# murach's
# C++
# Programming

## (Chapter 1)

Thanks for downloading this chapter from *Murach's C++ Programming*. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its "how-to" headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our **website**. From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books on related topics.

Thanks for your interest in our books!

**MIKE MURACH & ASSOCIATES, INC.**

# What to expect in a Murach book

"I had a development environment up & running within half an hour and was dabbling with code 15 minutes after that!"

Andy Bonner, Solutions Architect, New Zealand

"Every two-page spread is designed to have all the detail about a topic on the left hand page and everything that a developer would need for reference material on the right hand page. This is a fantastic approach that I have not seen before in technical books."

Jim O'Neil, Denver Visual Studio .NET

"Another thing I like is the exercises at the end of each chapter. They're a great way to reinforce the main points of each chapter and force you to get your hands dirty."

Hien Luu, SD Forum/Java SIG

"Great book. I'm hard to please, and as a programmer and engineer, I was surprised by how informative this book was."

Posted at an online bookseller

"Your books are by far the best resource I have come across for aspiring developers like me. Most programming intructional resources teach us fundamentals and concepts of OOP (which is fine), but for people who wish to learn how to develop truly professional applications, Murach bridges the gap."

Blake Biessener, Graduate Student

"I kept your book at my side throughout the entire project. It was indispensable to me. The answers were right there at every turn. All the examples made sense to me, and they all worked!"

Alan Vogt, ETL Consultant, Information Builders, Inc.

"You folks make the hard stuff seem easy."

Thomas Finn, Sr. Software Developer, Illinois

# 1

# An introduction to C++ programming

This chapter starts by presenting some background information about C++. Although this information isn't essential to developing C++ programs, it does show how C++ works and how it compares to other languages. So it's important for you to at least skim this information.

After the background information, this chapter shows how to use an IDE (Integrated Development Environment) to develop a C++ program. If you're using a Windows computer, we recommend using the Visual Studio IDE that's developed by Microsoft. If you're using a Mac, we recommend using the Xcode IDE that's developed by Apple. As you'll see, there are many other IDEs that support C++ development, including the open-source Eclipse and NetBeans IDEs. These IDEs run on all modern operating systems, and you can use them if you prefer.

# An overview of programming and C++

In 1979, Bjarne Stroustrup began working on the language that would eventually become C++ (pronounced "see plus plus"). Today, C++ remains one of the most important programming languages in the world, with billions of lines of code in production. That includes many applications that are crucial to modern computing as we know it.

## Four general-purpose programming languages

C++ is a general-purpose programming language that was originally based on the C language. Figure 1-1 begins by presenting a chronological listing of four popular general-purpose programming languages including C++. Of these languages, C was developed first and had a commercial release in 1972. After that, C++ was developed as an extension of the C language that added the capability for object-oriented programming and had a commercial release in 1985. Since then, many other programming languages have used a syntax that's similar to C++, including Java (first commercial release in 1996) and C# (first commercial release in 2002).

At this point, you may wonder why you should learn C++ when there are newer languages like Java and C# available. To start, C++ is still one of the fastest and most efficient languages available. So, if you're developing a program that needs to be fast and use memory efficiently to conserve system resources, C++ is still a great choice. This is an advantage that C++ has over many other languages such as Java and C#.

Like many other modern languages, C++ is portable, which means that it works with many different operating systems and devices. Like many other object-oriented languages, C++ is ideal for developing large and complex applications. In fact, over the past 30 years, thousands of large and complex applications have been developed using C++ and billions of lines of code have been deployed. All of this adds up to a significant demand for C++ programmers in the job market, both for developing new projects and maintaining existing ones.

So, what types of applications are typically developed using C++? Due to its speed and efficiency, C++ is commonly used for the types of programming listed in this figure, including systems programming, desktop applications, mobile apps, video games, performance-critical applications, science applications, engineering applications, and embedded systems programming. In other words, C++ is commonly used for a wide range of applications, especially where speed and efficiency are critical.

### Four general-purpose programming languages

| Language | Year of first commercial release |
|----------|----------------------------------|
| C | 1972 |
| C++ | 1985 |
| Java | 1996 |
| C# | 2002 |

### Why it still makes sense to learn C++ today

- **Speed**. After all these years, C++ is still one of the fastest and most efficient languages available.
- **Portability**. C++ is designed to work with many different operating systems and devices.
- **Scale**. C++ is an object-oriented programming (OOP) language. This makes it ideal for developing large and complex applications that may have millions of lines of code.
- **Job security**. After all these years, there is still a robust demand for C++ developers from many of the largest companies in the world. Many new C++ projects are started every year, and billions of lines of existing C++ code will need to be maintained for many years to come.

