Contents

How to use this book

The creation of this book has provided me, Mike McGrath, a welcome opportunity to update my previous books on coding programs with the C, C++ and C# programming languages. Examples I provide in this book demonstrate C and C++ features supported by current compilers on both Windows and Linux operating systems, and demonstrate C# features in the Microsoft Visual Studio development suite – all **in easy steps**.

Conventions in this book

In order to clarify the code listed in the steps, I have adopted certain colorization conventions. Keywords of each language are **Blue**, numeric and string values are **Red**, programmer-specified names are **Black**, comments are **Green**, C# classes are **LightBlue** and C# methods are **Brown**:

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

// Store then output a text string value.
string myMessage = "Hello from C++!" ;
cout << myMessage ;

// Store then output a text string value.
string myMessage = "Hello from C#!" ;
Console.Write( 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:

C hello.c   C++ hello.cpp   C# Hello   H header.h

Grabbing the source code

For convenience I have placed source code files from all examples featured in this book into a single ZIP archive. To download the archive visit **www.ineasysteps.com** and sign in, then select **Free Resources**, then select **BROWSE NOW** in the **Source code and other book resources** section and choose **All code examples** for the C, C++ & C# book. This book's screenshots illustrate the actual results produced by compiling and executing the listed code in each example. If you don't achieve the illustrated result, simply compare your code to that in the original source code to discover where you went wrong.

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

Mike McGrath

1 Getting started with C

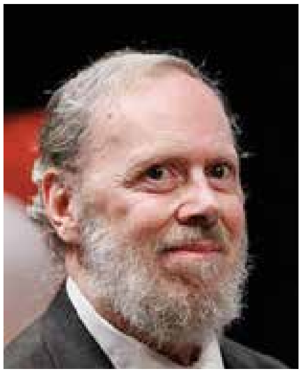
Welcome to the world of C. This chapter demonstrates how to create a C program in text, then how to compile it into executable byte form.

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.

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.


Dennis M Ritchie, creator of the C programming language.

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.

Don't forget

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

Beware

The keywords listed below have special significance in C programming and may not be used for other purposes.

C Keywords	
auto	**int**
break	**long**
case	**register**
char	**return**
const	**short**
continue	**signed**
default	**sizeof**
do	**static**
double	**struct**
else	**switch**
enum	**typedef**
extern	**union**
float	**unsigned**
for	**void**
goto	**volatile**
if	**while**

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.

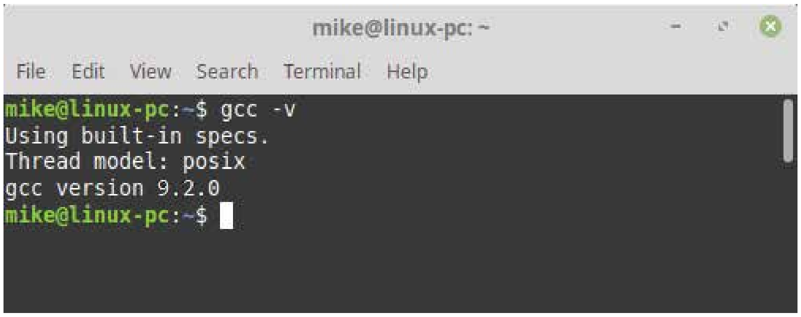
In order to execute a C program it must first be "compiled" into byte code that can be understood by the computer. A C compiler reads the original text version of the program and translates it into a second file, which is in machine-readable executable byte format.

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 the C and C++ sections of this book into executable byte code.

"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**

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:



```
mike@linux-pc:~$ gcc -v
Using built-in specs.
Thread model: posix
gcc version 9.2.0
mike@linux-pc:~$
```

Don't forget

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

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.

...cont'd

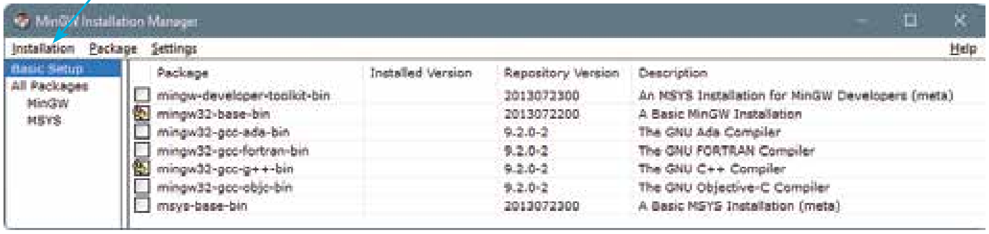
1. With an internet connection open, launch a web browser then navigate to **sourceforge.net/projects/mingw** and click the "Download" button to get the MinGW setup installer

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

Hot tip

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

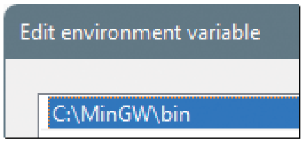


	Package	Installed Version	Repository Version	Description
☐	mingw-developer-toolkit-bin		2013072300	An MSYS Installation for MinGW Developers (meta)
☑	mingw32-base-bin		2013072200	A Basic MinGW Installation
☐	mingw32-gcc-ada-bin		9.2.0-2	The GNU Ada Compiler
☐	mingw32-gcc-fortran-bin		9.2.0-2	The GNU FORTRAN Compiler
☑	mingw32-gcc-g++-bin		9.2.0-2	The GNU C++ Compiler
☐	mingw32-gcc-objc-bin		9.2.0-2	The GNU Objective-C Compiler
☐	msys-base-bin		2013072300	A Basic MSYS Installation (meta)

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:

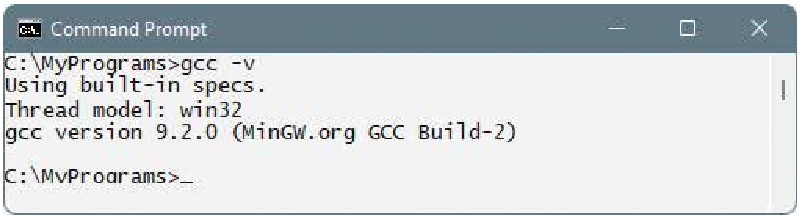
4. In Windows' Control Panel, click the **System** icon then select the **Advanced System Settings** item to launch the "System Properties" dialog

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**



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



```
C:\MyPrograms>gcc -v
Using built-in specs.
Thread model: win32
gcc version 9.2.0 (MinGW.org GCC Build-2)

C:\MyPrograms>_
```

Don't forget

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

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.

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 semicolon, 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".

Beware

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

Don't forget

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

C

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.

