Pointers in C Language

MEMORY AND ITS REFERENCE

- Each variable created in **memory** has a unique location in memory known as its **Address.**
- A variable declaration reserves specific amount of space in memory.
- When a variable is declared it associates these three Attributes
  - Variable **name**
  - Variable **data type**
  - Memory **address**

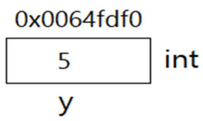
Example:

```
int y;
```

- name of variable is **y**
- type is **int**
- address of variable is ***unknown***

Memory address is hexadecimal value, that refer to a particular location

- Variable name directly refer to the value stored at memory location.
- `y = 5;`



- Box represents the memory location allocated for the variable **y**
- The hexadecimal value above the box is its assumed address
- Actual address can be displayed by using **Reference operator &** .
  - It is also known as **Address operator**

Program-1:

```
/* this program shows the address of a variable
using address operator*/

#include<stdio.h>

int main()
{
    int a=10;
    printf("Value of a is = %d\n",a);
    printf("Address of a in HEX is = %X\n",&a);

}
```

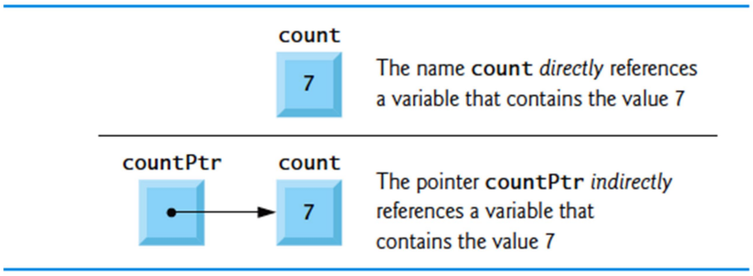
Output of program-1:

```
Value of a is = 10
Address of a in HEX is = 62FE4C

--------------------------------
```

POINTERS

- **Pointers** are variables whose values are *memory addresses*.
- A pointer stores an *address* of a variable.
- **Reference operator (&)** is used to access the memory address of a variable and store it in a pointer
- A variable name *directly* references a value, and a pointer *indirectly* references a value.
- Referencing a value through a **pointer** is called **`indirection.`**



Directly and indirectly referencing a variable.

DECLARING A POINTER

- Declaring a pointer is same as declaring a simple variable
- An asterisk * is used in the declaration that indicate that the variable is a pointer variable.

Syntax:	Examples:
DataType * Var;	int *countPtr; float *p1,*p2;

*** indicates in declaration** statement that the variable being defined is a **pointer.**
*It is preferred to include the letters **Ptr** in pointer variable names to make it clear that these variables are pointers and thus need to be handled appropriately.*

POINTER OPERATORS

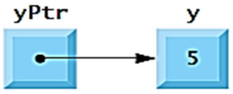
1. The &, or **Address Operator or Reference Operator**
2. The unary * operator, commonly referred to as the **Indirection operator** or **Dereferencing Operator**

The &, or **address operator**, is a unary operator that returns the *address* of its operand. For example,

```
int y = 5;
int *yPtr;
The following statement:
yPtr = &y;
```

Assigns the *address* of the variable **y** to pointer variable **yPtr**. Variable **yPtr** is then said to "point to" y.

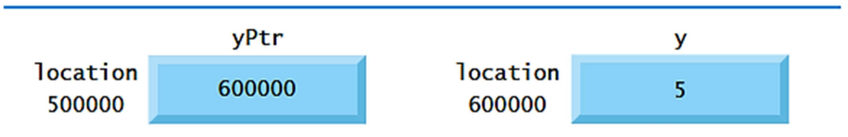
Following Figure shows a schematic representation of memory.



Graphical representation of a pointer pointing to an integer variable in memory.

Pointer Representation in Memory

Following Figure shows the representation of the pointer in memory, assuming that integer variable **y** is stored at location 600000, and pointer variable **yPtr** is stored at location 500000.
The operand of the address operator must be a variable; the address operator *cannot* be applied to constants or expressions.



Representation of y and yPtr in memory.

Program-2:

```c
/* this program shows the address of a variable
using pointer variable */

#include<stdio.h>

int main()
{
    int a=10;
    int *ptr;     //pointer variable
    ptr=&a;      // giving address of a to ptr
    printf("Value of a is = %d\n",a);
    printf("Address of a in HEX is = %X\n",ptr);
}
```

Output of program-2:

```
Value of a is = 10
Address of a in HEX is = 62FE44

--------------------------------
```

*The Indirection * Operator / Dereference Operator*

The unary * operator, commonly referred to as the **Indirection operator** or **Dereferencing operator**.

