Introduction to Pointers in C

Address in C

Whenever a variable is declared in C language, a memory location is assigned for it, in which it's value will be stored. The size of the allocated memory space depends on the datatype of the variable. Also this size varies depending on the compiler that you are using. You may consider the name of the declared variable as the name of that allocated memory space.

We can easily check this memory address, using the *& (address of)* symbol.

If ***var*** is the name of a variable, then ***&var*** will give it's address.

You can print that address as follows:

*printf(“%u”,&var); // u stands for unsigned (addresses are always unsigned)*

Say, in your program the value of ***var*** (considering it as an integer) is 5 (assigned by you in your program). Then to print the value of ***var*** we use:

*printf(“%d”,var); // Output is 5*

Now, alternatively this can be done as follows:

*printf(“%d”,\*(&var)); // Here also you get the output 5*

Here, \* means value at some address. So, ***\*(&var)*** means value at address (&var) which is 5.

Pointers in C

A Pointer in C language is a variable which holds the address of another variable of the same data type. Pointers are used to access memory and manipulate the address.

So, we may require a variable that will hold the address of another variable (or simply an address). Let us consider the following example:

int var, \*adr;

Here, I have declared ***adr*** to store the address of ***var*** and also have used a star (\*) before ***adr*** to signify that it is a pointer (that can point to some memory location). Don’t confuse this \* with the ‘value at address’ operator that we use to print the value stored at some memory address (as discussed in the ***Address in C*** section).

Now, we have a pointer ***adr*** that can point to another memory location. The following assignment will store the address of ***var*** in ***adr***:

*adr=&var;*

Now ,

*printf(“%u”,adr);*

will print the address of ***var***.

We can also print the value of ***var*** using the pointer ***adr*** as follows:

*printf(“%d”,\*adr); // Here \* is the ‘value at address’ operator*

Now, if we want to increase the value of ***var*** (say, 5) by 2 we can do that as follows:

var=var+2;

The same can be done with the help of the pointer ***adr*** as follows:

\*adr=\*adr+2; // Here \*adr will work as an variable like var=var+2

**Pointer to pointer** ***(A Pointer variable that holds the address of another pointer variable)*** **:-** Now let us move one step ahead. As discussed above ***adr*** is a pointer variable. So this pointer variable also occupies some memory space and that memory space also has some address. We can get that address by using ***&*** before ***adr*** as follows:

printf(“%u”,&adr);

This is same as -----> printf(“%u”,&(&var));

The question is -----> How to declare a pointer to pointer?

If we require one star (\*) before the name of a pointer variable to declare it, then why should we not use two stars (\*\*) before the name of  a pointer to pointer variable?

Yes, we do so. Consider the following example:

int \*\*adrtoadr; // Here adrtoadr is a pointer to pointer

adrtoadr=&adr; // Assignment of the address of an pointer variable to pointer to pointer

printf(“%u”,adrtoadr); // This displays the address of adr

printf(“%u”,&adr); // Same as -----> printf(“%u”,adrtoadr);

printf(“%u”,\*adrtoadr); // Same as -------> printf(“%u”,\*(&adr));

The last printf statement prints the value at adrtoadr i.e., value at ***&adr***. As we have assigned ***adr=&var*** in the earlier section, the value contained at the memory address adr is ***&var***. So the last printf statement will display the address of ***var***.

printf(“%d”,\*\*adrtoadr);

The above statement is same as

-------> printf(“%d”,\*\*(&adr)); -------> printf(“%d”,\*(adr)); -----> printf(“%d”,\*(&var)); -----> printf(“%d”,var); // (note that ***\*&*** (& followed by \*) can be cancelled out)

So, we can print the value stored at ***var*** using a pointer to pointer as discussed above.

Please follow FIg. 1 and Fig. 2 in the image file (.jpg).