UNIT – V

16

CHAPTER

STRUCTURES AND UNIONS

16.1 STRUCTURES

A structure is a collection of one or more variables of different data types of data items that are stored under a single name in a block of memory. By using structures, we can make a group of variables, arrays, pointers etc.

16.1.1 Features of Structures

- ❖ One structure element of different data types of data items can be copied to another structure variable using assignment operators like variable. Example:book2=book1
- ❖ Nesting of structure is possible.
- ❖ Structures support to pass their elements to a function.
- ❖ Structure pointers can be created.

16.2 DECLARATION AND INITIALIZATION OF STRUCTURES

Structures can be declared as follows:

```
struct struct_name
{
datatype variable1;
datatype variable2;
};
struct struct_name variable1,variable2;
```

where, struct is a keyword.struct_name is a name of structure that identifies structure. variable1 and variable2 are individual member declaration.

Accessing Structure Members

The structure member is accessed using *member operator* "." which is also known as 'dot operator'.

Three ways to access members

• Using dot notation	:	v.x
• Using indirection notation	:	(*ptr).x
• Using selection notation	:	ptr->x

Example

Creating a structure for Student record

```
struct student
{       int rollno;
            char name[20];
            int marks[5];
            int avg;
 }
struct student s1
```

Example Program

```
#include <stdio.h>
void main()
{
struct student
{
int regno;
char name[30];
char branch[30];
int marks[10];
};
struct student s;
int total=0,i,n;
clrscr();
printf("Enter the reg.No");
scanf("%d",&s.regno);
fflush(stdin);
printf("\nEnter the name");
gets(s.name);
fflush(stdin);
printf("\nEnter the branch");
gets(s.branch);
printf("\nEnter the marks one by one\n");
for(i=0;i<7;i++)
{
scanf("%d",&s.marks[i]);
total=total+s.marks[i];
}
clrscr();
printf("\n\n_______STUDENT MARKS PROCESSING_______________\n");
printf("\n\t Register No            :  %d",s.regno);
printf("\n\t Name of the Student    :  %s",s.name);
printf("\n\t Branch                 :  %s",s.branch);
printf("\n\t  English               :  %d",s.marks[0]);
printf("\n\t  Mathematics           :  %d",s.marks[1]);
printf("\n\t  Chemistry             :  %d",s.marks[2]);
printf("\n\t  Physics               :  %d",s.marks[3]);
printf("\n\t  Computer Programming  :  %d",s.marks[4]);
printf("\n\t Electronic Devices     :  %d",s.marks[5]);
if(s.marks[0]>=50&&s.marks[1]>=50&&s.marks[2]>=50&&s.marks[3]>=50&&
s.marks[4]>=50&&s.marks[5]>=50&&s.marks[6]>=50)
printf("\n\t  Result                :  PASS");
else
printf("\n\t  Result                :  FAIL");
printf("\n\t  Total                 :  %d",total);
getch();
}
```

OUTPUT

```
Enter the reg.No101
Enter the nameaarthi
Enter the branchcse
Enter the marks one by one
89
90
99
98
99
88
____________STUDENT MARKS PROCESSING_______________
Register No              :  101
Name of the Student      :  aarthi
Branch                   :  cse
English                  :  89
Mathematics              :  90
Chemistry                :  99
Physics                  :  98
Computer Programming     :  99
Electronic Devices       :  88
Result                   :  PASS
Total                    :  641
```

16.3 ARRAYS OF STRUCTURES

Arrays of structures are collection of structures which are used to store different types of structure member variables. They are used to handle more records within one structure.

EXAMPLE

```
struct book
   {
     char name[10];
     int price;
    };
    struct book b[3];
    void main( )
    {
       int i;
       for(i=1;i<=3;i++)
        {
             printf("Enter the book name, price:\n");
             scanf("%s%d",&b[i].name,&b[i].price);
         }
           for(i=1;i<=3;i++)
           printf("\n%s\t %d",b[i].name,b[i].price);
           getch( );
    }
Syntax
    struct structure_name1
  {
  datatype declarations;
  };
    struct structure_name2
   {
   declarations;
   ........................
   ........................
   struct structure_name1;
   variable_name1;
  .....................
  .....................
 };
```

OUTPUT

```
Enter the book name,price
English  165
Enter the book name,price
Maths    250
```

```
Enter the book name,price
Physics  193
```

16.4 STRUCTURES WITHIN STRUCTURES

One structure declared inside other structure is called as **structures within a structure** also known as ***nesting* of structures.** We can write one structure inside another structure as **member** of another structure. The structure variables may be a normal structure variable or a pointer variable to access the data in the structure.

