Structure

So far I told you about the data types which can store only one type of data. E.g. int can only store integers, char can only store characters and so on. But in real life programming we hardly find some data which can be stored by using only one data type. Suppose we want to store the information of a student in a school. His information may contain his name which will be a string, his class which will be integer, his marks which will be float and so on.
Obviously we can also use arrays to store this information but it would be inefficient if I want to store information of 1000-2000 students. So basically we need a data type which can store values of dis-similar types. We need a data type which can be used to access those dis-similar type values easily. To solve this problem the concept of structures was introduced. So today I will give a basic overview of structure, its declaration and initialization.

Structure is the derived defined data types that hold variables of different data types. Basically it gives a common name to the collection of different data types. We can also say that structure is a collection of dissimilar type of elements.

Defining a Structure

We have to define a structure first before using it. Given below is one example that will show how structure is defined.

```
struct student
{
char name[10];
int marks;
int roll_no;
};
```

Here struct is keyword and student is the name of structure. name, marks and roll_no are the elements of this structure. You can clearly see that all these elements are of different data type. Remember declaration of structure does not reserve any space in memory.

struct student s1,s2,s3; Here s1, s2 and s3 are student type variables. We can also declare structure variables while defining the structure. This is shown in below example.

```
struct student
{
char name[10];
int marks;
int roll_no;
}s1,s2,s3;
```

Structure Initialization

```
struct book
{
char name[10] ;
float price ;
int pages ;
};
struct book a1={"Chemistry",190.00,210};
struct book a2={"GK",250.80,559};
```

In the above code a1 and a2 are the structure variables. Consider carefully I have stored name, price and pages one by one.

Tips while using Structure

· Generally people forget to write semi colon at the end of closing brace of structure. It may give some big errors in your program. So write semi colon every time at the end of structure definition.

· Usually people declare structure variables at the top of the program. While making some big programs, programmers prefer to make separate header file for structure declaration. They include them at the beginning of the program.

Accessing Elements of Structure

It is important to define the structure before accessing it. In arrays we use subscript to access the array elements. But this concept is slightly different from them. To access the structure elements we use dot operator or member access operator (.). We have to write the name of structure variable followed by a dot and structure element. One small example is given below.

```
//structure definition
struct student
{
char name[10];
int roll;
int marks;
}s1;
struct student s1={"Rishi",4,56}; //storing values
//accessing structure elements
printf("%s%d%d",s1.name,s1.roll,s1.marks);
```

Above code will display the name, roll number and marks of student s1. Consider carefully I have accessed the elements using dot operator.

Memory Allocation of Structure Elements

Structure elements are stored similar to array elements. It means the structure elements are also stored in contiguous memory locations. As I have said earlier, at the time of structure variable declaration, memory is allotted to it. In our above example 14 bytes (10 for string and 4 for two integers) will be allocated for s1.

Array of Structure

Before proceeding to this topic I want to ask you – Is it possible to create array of structure? Well of course Yes. If we can make array of pointers, array of chars and so on then we can also make array of structure.

Making an array of structure is similar to creating a normal array. For this we only need to define a structure. After that we have to make an array of structure. Consider below program to understand it properly.

```
#include<stdio.h>
int main()
{
struct library
{
char lang[10];
int pages;
char name[20];
};
```

```
struct library lib[3];
int i;
for(i=0;i<3;i++)
{
printf("\nEnter language, pages and name\n");
scanf("%s%d%s",&lib[i].lang,&lib[i].pages,&lib[i].name);
}
for(i=0;i<3;i++)
printf ( "\n%s\t%d\t%s\n",lib[i].lang,lib[i].pages,lib[i].name);
return 0;
}
```

Assign one structure variable to another

C language gave user the power to create your own variables. Structures are generally used to create user defined data types. To make the functionality of structures similar to other variables, the elements of structures are stored in contiguous memory locations. Due to this, one can easily assign all the values of one structure variable to other structure variable.
Remember we can also do piece meal copy by copying every element of structure one by one. Lets make one program to understand this.

```
#include<stdio.h>
#include<string.h>
void main()
{
struct member
{
char name[10];
int age;
float mem_no;
};
struct member a1={"Ashu",34,53};
struct member a2,a3;
/* piece-meal copying */
strcpy(a2.name,a1.name);
a2.age=a1.age;
a2.mem_no=a1.mem_no;
/* copying all elements at one go */
a3=a2;
```

```
printf("%s %d %f",a1.name,a1.age,a1.mem_no);
printf("\n%s %d %f",a2.name,a2.age,a2.mem_no);
printf("\n%s %d %f\n",a3.name,a3.age,a3.mem_no);
}
```