Hot tip

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

Don't forget

Each statement must be terminated by a semicolon character.

Compiling a C program

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 18-19, is saved in this directory awaiting compilation to produce a version in executable byte code format.

Hot tip

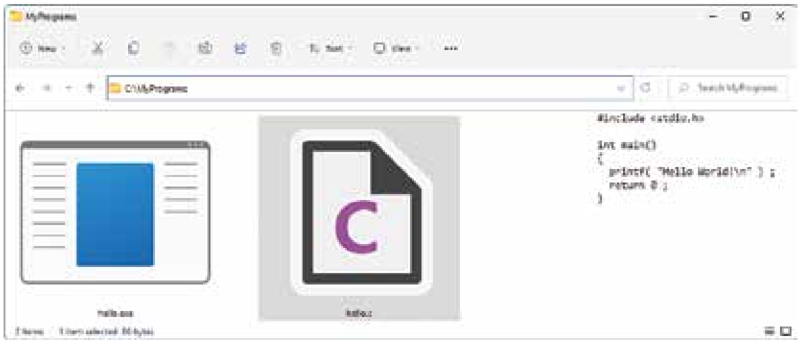
At a command prompt, type **gcc --help** then hit Return to see a list of all compiler options.

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.

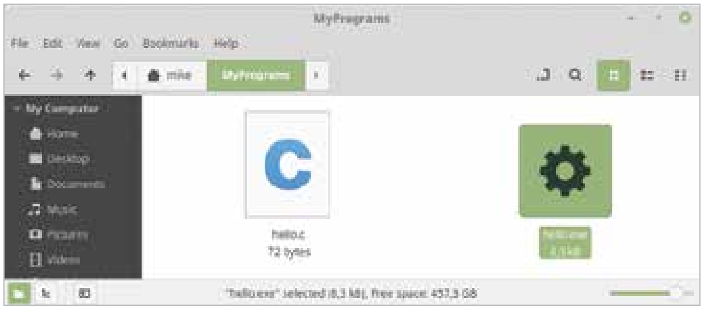
3. At a command prompt in the **MyPrograms** directory, type **gcc hello.c -o hello.exe** then hit Return to compile the program once more

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

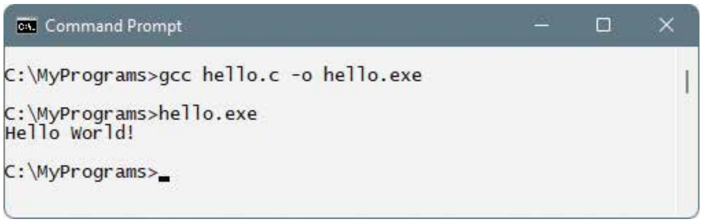


```
#include <stdio.h>

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



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



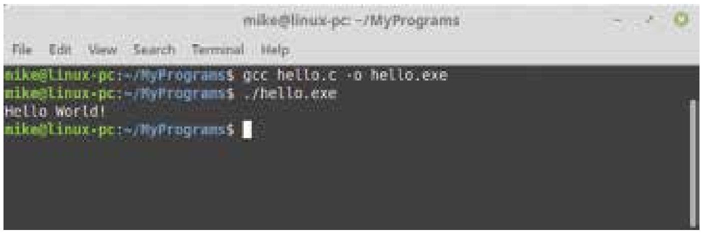
```
C:\MyPrograms>gcc hello.c -o hello.exe

C:\MyPrograms>hello.exe
Hello World!

C:\MyPrograms>_
```

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


mike@linux-pc: ~/MyPrograms
File Edit View Search Terminal Help

