Introduction

In this lab session, you'll learn about structures, unions and pointers in C.

Structure is the collection of variables of different types under a single name for better handling. For example: You want to store the information about person about his/her name, citizenship number and salary. You can create this information separately but, better approach will be collection of this information under single name because all these information are related to person.

Structure Definition in C

Keyword **struct** is used for creating a structure.

Defining a structure
Syntax of structure

```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeber;
};
```

We can create the structure for a person as mentioned above as:

```
struct person
{
   char name[50];
   int cit_no;
   float salary;
};
```

When a structure is defined, it creates a user-defined type but, no storage is allocated. For the above structure of person, variable can be declared as:

```
struct struct_name  var_name;
struct person
{
   char name[50];
   int cit_no;
   float salary;
};

Inside main function:
struct person p1, p2, p[20]; //declaring variables of person separately
```

Another way of creating sturcture variable is:

```
struct person
{
   char name[50];
   int cit_no;
   float salary;
}p1 ,p2 ,p[20];//declaring variables of person with structure definition
```

P1 and p2 are variables of structure person. However this approach is not much recommended.

In cases, 2 variables p1, p2 and array p having 20 elements of type struct person are created.

Array of Structure

We can also declare an array of structure. Each element of the array representing a structure variable. Example: struct person p[20];
The above code defines an array p of size 20 elements. Each element of array p is of type person.

Accessing structure members

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

1. Member operator (.)
2. Structure pointer operator (will be discussed in pointers)

Any member of a structure can be accessed as: **structure_variable_name.member_name**
Suppose, we want to access salary for variable p2. Then, it can be accessed as: **p1.salary**

We can also use **scanf( )** to give values to structure members through keyboard.

```
scanf("%lf",&p1.salary);
scanf("%d",&p1.city_no);
```

Structure Initialization

How to assign values to struct members? There are two ways

1) Using Dot(.) operator

```
var_name.memeber_name = value;

p2.salary=20000.00;
```

2) Whole structure's variables assignment in one statement

```
struct struct_name var_name =

{value for memeber1, value for memeber2 …so on for all the members}

person = {"Chaitanya", 123, 20000.00};
```

Struct inside another struct

You can use a structure inside another structure, which is fairly possible. As I explained you above that once you declared a structure, the **struct struct_name** would act as a new data type so you can include it in another struct just like a normal data member of any data type. See Below example:

Structure 1:

```
struct stu_addres
{
    int street;
    char state[10];
    char city[15];
    char country[20];
}
```

Structure 2:

```
struct stu_data
{
   int stu_id;
   int stu_age;
   char stu_name[30];
   struct stu_address stuAddress;
}
```

Observe above code – used nested a structure inside another structure. With the above example you would have understood the need also

Assignment for struct inside struct (Nested struct)

Take the above example then assignment of values would happen in this way

```
struct stu_data mydata =
{123, 25, “Chaitanya”, {34,“UP”, “Delhi”, “India”}}
```

Did you notice braces inside braces? Nested values are for nested struct’s members.

How to access nested struct members?
Using chain of “.” operator.

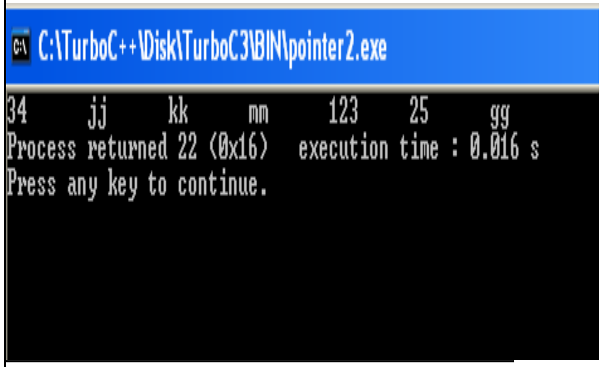
Suppose you want to display the city alone from nested struct

```
printf("%s", mydata.stuAddress.city);
```

Above can be used like this

```
struct_variable.nested_struct_variable.member_name.
```

```
#include<stdio.h>
struct stu_address{
   int street;
   char state[10];
   char city[15];
   char country[20];};
struct stu_data{
   int stu_id;
   int stu_age;
   char stu_name[30];
   struct stu_address stuAddress;} ;
void main(){
   struct stu_data mydata= {123,25,"gg",{34,"jj","kk","mm"}};
   printf("%d-%s\t%s-%s-%d\t-%d\t-%s\t",mydata.stuAddress.street,
        mydata.stuAddress.state,
        mydata.stuAddress.city,
        mydata.stuAddress.country,
        mydata.stu_id,mydata.stu_age,
        mydata.stu_name);
}
```

