# Structures in C

Rab Nawaz Jadoon

Assistant Professor

COMSATS IIT, Abbottabad

Pakistan

Department of Computer Science

**DCS**

COMSATS Institute of Information Technology

**Introduction to Computer Programming (ICP)**

# Structures in C

- Which mechanic is good enough who knows how to repair only one type of vehicle? None.
  - Same thing is true about C language. It wouldn’t have been so popular had it been able to handle only all ints, or all floats or all chars at a time.
  - In fact when we handle real world data, we don’t usually deal with little atoms of information by themselves—things like integers, characters and such.
    - Instead we deal with entities that are collections of things, each thing having its own attributes, just as the entity we call a ‘book’ is a collection of things such as title, author, call number, publisher, number of pages, date of publication, etc.

# Structures in C

- As you can see all this data is dissimilar, for example 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.

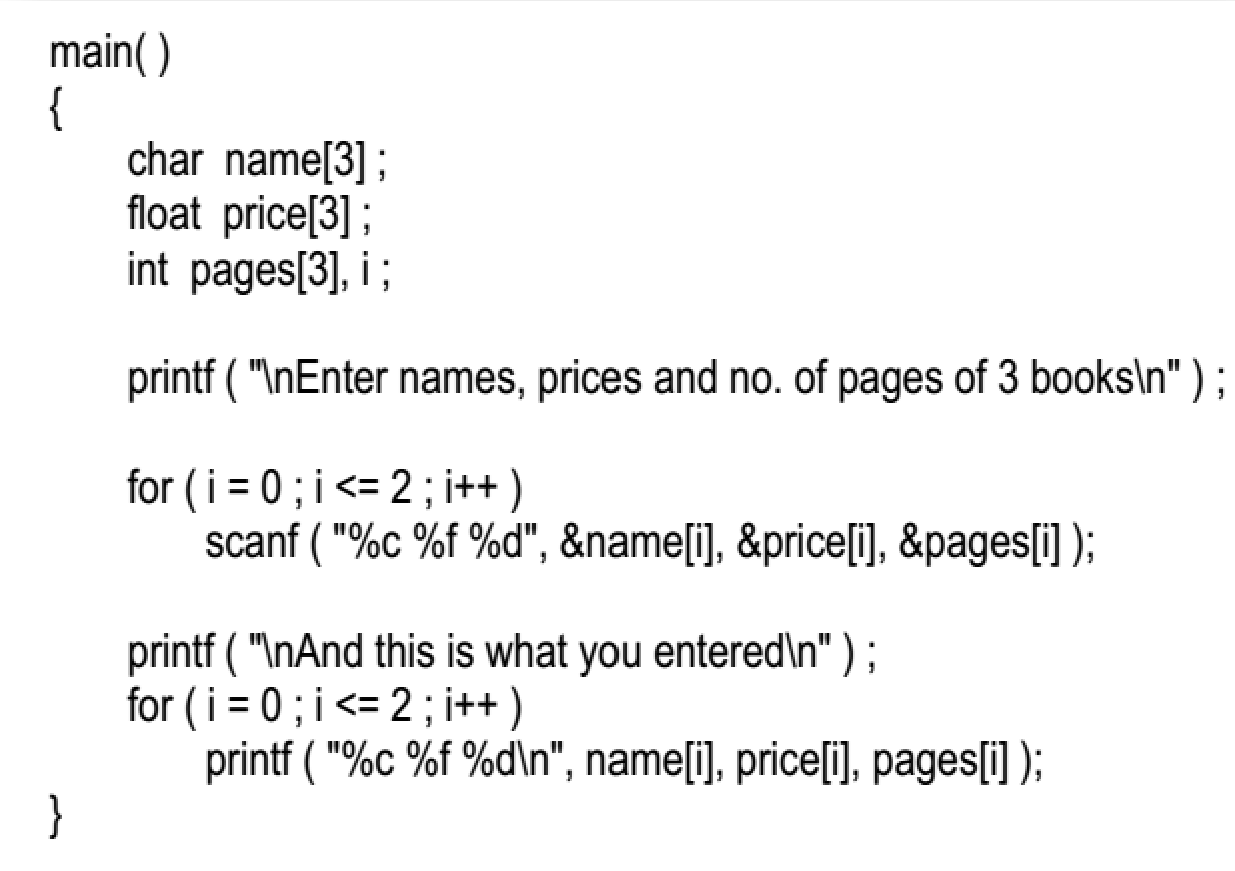
# Why use structures???

- We have seen earlier how ordinary variables can hold one piece of information and how arrays can hold a number of pieces of information of the same data type.
  - These two  data types can handle a great variety of situations.
  - But quite often we deal with entities that are collection of dissimilar data types.

