# C Language Tutorial

Version 0.042
March, 1999

Original MS-DOS tutorial by
Gordon Dodrill, Coronado Enterprises.

Moved to Applix by Tim Ward
Typed by Karen Ward

C programs converted by
Tim Ward and Mark Harvey
with assistance from Kathy Morton
for Visual Calculator
Pretty printed by Eric Lindsay

Applix 1616 microcomputer project
Applix Pty Ltd

# Introduction

The **C** programming language was originally developed by Dennis Ritchie of Bell Laboratories, and was designed to run on a PDP-11 with a UNIX operating system. Although it was originally intended to run under UNIX, there was a great interest in running it on the IBM PC and compatibles, and other systems. C is excellent for actually writing system level programs, and the entire Applix 1616/OS operating system is written in C (except for a few assembler routines). It is an excellent language for this environment because of the simplicity of expression, the compactness of the code, and the wide range of applicability.

It is not a good "beginning" language because it is somewhat cryptic in nature. It allows the programmer a wide range of operations from high level down to a very low level approaching the level of assembly language. There seems to be no limit to the flexibility available. One experienced C programmer made the statement, "You can program anything in C", and the statement is well supported by my own experience with the language. Along with the resulting freedom however, you take on a great deal of responsibility. It is very easy to write a program that destroys itself due to the silly little errors that, say, a Pascal compiler will flag and call a fatal error. In C, you are very much on your own, as you will soon find.

Since C is not a beginners language, I will assume you are not a beginning programmer, and I will not attempt to bore you by defining a constant and a variable. You will be expected to know these basic concepts. You will, however, not be expected to know anything of the C programming language. I will begin with the highest level of C programming, including the usually intimidating concepts of pointers, structures, and dynamic allocation. To fully understand these concepts, it will take a good bit of time and work on your part, because they not particularly easy to grasp, but they are very powerful tools. Enough said about that, you will see their power when we get there, just don't allow yourself to worry about them yet.

Programming in C is a tremendous asset in those areas where you may want to use Assembly Language, but would rather keep it a simple to write and easy to maintain program. It has been said that a program written in C will pay a premium of a 50 to 100% increase in runtime, because no language is as compact or fast as Assembly Language. However, the time saved in coding can be tremendous, making it the most desirable language for many programming chores. In addition, since most programs spend 90 percent of their operating time in only 10 percent or less of the code, it is possible to write a program in C, then rewrite a small portion of the code in Assembly Language and approach the execution speed of the same program if it were written entirely in Assembly Language.

Approximately 75 percent of all new commercial programs introduced for the IBM PC have been written in C, and the percentage is probably growing. Apple Macintosh system software was formerly written in Pascal, but is now almost always written in C. The entire Applix 1616 operating system is written in C, with some assembler routines.

Since C was designed essentially by one person, and not by a committee, it is a very usable language but not too closely defined. There was no official standard for the C language, but the American National Standards Association (ANSI) has developed a standard for the language, so it will follow rigid rules. It is interesting to note, however, that even though it did not have a standard, the differences between implementations are usually small. This is probably due to the fact that the original unofficial definition was so well thought out and carefully planned that extensions to the language are not needed.

Even though the C language enjoys a good record when programs are transported from one implementation to another, there are differences in compilers, as you will find any time you try to use another compiler. Most of the differences become apparent when you use nonstandard extensions such as calls to the MS-DOS BIOS, or the Applix 1616/OS system calls, but even these differences can be minimized by careful choice of programming means.

Applix 1616 builders have only the HiTech C compiler available. This version of the tutorial is customised to suit HiTech C. The original MS-DOS version by Gordon Dodrill was ported to the Applix 1616 (with great effort) by Tim Ward, and typed up by Karen Ward. The programs have been converted to HiTech C by Tim Ward and Mark Harvey, while Kathy Morton assisted greatly in getting Visual Calculator working. All have been tested on the Applix 1616/OS multitasking operating system. The Applix distribution disks contain the complete original text of this tutorial, plus all the converted C source code. The second disk contains executable, relocatable versions of all the programs, ready to run on an Applix 1616. There is also a directory of the original IBM source code, for those using IBM computers, who may wish to try them with a different compiler. This printed version has been edited, indexed and pretty printed by Eric Lindsay, who added the Applix specific material.

This printed version of the tutorial includes copies of all the code, for easier reference. It also includes a comprehensive table of contents, and index.

# 1
# Getting Started

This tutorial can be read simply as a text, however it is intended to be interactive. That is, you should be compiling, modifying and using the programs that are presented herein.

All the programs have been tested using the HiTech C compiler, and we assume that you have a copy of this. In addition, you should have a copy of various updates and header files for the C compiler, which appear on Applix User disks.

