Pointers in C

Extends Pointers in C and C++

Pointers in C and C++ are very complex issues; however, it is very easy to understand if you are in a right direction.

First, I am going to describe what pointers do and then give some examples. Second, I will create some confusion to make you understand what problem you will face in future but at the end, I will clear up all.

In simple term, pointer is a variable, which holds a memory address of another variable. For **16 bit** compilers, pointers themselves are **2 bytes** (one is for their existence and the other byte is for the indication of memory location), however for 32 bit compilers it would be **4 bytes**. There are two types of pointers **huge (4 Bytes)** and **far (4 Bytes).** Luckily, you do not have to worry about huge and far.

When we first declare a pointer, it is a null pointer unless it points to something. Let us see some example in the below code.

**(read the comments to understand more)**

|  |
| --- |
| /\* | Author : Alim Ul Karim |  | Email : [auk.junk@live.com](mailto:auk.junk@live.com) |  | Portfolio : <auk-port.webs.com> |  | Blog : <bit.ly/auk-blog> |  |------------------------------------------|  | Code tested on CodeBlocks , GCC Compiler |  | Example of Null Pointer |  |------------------------------------------|  \*/  **#include <stdio.h>**  /\* To understand more about pointers : <http://en.wikipedia.org/wiki/Pointer_(computing)> \*/  **int** main(){  /\* Below int \*x is a null pointer because it doesn’t point to any variable in the memory \*/  **int** \*x , a; /\* Null Pointer is x, and a is a simple integer variable \*/  /\* Now locate a memory address for this pointer  For C++ and must #include <iostream> and below commented code works fine.  \*/  //x = new int; //code for C++  **printf**("Example with pointers:\n");  /\*For C and C++ we can use malloc function, which allocates a memory address to a variable \*/  //x = (int \*)malloc(1); // initalize/allocate a memory location to a integer variable  x = (**int** \*)**malloc**(**sizeof**(**int**)); // most appropriate  /\*For scanf: we always use the ‘&’ + ‘the variable name’ ,  to get the memory location of that variable and scanf replace the  user-inputted value to that memory location.  \*/  /\*For pointers, \*x means the value of the pointer and  only x means the first located elements memory address \*/  **scanf** ("%d" , x); //memory address  **printf**("%d\n" , \*x); //getting value from that memory address , without assigning any memory location to **x** , **executing** this line will **crash** your **program**.  **printf**("\nExample with regular variable:\n");  **scanf** ("%d" , &a); // memory address  **printf**("%d\n" , a); //getting value from that memory address , understand the similarities  /\* So , we can hence that ,  **x is equivalent &a**  **\*x is equivalent a**  Equivalent: [**http://en.wikipedia.org/wiki/Logical\_equivalence**](http://en.wikipedia.org/wiki/Logical_equivalence)  \*/  } |

Now that you have understood what a NULL pointer is, we can move on to next few examples.

|  |
| --- |
| /\* | Author : Alim Ul Karim |  | Email : [auk.junk@live.com](mailto:auk.junk@live.com) |  | Portfolio : <auk-port.webs.com> |  | Blog : <bit.ly/auk-blog> |  |------------------------------------------|  | Code tested on CodeBlocks , GCC Compiler |  | Pointer Example Part 2 |  |------------------------------------------|  \*/  **#include <stdio.h>**  /\* To understand more about pointers : <http://en.wikipedia.org/wiki/Pointer_(computing)> \*/  **int** main(){  **int** \*x , a = 120;  **printf**("a = %d:\n" , a);  /\* as previous example , now we don't need to allocate memory address \*/  //x = (int \*)malloc(sizeof(int));  /\* we already have a variable and which must exist in a memory location  so rather than allocating new memory location, we can simply point to that variable  via our pointer.  \*/  x = &a ; // Now we are pointing 'a' memory location via pointer of \*x.  **printf**("\n Memory Location Assigned to \*x:\n\n" , a);  \*x = 500;  /\* we know from previous example that,  **x is equivalent &a**  **\*x is equivalent a**  Equivalent: http://en.wikipedia.org/wiki/Logical\_equivalence  \*/  //So what would be your guess  **printf**("a = %d: after preserving \*x = 500\n" , a); // it prints 500, rather than 120  a = 920;  **printf**("\*x = %d: after preserving a = 920\n" , \*x); // it prints 920, rather than 500  **printf**("a = %d: after preserving a = 920\n" , a); // it prints 920, rather than 120 or 500  **printf**("\n-------------Memory Locations ----------\n\n" , a);  **printf**("x = %d: \n" , x ); // memory location of x  **printf**("&a = %d: \n" , &a); // same as x's memory location  } |

Another great way of learning is thinking things simply ; the entire processing picture of the pointers may seem cumbersome, however thinking things in a simple form can make you learn the entire result easily than it is. Now we can see some interesting parts of pointers in below.

|  |
| --- |
| /\* | Author : Alim Ul Karim |  | Email : <mailto:auk.junk@live.com> |  | Portfolio : <auk-port.webs.com> |  | Blog : <bit.ly/auk-blog> |  |------------------------------------------|  | Code tested on CodeBlocks , GCC Compiler |  | Pointer Example Part 2 |  |------------------------------------------|  \*/  **#include <stdio.h>**  **int** main(){  **int** \*x , a = 120;  **printf**("a = %d:\n" , a);  /\* as previous example , now we don't need to allocate memory address \*/  //x = (int \*)malloc(sizeof(int));  /\* we already have a variable and which must exist in a memory location  so rather than allocating new memory location, we can simply point to that variable  via our pointer.  \*/  x = &a; // Now we are pointing 'a' memory location via pointer of \*x.  \*x = 500;  **printf** ("\n\*x , a: %d\n\n" , a);  /\* we know from previous example that,  x is equivalent &a  \*x is equivalent a  Equivalent: http://en.wikipedia.org/wiki/Logical\_equivalence  \*/  **printf** ("value of x = %d\n" , x );  **printf** ("value of &x = %d\n" , &x );  **printf** ("value of \*&x = %d\n" , \*&x ); // it seems that **‘\*&’ always canceled out**  **printf** ("value of \*\*&x = %d\n" , \*\*&x ); // so if **'\*&' canceled out from \*\*&x only remain \*x**  **printf** ("value of &\*&x = %d\n" , &\*&x ); // so if **'\*&' canceled out from &\*&x only remain &x**  } |

Now we can visit the **confusion** part, which will conclude the promise that I had done earlier.

|  |
| --- |
| /\* | Author : Alim Ul Karim |  | Email : <mailto:auk.junk@live.com> |  | Portfolio : <auk-port.webs.com> |  | Blog : <bit.ly/auk-blog> |  |------------------------------------------|  | Code tested on CodeBlocks , GCC Compiler |  | Pointer Example Part 2 |  |------------------------------------------|  \*/  **Confusion part** |