Nesting one structure to another

Nesting is one main feature in C which gives the flexibility to create complex programs. Structures can also be nested to create some complex structures. Generally programmers prefer to stay within 3 layers of nesting. But remember practically there is no limit of nesting in C. Consider the below program to understand this.

```
#include<stdio.h>
void main()
{
struct stud
{
char name[10];
int age;
};
struct parent_name
{
char fath_name[10];
char moth_name[10];
struct stud a1;
};
struct parent_name s1={"Griag","Ema","John",23};
printf("%s %s %s %d\n",s1.fath_name,s1.moth_name,s1.a1.name,s1.a1.age);
}
```

Explanation

Nesting of structure is quite simple. Consider carefully the two dot operators I have used inside printf function to access the elements of first structure

Pointer to structure

So far I told you about the pointer to integer, pointer to character, pointer to array and so on. Similar to this we can also make pointer to a structure. It is used to store the address of a structure. Remember we cannot use (.) dot operator to access the structure elements using structure pointer. We have to use arrow operator (->) for that purpose. Consider the below code.

```
#include<stdio.h>
void main()
{
struct stud
{
char name[10];
int age;
};
struct stud s1={"Shawn",32};
struct stud *s2;
s2=&s1;
printf("%s %d",s1.name,s1.age);
printf("\n%s %d\n",s2->name,s2->age);
}
```

Explanation

Consider carefully I have used arrow operator to access elements at that address

Passing structure elements to function

We can pass each element of the structure through function but passing individual element is difficult when number of structure element increases. To overcome this, we use to pass the whole structure through function instead of passing individual element.

```
#include<stdio.h>
#include<string.h>
void main()
{
struct student
{
char name[30];
char branch[25];
int roll;
}struct student s;
printf(“\n enter name=”);
```



```
gets(s.name);
```

```
printf("\nEnter roll:");
scanf("%d",&s.roll);
printf("\nEnter branch:");
gets(s.branch);
display(name,roll,branch);
}
display(char name, int roll, char branch)
{
printf("\n name=%s,\n roll=%d, \n branch=%s", s.name, s.roll. s.branch);
}
```

Passing entire structure to function

```
#include<stdio.h>
#include<string.h>
struct student
{
char name[30];
int age,roll;
};
display(struct student); //passing entire structure
void main()
108 *Under revision
{
struct student s1={"sona",16,101 };
struct student s2={"rupa",17,102 };
display(s1);
display(s2);
}
display(struct student s)
{
printf("\n name=%s, \n age=%d ,\n roll=%d", s.name, s.age, s.roll);
}
Output: name=sona
roll=16
```

Uses of Structure

Structures are generally used to maintain databases in some organisation. One separate concept of Data Structures is also made to maintain the databases using structures. Apart from this some very common uses are given below.
a) Formatting a floppy
b) Receiving a key from the keyboard
c) Moving the cursor on the display
d) Making small games like Tic Tac Toe
e) Sending the output to printer

UNION

Union is derived data type contains collection of different data type or dissimilar elements. All definition declaration of union variable and accessing member is similar to structure, but instead of keyword struct the keyword union is used, the main difference between union and structure is

Each member of structure occupy the memory location, but in the unions members share same memory location. Union is used for saving memory and concept is useful when it is not necessary to use all members of union at a time.
Where union offers a memory treated as variable of one type on one occasion where (struct), it read number of different variables stored at different place of memory.

Syntax of union:

```
union student
{
datatype member1;
datatype member2;
};
```

Like structure variable, union variable can be declared with definition or separately such as

union union name

{
Datatype member1;
}var1;
Example:- union student s;

Union members can also be accessed by the dot operator with union variable and if we have pointer to union then member can be accessed by using (arrow) operator as with structure.
110 *Under revision
Example:- struct student
struct student
{
int i;
char ch[10];
};struct student s;

Here datatype/member structure occupy 12 byte of location is memory, where as in the union side it occupy only 10 byte.