### What C++ is used for

- **Systems programming** such as key parts of operating systems, device drivers, Internet routers, web servers, database servers, and even compilers and infrastructure for other languages such as Java and C#.
- **Desktop applications** such as web browsers, word processors, spreadsheets, and image editors.
- **Mobile apps** such as apps that need to run as efficiently as possible on Android devices.
- **Video games** that require extensive computation and graphics processing.
- **Performance-critical applications** such as financial, telecommunications, and military applications.
- **Science and engineering applications** that perform extensive numerical computation and graphics processing.
- **Embedded systems programming** such as applications for medical equipment, flight control software, and automobile software.

### Description

- C++ is a general-purpose programming language that was originally based on the C language.
- Many other programming languages have been influenced by C++ including Java and C#.

Figure 1-1    Four general-purpose programming languages

## A brief history of C++

Since 1998, C++ has been standardized by the *International Organization for Standardization* (*ISO*). Although these standards have a long formal name, they're typically referred to by a short informal name as shown in figure 1-2. So, the C++ standard released in 1998 is typically referred to as C++98. Similarly, the standard released in 2017 is typically referred to as C++17, and the standard that's due to be released in 2020 is typically referred to as C++20.

This book shows how to use C++17. However, C++ is *backwards compatible*, which means that C++17 works with older versions of C++ too. In addition, most of the skills described in this book have been a part of C++ since its earliest versions. As a result, earlier versions of C++ such as C++ 2011 work with most of the skills described in this book.

This figure also presents a brief history of C++. This history shows that Bjarne Stroustrup began developing a language called "C with Classes" in 1979. In 1983, this language was renamed to C++. In 1985, the first commercial implementation of C++ was released, though the language was not yet standardized by the ISO. In 1998, the ISO released the C++98 standard, followed by subsequent standards in 2003, 2011, 2014, and 2017. As you progress through this book, you'll learn more about what these standards mean for you as a programmer.

## C++ ISO standards

| Year | C++ Standard | Informal name |
|------|--------------|---------------|
| 1998 | ISO/IEC 14882:1998 | C++98 |
| 2003 | ISO/IEC 14882:2003 | C++03 |
| 2011 | ISO/IEC 14882:2011 | C++11 |
| 2014 | ISO/IEC 14882:2014 | C++14 |
| 2017 | ISO/IEC 14882:2017 | C++17 |

## C++ history

| Year | Event |
|------|-------|
| 1979 | Bjarne Stroustrup begins work on "C with Classes". |
| 1983 | "C with Classes" is renamed to C++. |
| 1985 | The first commercial implementation of C++ is released. |
| 1985 | The first edition of *The C++ Programming Language* is published. This book became the definitive reference for the language, as there was not yet an official standard. |
| 1990 | *The Annotated C++ Reference Manual* is published. This book became the basis for the 1998 standard. |
| 1998 | The C++98 standard is finalized. |
| 2003 | The C++03 standard is released, fixing some bugs with the previous standard. |
| 2011 | The C++11 standard is released, adding many new features. |
| 2014 | The C++14 standard is released, fixing some bugs with the previous standard. |
| 2017 | The C++17 standard is released, adding many new features. |

## Description

- Since 1998, C++ has been standardized by the *International Organization for Standardization* (*ISO*).
- C++ 2020 is the next planned standard.

Figure 1-2    A brief history of C++

# A quick look at C++ development

At this point, you're ready to see the user interface and source code for a C++ program. After that, you'll be ready to learn how C++ converts this source code so it can be run by a computer.

# The user interface for a console application

An *application*, or *app*, is computer software that performs a task or related set of tasks. However, an application can also be referred to as a *program*, even though one application may actually consist of many related programs. In practice, most people use these terms interchangeably.