# Example

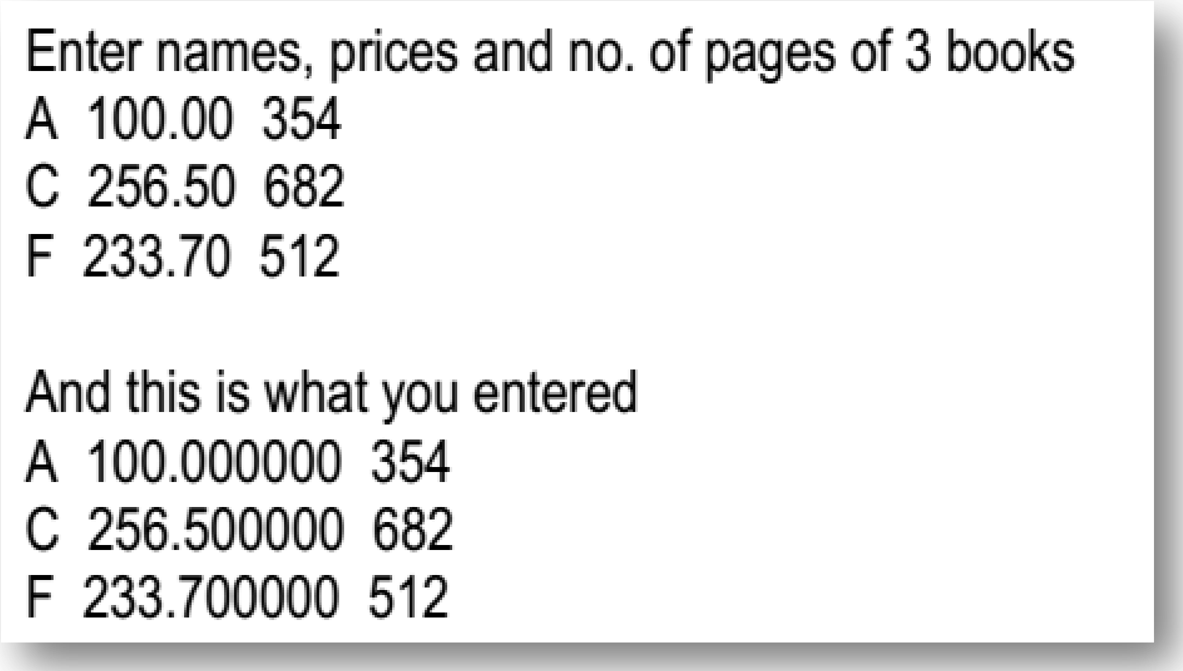
- For example, suppose you want to store data about a book.
  - You might want to store its name(a string), its price (a float) and number of pages in it (an int). If data about say3 such books is to be stored, then we can follow two approaches:
    - Construct individual arrays, one for storing names, another for storing prices and still another for storing number of pages.
    - Use a structure variable.

# Solution 1



```
main( )
{
     char  name[3] ;
     float  price[3] ;
     int  pages[3], i ;

     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] );
}
```

# Output



```
Enter names, prices and no. of pages of 3 books
A  100.00  354
C  256.50  682
F  233.70  512

And this is what you entered
A  100.000000  354
C  256.500000  682
F  233.700000  512
```

