STRUCTURE :

A structure creates a data type that can be used to group items of possibly different types into a single type. Structure can have elements of different types.

How to create a structure?

‘struct’ keyword is used to create a structure.

SYNTAX OF STRUCTURE :

```
struct   tag_name

{

  datatype  member1;

  datatype  memeber2;

     .

     .

};
```

The structure tag is optional and each member definition is a normal variable definition,such as int i, or float f, or any other valid variable definition.At the end of the structure’s definition, before the final semicolon, one or more structure variables can be created.

```
struct   student

{

          char name [20];
```

```
        int regno;
        int mark1;
        int mark2;
        int mark3;
};
struct student m1,m2,m3;
```

HOW TO ACCESS THE STRUCTURE MEMEBERS ?

The member access operator (.) is used to access any member of a structure. The member access operator is coded as a period between the structure variable name and the structure member.

/*program using structures*/

```
#include<stdio.h>
#include<conio.h>
struct student
{
    int eng;
    int tam;
    int major;
};
```

```
void main()
{
      struct student s[3];
      int i,total=0;
      clrscr();
for(i=0;i<=2;i++)
{
     printf(" enter marks for 3 students\n ");
     printf("enter eng marks \n");
     scanf("%d", &s[i].eng);
     printf("enter tam marks \n");
     scanf("%d", &s[i].tam);
     printf("enter major marks \n");
     scanf("%d", &s[i].major);
      total=s[i].eng+s[i].tam+s[i].major;
      printf("total marks to s[%d] student=%d", i, total);
      getch();
}
}
```

<u>Output :</u>

```
Enter marks for 3 students
Enter eng mark
56
Enter tam mark
67
Enter major mark
78
total marks to s[0] student=201
Enter marks for 3 students
Enter eng mark
78
Enter tam mark
98
Enter major mark
90
Total marks to s[1]=266
Enter marks for 3 students
Enter eng mark
98
```

```
Enter tam mark
87
Enter major mark
67
Total marks to s[2]=252
```

COPYING AND COMPARING STRUCTURE VARIABLES :

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

Program Example :

```
#include<stdio.h>
#include<conio.h>
  struct class
  {
            int no;
            char name[20];
           float marks;
    }
     void main ( )
      {
                 int x;
            struct class s1 = {100,"Nithya",70};

            struct class s2  = {200,"Ramya",90};

        s3 = s2;

       x = (( s3.no == s2.no)  && (s3.marks == s2.marks )) ? 1 : 0;

        if (x==1)

        {

            printf ("Student 2 and Student3 are same");

            printf ("%d %s %f", s3.no,s3.name,s3.marks);

}

 else

             printf ("Student2 and student3 are different");

}
```

Output :

```
    Student 2 and student3 are same

    200 Ramya 90
```

ARRAYS OF STRUCTURES :

An array of structures is stored inside the memory in the same way as a multi-dimensional array.

Program Example :

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

```
 #include<conio.h>
struct marks
{
      int sub1;
      int sub2;
      int sub3;
      int total;
};
void main ( )
{
         int i;
         struct  marks student [ 3 ] =
            { {45,67,81,0},{75,53,69,0},{57,36,71,0};
          struct  marks total;
          for (i=0; i<= 2; i++)
          {
          student[i].total = student[i].sub1 +
                                   student[i].sub2+student[i].sub3;
          total.sub1 = total.sub1 + student[i].sub1;
          total.sub2 = total.sub2 + student[i].sub2;
          total.sub3 = total.sub3 + student[i].sub3;
total.total = total.total + student[i].total;
}
 printf (“STUDENT              TOTAL\n\n”);
for (i=0; i<=2; i++)
  printf(“Student[%d]  %d”, i+1,student[i].total);
  printf (“\n SUBJECT      TOTAL\n\n”);
  printf (“%s    %d\n%s                %d\n%s  %d\n”),
      “Subject 1 “, total.sub1, “Subject 2 “,total.sub2, “Subject 3 “,
total.sub3);
 printf (“\n Grand Total = %d\n, total .total);
}
```

Output :

```
    STUDENT   TOTAL
    Student [1]    193
    Student [2]     197
   Student [3]      164

  SUBJECT    TOTAL
      Subject1         177
      Subject2          156
      Subject3          221
```

Grand Total = 554

STRUCTURES AND FUNCTIONS :

A structure can be passed to any function from main function or from any sub function.

