Unit – 11

Understand basics of structures and unions

11.1 Define structure: -

Structure is defined as the collection of elements of different data types .

(OR)

A structure is a data – type that can handle different types of data. It is possible to store different types of elements in the same structure.

11.2 Illustrate creating a structure: -

Unlike arrays, structure must be defined first for their format that may be used later to declare structure variables.

The general format of a structure definition is as follows:

```
struct tag _ name
{
data _type member1;
data _ type member2;
  - - - - - - - - - - - - -
  -- - - - - - - - - - - -
data _ type member n;
};
```

In defining a structure you may note the following syntax:

1. The template is terminated with semicolon.
2. While the entire definition is considered as a statement ,each member is declared independently for its name and type in a separate statement inside the template.
3. The tag-name can be used to be declare structure variables of its type.

Example:

```
struct book-bank
{
 char title[20];
char author[20];
int pages;
float price;
}; /* this semicolon is compulsory*/
```

A structure keyword **struct** declares structure to hold details of four data fields,namely title, author, pages , price. These fields are called structure **elements (or) members** within the left and right curly brackets, we should declare the elements that consist of the structure. Each member may belong to a different type of data.**book-bank** is the name of the structure.

11.3 Illustrate declaring structure variables:

After defining a structure format we can declare variables of that type. It includes

1. The keyword struct .
2. The structure tag name.
3. List of variables names separated by commas .
4. A terminating semicolon .

we can declare the variables of structure as:

Syntax: struct tag_name variable1, variable2……., variablen;

Example: struct book_bank , book1 , book2 , book3 ;

Declares **book1,book2** and **book3** as variables of type **strut book_bank**. Each one of these variables has four members as specified by the template. The complete declaration look like:

```
struct book-bank
{
  char title[20];
  char author[20];
  int pages;
 float price;
};
struct book_bank , book1 , book2 , book3;
```

(or)

```
struct book-bank
{
char title[20];
char author[20];
int pages;
float price;
}book1,book2,book3 ;
```

- The use of tag name is optional .

```
struct
{
 char title[20];
 char author[20];
  int pages;
float price;
}book1,book2,book3;
```

Declares **book1,book2** and **book3** as structure variables.

This approach is not used for two reasons :

1. Without a tag name , we cannot use it for future declaration .
2. Normally , structure definitions appear at the beginning of the program file , before any variables or functions defined. They may also appear before the main , along with macro definitions .

11.4 Explain accessing of the structure members:

Once a structure variable is declared, we can refer to the elements of the structure using that structure variable. There is an operator called 'structure member operator' (or) 'dot operator' denoted by a dot(.) that is useful to refer to the elements through a structure variable.

Syntax : Structure variable .structure member;

Ex: Book1.title;

Here above example book1.title is the variable representing the title of book1 and can be treated like other ordinary variable.

Ex: strcpy(book1.title "basic");

We can also use scanf to give the values through the keyboard.

scanf("%s",book1.title);

Program

```
#include<stdio.h>
struct  personal
{
char name[20];
 int day;
```

```
Input values
abc 27  jan 2020 2000000
abc     27      jan     2020    2000000.000000
```

```
char month[20];
  int year;
  float salary;
};
main()
  {
    struct personal  person;
    printf("Input values \n");
     scanf("%s%d%s%d%f", person.name , &person.day , person.month, &person.year
  ,&person.salary);
    printf("%s\t%d\t%s\t%d\t%f", person.name , person.day , person.month, person.year ,
       person.salary);
}
```

11.5 Explain array of structures:

We may declare an array of structures, each element of the array representing Structurevariable.

