

Module 5

STRUCTURES

INTRODUCTION:

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. The, **struct** keyword is used to define the structure.

Syntax:

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memberN;
};
```

Let's see the example to define a structure for an entity employee in c.

```
struct employee
{
    int id;
    char name[20];
    float salary;
}emp;
```

The following image shows the memory allocation of the structure employee that is defined in the above example.

$\text{Sizeof}(\text{emp}) = 2 + 20 + 4 = 26 \text{ bytes}$

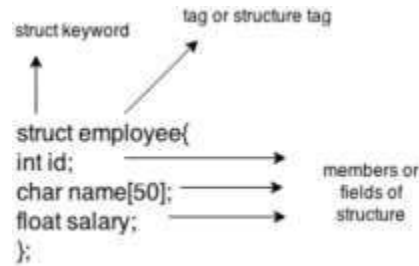
Where,

$\text{Sizeof}(\text{int}) = 2 \text{ bytes}$

$\text{Sizeof}(\text{char}) = 1 \text{ byte (char name[50] = 1 * 50)}$

$\text{Sizeof}(\text{float}) = 4 \text{ bytes}$

Here, struct is the keyword; employee is the name of the structure; id, name, and salary are the members or fields of the structure. Let's understand it by the diagram given below:



Declaring structure variable

We can declare a variable for the structure so that we can access the member of the structure easily.

There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

Method 1:

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

```
struct employee  
{ int id;  
  char name[50];  
  float salary;  
};
```

Now write given code inside the main() function.

```
struct employee e1, e2;
```

The variables e1 and e2 can be used to access the values stored in the structure.

Method 2:

Let's see another way to declare variable at the time of defining the structure.

```
struct employee  
{ int id;  
  char name[50];  
  float salary;  
}e1,e2;
```

Accessing members of the structure

There are two ways to access structure members:

1. By . (Member or dot operator)
2. By -> (structure pointer operator)

C Program to show the working of structure.

```
#include<stdio.h>
#include <string.h>
struct employee
{
    int id;
    char name[50];
}e1;
void main( )
{
    e1.id=101;
    strcpy(e1.name, "Sonoo Jaiswal");
    printf( "employee 1 id : %d\n", e1.id);
    printf( "employee 1 name : %s\n", e1.name);
}
```

C program to store the books details using structure.

```
#include <stdio.h>
#include <string.h>
struct Books
{
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};
void main( )
{
    struct Books Book1;
    struct Books Book2;
    strcpy( Book1.title, "C Programming");
    strcpy( Book1.author, "Nuha Ali");
    strcpy( Book1.subject, "C Programming Tutorial");
    Book1.book_id = 6495407;
```

```
strcpy( Book2.title, "Telecom Billing");
strcpy( Book2.author, "Zara Ali");
strcpy( Book2.subject, "Telecom Billing Tutorial");
Book2.book_id = 6495700;
printf( "Book 1 title : %s\n", Book1.title);
printf( "Book 1 author : %s\n", Book1.author);
printf( "Book 1 subject : %s\n", Book1.subject);
printf( "Book 1 book_id : %d\n", Book1.book_id);
printf( "Book 2 title : %s\n", Book2.title);
printf( "Book 2 author : %s\n", Book2.author);
printf( "Book 2 subject : %s\n", Book2.subject);
printf( "Book 2 book_id : %d\n", Book2.book_id);
}
```

Structure And Arrays

An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

Example Program:

```
#include<stdio.h>
struct student
{
    int rollno;
    char name[10];
};
void main()
{
    int i;
    struct student st[5];
    printf("Enter Records of 5 students");
    for(i=0;i<5;i++)
    {
        printf("\nEnter Rollno:");
        scanf("%d",&st[i].rollno);
        printf("\nEnter Name:");
        scanf("%s",&st[i].name);
    }
    printf("\nStudent Information List:");
    for(i=0;i<5;i++)
    {
        printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
    }
}
```

Structures And Functions:

C allows programmers to pass a single or entire structure information to or from a function. A structure information can be passed as a function argument. The structure variable may be passed as a value or reference. The function will return the value by using the return statement.

Example: C program - Passing Structure Members to Functions

