Chapter 1: Getting Started

WHAT IS AN IDENTIFIER ?

Before you can do anything in any language, you must at least know how to name an identifier. An identifier 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. In the case of some compilers, a dollar sign is permitted but not as the first character of an identifier. It should be pointed out that even though a dollar sign may be permitted by your C compiler, it is not used anywhere in this tutorial since it is not in general use by C programmers, and is not even allowed by most compilers. If you do not plan to write any portable code, you can use it at will if you feel it makes your code more readable.

Two rules must be kept in mind when naming identifiers.

1. The case of alphabetic characters is significant. Using **INDEX** for a variable name is not the same as using **index** and neither of them is the same as using **InDeX** for a variable name. All three refer to different variables.
2. According to the ANSI-C standard, at least 31 significant characters can be used and will be considered significant by a conforming ANSI-C compiler. If more than 31 are used, they may be ignored by any given compiler.

WHAT ABOUT THE UNDERLINE ?

Even though the underline can be used as part of a variable name, and adds greatly to the readability of the resulting code, it seems to be used very little by experienced C programmers. A few underlines are used for illustration in this tutorial. Since most compiler writers use the underline as the first character for variable names internal to the system, you should refrain from using the underline to begin a variable to avoid the possibility of a name clash. To get specific, identifiers with two leading underscores are reserved for the compiler as well as identifiers beginning with a single underscore and using an upper case alphabetic character for the second. If you make it a point of style to never use an identifier with a leading underline, you will never have a naming clash with the system.

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 that reason, but a few longer names are used for illustrative purposes.

KEYWORDS

There are 32 words defined as keywords in C. These have predefined uses and cannot be used for any other purpose in a C program. They are used by the compiler as an aid to compiling the program. They are always written in lower case. A complete list follows;

```
auto      double  int       struct
break     else    long      switch
case      enum    register  typedef
```

```
char      extern  return    union
const     float   short     unsigned
continue  for     signed    void
default   goto    sizeof    volatile
do        if      static    while
```

In addition to this list of keywords, your compiler may define a few more. If it does, they will be listed in the documentation that came with your compiler. Each of the above keywords will be defined, illustrated, and used in this tutorial.

WE NEED DATA AND A PROGRAM

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 that 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 coding the program, time spent planning the data structures will be well spent and the quality of the final program will reflect the original planning.

HOW THIS TUTORIAL IS WRITTEN

As you go through the example programs, you will find that every program is complete. There are no program fragments that could be confusing. This allows you to see every requirement that is needed to use any of the features of C as they are presented. Some tutorials I have seen give very few, and very complex examples. They really serve more to confuse the student. This tutorial is the complete opposite because it strives to cover each new aspect of programming in as simple a context as possible. This method, however, leads to a lack of knowledge in how the various parts are combined. For that reason, the last chapter is devoted entirely to using the features taught in the earlier chapters. It will illustrate how to put the various features together to create a usable program. They are given for your study, and are not completely explained. Enough details of their operation are given to allow you to understand how they work after you have completed all of the previous lessons.

Throughout this tutorial, **keywords, variable names, and function names** will be given in boldface as an aid to the student. These terms will be completely defined throughout the tutorial.

RESULT OF EXECUTION

The result of executing each program will be given in comments at the end of the program listing, after the comment is defined in about the fourth program of chapter 2. If you feel confident that you completely understand the program, you can simply refer to the result of execution to see if you understand the result. In this case, it will not be necessary for you to compile and execute every program. It would be a good exercise for you to compile and execute some of them however, because all C compilers will not generate exactly the same results and you need to get familiar with your own compiler.

At this point, you should load and run firstex.c if you have not yet done so, to see that your C compiler is properly loaded and operating. Don't worry about what the program does yet. In due time you will understand it completely.

Note that this program may give you a warning that **printf()** is undefined. Whether or not you get the warning depends on your compiler and how it is set up. At this point, you can completely ignore this warning. We will cover the reason for the warning later in this tutorial.

A WORD ABOUT COMPILERS

All of the example programs in this tutorial will compile and execute correctly with any good ANSI compatible C compiler. Some compilers have gotten extremely complex and hard to use for a beginning C programmer, and some only compile and build MS Windows programs. Fortunately, most of the C compilers available have a means of compiling a standard C program which is written for the DOS environment. You should check your documentation for the capabilities and limitations of your compiler. If you have not yet purchased a C compiler, you should find one that is ANSI-C compliant, and that generates a DOS executable.

ANSWERS TO PROGRAMMING EXERCISES

There are programming exercises at the end of most of the chapters. You should attempt to do original work on each of the exercises before referring to the answers (all of which are zipped into cans.zip) in order to gain your own programming experience. These answers are given for your information in case you are completely stuck on how to solve a particular problem. These answers are not meant to be the only answer, since there are many ways to program anything, but they are meant to illustrate one way to solve the suggested programming problem.

The answers are all in surce files named in the format CHnn_m.C where nn is the chapter number, and m is the exercise number. If more than one answer is required, an A, B, or C is included following the exercise number.

Index Previous Next . . . . . *C Tutorial*

The Webwizard