

Unit-II Structure and union

Structure

Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.

“Structure is a collection of heterogenous elements (different data type elements) in which values of different data types can be store into single variable called as structure variable.”

or

Structure is a collection of different types of variables under a single name.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- pages
- price

Declaration of Structure (or Defining a Structure)

Structure variable can be declared using a keyword '**struct**' and structure name.

The general form of the declaration of Structure variable is as follows:-

```
struct    structure_name
{
    data_type  member 1;
    data_type  member 2;
    .
    .
    data_type  member n;
} var1, var2, var3,.....;
```

or

```
struct    structure_name
{
    data_type  member 1;
    data_type  member 2;
    .
    .
    data_type  member n;
};
struct structure_name var1, var2, var3.....
```

At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional.

Here, **struct** = is a **keyword**

structure_name = is the **name of the structure**

data_type = may be any **basic data type** (int, char, float) or **derived data type** (array, pointer)

var1, var2, var3 = are the **structure variables**

member1, member2, ..., member n = these are the fields called as **structure members**.

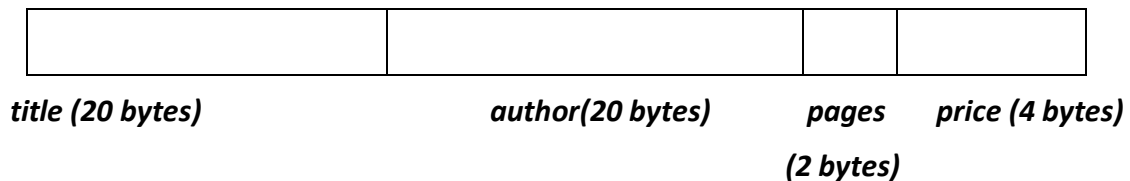
ex. declare the Book structure –

```
struct Books ← structure-name
{
    char title[20];
    char author[20];
    int pages;
    float price;
}
struct Books book1, book2; ← structure-variables
```

structure-members

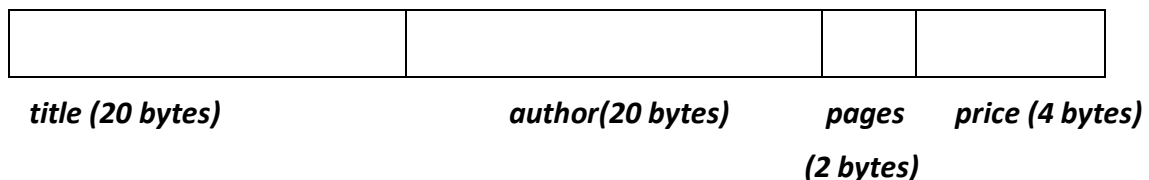
size of book1

(46 bytes)



Size of book2

(46 bytes)



Initialization of Structure variable

(Initialization= Assigning constant value to the variable at the time of declaration)

- “When user want to assign constant value at the time of declaration of structure variable, then it is called as initialization of structure variable.”
- To declare/define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –

```
struct structure_name
{
    data_type member 1;
    data_type member 2;
    .
    .
    data_type member n;
};
struct structure_name var1 = {value1,value2,...value n};
```

At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional.

Here, **struct** = ia a **keyword**

structure_name =is the name of the structure

data_type =may be any **basic data type**(int, char, float)or **derived data type** (array, pointer)

var1, var2, var3= are the **structure variables**

value1, value2,value n = **values** assign to var1, var2,...var n

ex. initializing the Book structure –

```
struct Books
{
    char title[20];
    char author[20];
    int  pages;
    float price;
};
Struct Books book1 = {"C Programming","Yashwant Kanetkar",250,
300.50};
```

initialization of book1

(46 bytes)

C Programming	Yashwant Kanetkar	250	300.50
<i>title (20 bytes)</i>	<i>author(20 bytes)</i>	<i>pages (2 bytes)</i>	<i>price (4 bytes)</i>

Example: C structure

```
#include <stdio.h>
#include <conio.h>
void main( )
{
    struct Books
    {
        char title[20];
        char author[20];
        int  pages;
        float price;
    };

    struct Books Book1={"C Programming","Yashwant Kanetkar",250,
300.50};

    printf( "Book 1 title : %s ", Book1.title);
    printf( "Book 1 author : %s ", Book1.author); /* print Book1 info */
    printf( "Book 1 pages : %d ", Book1.pages);
    printf( "Book 1 price : %f", Book1.price);
    getch( );
}
```

