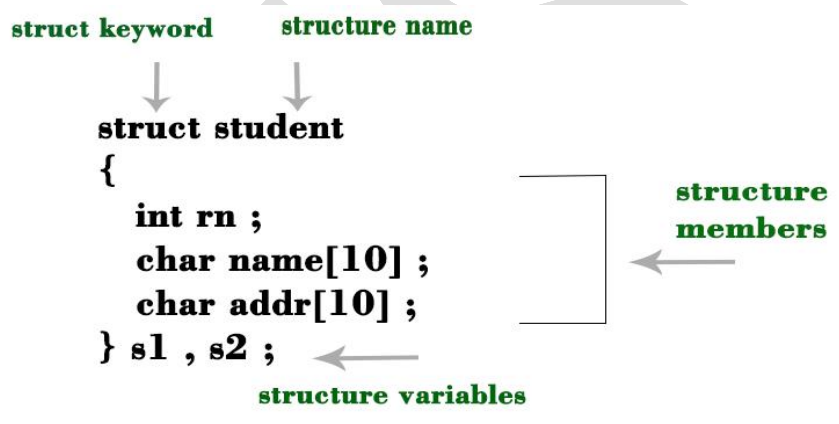
C Programming Short Notes

Part 7

Structure

- A structure is a **user-defined data type** that allows you to **group together related data items under a single name.**
- It is used to represent a collection of variables (**of different or same data types**) that are treated as a **single unit.**
- Each variable in the structure is called a **member or field.**
- **This grouping enables you to create more complex data structures to represent real-world entities.**



```
struct student
{
   int rn ;
   char name[10] ;
   char addr[10] ;
} s1 , s2 ;
```

Defining a Structure:

- Define a structure using the **`struct`** keyword followed by a structure tag (optional) and a list of member variables enclosed in braces **`{}`.**
- Each member variable within the structure is called a **member or field.**
- Inside the structure, you list **the member variables, each with its own data type.**

Example :

```
struct Person {
   char name[50];
   int age;
   float salary;
};
```

Creating Structure Variables:

- Once a structure is defined, you can **create variables of that structure type.**
- **Each variable will have its own set of member variables, but they all share the same structure definition.**

Example:

Eg1:

```
struct Person employee1;
```

Eg2 :

```
struct Person person1, person2;
```

Accessing Structure Members:

- You can access structure members using the dot **`.` operator.**

Example:

```
strcpy(employee1.name, "John");

employee1.age = 30;

employee1.salary = 50000.0;
```

Initializing Structures:

- You can **initialize a structure at the time of declaration using the `{}` syntax.**

Example:

```
struct Person p2 = {"Alice", 25, 60000.0};
```

Printing structure variable values:

- To print the contents of a structure in C, you typically access **each member of the structure and print its value individually.**

Example:

```
#include <stdio.h>
struct Person {                                    // Define a structure
    char name[50];
    int age;
};
int main() {
    struct Person person1;                         // Declare a structure variable
    strcpy(person1.name, "John");                  // Assign values to the structure members
    person1.age = 30;
    printf("Name: %s\n", person1.name);            // Print the structure content
    printf("Age: %d\n", person1.age);              // Print the structure content
    return 0;
}
```

- **To use scanf() with a structure in C, you can input values for each member of the structure using format specifiers corresponding to the data types of the structure members.**

Example:

```
#include <stdio.h>
struct Person {
    char name[50];
    int age;
};
int main() {
    struct Person person1;
    printf("Enter name: ");
    scanf("%s", person1.name);                     // Input value for the structure member
    printf("Enter age: ");
    scanf("%d", &person1.age);                     // Input value for the structure member
    printf("Name: %s\n", person1.name);            // Print the entered value
    printf("Age: %d\n", person1.age);              // Print the entered value
    return 0;
}
```

- However, using scanf() directly for inputting strings (%s) is **not safe,** as it's prone to **buffer overflow** if the user inputs more characters than the array size allocated for name. To make it safer, you can use **fgets()** instead.

Example:

```
printf("Enter name: ");
fgets(person1.name, sizeof(person1.name), stdin);
```

Nested Structures:

- **Structures can contain other structures as members,** allowing for hierarchical organization of data.

Example:

```
#include <stdio.h>
struct Address {                              // Define a structure for an address
   char street[50];
   char city[50];
   int pinCode;
};
struct Employee {                             // Define a structure for an employee
   char name[50];
   int age;
   struct Address address;
};
int main() {
   struct Employee emp;                       // Declare a variable of type Employee
   printf("Enter employee name: ");           // Input employee information
   scanf("%s", emp.name);
   printf("Enter employee age: ");
   scanf("%d", &emp.age);
   printf("Enter employee street: ");
   scanf("%s", emp.address.street);
   printf("Enter employee city: ");
   scanf("%s", emp.address.city);
   printf("Enter employee pin code: ");
   scanf("%d", &emp.address.pinCode);
   printf("\nEmployee Information:\n");       // Display employee information
   printf("Name: %s\n", emp.name);
   printf("Age: %d\n", emp.age);
   printf("Address:   %s,   %s,   %d\n",   emp.address.street,   emp.address.city,
emp.address.pinCode);
   return 0;
}
```

Structure using pointer:

Passing Structures to Functions:

- You can **pass structures to functions either by value or by reference.**
- **Passing by value** creates a **copy of the structure**, while passing by **reference** allows **modifications to the original structure.**

Passing by Value:

- When you pass a structure by value, a copy of the entire structure is made and passed to the function. **Any modifications made to the structure inside the function do not affect the original structure.**

Example

```
#include <stdio.h>
struct Point {                                    // Define a structure
    int x;
    int y;
};
void modifyPoint(struct Point p) {           // Function to modify a Point structure
    p.x = 100;
    p.y = 200;
}
int main() {
    struct Point point = {10, 20};
    modifyPoint(point);                          // Call the function with a Point structure
    printf("Original Point: (%d, %d)\n", point.x, point.y);  // Print the original structure
    return 0;
}
```

Output :

```
Original Point: (10, 20)
```

Passing by Reference (Using Pointers):

- When you pass a structure by reference using pointers, you pass the address of the structure**, allowing the function to directly modify the original structure.**

Example:

```
#include <stdio.h>

struct Point {                    // Define a structure
    int x;
    int y;
};

// Function to modify a Point structure (passed by reference)
void modifyPoint(struct Point *p) {
    p->x = 100;
    p->y = 200;
}
int main() {
    struct Point point = {10, 20};
    modifyPoint(&point);       // Call the function with a reference to the Point structure
  printf("Modified Point: (%d, %d)\n", point.x, point.y); // Print the modified structure

    return 0;
}
```

Output

```
Modified Point: (100, 200)
```