```
#include <stdio.h>
int add(int, int) ;
struct addition
{
    int a, b;
    int c;
}sum;
void main()
{
    printf("Enter the value of a : ");
    scanf("%d",&sum.a);
    printf("\nEnter the value of b : ");
    scanf("%d",&sum.b);
    sum.c = add(sum.a, sum.b);
    printf("\nThe sum of two value are : ");
    printf("%d ", sum.c);
}
int add(int x, int y)
{
    int sum1;
    sum1 = x + y;
    return(sum1);
}
```

Example: C program - Passing The Entire Structure To Functions

```
#include <stdio.h>
typedef struct
{
    int a, b;
    int c;
}sum;

void add(sum) ;
void main()
{
    sum s1;
    printf("Enter the value of a : ");
    scanf("%d",&s1.a);
    printf("\nEnter the value of b : ");
    scanf("%d",&s1.b);
    add(s1);
}
```

```
    }  
    void add(sum s)  
    {  
        int sum1;  
        sum1 = s.a + s.b;  
        printf("\nThe sum of two values are :%d ", sum1);  
    }
```

UNION

INTRODUCTION

A union is a user-defined type similar to structs in C except for one key difference. Structures allocate enough space to store all their members, whereas unions can only hold one member value at a time.

Declaration of Union:

Syntax:

```
union car  
{  
    char name[50];  
    int price;  
};
```

Creating Union Variables:

When a union is defined, it creates a user-defined data type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

Method 1:

```
union car  
{  
    char name[50];  
    int price;  
};  
void main()  
{  
    union car car1, car2, *car3;  
}
```

Method 2:

```
union car  
{  
    char name[50];  
    int price;  
} car1, car2, *car3;
```

In both cases, union variables car1, car2, and a union pointer car3 of union car type are created.

Access members of a union

We use the . (dot) operator to access members of a union. And to access pointer variables, we use the -> operator.

In the above example,

1. To access price for car1, car1.price is used.
2. To access price using car3, either (*car3). price or car3->price can be used.

Difference between Structure and Union

Struct	Union
The struct keyword is used to define a structure.	The union keyword is used to define union.
When the variables are declared in a structure, the compiler allocates memory to each variable's member. The size of a structure is equal or greater to the sum of the sizes of each data member.	When the variable is declared in the union, the compiler allocates memory to the largest size variable member. The size of a union is equal to the size of its largest data member size.
Each variable member occupied a unique memory space.	Variable's members share the memory space of the largest size variable.
Changing the value of a member will not affect other variables members.	Changing the value of one member will also affect other variables members.
It is used to store different data type values.	It is used for storing one at a time from different data type values.
The structure allows initializing multiple variable members at once.	Union allows initializing only one variable member at once.
It allows accessing and retrieving any data member at a time.	It allows accessing and retrieving any one data member at a time.

POINTERS

INTRODUCTION

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer.

Declaration of pointer variable:

Syntax:

```
datatype *pointer_variable;
```

Example:

```
int *p;
```

Initialization of Pointer Variable:

Syntax:

```
int a=5;
```

```
int *p;
```

```
p = &a;
```

Accessing Pointer Variable:

To access pointer variable, we use * (asterisk) symbol.

Syntax:

```
int a=5;
```

```
int *p;
```

```
p = &a;
```

```
printf ("Accessing pointer variable = %d", *p); //Output will be 5.
```

Example Program to show working of pointer variable:

```
#include<stdio.h>

Void main()
{
    int a, *p;
    printf("Enter the value of a\n");
    scanf("%d", &a);
    p = &a;
```



```
printf("Value of a is %d\n",a);  
printf("Value of p is %d\n", p);  
printf("Accessing pointer p %d\n", *p);  
}
```

NULL Pointer

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer.

Example:

```
#include <stdio.h>  
void main ()  
{  
    int *ptr = NULL;  
    printf("The value of ptr is : %x\n", ptr );  
}
```

When the above code is compiled and executed, it produces the following result –

The value of ptr is 0

Double Pointer

It is a variable which holds the address of another pointer variable.

Declaration of double pointer

Syntax:

Datatype **variable;

Example:

