

Unit-9

Structure and Union

Introduction

A structure is a collection of logically related data items grouped together under a single name. In structure the individual elements may differ in type, that's why we can regard structure as a heterogeneous user-defined data type. The data items enclosed within a structure are known as members.

Syntax for structure definition is:

```
struct struct_name
{
    data_type mem1;
    data_type mem2;
    .....
    data_type memn;
};
```

The structure definition starts with keyword `struct` followed by an identifier which is a tag name. The tag name is structure name and can be used for instantiating structure variable. `struct_name` is referred to as structure name or structure tag name, and `mem1`, `mem2`, `memn` are known as structure members.

After the structure has been specified, the structure variable can be declared as standard data type:

```
struct struct_name variable_name;
```

Where `struct` is a required keyword, `struct_name` is the name that appeared in the structure definition and `variable_name` is structure variable of type `struct_name`.

Example

```
struct student
{
    char name[15];
    int roll;
    float fee;
};
struct student st;
```

Accessing members of structure

For accessing any member of a structure variable, we use the dot(.) operator which is also known as the period or membership operator. Syntax:

```
structvariable_name.member
```

Initialization of structure Variables

The syntax of initializing structure variables is similar to that of arrays. All the values are given in curly braces and the number, order and type of these values should be same as in the structure template definition.



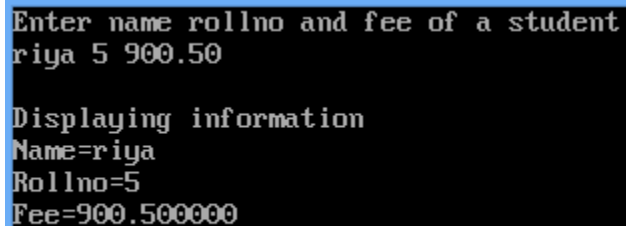
Example

```
struct student
{
    char name[15];
    int roll;
    float fee;
};
struct student st={"Sonia",23,1450.50};
```

Write a program to read name, rollno and fee of a student and display it on screen using structure.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    struct student
    {
        char name[15];
        int roll;
        float fee;
    };
    struct student st;
    clrscr();
    printf("Enter name rollno and fee of a student\n");
    scanf("%s%d%f",st.name,&st.roll,&st.fee);
    printf("\nDisplaying information\n");
    printf("Name=%s \nRollno=%d \nFee=%f",st.name,st.roll,st.fee);
    getch();
}
```

Output



```
Enter name rollno and fee of a student
riya 5 900.50

Displaying information
Name=riya
Rollno=5
Fee=900.500000
```

Array of structures

We can declare array of structures where each element of array is of structure type.

Example

- Define a structure containing members as roll no, name, course and semester and write program to input information about 'n' students and display the name and course of all students.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    struct student
    {
        int roll;
        char name[15];
```

```

char course[15];
char sem[15];
};
struct student st[100];
int i,n;
clrscr();
printf("\nHow many students are there: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter rollno, name, course and semester: ");
scanf("%d%s%s", &st[i].roll, st[i].name, st[i].course, st[i].sem);
}
printf("\nName\tCourse");
for(i=0;i<n;i++)
{
printf("\n%s\t%s", st[i].name, st[i].course);
}
getch();
}

```

Output

```

How many students are there: 3

Enter rollno, name, course and semester: 1 mohan digital 1st

Enter rollno, name, course and semester: 2 rohan Cprog 2nd

Enter rollno, name, course and semester: 3 rakesh english 3rd

Name      Course
mohan     digital
rohan     Cprog
rakesh    english

```

Arrays within structure

We can have an array as a member of structure.

Example

- Write a program to read rollno, name and marks of students in 5 different subjects for 'n' students and display all records of students in appropriate format.

```

#include<stdio.h>
#include<conio.h>
void main()
{
struct student
{
int roll;

```

```

char name[15];
int submarks[5];
};
struct student st[100];
int i,j,n;
clrscr();
printf("\nHow many students are there: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter rollno, name: ");
scanf("%d%s",&st[i].roll,st[i].name);
for(j=0;j<5;j++)
{
printf("\n Enter subject marks for %s: ",st[i].name);
scanf("%d",&st[i].submarks[j]);
}
}
printf("\nRollno\tName\tsub1\tsub2\tsub3\tsub4\tsub5");
for(i=0;i<n;i++)
{
printf("\n%d\t%s",st[i].roll,st[i].name);
for(j=0;j<5;j++)
{
printf("\t%d",st[i].submarks[j]);
}
}
getch();
}

