Unit 10 Structures and Unions

Structure:

10.1 Introduction

In the previous units, you studied about the pointers as a most powerful tool in C programming language. It is due to the pointer only that makes C as the most beautiful language. In this unit, you will study another useful ways of making the program easy and efficient. This unit will enable you to learn about the structures and unions.

As we know an array is a data structure whose elements are all of the same data type. We now turn our attention to the structure, which is a data structure whose individual elements can differ in type. Thus, a single structure might contain integer elements, floating-point elements and character elements. Pointers, arrays and other structures can also be included as elements within a structure.

This unit is concerned with the use of structures within a C program. We will see how structures are defined, and how their individual members are accessed and processed within a program.

Closely associated with the structure is the union, which also contains multiple members. Unlike a structure, however, the members of a union share the same storage area, even though the individual members may differ in type.

Objectives:

After studying this unit, you should be able to:

- handle a group of logically related data items known as structures.
- declare an array of structures, each element of the array representing a structure variable.
- pass structure as an argument to functions and return structure from functions.
- refer to (i.e., point to) an incomplete type, including itself.
- handle a group of logically related data items in terms of unions.

10.2 Basics of Structures

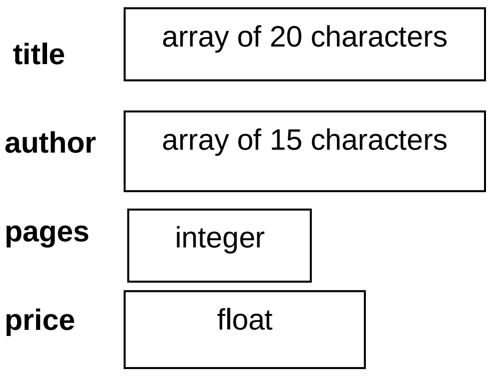
C supports a constructed data type known as structure, which is a method for packing data of different types. A structure is a convenient tool for handling a group of logically related data items. Structures help to organize complex data in a more meaningful way. It is a powerful concept that we may often need to use in our program design.

Structure Definition: A Structure definition creates a format that may be used to declare structure variables. For e.g., Consider a book database consisting of book name, author, number of pages and price.

```
struct book_bank
{
  char title[20];
  char author[15];
  int pages;
  float price;
};
```

The keyword **struct** declares a structure to hold the details of four fields, namely title, author, pages and price. These fields are called structure elements or members. Each member may belong to a different type of data. book_bank is the name of the structure and is called the structure tag. The tag name may be used subsequently to declare variables that have the tag's structure. Note that the above declaration has not declared any variables. It simply describes a format called template to represent information as shown below:

struct book_bank



We can declare structure variables using the tag name anywhere in the program. e.g, the statement:

struct book_bank book1, book2, book3;

declares book1, book2 and book3 as variables of type **book_bank.**

Each one of these variables has four members as specified by the template. The complete declaration might look like this:

```
struct  book_bank
{
  char title[20];
  char author[15];
  int pages;
  float price;
};
struct  book_bank book1, book2, book3;
```

It is also allowed to combine both the template declaration and variables declaration in one statement.

```
struct book_bank
{
  char title[20];
  char author[15];
  int pages;
  float price;
} book1, book2, book3;
```

General format of a Structure Definition:

The general format of a structure definition is as follows:

```
struct tag_name
{
   data_type member1;
   data_type member2;
   -------
};
```

In defining a structure you may note the following syntax:

1. The template is terminated with a semicolon.
2. While the entire declaration 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 such as tag_name can be used to declare structure variables of its type, later in the program.

Giving values to Members:

Structure members need to be linked to the structure variables in order to make them meaningful members. The link between a member and a variable is established using the member operator '.' which is also known as 'dot operator' or 'period operator'.