```
int **dp;
```

Initialization of double pointer

Syntax:

```
int a=5;  
int *p, **dp;  
p = &a;  
dp = &p;
```

Accessing double pointer

To access pointer variable, we use * *(asterisk) symbol.

Syntax:

```
int a=5;
int *p, **dp;
p = &a;
dp = &p;
printf ("Accessing double pointer variable = %d", **dp);    //Output will be 5.
```

Example Program to show working of double pointer variable:

```
#include<stdio.h>
void main()
{
    int a, *p, *dp;
    printf("Enter the value of a\n");
    scanf("%d", &a);
    p = &a;
    dp = &p;
    printf("Value of a is %d\n",a);
    printf("Value of p is %d\n", p);
    printf("Value of dp is %d\n", dp);
    printf("Accessing pointer p %d\n", *p);
    printf("Accessing double pointer variable dp is %d\n", **dp);
}
```

Pointers And Arrays

Use a pointer to an array, and then use that pointer to access the array elements. For example,

```
#include<stdio.h>
void main()
{
    int a[3] = { 1, 2, 3 };
    int *p = a;
    for (int i = 0; i < 3; i++)
    {
        printf("%d", *p);
    }
}
```

```
        p++;  
    }  
}
```

Replacing the **printf("%d", *p);** statement of above example, with below mentioned statements. Lets see what will be the result.

printf("%d", a[i]); → **prints the array, by incrementing index**

printf("%d", i[a]); → **this will also print elements of array**

printf("%d", a+i); → **This will print address of all the array elements**

printf("%d", *(a+i)); → **Will print value of array element.**

printf("%d", *a); → **will print value of a[0] only**

a++; → **Compile time error, we cannot change base address of the array.**

Note: *(a+i) is same as i[a].

Write a C program to find the sum, mean and standard deviation of an array elements using pointer concept.

```
#include <stdio.h>  
#include <math.h>  
void main()  
{  
    int i, n;  
    float x[10], *p, mean, variance, sd, sum = 0, sum1 = 0;  
    printf("Enter the value of N \n");  
    scanf("%d", &n);  
    printf("Enter %d array elements \n", n);  
    for (i = 0; i < n; i++)  
    {  
        scanf("%f", &x[i]);  
    }  
    p=x;  
    for (i = 0; i < n; i++)
```

```
        {
            sum = sum + *p;
            p++;
        }
    mean = sum / n;
    p=x;
    for (i = 0; i < n; i++)
    {
        sum1 = sum1 + pow((*p - mean), 2);
        p++;
    }
    variance = sum1 / n;
    sd= sqrt(variance);
    printf("SUM = %.2f\n", sum);
    printf("MEAN = %.2f\n", mean);
    printf("VARIANCE= %.2f\n", variance);
    printf("STANDARD DEVIATION = %.2f\n", sd);
}
```

Pointers and Functions

Pointer as a function parameter is used to hold addresses of arguments passed during function call. This is also known as call by reference. When a function is called by reference any change made to the reference variable will affect the original variable.

Example: Swapping two numbers using Pointer and Functions concept.

```
#include <stdio.h>
void swap(int *a, int *b);
void main()
{
    int m = 10, n = 20;
    printf("m = %d\n", m);
    printf("n = %d\n\n", n);
    swap(&m, &n);
    printf("After Swapping:\n\n");
    printf("m = %d\n", m);
    printf("n = %d", n);
}

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

Pointers And Structure:

Pointer to structure holds the address of the entire structure. It is used to create complex data structures such as linked lists, trees, graphs and so on. The members of the structure can be accessed using a special operator called as an arrow operator (->).

