Pointers in C

Rab Nawaz Jadoon

Assistant Professor

COMSATS IIT, Abbottabad

Pakistan

Department of Computer Science

DCS

COMSATS Institute of
Information Technology

Introduction to pointers

- Which feature of C do beginners find most difficult to understand?
- The answer is easy:

Pointers

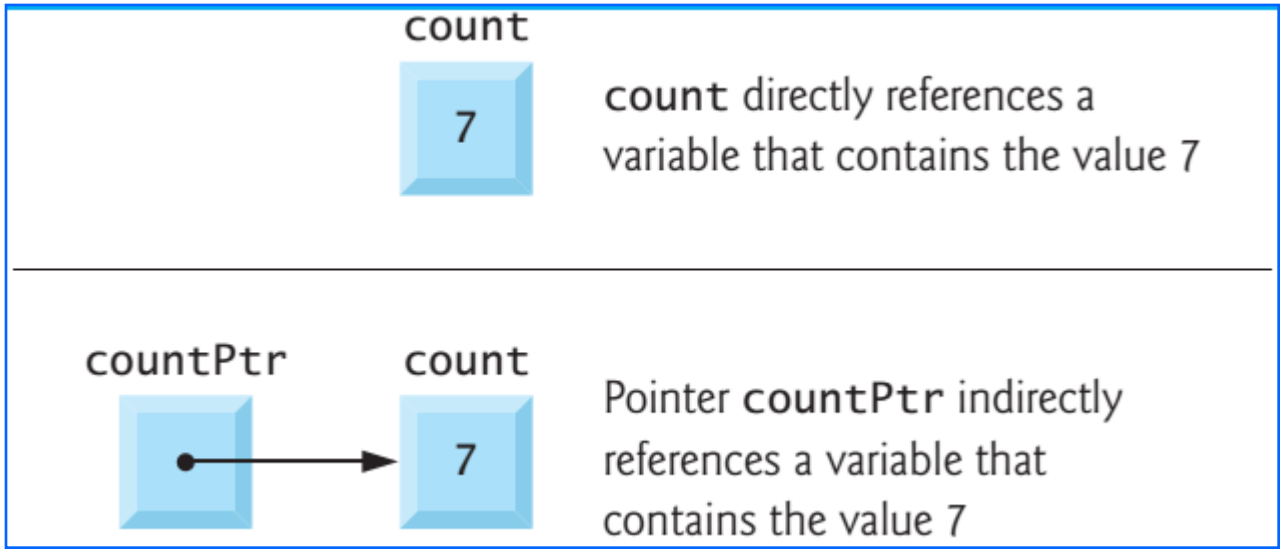
Pointers

- Pointers enable programs to simulate call-by-reference and to create and manipulate dynamic data structures, i.e.,
  - Data structures that can grow and shrink at execution time, such as linked lists, queues, stacks and trees.

Pointer

- Pointers are variables whose values are memory addresses.
- A pointer, on the other hand, contains an address of a variable that contains a specific value.
- In this sense, a variable name directly references a value, and a pointer indirectly references a value.
- Referencing a value through a pointer is called indirection.

Pointer



Pointers, like all variables, must be defined before they can be used.

Pointer definition

- Int *countPtr, count;
- specifies that variable countPtr is of type int * (i.e., a pointer to an integer) and is read,
  - "countPtr is a pointer to int" or
  - "countPtr points to an object of type int."
- Also, the variable count is defined to be an int, not a pointer to an int.
- The * only applies to countPtr in the definition.
  - If you wish to declare xPtr and yPtr as int pointers, use int *xPtr, *yPtr

Pointers

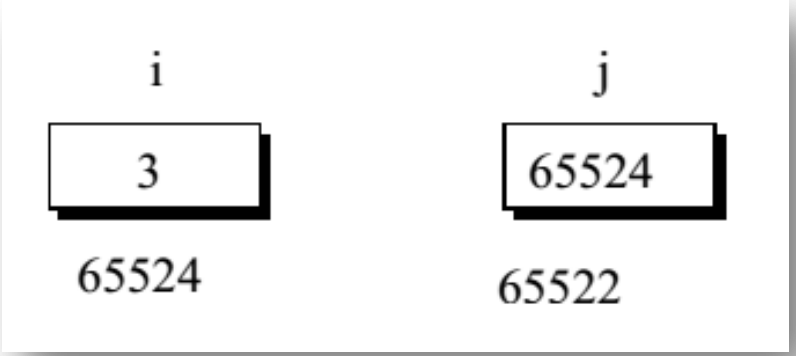
```
main( )
{
      int  i = 3 ;

      printf ( "\nAddress of i = %u", &i ) ;
      printf ( "\nValue of i = %d", i ) ;
      printf ( "\nValue of i = %d", *( &i ) ) ;
}
```

The output of the above program would be:

```
Address of i = 65524
Value of i = 3
Value of i = 3
```

Note that printing the value of *(&i)is same as printing the value of i.

