

Unit – V

Structures, Unions, Bit Fields: Introduction, Nested Structures, Arrays of Structures, Structures and Functions, Self-Referential Structures, Unions, Enumerated Data Type – enum variables, Using Typedef keyword, Bit Fields. Data Files: Introduction to Files, Using Files in C, Reading from TextFiles, Writing to Text Files, Random File Access.

Structure

A structure is a user defined data type. We know 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 array if we want to represent a collection of data items of different types using a single name. A structure is a convenient tool for handling a group of logically related data items.

Structure is a user defined data type used to represent a group of data items of different types using a single name.

The syntax of structure declaration

```
isstruct structure_name
```

```
{  
    type element 1;  
    type element 2;  
    .....  
    type element n;  
};
```

In structure declaration the keyword **struct** appears first, this followed by structure name. The member of structure should be enclosed between a pair of braces and it defines one by one each ending with a semicolon. It can also be array of structure. There is an enclosing brace at the end of declaration and it end with a semicolon.

We can **declare structure variables** as follows

```
struct structure_name var1,var2,.....,var n;
```

Example:

To store the names, roll number and total mark of a student you can declare 3 variables. To store this data for more than one student 3 separate arrays may be declared. Another choice is to make a structure. No memory is allocated when a structure is declared. It just defines the “form” of the structure. When a variable is made then memory is allocated. This is equivalent to saying that there's no memory for “int”, but when we declare an integer that is `int var;` only then memory is allocated. The structure for the above- mentioned case will look like

```
struct student
{
    int rollno;
    char name[25];
    float totalmark;
};
```

We can now declare structure variables stud1, stud2 as follows

```
struct student stud1,stud2;
```

Thus, the stud1 and stud2 are structure variables of type student. The above structure can hold information of 2 students.

It is possible to combine the declaration of structure combination with that of the structure variables, as shown below.

```
struct structure_name  
  
    {  
  
        type element 1;  
  
        type element 2;  
  
        .....  
  
        type element n;  
  
    } var1,var2,...,varn;
```

The following single declaration is equivalent to the two declaration presented in the previous example.

```
struct student  
  
    {  
  
        int rollno;  
  
        char name[25];  
        float totalmark;  
  
    } stud1, stud2;
```

Accessing structure Variable

The different variable types stored in a structure are called its members. The structure member can be accessed by using a **dot (.) operator**, so the dot operator is known as **structure member operator**.

Example:

In the above example stud1 is a structure variable of type student. To access the member name, we would write stud1.name. Similarly, stud1's rollno and stud1's totalmark can be accessed by writing stud1.rollno and stud1.totalmark respectively.

Initializing Structure Members

Structure members can be initialized at declaration. This much the same manner as the element of an array; the initial value must appear in the order in which they will be assigned to their corresponding structure members, enclosed in braces and separated by commas.

The **general form** is

```
struct structure_name var={val1,val2,val3.....};
```

Example:

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

```

#include<conio.h>
void main()

{

    struct student

    {

        char *name;
        int rollno;

        float totalmark;

    };

    struct student stud1={"Venkat",1,98};
    struct student stud3= {"Shweta",3,97};
    struct student stud2={"Arpita",2,99};
    clrscr();

    printf("STUDENTS DETAILS:\n");

    printf("\n\n Roll number:%d\n Name:%s\n Total Marks:%f", stud1.rollno, stud1.name,
    stud1.totalmark);

    printf("\n\n Roll number:%d\n Name:%s\n Total Marks:%f", stud2.rollno, stud2.name,
    stud2.totalmark);

    printf("\n\n Roll number:%d\n Name:%s\n Total Marks:%f", stud3.rollno, stud3.name,
    stud3.totalmark);

    getch();

}

```

Output

STUDENTS DETAILS:

Roll number: 1
Name: Venkat

Total Marks:98.000000

Roll number: 2
Name: Arpita

Total Marks:99.000000

Roll number: 2

Name:Shweta

Total Marks:99.000000

Array of structures:

It is possible to store a structure as an array element. i.e., an array in which each element is a structure. Just as arrays of any basic type of variable are allowed, so are arrays of a given type of structure. Although a structure contains many different types, the compiler never gets to know this information because it is hidden away inside a sealed structure capsule, so it can believe that all the elements in the array have the same type, even though that type is itself made up of lots of different types.

The declaration statement is given below.

```

struct struct_name

{

    type element 1;

    type element 2;

    .....

    type element n;

    }array name[size];

```

Example:

struct student

```

{

    int rollno;

    char name[25];
    float totalmark;

} stud[100];

```

In this declaration stud is a 100-element array of structures. Hence, each element of stud is a separate structure of type student. An array of structure can be assigned initial values just as any other array. So the above structure can hold information of 100 students.

