IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

THE DEMONSTRATION POINTERS TO POST GRADUATE STUDENTS

Thakur Janhavee Dipak
Msc IT Department CS &IT
GMVSC, Tala

Jadhav Mrunali Nandkumar
Msc IT Department CS &IT
GMVSC, Tala

Jadhav Ankita Mahadev
Msc IT Department CS &IT
GMVSC, Tala

Mohite Sonali Ravindra
Msc IT Department CS &IT
GMVSC, Tala

Prof. Raghvendra Singh
Assistant Professor GMVSC & GMVIT
University of Mumbai

ABSTRACT

A Pointer in C language is a variable which holds the address of another variable of same data type. Pointers are used to pierce memory and manipulate the address. Pointers are one of the most distinct and instigative features of C language. It provides power and inflexibility to the language. Although pointers may appear a little confusing and complicated in the morning, but trust me, once you understand the conception, you'll be suitable to do so much more with C language. Before we start understanding what pointers are and what they can do, let's start by understanding what does" Address of a memory position"means? Its done with compilers

Keywords pointers, c language, compilers.

1. PREFACE

The Pointer in C, is a variable that stores address of another variable. A pointer can also be used to relate to another pointer function. A pointer can be incremented/ decremented, i.e., to point to the coming/ former memory position. The purpose of pointer is to save memory space and achieve briskly prosecution time.

2. LITERATURE REVIEW

In computer wisdom, a pointer is an object in numerous programming languages that stores a memory address. This can be that of another value located in computer memory, or in some cases, that of memory- counterplotted computer tackle. A pointer references a position in memory, and carrying the value stored at that position is known as dereferencing the pointer. As an analogy, a runner number in a book's indicator could be considered a pointer to the corresponding runner; dereferencing such a pointer would be done by flipping to the runner with the given runner number and reading the textbook plant on that runner. The factual format and content of a pointer variable is dependent on the underpinning computer armature.

Using pointers significantly improves performance for repetitious operations, like covering iterable data structures (e.g. strings, lookup tables, control tables and tree structures). In particular, it's frequently much cheaper in time and space to copy and dereference pointers than it's to copy and pierce the data to which the pointers point.

Pointers are also used to hold the addresses of entry points for called subroutines in procedural programming and for run- time linking to dynamic link libraries (DLLs). In object- acquainted programming, pointers to functions are used for binding styles, frequently using virtual system tables.

A pointer is a simple, more concrete perpetration of the further abstract reference data type. Several languages, especially low- position languages, support some type of pointer, although some have further restrictions on their use than others. While" pointer"has been used to relate to references in general, it more duly applies to data structures whose interface explicitly allows the pointer to be manipulated (arithmetically via pointer computation) as a memory address, as opposed to a magic cookie or capability which doesn't allow similar. (citation demanded) Because pointers allow both defended and vulnerable access to memory addresses, there are pitfalls associated with using them, particularly in the ultimate case. Primitive pointers are frequently stored in a format analogous to an integer; still, trying In 1955, Soviet computer scientist Kateryna Yushchenko constructed the Address programming language that made possible indirect addressing and addresses of the topmost rank – analogous to pointers. This language was considerably used on the Soviet Union computers. Still, it was unknown outside the Soviet Union and generally Harold Lawson is credited with the invention, in 1964, of the pointer. (2) In 2000, Lawson was presented the Computer Pioneer Award by the IEEE" (f) or contriving the pointer variable and introducing this generality into PL/ I, thus furnishing for the first time, the capability to flexibly treat linked lists in a general- purpose high- position language". (3) His seminal paper on the generalities appeared in the June 1967 issue of CACM entitled PL/ I List Processing. According to the Oxford English Dictionary, the word pointer first appeared in print as a mound pointer in a technical memorandum by the System Development Corporation.

Formal structure

In computer wisdom, a pointer is a kind of reference.

A data primitive (or just primitive) is any detail that can be read from or written to computer memory using one memory access (for case, both a byte and a word are barbarians).