```
mike@linux-pc:~/MyPrograms$ gcc hello.c -o hello.exe
mike@linux-pc:~/MyPrograms$ ./hello.exe
Hello World!
mike@linux-pc:~/MyPrograms$ 
```

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

Don't forget

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.

Hot tip

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

12 Getting started with C++

Welcome to the exciting world of C++ programming. This chapter demonstrates how to create a simple C++ program and how to store data within a program.

Introducing C++

C++ is an extension of the C programming language that was first implemented on the UNIX operating system by Dennis Ritchie way back in 1972. C is a flexible programming language that remains popular today, and is used on a large number of platforms for everything from microcontrollers to the most advanced scientific systems.

C++ was developed by Dr. Bjarne Stroustrup between 1983 and 1985 while working at AT&T Bell Labs in New Jersey. He added features to the original C language to produce what he called "C with classes". These classes define programming objects with specific features that transform the procedural nature of C into the object-oriented programming language of C++.

Hot tip

A powerful programming language (pronounced "see plus plus"), designed to let you express ideas.

The C programming language was so named as it succeeded an earlier programming language named "B" that had been introduced around 1970. The name "C++" displays some programmers' humor because the programming ++ increment operator denotes that C++ is an extension of the C language.

C++, like C, is not platform-dependent, so programs can be created on any operating system. Most illustrations in this book depict output on the Windows operating system purely because it is the most widely used desktop platform. The examples can also be created on other platforms such as Linux or macOS.

Why learn C++ programming?

The C++ language is favored by many professional programmers because it allows them to create fast, compact programs that are robust and portable.

Don't forget

Microsoft's free Visual Studio Community Edition IDE is used in this book to demonstrate visual programming.

Using a modern C++ Integrated Development Environment (IDE), such as Microsoft's Visual Studio Community Edition, the programmer can quickly create complex applications. But to use these tools to greatest effect, the programmer must first learn quite a bit about the C++ language itself.

This section of the book is an introduction to programming with C++, giving examples of program code and its output to demonstrate the basics of this powerful language.

Should I learn C first?

Opinion is divided on the question of whether it is an advantage to be familiar with C programming before moving on to C++. It would seem logical to learn the original language first in order to understand the larger extended language more readily. However, C++ is not simply a larger version of C, as the approach to object-oriented programming with C++ is markedly different to the procedural nature of C. It is, therefore, arguably better to learn C++ without previous knowledge of C to avoid confusion.

This section of the book makes no assumption that the reader has previous knowledge of any programming language, so it is suitable for the beginner to programming in C++, whether they know C or not.

If you do feel that you would benefit from learning to program in C before moving on to C++, we recommend you try the examples in the C section of this book before moving on to this C++ section.

Standardization of C++

As the C++ programming language gained in popularity, it was adopted by many programmers around the world as their programming language of choice. Some of these programmers began to add their own extensions to the language, so it became necessary to agree upon a precise version of C++ that could be commonly shared internationally by all programmers.

A standard version of C++ was defined by a joint committee of the American National Standards Institute (ANSI) and the Industry Organization for Standardization (ISO). This version is sometimes known as ANSI C++, and is portable to any platform and to any development environment.

The examples given in this section conform to ANSI C++. Example programs run in a console window, such as the Command Prompt window on Windows systems or a shell terminal window on Linux systems, to demonstrate the mechanics of the C++ language itself. An example in Chapter 21 illustrates how code generated automatically by a visual development tool on the Windows platform can, once you're familiar with the C++ language, be edited to create a graphical, windowed application.

Beware

In addition to the C keywords listed on page 15, C++ has the extra keywords listed below. All keywords have special significance and may not be used for other purposes.

Extra C++ Keywords	
catch	private
class	protected
delete	public
friend	template
mutable	this
new	throw
operator	virtual

Hot tip

"ISO" is not an acronym but is derived from the Greek word "isos" meaning "equal" – as in "isometric".

Installing a compiler

The GNU C++ compiler is available free under the terms and conditions of the General Public License (GPL) that can be found online at **gnu.org/copyleft/gpl.html**

C++ programs are initially created as plain text files, saved with the file extension of ".cpp". These can be written in any text editor, such as Windows' Notepad application or the Vi editor on Linux.

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 text version of the program and translates it into a second file – in machine-readable, executable format.

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

If you are using the Windows platform and have a C++ Integrated Development Environment (IDE) installed, then you will already have a C++ compiler available, as the compiler is an integral part of the visual IDE. The excellent, free Microsoft Visual C++ Express IDE provides an editor window, where the program code can be written, and buttons to compile and execute the program. Visual IDEs can, however, seem unwieldy when starting out with C++ because they always create a large number of "project" files that are used by advanced programs.

The popular free GNU Compiler Collection, which includes a C Compiler, is included with most distributions of the Linux operating system. The GNU C++ Compiler is also available for Windows platforms and is used to compile examples throughout the C and C++ sections of this book.

Hot tip

To open a Windows Command Prompt, press the **Windows** + **R** keys to launch a Run dialog, then type **cmd** into the dialog and hit **Enter**.

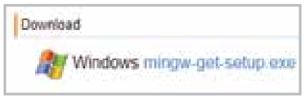
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

To discover if you already have the GNU C++ Compiler on your system, type **c++ -v** at a command prompt then hit **Return**. If it's available, the compiler will respond with version information. If you are using the Linux platform and the GNU C++ Compiler is not available on your computer, install it from the distribution disc, download it from the GNU website, or ask your system administrator to install it.

The GNU (pronounced "guh-new") Project was launched back in 1984 to develop a complete free Unix-like operating system. Part of GNU is "Minimalist GNU for Windows" (MinGW). MinGW includes the GNU C++ Compiler that can be used on Windows systems to create executable C++ programs. Windows users can download and install the GNU C++ Compiler by following the instructions on the opposite page.

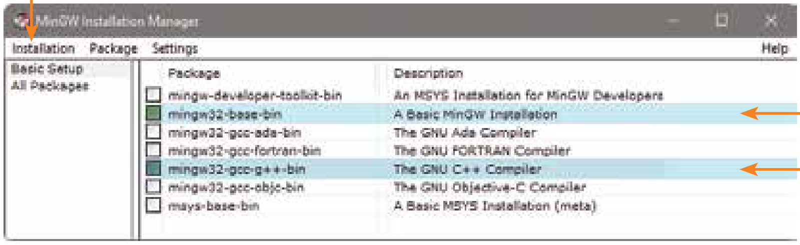
...cont'd

1. With an internet connection, launch a web browser then navigate to **osdn.net/projects/mingw** and click the link to download the MinGW installer **mingw-get-setup.exe**


Download
Windows mingw-get-setup.exe

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

3. Check the **Basic MinGW** and **C++ Compiler** items, then click **Installation**, **Apply Changes**, **Apply** to install



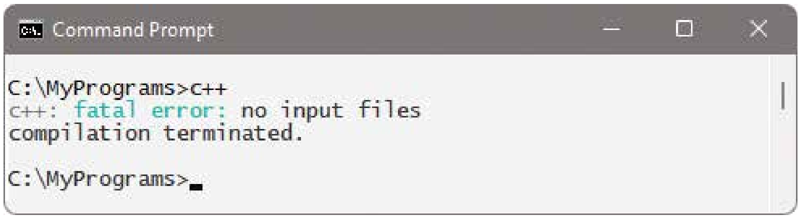
	Package	Description
☐	mingw-developer-toolkit-bin	An MSYS Installation for MinGW Developers
🟩	mingw32-base-bin	A Basic MinGW Installation
☐	mingw32-gcc-ada-bin	The GNU Ada Compiler
☐	mingw32-gcc-fortran-bin	The GNU FORTRAN Compiler
🟩	mingw32-gcc-g++-bin	The GNU C++ Compiler
☐	mingw32-gcc-objc-bin	The GNU Objective-C Compiler
☐	msys-base-bin	A Basic MSYS Installation (meta)

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. Open Windows' "System Properties" dialog, then select the **Advanced** tab and click the **Environment Variables** button – to open the "Environment Variables" dialog

5. 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 **c++**. If the installation is successful, the compiler should respond that you have not specified any input files for compilation:

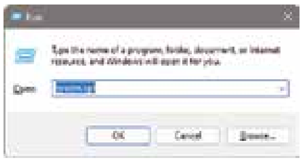


```
C:\MyPrograms>c++
c++: fatal error: no input files
compilation terminated.

C:\MyPrograms>_
```

Hot tip

To open a System Properties dialog, press the **Windows** + **R** keys to launch a Run dialog, then type **sysdm.cpl** into the dialog and hit **Enter**.



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open:

Writing your first program

Follow these steps, copying the code exactly as it is listed, to create a simple C++ program that will output the traditional first program greeting:

1. Open a plain text editor, such as Windows' Notepad, then type these "preprocessor directives"

```
#include <iostream>
using namespace std ;
```

2. A few lines below the preprocessor directives, add a "comment" describing the program

```
// A C++ Program to output a greeting.
```

3. Below the comment, add a "main function" declaration to contain the program statements

```
int main()
{

}
```

4. Between the curly brackets (braces) of the main function, insert this output "statement"