Program to demonstrate use of array of structure

```

#include    <stdio.h>
#include    <conio.h>
void main()

{

struct student

{

    int rollno;

    char name[25];
    int totalmark;

```

```
}stud[100];
```

```
int n,i;  
clrscr();
```

```
printf("Enter total number of students\n\n");  
scanf("%d",&n);
```

```
for(i=0;i<n;i++)
```

```
{
```

```
    printf("Enter details of %d-th student\n",i+1);  
    printf("Name:\n");  
    scanf("%s",&stud[i].name);
```

```
    printf("Roll number:\n");  
    scanf("%d",&stud[i].rollno);  
    printf("Total mark:\n");  
    scanf("%d",&stud[i].totalmark);
```

```
}
```



```
printf("STUDENTS DETAILS:\n");  
  
for(i=0;i<n;i++)  
{  
  
    printf("\nRoll number:%d\n",stud[i].rollno);  
    printf("Name:%s\n",stud[i].name);  
    printf("Total mark:%d\n",stud[i].totalmark);  
  
}  
  
getch();  
}
```

OUTPUT

Enter total number of students:

3

Enter details of 1-th student

Name:SUBAHAS

Roll number:11

Total mark:589

Enter details of 2-th student

Name:RUKSANA

Roll number:12

Total mark:594

Enter details of 3-th student

Name:SANA

Roll number:13

Total mark:595

Roll number:13

Name: SANA

Total mark:595

Structure as structure member (Embedded structure):

A structure inside another structure is called an **embedded structure**. A structure can have one or more of its member as another structure, but a structure cannot remember to itself when a structure is used as structure member. In such situation, the

declaration of the embedded structure must appear before the declaration of the outer structure. For example

```
#include <stdio.h>
#include <conio.h>
void main()
{
    struct dob
    {
        int day;
        int month;
        int year;
    };

    struct student
    {
        struct dob d;
        int rollno;

        char name[25];
        int totalmark;
    }stud[25];

    int n,i;
    clrscr();

    printf("Enter total number of students");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("\n\nEnter details of %d student",i+1);
        printf("\nName:");
        scanf("%s",&stud[i].name);
```

```
printf("\nRoll number:");
scanf("%d",&stud[i].rollno);
printf("\nTotal mark:");
scanf("%d",&stud[i].totalmark);

printf("\nDate of birth (Format:01 06 2010):");
scanf("%d%d%d",&stud[i].d.day,&stud[i].d.month,&stud[i].d.year);

}
```

```
printf("\nSTUDENTS
DETAILS:\n");for(i=0;i<n;i++)

{
```

```

        printf("\nRoll number:%d\n",stud[i].rollno);
        printf("Name:%s\n",stud[i].name);
        printf("Total mark:%d\n",stud[i].totalmark);

        printf("Date      of      birth      :      %d      /      %d      /      %d      \n\n",
stud[i].d.day,stud[i].d.month,stud[i].d.year);

    }

getch();

}

```

OUTPUT

Enter total number of students 2

Enter details of 1 student

Name: karthik

Roll number:12

Total mark:588

Date of birth (Format:01 06 2010):11 12 1997

Enter details of 2 student

Name: sarita

Roll number:18

Total mark:598

Date of birth (Format:01 06 2010):1 2 1997

STUDENTS DETAILS:

Roll number:12

Name: karthik Total
mark: 588

Date of birth : 11 / 12 / 1997

Roll number:18

Name: sarita

Total mark:598

Date of birth : 1 / 2 / 1997

Union

Union is a user created data type similar to structure but in this case all the members share a common memory location. The size of the union corresponds to the length of the largest member. Since the member share a common location they have the same starting address.

The real purpose of unions is to prevent memory fragmentation by arranging for a standard size for data in the memory. By having a standard data size we can guarantee that any hole left when dynamically allocated memory is freed will always be reusable by another instance of the same type of union. This is a natural strategy in system programming where many instances of different kinds of variables with a related purpose and stored dynamically.

