Contents

Preface

The creation of this book has provided me, Mike McGrath, a welcome opportunity to update my previous books on C programming with the latest techniques. All examples I have given in this book demonstrate C features supported by current compilers on both Windows and Linux operating systems, and the book's screenshots illustrate the actual results produced by compiling and executing the listed code.

Conventions in this book

In order to clarify the code listed in the steps given in each example, I have adopted certain colorization conventions. Components of the C language itself are colored blue; numeric and string values are red; programmer-specified names are black; and comments are green, like this:

```
/* Store then output a text string value. */
char *myMessage = "Hello from C" ;
printf( myMessage ) ;
```

Additionally, in order to identify each source code file described in the steps, a colored icon and file name appears in the margin alongside the steps:

main.c

header.h

Grabbing the source code

For convenience I have placed source code files from the examples featured in this book into a single ZIP archive, which you can obtain by following these easy steps:

1. Browse to **www.ineasysteps.com** then navigate to Free Resources and choose the Downloads section

2. Find C Programming in easy steps, 5th edition in the list, then click on the hyperlink entitled All Code Examples to download the archive

3. Now, extract the archive contents to any convenient location on your computer

I sincerely hope you enjoy discovering the powerful expressive possibilities of C programming and have as much fun with it as I did in writing this book.

Mike McGrath

1 Getting Started

Welcome to the world of C. This chapter demonstrates how to create a C program in text, then how to compile it into executable byte form.

Introducing the C language

C is a compact, general-purpose computer programming language that was originally developed by Dennis MacAlistair Ritchie for the Unix operating system. It was first implemented on the Digital Equipment Corporation PDP-11 computer in 1972.

Dennis M Ritchie, creator of the C programming language.

This new programming language was named "C" as it succeeded an earlier programming language named "B" that had been introduced around 1970.

The Unix operating system and virtually all Unix applications are written in the C language. However, C is not limited to a particular platform and programs can be created on any machine that supports C, including those running the Windows platform.

The flexibility and portability of C made it very popular and the language was formalized in 1989 by the American National Standards Institute (ANSI). The ANSI standard unambiguously defined each aspect of C, thereby eliminating previous uncertainty about the precise syntax of the language.

ANSI C has become the recognized standard for the C language and is described, and demonstrated by examples, in this book.

Why learn C programming?

The C language has been around for quite some time and has seen the introduction of newer programming languages like Java, C++, and C#. Many of these new languages are derived, at least in part, from C – but are much larger in size. The more compact C is better to start out in programming because it's simpler to learn.

Programs written 20 years ago in C are still just as valid today as they were back then.

It is easier to move on to learn the newer languages once the principles of C programming have been grasped. For instance, C++ is an extension of C and can be difficult to learn unless you have mastered C programming first.

Despite the extra features available in newer languages, C remains popular because it is versatile and efficient. It is used today on a large number of platforms, for everything from micro-controllers to the most advanced scientific systems. Programmers around the world embrace C because it allows them maximum control and efficiency in their programs.

Standard C libraries

ANSI C defines a number of standard libraries that contain tried-and-tested functions, which can be used in your own C programs.

The libraries are contained in "header files" that each has a file extension of ".h". The names of the standard C library header files are listed in the table below with a description of their purpose:

Library:	Description:
stdio.h	Contains input and output functions, types, and macro definitions. This library is used by most C programs and represents almost one third of the entire C libraries
ctype.h	Contains functions for testing characters
string.h	Contains functions for manipulating strings
math.h	Contains mathematical functions
stdlib.h	Contains utility functions for number conversion, storage allocation, etc.
assert.h	Contains a function that can be used to add diagnostics to a program
stdarg.h	Contains a function that can be used to step through a list of function arguments
setjmp.h	Contains a function that can be used to avoid the normal call and return sequence
signal.h	Contains functions for handling exceptional conditions that may arise in a program
time.h	Contains functions for manipulating date and time components
limits.h	Contains constant definitions for the size of C data types
float.h	Contains constant definitions relating to floating-point arithmetic

A function is a piece of code that can be re-used repeatedly in a C program. A description of each function in the C library is given in the Reference section starting on page 161.