A *desktop application* is a program that runs directly on your computer. There are two types of desktop applications that you can create with C++. The first type of desktop application is known as a *GUI application* because it uses a *graphical user interface* to interact with the user. (In conversation, GUI is pronounced *G-U-I* or *gooey*.) Examples of GUI applications include web browsers, word processors, and spreadsheets. So, if you've ever used a web browser, you are already familiar with GUI applications.
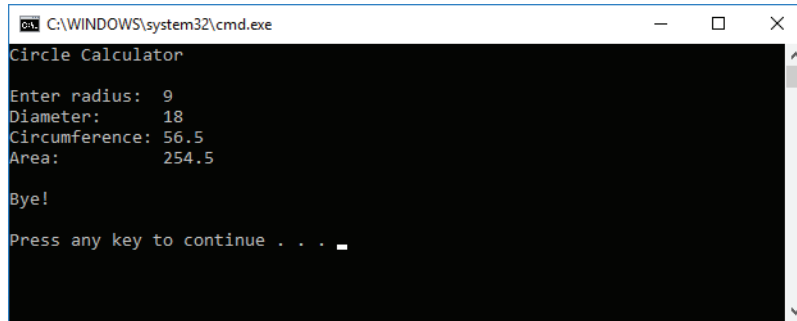
However, you might not be familiar with the second type of desktop application. This type of application is known as a *console application* because it uses the *console*, also known as a *command prompt* or *terminal*, to interact with the user by allowing the user to enter commands and data. In figure 1-3, for example, the console application prompts the user to enter the radius for a circle. Here, the user enters a value by typing a number and pressing the Enter key. Then, the application calculates the diameter, circumference, and area of the circle and displays the results on the console.

The appearance of the console may differ from one operating system to another, but the functionality should be the same. In this figure, for example, the first console is the Command Prompt window that's available from Windows, and the second console is the Terminal window that's available from macOS. However, both get input from the user in the same way and display the same result to the user.

Console applications are easier to code than GUI applications. As a result, they're often used by programmers when learning a language. That's why this book uses console applications to show how C++ works. Once you understand the basics of C++, you'll be ready to move on to writing GUI applications, if that's what you want to do.
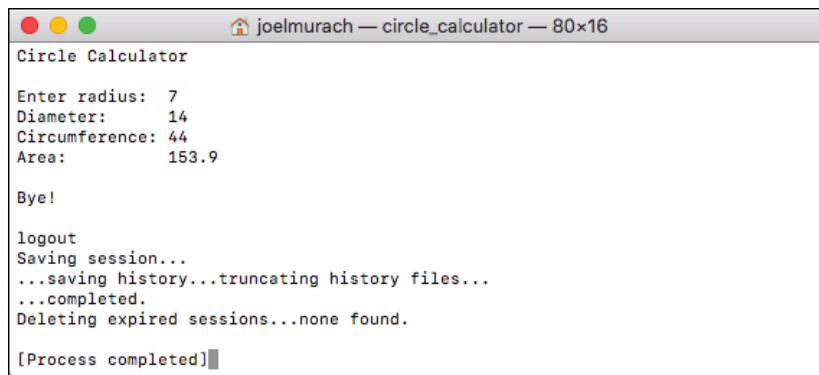
### A console application

#### Running in the console for Windows 10



#### Running in the console for macOS



### Description

- An *application*, or *app*, is computer software that performs a task or related set of tasks.

- An application can also be referred to as a *program*.

- A *console application* uses the console to interact with the user.

- The appearance of the console may differ from one operating system to another, but the functionality should be the same.

- A *GUI application* uses a *graphical user interface* to interact with the user. Examples of GUI applications include web browsers, word processors, and spreadsheets.

- Console applications are easier to code than GUI applications. As a result, they're often used by programmers when learning a language.

Figure 1-3     The user interface for a console application

# The source code for a console application

When you develop a C++ application, you start by entering and editing the *source code* for the program. To give you an idea of how the source code for a C++ program works, figure 1-4 presents the code for the Circle Calculator program shown in the previous figure.

If you have experience with other programming languages, you may be able to understand much of this code already. If not, don't worry! You'll learn how all of this code works in the next few chapters. For now, here's a brief explanation of this code.

The first two lines are #include directives that specify that this code uses two header files (iostream and cmath). These headers are available from the C++ standard library. Here, the iostream header provides the code for working with input and output (I/O) from the console. The cmath header, on the other hand, provides the code for accessing libraries written in the C language that provide mathematical functions. These functions include the pow() and round() functions used in this source code. Then, the third line makes it easier to work with the code in these headers, which are both available from the namespace named std, which is short for standard.

Most of the code for this program is stored in a function named main(). Before the function name, this code uses the int keyword to specify that this function returns an integer to the operating system. The code for this function is stored between the function's opening brace (`{`) and its closing brace (`}`). When a C++ program starts, it automatically executes the main() function and runs the code between its braces. Because of that, this function is known as the entry point for the application.