The following program shows the usage of pointers to structures –

```
#include<stdio.h>
struct student
{
    int sno;
    char sname[30];
    float marks;
};
void main ( )
{
    struct student s;
    struct student *st;
    printf("enter sno, sname, marks:");
    scanf ("%d%s%f", & s.sno, s.sname, &s. marks);
    st = &s;
    printf ("details of the student are");
    printf ("Number = %d\n", st ->sno);
    printf ("name = %s\n", st->sname);
    printf ("marks =%f\n", st ->marks);
}
```

PRE-PROCESSOR DIRECTIVES

INTRODUCTION

All pre-processor commands begin with a hash symbol (#)

#define - Substitutes a pre-processor macro.

#include - Inserts a particular header from another file.

#if - Tests if a compile time condition is true.

#ifdef - Returns true if this macro is defined.

#endif - Ends pre-processor conditional.

#else - The alternative for #if.

Examples:

```
#define MAX_ARRAY_LENGTH 20
```

This directive tells the CPP to replace instances of MAX_ARRAY_LENGTH with 20. Use #define for constants to increase readability.

```
#include <stdio.h>

#include "myheader.h"
```

These directives tell the CPP to get stdio.h from System Libraries and add the text to the current source file. The next line tells CPP to get myheader.h from the local directory and add the content to the current source file.

```
#ifndef MESSAGE

#define MESSAGE "You wish!"

#endif
```

It tells the CPP to define MESSAGE only if MESSAGE isn't already defined.

DYNAMIC MEMORY ALLOCATION

INTRODUCTION

The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime.

Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

1. malloc()
2. calloc()
3. realloc()
4. free()

Difference between static memory and dynamic memory allocation

Static Memory	Dynamic Memory
memory is allocated at compile time.	memory is allocated at run time.
memory can't be increased while executing program.	memory can be increased while executing program.
used in array.	used in linked list.

malloc() function in C

- The malloc() function allocates single block of requested memory.
- It doesn't initialize memory at execution time, so it has garbage value initially.
- It returns NULL if memory is not sufficient.

- The syntax of malloc() function is given below:

`ptr=(cast-type*)malloc(byte-size)`

calloc() function in C

- The calloc() function allocates multiple block of requested memory.
- It initially initialize all bytes to zero.
- It returns NULL if memory is not sufficient.
- The syntax of calloc() function is given below:

`ptr=(cast-type*)calloc(number, byte-size)`

realloc() function in C

If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

Let's see the syntax of realloc() function:

`ptr=realloc(ptr, new-size)`

free() function in C

The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.

Let's see the syntax of free() function.

`free(ptr)`

C FILE HANDLING

INTRODUCTION

A File is a collection of data stored in the secondary memory. So far data was entered into the programs through the keyboard. So, Files are used for storing information that can be processed by the programs.

Files are not only used for storing the data, programs are also stored in files. In order to use files, we have to learn file input and output operations. That is, how data is read and how to write into a file.

Types of Files

When dealing with files, there are two types of files you should know about:

1. Text files
2. Binary files

Text files

Text files are the normal .txt files. You can easily create text files using any simple text editors such as Notepad. When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents. They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

Binary files

Binary files are mostly the .bin files in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold a higher amount of data, are not readable easily, and provides better security than text files.

File Operations

In C, you can perform four major operations on files, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Closing a file

Opening a file - for creation and edit

Opening a file is performed using the fopen() function defined in the stdio.h header file.

The syntax for opening a file in standard I/O is:

```
ptr = fopen("filename", "mode");
```

For example,

```
fopen("E:\\cprogram\\newprogram.txt", "w");
```

```
fopen("E:\\cprogram\\oldprogram.bin", "rb");
```

- Let's suppose the file `newprogram.txt` doesn't exist in the location `E:\\cprogram`. The first function creates a new file named `newprogram.txt` and opens it for writing as per the mode `'w'`. The writing mode allows you to create and edit (overwrite) the contents of the file.
- Now let's suppose the second binary file `oldprogram.bin` exists in the location `E:\\cprogram`. The second function opens the existing file for reading in binary mode `'rb'`. The reading mode only allows you to read the file, you cannot write into the file.

Closing a File

The file (both text and binary) should be closed after reading/writing. Closing a file is performed using the `fclose()` function.

Syntax:

```
fclose(fptr);
```

Here, `fptr` is a file pointer associated with the file to be closed.