- The following memory map would illustrate the contents of i and j.



As you can see, i's value is 3 and j's value is i's address.

We can't use j in a program without declaring it. And since j is a variable that contains the address of i, it is declared as, int *j ;

Pointers

- **int *j ;**
  - * stands for "value at address".
  - Thus, int *j would mean, the value at the address contained in j is an int.

Pointers

```
main( )
{
 int i = 3 ;
 int *j;
 j = &i;
 printf ( "\nAddress of i = %u", &i );
 printf ( "\nAddress of i = %u", j );
 printf ( "\nAddress of j = %u", &j );
 printf ( "\nValue of j = %u", j );
 printf ( "\nValue of i = %d", i );
 printf ( "\nValue of i = %d", *( &i ) );
 printf ( "\nValue of i = %d", *j );
}
```

Pointers

```
int  *alpha ;
char  *ch ;
float  *s ;
```

- The declaration float *s does not mean that s is going to contain a floating-point value.
  - What it means is, s is going to contain the address of a floating-point value.
  - Similarly, char *ch means that ch is going to contain the address of a char value. Or in other words,
    - the value at address stored in ch is going to be a char.

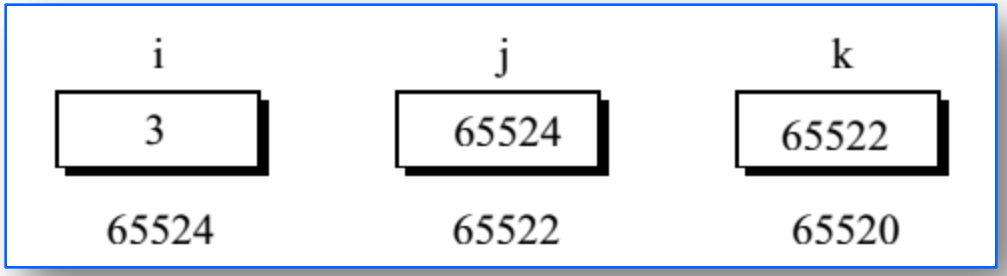
Conclusion

Pointers are variables that contain addresses,

and since addresses are always whole numbers, pointers would always contain whole numbers.

More on pointers

- Pointer, we know is a variable that contains address of another variable.
- Now this variable itself might be another pointer.
- Thus, we now have a pointer that contains another pointer's address.



Program

```
#include<stdio.h>
#include<conio.h>
main( )
{
 int i = 3, *j, **k ; //k is a pointer to an integer pointer
 j = &i ;
 k = &j ;
 printf ( "\nAddress of i = %u", &i ) ;
 printf ( "\nAddress of i = %u", j ) ;
 printf ( "\nAddress of i = %u", *k ) ;
 printf ( "\nAddress of j = %u", &j ) ;
 printf ( "\nAddress of j = %u", k ) ;
 printf ( "\nAddress of k = %u", &k ) ;
 printf ( "\nValue of j = %u", j ) ;
 printf ( "\nValue of k = %u", k ) ;
 printf ( "\nValue of i = %d", i ) ;
 printf ( "\nValue of i = %d", * ( &i ) ) ;
 printf ( "\nValue of i = %d", *j ) ;
 printf ( "\nValue of i = %d", **k ) ;
 getche();
}
```

```
Address of i = 1638224
Address of i = 1638224
Address of i = 1638224
Address of j = 1638220
Address of j = 1638220
Address of k = 1638216
Value of j = 1638224
Value of k = 1638220
Value of i = 3
Value of i = 3
Value of i = 3
Value of i = 3
```

Function calls by value and by reference

Call by Value

```
main( )
{
      int  a = 10, b = 20 ;

      swapv ( a, b ) ;
      printf ( "\na = %d b = %d", a, b ) ;
}

swapv ( int  x, int  y )
{
      int  t ;

      t = x ;
      x = y ;
      y = t ;

      printf ( "\nx = %d y = %d", x, y ) ;
}
```

Call by Reference

```
main( )
{
      int  a = 10, b = 20 ;

      swapr ( &a, &b ) ;
      printf ( "\na = %d b = %d", a, b ) ;
}

swapr( int  *x, int  *y )
{
      int  t ;

      t = *x ;
      *x = *y ;
      *y = t ;
}
```


THANKS