Unit
6

Structures Unions and Files

6.1 USER DEFINED DATA TYPES: TYPEDEF AND ENUM

6.1.1 Typedef

In C language a user can define an identifier that represents an existing data type. The user defined datatype identifier can later be used to declare variables. The general syntax is

```
typedef type identifier;
```

Here type represents existing data type and 'identifier' refers to the 'row' name given to the data type.

Example:

```
typedef int salary;
typedef float average;
```

Here salary symbolizes int and average symbolizes float. They can be later used to declare variables as follows:

```
Salary dept1, dept2;
Average section1, section2;
```

Therefore dept1 and dept2 are indirectly declared as integer datatype and section1 and section2 are indirectly float data type.

Example:

```
#include <stdio.h>

typedef unsigned short int USHORT;

int main()
{
  USHORT i = 2004;

  printf("Year %d\n", i);

  return 0;
}
```

Output:

```
Year 2004
```

6.1.2 Enumerated Data Type

An enumeration consists of a set of named integer constants. Enumerated data type is also user defined data type which is defined as follows:

```
enum identifier {value1, value2 .... Value n};
```

The identifier is a user defined enumerated datatype which can be used to declare variables that have one of the values enclosed within the braces. After the definition we can declare variables to be of this 'new' type as below.

```
enum identifier V1, V2, V3, ..... Vn
```

The enumerated variables V1, V2, ... Vn can have only one of the values value1, value2 ..... Value n

Example:

```
enum DAY /* Defines an enumeration type                    */
{
        saturday,          /* Names day and declares a      */
        sunday = 0,        /* variable named workday with   */
        monday,            /* that type                     */
    tuesday,
    wednesday,             /* wednesday is associated with 3 */
    thursday,
    friday
} workday;
```

The value 0 is associated with saturday by default. The identifier sunday is explicitly set to 0. The remaining identifiers are given the values 1 through 5 by default.

Example Program:

```
#include <stdio.h>
int main()
{
  enum {RED=5, YELLOW, GREEN=-4, BLUE};

  printf("RED = %d\n", RED);
  printf("YELLOW = %d\n", YELLOW);
  printf("GREEN = %d\n", GREEN);
  printf("BLUE = %d\n", BLUE);
  return 0;
}
```

Output:

```
RED = 5
YELLOW = 6
GREEN = -4
BLUE = -5
```

6.2 INTRODUCTIONS TO STRUCTURE

Structure is a mechanism for packing data of different types into single unit. A structure is a convenient tool for handling a group of logically related data items.

- Structure is a user defined data type, which contains individual element that can defer in type.
- Hence structure can contain integer, char, float, double etc... elements.
- Arrays, pointer and other structures can be included in structures.
- Individual elements are referred to as members.

6.3 DECLARATION INITIALIZATION AND ACCESSING STRUCTURE

6.3.1 Defining a Structure

A structure being complicated than array should be defined in terms of its individual members.General format of a structure definition is:

```
struct <name>
{
member 1;
member 2;


member n;
};
```

In above declaration:

- struct is a keyword,
- <name> is the name that identifies the structure of this type.

member1, member2...etc; are individual member declarations.

Example:

```
struct person
{
char name[20];
int day;
int month;
int year;
};
```

Declaring Structure Variables

After defining a structure format we can declare variables of that type. A structure variable declaration is similar to the declaration of variables of any other data types. It includes the following elements:

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

General format of a declaration of structure variables is:

struct structure_name variable_name;

Example:

```
struct person p1,p2,p3;
```

We can also declare the variable of the structure at the time of declaring the structure.For Example:

```
struct person
{
char name[20];
int day;
int month;
int year;
}p1,p2,p3;
```

6.3.2 Structure Initialization

Like any other data type, a structure variable can be initialized at compile time.

Example

```
main()
{
struct person
{
char name[20];
int day;
int month;
int year;
};
struct personp1={"Jainik",11,3,2010};
struct person p2={"Krinal",11,9,2012};
……..
……..
}
```

Above code block assigns initial values to structure variables as follows:

Structure: Person		
Members of structures	**Variables of Structures**	
	P1	**P2**
Name	Jainik	Krinal
Day	11	11
Month	3	9
Year	2010	2012

6.3.3 Accessing Structure Members

The members of a structure are usually accessed individually as separate entities

```
variable.member
```

- Variable refers to the name of the struct type variable.
- Member refers to the name of the member within the structure.