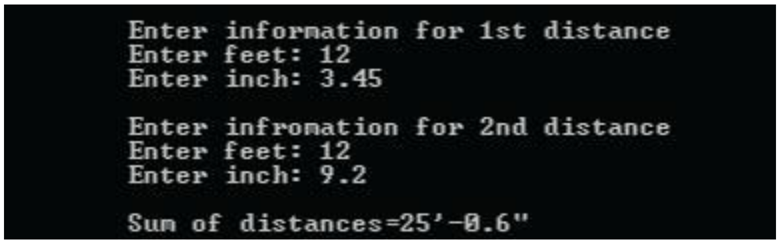

C:\TurboC++\Disk\TurboC3\BIN\pointer2.exe

```
34      jj      kk      mm      123     25      gg
Process returned 22 (0x16)   execution time : 0.016 s
Press any key to continue.
```

Exercises:

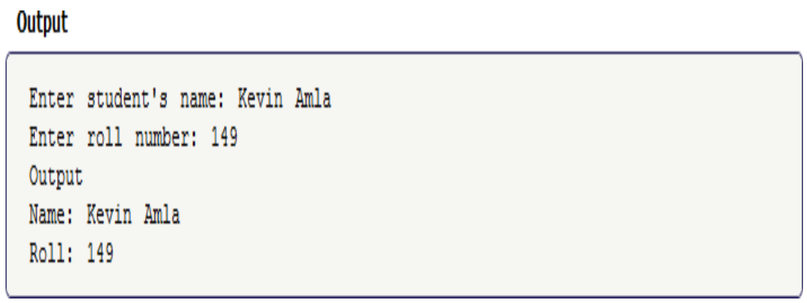
1. Write a program to store information of a student using structure. The program should have the following functionalities
   a. Declaration of structure
   b. Allowing the user to enter the information and store them
   c. Print all the information

2. Write a C program to Add two distances (in inch-feet) using Structures. Your program should be as follows.



```
Enter information for 1st distance
Enter feet: 12
Enter inch: 3.45

Enter infromation for 2nd distance
Enter feet: 12
Enter inch: 9.2

Sum of distances=25'-0.6"
```

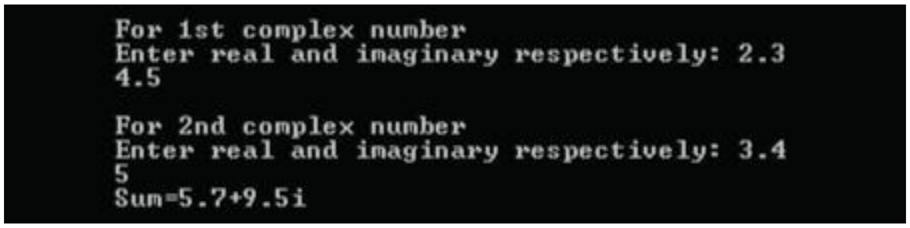
3. Write a C program to create a structure student, containing name and roll. Ask user the name and roll of a student in main function. Pass this structure to a function and display the information in that function. your program should be as follows.



Output

```
Enter student's name: Kevin Amla
Enter roll number: 149
Output
Name: Kevin Amla
Roll: 149
```

4. Write a C program to add two complex numbers by passing structure to a function. Your program should be as follows.



```
For 1st complex number
Enter real and imaginary respectively: 2.3
4.5

For 2nd complex number
Enter real and imaginary respectively: 3.4
5
Sum=5.7+9.5i
```