Installing a C compiler

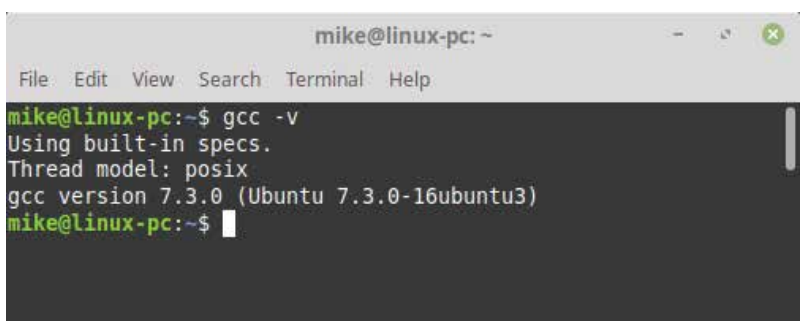
C programs are initially created as plain text files, saved with a ".c" file extension. These can be written in any plain text editor such as Windows' Notepad application – no special software is needed.

In order to execute a C program it must first be "compiled" into byte code that can be understood by the computer. A C compiler reads the original text version of the program and translates it into a second file, which is in machine-readable executable byte format.

If the text program contains any syntax errors these will be reported by the compiler, and the executable file will not be built.

One of the most popular C compilers is the GNU C Compiler (GCC) that is available free under the terms of the General Public License (GPL). It is included with almost all distributions of the Linux operating system. The GNU C Compiler is used to compile all the examples in this book into executable byte code.

To discover if you already have the GNU C Compiler on your system, type **gcc -v** at a command prompt. If it is available the compiler will respond with version information:



If you are using the Linux operating system and the GNU C Compiler is not available, install it from the distribution disk or online repository, or ask your system administrator to install it.

If you are using the Windows operating system and the GNU C Compiler is not already available, you can download and install the Minimalist GNU for Windows (MinGW) package, which includes the GNU C Compiler, by following the steps opposite.

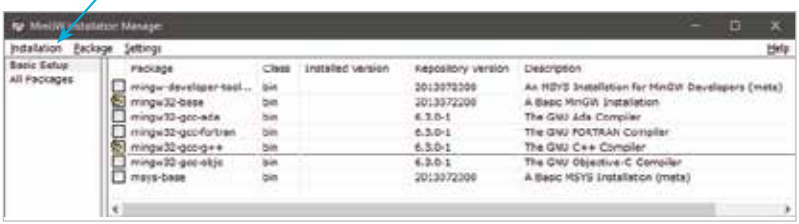
"GNU" is a recursive acronym for "Gnu's Not Unix" and it is pronounced "guh-new". You can find more details at www.gnu.org

When a C compiler is installed the standard C library header files (listed on the previous page) will also be installed.

1. With an internet connection open, launch a web browser then navigate to **sourceforge.net/projects/mingw** and click the "Download" button to get the MinGW setup installer

2. Launch the setup installer and accept the suggested location of **C:\MinGW** in the "Installation Manager" dialog

3. Choose the "Basic" and "C++ Compiler" items then click **Installation, Apply Changes** to complete the installation

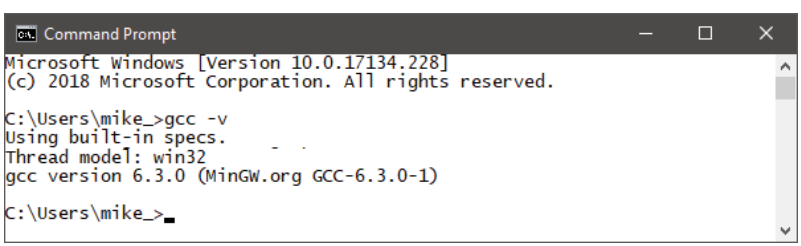


The MinGW C++ Compiler is a binary executable file located at **C:\MinGW\bin**. To allow it to be accessible from any system location this folder should now be added to the System Path:

4. In Windows' Control Panel, click the **System** icon then select the **Advanced System Settings** item to launch the "System Properties" dialog

5. In the System Properties dialog, click the **Environment Variables** button, select the **Path** system variable, then click the **Edit** button and add the location **C:\MinGW\bin;**

6. Click **OK** to close each dialog, then open a "Command Prompt" window and enter the command **gcc -v** to see the compiler respond with version information



Because C++ is an extension of C any C++ development tool can also be used to compile C programs.

Location addresses in the Path statement must end with a **;** semi-colon.

The MinGW installation process may be subject to change, but current guidance can be found at **mingw.org/wiki/Getting_Started**

Writing a C program

Do not use word processor applications to create program code as they store additional formatting information that prevents code compilation.

In C programs the code statements to be executed are contained within "functions", which are defined using this syntax format:

data-type function-name **( )** **{** ***statements-to-be-executed*** **}**

After a function has been called upon to execute the statements it contains, it can return a value to the caller. This value must be of the data type specified before the function name.