```
cout << "Hello World!" << endl ;
```

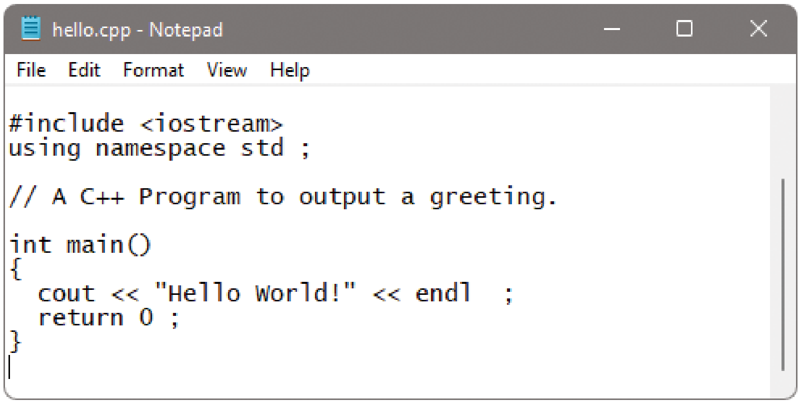
5. Next, insert a final "return" statement in the main function

```
return 0 ;
```

6. Save the program to any convenient location as "hello.cpp" – the complete program should look like this:

Don't forget

Comments throughout this book are shown in green – to differentiate them from other code.



```
#include <iostream>
using namespace std ;

// A C++ Program to output a greeting.

