Pointers

Pointers

- As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.
- Consider the following example, which prints the address of the variables defined –

Pointers Program 1

```
#include <stdio.h>
int main () {
   int  var1;
   char var2[10];
   printf("Address  of var1 variable: %x\n", &var1  );
   printf("Address  of var2 variable: %x\n", &var2  );
   return 0;
}
```

Output

```
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
```

What are pointers essentially?

- 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.

Sample pointer declarations

- int   *ip;   /* pointer to an integer */
- double *dp;   /* pointer to a double */
- float *fp;   /* pointer to a float */
- char   *ch   /* pointer to a character */

Example:

```
#include <stdio.h>
int main()
{
    printf("Hello World");
    int x_main = 10;
    int *x_pointer;
    x_pointer = &x_main;
    printf("\n The value of x_main (directly): %d", x_main);
    printf("\n The address of x_main (directly) : %u", &x_main);
    printf("\n The address of x_main (indirect): %u", x_pointer);
    printf("\n The value of x_main (indirect): %d", *x_pointer);
    printf("\nThe address of pointer x_pointer: %u",&x_pointer);
    return 0;
}
```

How to Use Pointers?

- There are a few important operations, which we will do with the help of pointers very frequently.
- **(a)** We define a pointer variable,
- **(b)** assign the address of a variable to a pointer and
- **(c)** finally access the value at the address (using *) available in the pointer variable.

```
int main () {
  int  var = 20;   /* actual variable declaration */
  int  *ip;        /* pointer variable declaration */
  ip = &var;  /* store address of var in pointer variable*/
  printf("Address of var variable: %x\n", &var  );
  /* address stored in pointer variable */
  printf("Address stored in ip variable: %x\n", ip );
  /* access the value using the pointer */
  printf("Value of *ip variable: %d\n", *ip );
  return 0;}
```

Output

```
Address of var variable:
bffd8b3c
Address stored in ip variable:
bffd8b3c
Value of *ip variable: 20
```

NULL Pointers

- 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.

```
int main () {
  int  *ptr = NULL;
  printf("The  value of ptr is : %x\n", ptr  );
  return 0;
}
```

To check for a null pointer, you can use an 'if' statement as follows if(ptr)   /* succeeds if p is not null */
if(!ptr)   /* succeeds if p is null */

Output

```
The value of ptr is 0
```

Get value by pointer

```
int* pc, c;
c = 5;
*pc = &c;
printf("%d", *pc);   // Output: 5
```

- Here, the address of c is assigned to the pc pointer. To get the value stored in that address, we used *pc.
- By the way, * is called the dereference operator (when working with pointers). It operates on a pointer and gives the value stored in that pointer.

Changing Value Pointed by Pointers

```
int* pc, c;
c = 5;
pc = &c;
c = 1;
printf("%d", c);    // Output: 1
printf("%d", *pc);  // Ouptut: 1
```

Changing Value Pointed by Pointers – Another approach

```
int* pc, c;
c = 5;
pc = &c;
*pc = 1;
printf("%d", *pc);  // Ouptut: 1
printf("%d", c);    // Output: 1
```

Common mistakes when working with pointers

```
int c, *pc;

// pc is address but c is not
pc = c;  // Error

// &c is address but *pc is not
*pc = &c;  // Error

// both &c and pc are addresses
pc = &c;  // Not an error

// both c and *pc are values
*pc = c;  // Not an error
```

Relationship Between Arrays and Pointers

```
#include <stdio.h>
int main() {
  int x[4];
  int i;
  for(i = 0; i < 4; ++i) {
    printf("&x[%d] = %p\n", i, &x[i]);
  }
  printf("Address of array x: %p", x);
  return 0;
}
```

```
&x[0] = 1450734448
&x[1] = 1450734452
&x[2] = 1450734456
&x[3] = 1450734460
Address of array x: 1450734448
```

Relationship Between Arrays and Pointers

```
#include <stdio.h>
int main() {
        int x[3]={1,2,3};
        int *y;
        y=x;
        int i;
        for(i = 0; i < 3; ++i) {
                printf("&x[%d] using array x        = %p\n", i, &x[i]);
                printf("&x[%d] using pointer to x    = %p\n", i, (y+i));
                printf("Value in x[%d] using array x = %d\n", i, x[i]);
                printf("using pointer Value in x[%d] = %d\n", i, *(y+i));   }
        return 0;
}
```

```
&x[0] using array x         = 1450734448
&x[0] using pointer to x    = 1450734448
Value in x[0] using array x = 1
using pointer Value in x[0]= 1
&x[1] using array x         = 1450734452
&x[1] using pointer to x    = 1450734452
Value in x[1] using array x = 2
using pointer Value in x[1]= 2
&x[2] using array x         = 1450734456
&x[2] using pointer to x    = 1450734456
Value in x[2] using array x = 3
using pointer Value in x[2]= 3
```

Relationship Between Arrays and Pointers

- &x[0] and x+0 or x is the same. It's because the variable name x points to the first element of the array. And, x[0] is equivalent to *x   or *(x+0).

Similarly,

- &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
- &x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).
- ...
- Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

Pointer Arithmetic

- A pointer in c is an address, which is a numeric value.
- Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value.
- There are four arithmetic operators that can be used on pointers: ++, --, +, and -

Incrementing a Pointer

```
int main () {
  int  var[] = {10, 100, 200};
  int  i, *ptr;
  /* let us have array address in pointer */
  ptr = var;
  for ( i = 0; i < 3; i++) {
    printf("Address of var[%d] = %x\n", i, ptr );
    printf("Value of var[%d] = %d\n", i, *ptr );
    /* move to the next location */
    ptr++;
  }
  return 0;
}
```

Output

```
Address of var[0] = bf882b30
Value of var[0] = 10
Address of var[1] = bf882b34
Value of var[1] = 100
Address of var[2] = bf882b38
Value of var[2] = 200
```

Decrementing a Pointer

- You may use the same program to do it

Pointer Comparisons

- Pointers may be compared by using relational operators, such as ==, <, and >.
- If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared.

Ex..

```
int main () {
  double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
   double *p;
   int i;
   p = balance;
   printf( "Array values using pointer\n");
   for ( i = 0; i < 5; i++ ) {
      printf("*(p + %d) : %f\n",  i, *(p + i) );
   }
   printf( "Array values using balance as address\n");
   for ( i = 0; i < 5; i++ ) {
      printf("*(balance + %d) : %f\n",  i, *(balance + i) );
   }
   return 0;}
```

```
Array values using pointer
*(p + 0) : 1000.000000
*(p + 1) : 2.000000
*(p + 2) : 3.400000
*(p + 3) : 17.000000
*(p + 4) : 50.000000
Array values using balance as address
*(balance + 0) : 1000.000000
*(balance + 1) : 2.000000
*(balance + 2) : 3.400000
*(balance + 3) : 17.000000
*(balance + 4) : 50.000000
```

Array of pointers

- There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available. Following is the declaration of an array of pointers to an integer −

- int *ptr[MAX];

Array of Pointers

```
#include <stdio.h>
int main () {
  int  var[] = {10, 100, 200};
  int i, *ptr[3];
  for ( i = 0; i < 3; i++) {
    ptr[i] = &var[i]; /* assign the address of integer. */
  }
  for ( i = 0; i < 3; i++) {
    printf("Value of var[%d] = %d\n", i, *ptr[i] );
  }
  return 0;
}
```

Output

```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```