UNIT – 4 (Part-2)
POINTERS

BCA SEM – 1

PROBLEM SOLVING METHODOLOGIS AND PROGRAMMING IN C

Code : CS-01

Presented by : Dhruvita Savaliya

Topics :

- Introduction of Pointers
- Use of pointers in Dynamic Programming
- Pointer to Variables
- Pointer to Array
- Pointer within Array
- Array of Pointer
- Pointer To Structure (Unit 5)
- Pointers within structure (Unit 5)
- Pointer to Pointer
- Dangling Pointer Problem

❖ Introduction of Pointers :

- A pointer can be used to store the **memory address** of other variables, functions, or even other pointers.
- In other words, pointer always points an address of another similar type of variable.
- **Some Important Points of pointer :**

1. Pointer is derived data type.
2. Pointer variable is declare using * operator.
3. Pointer's value is always addressed of other variable.
4. Pointer variable is also known as indirection variable.
5. Pointer can store address of same it data type can declare.
6. Pointer variable value address always positive value because of it data type always unsigned integers.
7. You can not store value in pointer variable.

❖ Pointer to Variable :

- The pointer in C language is a variable which stores the address of another variable.
- **Syntax**
- The syntax of pointers is similar to the variable declaration in C, but we use the **(*) dereferencing operator** in the pointer declaration.

  datatype * ***ptr***;

  ptr is the name of the pointer.
- **datatype** is the type of data it is pointing to.
- The above syntax is used to define a pointer to a variable. We can also define pointers to functions, structures, etc.
- **How to Use Pointers?**
- The use of pointers can be divided into three steps:

1. **Pointer Declaration**
2. **Pointer Initialization**
3. **Dereferencing**

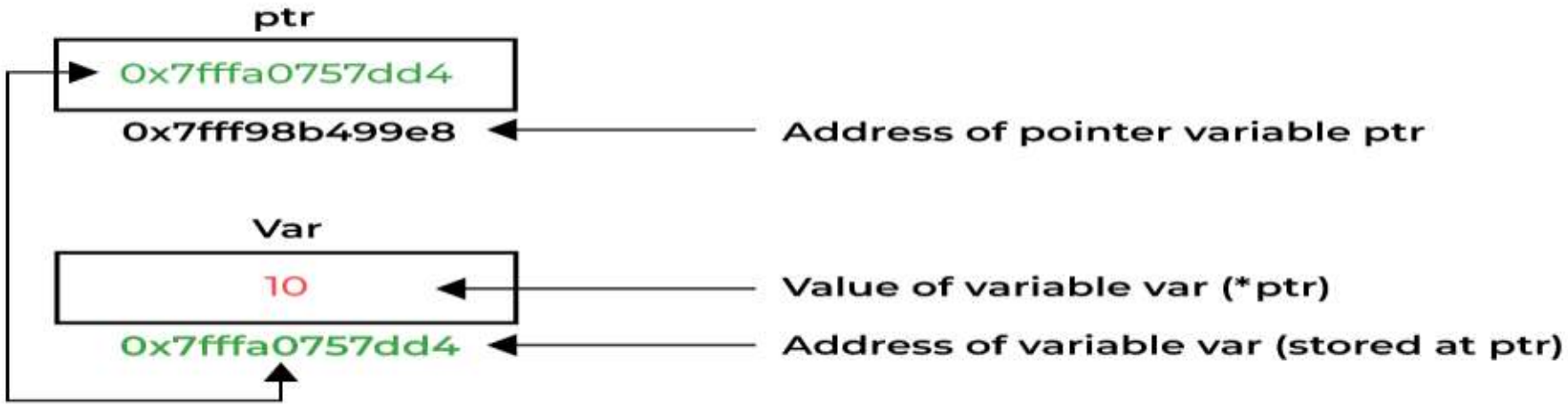
- **1. Pointer Declaration**
- In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the **(*) dereference operator** before its name.
- **Example**

```
int *ptr;
```

- The pointer declared here will point to some random memory address as it is not initialized. Such pointers are called wild pointers.
- **2. Pointer Initialization**
- Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the **(&) addressof operator** to get the memory address of a variable and then store it in the pointer variable.
- **Example**

```
int var = 10;
int * ptr;
ptr = &var;
```

- We can also declare and initialize the pointer in a single step. This method is called **pointer definition** as the pointer is declared and initialized at the same time.
- **Example**
- int ***ptr = &**var;
- **Note:** It is recommended that the pointers should always be initialized to some value before starting using it. Otherwise, it may lead to number of errors.
- **3. Dereferencing**
- Dereferencing a pointer is the process of accessing the value stored in the memory address specified in the pointer. We use the same **(*) dereferencing operator** that we used in the pointer declaration.




```
#include <stdio.h>
void main()
{
    int var = 10;    // declare pointer variable
    int* ptr;
     clrscr();
    // note that data type of ptr and var must be same
    ptr = &var;    // assign the address of a variable to a pointer
    printf("Value at ptr = %p \n", ptr);
    printf("Value at var = %d \n", var);
    printf("Value at *ptr = %d \n", *ptr);
    getch();
}
```

❖ Pointer to Pointer :

- The pointer to a pointer in C is used when we want to store the address of another pointer.
- The first pointer is used to store the address of the variable , And the second pointer is used to store the address of the first pointer.
- That is why they are also known as ***double-pointers***.
- We can use a pointer to a pointer to change the values of normal pointers.
- A double pointer occupies the same amount of space in the memory stack as a normal pointer.