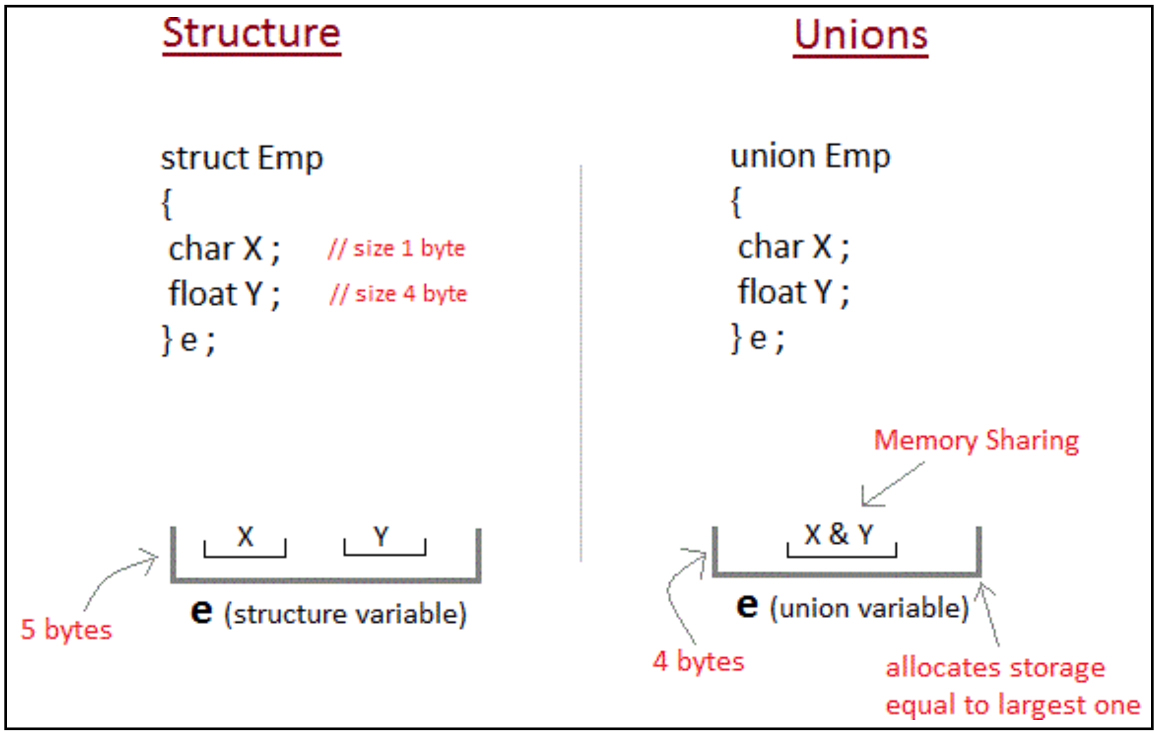
5. Write a C Program to information of 10 students using structures. Your program should be as follows.

Output

```
Enter information of students:

For roll number 1
Enter name: Tom
Enter marks: 98

For roll number 2
Enter name: Jerry
Enter marks: 89
.
.
.
Displaying information of students:

Information for roll number 1:
Name: Tom
Marks: 98
.
.
.
```

Unions in C Language

Unions are conceptually similar to structures. The syntax of union is also similar to that of structure. The only difference is in terms of storage. In structure each member has its own storage location, whereas all members of union use a single shared memory location which is equal to the size of its largest data member.



This implies that although a union may contain many members of different types, it cannot handle all the members at same time. A union is declared using union keyword.

```
union item
{
int m;
float x;
char c;
}It1;
```

This declares a variable **It1** of type union **item**. This **union** contains three members each with a different data type. However only one of them can be used at a time. This is due to the fact that only one location is allocated for a **union** variable, irrespective of its size. The compiler allocates the storage that is large enough to hold largest variable type in the **union**. In the **union** declared above the member **x** requires 4 bytes which is largest among the members in 16-bit machine. Other members of **union** will share the same address.

Accessing a union member

Syntax for accessing **union** member is similar to accessing structure member,

```
union test
{
 int a;
 float b;
 char c;
}t;

t.a ;     //to access members of union t
t.b ;
t.c ;
```

Example for Union

```
#include < stdio.h>
#include < conio.h>

union item
{
 int a;
 float b;
 char ch;
};

int main( )
{
 union item it;
 it.a = 12;
 it.b = 20.2;
 it.ch='z';
 clrscr();
 printf("%d\n",it.a);
 printf("%f\n",it.b);
 printf("%c\n",it.ch);
 getch();
 return 0;
}

Output :
-26426
20.1999
z
```

As you can see here, the values of **a** and **b** get corrupted and only variable **c** prints the expected result. Because in **union**, the only member whose value is currently stored will have the memory.

<u>Difference between structure and union in C:</u>

	C Structure	C Union
1	Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
2	Structure occupies higher memory space.	Union occupies lower memory space over structure.
3	We can access all members of structure at a time.	We can access only one member of union at a time.

4	Structure example:
	struct student
	{
	int mark; char
	name[6];
	double average;
	};
		Union example:
		union student
		{
		int mark; char
		name[6];
		double average;
		};

5	For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B Total memory allocation = 2+6+8 = 16 Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memory allocation = 8 Bytes

Introduction to pointers

Pointers are variables that hold address of another variable of same data type. Pointers are one of the most distinct and excising features of C language. It provides power and flexibility to the language.