A program can contain one or many functions but must always have a function named "main". The **main()** function is the starting point of all C programs, and the C compiler will not compile the code unless it finds a **main()** function within the program.

Other functions in a program may be given any name you like using letters, digits, and the underscore character, but the name may not begin with a digit. Also, the C keywords, listed in the table on the front inner cover of this book, must be avoided.

Preprocessor instructions begin with a **#** hash character and must enclose standard library names within **< >** angled brackets.

The **( )** parentheses that follow the function name may, optionally, contain values to be used by that function. These take the form of a comma-separated list and are known as function "arguments" or "parameters".

The **{ }** curly brackets (braces) contain the statements to be executed whenever that function is called. Each statement must be terminated by a semi-colon, in the same way that English language sentences must be terminated by a period/full stop.

Traditionally, the first program to attempt when learning any programming language is that which simply generates the message "Hello World".

hello.c

1. Open a plain text editor, such as Notepad, then type this line of code at the start of the page, exactly as it is listed

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

The program begins with an instruction to the C compiler to include information from the standard input/output **stdio.h** library file. This makes the functions contained within that library available for use within this program. The instruction is more properly called a "preprocessor instruction" or "preprocessor directive" and must always appear at the start of the page, before the actual program code is processed.

2 Two lines below the preprocessor instruction, add an empty main function

```
int main()
{

}
```

This function declaration specifies that an integer value, of the **int** data type, should be returned by the function upon completion.

3 Between the braces, insert a line of code that calls upon one of the functions defined in the standard input/output library – made available by the preprocessor instruction

```
printf ( "Hello World!\n" ) ;
```

Here the **printf()** function specifies a single string argument between its parentheses. In C programming, strings must always be enclosed within double quotes. This string contains the text **Hello World** and the **\n** "newline" escape sequence that moves the print head to the left margin of the next line.

4 Between the braces, insert a final line of code to return a zero integer value, as required by the function declaration

```
return 0 ;
```

Traditionally, returning a value of zero after the execution of a program indicates to the operating system that the program executed correctly.

5 Check that the program code looks exactly like the listing below, then add a final newline character (hit Return after the closing brace) and save the program as "hello.c"

```
#include <stdio.h>

int main()
{
  printf( "Hello World!\n" ) ;
  return 0 ;
}
```

The complete program in text format is now ready to be compiled into machine-readable byte format as an executable file.

Whitespace between the code is ignored by the C compiler but program code should always end with a newline character.

Each statement must be terminated by a semi-colon character.

Compiling a C program

At a command prompt, type **gcc --help** then hit Return to see a list of all compiler options.

The C source code files for the examples in this book are stored in a directory created expressly for that purpose. The directory is named "MyPrograms" and its absolute address on Windows is **C:\MyPrograms**, whereas on Linux it's at **/home/*user*/MyPrograms**. The **hello.c** source code file, created by following the steps on pages 12-13, is saved in this directory awaiting compilation to produce a version in executable byte code format.

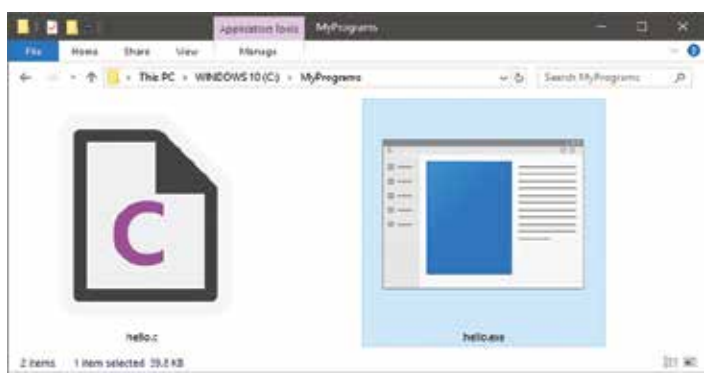
1. At a command prompt, issue a **cd** command with the path to the **MyPrograms** directory to navigate there

2. At a command prompt in the **MyPrograms** directory, type **gcc hello.c** then hit Return to compile the program

When the compilation succeeds, the compiler creates an executable file alongside the original source code file. By default, this file will be named **a.out** on Linux systems and **a.exe** on Windows systems. Compiling a different C source code file in the **MyPrograms** directory would now overwrite the first executable file without warning. This is obviously unsatisfactory so a custom name for the executable file must be specified when compiling **hello.c**. This can be achieved by including a **-o** option followed by a custom name in the compiler command.

