Unit –V
STRUCTURES

Introduction: We have seen that arrays can be used to represent a group of data items that belong to the same type, such as int or float. However, we cannot use an arrays if we want to represent a collection of data items of different types using a single name. C++ support a special data type known as structure.

Definition: A structure is a convenient tool for handling a group of logically data items

Or

A structure is a collection of different data type that are stored under single name.

Defining a structure: Unlike arrays, structures must be defined first that may be used later to declare structure variables. Let us use an example the process of structure definition. Consider a book data base consisting of book name, author, no. of pages and price. We can define a structure to hold this information as follows.

```
Struct book
{
        Char title[20];
        Char auther[15];
        int pages;
        float price;
}
```

The keyword struct declare a structure.

The general format of structure definition is as follows.

```
Struct structure name
{
        Data type member1;
        Data type member2;
        .               .
        .               .
}
```

Declaring structure variables:A structure variable declaration is similar to the declaration of variable of any other datatypes.

```
Struct book
{
        Char title[20];
        Char auther[15[;
        Int pages;
        Float price;
}s
truct book, book1, book2, book3;
```

In c we can combine both the structure definition and variables as follows.

```
Struct book
{
        Char title[20];
        Char auther[15[;
        Int pages;
        Float price;
}struct book, book1, book2, book3;
```

Structure initialization: Like any other datatype a structure variable can be initialized at compile time

```
Main ( )
{
        Struct stud
        {
                Int weight;
                Float height;
        }student={60,183.5}
        Struct book
        {
                Char title[20];
                Char author[15];
                Int pages;
                Float price;
        }book1={C++,balaguru,512,112}
}
```

The following statement initialize 2 structure variable

```
Main ( )
{
    Struct stud
    {
        Int weight;
        Float height;
    }struct stud S1={60,173.2};
     Struct stud S2={80,183.5};
```

Accessing structure members:

We can access and assign values to the members of a structure in a number of ways. They should be linked to the structure variables. The link between a member and a variable is established using the member operator, which is also known as dot operator.

For example: book1.price;
Book1.pages;
Book1.price=50

We can also use scanf to give the values

```
Scanf("%s",&book1.title);
Scanf("%s",&book1.price);
```

Unions

Unions are next implementation of Structure. Therefore it follows the same syntax is Structure. However this is a major difference between them in term of storage.

In structures, each member has its own storage location, where as all the members of a Union use the same location.

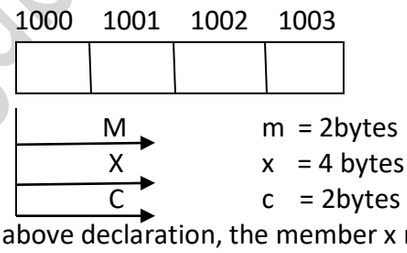
Declaration: Like structures a Union can be declared using the keyword union can be as follows.

```
Union union name
{
    Datatype member
    Datatype member
    .          .
    .          .
}
```

Example:

```
Union item
{
    Int m;
    Float x;
    Char c;
}code;
```

This declares a variable code of type union item. The union contains 3 members each with defferent datatypes. However we can use only one of them at a time. This is due to the fact that only one location is allocated of a union variable.



In the above declaration, the member x requires 4 bytes which is the largest among the members. The 3 variable share same memory.

Initialization of Unions: Unions maybe initialize when the variable is declared. But unlike structures, it can be initialized only with a value of same type as the first union member.

```
Union item abc={100};//valid
Union item abc=(10.75}, is invalid. This is because the type of first member is int.
```

Ex: code.m=379;
Code.x=7859.36;

Accessing the members of a union: To access a union member we can use the same syntax that we use for the structure member. That is by using dot operator to make a link between union members and variable.

Ex: code.m
Code.x
By using scanf & print f function

```
Scanf("%d",&code.m);
Scanf("d",&code.x);
Printf("%d",code.m);
```

Difference between Structures and Unions?

Structure	Union
1. We can access all the members of structure at any time. 2. Memory is allocated for all variable. 3. All the members of structure can be initialized. 4. Struct keyword is used to creat a structure. 5. Syntax: Struct tagname { Datatype member . . . . }s1,s2;	1. we can access only one member at a time 2. Memory is allocated only for the largest data. 3. Only the first member of a union can be initialized. 4. Union keyword is used to created a union. 5. Syntax: Union tagname { Data type member . . . . } u1,u2;

Structures using Arrays: We use structure to describe the format of an array variables. For example in the marks obtained by a class of students,we may use a format to describe marks obtained in variable subjects and then declares all the students as structure variables. In such occurs we may declares on array of structures.