int main()
{
  cout << "Hello World!" << endl  ;
  return 0 ;
}
```

Beware

After typing the final closing **}** brace of the main method, always hit **Return** to add a newline character – your compiler may insist that a source file should end with a newline character.

The separate parts of the program code on the opposite page can be examined individually to understand each part more clearly:

- **Preprocessor Directives** – these are processed by the compiler before the program code, so must always appear at the start of the page. Here, the **#include** directive instructs the compiler to use the standard C++ input/output library named **iostream**, specifying the library name between **< >** angled brackets. The next line is the "using directive" that allows functions in the specified namespace to be used without their namespace prefix. Functions of the **iostream** library are within the **std** namespace – so this **using** directive allows functions such as **std::cout** and **std::endl** to be simply written as **cout** and **endl**.
- **Comments** – these should be used to make the code more easily understood by others, and by yourself when revisiting the code later. In C++ programming, everything on a single line after a **//** double-slash is ignored by the compiler.
- **Main function** – this is the mandatory entry point of every C++ program. Programs may contain many functions, but they must always contain one named **main**, otherwise the compiler will not compile the program. Optionally, the parentheses after the function name may specify a comma-separated list of "argument" values to be used by that function. Following execution, the function must return a value to the operating system of the data type specified in its declaration – in this case, an **int** (integer) value.
- **Statements** – these are the actions that the program will execute when it runs. Each statement must be terminated by a semicolon, in the same way that English language sentences must be terminated by a period (full stop). Here, the first statement calls upon the **cout** library function to output text and an **endl** carriage return. These are directed to standard output by the **<<** output stream operator. Notice that text strings in C++ must always be enclosed within double quotes. The final statement employs the C++ **return** keyword to return a zero integer value to the operating system – as required by the main function declaration. Traditionally, returning a zero value indicates that the program executed successfully.

Hot tip

The C++ compiler also supports multiple-line C-style comments between /* and */ – but these should only ever be used in C++ programming to "comment-out" sections of code when debugging.

Hot tip

Notice how the program code is formatted using spacing and indentation (collectively known as whitespace) to improve readability. All whitespace is ignored by the C++ compiler.

Compiling & running programs

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" – its absolute address on a Windows system is **C:\MyPrograms** and on Linux it's **/home/*user*/MyPrograms**. You can recreate this directory to store programs awaiting compilation:

1. Move the "hello.cpp" program source code file, created on page 198, to the "MyPrograms" directory on your system

2. At a command prompt, use the "cd" command to navigate to the "MyPrograms" directory

3. Enter a command to attempt to compile the program
c++ hello.cpp

Hot tip

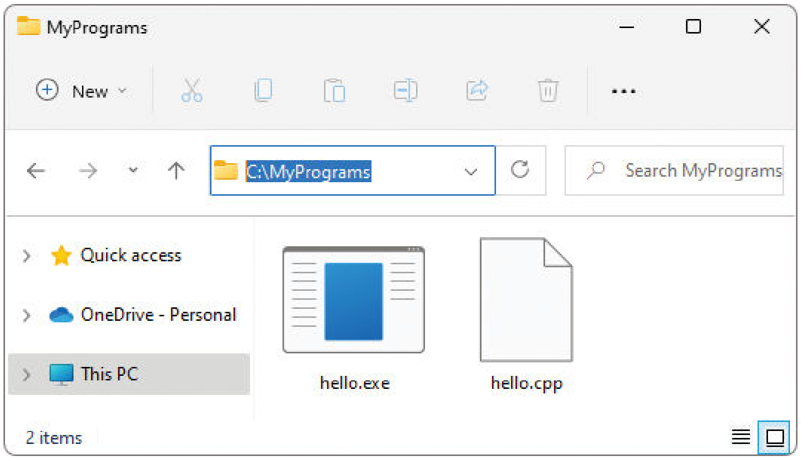
You can see the compiler version number with the command **c++ --version** and display all its options with **c++ --help**

When the attempt succeeds, the compiler creates an executable file alongside the original source code file. By default, the executable file is named **a.exe** on Windows systems and **a.out** on Linux. Compiling a different source code file in the same directory would now overwrite the first executable file without warning. This is obviously undesirable, so a custom name for the executable file should be specified when compiling programs, using the compiler's **-o** option in the compile command.

4. Enter a command to compile the program, creating an executable file named "hello.exe" alongside the source file
c++ hello.cpp -o hello.exe

Don't forget

The command **c++** is an alias for the GNU C++ Compiler – the command **g++** can also be used.



22 Getting started with C#

Welcome to the exciting world of C# programming. This chapter introduces the Visual Studio Integrated Development Environment and shows you how to create a real Windows application.

Introducing C#

The introduction of the Microsoft .NET framework at the Professional Developers Conference in July 2000 also saw Microsoft introduce a new programming language called C# (pronounced "see-sharp"). The name was inspired by musical notation where a # sharp symbol indicates that a written note should be a semitone higher in pitch. This notion is similar to the naming of the C++ programming language where the ++ symbol indicates that a written value should be incremented by 1.

- C# is designed to be a simple, modern, general-purpose, object-oriented programming language, borrowing key concepts from several other languages – most notably the Java programming language. Consequently, everything in C# is a class "object" with "properties" and "methods" that can be employed by a program.