A data aggregate (or just aggregate) is a group of barbarians that are logically conterminous in memory and that are viewed collectively as one detail (for case, an aggregate could be 3 logically conterminous bytes, the values of which represent the 3 coordinates of a point in space). When an aggregate is entirely composed of the same type of primitive, the aggregate may be called an array; in a sense, amulti-byte word primitive is an array of bytes, and some programs use words in this way.

In the terrain of these delineations, a byte is the smallest primitive; each memory address specifies a different byte. The memory address of the original byte of a detail is considered the memory address (or base memory address) of the entire detail.

A memory pointer (or just pointer) is a primitive, the value of which is intended to be used as a memory address; it's said that a pointer points to a memory address. It's also said that a pointer points to a detail (in memory) when the pointer's value is the detail's memory address.

More generally, a pointer is a kind of reference, and it's said that a pointer references a detail stored nearly in memory; to gain that detail is to dereference the pointer. The point that separates pointers from other kinds of reference is that a pointer's value is meant to be interpreted as a memory address, which is a rather low- position generality.

References serve as a position of indirection A pointer's value determines which memory address (that is, which datum) is to be used in a calculation. Because indirection is a fundamental aspect of algorithms, pointers are constantly expressed as a fundamental data type in programming languages; in statically (or strongly) compartmented programming languages, the type of a pointer determines the type of the detail to which the pointer points.

Architectural points

Pointers are a truly thin abstraction on top of the addressing capabilities handed by utmost modern architectures. In the simplest scheme, an address, or a numeric index, is assigned to each unit of memory in the system, where the unit is generally also a byte or a word – depending on whether the architecture is byte-addressable or word-addressable – effectively converting all of memory into a truly large array. The system would also also give an operation to recoup the value stored in the memory unit at a given address (generally exercising the machine's general purpose registers).

In the usual case, a pointer is large enough to hold farther addresses than there are units of memory in the system. This introduces the possibility that a program may essay to pierce an address which corresponds to no unit of memory, either because not enough memory is installed (i.e. beyond the range of available memory) or the architecture does not support analogous addresses. The first case may, in certain platforms analogous as the Intel x86 architecture, be called a segmentation fault (segfault). The alternate case is possible in the current performance of AMD64, where pointers are 64 bit long and addresses only extend to 48 bits. Pointers must conform to certain rules (canonical addresses), so if anon-canonical pointer is dereferenced, the processor raises a general protection fault.

On the other hand, some systems have farther units of memory than there are addresses. In this case, a more complex scheme analogous as memory segmentation or paging is employed to use different corridor of the memory at different times. The last incarnations of the x86 architecture support up to 36 bits of physical memory addresses, which were colluded to the 32- bit direct address space through the PAE paging medium. Thus, only1/ 16 of the possible total memory may be entered at a time. Another illustration in the same computer family was the 16- bit protected mode of the 80286 processor, which, though supporting only 16 MB of physical memory, could pierce up to 1 GB of virtual memory, but the combination of 16- bit address and member registers made piercing further than 64 KB in one data structure clumsy.

In order to give a harmonious interface, some architectures give memory- colluded I/ O, which allows some addresses to relate to units of memory while others relate to device registers of other bias in the computer. There are analogous generalities analogous as train counterpoises, array pointers, and remote object references that serve some of the same purposes as addresses for other types of objects.

3. PERPETRATION OF C

Address in C

Whenever a variable is defined in C language, a memory position is assigned for it, in which it's value will be stored. We can fluently check this memory address, using the & symbol.

Still, also & var will give it's address, If var is the name of the variable.

Let's write a small program to see memory address of any variable that we define in our program.

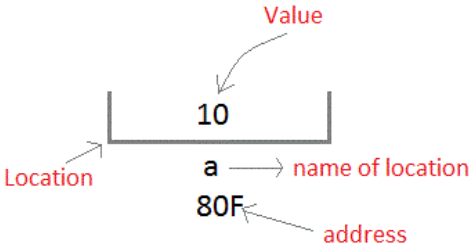
```
#include<stdio.h>

void main()
{
   int var = 7;
   printf("Value of the variable var is: %d\n", var);
   printf("Memory address of the variable var is: %x\n", &var);
}
Copy
```

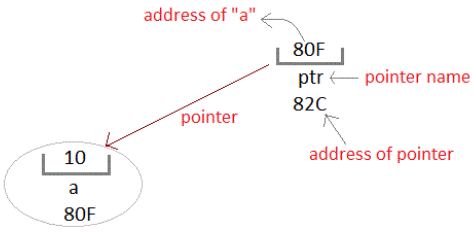
Value of the variable var is: 7
Memory address of the variable var is: bcc7a00
You must have also seen in the function scanf(), we mention &var to take user input for any variable var.
scanf("%d", &var);
Copy
This is used to store the user inputted value to the address of the variable var.