Output:-

```
Book 1 title : C Programming
Book 1 author : Yashwant Kanetkar
Book 1 pages: 250
Book 1 price: 300.50
```

Array of structure

- When user wants to store more than one value into structure variable, then array of structure is used.
- Array variable must want to be declare by using the subscript
- Size enclosed within square bracket to specify the size of the array
- The general form of declaration of array of structure as below:-

```
struct  structure_name
{
    data_type  member 1;
    data_type  member 2;
    .
    .
    data_type  member n;
};
struct structure_name structure_variable[size];
```

↑
Array of structure

Here, **struct** = ia a **keyword**

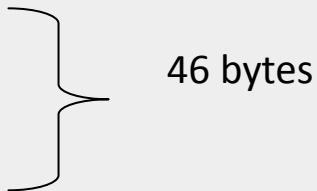
structure_name =is the **name of the structure**

data_type =may be any **basic data type**(int, char, float)or **derived data type** (array, pointer)

structure variable[size]= **aray of structure variable**

ex.

```
struct Books
{
    char title[20];
    char author[20];
    int pages;
    float price;
};
struct Books book[5];
```



In above example “struct Books **book[5]**” it creates five variables and these are:-

book[0] , book[1], book[2], book[3], book[4]

Each variable of size 46 bytes

Example: C structure

```
#include <stdio.h>
#include <conio.h>
void main( )
{
    struct Books
    {
        char title[20];
        char author[20];
        int pages;
        float price;
    };
    struct Books Book[2];
    for(i=0;i<3;i++)
    {
        printf( "Enter title of book ");
```



```
scanf("%s",Book[i].title);
printf( "Book title : %s ", Book[i].title);

printf( "Enter author of book ");
scanf("%s",Book[i].author);

printf( "Book author : %s ", Book[i].author);

printf( "Enter pages of book ");
scanf("%d",Book[i].pages);
printf( "Book pages : %d ", Book[i].pages);

printf( "Enter price of book ");
scanf("%f",Book[i].price);
printf( "Book price : %f", Book[i].price);
}
getch( );
}
```

Output:-

```
Enter title of book   C programming
Book title : C programming

Enter author of book  Yashwant Kanetkar
Book author : Yashwant Kanetkar

Enter pages of book  250
Book pages : 250

Enter price of book  300.50
Book price : 300.50

-----
Enter title of book   C programming
Book title : C programming
```

Enter author of book Balgurusamy
Book author : Balgurusamy

Enter pages of book 350
Book pages : 350

Enter price of book 400.50
Book price : 400.50

How to Access members of a structure?

There are two types of operators used for accessing members of a structure.

1. Member operator/ dot operator(.)

Structure_var . member_name

Ex. book1.title;

book1.price;

2. Pointer to member operator (->)

pointer -> member_name

Ex. ptr->title;

Ptr->price;

Pointers to Structures

- As we store the address of simple variable and array in the pointer, it is possible to store the address of structure_variable in the pointer.

Ex. `int a , *p1;` ←———— a = simple variable, p1 = pointer variable
 `p1 = &a;` ←———— address of 'a' assign to p1

`int b[10] , *p2;` ←———— b = array variable, p2 = pointer variable
 `p2 = &b;` ←———— address of 'b' assign to p2

- Address of pointer variable can be obtained using '&' operator.
- **& (Address of operator)**
- Then the address of structure variable assigned to the pointer variable.
- The data type of pointer and structure variable must be same.

The syntax for declaring pointer to structure is as below:-

```
struct   structure_name   structure_variable;  
struct   structure_name   *pointer_name;  
Pointer_name = &structure_variable;  
  
Pointer_name -> structure_member;  
*Pointer_name . structure_member;
```

Ex.

```
struct Books
{
    char title[20];
    float price;
};
```

declaration of structure Books

```
struct Books book1;
```

```
struct Books * ptr;
```

```
ptr = &book1;
```

```
book1 . title;
```

```
ptr -> title; ← accessing title member using pointer name
```

- In above example, while accessing title member using pointer:-
book1 (structure name) is replaced by ptr (pointer name) and dot operator (.) is replaced by (->) pointer to member operator

book1

title(20 bytes)

price(4 bytes)