- C# is an elegant and "type-safe" programming language that enables developers to build a variety of secure and robust applications. You can use C# to create Windows client applications, XML web services, distributed components, client-server applications, database applications, and much, much more.
- C# is specifically designed to utilize the proven functionality built into the .NET framework "class libraries". Windows applications written in C# therefore require the Microsoft .NET framework to be installed on the computer running the application – typically, an integral component of the system.

The Microsoft .NET Framework

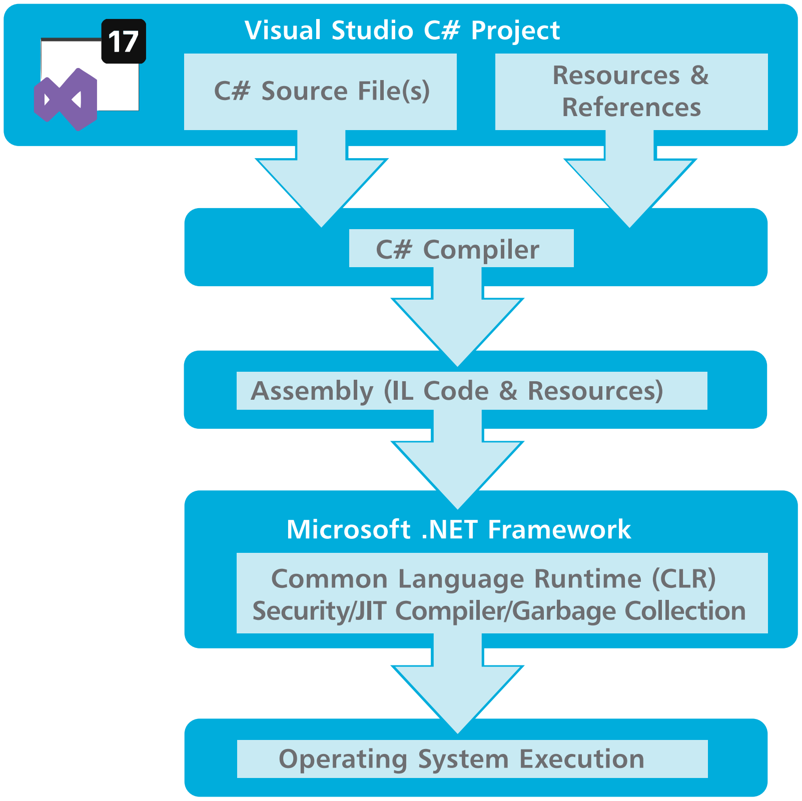
Each version of the Microsoft .NET framework includes a unified set of class libraries and a virtual execution system called the Common Language Runtime (CLR). The CLR allows the C# language and the class libraries to work together seamlessly.

To create an executable program, source code written in the C# language is compiled by the C# Compiler into Intermediate Language (IL) code. This is stored on disk, together with other program resources such as images, in an "assembly". Typically, the assembly will have a file extension of **.exe** or **.dll**. Each assembly contains a "manifest" that provides information about that program's security requirements.

When a C# program is executed, the assembly is loaded into the Common Language Runtime (CLR), and the security requirements specified in its assembly manifest are examined. When the security requirements are satisfied, the CLR performs Just-In-Time (JIT) compilation of the IL code into native machine instructions. The CLR then performs "garbage collection", exception handling, and resource management tasks before calling upon the operating system to execute the program:

Hot tip

Just-In-Time compilation is also known as "Dynamic Translation".



Visual Studio C# Project
17
C# Source File(s)
Resources & References
C# Compiler
Assembly (IL Code & Resources)
Microsoft .NET Framework
Common Language Runtime (CLR)
Security/JIT Compiler/Garbage Collection
Operating System Execution

Don't forget

Just-In-Time compilation occurs during program execution, rather than prior to its execution.

As language interoperability is a key feature of the Microsoft .NET framework, the IL code generated by the C# Compiler can interact with code generated by the .NET versions of other languages such as Visual Basic and Visual C++. The examples throughout the C# section of this book demonstrate Visual C# program code.

Installing Visual Studio

In order to create Windows applications with the C# programming language, you will first need to install a Visual Studio Integrated Development Environment (IDE).

Microsoft Visual Studio is the professional development tool that provides a fully Integrated Development Environment for Visual Basic, Visual C++, Visual J#, and Visual C#. Within its IDE, code can be written in Visual Basic, C++, J#, or the C# programming language to create Windows applications.

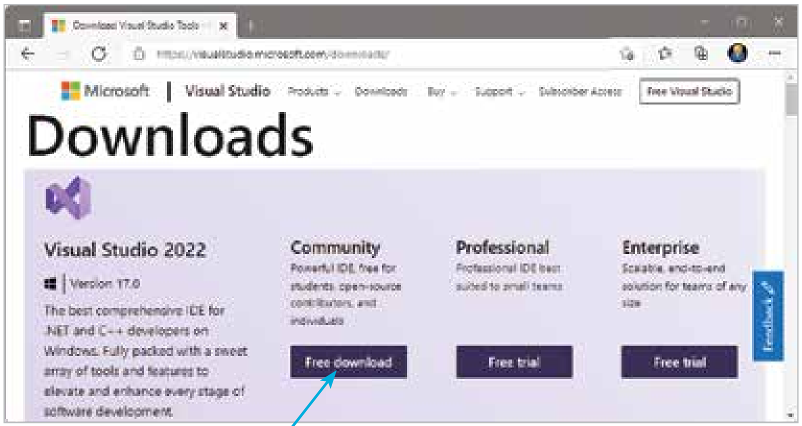
Visual Studio Community edition is a streamlined version of Visual Studio, specially created for those people learning programming. It has a simplified user interface and omits advanced features of the professional edition to avoid confusion. C# code can be written within the **Code Editor** of either version of the Visual Studio IDE to create Windows applications.

Both Visual Studio and Visual Studio Community provide an IDE for C# programming but, unlike the fully-featured Visual Studio product, the Visual Studio Community edition is completely free and can be installed on any system meeting the following minimum requirements:

Component	Requirement
Operating system	Windows 11 Windows 10 (version 1909 or higher) Windows Server 2016 or 2019 *Must be the 64-bit version of any of the above the operating systems.
CPU (processor)	1.8 GHz or faster, 64-bit processor
RAM (memory)	4 GB (16 GB recommended)
HDD (hard drive)	Up to 210 GB available space
Video Card	Minimum resolution of 1366 x 768 Optimum resolution of 1920 x 1080

The Visual Studio Community edition is used throughout the C# section of this book to demonstrate programming with the C# language, but the examples can also be recreated in Visual Studio. Follow the steps opposite to install the Visual Studio Community edition.

1. Open your web browser and navigate to the Visual Studio download page – at the time of writing, this can be found at **visualstudio.microsoft.com/downloads**


Download Visual Studio Tools
https://visualstudio.microsoft.com/downloads/
Microsoft | Visual Studio Products Downloads Buy Support Subscriber Access Free Visual Studio
Feedback
Free download
Free trial
Free trial

Downloads

Visual Studio 2022

Version 17.0

The best comprehensive IDE for .NET and C++ developers on Windows. Fully packed with a sweet array of tools and features to elevate and enhance every stage of software development.

Community

Powerful IDE, free for students, open-source contributors, and individuals

Professional

Professional IDE best suited to small teams

Enterprise

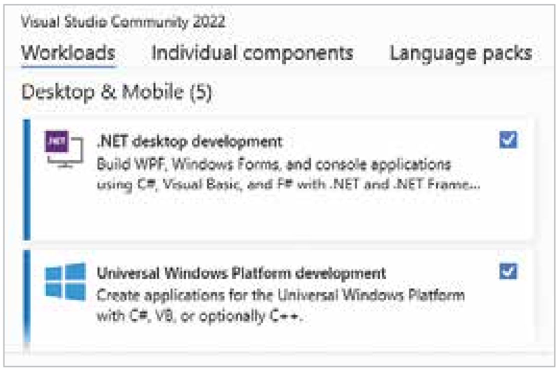
Scalable, end-to-end solution for teams of any size

2. Click the button in the Community edition section to download a **VisualStudioSetup.exe** setup file