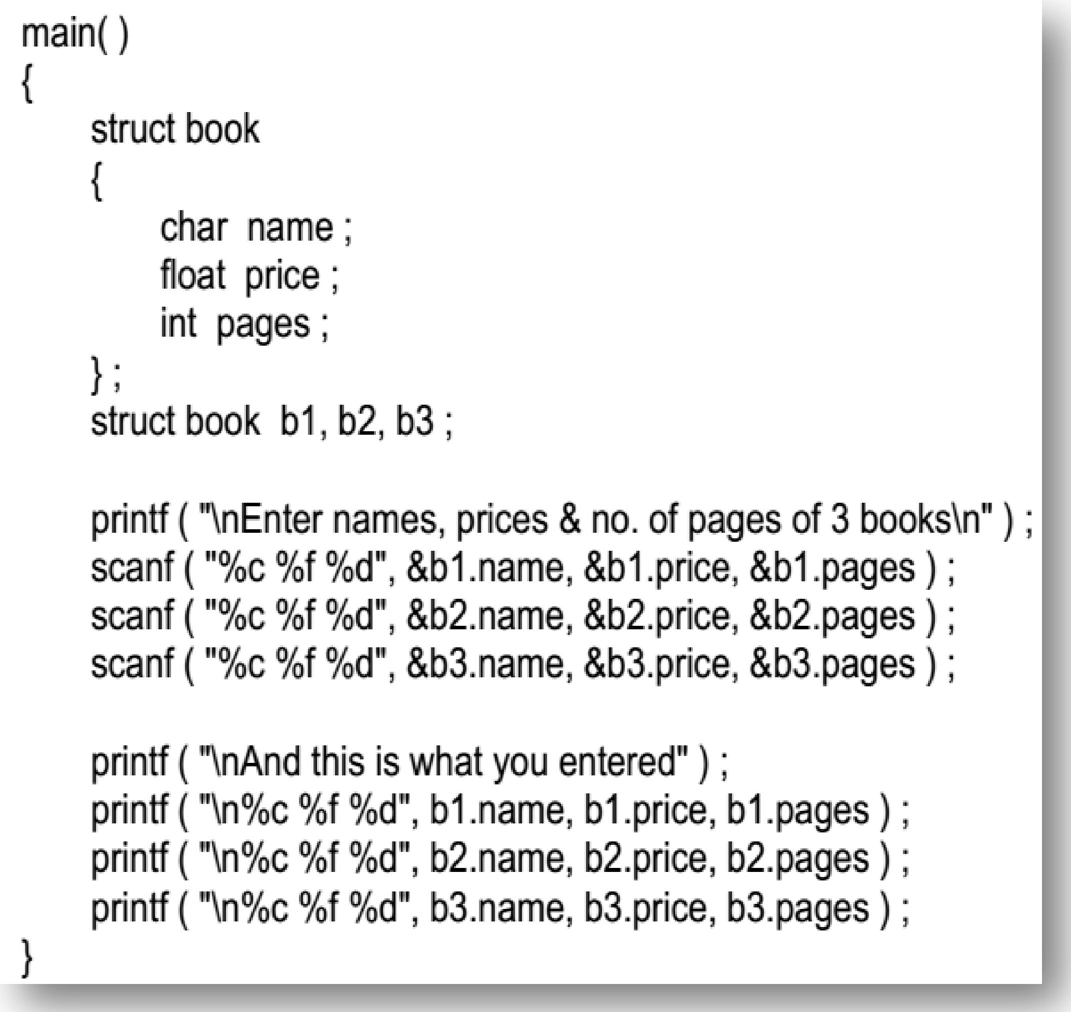
# Structures

- The program becomes more difficult to handle as the number of items relating to the book go on increasing.
  - For example, we would be required to use a number of arrays, if we also decide to store name of the publisher, date of purchase of book, etc.
  - To solve this problem, C provides a special data type—the structure.

# Structures

- A structure contains a number of data types grouped together.
- These data types may or may not be of the same type.
- The following example illustrates the use of this data type.
  - Program at slide 6 can be dealt efficiently using structure on the next slide.

# 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 ) ;
}
```

# Declaration of structures

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

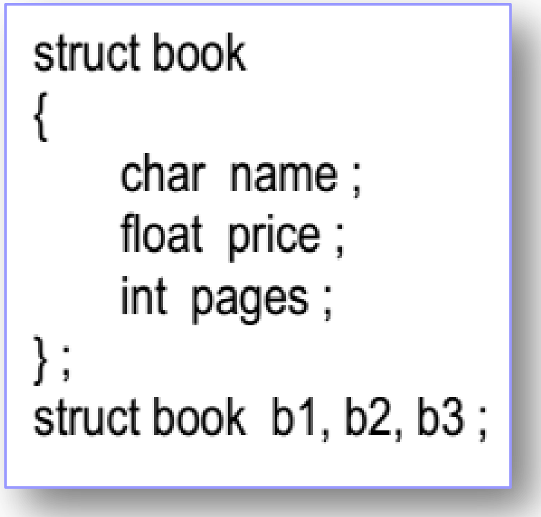
```
struct book
{
  char name;
  float price;
  int pages;
};
```

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.

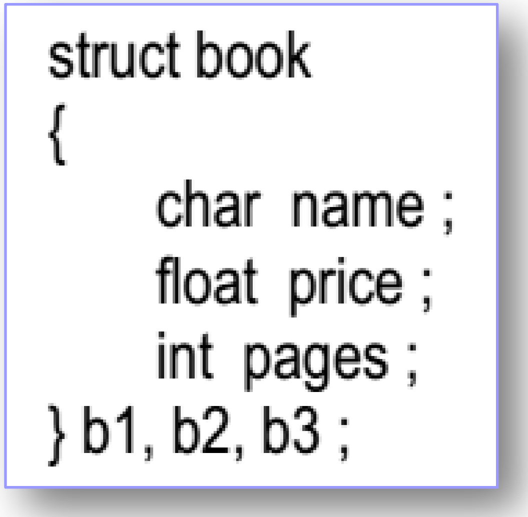
# Declaration of structures

- 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 ; This statement sets aside space in memory.
  - It makes available space to hold all the elements in the structure—in this case, 7 bytes—one for name, four for priceand two for pages.
  - These bytes are always in adjacent memory locations.

# Declaration of structures



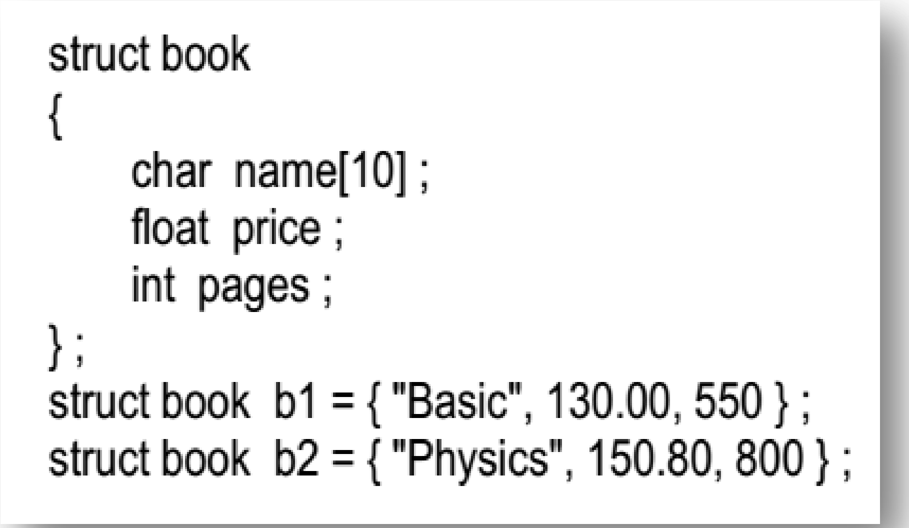
```
struct book
{
        char  name ;
        float  price ;
        int  pages ;
} ;
struct book  b1, b2, b3 ;
```



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

# Declaration of structures

- Like primary variables and arrays, structure variables can also be initialized where they are declared.
  - The format used is quite similar to that used to initiate arrays.