Example

For the structure defined in above example we can use following statements to input the value of the variables.

```
strcpy(p1.name,"xyz");
p1.day=1;
p1.month=1;
p1.year=2009;
```

Or

```
printf ("\n enter name of the person: ");
scanf ("%s", p1.name);
printf ("\n date of birth: ");
scanf ("%d - %d - %d", &p1.day, &p1.month, &p1.year);
```

To display the value of the variables use following statements.

```
printf ("\n name of the person : %s", p1.name);
printf ("\n date of birth: %d-%d-%d", p1.day, p1.month, p1.year);
```

6.3.4 Program Examples of Structures

1.
```
#include<stdio.h>
#include<conio.h>
void main()
{
    struct book
    {
        char book_name[100];
        char author[100];
        int pages;
        float price;
    };
    struct book book1;
    printf("Book Name : ");
    gets(book1.book_name);
    fflush(stdin);
```

```
    printf("Book Author : ");
    gets(book1.author);
    printf("Book Pages : ");
    scanf("%d",&book1.pages);
    printf("Book Price : ");
    scanf("%f",&book1.price);
    printf("\nBook Information \n");
    printf("Book Name   : %s\n",book1.book_name);
    printf("Book Author : %s\n",book1.author);
    printf("Book Pages  : %d\n",book1.pages);
    printf("Book Price  : %f\n",book1.price);
    getch();
}
```

Or

```
#include <stdio.h>
#include<conio.h>
struct book
{
    char book_name[100];
    char author[100];
    int pages;
    float price;
};
void main()
{
    struct book book1;
    printf("Book Name : ");
    gets(book1.book_name);
    fflush(stdin);
    printf("Book Author : ");
    gets(book1.author);
    printf("Book Pages : ");
    scanf("%d",&book1.pages);
    printf("Book Price : ");
    scanf("%f",&book1.price);
    printf("\nBook Information \n");
    printf("Book Name   : %s\n",book1.book_name);
    printf("Book Author : %s\n",book1.author);
    printf("Book Pages  : %d\n",book1.pages);
    printf("Book Price  : %f\n",book1.price);
    getch();
}
```

Or

```
#include <stdio.h>
#include<conio.h>
struct book
{
    char book_name[100];
    char author[100];
    int pages;
    float price;
}book1;
void main()
{
    printf("Book Name : ");
    gets(book1.book_name);
    fflush(stdin);
    printf("Book Author : ");
    gets(book1.author);
    printf("Book Pages : ");
    scanf("%d",&book1.pages);
    printf("Book Price : ");
    scanf("%f",&book1.price);
    printf("\nBook Information \n");
    printf("Book Name  : %s\n",book1.book_name);
    printf("Book Author : %s\n",book1.author);
    printf("Book Pages  : %d\n",book1.pages);
    printf("Book Price  : %f\n",book1.price);
    getch();
}
```

Output:

```
Book Name : Computer Utilization and Programming
Book Author : xyz
Book Pages : 406
Book Price : 450.40

Book Information
Book Name  : Computer Utilization and Programming
Book Author : xyz
Book Pages  : 406
Book Price  : 450.40
```

2.
```
#include<stdio.h>
#include<conio.h>
struct book
{
        char book_name[100];
        char author[100];
        int pages;
        float price;
}b1,b2;
void main()
{
        printf("\nBook1 Detail\nBook Name : ");
        gets(b1.book_name);
        fflush(stdin);
        printf("Book Author : ");
        gets(b1.author);
        printf("Book Pages : ");
        scanf("%d",&b1.pages);
        printf("Book Price : ");
        scanf("%f",&b1.price);
        printf("\nBook2 Detail\nBook Name : ");
        fflush(stdin);
        gets(b2.book_name);
        fflush(stdin);
        printf("Book Author : ");
        gets(b2.author);
        printf("Book Pages : ");
        scanf("%d",&b2.pages);
        printf("Book Price : ");
        scanf("%f",&b2.price);
        printf("\nBook1 Information \n");
        printf("Book Name   : %s\n",b1.book_name);
        printf("Book Author : %s\n",b1.author);
        printf("Book Pages  : %d\n",b1.pages);
        printf("Book Price  : %f\n",b1.price);
        printf("\nBook2 Information \n");
        printf("Book Name   : %s\n",b2.book_name);
        printf("Book Author : %s\n",b2.author);
        printf("Book Pages  : %d\n",b2.pages);
        printf("Book Price  : %f\n",b2.price);
        getch();
}
```

Output:

```
Book1 Detail
Book Name : Computer Utilization and Programming
Book Author : xyz
Book Pages : 406
Book Price : 450.40

Book2 detail
Book Name  : C Programming
Book Author : abc
Book Pages  : 150
Book Price  : 320.40

Book1 Information
Book Name : Computer Utilization and Programming
Book Author : xyz
Book Pages : 406
Book Price : 450.40

Book2 Information
Book Name  : C Programming
Book Author : abc
Book Pages  : 150
Book Price  : 320.40
```

6.4 ARRAY OF STRUCTURES

If we need the details of 100 books, then it is difficult to declare 100 structure variables using the above two methods, but this problem can be solved by array declaration of structure variables. We can declare array of structures just like we declare array of integers or array of variables of any other data type.

Consider the following,

```
struct  date
{
int month, day, year;
};
```

Let's now create an array of birthdays:

```
struct date birthdays[5];
```

This creates an array of 5 elements which have the structure of date.

```
birthdays[1].month = 12;
birthdays[1].day  = 04;
birthdays[1].year  = 1998;
—birthdays[1].year;
```