A TUTORIAL ON POINTERS AND ARRAYS IN C

by Ted Jensen

Version 1.2 (PDF Version)
Sept. 2003

Available in various formats via
http://pweb.netcom.com/~tjensen/ptr/cpoint.htm

TABLE OF CONTENTS

PREFACE **2**

INTRODUCTION **4**

CHAPTER 1: What is a pointer? **5**

CHAPTER 2: Pointer types and Arrays **9**

CHAPTER 3: Pointers and Strings **14**

CHAPTER 4: More on Strings **19**

CHAPTER 5: Pointers and Structures **22**

CHAPTER 6: Some more on Strings, and Arrays of Strings **26**

CHAPTER 7: More on Multi-Dimensional Arrays **30**

CHAPTER 8: Pointers to Arrays **32**

CHAPTER 9: Pointers and Dynamic Allocation of Memory **34**

CHAPTER 10: Pointers to Functions **42**

EPILOG **53**

PREFACE

This document is intended to introduce pointers to beginning programmers in the C programming language. Over several years of reading and contributing to various conferences on C including those on the FidoNet and UseNet, I have noted a large number of newcomers to C appear to have a difficult time in grasping the fundamentals of pointers. I therefore undertook the task of trying to explain them in plain language with lots of examples.

The first version of this document was placed in the public domain, as is this one. It was picked up by Bob Stout who included it as a file called PTR-HELP.TXT in his widely distributed collection of SNIPPETS. Since that original 1995 release, I have added a significant amount of material and made some minor corrections in the original work.

I subsequently posted an HTML version around 1998 on my website at:

http://pweb.netcom.com/~tjensen/ptr/cpoint.htm

After numerous requests, I've finally come out with this PDF version which is identical to that HTML version cited above, and which can be obtained from that same web site.

Acknowledgements:

There are so many people who have unknowingly contributed to this work because of the questions they have posed in the FidoNet C Echo, or the UseNet Newsgroup comp.lang.c, or several other conferences in other networks, that it would be impossible to list them all. Special thanks go to Bob Stout who was kind enough to include the first version of this material in his SNIPPETS file.

About the Author:

Ted Jensen is a retired Electronics Engineer who worked as a hardware designer or manager of hardware designers in the field of magnetic recording. Programming has been a hobby of his off and on since 1968 when he learned how to keypunch cards for submission to be run on a mainframe. (The mainframe had 64K of magnetic core memory!).

Use of this Material:

note at one of the addresses below informing me of this. I have written this with the hope that it will be useful to others and since I'm not asking any financial remuneration, the only way I know that I have at least partially reached that goal is via feedback from those who find this material useful.

By the way, you needn't be an instructor or teacher to contact me. I would appreciate a note from anyone who finds the material useful, or who has constructive criticism to offer. I'm also willing to answer questions submitted by email at the addresses shown below.

Ted Jensen
Redwood City, California
tjensen@ix.netcom.com
July 1998

INTRODUCTION

If you want to be proficient in the writing of code in the C programming language, you must have a thorough working knowledge of how to use pointers. Unfortunately, C pointers appear to represent a stumbling block to newcomers, particularly those coming from other computer languages such as Fortran, Pascal or Basic.

To aid those newcomers in the understanding of pointers I have written the following material. To get the maximum benefit from this material, I feel it is important that the user be able to run the code in the various listings contained in the article. I have attempted, therefore, to keep all code ANSI compliant so that it will work with any ANSI compliant compiler. I have also tried to carefully block the code within the text. That way, with the help of an ASCII text editor, you can copy a given block of code to a new file and compile it on your system. I recommend that readers do this as it will help in understanding the material.

CHAPTER 1: What is a pointer?

One of those things beginners in C find difficult is the concept of pointers. The purpose of this tutorial is to provide an introduction to pointers and their use to these beginners.

I have found that often the main reason beginners have a problem with pointers is that they have a weak or minimal feeling for variables, (as they are used in C). Thus we start with a discussion of C variables in general.

A variable in a program is something with a name, the value of which can vary. The way the compiler and linker handles this is that it assigns a specific block of memory within the computer to hold the value of that variable. The size of that block depends on the range over which the variable is allowed to vary. For example, on PC's the size of an integer variable is 2 bytes, and that of a long integer is 4 bytes. In C the size of a variable type such as an integer need not be the same on all types of machines.

When we declare a variable we inform the compiler of two things, the name of the variable and the type of the variable. For example, we declare a variable of type integer with the name **k** by writing:

```
int k;
```

On seeing the "int" part of this statement the compiler sets aside 2 bytes of memory (on a PC) to hold the value of the integer. It also sets up a symbol table. In that table it adds the symbol **k** and the relative address in memory where those 2 bytes were set aside.

Thus, later if we write:

```
k = 2;
```

we expect that, at run time when this statement is executed, the value 2 will be placed in that memory location reserved for the storage of the value of **k**. In C we refer to a variable such as the integer **k** as an "object".

In a sense there are two "values" associated with the object **k**. One is the value of the integer stored there (2 in the above example) and the other the "value" of the memory location, i.e., the address of **k**. Some texts refer to these two values with the nomenclature ***rvalue*** (right value, pronounced "are value") and ***lvalue*** (left value, pronounced "el value") respectively.

In some languages, the lvalue is the value permitted on the left side of the assignment operator '=' (i.e. the address where the result of evaluation of the right side ends up). The rvalue is that which is on the right side of the assignment statement, the **2** above. Rvalues cannot be used on the left side of the assignment statement. Thus: **2 = k**; is illegal.