```
struct book
{
        char  name[10] ;
        float  price ;
        int  pages ;
} ;
struct book  b1 = { "Basic", 130.00, 550 } ;
struct book  b2 = { "Physics", 150.80, 800 } ;
```

# Important points

- The closing brace in the structure type declaration must be followed by a semicolon.
- It is important to understand that a structure type declaration does not tell the compiler to reserve any space in memory. All a structure declaration does is, it defines the ‘form’ of the structure.
- Usually structure type declaration appears at the top of the source code file, before any variables or functions are defined.

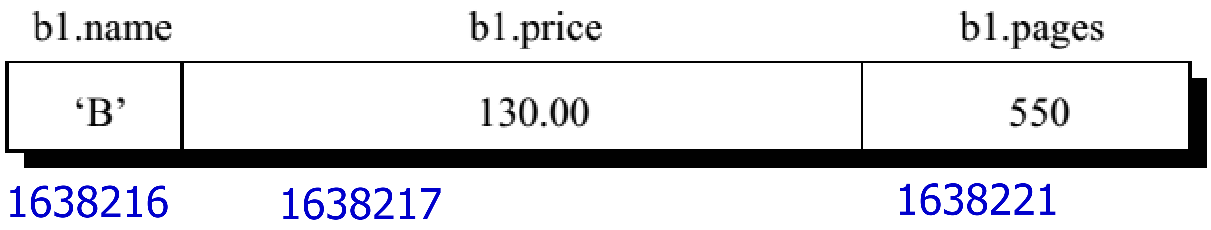
# Accessing structure elements

- In arrays we can access individual elements of an array using a subscript.
- Structures use a different scheme.
  - They use a dot (.) operator.
  - 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?

```
#include<stdio.h>
#include<conio.h>
main( )
{
struct book
{
        char name;
        float price;
        int pages;
};
 struct book b1 = { 'B', 130.00, 550, 1234 };
 printf ( "\nAddress of name = %u", &b1.name );
 printf ( "\nAddress of price = %u", &b1.price);
 printf ( "\nAddress of pages = %u", &b1.pages );
 getche();
}
```