Struct structure name student[100];

Defines an array called students that contains of 100 elements. Each element is defined to be of the type struct calss, consider the following

```
Struct marks
{
        Int s1,s2,s3;
}
Main ( )
{
        Struct marks student[3]={(46,58,81),(75,53,69),(57,36,71)};
```

This declares the student as any array of these elements students[0],student[1] and student[2] and initializes their members as follows.

```
Student[0].s1=46
Student[1].s2=58
Student[0].s3=81
        .       .
        .       .
```

An array of structure is stored inside the memory in the same way as multi dimensional array as follows.

Enumerated Data Type

An enumerated data type is a user defined data type which provides a way for attaching numbers to names. The "*enum"* key word is used for creating Enumerated Data Type. In this a list of words by assigning with numbers 0,1,2,3…..

The basic syntax of Enum statement is as follows.

```
Syn:    enum enum name
        {
        List of enumerators
        }
Ex:     enum day
        {
                Sun, mon, tue, wed, thu, fri, sat
        }
Ex:     enum shape
        {
                Circle, square, rectangle, triangle
        }
```

In C, the enumerated name i.e., day and shape becomes new datatypes, by using this we can declares new variables.

Ex: shape triangle.

By default the enumerators are assigned with integer values starting with zero. For first enumerators , one, for second enumerator, two and so on we can over ride the default integer value to the enumerators.

Ex: enum day
{
Sun=1,mon=2,tue=3,wed=4,thu=5,fri=6,sat=7;
}

Let us consider a C Program to create Edt.

```
#include<stido.h>
Enum day
{
 Sun,mon,tue,wed,thu,fri,sat;
}
Void main( )
{
 int ch;
 printf(“enter a value”);
 scanf(“%d”,&ch);
switch(ch)
{
 case 0: printf(“sunday”);
          break;
 case1:printf(“monday”);
          break;
 case2:printf(“tuesday”);
          break;
 case3:printf(“wednesday”);
          break;
 case4:printf(“thursday”);
          break;
 case5:printf(“friday”);
          break;
 case6:printf(“saturday”);
          break;
 default:printf(“please pass 0-6 only”)
}
```

Pointers and its features?

A pointer is a derived data types in C. It is build from one the fundamental datatype available in C. A pointer is a variable that contains memory address as their values.

Features / Advantages of pointers:

- A pointer is more efficient in handling arrays & tables.
- Pointers can be used to return multiple values from a function via function arguments.
- Pointers permit reference to function as arguments to other.
- The use of pointers arrays to character array result in saving of data storage space memory.
- Pointers allow C to supports dynamic memory management.
- Pointers provide tools for manipulating dynamic data structures such as structures, linked list.
- Pointers reduce length of a program.
- They increase execution speed.

Write about pointers?

Introduction: The computer memory is a sequential collection of storage cells as shown.

Memory Cell	Address
	0
	1
	2
	3
	4
	5
	.
	.

Each cell commonly known as byte, has number called address with it.

Consider the following statement.

Int quantity=179;

Let us assume that the system has chosen the address location 5000 for quantity and value 179.

Quantity	Variable
179	Value
5000	Address

Suppose we assign the address of quantity to a pointer variable P. the link between P and quantity.

Variable	Value	Address
Quantity	179	5000
P	5000	5048

Declaring Pointer Variables: In C, every variable must be declared since pointer variables contains address that belong to a separate datatype, they must be declared as pointers before we use them.

Syntax: datatype *pointer name

Ex: int *P;

Declares the variable P as pointer variable that points to an integer datatype.

Initialization of pointer variable: The process of assigning the address of a variable to a pointer variables is known as initialization. We can use assignment operator to initialize the variable.

Ex: int quantity;
Int *p;
P=&quantity;

We can also define a pointer variable with an initial value of Null or 0. That is the following statements are

Int *p=Null;
Int *p=0;

Accessing variable using pointers: Once the pointer has been assigned the address of a variable, the question is how to access the value of variable using pointer? Its done by using another unary operator * known as indirection operator another name is dereferencing operator.

```
Int quantity, *p, n;
Quantity = 179;
P=&quantity;
N=*p;
```

Variable	Value	Address
Quantity	179	5000
P	5000	5004
N	179	5008

Write a C Program to accept student details using structure?

```
#include<stdio.h>
struct student
{
        int sno;
        char sname[20];
        char course[10];
        float marks;
}s1;
void main( )
{
        printf("enter student number");
        scanf("%d",&s1.sno);
        printf("enter student name");
        scanf("%c",&s1.sname);
        printf("enter course");
        scanf("%c",&s1.course);
        printf("enter marks");
        scanf("%f",&s1.marks);
        printf("student details");
        printf("student no.%d", sl.no);
        printf("student name:%c", s1.name);
        printf("student course:%c",s1.course);
        printf("student marks:%f",s1.marks);
}
```

