STRUCTURES IN C PROGRAMMING

A structure can be considered as a template used for defining a collection of variables under a single name. Structures help programmers to group elements of different data types into a single logical unit (Unlike arrays which permit a programmer to group only elements of same data type).

Suppose we want to store a date inside a C program. Then, we can define a structure called *date* with three elements day, month and year. The syntax of this structure is as follows:

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

A structure type is usually defined at the beginning of a program. This usually occurs just after the main() statement in a file. Then a variable of this structure type is declared and used in the program. For example:

```
struct date order_date;
```

Note : When you first define a structure in a file, the statement simply tells the C compiler that a structure exists, but causes no memory allocation. Only when a structure variable is declared, memory allocation takes place.

Initializing Structures

Unlike standard variables, the syntax for initialising structure variables is different.The structure elements are accessed using the dot notation. The individual elements are initialised as follows :

```
order_date.day = 9;
order_date.month = 12;
order_date.year = 1995;
```

One can either initialize a structure by initializing the individual elements as shown above or by simply listing the element's value inside curly braces, with each value separated by a comma.

static struct date order_date = {9,12,1995};

Similar to initialization of arrays, partial initialization of a structure can be done as follows :

static struct date order_date = {9,12};

In the above case the *year* element is initialised to zero.

Current ANSI standards allow entire structures to be assigned one to another providing that the structures have the same number of members and that the members are of the same data type.

Using Typedef keyword

C language allows a programmer to rename data types using the keyword *typedef*.For example:

typedef unsigned int Uint;
Uint EmpID;

Here you have typedefined a unsigned integer as Uint, you can then use Uint in your program as any native data type, and declare other variables with its data type. You can similarly typedef a structure too.

```
typedef struct point{
  float x;
  float y;
} center_of_circle;
```

Then you can use the newly defined data type, as in the following example:

center_of_circle center1;
center_of_circle center2;

Nested structures

One can define a structure which in turn can contain another structure as one of its members.

```
typedef struct circle_parameters{
```

```
    center_of_circle center;
    float radius;
} circle;
```

circle c1;

The definition of a circle structure requires that a center_of_circle structure be previously defined to the compiler, or a compiler error is generated.

To access a particular member inside the nested structure requires the usage of the dot notation.

```
c1.center.x = 10;
```

This statement sets the x value of the center of the circle to 10.

Arrays of structures

C does not limit a programmer to storing simple data types inside an array. User defined structures too can be elements of an array.

```
struct date birthdays[10];
```

This defines an array called birthdays that has 10 elements. Each element inside the array will be of type struct *date*. Referencing an element in the array is quite simple.

birthdays[1].month = 09;

birthdays[1].day = 20;

birthdays[1].year = 1965;

Initialisation of structure arrays is similar to initialization of multidimensional arrays :

static struct birthdays[10] = {{9,30,1965},{9,26,1971}};

will initialise the first two elements of the *birthdays* array.

Structures containing arrays

There can also be structures containing arrays as its elements. Do not confuse this concept with array of structures.

```
struct car{
  char name[5];
  int model_number;
  float price;
};
```

This sets up a structure with a name car which has an integer member variable *model_number*, a float member variable *price* and a character array member called *name*. The character array member has an array of 5 characters.

We can now define a variable of type struct car.

struct car car1;

and initialize its values:

static struct car car1 = {{'h','o','n','d','a'},1990,10000};

The above statement indicates the syntax to be used when initializing a structure containing an array.

Structures and Pointers

Just like a variable, you can declare a pointer pointing to a structure and assign the beginning address of a structure to it. The following piece of code will help understand this concept.

```
typedef struct {
  int account_num;
  char account_type;
  char f_name[40];
  char l_name[40];
  float balance;
} account;
```

Then a pointer can be define as pointing to a variable, as follows:

account acct1, *pt1; pt1 = &acct1;

Observe that the character "*" is used in front of pt1 to specify that pt1 is a pointer, and the character "&" is used in front of acct1 to pass its address to pt1 rather than its value.

Another important concept one must learn is the usage of the -> operator in conjunction with a pointer variable pointing to a structure.

Source : http://www.peoi.org/Courses/Coursesen/cprog/frame8.html