Concept of Pointers

Whenever a variable is declared in a program, system allocates a positioned an address to that variable in the memory, to hold the assigned value. This position has its own address number, which we just saw over. Let us assume that system has allocated memory location 80F for a variable a.
int a = 10;



We can access the value 10 either by using the variable name a or by using its address 80F.
The question is how we can access a variable using it's address? Since the memory addresses are also just numbers, they can also be assigned to some other variable. The variables which are used to hold memory addresses are called **Pointer variables**.
A **pointer** variable is therefore nothing but a variable which holds an address of some other variable. And the value of a **pointer variable** gets stored in another memory location.



Pointers in C are eas ; tasks are performed more easily with pointers, and other tasks, such ot be performed without using pointers. So it becomes necessary t rogrammer. Let's start learning them in simple and easy steps.

As you know, ev very 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 −

```
#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;
}
```

When the above code is compiled and executed, it produces the following result −
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
What are Pointers?
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. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid 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 */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

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 available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. The following example makes use of these operations −

Live Demo

```
#include <stdio.h>

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;
}
```

When the above code is compiled and executed, it produces the following result −

```
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.

The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program −

Live Demo

```
#include <stdio.h>
int main () {
   int  *ptr = NULL;
   printf("The value of ptr is : %x\n", ptr  );
    return 0;
}
```

When the above code is compiled and executed, it produces the following result −

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

In most of the operating systems, programs are not permitted to access memory at address 0 because that memory is reserved by the operating system. However, the memory address 0 has special significance; it signals that the pointer is not intended to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.

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 */
```

Pointers in Detail

Pointers have many but easy concepts and they are very important to C programming. The following important pointer concepts should be clear to any C programmer −

Concept & Description
1 Pointer arithmetic There are four arithmetic operators that can be used in pointers: ++, --, +, -
2 Array of pointers You can define arrays to hold a number of pointers.
3 Pointer to pointer C allows you to have pointer on a pointer and so on.
4 Passing pointers to functions in C Passing an argument by reference or by address enable the passed argument to be changed in the calling function by the called function.
5 Return pointer from functions in C C allows a function to return a pointer to the local variable, static variable, and dynamically allocated memory as well.

Benefits of using pointers

Below we have listed a few benefits of using pointers:

1. Pointers are more efficient in handling Arrays and Structures.
2. Pointers allow references to function and thereby helps in passing of function as arguments to other functions.
3. It reduces length of the program and its execution time as well.
4. It allows C language to support Dynamic Memory management.

In the next tutorial we will learn syntax of pointers, how to declare and define a pointer, and using a pointer.

4.REVIEW CONCLUSION

Pointers are more efficient in handling Arrays and Structures. Pointers allow references to function and thereby helps in passing of function as arguments to other functions. It reduces length of the program and its execution time as well. It allows C language to support Dynamic Memory management.

REFERENCES

1. Donald Knuth (1974). "Structured Programming with go to Statements" (PDF). Computing Surveys. 6 (5): 261–301. CiteSeerX 10.1.1.103.6084. doi:10.1145/356635.356640. S2CID 207630080. Archived from the original (PDF) on August 24, 2009.
2. Reilly, Edwin D. (2003). Milestones in Computer Science and Information Technology. Greenwood Publishing Group. p. 204. ISBN 9781573565219. Retrieved 2018-04-13. Harold Lawson pointer.
3. "IEEE Computer Society awards list". Awards.computer.org. Archived from the original on 2011-03-22. Retrieved 2018-04-13.
4. Jump up to:a b Plauger, P J; Brodie, Jim (1992). ANSI and ISO Standard C Programmer