Syntax

struct tag-name array_name[size];

Eg: struct class student[60];

Defines an array called student , consists of sixty elements. Each element is defined to be of the type **struct class.**

```
 Ex: struct marks
{
int sub1;
int sub2;
int sub3;
};
main()
{
   struct  marks student[3]={{45,56,58},{56,89,79},{78,98,85}};
}
```

This declares the student as an array of three elements student[0] , student[1], student[2] and initializes their members as follows :

```
student[0].sub1=45;
student[0].sub2=56;
-------------------------
-------------------------
student[2].sub3=85;
```

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

Student[0].sub1	45
Student[0].sub2	56
Student[0].sub3	58
Student[1].sub1	56
Student[1].sub2	89
Student[1].sub3	79
Student[2].sub1	78
Student[2].sub2	98
Student[2].sub3	85

We use the usual array accessing methods to access individual element and then the member operator to access members.

Example program:

```
#include<stdio.h>
struct marks
{
int sub1;
int sub2;
int sub3;
int total;
};
main()
{
int i;
   struct  marks student[3]={{45,56,58,0},{56,89,79,0},{78,98,85,0}};
struct marks total={0,0,0,0};
for(i=0;i<3;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<3;i++)
{
printf(" student[%d]:    %d \n",i+1,student[i].total);
}
printf("SUBJECT       TOTAL\n\n");
printf("%s   %d\n%s   %d\n%s   %d\n","Subject1  ",total.sub1,"Subject2 ",total.sub2,"Subject3
",total.sub3);
printf("Grand total=%d\n",total.total);
}
```

```
STUDENT       TOTAL

 student[1]:    159
 student[2]:    224
 student[3]:    261
SUBJECT       TOTAL

Subject1     179
Subject2    243
Subject3     222
Grand total=644
```

Illustrate concept of structure intilization

- A structure variable can be initialized at compile time.

```
main()
 {
        struct
        {
          int id;
          char name[10];
          float marks;
         }s1= { 50,"sai",180.75};
```

There is a one- to-one correspondence between the members and their initializing values .

- A lot of variation is possible in initializing a structure. It is essential to use a tag name

```
main()
{
struct  student
```

```
{
int id;
char name[10];
float marks;
};
struct student s1= { 50,"sai",180.75};
struct student s2= { 60, "hari",170.75};
-----------------------------------------
      }
```

- Another method is to initialize a structure variable outside the function .

```
struct  student
                {
int id;
char name[10];
float marks;
}s1= { 50,"sai",180.75};
main()
{
     struct student s2= { 60, "hari",170.75};
      }
```

The compile time initialization of a structure variable must have the following elements:

1. The keyword struct .
2. The structure tag name.
3. The name of the variable to be declared .
4. The assignment operator =
5. A set of values for the members of the structure variable , separated by commas and enclosed in braces .
6. A terminating semicolon .

Rules for initializing structures:

1. We cannot initialize individual members inside the structure template.

2. The order of values enclosed in braces must match the order of members in the structure definition.

3. It is permitted to have a partial initialization we can initialize only the few members and leave the remaining blank.

4. The uninitialized members should be only at the end of the list. 5. The uninitialized members will be assigned default values as follows:

- Zero for integer & floating point number
- "\0" for characters & string.

Intilaizing values at compiletime	**Intilaizing values at runtime**

#include<stdio.h>
struct student
{
int id;
char name[10];
float marks;
};
main()
{
int x;
struct student s1= { 50,"sai",180.75};
	#include<stdio.h>
	struct student
	{
	int id;
	char name[10];
	float marks;
	};
	main()
	{
	int x;
	struct student s1;

printf("%d %s%f\n",s1.id,s1.name,s1.marks);
}
	printf("enter details\n");
	scanf("%d%s%f",&s1.id,s1.name,&s1.marks);
	printf("%d %s %f\n",s1.id,s1.name,s1.marks);
	}

11.6 Illustrate concept of structure assignment

It is possible to copy the contents of one structure variable in another structure variable. But the variables should belong to the same structure. For example, take 'student' structure shown here:

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

Here s1,s2 are the structure variables. We want to copy the contents of s1 in s2. There are two ways:

1. We can copy each element from s1 in s2 as:
   s2.id=s1.id;
   strcpy(s2.name,s1.name);
2. We can copy the entire structure variable in another variable as:
   s2=s1;