Benefit of using pointers

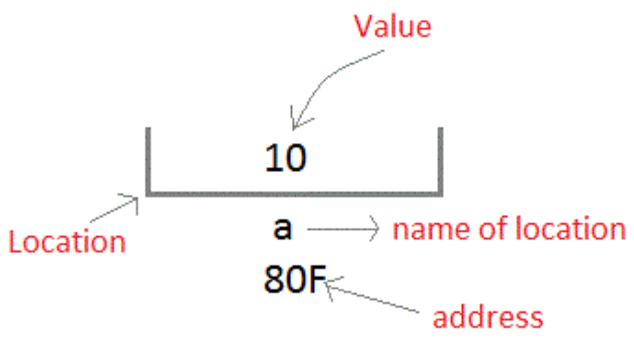
- Pointers are more efficient in handling Array and Structures.
- Pointer allows references to function and thereby helps in passing of function as arguments to other function.
- It reduces length and the program execution time
- It allows C to support dynamic memory management.

Concept of Pointer

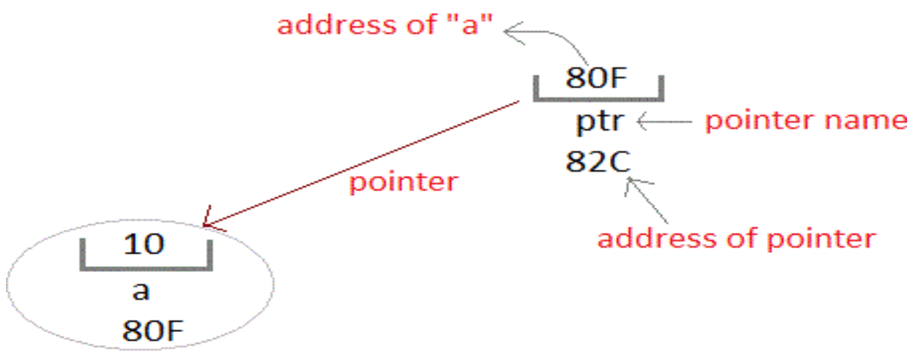
Whenever a variable is declared, system will allocate a location to that variable in the memory, to hold value. This location will have its own address number.

Let us assume that system has allocated memory location 80F for a variable **a**.

```
int a = 10 ;
```



We can access the value 10 by either using the variable name **a** or the address 80F. Since the memory addresses are simply numbers they can be assigned to some other variable. The variable that holds memory address are called **pointer variables**. A **pointer** variable is therefore nothing but a variable that contains an address, which is a location of another variable. Value of **pointer variable** will be stored in another memory location.



Declaring a pointer variable

General syntax of pointer declaration is,

```
data-type *pointer_name;
```

Data type of pointer must be same as the variable, which the pointer is pointing. **void** type pointer works with all data types, but isn't used oftenly.

Initialization of pointer variable

Pointer Initialization is the process of assigning address of a variable to **pointer** variable. Pointer variable contains address of variable of same data type. In C language **address operator** `&` is used to determine the address of a variable. The `&` (immediately preceding a variable name) returns the address of the variable associated with it.

```
int a = 10 ;
int *ptr ;        //pointer declaration
ptr = &a ;        //pointer initialization
or,
int *ptr = &a ;      //initialization and declaration together
```

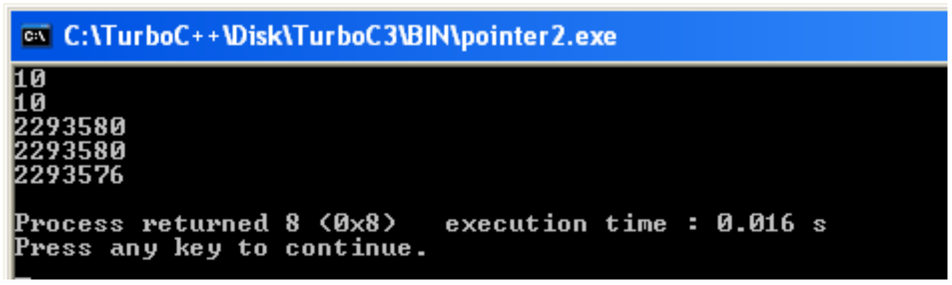
Pointer variable always points to same type of data.

```
float a;
int *ptr;
ptr = &a;    //ERROR, type mismatch
```

Dereferencing of a pointer

Once a pointer has been assigned the address of a variable. To access the value of variable, pointer is dereferenced, using the **indirection operator** `*`.

```
int a,*p;
a = 10;
p = &a;

printf("%d",*p);    //this will print the value of a.

printf("%d",*&a);  //this will also print the value of a.

printf("%u",&a);  //this will print the address of a.

printf("%u",p);  //this will also print the address of a.

printf("%u",&p);  //this will also print the address of p.
```


