Rajmani Kumar,
Lecturer, Dept. of BCA
S.U.College, Hilsa (Nalanda)
Patliputra University, Patna

BCA-2nd Year **Paper-III**

STRUCTURE AND UNION

STRUCTURE

A Structure is a user defined data type that can store related information together. The variable within a structure are of different data types and each has a name that is used to select it from the structure. C arrays allow you to define type of variables that can hold several data items of the same kind but **structure** is another user defined data type available in C programming, which allows you to combine data items of different kinds.

Structures are used to represent a record, Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book:

- Title
- Author
- Subject
- Book ID

Structure Declaration

It is declared using a keyword struct followed by the name of the structure. The variables of the structure are declared within the structure.

Example:

```
Struct struct-name
{
data_type var-name;
data_type var-name;
};
```

Structure Initialization

Assigning constants to the members of the structure is called initializing of structure.

Syntax:

struct struct_name
{
data _type member_name1;
data _type member_name2;
} struct_var={constant1,constant2};

Accessing the Members of a structure

A structure member variable is generally accessed using a '.' operator.

Syntax: *strcut_var.*
member_name;

The dot operator is used to select a particular member of the structure. To assign value to the individual
Data members of the structure variable stud, we write,

stud.roll=01;
stud.name="Rahul";

To input values for data members of the structure variable stud, can be written as,

scanf("%d",&stud.roll);
scanf(''%s",&stud.name);

To print the values of structure variable stud, can be written as:

printf("%s",stud.roll);
printf("%f",stud.name);

Size of structure-

Size of structure can be found out using sizeof() operator with structure variable name or tag name with keyword.

sizeof(struct student); or
sizeof(s1);
sizeof(s2);

Size of structure is different in different machines. So size of whole structure may not be equal to sum of size of its members.

Nested structure

When a structure is within another structure, it is called Nested structure. A structure variable can be a member of another structure and it is represented as

struct student
{

```
element 1;
element 2;
……
……
struct student1
{
member 1;
member 2;
}variable 1;
…….
…….
element n;
}variable 2;
```

It is possible to define structure outside & declare its variable inside other structure.

```
struct date
{
int date,month;
};
struct student
{
char nm[20];
int roll;
struct date d;
}; struct student s1;
struct student s2,s3;
```

Nested structure may also be initialized at the time of declaration like in above **example.**

```
struct student s={“name”,200, {date, month}};
                  {“ram”,201, {12,11}};
```

Nesting of structure within itself is not valid. Nesting of structure can be extended to any level.

```
struct time
{
int hr,min;
};
struct day
{
```

```
int date,month;
struct time t1;
};
struct student
{
char nm[20];
struct day d;
}stud1, stud2, stud3;
```

SELF REFERENTIAL STRUCTURE

Self –referential structures are those structures that contain a reference to data of its same type as that of structure.

Example

```
struct node
{
int val;
struct node*next;
};
```

Array of structures

When database of any element is used in huge amount, we prefer Array of structures.

Example:

suppose we want to maintain data base of 200 students, Array of structures is used.

```
#include<stdio.h>
#include<string.h>
struct student
{
char name[30];
char branch[25];
int roll;
};
void main()
{
struct student s[200];
int i;
s[i].roll=i+1;
printf("\nEnter information of students:");
```

```
for(i=0;i<200;i++)
{
printf("\nEnter the roll no:%d\n",s[i].roll);
printf("\nEnter the name:");
scanf("%s",s[i].name);
printf("\nEnter the branch:");
scanf("%s",s[i].branch);
printf("\n");
}
printf("\nDisplaying information of students:\n\n");
for(i=0;i<200;i++)
{
printf("\n\nInformation for roll no%d:\n",i+1);
printf("\nName:");
puts(s[i].name);
printf("\nBranch:");
puts(s[i].branch);
}
}
```

In Array of structures each element of array is of structure type as in above example.

Array within structures

```
struct student
{
char name[30];
int roll,age,marks[5];
}; struct student s[200];
```

We can also initialize using same syntax as in array.

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()
{
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

UNION

Union is a collection of variables of different data types, in case of union information can only be stored In one field at any one time. A **union** is a special data type available in C that enables you to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multi-purpose.

Declaring Union

```
union union-name
{
data_type var-name;
data_type var-name;
};
```

The **union 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 union's definition, before the final semicolon, you can specify one or more union variables but it is optional. Here is the way you would define a union type named Data which has the three members i, f, and str. Now, a variable of **Data** type can store an integer, a floating-point number, or a string of characters. This means that a single variable ie. same memory location can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

The memory occupied by a union will be large enough to hold the largest member of the union. For example, in above example Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by character string. Following is the example which will display total memory size occupied by the above union:

Accessing a Member of a Union

```
#include <stdio.h>
#include <string.h>
union Data
{
int i;
float f;
char str[20];
};
int main( )
```

```
{
union Data data;
data.i = 10;
data.f = 220.5;
strcpy( data.str, "C Programming");
printf( "data.i : %d\n", data.i);
printf( "data.f : %f\n", data.f);
printf( "data.str : %s\n", data.str); return 0;
}
```

Dot operator can be used to access a member of the union . he member access operator is coded as a period between the union variable name and the union member that we wish to access. You would use **union** keyword to define variables of union type. Following is the example to explain usage of union:

Exercises:

1. Write a program to define a union and a structure both having exactly the same members. Using the sizeof operator, print the size of structure variable as well as union variable and comment on the result.