Within the main() function, the source code uses the cout, cin, and endl objects that are available from the iostream header to write output to the console (cout), read input from the console (cin), and specify the end of a line (endl). To help with that, this code uses the stream insertion operator (`<<`) to insert data into the output stream and the stream extraction operator (`>>`) to extract data from the input stream. In addition, this code uses the pow() and round() functions that are available from the cmath header to raise a number to the power of 2 and to round a number to 1 decimal place. Finally, the last statement in the method returns a value of 0 to tell the operating system that the program has ended normally.

In the next chapter, you'll learn how to write the code for this program. Then, you'll be able to write comparable programs of your own.

## The source code for a console application

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    // print name of program
    cout << "Circle Calculator" << endl << endl;

    // get radius from user
    double radius;
    cout << "Enter radius:  ";
    cin >> radius;

    // make calculations
    double pi = 3.14159;
    double diameter = 2 * radius;
    double circumference = diameter * pi;
    double area = pi * pow(radius, 2.0);

    // round to 1 decimal place
    circumference = round(circumference * 10) / 10;
    area = round(area * 10) / 10;

    // write output to console
    cout << "Diameter:      " << diameter << endl
        << "Circumference: " << circumference << endl
        << "Area:          " << area << endl << endl
        << "Bye!" << endl << endl;