You can use either the builtin Applix 1616/OS editor `edit`, or the $30 *Dr Doc* editor in non-document mode. *Dr Doc* is somewhat more powerful, however as it loads from disk, it is slightly slower to get started. The source code has been edited to suit a tab setting of 5, so invoke your editor with tabs set to a spacing of 5. For example, `edit sourcecode.c 5` would let you edit a file called `sourcecode.c`.

Before you can really use C, there are certain equipment requirements that must be met. You must have a disk co-processor card, and at least one disk drive. If your drives are smaller than 800k, you will probably require two disk drives. We assume you either have 1616/OS Version 4 multitasking, or else have an `assign` MRD available on your boot disk.

You should make use of the `xpath`, and the `assign` commands to set up your boot disk in a form suitable for use with C. This should be done in the `autoexec.shell` file on your boot disk, as set out below.

## 1.1 C Boot Disk

Make a new, bootable copy of your 1616 User disk, following the directions in your *Users Manual.* To ensure sufficient space, delete any obviously unwanted files you notice on the copy.

Copy the contents of your HiTech C distribution disk to the new disk, keeping the subdirectories the same as on the HiTech disk.

If you have received any updated C header files or other updates, copy these also to their respective subdirectories on your new disk.

Using `edit`, alter the `xpath` and `assign` commands in your `autoexec.shell` file in the root directory of your new disk.

Your `xpath` should include `/F0/bin` (if it is not already included).

Add the following lines to your `autoexec.shell`, to recreate the environment used by Tim Ward when originally running these programs.

```
assign /hitech /f0/bin
assign /sys  /f0/include
assign /temp /rd
```

This will allow code to be written without regard to where you actually put your files. If you are using a second drive, or a hard disk, simply change the assign to point `/hitech` to the correct drive. C tends to use temporary files extensively. If you have sufficient memory available on your ram disk, use `/rd` for temporary files. If not, use the current drive and directory, as indicated by the `assign /temp .`

Make sure you copy the new C preprocessor `relcc.xrel` from the user disk into the `/bin` subdirectory of your new C disk.

Note that `relcc` expects by default to find its C library files on the current drive in the `/hitech` directory. It also expects to find its include files on the current drive in the `/hitech/include` directory. We will explain what this means later, and there is a detailed discussion of the HiTech C compiler at the end of the tutorial.

If all is correct, you can now compile a C file by typing

```
relcc -v file.c
```

The `-v` flag is to invoke the verbose mode, which produces the maximum information from the compiler.

If you are experimenting, you may prefer to capture any errors encountered in a file, for later study. If so, use

```
relcc -v file.c } errorfile
```

## 1.2 What Is An Identifier?

Before you can do anything in any language, you must at least know how you name an identifier. An indentifier is used for any variable, function, data definition, etc. In the programming language C, an identifier is a combination of alphanumeric characters, the first being a letter of the alphabet or an underline, and the remaining being any letter of the alphabet, any numeric digit, or the underline. Two rules must be kept in mind when naming identifiers.

1. The case of alphabetic characters is significant. Using "INDEX" for a variable is not the same as using "index" and neither of them is the same as using "InDex" for a variable. All three refer to different variables.

2. As C is defined, up to eight significant characters can be used and will be considered significant. If more than eight are used, they may be ignored by the compiler. This may or may not be true of your compiler. You should check your reference manual to find out how many characters are significant for your compiler. The HiTech C compiler used with the Applix 1616 allows 31 significant characters, and prepends an underscore (_)

It should be pointed out that some C compilers allow use of a dollar sign in an identifier name, but since it is not universal, it will not be used anywhere in this tutorial. Check your documentation to see if it is permissible for your particular compiler.

## 1.3 What About The Underline?

Even though the underline can be used as part of a variable name, it seems to be used very little by experienced C programmers. It adds greatly to the readability of a program to use descriptive names for variables and it would be to your advantage to do so. Pascal programmers tend to use long descriptive names, but most C programmers tend to use short cryptic names. Most of the example programs in this tutorial use very short names for this reason.

## 1.4 How This Tutorial Is Written

Any computer program has two entities to consider, the data, and the program. They are highly dependent on one another and careful planning of both will lead to a well planned and well written program. Unfortunately, it is not possible to study either completely without a good working knowledge of the other. For this reason, this tutorial will jump back and forth between teaching methods of program writing and methods of data definition. Simply follow along and you will have a good understanding of both. Keep in mind that, even though it seems expedient to sometimes jump right into the program coding, time spent planning the data structures will be well spent and the final program will reflect the original planning.