Syntax

```
struct structure_name1
  {
datatype declarations;
  };
struct structure_name2
  {
declarations;
  ……………………
  ……………………
struct structure_name1;
  variable_name1;
  …………………
  …………………
};
```

EXAMPLE

```
#include<stdio.h>
main( )
{
  struct date
 {
   int day;
  char month[20];
  int year;
 };
  struct employee
  {
    int code;
    char name[30];
    float salary;
   struct date doj;
  };
struct employee emp1;
printf("\n Enter  Employee code:");
scanf("%d",&emp1.code);
printf("\n Enter  Employee name:");
scanf("%s",&emp1.name);
printf("\n Enter  Employee salary:");
scanf("%f",&emp1.salary);
printf("\n Enter  Employee date of  joining:");
scanf("%d %s %d",&emp1.doj.day,&emp1.doj.month,&emp1.doj.year);
printf("\n The Employee code is %d",emp1.code);
printf("\n The Employee Name is %s",emp1.name);
printf("\n The Employee Salary is %f",emp1.salary);
printf("\n The Employee DOJ %d %s %d",emp1.doj.day,emp1.doj.month,emp1.doj.year);
}
```

INPUT

```
Enter Employee code    :  200
Enter Employee name    :  Ram
Enter Employee salary  :  22000.0
Enter Employee Doj is  :  12 December 2014
```

OUTPUT

```
The Employee Code is 200
The Employee Name is Ram
The Employee Salary is22000.0
The Employee DOJ is 12 December 2014
```

16.5 STRUCTURES AND FUNCTION

C supports to pass structure values as arguments to function. There are three methods in C language to transfer values of a structure from one function to another function:

- ❖ Pass each member of the structure as an individual argument of the function (Single element in a structure).
- ❖ Pass a copy of the entire structure to the function (All elements in a structure).
- ❖ Pass the address location of the structure to the function.

Syntax

```
function_name(structure_variable_name);
datatypefunction_name(struct_typestruct_name)
{
……………………..
……………………..
return(expression);
}
```

Example

```
/*Passing a copy of entire structure to a function*/
struct std
 {
  int no;
  float avg;
  };
  void fun(struct std p);
  void main( )
  {
    struct std a;
   clrscr( );
   a.no=1;
   a.avg=90.6;
   fun(a);
   getch( );
  }
  void fun(struct std p)
 {
  printf("Number is:%d\n",p.no);
 printf("Average is:%f\n",p.avg);
 }
Ref page no:16.11
Syntax:
                       struct tag_name
                      {
                      datatype declarations;
                      }*structure_name;
"->"  is called as structure pointer symbol.
```

Output

```
Number is     : 1
Average is    : 90.599998
```

Passing Structures to a Function

- Passing by value (passing actual value as argument).
- Passing by reference (passing address as argument).

Passing structure by value: A structure variable can be passed to the function, this is known as passing structure by value. The change made in structure variable in function definition does not reflect in original structure variable in calling place.

EXAMPLE

```
#include<stdio.h>
struct student{
char name[50];
introllno;
};
void display(struct student stu);
int main( )
{
struct student s1;
printf("\n enter the name:");
scanf("%s",s1.name);
printf("\n enter the roll no:");
scanf("%d',s1.rollno);
display(s1);
return 0;
}
void display(struct student stu)
{
Printf("\n Name:%s",stu.name);
Printf("\n Roll no:%d",stu.rollno);
}
```

Output

```
Enter student name : Max
Enter rollno       : 101
Name               : Max
Roll no            : 101
```

Passing structure by reference: The address location of structure variable can be passed to function, this is known as passing structure by reference. The change made in structure variable in function definition reflects in original structure variable in the calling place.

EXAMPLE

```
#include<stdio.h>
struct distance
{
int feet;
float inch;
};
void Add(struct distance d1,struct distance d2, struct distance *d3);

int main()
{
```

```
struct distance dis1, dis2, dis3;
printf("First distance feet and inch\n");
scanf("%d",&dist1.feet);
scanf("%f",&dist1.inch);
printf("Second distancefeet and inch\n ");
scanf("%d",&dist2.feet);
scanf("%f",&dist2.inch);
Add(dis1, dis2, &dis3);
printf("\nSum of distances = %d\'-   %.1f\"",dis3.feet, dis3.inch);
return 0;
}
void Add(struct distance d1,struct distance d2, struct distance *d3)
{
     d3->feet=d1.feet+d2.feet;
     d3->inch=d1.inch+d2.inch;
if (d3->inch>=12)
{
         d3->inch-=12;
         ++d3->feet;
     }
}
```

Output

```
First distance feet and inch
13
7.7
Second distance feet and inch
6
8.5
Sum of distances = 20'-4.2"
```

16.6 STRUCTURE AND POINTERS

In C language, structure can be accessed in two ways, using normal structure variable and using pointer variable.