Example program for structure assignment:

```
#include<stdio.h>
struct  student
{
int id;
char name[10];
float marks;
};
main()
{
int x;
struct student s1= { 50,"sai",180.75};
struct student  s2;
 s2=s1;
printf("%d %s %f\n",s1.id,s1.name,s1.marks);
printf("%d %s %f",s2.id,s2.name,s2.marks);
}
```

```
50 sai 180.750000
50 sai 180.750000
```

11.7 Explain how to find size of structure:

We normally use structures, arrays to create variables of large sizes. The actual size of these variable in terms of bytes may change from machine to machine. We may use the unary operator size of to tell us the size of a structure. The expression:

sizeof(struct x)

Will evaluate the number of bytes required to hold all the members of the structure x. if "y" is a simple structure variable of type struct x, then the expression:

sizeof(y)

Would also give the same answer. If "y" is an array variable of type struct x, then

sizeof (x)

Would give the total number of bytes the array "y" requires.

This kind of information would be useful to determine the number of record in a database. The expression

size of(y)/size of(x)

Would give the number of element in the array y.

Example program:

```
#include<stdio.h>
struct  student
                    {
int id;
char name[10];
float marks;
}s1,s2[3];
main()
                     {
printf("%d\n",sizeof(s1));
printf("%d\n",sizeof(struct student));
printf("%d\n",sizeof(s2));
printf("%d",sizeof(s2)/sizeof(struct student));
                }
```

Output:

20
20
60
3

11.8 Discuss nested structure concept:

- Structure within a structure means nesting of structures.

  Ex: struct student

```
{
  int id;
  char name[10];
  int dd;
  int mon;
  int yyyy;
}s;
```

- We can group all the items related to dob together and declare them under a substructure.

```
 Ex: struct student
 {
  int id;
 char name;
 Struct dob
  {
    int dd;
    int mon;
    int yyyy;
  }d;
}s;
```

- An inner-most member in a nested structure can be accessed by chaining all the structure variable with the member using dot operator.
- **Syntax**: outerstructure variable. innerstructure variable.datamember;

  Ex: **s. d. dd;**
  s. d. mon;
  s. d. yyyy;
- The following are valid.

  Ex: s .d(actual member is missing).
  s.dd(inner structure variable is missing).
- An inner structure can have more than one variable

```
struct student
{
 int id;
char name;
struct dob
 {
    int dd;
    int mon;
    int yyyy;
  }d1,d2;
 }s[2];
```

- Ex: A base member can be accessed as follows:

  s[1].d1.dd
  s[1].d2.dd
- We can also use tag names to define inner structure

```
 Ex: struct pay
 {
int dearness;
int house_rent;
int city;
};
 struct salary
 {
char name;
char department;
struct pay allowance;
struct pay arrears;
 };
```

- It is also permissible to nest more than one type of structures.

```
 Ex: struct personal_record
 {
Sruct  name_part name;
Sruct addr_part address;
struct date date_of_birth;
 ----------------------
----------------------
}person1;
```

Example program

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

```
#include <string.h>
struct student
{
  int id;
  char name[20];
    struct dob
    {
    int dd;
    int mm;
    int yyyy;
     }d;
}s;
main( )
{
        s.id=101;
        strcpy(s.name, "sai");
        s.d.dd=10;
        s.d.mm=11;
        s.d.yyyy=2020;
       printf( "student id : %d\n", s.id);
        printf( "student name : %s\n", s.name);
        printf( "student date of joining (dd/mm/yyyy) : %d/%d/%d\n", s.d.dd,s.d.mm,s.d.yyyy);
     }
```

```
student name : sai
student date of joining (dd/mm/yyyy) : 10/11/2020
```

11.9 Illustrate use of pointer to structure :

The name of an array stands for the address of its zeroth element. The same thing is true of the names of arrays of structure variable.