```

Output

```

How many students are there: 1

Enter rollno, name: 2 ram

Enter subject marks for ram: 56

Enter subject marks for ram: 78

Enter subject marks for ram: 90

Enter subject marks for ram: 88

Enter subject marks for ram: 56

Rollno  Name    sub1    sub2    sub3    sub4    sub5
2       ram      56      78      90      88      56_

```

Nested structure

The members of a structure can be of any data type including another structure type i.e we can include a structure within another structure. A structure variable can be a member of another structure. This is called nesting of structure.

Syntax:

```
struct struct_name1
{
    data_type mem1;
    data_type mem2;
    .....
    data_type memn;
};

struct struct_name2
{
    data_type mem1;
    data_type mem2;
    .....
    struct_name1 structure_variable1;
    data_type memn;
};

struct_name2 structure_variable2;
```

Example

- Define a structure **date** having integer members to store day, month and year. Define another structure **student** having members as rollno, name and date_of_birth. Now write a program to accept and display information about 'n' students.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    struct date
    {
        int year;
        int month;
        int day;
    };
    struct student
    {
        int roll;
        char name[15];
        struct date dob;
    };
    struct student st[100];
    int i,n;
    clrscr();
    printf("\n How many students are there: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Enter rollno,name and date of birth in (year-month-day): ");
        scanf("%d%s%d%d%d",&st[i].roll,st[i].name,&st[i].dob.year,&st[i].dob.month,&
        st[i].dob.day);
    }
    printf("\nRollno\tName\tDOB (Year-Month-Day)");
    for(i=0;i<n;i++)
```

```

{
printf("\n%d\t%s\t%d-%d-%d",st[i].roll,st[i].name,st[i].dob.year,st[i].dob.month,st[i].dob.day);
}
getch();
}

```

Output

```

How many students are there: 2

Enter rollno,name and date of birth in (year-month-day): 1 salina 2048 1 7

Enter rollno,name and date of birth in (year-month-day): 2 riya 2045 3 25

Rollno   Name      DOB(Year-Month-Day)
1        salina   2048-1-7
2        riya     2045-3-25_

```

Pointer to structure

We can have pointer to structure, which can point to the starting address of a structure variable. These pointers are called structure pointers. While accessing structure members through pointers we have to use arrow operator (->) which is formed by hyphen symbol and greater than symbol.

Example

```

#include<stdio.h>
#include<conio.h>
void main()
{
struct customer
{
int id;
char name[15];
char address[15];
};
struct customer cu={1,"Binod","Bharatpur"};
struct customer *ptr;
ptr=&cu;
clrscr();
printf("\n ID :%d",ptr->id);
printf("\n Name :%s",ptr->name);
printf("\n Address :%s",ptr->address);
getch();
}

```

Output

```

ID :1
Name :Binod
Address :Bharatpur

```

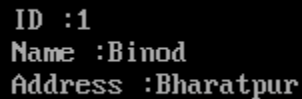
Passing structure to function

We can pass whole structure as an argument to a function.

Example

```
#include<stdio.h>
#include<conio.h>
void display(struct customer);
struct customer
{
    int id;
    char name[15];
    char address[15];
};
void main()
{
    struct customer cu={1,"Binod","Bharatpur"};
    clrscr();
    display(cu);
    getch();
}
void display(struct customer c)
{
    printf("\n ID :%d",c.id);
    printf("\n Name :%s",c.name);
    printf("\n Address :%s",c.address);
}
```

Output

A screenshot of a terminal window with a black background and white text. It displays the output of the program: 'ID :1', 'Name :Binod', and 'Address :Bharatpur' on three separate lines.