3. Click on the setup file icon to begin setup and to run the **Visual Studio Installer**

4. Accept the suggested installation location, then click **Next**

5. Check the two C# **Installer** options shown below



.NET desktop development ✓

Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...

Universal Windows Platform development ✓

Create applications for the Universal Windows Platform with C#, VB, or optionally C++.

6. Click the **Install** button at the bottom-right of the installer to begin the download and installation process

Beware

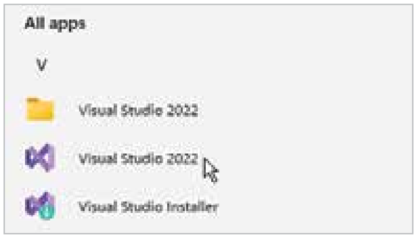
Choosing a different destination folder may require other paths to be adjusted later – it's simpler to just accept the suggested default.

Hot tip

Both **Visual Studio** and **Visual Studio Installer** items get added to your **All Apps** menu. You can re-run the installer at a later date to add or remove features.

Exploring the IDE

1. Go to your **All apps** menu, then select the Visual Studio 2022 menu item added there by the installer:

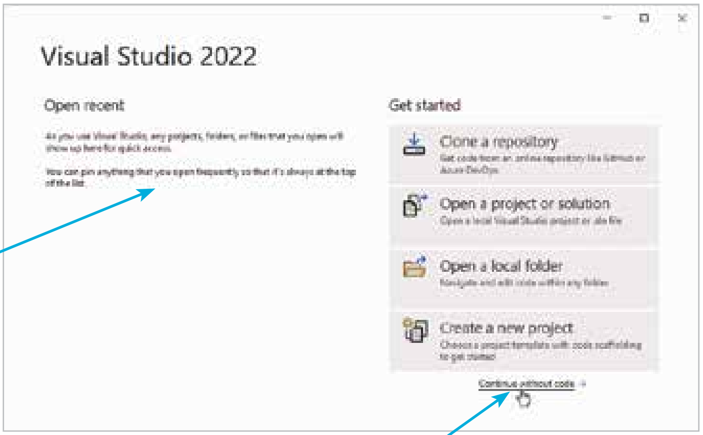


2. Sign in with your Microsoft account, or register an account then sign in, to continue

3. See a default **Start Page** appear where recent projects will be listed alongside several "Get started" options

Beware

The first time Visual Studio starts it takes a few minutes as it performs configuration routines.



Visual Studio 2022

Open recent

As you use Visual Studio, any projects, folders, or files that you open will show up here for quick access.

You can pin anything that you open frequently so that it's always at the top of the list.

Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

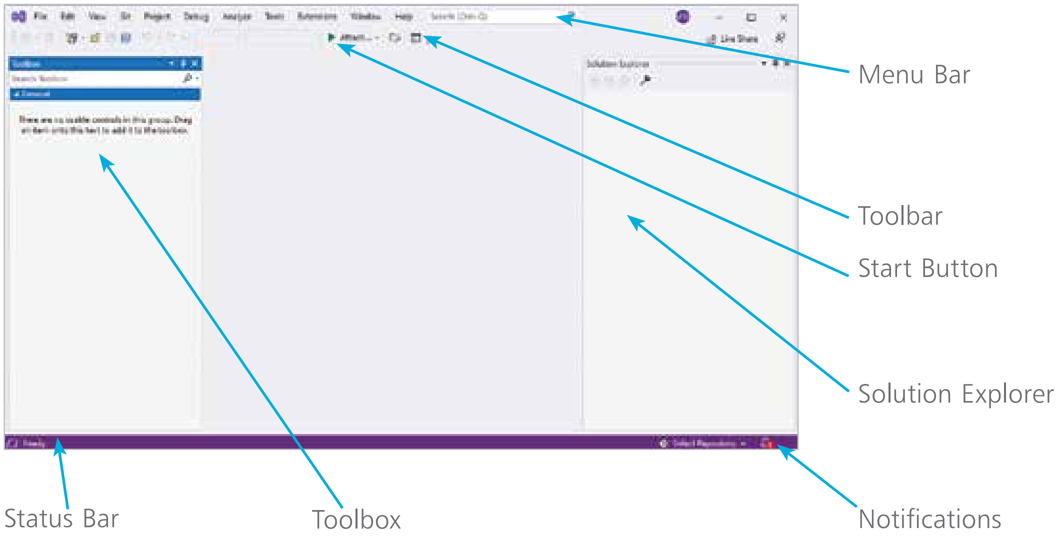
Continue without code →

Hot tip

In the future your recent projects will be listed here so you can easily reopen them.

4. For now, just click the **Continue without code** link to launch the Visual Studio application

The Visual Studio Integrated Development Environment (IDE) appears, from which you have instant access to everything needed to produce complete Windows applications – from here, you can create exciting visual interfaces, enter code, compile and execute applications, debug errors, and much more.



Visual Studio IDE components

The Visual Studio IDE initially provides these standard features:

- **Menu Bar** – Where you can select actions to perform on all your project files and to access Help. When a project is open, extra menus of Project and Build are shown in addition to the default menu selection of File, Edit, View, Git, Project, Debug, Analyze, Tools, Extensions, Window, and Help.
- **Toolbar** – Where you can perform the most popular menu actions with a single click on their associated shortcut icons.
- **Toolbox** – Where you can select visual elements to add to a project. Click View, Toolbox or a side bar button to see its contents. When a project is open, "controls" such as Button, Label, CheckBox, RadioButton, and TextBox are shown here.
- **Solution Explorer** – Where you can see at a glance all the files and resource components contained within an open project.
- **Status Bar** – Where you can read the state of the current activity being undertaken. When building an application, a "Build started" message is displayed here, changing to a "Build succeeded" or "Build failed" message upon completion.

Hot tip

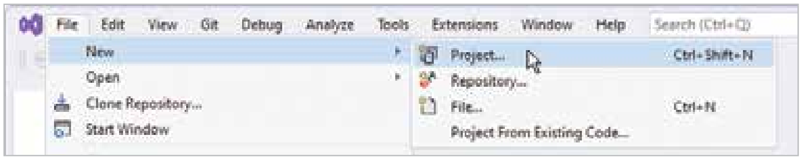
The IDE may have a **Light** color theme by default. To change the color theme, choose the **Tools**, **Options** menu then select **Environment**, **General**, **Color Theme** and select **Blue** or **Dark** theme, or select the **Use system setting** option.

Starting a Console project

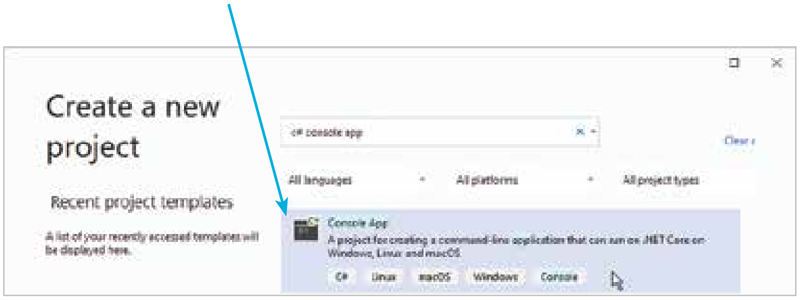
C#

GettingStarted

1. On the Menu Bar, click **File**, **New**, **Project...** or press **Ctrl** + **Shift** + **N**, to open the "Create a new project" dialog



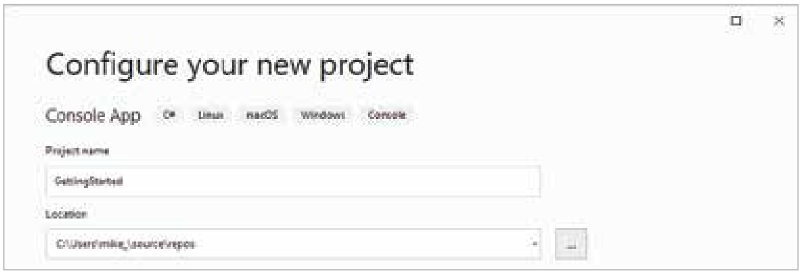
2. In the "Create a new project" dialog box, select the **C# Console App** item (for .NET Core on Windows, Linux and macOS), then click **Next**



A list of your recently accessed templates will be displayed here.

A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS

3. In the next dialog, enter a project name plus location and click the **Create** button, then select the .NET 6.0 (Long-term support) framework and click **Create** again



Project name: GettingStarted

Location: C:\Users\mike_\source\repos

Visual Studio now creates your new project and loads it into the IDE. A **Code Editor** window appears, containing default skeleton project code generated by Visual Studio.

4. Drag the **Code Editor** window tab to undock the **Code Editor** window from the Visual Studio IDE frame

Hot tip

The source code of all examples in this book is available for free download at **www.ineasysteps.com/downloads**

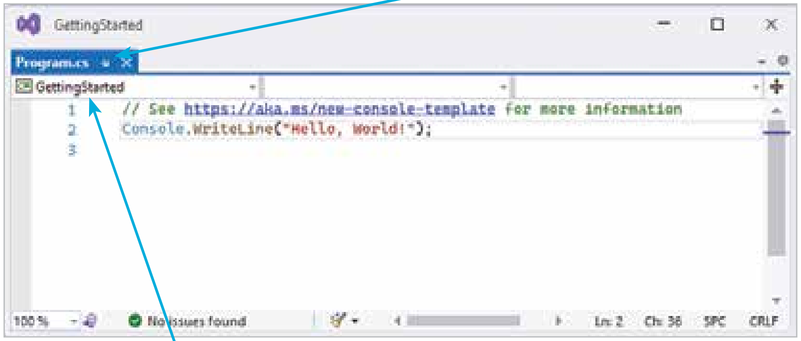
Hot tip

The default location for Visual Studio projects is a **C:\Users\*username*\source\repos** directory.

Hot tip

If the **Code Editor** window does not open automatically, click the **Program.cs** file icon in Solution Explorer to open the **Code Editor**.

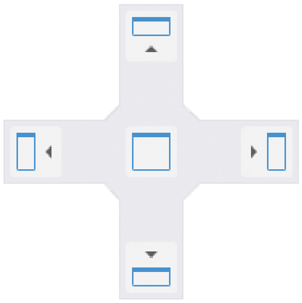
The undocked window title displays the project name, and the tab displays the file name of the code as "Program.cs".



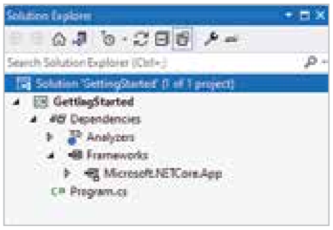
```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