    // return value that indicates normal program exit
    return 0;
}
```

### Description

- When you develop a C++ application, you write the *source code* for the program. Later, a program known as a *compiler* converts the source code into machine code that can be run by a computer.

- The two #include directives specify that this code uses two header files (iostream and cmath). These headers are available from the C++ standard library.

- The main() function contains the code that's run when the program starts.

- This code uses the cout, cin, and endl objects that are available from the iostream header to write output to the console (cout), read input from the console (cin), and specify the end of a line (endl).

- This code uses the pow() and round() functions that are available from the cmath header to raise a number to the power of 2 and to round a number to 1 decimal place.

- The return statement returns a value of 0 to the operating system to indicate that the program has ended normally.

Figure 1-4    The source code for a console application

# How source code compiles to an executable file

Once the source code has been written, you must *compile* the source code into machine language as shown in figure 1-5. *Machine language* consists of the 1s and 0s necessary to run the program on the computer's operating system.

Machine language is an example of a *low-level language* that's hard for humans to read and understand. In contrast, C++ is an example of a *high-level language* that's relatively easy for humans to read and understand but not usable by a computer until it's compiled into a low-level language.

If you take a closer look at the compilation process, it shows that you typically use an IDE (Integrated Development Environment) to enter and edit the source code for a C++ program. Files that contain C++ source code often have an extension of .cpp. In this figure, for example, the source code file is named main.cpp because it contains the main() function for the program.

In the first step, the *preprocessor* combines your source code with the source code from any header files that haven't already been compiled. This results in a temporary file of expanded source code.

In the second step, the *compiler* converts this expanded source code into machine language. The machine language at this intermediate step is called *object code*, and it's stored in a file called an *object file*. Object files often have an extension of .o or .obj.

In the third step, the *linker* links the machine language in the object file with the object code from other libraries that have already been compiled. These libraries contain prewritten C++ code that performs various operations and tasks. Together, they are known as the *runtime library*.

In the end, the compilation process yields a machine language file that can be executed on an operating system. On Windows, an executable file typically has an extension of .exe. By contrast, an executable file on macOS typically has an extension of .app.

Although this compilation process is complicated, it's done automatically with the push of a single key if you're using an IDE to develop your programs, as shown later in this chapter. For now, all you need to take away from this figure is that C++ source code is compiled into machine language that's run by the computer's operating system.

At this point, you should realize that you must compile your source code on each operating system that you want to run it on. Then, you end up with one executable file for each operating system. If you write your source code correctly, you should be able to use the same source code to create an executable file for each operating system. This is known as *cross compiling*. In other words, with C++, the source code is portable between operating systems, but the executable file isn't. One of the goals of this book is to show you how to write C++ source code that's portable.

## How C++ compiles source code into machine language
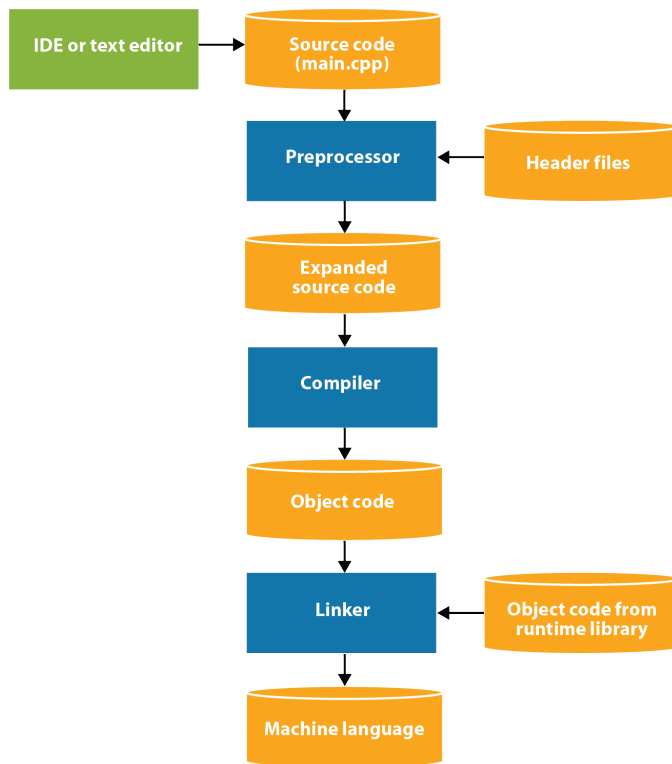


## Description

- *Machine language* consists of the 1s and 0s that a computer can run.
- A *low-level language* such as machine language is hard for humans to read and understand but can be run by a computer.
- A *high-level language* such as C++ is relatively easy for humans to read and understand but can't be run by a computer until it is translated into a low-level language.
- A *compiler* is a program that translates a high-level language to a low-level language.
- When you develop a C++ application, you typically use an *Integrated Development Environment* (*IDE*) to enter and edit the *source code* for the application. Files that contain C++ source code often have a .cpp extension.
- The C++ compilation process translates C++ source code into a machine language known as *object code* and then into machine language that can be run directly by an operating system.