Here is how we would assign values to the members of book1.

```
strcpy(book1.title,"BASIC");

strcpy(book1.author,"Balagurusamy");

book1.pages = 250;

book1.price = 28.50;
```

We can also give the values through the keyboard.

```
gets(book1.title);

gets(book1.author);

printf("%d",book1.pages);

printf("%f",book1.price);
```

Structure Initialization:

```
void main( )
{
        struct st_record
        {
          char name[20];
          int weight;
          float height;
        };
        static struct st_record student1 = {“Suresh”, 60, 180.75};
        static struct st_record student2 = {“Umesh”, 53, 170.60};
}
```

Program 10.1: To print the date of joining of a person

```
#include<conio.h>
#include<stdio.h>
struct personal
{
        char name[30];
        int day;
        int month;
        int year;
        float salary;
};
void main()
{
        struct personal p;
        printf(“Enter the name:\n)";
        gets(p.name);
        printf(“Enter the day of joining:\n)";
        scanf(“%d”,&p.day);
        printf(“Enter the month of joining:\n");
        scanf(“%d”,&p.month);
        printf(“Enter the year of joining:\n)";
        scanf(“%d”,&p.year);
        printf(“Enter the salary:\n)";
        scanf(“%f”, & p.salary);
        printf(“\nName:",p.name);
        printf("\nDate of joining:%d  %d  %d",p.day,p.month,p.year);
        printf(“Salary:",p.salary);
        getch();
}
```

Comparison of structure variables

Two variables of same structure type can be compared in the same way as ordinary variables. If **person1** and **person2** belong to the same structure, then the following operations are valid:

Operation	Meaning
person1 = person2	Assign **person2** to **person1**
person1 == person2	Compare all members of **person1** and **person2** and return 1 if they are equal, 0 otherwise.
person1 != person2	Return 1 if all the members are not equal, 0 otherwise

Program 10.2: To Compare structure variables

```
#include <stdio.h>
#include<conio.h>
struct stclass{
  int number;
  char name[20];
  float marks;
};

void main()
```

```
{
  int x;
  static struct stclass student1 = {111,"Rao",72.50};
  static struct stclass student2 = {222,"Reddy",67.00};
  struct stclass student3;
  student3 = student2;
  x=((student3.number   ==   student2.number)   &&   (student3.marks   ==
student2.marks))?  1:0;
  if(x==1)
  {
    printf("\nStudent2 and Student3 are same ");
   printf(“ %d\t %s\t  %f\t“,student3.number,student3.name,student3.marks);
  }
  else
  {
    printf("\nStudent2 and student3 are different)";
  }
  getch();
}
```

Self Assessment Questions

1. A ___________ is a convenient tool for handling a group of logically related data items.
2. We can declare structure variables using the tag name anywhere in the program. (True/False)
3. ____________ is a method for packing data of different types.
4. If person1 and person2 are variables of the same type structure then the expression person1>person2 is valid. (True/False)
5. _______________ is a method for packing data of different types.
6. The link between a member and a variable is established using the member operator ______________.

10.3 Structures and Functions

We can write programs with structures by using modular programming. We can write a function that returns the structure. While writing the function, you should indicate the type of structure that is returned by the function. The **return** statement should return the structure using a variable. It is possible to pass a structure as an argument. We can modify a member of the structure by passing the structure as an argument. The changes in the member made by the function are retained in the called module. This is not against the principle of call by value because we are not modifying the structure variable, but are instead modifying the members of the structure.

Program 10.3: To illustrate the concept of structures and functions

```
struct student
{
   name char[30];
   marks float;
};
main ( )
{
   struct student student1;
   student1 = read_student ( );
   print_student( student1);
   read_student_p(student1);
   print_student (student1);
}
struct student read_student( )
{
   struct student student2;
   gets(student2.name);
   scanf("%d",&student2.marks);
   return (student2);
}
void print_student (struct student student2)
{
   printf( "name is %s\n", student2.name);
   printf( "marks are%d\n", student2.marks);
}
void read_student_p(struct student student2)
{
   gets(student2.name);
   scanf("%d",&student2.marks);
}
```

Explanation

1. The function read_student reads values in structures and returns the structure.
2. The function print_student takes the structure variable as input and prints the content in the structure.
3. The function read_student_p reads the data in the structure similarly to read_student. It takes the structure **student** as an argument and puts the data in the structure. Since the data of a member of the structure is modified, you need not pass the structure as a pointer even though structure members are modified. Here you are not modifying the structure, but you are modifying the structure members through the structure.

Self Assessment Questions

7. We cannot write a function that returns the structure. (True/False)
8. We can modify a member of the structure by passing the structure as a _____________.

