Module-5
Structure, Union, Pointers and Preprocessor Directives

Structure: - Structure is a collection of one or more variables of same or different data types grouped to gather under a single name for easy handling.
Structure is a user defined data type that can store related information about an entity.
Structure is nothing but records about a particular entity.

Declaration of a Structure :- A structure is declared using the keyword struct followed by structure name and variables are declared within a structure
The structure is usually declared before the main( ) function.
Syntax : -

```
struct structure_name
{
        datatype member 1;
        datatype member 2;
        datatype member 3;
        ……..
        ……..
        datatype member n;
};
```

Example :-

```
struct employee
{
        int emp_no;
        char name[20];
        int age;
        float emp_sal;
};
```

Initialization of structure :- A structure initialization is done after the declaration of the structure
Syntax :-
struct structure_name structure_variable _name;
Example :-
struct employee emp1;

Accessing structure members :- A structure uses a **.**(dot) [Member Access Operator] to access any member of the structure.
Example :-
emp1.emp_no =12345;

Initialization of structure variables :-

```
struct employee
        {
                int emp_no;
                char name[20];
                int age;
                float emp_sal;
        } emp1={65421, “Hari”,29,25000.00};
```

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

/* C program to read and print the details of employee using structures –static allocation of variables */

```
#include<stdio.h>
#include<string.h>
struct employee
{
   int emp_no;
   char empname[20];
   int age;
   float emp_sal;
};
void main()
{
   struct employee emp1;
   emp1.emp_no = 65421;
   strcpy(emp1.empname,"Hari");
   emp1.age=29;
   emp1.emp_sal=25000.00;
   printf("Employee Number=%d\n",emp1.emp_no);
   printf("Employee Name=%s\n",emp1.empname);
   printf("Employee Age=%d\n",emp1.age);
   printf("Employee Salary=%f\n",emp1.emp_sal);
}
```

/* C program to read and print the details of employee using structures –run time allocation of variables */

```
#include<stdio.h>
#include<string.h>
struct employee
{
   int emp_no;
   char empname[20];
   int age;
   float emp_sal;
};
```

```
void main()
{
   struct employee emp1;
   printf("Enter Employee Number:");
   scanf("%d",&emp1.emp_no);
   printf("Enter Employee Name:");
   scanf("%s",emp1.empname);
   printf("Enter Employee age:");
   scanf("%d",&emp1.age);
   printf("Enter Employee salary:");
   scanf("%f",&emp1.emp_sal);
   printf("Employee Number=%d\n",emp1.emp_no);
   printf("Employee Name=%s\n",emp1.empname);
   printf("Employee Age=%d\n",emp1.age);
   printf("Employee Salary=%f\n",emp1.emp_sal);
}
```

/* C Program to find Sum of Two Complex Numbers*/

```
#include<stdio.h>
struct complex
{
        int real;
        int img;
};

void main()
{
        struct complex a,b,c;
        printf("Enter the First Complex Number\n");
        printf("Enter the real part:");
        scanf("%d",&a.real);
        printf("Enter the imaginary part:");
        scanf("%d",&a.img);
        printf("Enter the Second Complex Number\n");
        printf("Enter the real part:");
        scanf("%d",&b.real);
        printf("Enter the imaginary part:");
        scanf("%d",&b.img);
        c.real=a.real+b.real;
        c.img=a.img+b.img;
        if(c.img>=0)
                printf("Sum of two Complex Number = %d+i%d\n",c.real,c.img);
        else
                printf("Sum of two Complex Number = %d %di\n",c.real,c.img);
}
```

Array of Structures :- if we want to store the data of 100 employees we would require 100 structure variables from emp1 to emp100 which is definitely impractical the better approach would be to use the array of structures.

/* C program to illustrate array of structures */

```
#include<stdio.h>
#include<string.h>
struct employee
{
   int emp_no;
   char empname[20];
   int age;
   float emp_sal;
};
void main()
{
   struct employee emp1[20];
   int n,i;
   printf("Enter the number of employee entries\n");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
            printf("Enter the details of employee %d\n",i+1);
            printf("Enter Employee Number:");
            scanf("%d",&emp1[i].emp_no);
            printf("Enter Employee Name:");
            scanf("%s",emp1[i].empname);
            printf("Enter Employee age:");
            scanf("%d",&emp1[i].age);
            printf("Enter Employee salary:");
            scanf("%f",&emp1[i].emp_sal);
   }
printf("\nEMP_NO\t EMP_NAME\t EMP_AGE\t\t EMP_SALARY \n");
   for(i=0;i<n;i++)
   {
        printf("%d\t%s\t%d\t%f\n",emp1[i].emp_no,emp1[i].empname,emp1[i].age,emp1[i].e
        mp_sal);
   }
}
```

Nested Structures :- A nested structure is a structure that contains another structure as its member.