```
ID :1
Name :Binod
Address :Bharatpur
```

Passing array of structure to function

We can pass the array of structure to function, where each element of array is of structure type.

Example

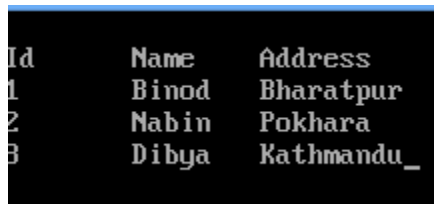
```
#include<stdio.h>
#include<conio.h>
struct customer
{
    int id;
    char name[15];
    char address[15];
};
void display(struct customer c[]);
void main()
{
```

```

struct customer
cu[3]={ {1,"Binod","Bharatpur"}, {2,"Nabin","Pokhara"}, {3,"Dibya","Kathmandu"}
};
clrscr();
display(cu);
getch();
}
void display(struct customer c[])
{
int i;
printf("\nId\tName\tAddress ");
for(i=0;i<3;i++)
{
printf("\n%d\t%s\t%s",c[i].id,c[i].name,c[i].address);
}
}
}

```

Output



```

Id      Name      Address
1       Binod    Bharatpur
2       Nabin    Pokhara
3       Dibya     Kathmandu_

```

- Write a program that reads names and ages of 'n' students into the computer and rearrange the names into alphabetical order using the structure variables.

```

#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
struct student
{
char name[15];
int age;
};
struct student st[100],temp;
int i,j,n;
clrscr();
printf("How many students are there: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter student name and age: ");
scanf("%s%d",st[i].name,&st[i].age);
}
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(strcmp(st[i].name,st[j].name)>0)

```



```

{
temp=st[i];
st[i]=st[j];
st[j]=temp;
}
}
}
printf("\nName\tAge");
for(i=0;i<n;i++)
{
printf("\n%s\t%d",st[i].name,st[i].age);
}
getch();
}

```

Output

```

How many students are there: 5

Enter student name and age: ramesh 45

Enter student name and age: mohan 23

Enter student name and age: riya 20

Enter student name and age: dibya 22

Enter student name and age: sohan 31

Name    Age
dibya   22
mohan   23
ramesh  45
riya    20
sohan   31

```

Union

Union is a derived data type like structure which can contain members of different data type. Union members share the same memory locations. Compiler allocates sufficient memory to hold the largest member in the union. We can use only one member at a time. Union is used for saving memory.

Syntax for defining union

```

union union_name
{
    data_type mem1;
    data_type mem2;
    .....
    data_type memn;
};

```

Like structure variable union variable is also needed to be declared for accessing members of union.

Syntax

```
union union_name union_variable;
```

Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
    union student
    {
        int roll;
        char name[15];
    };
    union student st;
    clrscr();
    printf("\n Enter roll no: ");
    scanf("%d",&st.roll);
    printf(" Rollno :%d",st.roll);
    printf("\n Enter name: ");
    scanf("%s",st.name);
    printf(" Name :%s",st.name);
    getch();
}
```

Output

```
Enter roll no: 12
Rollno :12
Enter name: ramesh
Name :ramesh
```

➤ Difference between structure and union

Structure	Union
1. Memory occupied by structure is sum of individual data type.	1. Memory occupied by union is of highest data type of all.
2. Can take part in complex data structure.	2. Cannot take part in complex data structure.
3. Keyword struct is used.	3. Keyword union is used.
4. Every element value's are independent to each other.	4. If any of the values of any element has been changed there is direct impact to the other elements values.
5. Memory allocation of every element is independent to each other thereby the memory allocation is sum of every element.	5. Memory allocation is performed by sharing the memory with highest data type.
6. All members can be accessed simultaneously.	6. Only one member is active at a time, so only one member can be accessed at a time.
7. Syntax struct struct_name { data_type mem1;	7. Syntax union union_name { data_type mem1;

<pre>data_type mem2; data_type memn; };</pre>	<pre>data_type mem2; data_type memn; };</pre>
---	---

➤ Difference between array and structure

Array	Structure
1. Array is a built-in data type.	1. Structure is a derived data type.
2. Array holds the group of same elements under a single name.	2. Structure holds the group of different elements under a single name.
3. We cannot have array of array.	3. We can have array of structure.
4. Memory occupied by an array is the multiple of no of index	4. Memory occupied by structure is sum of individual data type.
5. Cannot take part in complex data structure.	5. Can take part in complex data structure.
6. Cannot be used in program to interact with hardware.	6. Can be used in program to interact with hardware.
7. Syntax for declaration data_type arrayname[subscript];	7. Syntax for declaration struct struct_name { data_type mem1; data_type mem2; data_type memn; };