```
Address of name = 1638216
Address of price  = 1638217
Address of pages = 1638221
```



b1.name b1.price b1.pages

‘B’ 130.00 550

1638216 1638217 1638221

# Arrays of structures

- In our sample program, to store data of 100 books we would be required to use 100 different structure variables from b1to b100, which is definitely not very convenient.
- A better approach would be to use an array of structures.
  - Program on next slide shows how to use an array of structures.

# Program

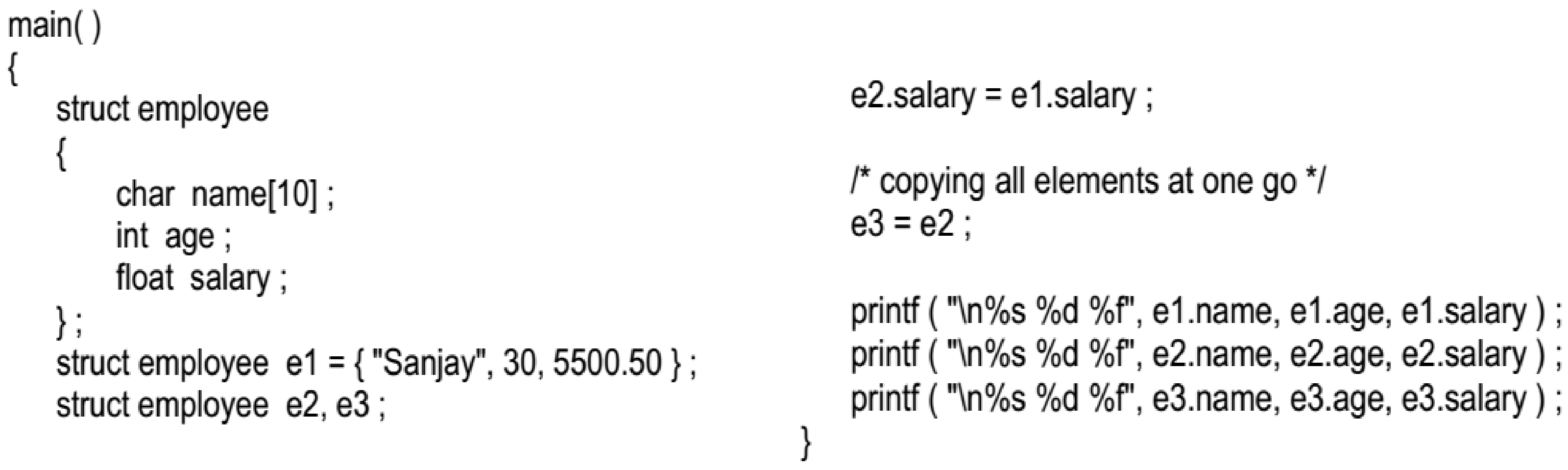
```
#include<stdio.h>
#include<conio.h>
main( )
{
struct book
{
        char name;
        float price;
        int pages;
};
struct book b[3];
int i ;
for ( i = 0 ; i <= 2 ; i++ ){
 printf ( "\nEnter name, price and pages " );
 fflush(stdin);
 scanf ( "%c %f %d", &b[i].name, &b[i].price, &b[i].pages );}
for ( i = 0 ; i <= 2 ; i++ ){
  printf ( "\n%c %f %d", b[i].name, b[i].price, b[i].pages );}
getche();
}
```

Output…

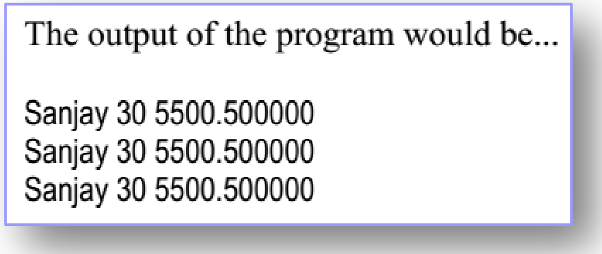
```
Enter name, price and pages a 12.4 123
Enter name, price and pages s 123.8 23
Enter name, price and pages g 23.6 2345

a 12.400000 123
s 123.800003 23
g 23.600000 2345
```

# Additional features

- The values of a structure variable can be assigned to another structure variable of the same type using the assignment operator.



```
main( )
{
    struct employee
    {
        char  name[10] ;
        int  age ;
        float  salary ;
    } ;
    struct employee  e1 = { "Sanjay", 30, 5500.50 } ;
    struct employee  e2, e3 ;

    /* piece-meal copying */
    strcpy ( e2.name, e1.name ) ;
    e2.age = e1.age ;
    e2.salary = e1.salary ;

    /* copying all elements at one go */
    e3 = e2 ;

    printf ( "\n%s %d %f", e1.name, e1.age, e1.salary ) ;
    printf ( "\n%s %d %f", e2.name, e2.age, e2.salary ) ;
    printf ( "\n%s %d %f", e3.name, e3.age, e3.salary ) ;
}
```



The output of the program would be...

