Second, it is robust. Over the last twenty-five years C has evolved into a stable, mature, language, controlled by a formal standard, and guided by a wealth of practical experience. You can find information about the standard at `http://std.dkuug.dk/JTC1/SC22/WG14/`, and purchase a copy of the current version, ISO/IEC 9899:1999, at `http://www.ansi.org`.

Third, it is appropriate to a wide range of applications. For example, much of the Unix operating system is written in C, and so are a wide range of programming and other software tools. Commercial software houses use it for product development, and hobbyists use it for their personal computing. And because its lineage stretches back to Fortran, a wide range of software – old, but perhaps still useful – can be rewritten in C and used if necessary, with a minimum of translation effort.

Finally, as a procedural language, there is a close mapping between constructs in the language and the facilities available in the hardware it executes on. Because of this close correspondence, a great deal of general-purpose programming work is carried out using C and related languages – programs are efficient when written in C. Advances both in programming language technology and in hardware have reduced this link in recent years, but there is still a sense in which C is close to the raw machine.

There are better languages for particular applications – just as a carpenter has many saws at their disposal. But if you are going to use only one saw, it needs to be a general-purpose one, a jack of all trades. And that is the role filled by C.

C is a robust, standardized, portable, and widely available language suitable for a broad range of computing, engineering, and scientific calculations.

1.3 A first C program

Having convinced you that C is a useful tool in your future career, it is time to look at a C program:

```
/* A first C program. Just writes a message, then exits.
   Alistair Moffat, alistair@cs.mu.oz.au, July 2002.
*/
#include <stdio.h>

int
main(int argc, char **argv) {
    printf("Hello world!\n");
    return 0;
}
```

The program in the box doesn't do much, nevertheless, it is always a useful task to get this program working on any new computer system you encounter, and to also write the equivalent program in any new language you must master.

There are a number of points to note. First, the text between the `/*` and `*/` pair is discarded by the C system, and is purely for the benefit of any human readers – it is a *comment*. Comments can be placed at almost any point of a program, and once

the `/*` is read, all further text is discarded until a `*/` combination is encountered. Unfortunately, comments cannot appear within other comments – there is no sense of nesting.

Programs typically commence with a comment that records the author of the program, a history of any modifications to the program and a set of associated dates, and a summary of the operation of the program. For brevity the programs shown in this book contain relatively terse commenting, and you should be more expansive in the software that you write.

Second, most of the rest of the program is a kind of standard recipe that is used without discussion for the next few chapters – the `#include` line and `int main` lines are going to appear exactly the same way in every program, as are the `return` statement and final closing brace.

In fact, the only interesting part in this program is the `printf` line, which says that the sequence of characters – or *string* – `Hello world!` is to be written to the output. The next box shows a possible interaction with this program on a computer running the Unix operating system, assuming that the program's lines have been typed into a file called `helloworld.c` using a program known as an *editor*. The word `vice:` is the prompt from the computer's operating system, and indicates that user input is expected. The commands beside each prompt (`ls`, which lists the files in the current directory, and `gcc`, which compiles the program) were typed by an imaginary user; the other lines resulted from the execution of those commands.

```
vice: ls
helloworld.c
vice: gcc -Wall -ansi -o helloworld helloworld.c
vice: ls
helloworld helloworld.c
vice: ./helloworld
Hello world!
vice:
```

The `gcc` command *compiles* the C source code in file `helloworld.c` using the C compiler distributed by the Free Software Foundation[2], and creates an executable file called `helloworld` that contains machine-language instructions corresponding to the C source program. On a Windows system the executable would have a `.exe` filename extension, but in Unix it is conventional to create executables that have no extension. All of the examples in this book presume a Unix environment, as shown in the example. The last command executes the compiled program, and the "Hello world!" output message appears.

Figure 1.2 shows a second C program, and an execution of it. This one reads a set of numbers, and calculates and prints their sum. It is presented to give you a feel for what a more substantial program looks like, and you are not expected to have yet mastered the intricacies of how it was put together (so please don't panic!). But with a bit of luck, by reading the code – and the comments – you can identify the basic building blocks: some variable declarations; a `while` loop that gets the input numbers one by one into variable `next`, and adds them onto a running total `sum` that is at first set to the value zero; and a `printf` that prints out the value of `sum`.

[2] See `http://www.gnu.org`.