Syntax :-

```
struct structure_name1
        {
                datatype member 1;
                datatype member 2;
        };
struct structure_name2
        {
                datatype member 1;
                datatype member 2;
                struct structure_name1 var1;
        };
```

/* C program to demonstrate nested structures */

```
#include<stdio.h>
struct stud_dob
{
   int day;
   int month;
   int year;
};

struct student
{
   int rollno;
   char sname[20];
   struct stud_dob date;
};

void main()
{
   struct student s;
   printf("Enter Student Rollno :");
   scanf("%d",&s.rollno);
   printf("Enter Student Name :");
   scanf("%s",s.sname);
   printf("Enter date of birth as day month year:");
   scanf("%d%d%d",&s.date.day,&s.date.month,&s.date.year);
   printf("Student Details are\n");
   printf("Student Rollno = %d\n",s.rollno);
   printf("Student Name=%s",s.sname);
   printf("Student DOB= %d-%d-%d\n", s.date.day, s.date.month,  s.date.year);
}
```

Structures and Functions :- Structures can be passed to functions and returned from it. Passing structures to functions can be done in the following ways

- Passing individual members
- Passing entire structure or structure variable.

Passing Individual Members :-
while invoking the function from main( ) in the place of actual arguments we can pass the structure member as an argument.

```
#include<stdio.h>
void add(int x,int y);
struct addition
{
   int a;
   int b;
};
void main()
{
   struct addition sum;
   printf(" Enter two numbers\n");
   scanf("%d%d",&sum.a,&sum.b);
   add(sum.a,sum.b);
}
void add(int x,int y)
{
   int res;
   res=x+y;
   printf("Resultant=%d\n",res);
}
```

Passing Entire structure or structure variable_:-while invoking the function instead of passing the individual members the entire structure, i.e. the structure variable is passed as a parameter to the function.

```
#include<stdio.h>
struct addition
{
   int a;
   int b;
};
void add(struct addition sum);

void main()
{
   struct addition sum;
   printf(" Enter two numbers\n");
   scanf("%d%d",&sum.a,&sum.b);
   add(sum);
}
```

```
void add(struct addition sum)
{
   int res;
   res=sum.a+sum.b;
   printf("Resultant=%d\n",res);
}
```

Union:- A union is a special data type available in C that allows 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 multiple-purpose.

Syntax for declaring a union is same as that of declaring a structure except the keyword struct.

```
union union_name {
   datatype field_name;
   datatype field_name;
   // more variables
};
```

Difference between Structure and Union

	Structure	Union
Keyword	The Keyword struct is used to define the Structure	The keyword union is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is greater than or equal to the sum of sizes of its members.	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of union is equal to the size of largest member.
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Pointers

Pointer: - A pointer is a variable which holds address of another variable or a memory-location.

Advantages of Pointers:

- Pointers are more efficient in handling arrays and data tables.
- Pointers can be used to return multiple values from a function.
- Pointers allow 'C' to support dynamic memory management.
- Pointers provide when efficient tool for manipulating dynamic data structures such as stack, queue etc.
- Pointers reduce length and complexity of programs.
- They increase execution speed and this reduces program execution time

Declaration and Initialization of pointers :- The operators used to represent pointers are

- Address Operator (&)
- Indirection Operator (*)

Syntax :-

```
ptr_data_type *ptr_var_name;
ptr_var_name = &var_name;
```

- where var_name is a variable whose address is to be stored in the pointer.