Ex: struct inventory

```
{
char name[20];
int number;
float price;
}product[2],*ptr;
```

This statement declares product as an array of two elements, each of the type struct inventory and ptr as a pointer to data objects of the type struct inventory. The assignment:

ptr=product;

Would assign the address of the zeroth element of product to ptr. Its members can be accessed using the following notation:

ptr-> name;

ptr->number;

ptr->pr ice;

The symbol -> is called the arrow operator also known as member selection operator. When a pointer ptr is incremented by one, it is made to pointer to the next record i.e.., product[1].

Ex: for(ptr=product; ptr name, ptr->number, ptr->price);

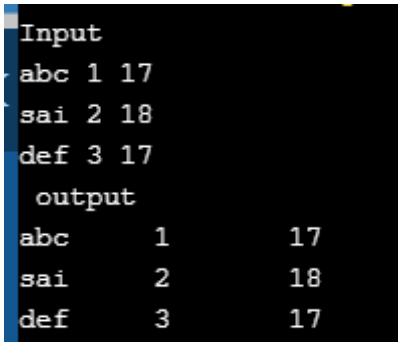
Printf("%s%d%f",ptr->name,ptr->number,ptr->price);

We could also use the pointer *(ptr).number to access the member number. The parenthesis around * ptr are necessary because the member operator has a higher precedence than the operator *.

Example program:

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

```
struct  student
{
char name[20];
int id;
int age;
};
main()
{
struct student  s[3],*ptr;
printf("Input\n");
for(ptr=s;ptr<s+3 ; ptr++)
scanf("%s%d%d",ptr->name,&ptr->id,&ptr->age);
printf("\n output \n");
for(ptr= s ; ptr< s+3 ; ptr++)
printf("%s\t%d\t%d\n",ptr->name,ptr->id,ptr->age);
}
```



11.10 Illustrate structure as function arguments and structure as function values:

C supports the passing of structure values as arguments to functions. There are methods by which the values of a structure can be transferred from one function to another.

1. The first method is a pass each member of the structure as an actual argument of the function call.
2. The (structure) second method involves passing of a copy of the entire structure to the called function. Since the function is working on a copy of the structure, any changes to structure members with in the function are not reflected in the original structure.
3. The third approach employs a concept called pointers. To pass the structure as an argument. In this case, the address location of the structure is passed to the called function .

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:

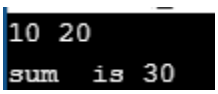
```
Data_type function_name (struct_ type st_name)
{
---------- ----------
return(expression);
}
```

Note: 1. The called function must be declared for its type, appropriate to the datatype it is expected to return.

2. The struct variable used as the actual argument and the corresponding formal argument in the in the called function must be of the same struct type.
3. The return statement is necessary only when the function is returning some data back to the calling function.
4. When a function returns a structure, it must be assigned to a structure of identical type in the calling function.
5. The called function must be declared in the calling function appropriately.

C program for finding sum of two numbers by passing individual structure members to function:

```
#include <stdio.h>
struct  addition
{
int x ,y;
```



```
};
int sum(int ,int );
main()
{
int  z;
struct  addition  a;
printf("enter two numbers\n");
scanf("%d%d",&a.x , &a.y);
   z = sum(a.x , a.y);
printf("sum  is %d\n",z );
}
int sum(int p,int q)
{
return(p+q);
}
```

C program for finding sum of two numbers by passing entire structure to function:

Program:

```
#include <stdio.h>
struct  addition
{
int x ,y;
};
int sum(struct addition );
main()
{
int  z;
struct  addition  a;
printf("enter two numbers\n");
scanf("%d%d",&a.x , &a.y);
   z = sum(a);
printf("sum  is %d\n",z );
}
int sum(struct addition a1)
{
return(a1.x+a1.y);
}
```

```
enter two numbers
10 56
sum  is 66
```

C program for finding sum of two numbers by passing address of a structure:

Program:

```
#include <stdio.h>
struct  addition
{
int x ,y;
};
int sum(struct addition *);
main()
{
struct  addition  a;
printf("enter two numbers\n");
```

```
enter two numbers
10 56
sum  is 66
```

```
scanf("%d%d",&a.x , &a.y);
 sum(&a);
printf("sum  is %d\n",a.x);
}
int sum(struct addition *a1)
{
a1->x=a1->x+a1->y;
}
```

11.11 Illustrate the structures containing arrays ,arrays of structures containing arrays

C permits the use of arrays structures members as well as array of structures that contain either single dimensional or multi dimensional arrays within the structure.