- The **dereference operator** is used to access the value of the variable whose memory address is stored in pointer.
- It is denoted by *
- it is also called **Indirection operator.**

For example, the statements

```c
int y = 5;
int *yPtr;
yPtr = &y;

printf( "%d", *yPtr );
```

Prints the value of variable **y**, that is **5**.

****

Program-3:

```c
/* this program shows the value  of variable  a,
using direct methods and using indirection operartor */

#include<stdio.h>

int main()
{
    int a=10;
    int *ptr;   //pointer variable
    ptr=&a;     // giving address of a to ptr
    printf("Value of a is = %d\n",a);    // display the value of a
    printf("value of a is = %d\n",*ptr); // display value of a using indirection opertor *
}
```

Output of program-3:

```
Value of a is = 10
value of a is = 10

-----------------------
```

DEMONSTRATING THE & AND * OPERATORS

- Following program's demonstrates the pointer operators & and *.

Program-4:

```
/* this program input and output using pointer
dereference or indirection operator
*/

#include<stdio.h>

int main()
{
    int a;
    int *ptr;   //pointer variable
    ptr=&a;     // giving address of a to ptr
    printf("Enter Value of a:: ");
    scanf("%d",& *ptr);

    printf("value of a is = %d\n",*ptr); // using indirection * opertor
}
```

Output of program-4:

```
Enter Value of a:: 7
value of a is = 7

------------------------
```

****

Program-5:

```
/* this program multiply the values of a and b and
store result in c using
indirection operartor */

#include<stdio.h>

int main()
{
    int a=5,b=6,c;
    int *aPtr,*bPtr,*cPtr;
    aPtr=&a;
    bPtr=&b;
    cPtr=&c;
    *cPtr=*aPtr * *bPtr;     // multiplying a and b
    printf("Value of a * b is = %d\n",c);

}
```

Output of program-5:

```
Value of a * b is = 30

------------------------
```

THE 'VOID' POINTER

- The data type of pointer variable and the data type of variable it refers must be same.
- **void pointer** can store the address of any type of variable.
- The keyword 'void' is used as the data type of the pointer as follows:

Example :

```
void *ptr;
```

The above statement declares a pointer variable **ptr**. The data type of the pointer is **void**. It means that it can store the memory address of any type of variable.

Program-6:

```
#include <stdio.h>

int main() {
    int n = 10;
    float f= 25.18;
    char c = '$';
    void *ptr;
    ptr = &n;
    printf("The value of n:%d\n",n);
    printf("The address of n: %x\n",ptr);
    ptr = &f;
    printf("The value of f:%f\n",f);
    printf("The address of f: %x\n",ptr);
    ptr = &c;
    printf("The value of c:%c\n",c);
    printf("The address of c: %x\n",ptr);
 }
```

Output of program-6:

```
The value of n:10
The address of n: 62fe44
The value of f:25.180000
The address of f: 62fe40
The value of c:$
The address of c: 62fe3f

--------------------------------
```

POINTER INITIALIZATION

The process of assigning a memory address to a pointer at the time of declaration is called point initialization.

Pointer variable is initialized so that it may not point to anything invalid. The pointer can be initialized to any valid memory address. It can also be initialized to a NULL or 0 value.

Syntax:

```
DataType *P = &Variable;
```

Example:

```
int n = 100;
int *p1 = &n;
int *p2 = NULL;
```

The pointer p1 is initialized to memory address of variable n. The pointer p2 is initialized to NULL.

- A pointer can access all elements of an array if the address of first element is assigned to it.
- The name of array represents the address of its first element.
- The address of first element can be assigned to a pointer by assigning the name of the array to pointer.

ACCESSING ARRAY ELEMENTS WITH POINTERS

- The array elements can be accessed with pointers by moving the pointer to the desired element.
- The contents of an array elements can be accessed using dereference operator *.
- Pointer reference can be moved forward and backward by using increment operator ++ and decrement operators --

Program-7:

```
/* this program shows the value of Array  arr,
using pointer   */

#include<stdio.h>

int main()
{
    int arr[]={15,24,30,40,20};
    int *ptr;
    ptr=arr;     //this will assign the addresss of first element to ptr
    for(int i=0;i<5;i++)
    {
    printf("%d\n",*ptr++);
    }

}
```