A union is declared in the same way as a structure. The **syntax** of union declaration is

```
union union_name  
  
    {  
  
        type element 1;  
  
        type element 2;  
  
        .....  
  
        type element n;  
  
    };
```

This declares a type template. **Variables are then declared**

```
as:union union_name x,y,z;
```

For example, the following code declares a union data type called Student and a union variable called stud:

```
union student  
  
    {  
  
        int rollno;  
  
        float totalmark;  
  
    };  
  
union student stud;
```

It is possible to combine the declaration of union combination with that of the union variables, as shown below.

```
union union_name  
  
    {
```

```
type element 1;  
type element 2;  
.....  
type element n;  
} var1, var2, ..., varn;
```

The following single declaration is equivalent to the two declaration presented in the previous example.

```
union student  
{  
    int rollno;  
    float totalmark;
```



```
}x,y,z;
```

Exercise: Compare structure and Union

The difference between structure and union is,

| Structure | Union |
|---|--|
| The amount of memory required to store a structure variable is the sum of the size of all the members. | The amount of memory required is always equal to that required by its largest member. |
| Each member has their own memory space. | One block is used by all the members of the union. |
| Keyword struct defines a structure | Keyword union defines a union. |
| <pre>struct s_tag { int ival; float fval; char *cptr; }s;</pre> | <pre>union u_tag { int ival; float fval; char *cptr; }u;</pre> |
| Within a structure all members get memory allocated; therefore any member can be retrieved at any time. | While retrieving data from a union the type that is being retrieved must be the type most recently stored. It is the programmer's responsibility to keep track of which type is currently stored in a union; the results are implementation-dependent if something is stored as one type and extracted as another. |
| One or more members of a structure can be initialized at once. | A union may only be initialized with a value of the type of its first member; thus union u described above (during example declaration) can only be initialized with an integer value. |

Structure:

```
#include<stdio.h>
#include<conio.h>
>void main()

{

struct testing

    {
```

```

        int a;
        char b;
        float c;

        }var
;clrscr();

printf("\nsizeof(var) is %d",sizeof(var));

printf("\nsizeof(var.a)          is
%d",sizeof(var.a));printf("\nsizeof(var.b) is
%d",sizeof(var.b));printf("\nsizeof(var.c) is
%d",sizeof(var.c));

var.a=10;

printf("\nvalue of var.a is %d",var.a);
var.b='b';

printf("\nvalue of var.b is %c",var.b);

var.c=15.55;

printf("\nvalue of var.c is %f",var.c);
printf("\nvalue of var.a is %d",var.a);
printf("\nvalue of var.b is %c",var.b);
printf("\nvalue of var.c is %f",var.c):

```

OUTPUT

```

sizeof(var) is 7

sizeof(var.a) is 2

sizeof(var.b) is 1

sizeof(var.c) is 4
value of var.a is 10
value of var.b is b

value of var.c is 15.550000

```

UNION:

```

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

```

>
v
o
i
d
m
a
i
n
(
)

{

union
testing

{

```

i      intf("\nsizeof(var.a)      is
n      %d",sizeof(var.a));
t      printf("\nsizeof(var.b)    is
a      %d",sizeof(var.b));
;      printf("\nsizeof(var.c)    is
c      %d",sizeof(var.c));
h
a      var.a=10;
r      printf("\nvalue of var.a is %d",var.a);
b      var.b='b';
;
f      printf("\nvalue of var.b is %c",var.b);
l
o      var.c=15.55;
a
t      printf("\nvalue of var.a is %f",var.c);
c      printf("\nvalue of var.a is %d",var.a);
;      printf("\nvalue of var.b is %c",var.b);
      printf("\nvalue of var.c is %f",var.c);
}      getch();

v
a
r
;
c
l
r
s
c
r
(
)
;

printf("\nsizeof(var) is
%d",sizeof(var));

p
r

```

OUTPUT

```

sizeof(var) is 4

sizeof(var.a) is 2

sizeof(var.b) is 1

sizeof(var.c) is 4

```

File Handling in C

- A file is a collection of related data that a computer treats as a single unit.
- Computers store files to secondary storage so that the contents of files remain intact when a computer turns off.
- When a computer reads a file, it copies the file from the storage device to memory; when it

writes to a file, it transfers data from memory to the storage device.

- C uses a structure called FILE (defined in stdio.h) to store the attributes of a file.

Steps in Processing a File

1. Create the stream via a pointer variable using the FILE structure: FILE *p;
2. Open the file, associating the stream name with the file name.
3. Read or write the data.
4. Close the file.