For example

```
struct marks
{
int sub[3];
int total;
}student[3];
```

Program:-

```
#include<stdio.h>
struct marks
{
int sub[3];
int total;
};
main()
{
int i,j;
struct marks student[3]={{45,56,58,0},{56,89,79,0},{78,98,85,0}};
for(i=0;i<3;i++)
{
student[i].total=0;
for(j=0;j<3;j++)
{
student[i].total=student[i].total+student[i].sub[j];
}
}
for(i=0;i<3;i++)
{
printf("total of student[%d]:%d \n",i,student[i].total);
}
}
```

```
total of student[0]:159
total of student[1]:224
total of student[2]:261
```

11.12 Illustrate concept of structures containing pointers

While using structure pointers, we should take care of the precedence of operators. The operators"->" and „." And () and [] enjoy the highest priority among the operators.

Ex: struct student

```
{
char name[20];
```

```
int id;
int age;
char  *p1;
int  *p2;
int  *p3;
};
```

C program on demonstrating structure containing pointers:

Program:

```
#include <stdio.h>
struct  student
{
char name[20];
int id;
int age;
char *p1;
int  *p2;
int  *p3;
};
main()
{
struct  student   s;
printf("enter name,id,age");
scanf("%s%d%d",s.name,&s.id,&s.age);
  s.p1=s.name;
  s.p2=&s.id;
  s.p3=&s.age;
printf("%s\t%d\t%d",s.p1,*s.p2,*s.p3);
}
```

```
enter name,id,age
sai 23 15
sai     23      15
```

11.13 Explain self-referential structures:

A self-referential structure is one of the data structures which refer to the pointer to be self-referential data structure. The node of a node is being pointed, which is of the same struct type.

A self-referential structure is used to create data structures like linked lists, stacks etc.

Syntax: struct tag_name

```
{
        datatypevariable_name;
        struct_name *pointer_name;
}
```

Example: struct node

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

In the above example, the node is a self-referential structure because the *next is of the type struct node.

Example program:

```
#include <stdio.h>
struct  node
{
int data;
```

```
10
20
30
```

```
struct node *next;
};
main()
{
   struct node node1,node2,node3;
   node1.data=10;
   node2.data=20;
   node3.data=30;
   node1.next=&node2;
   node2.next=&node3;
   node3.next=NULL;
   printf("%d\n",node1.data);
   printf("%d\n",node2.data);
   printf("%d\n",node3.data);
}
```

11.14 Define union and illustrate use of a union

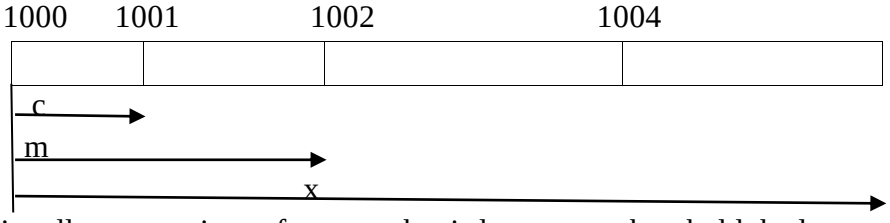
Unions are a concept borrowed from structures. In structures, each member has its own storage location, whereas all the members of a union use the same location. A union may contain many members of different types, it can handle only one member at a time. A union can be declared using the keyword union as follows:

```
union tag_name
{
        Data member;
       Member function;
 } objects;
```

Ex: union item

```
{
        int m;
        float x;
        char c;
} code;
```

This declares variable code of type union item.