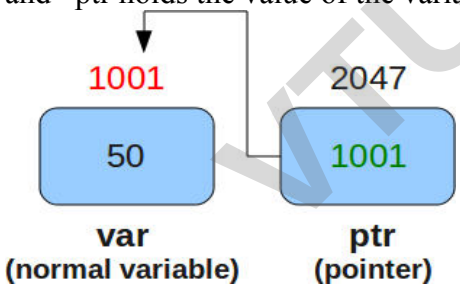
Example :-

```
int a=10;
int *ptr;
```

then

```
ptr = &a;
*ptr = a;
```

Here ptr is a pointer holding the address of variable 'a'
and *ptr holds the value of the variable a.




```
#include<stdio.h>
void main()
{
        int *ptr;
        int x=22;
        printf("Addres of x= %d\n",&x);
        printf("Content of x= %d\n",x);
        ptr=&x;
        printf("Addres of pointer= %d\n",ptr);
        printf("Content of pointer= %d\n",*ptr);
```

}

Pointers and Arrays :-

Consider an array: int arr[4];

Figure: Array as Pointer

/* c program to demonstrate arrays with pointers */

```
#include<stdio.h>
void main()
{
        int a[10]={11,13,15,17};
        int *ptr;
        int i;
        ptr=a;
        for(i=0;i<4;i++)
        {
        printf("%d\t",a[i]);
        printf("%d\n",&a[i]);
        printf("%d\t",*ptr);
        printf("%d\n",ptr);
        ptr++;
        }
}
```

Double Pointer: When a pointer holds the address of another pointer then such type of pointer is known as **pointer-to-pointer** or **double pointer**.

- Here the first pointer is used to store the address of the variable
- The second pointer is used to store the address of the first pointer.

Declaration of double pointer :

Syntax:

```
datatype **ptr;
```

the declaration is similar to that of the pointer the only difference is we place an additional *before the name of the pointer.

```
#include<stdio.h>
void main()
{
        int var=777;
        int *ptr2;
        int ** ptr1;
        ptr2=&var;
        ptr1=&ptr2;
        printf("value of var=%d\n",var);
        printf("value of var using single pointer=%d\n",*ptr2);
        printf("value of var using double pointer=%d\n",**ptr1);
}
```

OUTPUT
value of var=777
value of var using single pointer=777
value of var using double pointer=777

Preprocessor Directives

Preprocessor Directives:-Preprocessor is a program which is invoked by the compiler before the compilation of the user written program

–The declaration of preprocessor statements always begin with # symbol
–These statements are usually placed before the main( ) function.

Types of Preprocessors:-

–Macros
–File Inclusion
–Conditional compilation
–Other directives

Macros:-These are the piece of code in the program which is given some name.

• Whenever the name is encountered by the compiler in the program , the compiler replaces the name with the actual piece of code.
• The **"#define"** directive is used to define a macro.

/* C program to demonstrate macros */

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

```
# define pi 3.142
void main()
{
        float r,area;
        printf("Enter the value of radius\n");
        scanf("%f",&r);
        area=pi*r*r;
        printf("Area of circle = %f\n",area);
}
```

/*C Program that finds the addition of two squared numbers, by defining macro for Square(x).*/

```
 #include <stdio.h>
#define square(x) (x*x)
void main( )
 {
        int n1, n2,  sum;
        printf("Enter two numbers");
        scanf("%d%d", &n1, &n2);
        sum = square(n1) + square(n2);
        printf("Sum of two squared numbers = %d", sum);
}
```

File Inclusion:-This type of preprocessor directive tells the compiler to include a file in the source code program
These files contains the definition of the predefined functions like printf( ), scanf( ), etc. Different functions are declared in different header files.

/* C program to demonstrate file inclusion */

```
#include<stdio.h>
#include<math.h>
void main()
{
        int num;
        float res;
        printf("Enter a number\n");
        scanf("%d",&num);
        res=sqrt(num);
        printf("Squareroot of %d = %f\n",num,res);
}
```

Conditional Compilation:-these are the type of directives that helps to compile a specific portion of the program or to skip a specific part of the program based on some conditions. Some of the conditional compilation directive are

- #ifdef, #ifndef, #if, #else

/* C program to demonstrate use of #ifdef*/

```
#include<stdio.h>
#define age 18
void main()
{
        #ifdef age
                printf("The person is eligible to vote");
        #endif
}
```

/* C program to demonstrate use of #else*/

```
#include<stdio.h>
#define age 21
void main()
{
        #ifdefage<=18
        printf("The person is not eligible to vote");
        #else
        printf("The person is eligible to vote");
        #endif
}
```

Other Directives:-Apart from the above directive there are two more directives which are not commonly used

#undef directive:-this directive is used to undefined a value which was declare using the #define directive

#pragma directive:-this directive is used in parallel programming (thread programming) which is a major concept of operating system.