Syntax

```
            structtag_name
                        {
            datatype declarations;
                        }*structure_name;
```

"->" is called as structure pointer symbol.

A structure containing a pointer member to the same structure type is called self referential structure.

EXAMPLE

```
Structure using pointers
#include<stdio.h>
void main( )
{
struct
     {
introllno;
char name[30];
char branch[4];
int marks;
     }*stud;
clrscr();
printf("\n enter the rollno:");
scanf("%d",&stud->rollno);
printf("\n enter name:");
scanf("%s",stud->name);
printf("%s",enter the branch:");
printf("\n Roll Number:%d",stud->rollno);
printf("\n Name:%s",stud->name);
printf("\n Branch:%s",stud->branch);
getch();
}
```

Output

```
Enter rollno  :  1001
Enter name    :  Raj
Enter branch  :  CSE

Roll Number   :  1001
Name          :  Raj
Branch        :  CSE
```

Array and structure have similar properties. The name array indirectly specifies the address of its zeroth element. Similarly the name of the structure specifies the address of the zeroth element in the zeroth record.

Example

```
     struct item
     {  char name[20];
     int ID;
  float price;
     } product[5], *ptr;
```

The above segment declares product as an array of 5 elements, each of the type **struct item**and **ptr**is a pointer to the data object of the type**struct item.**

The assignmentptr = product; it assigns the address of the zeroth element of **product** to **ptr.**

Now ptr will point to product[0]. All the structure members name, ID, and price can be accessed through pointer as follows:

```
ptr -> name
ptr->ID
ptr ->price.
```

The symbol -> arrow operator is also known as member selection operator. This special operator is made up of a minus sign and greater than symbol. Each time when the pointer ptr is incremented by one, it automatically points to the next record. The scale factor depends on the size of the structure elements.

To access members of structure with structure variables, we used the dot . operator. But when we have a pointer of structure type, we use arrow -> to access structure members.

Example

```
struct book
       {
              char name[20];
                float price;
       }
     void main()
     {
      struct book b;
      struct book *ptr = &b;
      ptr->name = "William Stallings";
      ptr->price = 250;
    }
```

Example Program

```
#include<stdio.h>
#include<string.h>
struct student
      {  char name[25];
           int id;
           float avg;
      };
void main()
{ inti;
struct student rec1 = " Aswin",1,90.5}
struct student *ptr;
ptr = &rec1;
printf(" Record of the Student 1 : \n");
printf("\n Name of the student = %s \n", ptr->name);
printf("\n Id   is = %d", ptr->id);
printf("\n Average Mark = %f\n", ptr->avg);
}
```

Output

```
Record of the Student 1
Name of the student =Aswin
Id   is = 1
Average Mark = 90.50000
```

16.7 DIFFERENCE BETWEEN ARRAYS AND STRUCTURES

S. No.	ARRAYS	STRUCTURES
1	Single name that represent a group of data items of same data type.	Single name that represents a group of data items of different data types.
2	Individual data in an array are called elements.	Individual data in a structure are called members.
3	There is no keyword to represent arrays but the square braces[] preceding the variable name tells us that we are dealing with array.	The keyword struct tells us that we are dealing with structure.
4	The array elements are accessed by its name followed by square brackets[] within which the subscript value is placed.	The members of a structure are accessed by the dot operator.

EXAMPLE

```
#include <stdio.h>
typedef struct complex
{
float real;
floatimag;
}complex;
complex add(complex c1,complex c2);
int main(){
complex c1,c2,temp;
printf("Enter 1st complex number real and imaginary values: \n");
scanf("%f%f",&c1.real,&c1.imag);
printf("\nEnter 2nd complex number real and imaginary values: \n");
scanf("%f%f",&c2.real,&c2.imag);
temp=add(c1,c2);
printf("Sum=%.1f+%.1fi",temp.real,temp.imag);
return 0;
}
complex add(complex c1,complex c2){
```

```
complex temp;
temp.real=c1.real+c2.real;
temp.imag=c1.imag+c2.imag;
return(temp);
}
```

Output

```
Enter 1st complex number real and imaginary values
2.3
4.5

Enter 1st complex number real and imaginary values
3.4
5
Sum=5.7+9.5
```

16.8 UNION

Union is a variable which is similar to the structure. It contains data elements of different data types. The union requires bytes that are equal to the number of bytes required for the largest members.