Five major operations can be performed on file are:

1. Creation of a new file.
2. Opening an existing file.
3. Reading data from a file.
4. Writing data in a file.
5. Closing a file

To handling files in C, *input/output* functions available in the *stdio* library are:

| Function | Uses/Purpose |
|------------------------|----------------|
| fopen | Opens a file. |
| fclose | Closes a file. |

getc

Reads a character from a file

putc

Writes a character to a file

[getw](#)

Read integer

[putw](#)

Write an integer

[fprintf](#)

Prints formatted output to a file

[fscanf](#)

Reads formatted input from a file

[fgets](#)

Read string of characters from a file

[fputs](#)

Write string of characters to file

[feof](#)

Detects end-of-file marker in a file

The basic format of fopen is:

Syntax:

```
FILE *fopen( const char * filePath, const char * mode );
```

Parameters

- filePath: The first argument is a pointer to a string containing the name of the file to be opened.
- mode: The second argument is an access mode.

C *fopen()* access mode can be one of the following values:

Description

| | |
|---|--|
| r | Opens an existing text file. |
| w | Opens a text file for writing if the file doesn't exist then a new file is created. |
| a | Opens a text file for appending(writing at the end of existing file) and create the file if it does not exist. |

| | |
|----|--|
| r+ | Opens a text file for reading and writing. |
| w+ | Open for reading and writing and create the file if it does not exist. If the file exists then make it blank. |
| a+ | Open for reading and appending and create the file if it does not exist. The reading will start from the beginning writing can only be appended. |

Return Value

C fopen function returns `NULL` in case of a failure and returns a `FILE stream pointer` on success.

Example

:

```
#include<stdio.h>

int main()
{
    FILE *fp;

    fp = fopen("fileName.txt","w");

    return 0;
}
```

- The above example will create a file called `fileName.txt`.
- The `w` means that the file is being opened for writing, and if the file does not exist then the new file will be created.

The basic format of fclose is:

Syntax:

```
int fclose( FILE * stream );
```

Return Value

C `fclose` returns `EOF` in case of failure and returns `0` on success.

Example:

```
#include<stdio.h>

int main()
{
    FILE *fp;

    fp = fopen("fileName.txt","w");

    fprintf(fp, "%s", "Sample Texts");

    fclose(fp);

    return 0;
}
```

- The above example will create a file called `fileName.txt`.
- The `w` means that the file is being opened for writing, and if the file does not exist then the new file will be created.
- The `fprintf` function writes `Sample Texts` text to the file.
- The `fclose` function closes the file and releases the memory stream.

`getc()` function is C library function, and it's used to read a character from a file that has been

opened in read mode by `fopen()` function.

```
int getc( FILE * stream );
```

Return Value

- `getc()` function returns next requested object from the stream on success.
- Character values are returned as an unsigned char cast to an int or EOF on end of file or error.
- The function `feof()` and `ferror()` to distinguish between end-of-file and error must be used.

Example:

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

```
int main()
```

```
{
```

```
FILE *fp = fopen("fileName.txt", "r");
```

```
int ch = getc(fp);
```

```
while (ch != EOF)
```

```
{
```

```
/* To display the contents of the file on the screen */ putchar(ch);
```

```
ch = getc(fp);
```

```
}
```

```
if (feof(fp))
```

```
printf("\n Reached the end of file.");
```

```
else
```

```
printf("\n Something gone wrong.");
```

```
fclose(fp);
```

```
getchar();
```

```
return 0;
```

```
}
```

putc() is a C library function, and it's used to write a character to the file. This function is used for writing a single character in a stream along with that it moves forward the indicator's position.

```
int putc( int c, FILE * stream );
```

Example:

```
int main (void)

{

    FILE * fileName;

    char ch;

    fileName = fopen("anything.txt","wt");

    for (ch = 'D' ; ch <= 'S' ; ch++) {

        putc (ch , fileName);

    }

    fclose (fileName);
```

C getw function is used to read an integer from a file that has been opened in read mode. It is a file handling function, which is used for reading integer values.

```
int getw( FILE * stream );
```

C putw function is used to write an integer to the file.

Syntax:

```
int putw( int c, FILE * stream );
```


Example:

```
int main (void)
{
    FILE *fileName;
```

```
int i=2, j=3, k=4, n;  
  
fileName = fopen ("anything.c", "w");  
  
putw(i, fileName);
```

```
putw(j, fileName);  
  
putw(k, fileName);  
  
fclose(fileName);  
  
fileName = fopen ("test.c", "r");  
while(getw(fileName) != EOF)  
{  
    n= getw(fileName);  
    printf("Value is %d \t: ", n);  
}  
  
fclose(fp);  
  
return 0;  
}
```

C fprintf function pass arguments according to the specified format to the file indicated by the stream. This function is implemented in file related programs for writing formatted data in any file.

