**UNIT IV STRUCTURES AND UNION 9**

Structure - Nested structures – Pointer and Structures – Array of structures – Self referential structures – Dynamic memory allocation - Singly linked list – typedef – Union - Storage classes and Visibility.

---

## ARRAY OF STRUCTURES

There can be array of structures in C programming to store many information of different data types. The array of structures is also known as collection of structures. Let's see an example of a structure with an array that stores information of 5 students and prints it.

```
#include<stdio.h>
#include <string.h>
struct student
{
	int rollno;
	char name[10];
};
int 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);
	}
	return 0;
}
```

**Output:**

```
Enter Records of 5 students
Enter Rollno:1
Enter Name:Sonoo
Enter Rollno:2
Enter Name:Ratan
Enter Rollno:3
Enter Name:Vimal
Enter Rollno:4
Enter Name:James
Enter Rollno:5
Enter Name:Sarfraz
Student Information List:
Rollno:1, Name:Sonoo
Rollno:2, Name:Ratan
Rollno:3, Name:Vimal
Rollno:4, Name:James
Rollno:5, Name:Sarfraz
```

---

**DYNAMIC MEMORY ALLOCATION**

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 the stdlib.h header file.

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

Before learning the above functions, let's understand the difference between static memory allocation and dynamic memory allocation.

| Static memory allocation | Dynamic memory allocation |
|---|---|
| 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. |

Now let's have a quick look at the methods used for dynamic memory allocation.

- malloc() - Allocates a single block of requested memory.
- calloc() - Allocates multiple blocks of requested memory.
- realloc() - Reallocates the memory occupied by malloc() or calloc() functions.
- free() - Frees the dynamically allocated memory.

**malloc()**

The malloc() function allocates a 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)**

**Example:**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int n,i,*ptr,sum=0;
	printf("Enter number of elements: ");
	scanf("%d",&n);
	ptr=(int*)malloc(n*sizeof(int)); //memory allocated using malloc
	if(ptr==NULL)
	{
		printf("Sorry! unable to allocate memory");
		exit(0);
	}
	printf("Enter elements of array: ");
	for(i=0;i<n;++i)
	{
		scanf("%d",ptr+i);
		sum+=*(ptr+i);
	}
	printf("Sum=%d",sum);
	free(ptr);
	return 0;
}
```

**Output:**

```
Enter elements of array: 3
Enter elements of array: 10
10
10
Sum=30
```

**calloc()**

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)**

**Example:**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int n,i,*ptr,sum=0;
	printf("Enter number of elements: ");
	scanf("%d",&n);
	ptr=(int*)calloc(n,sizeof(int)); //memory allocated using calloc
	if(ptr==NULL)
	{
		printf("Sorry! unable to allocate memory");
		exit(0);
	}
	printf("Enter elements of array: ");
	for(i=0;i<n;++i)
	{
		scanf("%d",ptr+i);
		sum+=*(ptr+i);
	}
	printf("Sum=%d",sum);
	free(ptr);
	return 0;
}
```

**Output:**

```
Enter elements of array: 3
Enter elements of array: 10
10
10
Sum=30
```

**realloc()**

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()**

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

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

**free(ptr)**

---