<i>C Programming</i>	<i>300.50</i>
----------------------	---------------

Address **2001**-----2020 2021—2024

ptr

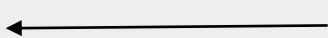

value →


2001

address → 5001

example(pointer to structure)

```
#include <stdio.h>
#include <string.h>
void main( )
{
    struct Books
    {
        char title[20];
        float price;
    };

    struct Books book1 = { "C Programming", 300.50 };
    struct Books *ptr;  declaring pointer
    ptr = &book1;  assigning address of structure variable to pointer

    printf( "Book title = %s ", book1.title);
    printf( "Book title = %s ", ptr -> title);  accessing structure member using pointer and ->
    printf( "price of book  = %f ", book1.price);
    printf( "price of book  = %f ", ptr -> price);
    getch( );
}
```

Output:-

```
Book title = C Programming
Book title = C Programming
price of book  = 300.50
price of book  = 300.50
```

Unit-II

For example: You want to store information about a person: his/her name, citizenship number and salary.

You can create different variables name, citNo and salary to store these information separately.

What if you need to store information of more than one person? Now, you need to create different variables for each information per person: name1,citNo1, salary1 and name2, citNo2, salary2 etc.

A better approach would be to have a collection of all related information under a single name Person structure, and use it for every person.

Nested Structures

You can create structures within a structure in C programming. For example:

```
struct complex
{
    int imag;
    float real;
};
```

```
struct number
{
    struct complex comp;
    int integers;
} num1, num2;
```

Suppose, you want to set imag of num2 variable to 11. Here's how you can do it:

```
num2.comp.imag = 11;
```

Union

Arrays allow to define type of variables that can hold several data items of the same kind.

Similarly union is another user defined data type available in C that allows to combine data items of different kinds.

“Union is a collection of heterogenous elements (different data type elements) in which values of different data types can be store into single variable called as union variable.”

or

union is a collection of variables (can be of different types) under a single name.

But difference between structure and union is that in union all the members shares the comman memory.

(whereas in structure all the members occupy different memory)

Unions are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

- Title
- Author
- pages
- price

Declaration of Union (or Defining a Union)

Union variable can be declared using a keyword 'union' and union_name.

The general form of the declaration of union variable is as follows:-

```
union    union_name
{
    data_type  member 1;
    data_type  member 2;
    .
    .
    data_type  member n;
} var1, var2, var3,.....;
```

or

```
union    union_name
{
    data_type  member 1;
    data_type  member 2;
    .
    .
    data_type  member n;
};
union    union_name  var1, var2, var3.....
```

Unit-II

At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional.

Here, **union** = is a **keyword**

union_name = is the **name of the union**

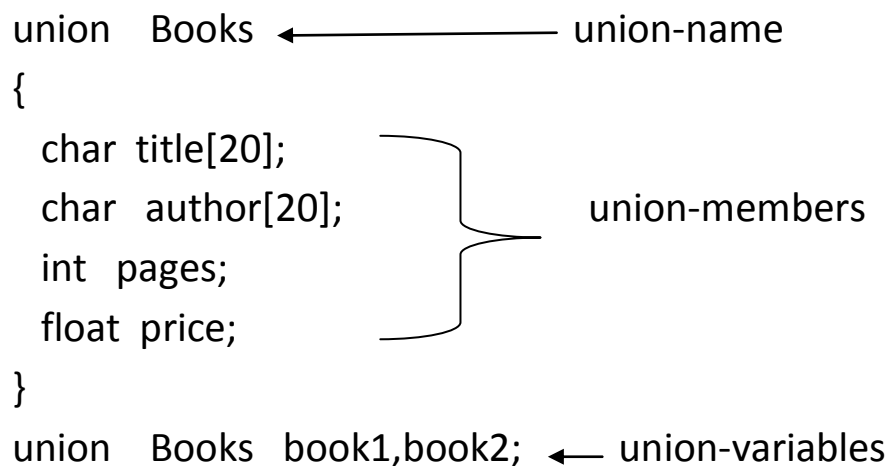
data_type = may be any **basic data type** (int, char, float) or **derived data type** (array, pointer)

var1, var2, var3 = are the **union variables**

member1, member2, ..., member n = these are the fields called as **union_members**.

ex. declare the Book union –

```
union  Books  ← union-name
{
    char title[20];
    char author[20];
    int  pages;
    float price;
}
union  Books  book1,book2; ← union-variables
```



size of union book1

(20bytes)

title	20 bytes
author	20 bytes
Pages	2 bytes
price	4 bytes

Initialization of union variable

(Initialization= Assigning constant value to the variable at the time of declaration)

