STRUCTURES

MODULE 4 – Second Half

INTRODUCTION

- It is not sufficient to handle only all ints, or all floats or all chars at a time.
- when we handle real world data, we deal with entities that are collections of things, each thing having its own attributes, eg: a ‘book’ is a collection of things such as title, author, publisher, number of pages, date of publication, etc.
- In this, author is a string, whereas number of pages is an integer.
- For dealing with such collections, C provides a data type called ‘structure’,
- A structure gathers together, different atoms of information that comprise a given entity.

Y USE STRUCTURES

- we use this to deal with entities that are collection of dissimilar data types.
- Eg: suppose you want to store data about a book. You might want to store its name (a string), its price (a float) and number of pages in it (an int). If data about say 3 such books is to be stored,

  then we can follow two approaches:

(a) Construct individual arrays, one for storing names, another for storing prices and still another for storing number of pages.

(b) Use a structure variable.

- When using arrays

```
Main()
{
char name[3] ;
 float price[3] ;
int pages[3];
printf ( "\nEnter names, prices and no. of pages of 3 books\n" ) ;
for ( i = 0 ; i <= 2 ; i++ )
      scanf ( "%c %f %d", &name[i], &price[i], &pages[i] );
 printf ( "\nAnd this is what you entered\n" ) ;
 for ( i = 0 ; i <= 2 ; i++ )
       printf ( "%c %f %d\n", name[i], price[i], pages[i] );
}
```

- But when we use structures

```
Main()
{
        struct book
        {
                 char name ;
                float price ;
                int pages ;
         } ;
         struct book b1, b2, b3 ;
        printf ( "\nEnter names, prices & no. of pages of 3 books\n" ) ;
        scanf ( "%c %f %d", &b1.name, &b1.price, &b1.pages ) ;
        scanf ( "%c %f %d", &b2.name, &b2.price, &b2.pages ) ;
         scanf ( "%c %f %d", &b3.name, &b3.price, &b3.pages ) ;
         printf ( "\nAnd this is what you entered" ) ;
        printf ( "\n%c %f %d", b1.name, b1.price, b1.pages ) ;
         printf ( "\n%c %f %d", b2.name, b2.price, b2.pages ) ;
        printf ( "\n%c %f %d", b3.name, b3.price, b3.pages ) ;
}
```

- We are going to study two things

(a) declaration of a structure

(b) accessing of structure elements

Declaring a Structure

In our example program, the following statement declares the structure type:

```
 struct book
 {
          char name ;
          float price ;
         int pages ;
} ;
```

This statement defines a new data type called struct book. Each variable of this data type will consist of a character variable called name, a float variable called price and an integer variable called pages.

The general form of a structure declaration statement is given below:

```
struct <structure_name>
{
 structure element 1 ;
 structure element 2 ;
 structure element 3 ;
......
 };
```

Once the new structure data type has been defined one or more variables can be declared to be of that type.

For example the variables b1, b2, b3 can be declared to be of the type struct book, as,

```
struct book b1, b2, b3 ;
struct book
{
       char name ;
      float price ;
       int pages ;
} ;
 struct book b1, b2, b3 ;
```

Is same as

```
struct book
 {
      char name ;
      float price ;
      int pages ;
 } b1, b2, b3 ;
```

points to be noted while declaring a structure type:

- The closing brace in the structure type declaration must be followed by a semicolon.
- A structure type declaration does not tell the compiler to reserve any space in memory.
- Usually structure type declaration appears at the top of the source code file, before any variables or functions are defined.

Accessing Structure Elements

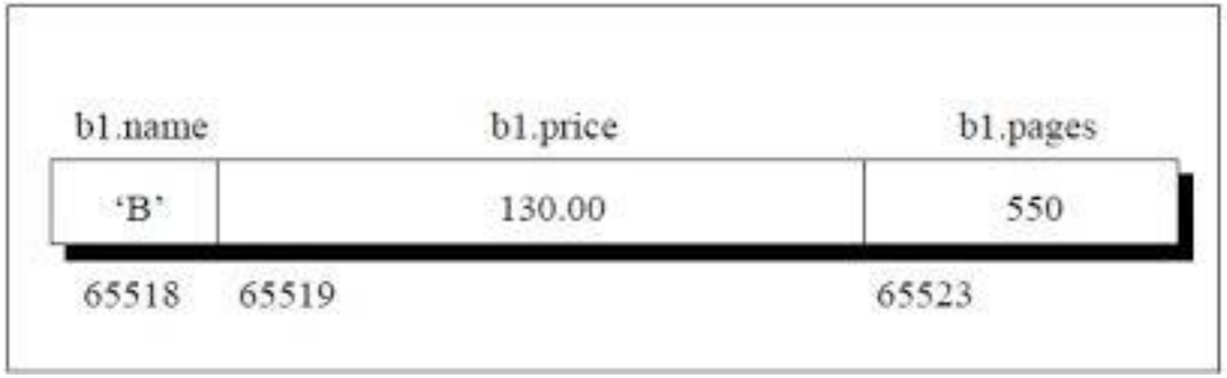
- They use a dot (.) operator to access elements in a structure.
- So to refer to pages of the structure defined in our sample program we have to use, *b1.pages*
- Similarly, to refer to price we would use, *b1.price*

Note that before the dot there must always be a structure variable and after the dot there must always be a structure element.

How Structure Elements are Stored

- Whatever be the elements of a structure, they are always stored in contiguous memory locations.
- struct book

```
{
     char name ;
    float price ;
    int pages ;
} ;
struct book b1 = { 'B', 130.00, 550 } ;     // initializing values for structure elements
```



b1.name
b1.price
b1.pages
'B'
130.00
550
65518
65519
65523