Output of program-7:

```
15
24
30
40
20

--------
```

POINTER ADDITION AND SUBTRACTION

Program-8:

```
/* this program initilize array and
perform increment operation on pointer variable
*/

#include<stdio.h>

int main()
{
    int arr[]={2,3,4,5,6};
    int *ptr;                       //pointer variable
    ptr=arr;                        // assigning ddress of arr to ptr

    printf("value is = %d\n",*ptr); // print value at index 0
    ptr++;
    printf("value is = %d\n",*ptr);  // print value at index 1
    ptr=ptr+2;
    printf("value is = %d\n",*ptr);  // print value at index 3

}
```

Program-9:

```
/* this program shows the value of Array  arr,
using pointer   */

#include<stdio.h>

int main()
{
     int arr[]={15,24,30,40,20};
     int *ptr;
     ptr=arr;

     printf("%d\n",*(ptr+3));

}
```

The first line of **program-9** declares an array **arr** and initializes it with five values. The second line declares a pointer **ptr** . Third line assigns the address of first element of the array to **ptr** . The printf() statement displays the value of 4th element that is **40.**

So the Array elements can be accessed without moving the pointer permanently

****

Program-10:

```
/* this program shows the value of Array  arr,
using pointer   */

#include<stdio.h>

int main()
{
     int arr[]={15,24,30,40,20};
     int *ptr;
     ptr=&arr[2];  // assign the address of 3rd element

     printf("%d\n",*ptr);

}
```

Third line of **program-10** now **assigns** the address of 3rd element of the array to **ptr** . The **printf()** statement displays the value of 3rd element that is **30.**

****

Program-11:

```
/* this program shows the value of Array  arr,
in reverse order using pointer   */

#include<stdio.h>

int main()
{
    int arr[]={15,24,30,40,20};
    int *ptr;
    ptr=&arr[4];     //this will assign the addresss of first element to ptr
    for(int i=0;i<5;i++)
    {
    printf("%d\n",*ptr--);
    }

}
```

- A string is and array of characters
- A pointer of char can refer to a string

EXAMPLE:

```
char str[] = "Hello";
char *ptr = str;
```

The first statement declares and initializes a string. The second statement declares a pointer variable and initializes it to string name.

```
char *ptr = "Hello";
```

This can also be done the string is stored in and unnamed memory location.

Program-12:

```
/* this program displays string using pointer   */

#include<stdio.h>
int main()
{
    char str[]="Hello";
    char *ptr;
    ptr=str;  //this will assign the addresss of string to ptr
    printf("%s",ptr);
}
```

****

Program-13:

```
/* this program displays the first element
of string using pointer   */

#include<stdio.h>

int main()
{
    char str[]="Hello";
    char *ptr;
    ptr=str;  //this will assign the addresss of string to ptr
    printf("%c",*ptr);
}
```

Output of program-13:

The **program-13** will display the first character **H**

****

Program-14:

```
/* this program displays string
one character each time using pointer   */

#include<stdio.h>

int main()
{
    char str[]="Hello";
    char *ptr;
    ptr=str;  //this will assign the addresss of string to ptr
    while (*ptr!= '\0')
    {
    printf("%c",*ptr);
    ptr++;
    }
}
```

Output of program-14:

The **program-14** will display the complete string **Hello ,** one character each time until **NULL** character is reached .

****

Program-15:

```
/* this program displays string
in Upper case   */

#include<stdio.h>
#include<ctype.h>

int main()
{
     char str[]="Hello";
     char *ptr;
     ptr=str;  //this will assign the addresss of string to ptr
     while (*ptr!= '\0')
     {
     printf("%c",toupper(*ptr));
     ptr++;
     }
}
```

Output of program-15:

The **program-15** will display the string **HELLO, i.e. in Upper case**.

****

Program-16:

Write a Program that declares and initializes a string .it inputs a character from the user and searches the character in the string.

```
/* this program find a character from Str
by using pointers */

#include<stdio.h>
#include<conio.h>
int main()
{
 char st[]="HELLO";
 char ch;
 char s='n';
 char *ptr;
 ptr=st;
 printf("WHAT TO FIND::");
 scanf("%c",&ch);
 while (*ptr != '\0')
 {
 if(*ptr==ch)
 s='y';
 ptr++;
 }
 if (s=='y')
 printf("found");
 else
 printf("NOT found");
}
```

Output of program-16:

```
WHAT TO FIND::L
found
--------------------
```

Output of program-16:

```
WHAT TO FIND::A
NOT found
-----------------
```

The pointers can refer to a structure variable in the same way as simple variables. However, the dot operator is not used to access the members of a structure when pointers are used to refer to a structure variable. The **selection operator ->** is used instead of **dot operator** to access the structure member. It consists of minus sign and greater than sign.

