CS8251- PROGRAMMING IN C

UNIT IV

1. Define Structure. How to create a structure? How do you access the datamembers inside the structure.

- A structure is a user-defined data type that can store related information (collection of different datatypes together.
- structure is a collection of variables under a single name. The variables within a structure are of different datatypes and each has a name.

Creating / Declaring a structure:

- Structure is created using a keyword called 'struct'.

Syntax:

```
struct structname
{
    datatype var_name;
    datatype var_name1;
} strvar;
```

Example:

```
struct student
{
    int roll_no;
    char name[20];
} s;
```

Initialization of structure:

- Initializing the data members of the structure is as follows

```
struct student
{
  int rollno;
  char name[20];
} s = {4106001, "James"};
```

Accessing the datamembers of a structure:

- Each member of a structure is like a normal variable.
- Data member inside the structure is accessed using dot operator.

Syntax: strvar.datamember.

Example Program to get date, month and year and display the same using Structure.

```
#include <stdio.h>
#include <conio.h>
struct date
{
   int day;
   int month;
   int year;
}d;
void main()
{
   printf (" Enter the day: ");
   scanf ("%d", &d.day);
   preatf (" Enter the month :");
    scanf ("%d", &d.month);
   printf (" Enter the year ");
   scanf ("%d", &d.year);
   printf (" The date is %d\t %d\t %d\t", d.day,
                                    d.month, d.year);
}
```

Output:

```
Enter the day:
19
Enter the month
3
Enter the year
2020
```

The date is 19 3 2020.

2. Explain in detail about Nested Structure.

- Nested structure is structure within structure.
- One structure can be declared inside other structure as we declare structure members inside a structure.
- Two types of nested structure are.
  a. By Separate structure.
  b. By Embedded structure.

Separate Structure:

Separate structure can be created as follows.

```
struct Date
{
   int dd;
   int mm;
   int yy;
};
struct Employee
{
   int id;
   char name[20];
   struct Date doj;
} emp1
```

→ structure variable for struct Date.

Embedded Structure:

Embedded structure can be created as follows.

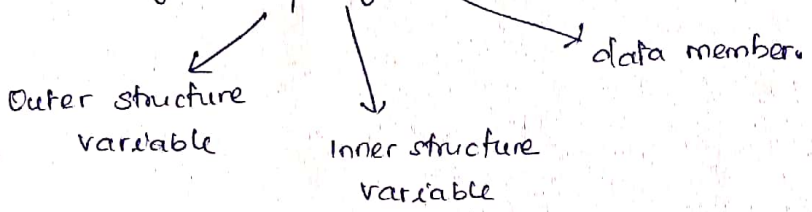
```
struct Employee
{
   int id;
   char name[20];
   struct Date
    {
        int dd;
        int mm;
        int yy;
    } doj;
} emp;
```

Accessing datamembers using Nested structure:

Syntax:

outerstructurevar . innerstructurvar . datamembers;

Eg: emp.doj.dd



Example Program:

```
#include <stdio.h>
#include <string.h>
 struct Employee
 {
    int id;
    char name [20];
    struct Date
    {
       int dd;
       int mm;
       int yy;
    } doj;
 } e1;
int main()
{
    e1.id = 101;
    strcpy (e1.name, "James");
    e1.doj.dd = 10;
    e1.doj.mm = 11;
    e1.doj.yy = 2019;
    printf ("Employee id : %d \n", e1.id);
    printf ("Employee name : %s \n", e1.name);
    printf ("Employee DOJ (dd/mm/yyyy) : %d %d %d \n", e1.doj.dd,
                                        e1.doj.mm, e1.doj.yy);
    return 0;
}
```

3. Explain the concept of array of structures with a suitable program.

- To collect 'n' number of details of the same structure, array of structure is used.

Example:

Define a structure called book with book name, author name, price and pages. Write a C program to read the details of 200 books. Display the information of book whose price amount is greater than Rs. 500

```
#include <stdio.h>
#include <conio.h>
struct book
{
   char name[10], aname[15];
   int price;
   int pages;
}b[200];
void main() {
printf(" Enter Book Details are :");
int i;
for(i=1; i<=200; i++)
{
  scanf("%s %s %d %d", &b[i].name, &b[i].aname,
                        &b[i].price, &b[i].pages);

 }
 printf(" Information of Books having price greater than Rs. 500");
 for(i=1; i<=200; i++)
 {
   if (b[i].price >= 500)
   {
     printf("%s %s %d %d", b[i].name, b[i].aname,
                          b[i].price, b[i].pages);

   }
}
```

4. Explain Dynamic Memory allocation in detail.

The concept of dynamic memory allocation in c language enables the c programmer to allocate memory at run time. The four predefined functions of dynamic memory allocation can be used by including stdlib.h header file.

Following are four dynamic memory allocation function.

a. malloc()
b. calloc()
c. realloc()
d. free()

a. malloc():

- The malloc() function allocates single block of requested memory
- Syntax of malloc()

```
ptr = (datatype *) malloc (sizeof (datatype));
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  int n, i, *ptr, sum = 0;
  printf (" Enter number of elements ");
  scanf ("%d", &n);
  ptr = (int*) malloc (n* sizeof (int));
  if (ptr = = NULL)
  {
     printf (" Error in memory allocation ");
  }
  printf (" Enter elements of array");
  for(i=0; i<n; i++)
  {
    scanf (" %d", &ptr+i);
    sum + = *(ptr+i);
  }
  printf (" Sum = %d", sum);
  free (ptr);
}
```

- free is a dynamic memory allocation function used to deallocate the memory allocated by malloc(), calloc(), or realloc().

b. calloc()

- The calloc() allocates multiple block of requested memory.

Syntax:

```
ptr = (datatype*) calloc (sizeof (datatype));
```

Program:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int n, i, *ptr, sum = 0;
    printf ("Enter the number of elements");
    scanf ("%d", &n);
    ptr = (int *) calloc (n* sizeof (int));
    if (ptr == NULL)
    {
        printf ("Error");
    }
    printf ("Enter the elements of array");
    for (i = 0; i<n; i++)
    {
        scanf ("%d", &ptr + i);
        sum += *(ptr+i);
    }
    printf ("Sum = %d", sum);
    free (ptr);
}
```

c. realloc():

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

Syntax:

```
ptr = (datatype *) realloc (ptr, new size);
```

Program:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

```
int main()
{
    char *str;
    str = (char*) malloc(5);
    strcpy(str, "India");
    printf("String = %s, Address = %u", str, str);
    str = (char*) realloc(str, 25);
    strcat(str, ".com");
    printf("String = %s", Address = %u", str, str);
    free(str);
    return 0;
}
```

5. Write a program to explain the usage of pointers in Structures.

```
#include <stdio.h>
#include <conio.h>
struct student
{
    int rno;
    char name[20];
    char course[20];
    float fees;
} *ptr_stud1;
int main()
{
    struct student *ptr_stud1;
    struct student stud1 = {01, "Rahul", "BCA", 45000};
    ptr_stud1 = &stud1;
    printf("Roll Number = %d", ptr_stud1 -> rno);
    printf("Name = %s", ptr_stud1 -> name);
    printf("Course = %s", ptr_stud1 -> course);
    printf("Fees = %d", ptr_stud1 -> fees);
    return 0;
}
```