A Tutorial Introduction

Let us begin with a quick introduction to C. Our aim is to show the essential elements of the language in real programs, but without getting bogged down in details, rules, and exceptions. At this point, we are not trying to be complete or even precise (save that the examples are meant to be correct). We want to get you as quickly as possible to the point where you can write useful programs, and to do that we have to concentrate on the basics: variables and constants, arithmetic, control flow, functions, and the rudiments of input and output. We are intentionally leaving out of this chapter features of C that are important for writing bigger programs. These include pointers, structures, most of C's rich set of operators, several control-flow statements, and the standard library.

This approach has its drawbacks. Most notable is that the complete story on any particular language feature is not found here, and the tutorial, by being brief, may also be misleading. And because the examples do not use the full power of C, they are not as concise and elegant as they might be. We have tried to minimize these effects, but be warned. Another drawback is that later chapters will necessarily repeat some of this chapter. We hope that the repetition will help you more than it annoys.

In any case, experienced programmers should be able to extrapolate from the material in this chapter to their own programming needs. Beginners should supplement it by writing small, similar programs of their own. Both groups can use it as a framework on which to hang the more detailed descriptions that begin in Chapter 2.

1. Getting Started

The only way to learn a new programming language is by writing programs in it. The first program to write is the same for all languages:

Print the words

```
hello, world
```

This is the big hurdle; to leap over it you must have to be able to create the program text somewhere, compile it successfully, load it, run it, and find out where your output went. With these mechanical details mastered, everything else is comparatively easy.

In C, the program to print "hello, world" is

```
#include <stdio.h>

main()
{
     printf("hello, world\n");
}
```

Just how to run this program depends on the system you are using. As a specific example, on the UNIX operating system you must create the program in a file whose name ends in ".c", such as hello.c, then compile it with the command

```
cc hello.c
```

If you haven't botched anything, such as omitting a character or misspelling something, the compilation will proceed silently, and make an executable file called a.out. If you run a.out by typing the command

```
a.out
```

it will print

```
hello, world
```

On other systems, the rules will be different; check with a local expert.

Now for some explanations about the program itself. A C program, whatever its size, consists of *functions* and *variables.* A function contains *statements* that specify the computing operations to be done, and variables store values used during the computation. C functions are like the subroutines and functions of Fortran or the procedures and functions of Pascal. Our example is a function named main. Normally you are at liberty to give functions whatever names you like, but "main" is special - your program begins executing at the beginning of main. This means that every program must have a main somewhere.

main will usually call other functions to help perform its job, some that you wrote, and others from libraries that are provided for you. The first line of the program,

```
#include <stdio.h>
```

tells the compiler to include information about the standard input/output library; this line appears at the beginning of many C source files. The standard library is described in Chapter 7 and Appendix B. One method of communicating data between functions is for the calling function to provide a list of values, called *arguments*, to the function it calls. The parentheses after the function name surround the argument list. In this example, *main* is defined to be a function that expects no arguments, which is indicated by the empty list ( ).

The statements of a function are enclosed in braces { }. The function *main* contains only one statement.

```
printf("hello, world\n");
```

A function is called by naming it, followed by a parenthesized list of arguments, so this calls the function *printf* with the argument *hello, world\n printf* is a library function that prints output, in this case the string of characters between the quotes.

A sequence of characters in double quotes like "hello, world\n", is called a *character string* or *string constant* For the moment our only use of character strings will be as arguments for printf and other functions.

The sequence \n in the string is C notation for the *newline character*, which when printed advances the output to the left margin on the next line. If you leave out the \n (a worthwhile experiment), you will find that there is no line advance after the output is printed. You must use \n to include a newline character in the printf argument; if you try something like

```
printf("hello, world
");
```

the C compiler will produce an error message.

printf never supplies a newline automatically, so several calls may be used to build up an output line in stages. Our first program could just as well have been written

```
#include <stdio.h>

main()
{
     printf("hello, ");
     printf("world");
     printf("\n");
}
```

to produce identical output.

Notice that \n represents only a single character. An *escape* like \n provides a general and extensible mechanism for representing hard-to-type or invisible characters. Among the others that C provides are \t for tab, \b for backspace, \" for the double quote, and \\ for the backslash itself. There is a complete list in Section 2.3.

Exercise 1-1. Run the "hello, world" program on your system. Experiment with leaving out parts of the program, to see what error messages you get.

Exercise 1-2. Experiment to find out what happens when printf's argument strings contains \c, where *c* is some character not listed above.

2. Variables and Arithmetic Expressions

The next programs uses the formula *°C = (5/9)(°F-32)* to print the following table of Fahrenheit temperatures and their contigrade or Celsius equivalents.

```
0       -17
20      -6
40      4
60      15
80      26
100     37
120     48
140     60
160     71
180     82
200     93
220     104
240     115
260     126
280     137
300     148
```

The program itself still consists of the definition of a single function named main. It is longer than the one that printed "hello, world", but not complicated. It introduces several new ideas, including comments, declarations, variables, arithmetic expressions, loops, and formatted output.

```
[...]
```