10.4 Arrays of Structures

We can use structures to describe the format of a number of related variables. For example, in analyzing the marks obtained by a class of students, we may use a template to describe student name and marks obtained in various subjects and then declare all the students as structure variables. In such cases, we may declare an array of structures, each element of the array representing a structure variable. e.g, **struct** stclass student[100]; defines an array called student, that consists of 100 elements. Each element is defined to be of the type struct stclass. Consider the following declaration:

```
struct marks
{
  int subject1;
  int subject2;
  int subject3;
};

main( )
```

```
{
  static struct marks student[3]={{45,68,81},{75,53,69},{57,36,71}};
}
```

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

student[0].subject1 = 45;
student[0].subject2 = 68;
……..
student[2].subject3 = 71;

Program 10.4: To process employee details using structures

```
#include<conio.h>
#include<stdio.h>
struct employee
{
int empno;
char name[30];
int basic;
int hra;
};
void main()
{
int i,j,n,net[50];
float avg;
employee e[50];
printf("\nEnter the number of employees:");
scanf("%d", &n);
printf("\nEnter Empno.\tName\tBasic\tHra of each employee:\n");
for(i=0;i<n;i++)
{
 scanf("%d",&e[i].empno);
 gets(e[i].name);
 scanf("%d",&e[i].basic);
 scanf(%d",&e[i].hra);
 net[i]= e[i].basic+e[i].hra;
 avg=avg+net[i];
}
avg=avg/n;
printf("\nEmpno.\tName\tNetpay\n");
for(i=0;i<n;i++)
{
if(net[i]>avg)
{
 printf(e[i].empno\t)";
 printf(e[i].name\t)";
 printf(net[i]\n");
}
}
getch();
}
```

Program 10.5: To process student details using structures

```
#include<conio.h>
#include<stdio.h>
struct student
{
int rollno;
char name[30];
int marks1;
int marks2;
int marks3;
};
void main()
{
int i,j,n,tot[50],t;
student s[50],temp;
printf("\nEnter the number of students:");
scanf("%d",&n);
printf("\nEnter Rollno.\tName\tMarks1\tMarks2\tMarks3 of each student:\n");
for(i=0;i<n;i++)
{
 scanf("%d",&s[i].rollno);
 gets(s[i].name);
 scanf("%d",&s[i].marks1);
 scanf(“%d”,&s[i].marks2);
 scanf(“%d”,&s[i].marks3);
tot[i]= s[i].marks1+s[i].marks2+s[i].marks3;
}
for(i=0;i<n-1;i++)
{
 for(j=i+1;j<n;j++)
 {
  if(tot[i]<tot[j])
  {
   temp=s[i];
   s[i]=s[j];
   s[j]=temp;
   t=tot[i];
   tot[i]=tot[j];
   tot[j]=t;
  }
 }
}
printf("\nRollno.\tName\tTotal marks in decreasing order of total marks
is:\n");
for(i=0;i<n;i++)
{
printf(“%d\t”,s[i].rollno);
printf(“%s\t”,s[i].name);
printf(“%d\t”,s[i].tot);
}
getch();
}
```

Self Assessment Questions

9. We can use structures to describe the format of a number of related variables. (True/False)
10. You can declare an array of structures where each element is defined to be of the type _________.

10.5 Pointers to Structures

Pass by value may be very inefficient if the structure is large (i.e., has many members). They have identical declaration syntax and member access, but they serve a very different purpose. Defining pointer types is the same as for variables of primitive types.

Example:

```
struct Point {
        int x;
        int y;
    };
struct Rectangle {
        struct Point topleft;
        struct Point bottomright;
    };
    struct Point pt = { 50, 50 };
    struct Point *pp;
    pp = &pt;
    (*pp).x = 100; /* pt.x is now 100. */
```

Notice the parentheses around the de referenced pointer.

```
(*pp).x = 100;
```

This is necessary to enforce correct precedence.

An alternative notation permits simpler pointer access to structure members.

```
(*pp).x = 100;
pp->x = 100; /* equivalent */
```

Another example,

```
struct Rectangle rect, *pr = &rect;
rect.topleft.x = 50; /* equivalent operations */
(*pr).topleft.x = 50;
pr->topleft.x = 50;
```

Self Assessment Questions

11. The parentheses around the de referenced pointer is necessary to enforce the correct _________.
12. An alternative notation other than dot, permits simpler pointer access to structure members is ____________.

10.6 Self-referential Structures

The ability to refer to (ie, point to) an incomplete type, including itself, is an important property for constructing a variety of data-structures. For example: linked-lists, binary trees, graphs, hash tables, and more.

Linked lists come in two basic varieties: singly linked and doubly linked.

We describe here a simple version of a singly linked list.