Syntax:

```
int fprintf(FILE *stream, const char *format, ...)
```

Example:

```
int main (void)
```

```
{  
    FILE *fileName;  
  
    fileName = fopen("anything.txt","r");  
  
    fprintf(fileName, "%s %s %d", "Welcome", "to", 2018); fclose(fileName);  
  
    return(0);  
}
```

C fscanf function reads formatted input from a file. This function is implemented in file related programs for reading formatted data from any file that is specified in the program.

Syntax:

```
int fscanf(FILE *stream, const char *format, ...)
```

It returns the number of variables that are assigned values, or EOF if no assignments could be made.

Example:

```
int main()
{
    char str1[10], str2[10];

    int yr;

    FILE* fileName;

    fileName = fopen("anything.txt", "w+");

    fputs("Welcome to", fileName);

    rewind(fileName);

    fscanf(fileName, "%s %s %d", str1, str2, &yr);

    printf(".....\n");

    printf("1st word %s \t", str1);
```

```
printf("2nd word %s \t", str2);

printf("Year-Name %d \t", yr);

fclose(fileName);

return (0);

}
```

C fgets function is implemented in file related programs for reading strings from any particular file. It gets the strings 1 line each time.

Syntax:

```
char *fgets(char *str, int n, FILE *stream)
```

Example:

```
void main(void)

{

    FILE* fileName;

    char ch[100];

    fileName = fopen("anything.txt", "r");

    printf("%s", fgets(ch, 50, fileName));

    fclose(fileName);

}
```

- On success, the function returns the same str parameter
- C fgets function returns a NULL pointer in case of a failure.

C fputs function is implemented in file related programs for writing string to any particular file.

□ Syntax:

- `int fputs(const char *str, FILE *stream)`

□ Example:

- `#include<stdio.h>`

```

• int main()
• {
•     FILE *fp;
•     fputs("This is a sample text file.", fp);
•     fputs("This file contains some sample text data.", fp);
•
•     fclose(fp);

```

- In this function returns non-negative value, otherwise returns EOF on error.

C feof function is used to determine if the end of the file (stream), specified has been reached or not. This function keeps on searching the end of file (eof) in your file program.

Syntax:

```
int feof(FILE *stream)
```

Here is a program showing the use of feof().

Example:


```
#include<stdio.h>

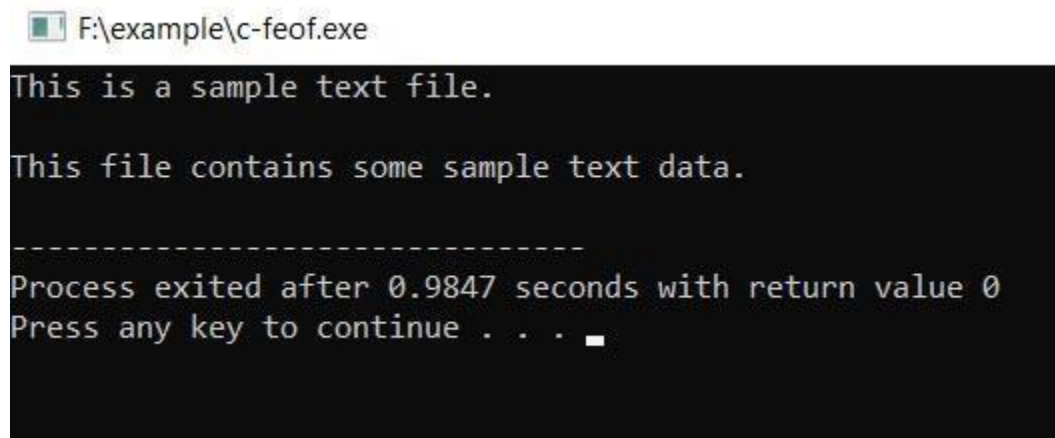
int main()

{
FILE *filee =
NULL;char buf[50];
filee = fopen("infor.txt","r");
if(filee)
{
while(!feof(filee))
{
fgets(buf, sizeof(buf), filee);

puts(buf);
```

```
    }  
    fclose(filee);  
}  
return 0;  
}
```

Output:



```
F:\example\c-feof.exe  
This is a sample text file.  
This file contains some sample text data.  
-----  
Process exited after 0.9847 seconds with return value 0  
Press any key to continue . . .
```

C feof function returns true in case end of file is reached, otherwise it's return false.[Explanation:](#)

1. It first tries to open a text file infor.txt as read-only mode.
2. Then as the file gets opened successfully to read, it initiates the while loop.
3. The iteration continues until all the statement/lines of your text file get to read as well as displayed.

Lastly, you have to close the file.

