Page No.
Date 02 12 24

Unit – 5

Structure – It is a collection of variables of different data types under a single name.

The variables are called members of the structure.

Also called User-defined data type.

Syntax:
```
struct Structure_name
{
    data_type member_variable 1 ;
    data_type member_variable N ;
};
```

Once structure ~~is~~ name is declared as new data type then,
struct structure_name structure_variable.

Here we use dynamic memory allocation.

Ex –
```
struct student {                    Keyword
    char name [20];
    int roll_no ;
    float marks ;
    char gender ;
    long int phone_no ;
};
```
members

we have to terminate it.

struct Student (St) ;

St → declaration
memory allocation

Bubble short - Compare with adjacent element.

NOTE: The member of a structure do not occupy memory until they are associated with a structure variable

Defining a structure

↳ Each variable of structure has its own Copy of member variable.

↳ The member variables are accessed using the dot (.) operator or member operator. Eg. st1.Gender
name of the variable & the member

Syntax:

struct structure_name structure_variable = {value 1, value 2, ... value N};

Partial Initialization

We can initialize the 1st few members & leave the remaining blank.
The uninitialized members are assigned default values as follows.
Zero - for integers & floating point numbers.
'\0' - for character, string.
Null Char

Copying and Comparing Structure variables.

Two variables of the same structure type can be copied in the same way as ordinary variables.

strcmp - Used for comparison
pre-define func.

lexico graphical order is also known as dictionary order

App:

Why 'structure' are best instead of 'string'?

We use structure in C when we need to represent an entity with multiple attributes, especially of diff. data types. strings are limited to storing & manipulating text, whereas structures provide a flexible way to organize & manage complex, related data in a program.
We can also create our own data type.
Structures in C are a powerful tool to organize & handle data efficiently. They form the basis for creating custom data types, enhancing the flexibility & functionality of your programs.

Q: Create a structure named date that has day, month, & year as its members. Include this structure as a member in another structure named employee which has name, id & Salary as other members. Use this structure to read & display Employee's name id, date of birthday & Salary.

```
#include <stdio.h>
struct Date {
    int day;
    int month;
    int year;
};
struct Employee {
    char name[50];
    int id;
```

```
    float salary;
    struct Date dob;
};
int main()
{
    struct Employee emp;
    printf("Enter Employee name:");
    scanf("%s", emp.name);
    printf("Enter Employee ID:");
    scanf("%d", &emp.id);
    printf("Enter Employee salary:");
    scanf("%f", &emp.salary);
    printf("Enter Employee date of birth (day, month, year):");
    scanf("%d %d %d", &emp.dob.day, &emp.dob.month,
     &emp.dob.year);

    printf("Displaying details:\n");
    printf("Name: %s", emp.name);
    printf("ID: %d\n", emp.id);
    printf("Salary: %.2f\n", emp.salary);
    printf("Date of Birth: %02d-%02d-%04d\n",
        emp.dob.day, emp.dob.month, emp.dob.year);

return 0;

}
```

structure within another structure (Nested structure) provides the reusability concept.

i)
```
#include <stdio.h>
void fun (int *ptr)
{
  *ptr = 30;
}
int main()
{
  int y = 20;
  fun (&y);
  printf ("%d", y);
  return 0;
}
```
o/p: 30
↳ Call by reference

ii)
```
void fun (int x)
{
  x = 30;
}
int main()
{
  int y = 20;
  fun (y);
  printf ("%d", y);
  return 0;
}
```
o/p: 20
↳ Call by value

iii)
```
int main()
{
  float arr5 = {12.5, 10.0, 13.5, 90.0, 0.5};
  float *ptr1 = &arr[0];
  float *ptr2 = ptr1 + 3;
  printf ("%f", *ptr2);
  printf ("%d", ptr2 - ptr1);
  return 0;
}
```
o/p:- 90.0
3

2012 − 2000/4 = 3
↳ returns the no. of elements in b/w these 2 elements, addresses

iv)
```
int main() {
  int *ptr;
  int x;
  ptr = &x;
  *ptr = 0;
  printf ("x = %d", x);
  printf ("*ptr = %d", *ptr);
```
x: 0, ptr: &x
*(&x) = x
o/p: 0 0

```
  *ptr += 5;
  printf ("x = %d", x);
  printf ("*ptr = %d", *ptr);
  (*ptr)++;
  printf ("x = %d", x);
  printf ("*ptr = %d", *ptr);
```
o/p: 5 5 6 6