Structure definition will be available within the function only.

It won't be available to other functions unless it is passed to those functions by value or by address.

The general format of sending a copy of a structure to the called function is :

```
function_name(structure_variable_name);
```

The called function takes the following form :

```
datatype functionname (struct_type  st_name)
 {
      -----
      -----
    return (expression);
  }
```

Program Example :

```
 struct student
{
      char name[50];
```

```
            int age;
};
void display(struct student s);
void main ( )
{
         struct student s1;
         printf (“Enter the Name”);
         scanf (“%s”, s1.name);
         printf (“Enter Age”);
         scanf (“%d”, s1.age);
         display (s1);
}
void display (struct student s)
 {
       printf ( “Displaying information”);
       printf (“Name %s”, s.name);
       printf (“Age %d”, s.age);
}
```

Union

Union is a user-defined data type, just like a structure. Union combines objects of different types and sizes together. The union variable allocates the

memory space equal to the space to hold the largest variable of union. It allows varying types of objects to share the same location.

Syntax of Declaring Union

```
union [name of union]
   {
      type member1;
      type member2;
      type member3;
   };
```

Union is declared using the "union" keyword and name of union. Number 1, number 2, number 3 are individual members of union. The body part is terminated with a semicolon (;).

Example of Union in C Programming

```
#include <stdio.h>

union item
{
   int x;
   float y;
   char ch;
};

int main( )
{
   union item it;
   it.x = 12;
   it.y = 20.2;
   it.ch = 'a';

   printf("%d\n", it.x);
   printf("%f\n", it.y);
   printf("%c\n", it.ch);

   return 0;
}
```

Output:

```
1101109601

20.199892

a
```

DIFFERENCE BETWEEN STRUCTURE AND UNION

Structure	Union
Struct keyword to define a structure.	Union keyword to define a union.
Every member within structure is assigned a unique memory location.	In union, a memory location is shared by all the data members.
Changing the value of one data member will not affect other data members in structure.	Changing the value of one data member will change the value of other data members in union.
It enables to initialize several members at once.	It enables to initialize only the first member of union.
The total size of the structure is the sum of the size of every data member.	The total size of the union is the size of the largest data member.
It is mainly used for storing various data types.	It is mainly used for storing one of the many data types that are available.
It occupies space for each and every member written in inner parameters.	It occupies space for a member having the highest size written in inner parameters.

It supports flexible array.	It does not support a flexible array.

BIT FIELD

A bit field is a set of adjacent bits whose size can be from 1 to 16 bits in length. A word can therefore be divided into a number of bit fields.

Bit Field Declaration

```
struct {
        datatype name1 : bit-length;
        datatype name2   : bit-length;
          -----
          -----
        datatype name N : bit-length;
};
```

Program Example :

```
#include <stdio.h>
#include <string.h>

struct {
  unsigned int age : 3;
} Age;

int main( ) {

  Age.age = 4;
  printf( "Sizeof( Age ) : %d\n", sizeof(Age) );
  printf( "Age.age : %d\n", Age.age );

  Age.age = 7;
  printf( "Age.age : %d\n", Age.age );

  Age.age = 8;
```

```
   printf( "Age.age : %d\n", Age.age );

   return 0;
}
```

Output :

```
Sizeof( Age ) : 4
Age.age : 4
Age.age : 7
Age.age : 0
```

Text Book :

E.Balaguruswamy, Programming in ANSI C- 4e.

References :

www.guru99.com

www.tutotialspoint.com