- Libraries that contain prewritten and precompiled C++ code that perform various operations and tasks are called the *runtime libraries*.
- For a C++ program to run on different operating systems, you must compile the source code once for each operating system.

Figure 1-5    How C++ compiles code

# Four popular IDEs and compilers

Although it's possible to use a simple text editor with command-line tools, an *IDE* (*Integrated Development Environment*) provides features that can make developing C++ programs considerably easier. Figure 1-6 begins by listing four popular IDEs for C++ development. All four of these IDEs are either free or have a free edition. That makes them particularly attractive to students as well as programmers who are learning on their own.

All four of these IDEs provide the features listed in this figure. For example, these IDEs help you complete your code and notify you of potential errors. They automatically compile your code before you run it. And they include a debugger that can help you find and fix errors (bugs) in your code.

Of these IDEs, Visual Studio is the most popular IDE for developing C++ programs for Windows. It's developed by Microsoft and uses the Microsoft Visual C++ (MSVC) compiler by default. This compiler is the most popular compiler for developing C++ programs for Windows. Recently, Microsoft released a version of Visual Studio that runs on macOS, but most Mac users seem to prefer other IDEs, especially the Xcode IDE.

Xcode is the most popular IDE for developing C++ programs for macOS. It's developed by Apple and uses the Clang compiler. This open-source compiler is supported by many major companies including Apple, Microsoft, and Google. It was designed to act as a drop-in replacement for the GNU Compiler Collection (GCC), which is the standard compiler for most Unix-like systems, including Linux.

Eclipse and NetBeans both run on all major operating systems. Both of these excellent IDEs are free and open-source. Neither of them is as popular as Visual Studio and Xcode for developing for Windows and macOS, but both are good options for Linux development.

In addition, other C++ IDEs are available that aren't listed here. These options include commercial products such as the CLion IDE that's developed by JetBrains as well as the open-source Code::Blocks IDE.

When choosing an IDE, keep in mind that you may have to configure it to work with a compiler such as one of the compilers described in this figure. That's especially true if you're using an IDE such as Eclipse or NetBeans that doesn't automatically install and configure a compiler. For example, if you want to use NetBeans for Windows development, you need to install a compiler such as the MinGW compiler separately and configure NetBeans to use it. By contrast, the MSVC compiler is automatically installed and configured when you install Visual Studio, and the Clang compiler is automatically installed and configured when you install Xcode.

When choosing a compiler, keep in mind that different compilers provide different levels of support for the ISO standards. If you choose a compiler that provides good support for the ISO standards, you can use the features specified by those ISO standards. In addition, you can easily port your source code from one compiler to another.

### Four popular C++ IDEs

| IDE | Description |
| --- | --- |
| Visual Studio | A free IDE that runs on Windows and macOS. This IDE is popular for developing C++ applications for Windows. |
| Xcode | A free IDE that runs on macOS. This IDE is popular for developing C++ applications for macOS. |
| Eclipse | A free, open-source IDE that runs on most modern operating systems. This IDE is popular for developing C++ applications for Linux. |
| NetBeans | A free, open-source IDE that runs on most modern operating systems. This IDE is popular for developing C++ applications for Linux. |

### Features provided by most IDEs

- Code completion
- Error detection
- Automatic compilation
- Debugging

### Four popular C++ compilers

| Compiler Name | Description |
| --- | --- |
| Microsoft Visual C++ (MSVC) | The compiler that comes with the Microsoft Visual Studio IDE. This compiler is popular for Windows. |
| Clang | An open-source compiler that comes with Xcode. This compiler is popular for macOS. It was designed as a drop-in replacement for the GCC compiler. |
| GNU Compiler Collection (GCC) | A collection of open-source compilers that includes a compiler for C++. This C++ compiler is standard for most Unix-like systems, including Linux. |
| Minimalist GNU for Windows (MinGW) | A native Windows version of the GCC compiler. |

### Description

- To develop C++ applications, you typically use an Integrated Development Environment (IDE). Many other IDEs exist beyond those listed above, including some commercial products such as CLion and more open-source offerings such as Code::Blocks.
- Each IDE must use a compiler to compile the source code for a particular operating system. Many vendors provide C++ compilers, including the Free Software Foundation, Microsoft, Apple, Intel, Oracle, and IBM.
