Pointers

- A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.

- Like any variable or constant, you must declare a pointer before using it to store any variable address.

- The general form of a pointer variable declaration is −

- type *var-name;

- Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable.

- The asterisk * used to declare a pointer is the same asterisk used for multiplication.

- However, in this statement the asterisk is being used to designate a variable as a pointer.

- int a;   int a=10;
- int *p;   int *p;
- p=&a;   p=&a;
- printf(“%d”,*p);printf(“%d %d”,*p,a);

```
#include <stdio.h>
 int main()
{
 int  num = 10;
printf("Value of variable num is: %d", num);

/* To print the address of a variable we use %p * format specifier and
ampersand (&) sign just * before the variable name like &num. */

printf("\nAddress of variable num is: %p", &num);
return 0;
 }
```

Output:

```
Value of variable num is: 10
Address of variable num is: 0x7fff5694dc58
```

Declaration of pointer variable

- int *a;   // a pointer to integer
- float *b;   // a pointer to float
- char *c;   // a pointer to character
- double *d;   // a pointer to double

Operations performed on pointers

- Assigning a pointer to a var of type pointer

- int a=10;
- int *pa=&a;
- int *p;
- p=pa

- Two or more pointers can point to same memory locations

Arithmetic operations on pointers

- #include<stdio.h>
- void main()
- {
- int a=10 ,b=20;
- int *pa=&a;
- int *pb=&b;
- int x= *pa + *pb;
- int y= *pa - *pb;
- int z= *pa * *pb;
- printf(“%d+%d=%d”,*pa,*pb,x);
- printf(“%d-%d=%d”,*pa,*pb,y);
- printf(“%d*%d=%d”,*pa,*pb,z);
- }

NULL pointer

- A NULL pointer is defined as a special pointer value that points to '\0' in the memory.
- It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned.
- This is done at the time of variable declaration.
- A pointer that is assigned NULL is called a **null** pointer.
- The NULL pointer is a constant with a value of zero defined in several standard libraries

```
#include<stdio.h>

int *p=NULL;
```

- **Here pointer var p is a null pointer.**

- **This indicates that the pointer var p does not point to any part of the memory**
- **If(p==NULL)**
- **printf("p does not point to any memory");**
- **else**
- **{**
- **printf("Access the value of p");**
- **}**

Using one pointer for many variables

- #include<stdio.h>
- Int main()
- {
- Int a;
- Int b;
- Int c;
- Int *p;
- Printf(“Enter three nums”);
- Scanf(“%d%d%d”,&a,&b,&c);
- P=&a;
- Printf(“%d”,*p)
- P=&b;
- Printf(“%d”,*p)
- P=&c
- Printf(“%d”,*p)
- Return 0;
- }

Using a variable with many pointers

- #include<stdio.h>
- Int main()
- {
- Int a;
- Int *p=&a;
- Int *q=&a;
- Int *r=&a;
- }
- Printf("Enter a num");
- Scanf("%d",&a);
- Printf("%d",*p);
- Printf("%d",*q);
- Printf("%d",*r);
- Return 0;
- }

Exchange two numbers using pointers

- void exchange(int* int*);
- int main
- {
- int a=5;
- int b=7;
- exchange(&a,&b);
- printf(“%d%d”,a,b);
- return 0;
- }
- void exchange(int *px, int *py)
- {
- int temp;
- temp=*px;
- *px=*py;
- *py=temp;
- return;
- }

Functions returning pointers

- Int *smaller (int *p1,int *p2);
- Int main()
- {
- Int a;
- Int b;
- Int *p;
- Scanf(“%d%d”,&a,&b);
- P=smaller(&a,&b);
- }
- Int * smaller(int *px,int *py)
- {
- return (*px <*py?px:py)
- }

Pointer to pointer

- #include<stdio.h>
- main()
- {
- int a=100;
- int *p;
- int **p1;
- p=&a;

p1=&p;

- printf(“value of a is %d”,a);
- printf(“%d’,*p);
- printf(“%d”,**p1);
- }
- Output?

Pointers and arrays

- Continuous memory locations are allocated for all the elements of array by the compiler.

- The base address is the location of the first element of the array.

- Int a[5]={10,20,30,40,50}
- Element a[0] a[1] a[2] a[3] a[4]
- Value 10 20 30 40 50
- Address 1000 1002 1004 1006 1008

- If p declared as an integer pointer then array can be pointed by
- P=&a[0];

- Every value of array can be accessed by using p++
- The length of datatype is scale factor or size
- Address of an element is calculated using its index and the scale factor of data type
- Ex: addrs of a[3]=base address+[3*scale factor of int]
- 1000+(3*2)
- 1000+6=1006
- Instead of using array indexing pointer can be used to access array
- *(p+3) gives value of a[3]
- A[i]=*(p+i)

Program on pointer and array

```
• #include<stdio.h>
• {
• int a[5];
• int *p, i;
printf(”Enter 5 elemets”);
•  for(i=0;i<5;i++)
• scanf(“%d”,&a[i]);
• p=&a[0];
• printf(“Elements of array are”);
• for(i=0;i<5;i++)
• printf(“%d”,*(p+i));
• }
```