
    scanf("%d", &carr[i].imaginary);
    display(carr[i]);
}
return 0;
}
```

Q6) Create a structure representing a bank account of a customer what fields did you use and why?

```
#include <stdio.h>
struct bankacc{
    int accNo;
    char name[34];
    char ifsc[12];
    float balance;
};
int main(){
    return 0;
}
```

Q7) Write a structure capable of storing dates write a function to compare those dates

```
#include <stdio.h>
typedef struct Date
{
    int mm;
    int dd;
    int yyyy;
} DT;

int compare (DT d1, DT d2)
{
    if ((d1.yyyy == d2.yyyy) && (d1.mm == d2.mm) && (d1.dd == d2.dd))
    {
        return 0;
    }
    if (d1.yyyy > d2.yyyy)
    {
        return 1;
    }
    else if (d1.yyyy < d2.yyyy)
    {
        return -1;
    }
    else if (d1.mm > d2.mm)
    {
        return 1;
    }
    else if (d1.mm < d2.mm)
    {
        return -1;
    }
```

```
else if (d1.dd > d2.dd)
  {
    return 1;
  }

else if (d1.dd < d2.dd)
  {
    return -1;
  }
}
int main()
{
  DT d1 = {12, 4, 2154};
  DT d2 = {12, 8, 2154};
  printf("%d", compare(d1, d2));
  return 0;
}
```