```
Sanjay 30 5500.500000
Sanjay 30 5500.500000
Sanjay 30 5500.500000
```

# Nested Structure

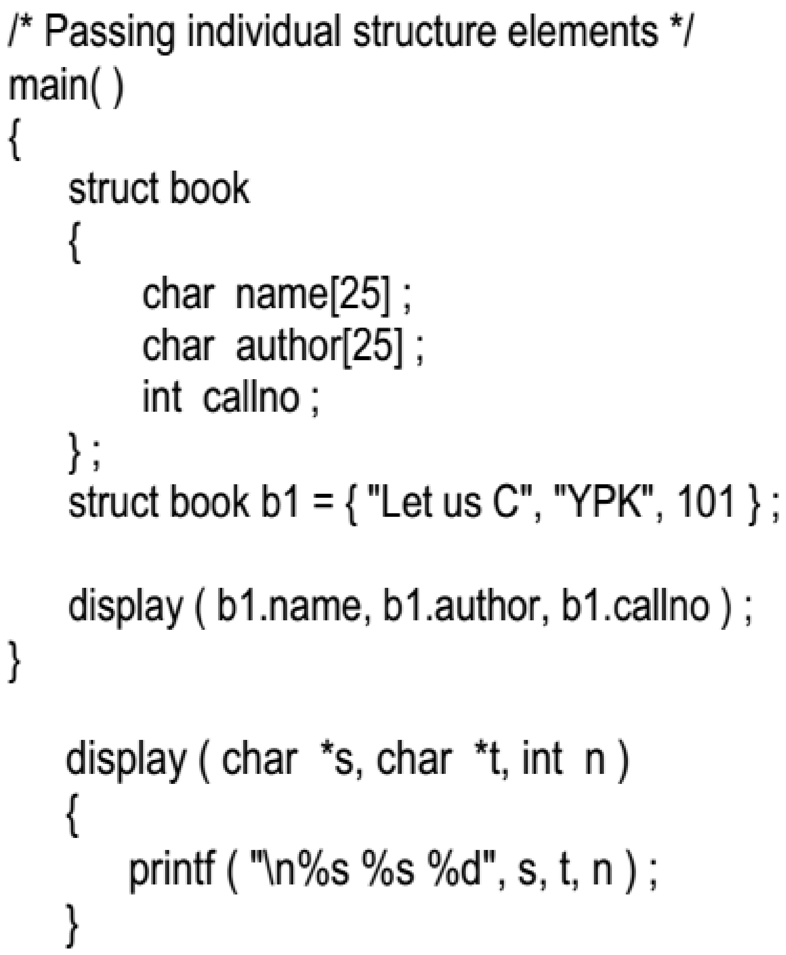
- One structure can be nested within another structure.

```
main( )
{
struct address
{
  char phone[15] ;
  char city[25] ;
  int pin ;
} ;
struct emp
{
  char name[25] ;
  struct address a ;
} ;
 struct emp e = { "jeru", "531046", "nagpur", 10 };
 printf ( "\nname = %s phone = %s", e.name, e.a.phone ) ;
 printf ( "\ncity = %s pin = %d", e.a.city, e.a.pin ) ;
}
```

# structure variable can also be passed to a function

- Like an ordinary variable, a structure variable can also be passed to a function.
  - We may either pass individual structure elements or the entire structure variable at one go.
    - Program on next slide…..

# Program



```
/* Passing individual structure elements */
main( )
{
    struct book
    {
        char  name[25] ;
        char  author[25] ;
        int  callno ;
    } ;
    struct book b1 = { "Let us C", "YPK", 101 } ;

    display ( b1.name, b1.author, b1.callno ) ;
}

    display ( char  *s, char  *t, int  n )
    {
        printf ( "\n%s %s %d", s, t, n ) ;
    }
```

And here is the output...

```
Let us C YPK 101
```

We are passing the base addresses of the arrays name and author, but the value stored in callno.
Thus, this is a mixed call—a call by reference as well as a call by value.

# Passing structure variable to functions

- It can be immediately realized that to pass individual elements would become more tedious as the number of structure elements go on increasing.
  - A better way would be to pass the entire structure variable at a time.
    - Example program on next slide…..

# Program

```
struct book
{
     char  name[25] ;
     char  author[25] ;
     int  callno ;
} ;

main( )
{
     struct book  b1 = { "Let us C", "YPK", 101 } ;
     display ( b1 ) ;
}