- “When user want to assign constant value at the time of declaration of union variable, then it is called as initialization of union variable.”
- To declare/define a union, you must use the union statement. The struct statement defines a new data type, with more than one member.
 - There is main difference between structure and union:-
In **structure** all the members **occupy different memory**
In **union** all the members **occupy same memor**
 - **Size of structure variable = sum of size of all members**
 - **Size of union variable = size of member of largest size**

The format of the union is as follows –

```
union union_name
{
    data_type member 1;
    data_type member 2;
    .
    .
    data_type member n;
};
union union var1={value1,value2,....valuen};
```

Unit-II

At the end of the union 's definition, before the final semicolon, you can specify one or more union variables but it is optional.

Here, **union** = is a **keyword**

union _name = is the **name of the union**

data_type = may be any **basic data type**(int, char, float) or **derived data type** (array, pointer)

var1, var2, var3 = are the **union variables**

value1, value2, ..., value n = **values** assign to var1, var2, ..., var n

ex. initializing the Book union –

```
union Books
{
    char title[20];
    char author[20];
    int  pages;
    float price;
};
```

```
union Books book1 = {"C Programming", "Yashwant Kanetkar", 250, 300.50};
```

initialization of book1

size of union book1 **(20bytes)=(size of member of largest size)**

title	<input type="text" value="C Programming"/>
author	<input type="text" value="Yashwant Kanetkar"/>
Pages	<input type="text" value="250"/>
Price	<input type="text" value="300.50"/>

Unit-II

Example: C structure

```
#include <stdio.h>

#include <conio.h>

void main( )
{
    struct Books
    {
        char title[20];
        char author[20];
        int pages;
        float price;
    };
    struct Books Book1={"C Programming","Yashwant Kanetkar",250,
300.50};

    printf( "Book 1 title : %s ", Book1.title);
    printf( "Book 1 author : %s ", Book1.author); /* print Book1 info */
    printf( "Book 1 pages : %d ", Book1.pages);
    printf( "Book 1 price : %f", Book1.price);
    getch( );
}
```

Output:-

Book 1 title : C Programming

Book 1 author : Yashwant Kanetkar

Book 1 pages: 250

Book 1 price: 300.50

Difference between structure and union

	Structure	union
1	In structure 'struct' keyword is used.	In union 'union' keyword is used.
2	In structure each member has separate memory.	In union all the members has common memory, which has largest data type.
3	Due to separate memory of each member, more than one members(all members) can be processed at a time.	Due to common memory, only one member can be processed at a time.
4	Due to separate memory of each member, processing speed of the structure is more as compare to union.	Due to common memory, processing speed of the union is less as compare to union.
5	In structure more memory require for the structure variable. Ex. struct book { char title[20]; int pages; float price; } book1;	In union less memory require for the union variable. Ex. union book { char title[20]; int pages; float price; } book1;
6	book is a structure variable requires 26 bytes to store into memory.	book is a union variable requires 26 bytes to store into memory.

7	<p>The memory representation for structure variable book1 is as follows:-</p> <table border="1"> <tr> <td>title</td><td>pages</td><td>price</td></tr> <tr> <td>20 bytes</td><td>2 bytes</td><td>4 bytes</td></tr> </table>	title	pages	price	20 bytes	2 bytes	4 bytes	<p>The memory representation for union variable book1 is as follows:-</p> <div style="text-align: center;"> <div style="border: 1px solid black; padding: 5px; display: inline-block;">(20 bytes)</div> <div style="text-align: center;"> <div style="width: 100px; border-top: 1px solid black; position: relative; margin: 0 auto;"> ← → </div> <p>Title (20 bytes)</p> </div> <div style="text-align: center; margin-top: 20px;"> <div style="width: 40px; border-top: 1px solid black; position: relative; margin: 0 auto;"> ← → </div> <p>Pages(2 bytes)</p> </div> <div style="text-align: center; margin-top: 20px;"> <div style="width: 40px; border-top: 1px solid black; position: relative; margin: 0 auto;"> ← → </div> <p>price(4 bytes)</p> </div> </div>
title	pages	price						
20 bytes	2 bytes	4 bytes						
8	<p>In structure, memory of the variable is the total bytes of each member.</p>	<p>In union, memory of the union variable is member which has highest memory.</p>						