Program-17:

```
/* this program uses structures an pointers
 */

#include<stdio.h>
#include<conio.h>
struct book
{
     char at[20];
     int pg,pr;
};
int main()
{
     book rec,*ptr;
     ptr=&rec;
     printf("Enter Name of Author::");
     gets(ptr->at);
     printf("Enter Book pages::");
     scanf("%d",&ptr->pg);
     printf("Enter Book price::");
     scanf("%d",&ptr->pr);

     printf("BOOK DATA \n");
     printf("Name of Author:: %s\n",ptr->at);
     printf("Book Pages:: %d\n",ptr->pg);
     printf("Book Price:: %d\n",ptr->pr);

}
```

Output of program-17:

```
Enter Name of Author::Kamran Akhtar
Enter Book pages::500
Enter Book price::850
BOOK DATA
Name of Author:: Kamran Akhtar
Book Pages:: 500
Book Price:: 850

--------------------------------
```

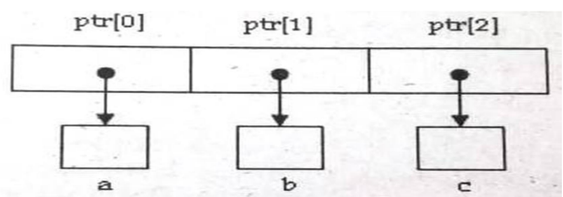
ARRAY OF POINTERS

An array of pointers is an array in which each element is a pointer. Each element in the array can store a memory address. The array can store the memory addresses of different objects of same type. **Program-18:**

```
#include<stdio.h>
int main()
{
    int *ptr[3],a,b,c;
    int i;
    ptr[0] = &a;
    ptr[1] = &b;
    ptr[2] = &c;
    printf("Enter three integers: ");
    scanf("%d%d%d",&a,&b,&c);
    printf("You Entered these integers: ");
    for(int i=0;i<3;i++)
    printf("%d\n",*ptr[i]);
}
```

How Program-18: Works?

The program declares an array of integer pointers and three integer variables. Each element of pointer array refers to different variable. The program inputs the values in the variables and displays them using array of pointers.



Review Questions

Q.1. What is reference operator?

In C **&** is a reference operator. It is also known as address operator. The reference operator is used to access the memory address of a variable,

Q2. Why are pointer used?

Pointers are used in programs to access memory and manipulate addresses. They can access and manipulate data in computer memory directly.

Q.3. Write any Two advantages of using pointer.

Pointer can access the memory address directly. It can save memory.

Q.4. How is a pointer variable declared?

The method of declaring a pointer variable is same as declaring a simple variable. The difference is that an **asterisk** is used in the declaration. It indicates that the variable is a pointer variable that can hold an address. The data type of a pointer must be same as the data type of the variable whose memory address is to be stored in the pointer.

Q.5. How is an address assigned to a pointer?

The memory address is assigned to a pointer by using the reference or address operator. It is denoted by ampersand (&) symbol.

Q6. What type of pointer can store the address of any type of variable?

The void pointer is a type of pointer that can store the address of any type of variable. The type of pointer variable and the type of variable it refers must be same, but A pointer variable can store the address of any type of variable if it is declared as **void.**

Q.7. How can you access the value of the variable referenced by a pointer?

The dereference operator is used to access the value of the variable whose memory address is stored in pointer. It is denoted by asterisk. It is also called **indirection operator**. It can also be used to input a value in the variable and process the data stored in the variable.

Q.8. What is the difference between int* q = p; and n = *p; ?

int* q = p; The value of p will be assigned to pointer q which will be treated as an address.

int n=*p; In this an integer variable n is initialized with the value stored at the address to which the pointer **p** points.

Q.9. What type of operations performed on pointers?

Only addition and subtraction operations can be performed on pointers. The arithmetic operations on pointers work differently than normal integer data types. The effect of both addition and subtraction depends on the size of the data type of the pointer.

Q.10. How can a pointer be used with arrays?

An array is a collection of many elements of same type. All elements of the array are stored in consecutive memory locations. A pointer can access all elements of an array if the address of first element is assigned to it. The name of array represents the address of its first element. The address of first element can be assigned to a pointer by assigning the name of the array to pointer.

Q.11. How an array element is accessed with pointer?

The array elements can be accessed with pointers by moving the pointer to the desired element.

The contents of an array elements can be accessed using dereference operator. The pointer reference can be moved forward and backward by using increment operator ++ and decrement operators --

Q.12. How can a pointer be used with structures?

The pointers can refer to a structure variable in the same way as simple variables. However, the dot operator is not used to access the members of a structure when pointers are used to refer to a structure variable. The **selection operator** -> is used instead of **dot operator** to access the structure member. It consists of minus sign and greater than sign.