C:\TurboC++\Disk\TurboC3\BIN\pointer2.exe

```
10
10
2293580
2293580
2293576

Process returned 8 (0x8)   execution time : 0.016 s
Press any key to continue.
```

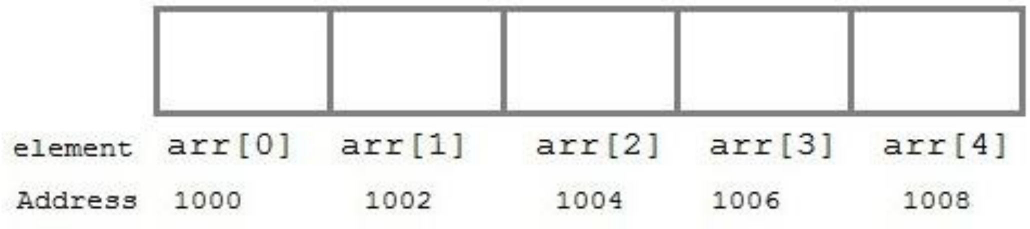
Pointer and Arrays

When an array is declared, compiler allocates sufficient amount of memory to contain all the elements of the array. Base address which gives location of the first element is also allocated by the compiler.

Suppose we declare an array **arr**,

```
int arr[5]={ 1, 2, 3, 4, 5 };
```

Assuming that the base address of **arr** is 1000 and each integer requires two byte, the five element will be stored as follows



Here variable **arr** will give the base address, which is a constant pointer pointing to the element, **arr[0]**. Therefore **arr** is containing the address of **arr[0]** i.e 1000.

```
arr is equal to &arr[0]   // by default
```

We can declare a pointer of type `int` to point to the array **arr**.

```
int *p;
p = arr;
or p = &arr[0];  //both the statements are equivalent.
```

Now we can access every element of array **arr** using **p++** to move from one element to another.

Pointer to array

As studied above, we can use a pointer to point to an Array, and then we can use that pointer to access the array. Lets have an example,

```
int i;
int a[5] = {1, 2, 3, 4, 5};
int *p = a;  // same as int*p = &a[0]
for (i=0; i<5; i++)
{
 printf("%d", *p);
 p++;
}
```

In the aboce program, the pointer ***p** will print all the values stored in the array one by one. We can also use the Base address (**a** in above case) to act as pointer and print all the values.

Replacing the **printf("%d", *p);** statement of above example, with below mentioned statements. Lets see what will be the result.

printf("%d", a[i]); $\longrightarrow$ **prints the array, by incrementing index**

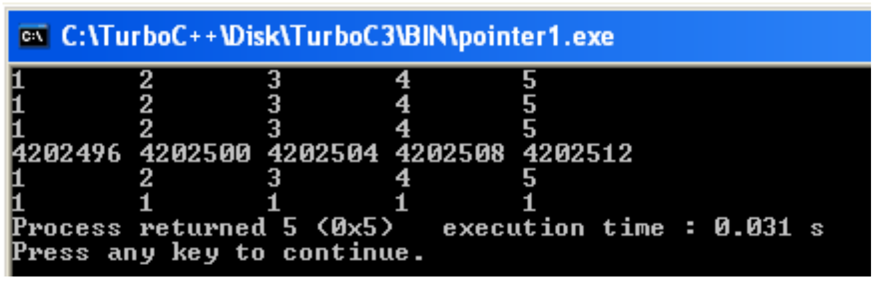
printf("%d", i[a]); $\longrightarrow$ **this will also print elements of array**

printf("%d", a+i); $\longrightarrow$ **This will print address of all the array elements**

printf("%d", *(a+i)); $\longrightarrow$ **Will print value of array element.**

printf("%d", *a); $\longrightarrow$ **will print value of a[0] only**

a++; $\longrightarrow$ **Compile time error, we cannot change base address of the array.**



```
1        2        3        4        5
1        2        3        4        5
1        2        3        4        5
4202496 4202500 4202504 4202508 4202512
1        2        3        4        5
1        1        1        1        1
Process returned 5 (0x5)   execution time : 0.031 s
Press any key to continue.
```

Pointer and Character String

Pointer can also be used to create strings. Pointer variables of **char** type are treated as string.

```
char *str = "Hello";
```

This creates a string and stores its address in the pointer variable **str**. The pointer **str** now points to the first character of the string "Hello". Another important thing to note that string created using **char** pointer can be assigned a value at **runtime**.

```
char *str;
str = "hello";    //Legal
```

The content of the string can be printed using `printf()` and `puts()`.

```
printf("%s", str);
puts(str);
```

Notice that **str** is pointer to the string, it is also name of the string. Therefore we do not need to use indirection operator `*`.