Write a C program to accept employee details using Union.

```
#include<stdio.h>
union employee
{
        int eno;
        char ename[20];
        char ejob;
        float salary;
}u1;
void main( )
{
        printf("enter employee number");
        scanf("%d",&u1.eno);
        printf("enter employee name");
        scanf("%c",&u1.ename);
        printf("enter employee job");
        scanf("%c",&u1.ejob);
        printf("enter employee salary");
        scanf("%f",&u1.esalary);
        printf("employee details");
        printf("employee no:%d",u1.no);
        printf("employee name:%c", u1.name);
        printf("employee job:%c", u1.job);
        printf("employee salary:%f",u1.salary);
}
```

Structures and functions

C supports passing of structure values as arguments to functions. There are 3 methods by which the values of structures can be transferred from one function to another.

Method – I: The first method is to pass each member of the structures as an actual argument of the function call. The actual arguments are then treated independently like ordinary variables.

Method – II: The second method involves passing of copy of the entire structure to the called function. Since the function is working on a copy of the structure any changes to structure member within the functions are not reflected in the original structure.

Method – III: The third approach employs a concept called pointers to pass structures as an arguments. In this case, the address location of the structure is passed to called function. This function can access indirectly the entire structure and work on it.

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

```
Datatype function-name(struct type struct name)
{
        ……..
        ……..
        Return;
}
```

Pointer expressions:

Pointer can be used in expressions for example P1 and P2 are declared and initialized pointer, then the following statements

```
Y=*p1*p2;
Sum=sum+*p1;
Z=5**p2;
```

C allows us to add integers to or subtract integer from pointers as well as to subtract one pointer from another. P1+4, p2-2 and p1-p2 are allowed.

We may also use short hand operators with the pointers.

```
P1++;
-p2;
Sum+=*p2;
```

Pointers and Arrays: Suppose we declares an array X as follows

```
Int x[5]={1,2,3,4,5};
```

Suppose the base address of x is 1000 and assuming that each integer requires 2 bytes, the 5 elements will be stored as follows.

Element → x[0]

The name x is defined as a constant pointer pointing to first element, x[0]and therefore the value of x is 1000 the location where x[0] is stored. That is

X=&*[0]=1000

If we declare P as an integer pointer, than we can make the pointer P to point to array x as the following.

P=x; p=&x[0]

Now we can access every value of x using P++ to move from one element to another.

P=&x[0]
P+1=&x[1]
P+2=x[2]
P+3=x[3]

We can also use pointers for characters array.

Ex: char *str= “good”

This creates a string for the literal and then stores it address in the pointer variable str the pointer str now pointer to the first character of the string “good” as

Pointers and Structure: We know that the name of an array stands for the address of its zero th element the same thing is true of the name of array of structure variables.

Suppose product is an array variable of struct type the name product represents the address of zero th element.

Consider the following declaration

```
Struct inventory
{
        Char   name[20];
        Int      number;
        Float   price;
}product[2], *p;
```

That is pointer P will now points to product [0]. It numbers can be assumed using the symbol → is called arrow operators and made up of minus sign and greater than sign

P → name
P → number
P → price

Pointers and functions: we have seen earlier that when array is passed to a function as an argument only the address of the first element of the array is passed but not actual value of the array element. It x is an array when we call sort(x) the address of x[0] is passed the function sort.

When we pass address to a function the parameter receiving the addresses should be pointers. The process of calling of a function using pointers to pass from address of variable is know “call by reference” the function which is called by reference can change the value of variable used in the call.

Consider the following code

```
Main( )
{
        Int x;
        X=20;
        Sort(&x);
        Printf(“%d”,x);
}
 Sort(int *p)
{
  *p=*p+10;
}
```

When the function sort( ) is called, the address of the variable x, not its value is passed into the sort( ), inside the sort( ) the variable P is declared as a pointer and therefore P is the address of the variable x.

*P=*P+10;

Means add 10 to the value stored at the address P.

Write a C Program to print the address of a variable along with its value?

```
#include<stdio.h>
main ( )
{
        char a=c;
        int x=10;
```

```
        float p=120.5;
        printf("%c is stored at %u /n", a, &a);
        printf("%d is stored at %u /n",x, &x);
        printf("%f is stored at %u /n", p, &p);
}
```

String Handling Functions

The C library supports a large number of strings handling functions that can be used to carry out many of the string manipulations.

Function	Meaning
Strcat( )	concatenates two strings
Strcmp( )	compares two strings
Strcpy( )	copies one string over another
Strlen( )	finds length of a string
Strrev( )	reverse is given string

Strcat(): The strcat function joins two strings together. It takes the following form.