- **Declaration of Pointer to a Pointer in C**
- Declaring Pointer to Pointer is similar to declaring a pointer in C.
- The difference is we have to place an additional '*' before the name of the pointer.
- **Syntax :**

  data_type * *name_of_variable =&pointer_variable;
- **Example :**

```
int val = 5;
int  *ptr = &val; // storing address of val to pointer ptr.
Int  **d_ptr = &ptr; // pointer to a pointer declared
// which is pointing to an integer.
```

Example of Double Pointer :

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=10;
    int *p=&a;        //pointer to variable
    int **q=&p;      //pointer to pointer
    clrscr();
    printf("val : a : %d",a);
    printf("\nadd : a : %p",&a);

    printf("\n\nPrint some data Using *p variable\n");
    printf("\nval : p : %p",p);
    //value of P and address of A
    printf("\nadd : p : %p",&p);
    printf("\nval : a : %d",*p);
    //value of A using *p

    printf("\n\nPrint some data Using **q variable\n");

    printf("\nval : q : %p",q);
    //value of Q and address of P
    printf("\nval : p : %p",*q);
    //value of P and address of P using *q
    printf("\nval : a : %d",**q);
    //value of A using **q

    **q=12;
    printf("\n\nChange value of A Using **q variable\n");
    printf("\nval : a : %d",a);
    //value of A using a
    printf("\nval : a : %d",*p);
    //value of A using *p
    printf("\nval : a : %d",**q);
    //value of A using **q
    getch();
}
```

❖ Pointer to an Array :

- As we know that the pointer is store address of other variable, in this point pointer is can store address of array type of variable.
- An array name is a constant pointer to the first element of the array.
- **Example :**

```
int *p;
int arr[5];
```

- here p is pointer variable that store address of array element.
- arr is array variable that stores 5 integer values.
  - **Lets assign value of p :**

```
p=&arr[0];
```

or

```
p=arr;
```

  - Both are same because arr have base address of array.

- **What is base address ?**
- In array name store base addressed and base address is array index 0's address.

  int arr[5];
- Here arr variable hold the first element address(&arr[0]).

Index Number →	0	1	2	3	4
Value of arr →	**11**	**13**	**54**	**66**	**98**
Address →	-22	-20	-18	-16	-14

- Here base address of arr is -22.

//POINTER TO ARRAY

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[5]={14,22,21,34,35};
   int *q;
   int i;
   q=a;  // q=&a[0];
   both are same
   clrscr();
   printf("\n\n****using a[i]
   ****\n\n");
   for(i=0;i<5;i++)
   {
         printf("\n%d",a[i]);
   }
   printf("\n\n****using *(q+i)
   ****\n\n");
   for(i=0;i<5;i++)
   {
         printf("\n%d",*(q+i));
   //because q is store add of a[i]
   }
   getch();
}
```

❖ Array of Pointer & array within Pointer :

- Pointer array is a homogeneous collection of indexed pointer variables that are references to a memory location.
- It is generally used in C Programming when we want to point at multiple memory locations of a similar data type in our C program.
- We can access the data by dereferencing the pointer pointing to it.
- **Syntax :**

  pointer_type *array_name [array_size];
- **pointer_type:** Type of data the pointer is pointing to.
- **array_name:** Name of the array of pointers.
- **array_size:** Size of the array of pointers.

```
#include <stdio.h>
#include<conio.h>
void main()
{
   int  var1 = 10 , var2 = 20 , var3 = 30 , i;
   int* ptr_arr[3];
   clrscr();
   ptr_arr[0]=&var1;
   ptr_arr[1]=&var2;
   ptr_arr[2]=&var3;
   // traversing using loop
   for (i = 0; i < 3; i++)
   {
         printf("Value of var%d: %d\tAddress: %p\n", i + 1, *ptr_arr[i],
   ptr_arr[i]);
   }
   getch();
}
```

❖ Dangling Pointer Problem :

- A dangling pointer in C is a pointer that points to a memory location that has been deallocated or is no longer valid.
- Dangling pointers can cause various problems in a program, including segmentation faults, memory leaks, and unpredictable behavior.
- One common cause of dangling pointers is using the free function to deallocate memory that was previously allocated using the malloc function.
- When the free function is called on a pointer, it deallocates the memory pointed to by the pointer, making it available for reuse.
- However, if the pointer is not set to NULL or reassigned to a different memory location after the memory is deallocated, it becomes a dangling pointer.

```
#include<conio.h>
#include<stdio.h>
int * demo();
void main()
{
    int *a;
    clrscr();
    printf("%d %d\n",a,&a);
    a=demo();
    printf("\n\n%d %d\n",a,*a);
    getch();
}
```

```
int * demo()
{
    int d=30;
    printf("\n%d %d\n",d,&d);
    return &d;
    // d is now dangling
}
```

Lets have another Example :

```
int *ptr = malloc(sizeof(int));
*ptr = 10;
free(ptr);
*ptr = 20;  // ptr is now a
dangling pointer
```