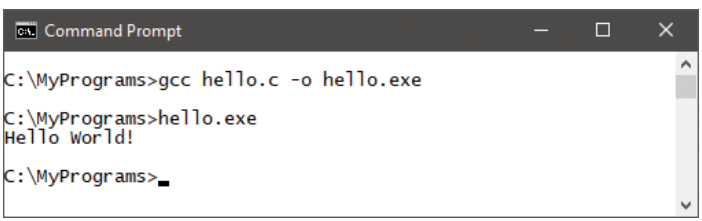
3. At a command prompt in the **MyPrograms** directory, type **gcc hello.c -o hello.exe** then hit Return to compile the program once more

On both Linux and Windows systems an executable file named **hello.exe** is now created alongside the C source code file:



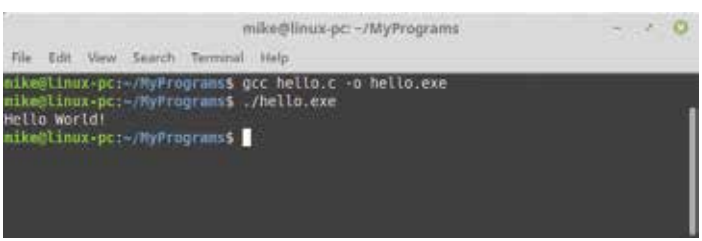
If the compiler complains that there is no new line at the end of the file add a carriage return to the end of the source code, then save and retry.

4 At a command prompt in Windows, type the executable filename then hit Return to run the program – the text string is output and the print head moves to the next line



Because Linux does not by default look in the current directory for executable files, unless it is specifically directed to do so, it is necessary to prefix the filename with **./** to execute the program.

5 At a command prompt in Linux, type **./hello.exe** then hit Return to run the program – the text string is output and the print head moves to the next line

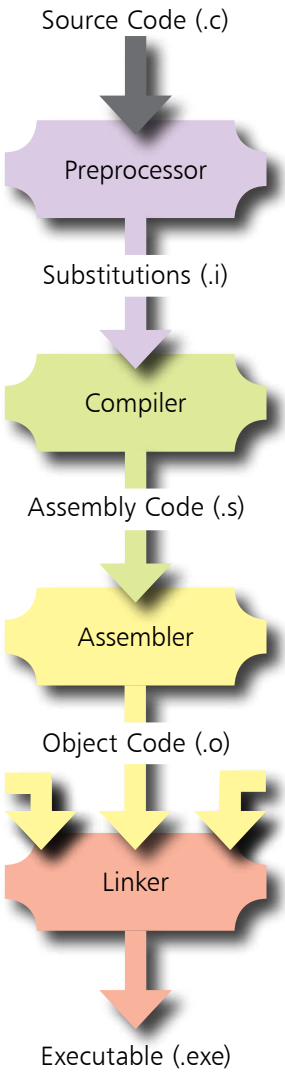


Windows users can even omit the file extension to run programs. In this case, typing just **hello** is sufficient.

You have now created, compiled, and executed the simple Hello World program that is the starting point in C programming. All other examples in this book will be created, compiled, and executed in the same way.

Understanding compilation

In producing an executable file from an original C source code file, the compilation process actually undergoes four separate stages, which each generate a new file:



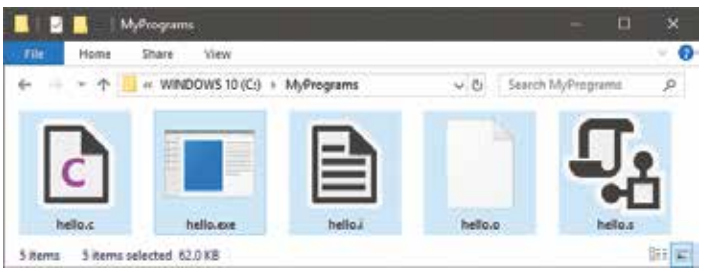
- Preprocessing – The preprocessor substitutes all preprocessor directives in the original source code **.c** file with actual library code that implements those directives. For instance, library code is substituted for **#include** directives. The generated file containing the substitutions is in text format and typically has a **.i** file extension.
- Translating – The compiler translates the high-level instructions in the **.i** file into low-level Assembly language instructions. The generated file containing the translation is in text format and typically has a **.s** file extension.
- Assembling – The assembler converts the Assembly language text instructions in the **.s** file into machine code. The generated object file containing the conversion is in binary format and typically has a **.o** file extension.
- Linking – The linker combines one or more binary object **.o** files into a single executable file. The generated file is in binary format and typically has a **.exe** file extension.