display ( struct book  b )
{
     printf ( "\n%s %s %d", b.name, b.author, b.callno ) ;
}
```

And here is the output...

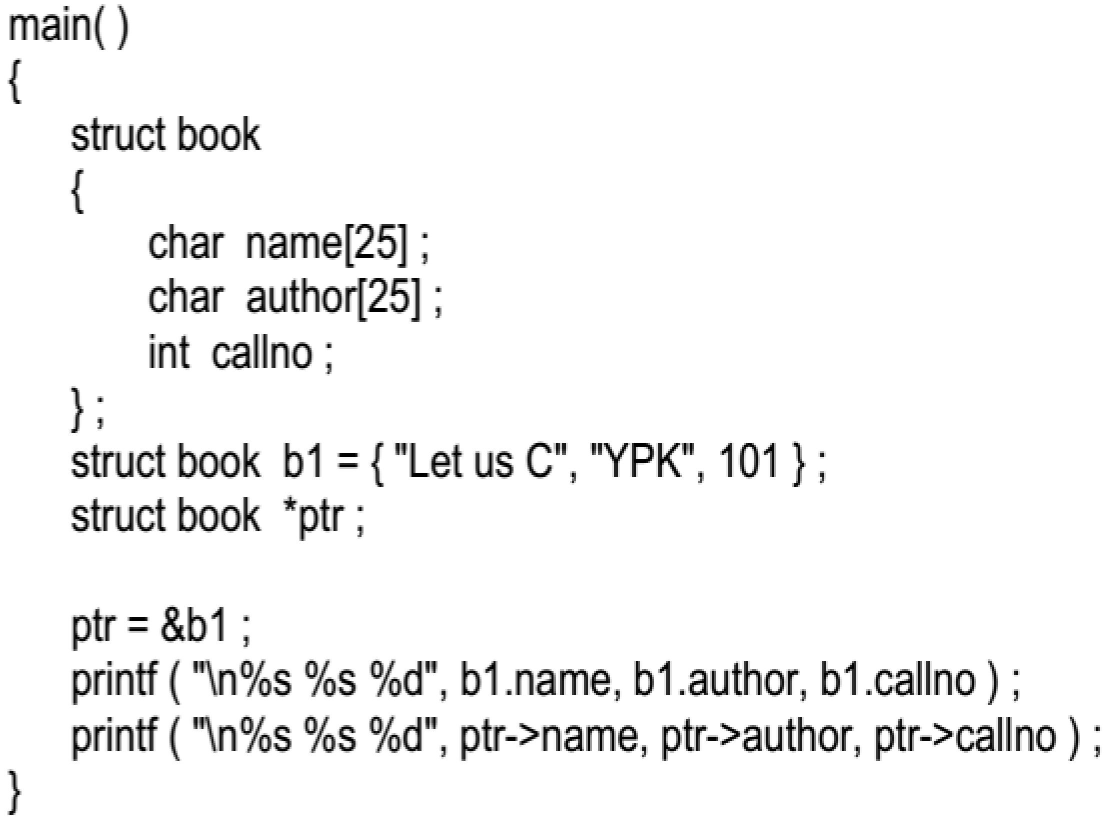
```
Let us C YPK 101
```

As the data type **struct book** is not known to the function display( ).
Therefore, it becomes necessary to define the structure type **struct book** outside main( ), so that it becomes known to all functions in the program.

# Structure pointers

- The way we can have a pointer pointing to an int, or a pointer pointing to a char, similarly we can have a pointer pointing to a struct.
  - Such pointers are known as ‘structure pointers’.

# Program (Structure pointer)



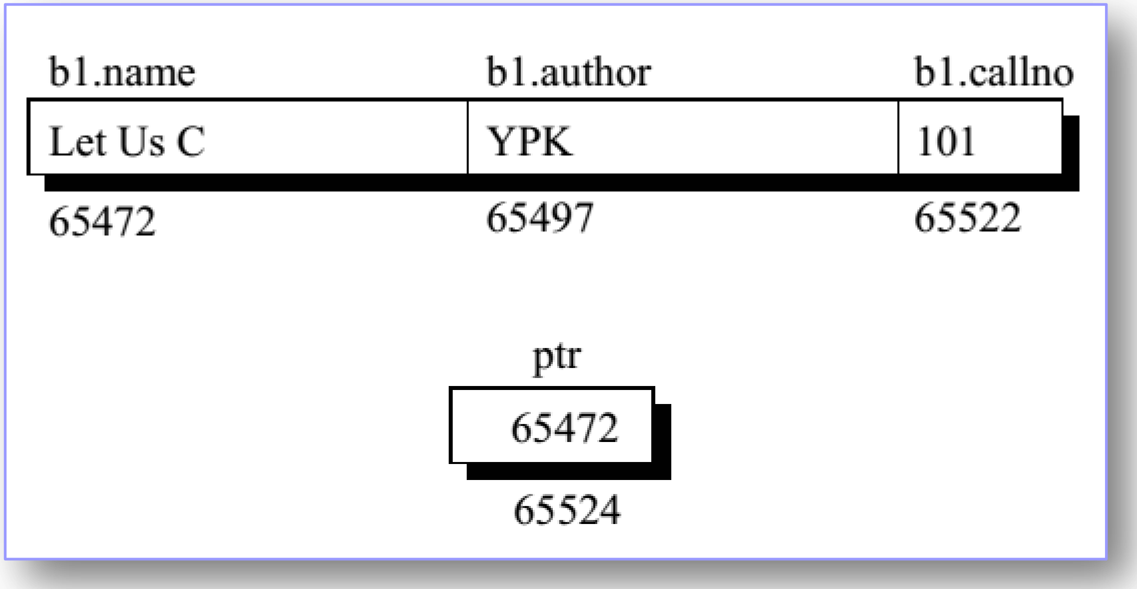
```
main( )
{
     struct book
     {
          char  name[25] ;
          char  author[25] ;
          int  callno ;
     } ;
     struct book  b1 = { "Let us C", "YPK", 101 } ;
     struct book  *ptr ;