The complier allocates a piece of storage that is large enough to hold the largest variable type in the union. In the above declaration, the member x requires 4 bytes which is the largest among the member.

- To access a union member, we can use the same syntax that we use of structure members.

```
code.m;
code. x;
code.c;
```

- During accessing, we should make that we are accessing the member whose value is currently stored.

```
code.m=79;
```

code.x=75.79;
printf(“%d”, code.m);

Will produce erroneous output.

- A union creates a storage location that can be used by any one of its members at a time.
- unions may be initialized when the variable is declared.
- unions may be used in all places where a structure is allowed.

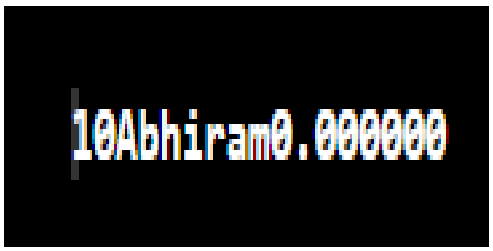
Example program:

```
#include <stdio.h>
union item
{
int a;
float b;
char ch;
};
int main( )
{
union item it;
it.a = 12;
it.b = 20.2;
it.ch = 'z';
printf("%d\n", it.a);
printf("%f\n", it.b);
printf("%c\n", it.ch);
return 0;
}
```

Output:1101109626
20.199940
Z

Another example

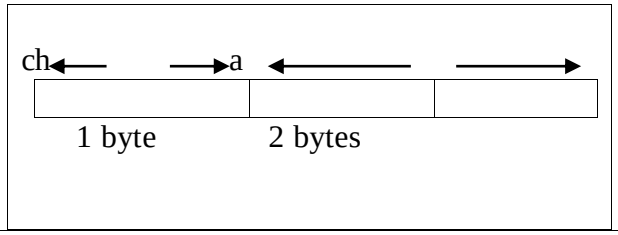
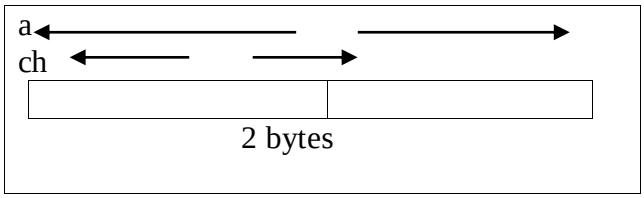
```
#include<stdio.h>
#include<string.h>
union abc
{
   int id;
   char name[10];
   float marks;
};
void main()
{
   union abc u;
   u.id=10;
   printf("%d",u.id);
   strcpy(u.name,"Abhiram");
   printf("%s",u.name);
   u.marks=0;
   printf("%f",u.marks);
}
```



11.15: Differences between structure and union

Structure	Union
1.Use 'struct' keyword.	1.Use 'union' keyword.

2.syntax:
struct tag _ name
{
data _type member1;
data _ type member2;
- - - - - - - - - - - - - -
-- - - - - - - - - - - - -
data _ type member n;
};
	2.syntax:
	union tag _ name
	{
	data _type member1;
	data _ type member2;
	- - - - - - - - - - - - - -
	-- - - - - - - - - - - - -
	data _ type member n;
	};

3.In structures, each member has its own storage location. Example:
struct mixed
{
char ch;
int a;
};
 
 	
	3.Whereas, all the members of the union use the same location. Example:
	union mixed
	{
	char ch;
	int a;
	};
	 

4.All the members of structure can be initialized.	4.Only the first member of a union can be initialized.
5.All the members of structure can be accessed at any time.	5.Only one member of a union can be accessed at any time.
6.Each member within a structure is assigned to unique storage area of location.	6.memory allocated is shared by individual members of a union.
7.Altering the value of a member will not effect the other members of a structure.	7.Altering the value of a member will alter other members of a union.