Strictly speaking, "compilation" describes the first three stages above, which operate on a single source code text file and ultimately generate a single binary object file. Where the program source code contains syntax errors, such as a missing semi-colon statement terminator or a missing parenthesis, they will be reported by the compiler and compilation will fail.

The linker, on the other hand, can operate on multiple object files and ultimately generates a single executable file. This allows the creation of large programs from modular object files that may each contain reusable functions. Where the linker finds a function of the same name defined in multiple object files it will report an error and the executable file will not be created.

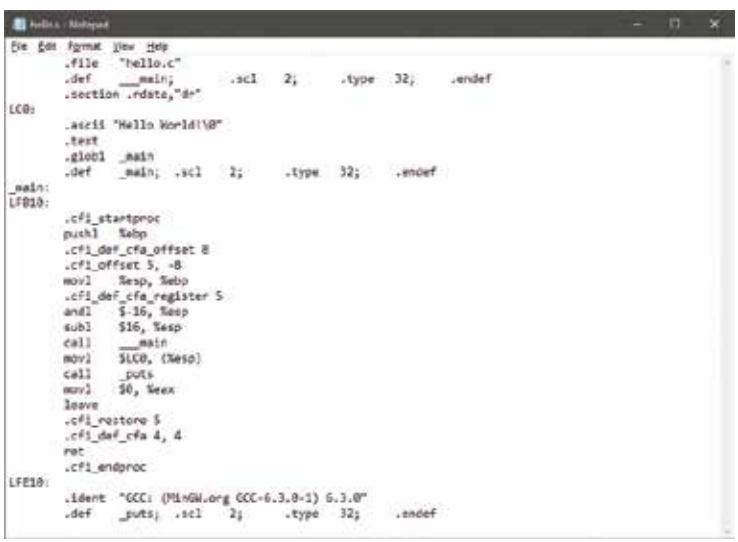
Normally, the temporary files created during the intermediary stages of the compilation process are automatically deleted, but they can be retained for inspection by including a **-save-temps** option in the compiler command:

1. At a command prompt in the **MyPrograms** directory, type **gcc hello.c -save-temps -o hello.exe** then hit Return to recompile the program and save the temporary files



2. Open the **hello.i** file in a plain text editor such as Windows' Notepad, to see your source code at the very end of the file preceded by substituted **stdio.h** library code

3. Now, open the **hello.s** file in a plain text editor to see the translation into low-level Assembly code and note how unfriendly that appears in contrast to the C code version



```
File  Edit  Format  View  Help
        .file   "hello.c"
        .def    ___main;        .scl    2;      .type   32;     .endef
        .section .rdata,"dr"
LC0:
        .ascii "Hello World!\0"
        .text
        .globl  _main
        .def    _main;  .scl    2;      .type   32;     .endef
_main:
LFB10:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        andl    $-16, %esp
        subl    $16, %esp
        call    ___main
        movl    $LC0, (%esp)
        call    _puts
        movl    $0, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
LFE10:
        .ident  "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
        .def    _puts;  .scl    2;      .type   32;     .endef
```

Programs tediously written in Assembly language can run faster than those written in C but are more difficult to develop and maintain. For traditional computer programming, C is almost always the first choice.

Summary

- The American National Standards Institute (ANSI) established the recognized standard for the C programming language.
- Other programming languages, such as C++ and C#, are derived in part from the C language.
- The C language has a number of standard libraries containing tried-and-tested functions that can be used in any program.
- C libraries are contained in header files whose names have a **.h** file extension.
- C programs are created as plain text files whose names have a **.c** file extension.
- The popular GNU C Compiler (GCC) is included in the Minimalist GNU for Windows (MinGW) package.
- Adding the compiler's host directory to the system path conveniently allows the compiler to be run from any directory.
- Programs have one or more functions containing statements to be executed whenever the function is called.
- Every C program must have a **main()** function.
- A function declaration begins by specifying the data type of the value to be returned after the function has been executed.
- The statements to be executed are contained within **{ }** braces and each statement must end with a **;** semi-colon terminator.
- Preprocessor instructions are implemented in the first stage of program compilation and will typically substitute library code.
- The GNU C Compiler is run with the **gcc** command and may include a **-o** option to name the executable output file.
- Temporary files created during the compilation process can be retained using the **-save-temps** compiler command option.