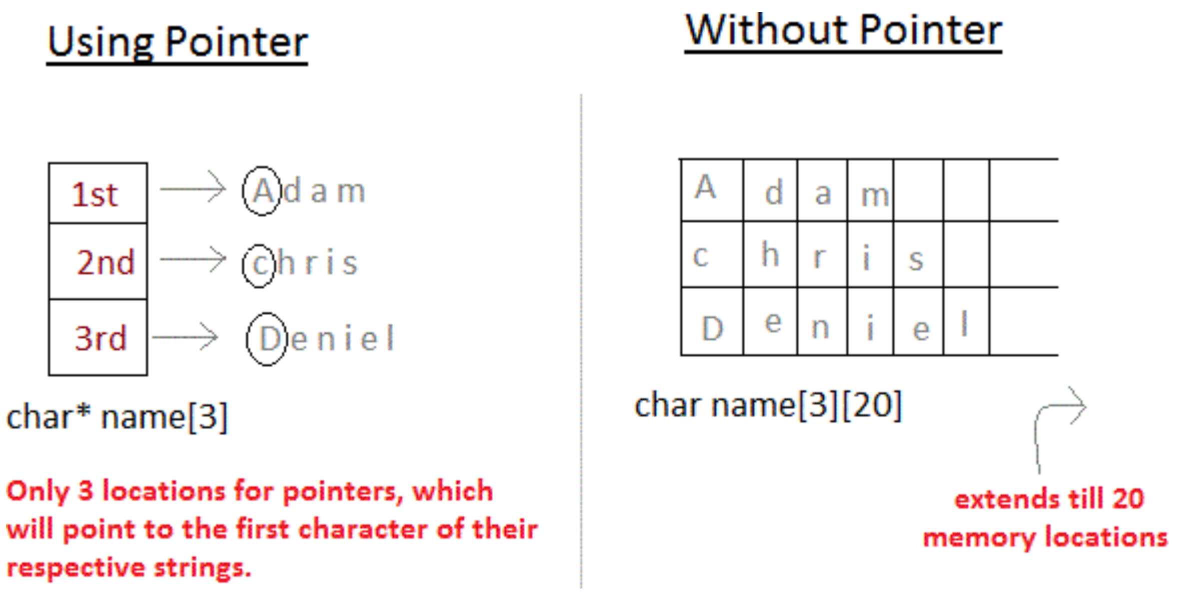
Array of Pointers

We can also have array of pointers. Pointers are very helpful in handling character array with rows of varying length.

```
char *name[3]={
                "Adam",
                "chris",
                "Deniel"
              };
//Now see same array without using pointer
char name[3][20]= {
                    "Adam",
                    "chris",
                    "Deniel"
                  };
```

```
#include<stdio.h>
void main()
{
   int i;
   char *name[3]= {"ss","sds","ff"};
   char name1[3][20]= {"ssw","sdr","ffy"};
   printf(" values of pointer array:");
   for(i=0; i<3; i++)
   {
      printf(" %s\t", name[i] );
   }
   printf(" \n");
   printf(" values of normal array:");
   for(i=0; i<3; i++)
   {
      printf(" %s\t", name1[i] );
   }}
```

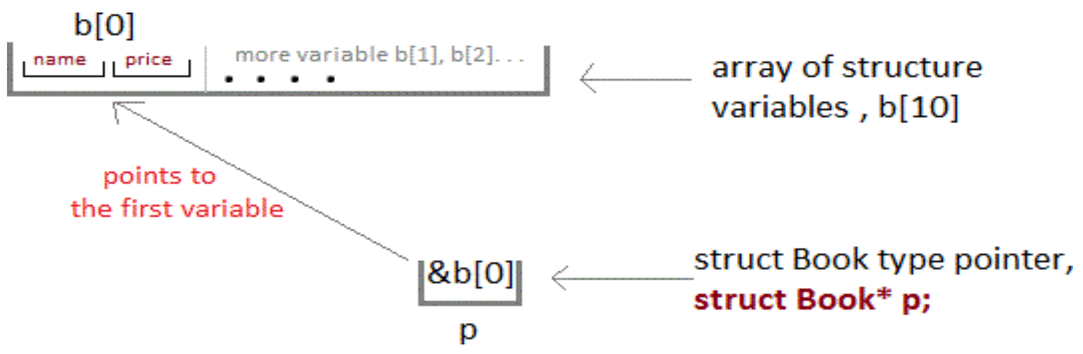
Try this example


Using Pointer
1st
2nd
3rd
Adam
Chris
Deniel
char* name[3]
Only 3 locations for pointers, which will point to the first character of their respective strings.
Without Pointer
A d a m
c h r i s
D e n i e l
char name[3][20]
extends till 20 memory locations