     ptr = &b1 ;
     printf ( "\n%s %s %d", b1.name, b1.author, b1.callno ) ;
     printf ( "\n%s %s %d", ptr->name, ptr->author, ptr->callno ) ;
}
```

# Structure pointer

- The first printf ( )is as usual.
  - The second printf( )however is peculiar. We can’t use ptr.name or ptr.callno because ptr is not a structure variable but a pointer to a structure, and the dot operator requires a structure variable on its left.
  - In such cases C provides an **operator ->,** called an arrow operator to refer to the structure elements.

# Memory presentation

- The arrangement of the structure variable and pointer to structure in memory is shown in the Figure below,



| b1.name | b1.author | b1.callno |
|---|---|---|
| Let Us C | YPK | 101 |
| 65472 | 65497 | 65522 |

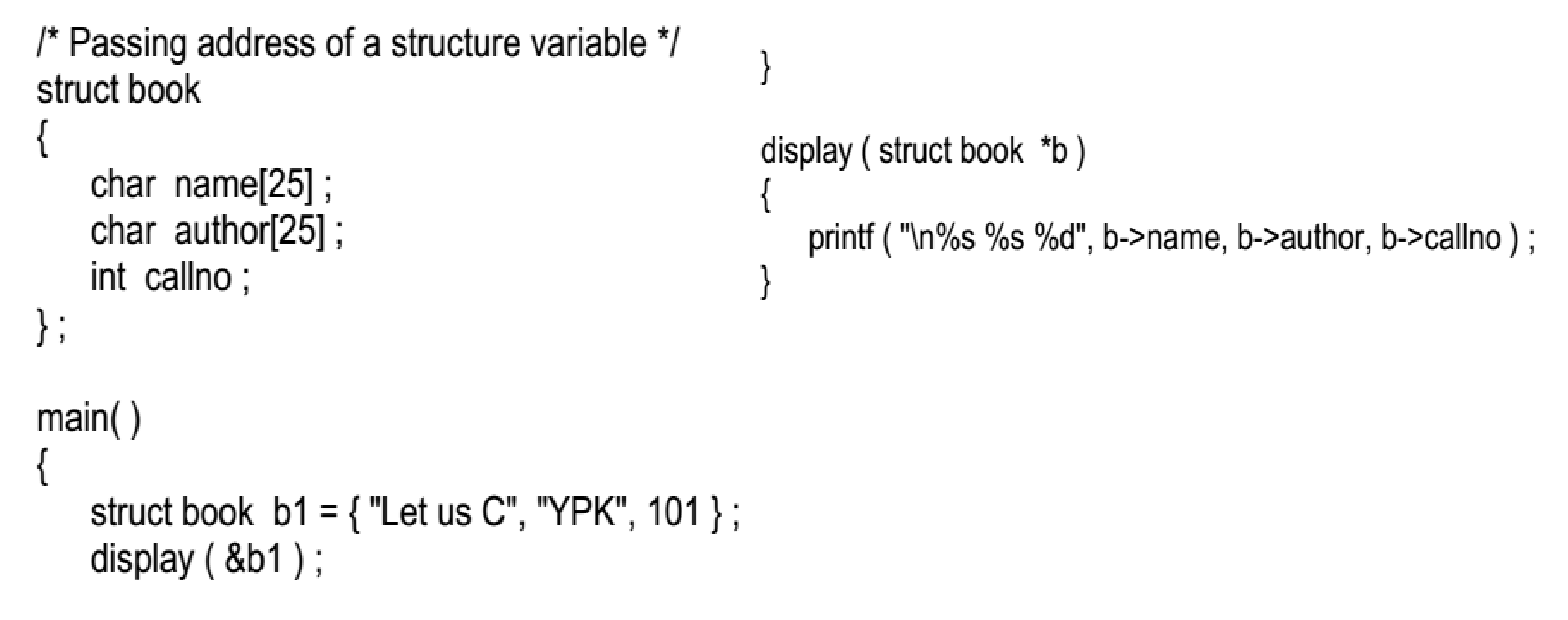
ptr

65472

65524

# Important point

- Can we not pass the address of a structure variable to a function ? We can.
  - The following program demonstrates this.



```
/* Passing address of a structure variable */
struct book
{
     char  name[25] ;
     char  author[25] ;
     int  callno ;
} ;

main( )
{
     struct book  b1 = { "Let us C", "YPK", 101 } ;
     display ( &b1 ) ;
}

display ( struct book  *b )
{
     printf ( "\n%s %s %d", b->name, b->author, b->callno ) ;
}
```

# Summary

A structure is usually used when we wish to store dissimilar data together.

Structure elements can be accessed through a structure variable using a dot (.) operator.

Structure elements can be accessed through a pointer to a structure using the arrow (->) operator.

All elements of one structure variable can be assigned to another structure variable using the assignment (=) operator.

It is possible to pass a structure variable to a function either by value or by address.

It is possible to create an array of structures.

Thank You!