Syntax

```
union result
{
int marks;
int grade;
}u1;
```

Here the memory allocation is 2 bytes, whereas if it is a structure the memory allocation is 4 bytes.

EXAMPLE

```
#include<stdio.h>
#include<conio.h>
void main()
{
union student
{
char name[20];
int rollno;
char branch[10];
}student;
clrscr();
printf("enter student name,roll number and branch");
scanf("%s%d%s",student.name,&student.rollno,student.branch);
printf("\n student name:%s",student.name);
printf("\n student rollno:%d",student.rollno);
printf("\n student branch:%s",student.branch);
getch();
}
```

Output

```
Enter student name,roll number and branch
Angel
1560
CSE
Student name   :  Angel
Student rollno :  1560
Student Branch :  CSE
```

16.9 DIFFERENCE BETWEEN UNION AND STRUCTURE

Unions are similar to structures in many ways, but memory allocation is different from the structure. This can be understood easily by the following example:

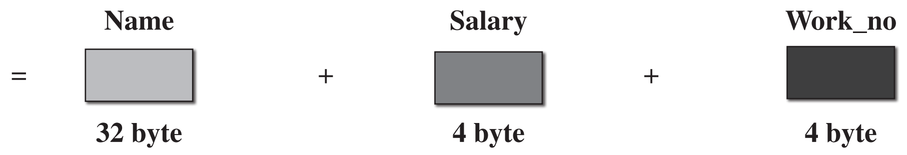
EXAMPLE 1

```
#include <stdio.h>
union job
{
   char emp_name[32];
   float emp_ salary;
   int emp_no;
}u;
struct job1 {
   char emp_name[32];
   float emp_salary;
   int empr_no;
}s;
int main(){
   printf("size of union is = %d" bytes,sizeof(u));
   printf("\n size of structure is = %d" bytes, sizeof(s));
   return 0;
}
```

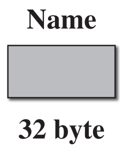
Output

```
Size of union is = 32 bytes
Size of structure is = 40 bytes
```

Structure memory allocation



Union memory allocation



In structure, all members can be accessed at any time. But in union,only one member can be accessed at a time and other members will contain **garbage** value which is explained in the following example:

EXAMPLE 2

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

union item
{
 int item_code;
 float item_price;
 char item_ available;
};

int main( )
{
 union item it;
 it.item_code = 104;
 it.item_price = 126.50;
 it.item_available='Y';
 clrscr();
 printf("%d\n", it.item_code);
 printf("%f\n", it.item_price);
 printf("%c\n", it.item_available);
 getch();
 return 0;
}
```

Output

```
-26426
125.1999
Z
```

Difference between Structure and Union

S. No.	Structure	Union
1.	The keyword is struct.	The keyword is union.

2.	Memory allocation is done for all the data members in the structures.
	
	Example
	struct student
	{
	introllno;
	char name[5];
	}s1;
	The memory allocation is 7 bytes.
		Memory allocation is done for the data member that requires maximum allocations.
		
		Example
		union student
		{
		introllno;
		char name[5];
		}
		The memory allocation is 5 bytes.

3.	All the data members are available in the primary memory at any time of execution.	Only the last stored data element is available in the primary memory at any time of execution.
4.	Since memory is allocated for all the data members, no data is deleted in the primary memory	Since memory is not allocated for all the data members, only one data is available and other data is deleted from the primary memory

16.10 POINTER TO UNION

A union pointer variable which stores the address of union is called as pointer to union.

Syntax

```
union item i1;
unionitem *ptr;
ptr = &i1;
```

EXAMPLE

```
#include<stdio.h>

union item {
  char *name;
  int code;
  float  price;
};

int main()
{
```

```
union item i1,*ptr = &i1;

i1.name = "Book";
printf("\nItem Name : %s",(*ptr).name);
printf("\n Item Name : %s",ptr->name);

return 0;
}
```

Output

```
Item Name: Book
Item Name: Book
```

Note

- **ptr** is the **pointer to union** address.
- **->** and **(*).** both represents the same.