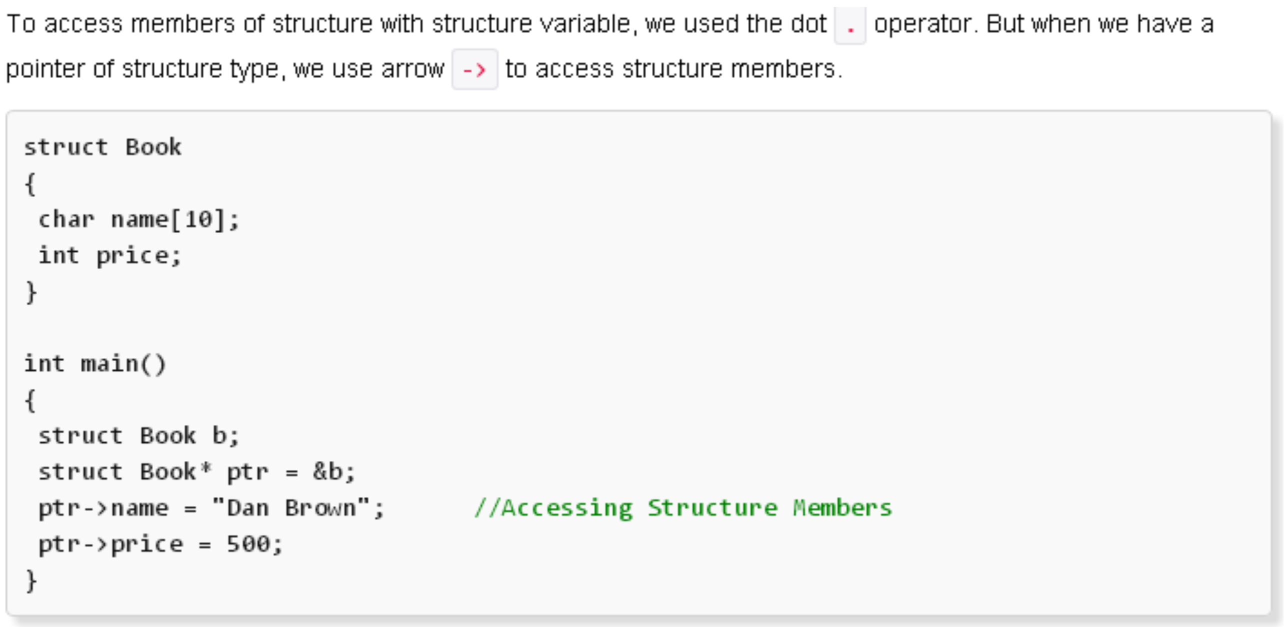
Pointer to Structure

Like we have array of integers, array of pointer etc, we can also have array of structure variables. And to make the use of array of structure variables efficient, we use **pointers of structure type**. We can also have pointer to a single structure variable, but it is mostly used with array of structure variables.

```
struct Book
{
 char name[10];
 int price;
}

int main()
{
 struct Book a;       //Single structure variable
 struct Book* ptr;    //Pointer of Structure type
 ptr = &a;

 struct Book b[10];     //Array of structure variables
 struct Book* p;        //Pointer of Structure type
 p = &b;
}
```


b[0]
name
price
more variable b[1], b[2]. . .
array of structure variables , b[10]
points to the first variable
&b[0]
p
struct Book type pointer,
struct Book* p;

Accessing Structure member with pointers



To access members of structure with structure variable, we used the dot `.` operator. But when we have a pointer of structure type, we use arrow `->` to access structure members.

```
struct Book
{
 char name[10];
 int price;
}

int main()
{
 struct Book b;
 struct Book* ptr = &b;
 ptr->name = "Dan Brown";      //Accessing Structure Members
 ptr->price = 500;
}
```

```
#include <stdio.h>
struct xampl {
  int x;
  int y;
};
int main()
{
    struct xampl structure;
    struct xampl *ptr;
    structure.x = 45;
    structure.x = 78;
    ptr = &structure; /* Yes, you need the & when dealing with
                        structures and using pointers to them*/
    printf( "%d\n%d\n", ptr->x,ptr->x ); /* The -> acts somewhat like the * when
                                 does when it is used with pointers
                                  It says, get whatever is at that memory
                                 address Not "get what that memory address
                                 is"*/
    getch();
return 0;
}
```

typedef

The C programming language provides a keyword called **typedef**, which you can use to give a type a new name. Following is an example to define a term **BYTE** for one-byte numbers:

```
typedef unsigned char BYTE;
```