The top-left drop-down box indicates the name of the project to which this file belongs – in this case, it's "GettingStarted".

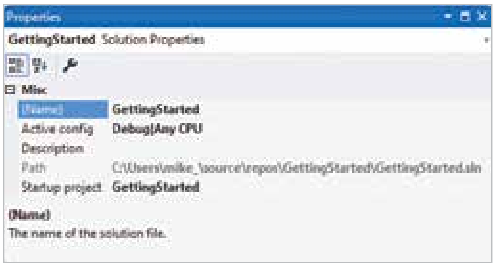
You can drag the title bar of any window to undock that window from the Visual Studio IDE frame. When dragging, you can drop a window on the "guide diamond" (shown below) to dock the window in your preferred position.



5. Select the **View, Solution Explorer** menu to open a **Solution Explorer** window, to discover all the items in your project – click the arrow buttons to expand or collapse categories



6. Select the **View, Properties Window** menu to open a **Properties** window, then select any item in the **Solution Explorer** window to see its properties then appear in the **Properties** window



(Name)	GettingStarted
Active config	Debug|Any CPU
Description	
Path	C:\Users\mike_\source\repos\GettingStarted\GettingStarted.sln
Startup project	GettingStarted

(Name)
The name of the solution file.

The **Code Editor** window is where you write C# code to create an application. The Visual Studio IDE has now gathered all the resources needed to build a default Console application.

Running a Console project

GettingStarted
(continued)

In order to run a program, Visual Studio will first build the app then execute the app. This can be done in one of two modes:

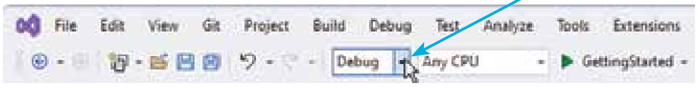
- **Debug Mode** – The program is compiled with symbolic debugging information included in the program files. This allows Visual Studio's built-in debugger to find bugs, but has optimization of Intermediate Language (IL) code disabled.
- **Release Mode** – The program is compiled without debugging information included in the program files, but has optimization of Intermediate Language (IL) code enabled.

During program development it is generally preferable to run your programs in Debug mode. There is an option to run a program in Debug mode without debugging, but it's seldom desirable as this executes the program without the possibility of stepping through the code to breakpoints.

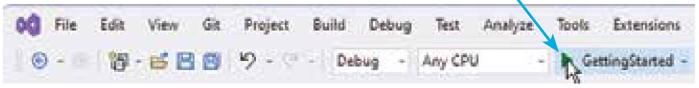
Hot tip

You will discover how to use breakpoints in Chapter 29 (Solving problems).

1. On the toolbar, set the solution configuration to **Debug**

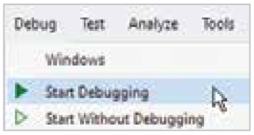

File Edit View Git Project Build Debug Test Analyze Tools Extensions
Debug
Any CPU
GettingStarted

2. Now, click the green arrow "**Start**" button to build and run the program with debugging enabled

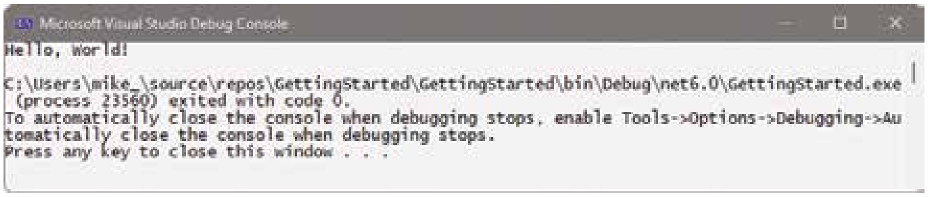

File Edit View Git Project Build Debug Test Analyze Tools Extensions
Debug
Any CPU
GettingStarted

Hot tip

Alternatively, select **Debug**, **Start Debugging** to build and run a program with debugging enabled.



3. See the app display a traditional greeting in the Console

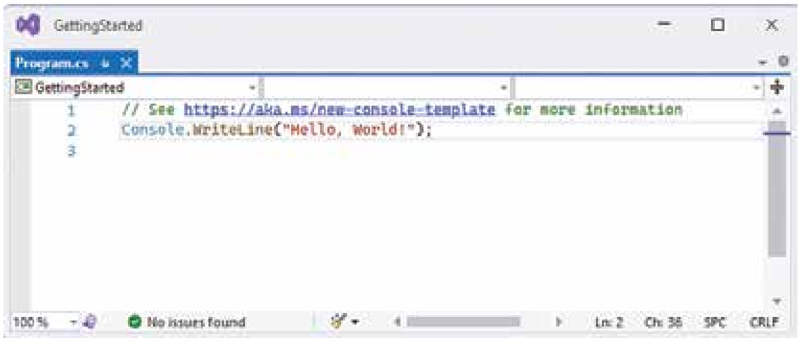


Microsoft Visual Studio Debug Console

```
Hello, World!

C:\Users\mike_\source\repos\GettingStarted\GettingStarted\bin\Debug\net6.0\GettingStarted.exe
 (process 23560) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Au
tomatically close the console when debugging stops.
Press any key to close this window . . .
```

4. Press any keyboard key to close the Console window, and return to the Visual Studio **Code Editor**



```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

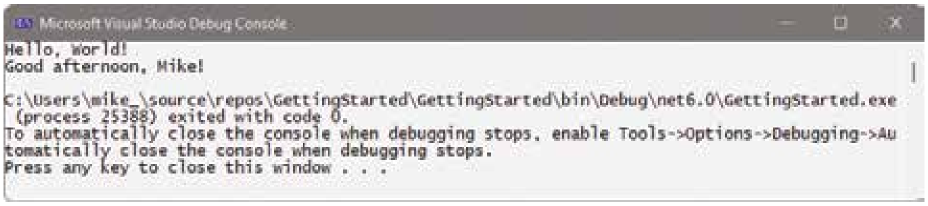
Code analysis

Examination of the code helps to understand what is happening:

- The first line is a comment. Anything on a line after // is ignored by the compiler. By default, comments are colored green in the Visual Studio **Code Editor**. In this case, the comment includes a hyperlink that will open a page in your web browser. The page explains how the .NET 6 framework uses new templates to simplify the creation of Console apps.
- **Console.WriteLine("Hello, World!") ;** This is a statement that calls upon the **WriteLine( )** method of the **Console** class to output the text string enclosed in quote marks within its parentheses. Notice that the statement is terminated by a **;** semicolon character. By default, class names are colored light blue, method names are colored brown, strings are colored red, and other code here is colored black.

5. Add another line to the code, as a statement to output a second text string containing your own name such as...
Console.WriteLine("Good afternoon, Mike!") ;

6. Run the modified code in Debug mode to see the result



```
Hello, World!
Good afternoon, Mike!

C:\Users\mike_\source\repos\GettingStarted\GettingStarted\bin\Debug\net6.0\GettingStarted.exe
 (process 25388) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Au
tomatically close the console when debugging stops.
Press any key to close this window . . .
```

Hot tip

New in .NET 6 are code templates, which the compiler uses to automatically generate namespace, class, and method elements when building a Console app.

Hot tip

To edit the default Console window colors and font, right-click its window Titlebar and choose **Properties**. For clarity, all other Console window screenshots in the C# section of this book feature Lucida Console 14-pixel **Font** in black **Screen Text** on a white **Screen Background**.