Strcat(string 1, string 2);

String 1 and string 2 are character arrays.

When the function strcat is executed, string 2 is added to string 1.

For example consider two strings

String 1

0	1	2	3	4
V	E	R	Y	\0

String 2

0	1	2	3	4
G	O	O	D	\0

Execution of the statement strcat(string1, string 2)

We result as

String 1 =

0	1	2	3	4	5	6	7	8	9
V	E	R	Y		G	O	O	D	\0

String 2 =

G	O	O	D	\0

Write a C program to concatenate of strings

```
#include<stdio.h>
Void main( )
{
 Char s1[30], s2[30];
 Printf("enter string1");
 Gets(S1);
 Printf("enter string2");
 Gets(s2);
 Printf("the concatenated string%S",S1);
}
```

Strcmp( ): The strcmp function compares two strings and has a value '0' if they are equal. If they are not, it has the numeric difference. The general form is

Strcmp(string 1, string 2);

Ex: Strcmp(name1, john);

Strcmp("their","there");

Will return a value of +ve value or –ve value.

Write to compare two strings.

```
#include<stdio.h>
Void main ()
{
        Char s1[30],s2[30];
        Printf("enter first string");
        Gets(s1);
        Printf("enter second string");
        Gets(s2);
        If(strcmp(s1,s2)==0);
        {
                Printf("string are equal");
                Else
                Printf("string are not equal");
        }
 }
```

Strcpy( ): the strcpy function is used to copy one string to another string. It takes the form

Strcpy(String1,string2);

The string 2 is assigned to string 1.

Write a C Program to copy of two strings using strcpy()

```
#include<stdio.h>
Void main( )
 {
        Char s1[30],s2[30];
        Printf("enter a string");
        Gets(s1);
        Strcpy(s2,s1);
        Printf("the copied string is %s",s2);
 }
```

Strlen( ): this function counts and return the number of characters in a string. It takes the form

N=strlen(string);

Where n is a integer variable which stores the length.

Write a C program to find the length of string

```
#include<stdio.h>
main( )
{
        char str[50];
        printf("enter a string);
        gets(str);
        printf("the length of the string%d",strlen(str));
}
```

Strrev( ): this function reverse the character in a string. It takes the form

Str1=strrev(Str);

Where str1 is a character array and it stores the reverse string of the given string.

Write a C program to check whether the given string is palindrome or not.

```
#include<stdio.h>
void main( )
{
        char s1[30],s2[30];
        printf("enter a string");
        getch(s1);
        strrev(s1,s2);
        if(strrev(s1,s2)==0);
        printf("given string is palindrome");
        else
        printf("not palindrome");
}
```

Strings

A string is a sequence of characters. (Or) A sequence of characters that are enclosed in between single or double quotation marks is known as "string".

Ex: char a[5]= "hello"

Declaring and Initializing strings: C doesn't support strings as a datatype. However it allows us to represents string by using characters arrays.

The general form of declaring a string is as follows.

Syntax: char string-name[size];

In this above syntax size specifies the number of characters in a string.

Ex: char sname[20];

Char ename[15];

The compiler assigns a null character "\0" automatically at the end of the string. Therefore the should be equal to maximum number of characters in the string plus one.

Ex: char a[5]= "hello";

H	E	L	L	O	\0
0	1	2	3	4	5

In the above example the size of the string is 5+1=6

In C language we can also initialize a string without specifying size of character array. In this situation size is automatically determined base on the string.

a) Reading a string: we can read a string in two ways.

i. Using scanf( ) function
ii. Using getchar() and gets() function.

i) **By using scanf( ) function**: we use scanf() function to read a string with "%s" format specification.

Ex: char sname[5];
Scanf ("%s",&sname);
The problem with scanf() function is while giving input if we give space the function terminates.

ii) **By using getchar( ) and gets( ) function**: the getchar( ) function is used to read character by character i.e., only one character at a time.

Ex: char ch;
Ch=getchar( );
Another function to read a string is gets( ) function. This function reads one string at a time.

Ex: char ch;
Gets(ch);

b) **Writing a string**: we can write a string in two ways.
i) By using printf( ) function
ii) By using putchar( ) and puts( ) functions.

i) **By using printf( ) function**: we use printf() to write a string with "%s" format specification.

Ex: char a[5]= "hello";
Printf("%s",a);

ii) **By using putchar( )and puts( ) functions**: the putchar( )function is used to print character by charcter i.e., only one.

EX: char ch[5]= "hello";
Put char(ch);
Another function to write a string is puts( ) function. This function writes one string at a time.

Ex: char ch[5]= "hello"
Puts(ch);