- All of the IDEs and compilers listed here are either free or have free editions.
- Different C++ compilers provide different levels of support for the ISO standards. When choosing a compiler, it's important to choose one that provides good support for the ISO standards.

---

Figure 1-6    Four popular IDEs and compilers for C++

# How to use Visual Studio for Windows development

Now that you have some background information about C++, you're ready to start learning how to use an IDE. In particular, you're ready to learn how to open and run any of the source code provided by this book. You can download this source code as described in the appendixes for this book.

If you're using a Windows computer, you can use Visual Studio as shown in the next four figures to develop programs that run on Windows. Of course, if you're using a Mac, you can skip or skim these figures and move on to the figures that show how to use Xcode to develop programs that run on macOS.

## How to open a project and work with source code

Figure 1-7 begins by showing Visual Studio with source code for a C++ project open in its *code editor*. In Visual Studio, a *solution* can contain one or more *projects* where each project is a folder within the solution. In this figure, the solution only contains one project, and both the project and the solution are named circle_calculator.

To open a project in Visual Studio, you can follow the procedure shown in figure 1-7 to open the solution that contains the project. For all of the Visual Studio applications presented in this book, the solutions are in this folder:

`C:\murach\cpp\vs\book_apps`

Once you open the solution for a project, you can view the source code for the project in the code editor by double-clicking on its file.
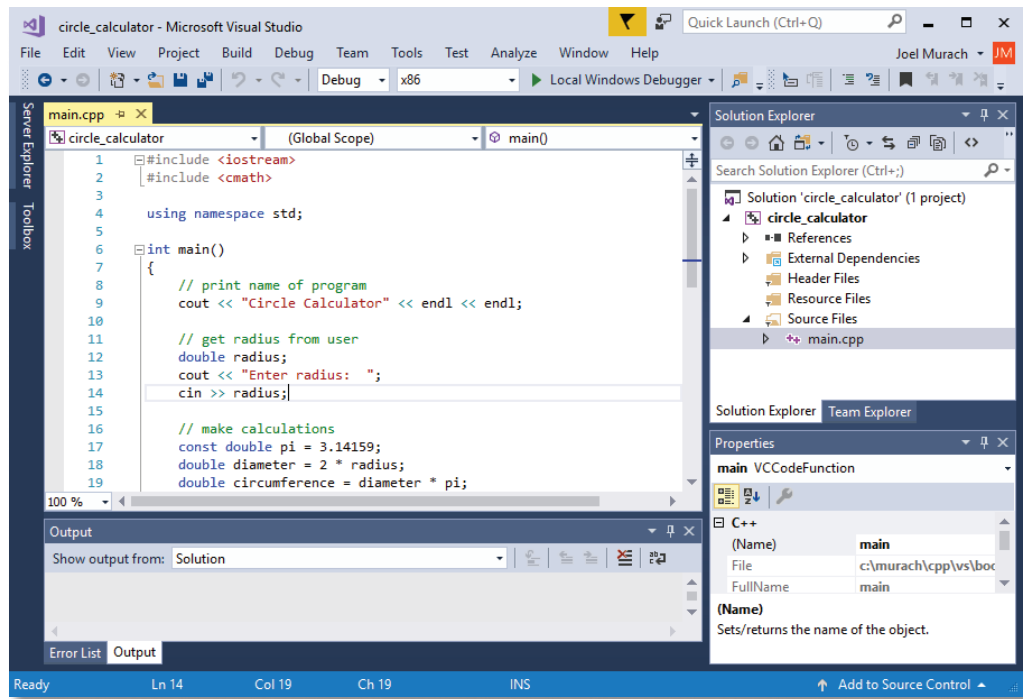
Before you open a source code file in the code editor, you may need to expand the folder for the project and the Source Files folder. To expand a folder, you can click the triangle to its left. In this figure, for example, both the circle_calculator folder and the Source Files folder have been expanded, and the source code file named main.cpp has been opened. This file contains the source code that contains the main() function for the Circle Calculator program described earlier in this chapter.

When you open a source code file in the code editor, it uses different colors for different language elements to make it easier for you to recognize the C++ syntax. In addition, Visual Studio provides standard File and Edit menus as well as keyboard shortcuts that let you save and edit the source code. For example, you can press Ctrl+S to save your source code, and you can use standard menu items and keyboard shortcuts to cut, copy, and paste code.

When you're done working with a solution, you can close it. To do that, you can select the Close item in the File menu. In addition, if you open another solution, Visual Studio automatically closes the current solution.

Before going on, you should realize that the appearance of the IDE as well as the menu structure depend on the Visual Studio development settings. For this book, we used the General settings, which should be the default if you don't have other versions of Visual Studio installed. To check your development

**Visual Studio with source code displayed in its code editor**



### How to open and close a solution

- To open a solution, press Ctrl+Shift+O or select the File→Open→Project/ Solution menu item. Then, use the Open Project dialog that's displayed to locate and select the solution file (.sln) and click the Open button.

- To close a solution, select the File→Close Solution menu item.

### How to open, edit, and save a source code file

- To open a source code file in the *code editor*, double-click on it in the Solution Explorer. Before you do that, you may need to expand the folder for the project by clicking on the triangle to its left, and you may need to expand the Source Files folder by clicking on the triangle to its left.

- To edit a source code file, you can use normal editing techniques to enter new code or edit existing code.

- To save a source code file, press Ctrl+S or click the Save button in the toolbar.

- To save all open files, press Ctrl+Shift+S or click the Save All button in the toolbar.

### Description

- A Visual Studio *solution* consists of one or more projects where each *project* consists of a folder that contains the folders and files for a program.

Figure 1-7    How to open an existing project and work with its source code

settings, select the Tools→Import and Export Settings menu item to display the Import and Export Settings Wizard. In the first dialog, select the "Reset all settings" option and click the Next button. Then, in the second dialog, select the "No" option and click the Next button. When the third dialog is displayed, the current settings will be highlighted. If those settings aren't General, select General and then click the Finish button. Otherwise, click the Cancel button.

