Pointers in C

A pointer is a variable whose value is the address of another variable. 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, data 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. Pointers can be declared to point to objects of any type.

To use pointers in C, we must understand below two operators.

- To access address of a variable to a pointer, we will use the unary **operator &** (ampersand) that returns the address of that variable. For example, &x gives us address of variable x.

```
#include <stdio.h>
int main()    {
int x;
    // Prints address of x
    printf("%p", &x);
    return 0; }
```

- To declare a pointer variable and to access the value stored in the address we use the unary **operator (*)** that returns the value of the variable located at the address specified by its operand

```
#include <stdio.h>
int main()
{
 int Var = 10;  // A normal integer variable
    int *ptr = &Var;
    printf("Value of Var = %d\n", *ptr);
    printf("Address of Var = %p\n", ptr);
   *ptr = 20; // Value at address is now 20
    printf("After doing *ptr = 20, *ptr is %d\n", *ptr);
    return 0;
 }
```

Important notes: The data type of pointer and the variable must match, an int pointer can hold the address of int variable, similarly a pointer declared with float data type can hold the address of a float variable.

❖ *Investigate the following two examples:*

Example1:

```
#include <stdio.h>
int main()
{
int a;
int *aPtr;
a = 7;
aPtr = &a;
printf( "\n The address of a is %p\n and the value of *aPtr is %p",
&a, aPtr);
printf( " \n The value of a is %d\n and the value of *aPtr is %d", a,
*aPtr);
printf( " \nShowing that * and & are inverses of each other. \n &*aPtr
= %p and *&aPtr = %p \n",&*aPtr, *&aPtr);
}
```

Example 2:

```
#include <stdio.h>
int main()
{
    int var =10;
    int *p;
    p= &var;

    printf ( "Address of var is: %p", &var);
    printf ( "\nAddress of var is: %p", p);

    printf ( "\nValue of var is: %d", var);
    printf ( "\nValue of var is: %d", *p);
    printf ( "\nValue of var is: %d", *( &var));

    printf( "\nValue of pointer p is: %p", p);
    printf ( "\nAddress of pointer p is: %p", &p);

    return 0;
}
```

Pointer Expressions and Pointer Arithmetic

A limited set of arithmetic operations can be performed on pointers. However, there are Two famous arithmetic operators that can be used on pointers:

- ❖ Incremented (++)
- ❖ Decremented (−−)

In C, pointers and arrays are related. Hence, Pointer arithmetic is meaningless unless performed on an array. An array name acts like a pointer constant. The value of this pointer constant is the address of the first element. For example, if we have an array named *val* then *val* and &*val*[0] can be used interchangeably.

❖ *Investigate the following two examples:*

Example1:

```
#include <stdio.h>
// Driver program
int main()
{
    // Declare an array
    int v[3] = {10, 100, 200};

    // Declare pointer variable
    int *ptr;

    // Assign the address of v[0] to ptr
    ptr = v;

    for (int i = 0; i < 3; i++)
    {
        printf("Value of *ptr = %d\n", *ptr);
        printf("Value of ptr = %p\n\n", ptr);

        // Increment pointer ptr by 1
        ptr++;
    }
}
```

Example 2:

```
#include <stdio.h>
const int MAX = 3;

int main () {

   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have array address in pointer */
   ptr = &var[MAX-1];

   for ( i = MAX; i > 0; i--) {

   printf("Address of var[%d] = %x\n", i-1, ptr );
   printf("Value of var[%d] = %d\n", i-1, *ptr );

   /* move to the previous location */
   ptr--;
   }

   return 0;
}
```

Array of pointers

Just like we can declare an array of int, float or char etc., we can also declare an array of pointers, here is the syntax to do the same.

```
datatype *array_name[size];
```

As an example let's consider the following:

```
int *arrop[5];
```

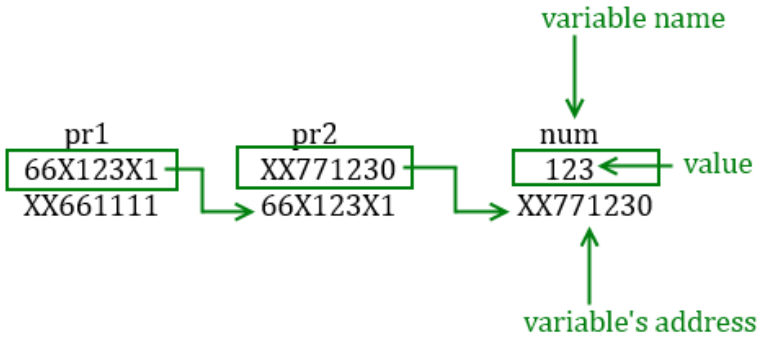
Here *arrop* is an array of 5 integer pointers. It means that this array can hold the address of 5 integer variables.

❖ *Investigate the following example:*

```
#include <stdio.h>

const int MAX = 3;

int main () {

   int  var[] = {10, 100, 200};
   int i, *ptr[MAX];

   for ( i = 0; i < MAX; i++) {
      ptr[i] = &var[i]; /* assign the address of integer. */
   }

   for ( i = 0; i < MAX; i++) {
      printf("Value of var[%d] = %d\n", i, *ptr[i] );
      printf("Value of ptr[%d] = %p\n", i, ptr[i] );
   }

   return 0;
}
```

Pointer to Pointer (Double Pointer)

We already know that a pointer holds the address of another variable of same type. When a pointer holds the address of another pointer then such type of pointer is known as pointer-to-pointer or double pointer.



In the shown diagram, $pr2$ is a normal pointer that holds the address of an integer variable num. There is another pointer $pr1$ in the diagram that holds the address of another pointer $pr2$, the pointer $pr1$ here is a pointer-to-pointer.

❖ *Investigate the following example:*

```
#include <stdio.h>

int main () {

int  var;
int  *ptr;
int  **pptr;

var = 3000;

/* take the address of var */
ptr = &var;

/* take the address of ptr */
pptr = &ptr;

/* take the value of the var using ptr and pptr */
printf(" Value of var is: %d\n", var );
printf(" Value of var using ptr is: %d\n", *ptr );
printf(" Value of var using pptr is: %d\n\n", **pptr);

/*Possible ways to find address of var*/
printf(" Address of var is: %p\n", &var);
printf(" Address of var using ptr is: %p\n", ptr);
printf(" Address of var using pptr is: %p\n\n", *pptr);

/*Possible ways to find address of ptr*/
printf(" Address of ptr is:%p\n",&ptr);
printf(" Address of ptr using pptr is:%p\n\n",pptr);

/*Possible way to find address of pptr*/
printf(" Address of pptr is:%p",&pptr);

return 0;
}
```

Passing pointers to functions in C

pointers can also be passed to a function as an argument like any other argument. When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value. So any change made by the function using the pointer is permanently made at the address of passed variable. This technique is known as call by reference in C.

The following example that shows how to swap numbers using call by reference.

```
#include <stdio.h>
void swapnum(int *num1, int *num2)
{
   int tempnum;

   tempnum = *num1;
   *num1 = *num2;
   *num2 = tempnum;
}
int main( )
{
   int v1 = 11, v2 = 77;
   printf("Before swapping:");
   printf("\nValue of v1 is: %d", v1);
   printf("\nValue of v2 is: %d", v2);

   /*calling swap function*/
   swapnum( &v1, &v2 );

   printf("\nAfter swapping:");
   printf("\nValue of v1 is: %d", v1);
   printf("\nValue of v2 is: %d", v2);

return 0;
}
```