List consists of a set of nodes, where each node contains an item and a pointer to another list node.

```
struct List {
    int item;
    struct List *next;
};
```

More about linked lists, you will study in the units to come.

10.7 Unions

Unions look similar to structures. They have identical declaration syntax and member access, but they serve a very different purpose.

```
union Utype {
    int ival;
    float fval;
    char *sval;
};

union Utype x, y, z;
```

Accessing members of a union is via "." member operator or, for pointers to unions, the -> operator.

A union holds the value of one-variable at a time. The compiler allocates storage for the *biggest* member of the union. The type retrieved from the union must be the type most recently stored. Otherwise, the result is implementation dependent.

```
union Utype x;
x.fval = 56.4;          /* x holds type float. */
printf("%f\n", x.fval); /* OK. */
printf("%d\n", x.ival); /* Implementation dependent. */
```

Unions are used to store one of a set of different types. These are commonly used to implement a "variant" array. (This is a form of generic programming.) There are other uses also, but they are quite advanced (e.g., concern the alignment properties of unions).

Self Assessment Questions

13. A __________ holds the value of one-variable at a time.
14. The compiler allocates storage for the *smallest* member of the union. (True/False)

10.8 Summary

A structure is a convenient tool for handling a group of logically related data items. Structure members need to be linked to the structure variables in order to make them meaningful members. We can write programs with structures by using modular programming. We can use structures to describe the format of a number of related variables. Passing a pointer to a structure is generally much more efficient than making a copy of the structure itself. The ability to refer to (i.e., point to) an incomplete type, including itself, is an important property for constructing a variety of data-structures.

Unions have identical declaration syntax and member access, but they serve a very different purpose. A union holds the value of one-variable at a time. The compiler allocates storage for the *biggest* member of the union.

10.9 Terminal Questions

1. Write the output that will be generated by the following C program:

```
typedef struct
{
        char *a;
        char *b;
```

```
            char *c;
        } colors;
        void main()
        {
            void fun( colors sample);
            static colors sample = {“red”, “green”, “blue”};
            printf( (“%s %s %s\n”, sample.a, sample.b,
             sample.c);
            fun(sample);
            printf( (“%s %s %s\n”, sample.a, sample.b,
             sample.c);
        }

        void  fun (colors sample)
        {
            strcpy (sample.a=”cyon”);
            strcpy (sample.b=”magenta”);
            strcpy (sample.c=”yellow”);
            printf(“%s %s %s\n”, sample.a, sample.b,
            sample.c);
            return;
        }
```

2. Describe the output generated by the following program. Distinguish between meaningful and meaningless output.

```
    #include <stdio.h>
    main()
    {
        union {
            int i;
            float f;
            double d;
            } u;
    printf(“%d\n”, sizeof(u));
    u.i= 100;
    printf(“%d %f %f\n”, u.i, u.f, u.d);
```

```
u.f=0.5;
printf(“%d %f %f\n”, u.i, u.f, u.d);
u.d = 0.0166667;
printf(“%d %f %f\n”, u.i, u.f, u.d);
}
```

10.10 Answers to Self Assessment Questions

1. structure
2. true
3. Array
4. false
5. Structure
6. dot(.)
7. true
8. argument.
9. True
10. struct
11. precedence
12. ->
13. union
14. false

10.11 Answers to Terminal Questions

1. red green blue
   cyan magenta yellow
   red blue green

2. 8
   100 0.000000 -0.000000
   0 0.500000 -0.000000
   -25098 391364288.000000 0.016667

The first line displays the size of the union (8 bytes, to accommodate double data). In the second line, only the first value (100) is meaningful. In the third line, only the second value (0.500000) is meaningful. In the last line, only the last value (0.016667) is meaningful.

10.12 Exercises

1. What is a structure? How does a structure differ from an array?
2. What is a member? What is the relationship between a member and a structure?
3. Describe what is wrong in the following structure declaration:

```
struct
{
        int number;
        float price;
}
main()
{
          ……………
……………
}
```

4. Describe Array of structures with an example program.
5. Define a structure called **cricket** that will describe the following information:

   (i) player name (ii) team name (iii) batting average

   Using **cricket**, declare an array **player** with 50 elements and write a program to read the information about all the 50 players and print a team-wise list containing names of players and print a team-wise list containing names of players with their batting average.
6. How is a structure type pointer variable declared?
7. Write a program to find the number of characters in a string using pointers.