After this type definitions, the identifier BYTE can be used as an abbreviation for the type **unsigned char, for example:**.

```
BYTE b1, b2;
```

By convention, uppercase letters are used for these definitions to remind the user that the type name is really a symbolic abbreviation, but you can use lowercase, as follows:

```
typedef unsigned char byte;
```

You can use **typedef** to give a name to user defined data type as well. For example you can use typedef with structure to define a new data type and then use that data type to define structure variables directly as follows:

```
#include <stdio.h>
#include <string.h>

typedef struct Books
{
  char  title[50];
  char author[50];
  char subject[100];
  int book_id;
} Book;

int main( )
{
  Book book;

  strcpy( book.title, "C Programming");
  strcpy( book.author, "Nuha Ali");
  strcpy( book.subject, "C Programming Tutorial");
  book.book_id = 6495407;

  printf( "Book title : %s\n", book.title);
  printf( "Book author : %s\n", book.author);
  printf( "Book subject : %s\n", book.subject);
  printf( "Book book_id : %d\n", book.book_id);

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
```

Example 2

```
#include< stdio.h>
#include< conio.h>
#include< string.h>

typedef struct employee
{
 char  name[50];
 int   salary;
} emp ;

void main( )
{
 emp e1;
 printf("\nEnter Employee record\n");
 printf("\nEmployee name\t");
 scanf("%s",e1.name);
 printf("\nEnter Employee salary \t");
 scanf("%d",&e1.salary);
 printf("\nstudent name is %s",e1.name);
 printf("\nroll is %d",e1.salary);
 getch();
}
```

Passing struct to function

It can be done in below 3 ways.

1. Passing structure to a function by value
2. Passing structure to a function by address(reference)
3. No need to pass a structure – Declare structure variable as global

Example program – passing structure to function in C by value:

In this program, the whole structure is passed to another function by value. It means the whole structure is passed to another function with all members and their values. So, this structure can be accessed from called function. This concept is very useful while writing very big programs in C.

```
#include <stdio.h>
#include <string.h>
struct student
{
        int id;
        char name[20];
        float percentage;
};

void func(struct student record);

int main()
{
        struct student record;

        record.id=1;
        strcpy(record.name, "Raju");
        record.percentage = 86.5;

        func(record);
        return 0;
}

void func(struct student record)
{
        printf(" Id is: %d \n", record.id);
        printf(" Name is: %s \n", record.name);
        printf(" Percentage is: %f \n", record.percentage);
}
```

Output:

```
Id is: 1

Name is: Raju

Percentage is: 86.500000
```

Example program – Passing structure to function in C by address:

In this program, the whole structure is passed to another function by address. It means only the address of the structure is passed to another function. The whole structure is not passed to another function with all members and their values. So, this structure can be accessed from called function by its address.

```
#include <stdio.h>
#include <string.h>
struct student
{
          int id;
          char name[20];
          float percentage;
};

void func(struct student *record);

int main()
{
         struct student record;

         record.id=1;
         strcpy(record.name, "Raju");
         record.percentage = 86.5;

         func(&record);
         return 0;
}

void func(struct student *record)
{
         printf(" Id is: %d \n", record->id);
         printf(" Name is: %s \n", record->name);
         printf(" Percentage is: %f \n", record->percentage);
}
```

<u>Output:</u>

```
Id is: 1

Name is: Raju

Percentage is: 86.500000
```

<u>Example program to declare a structure variable as global in C:</u>

Structure variables also can be declared as global variables as we declare other variables in C. So, When a structure variable is declared as global, then it is visible to all the functions in a program. In this scenario, we don’t need to pass the structure to any function separately.

```
#include <stdio.h>
#include <string.h>
struct student
{
        int id;
        char name[20];
        float percentage;
};
struct student record; // Global declaration of structure

void structure_demo();

int main()
{
        record.id=1;
        strcpy(record.name, "Raju");
        record.percentage = 86.5;

        structure_demo();
        return 0;
}

void structure_demo()
{
        printf(" Id is: %d \n", record.id);
        printf(" Name is: %s \n", record.name);
        printf(" Percentage is: %f \n", record.percentage);
}
```

Output:

```
Id is: 1

Name is: Raju

Percentage is: 86.500000
```

Exercises for Unions

01). Develop a program to assign some values to the members a union and to display the same on the screen. (Members of a union include the student name, roll number and marks).

Exercises for Pointers

01). Write a C program to add two integers entered from the keyboard using pointers.

02). Write a C program to find the product of two integers entered from the keyboard using pointers.