# How to compile and run a project

Figure 1-8 shows how to compile and run a project. One easy way to run a project is to press Ctrl+F5. Then, if the project needs to be compiled, Visual Studio automatically compiles (builds) the project and runs its main() function. But first, it may prompt you to make sure you want to build the project. That can happen if you've made changes to the project or if you're running the project for the first time. Since this automatically compiles the project and creates the executable file, this feature is known as *automatic compilation*.

When you use Visual Studio to run a console program on Windows, it starts a Command Prompt window like the one shown in this figure. Then, the program uses this window to display output to the user and get input from the user. In this figure, for example, the program starts by displaying the name of the program on the console. Then, it prompts the user to enter a radius. In this figure, the user entered a radius of 7 by typing 7 and pressing Enter. Next, the program displays the diameter, circumference, and area of the circle. After that, the program displays a message of "Bye!" to indicate that the program has exited.

In this figure, Visual Studio displays the "Press any key to continue…" message after the program exits. This allows the user to view the console output before the console closes. Then, when the user presses any key, the console closes. However, Visual Studio only displays this message when you press Ctrl+F5 or select the Start Without Debugging item from the Debug menu. If you accidentally press F5 or select the Debug→Start Debugging menu item, the console closes when the program exits. For the Circle Calculator program, this would prevent you from seeing the output of the program. You'll learn more about how this works in chapter 8.

When you run a project like the ones for this book that were created by another programmer, you may get a message indicating that there are build errors. In most cases, that's because the version of the Windows SDK that's used by the program is different from the version that was installed when you installed Visual Studio. In that case, you'll see an error in the Error List window indicating that the Windows SDK was not found. (The Error List window is displayed in a tab at the bottom of the IDE as shown in this figure. If this tab isn't displayed, you can use the View→Error List menu item to display it.)

To fix this error, you need to change the Windows SDK version that's used by the application as described in this figure. If you don't know what version to use, you can typically select the most current version, which is likely to be the one that was installed with Visual Studio.

## Visual Studio after compiling and running a console application
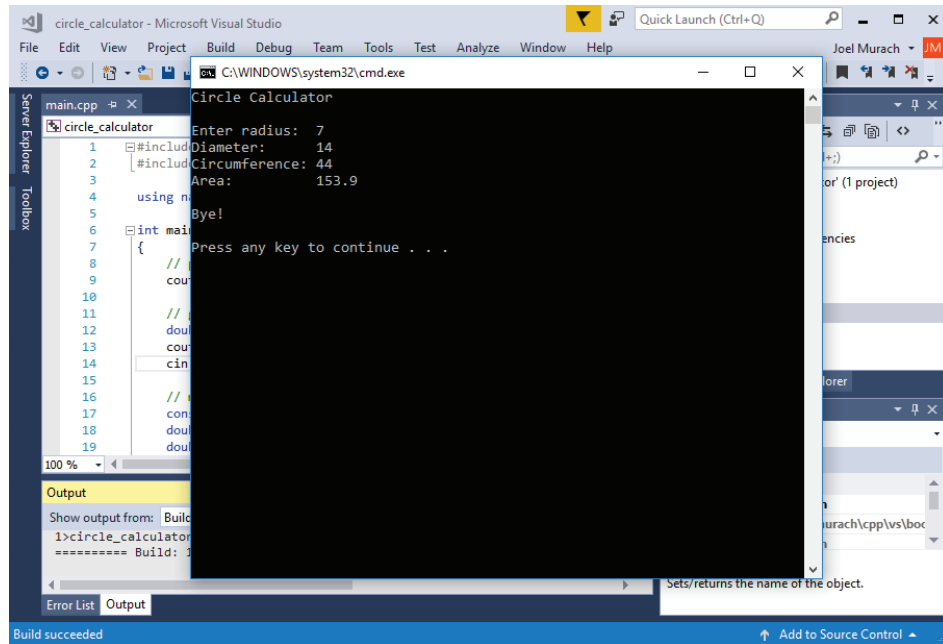


### Description

- To run a project, press Ctrl+F5 or select the Debug→Start Without Debugging menu item. If a dialog is displayed that asks if you want to build the project, click the Yes button to build the project before you run it.

- When you run a project, Visual Studio automatically compiles it. As a result, you don't need to manually compile a project before you run it. This is known as the *automatic compilation* feature.

- The Start Without Debugging menu item automatically displays a "Press any key to continue" message when the program exits. This keeps the console open until you press any key.

- If you accidentally press F5 or select the Debug→Start Debugging menu item, the console closes when the program exits. This might prevent you from seeing the output of the program. You'll learn more about how this works in chapter 8.

### How to correct a problem with the Windows SDK version

- If you compile a project and Visual Studio indicates that there are build errors, the problem may be that the Windows SDK version used by the application can't be found. This can happen if a newer version of the SDK is installed with Visual Studio.

- To change the SDK version, select the project in the Solution Explorer and then select the Project→Properties menu item to display the Property Pages dialog. Then, select the General item and set the Windows SDK Version property to the version of the SDK that was installed when you installed Visual Studio.

Figure 1-8     How to compile and run a project

# How to use code completion and error detection

Figure 1-9 shows how to use the *code completion* feature. In Visual Studio, the code completion feature is also known as *IntelliSense*. This feature helps you avoid typing mistakes, and it makes it easier to enter code. In this figure, for example, I started to enter the name of a variable. After I entered "rad", Visual Studio displayed a list that included "radius", which is the name of the variable that I wanted to enter. So, I pressed the Down Arrow key to highlight the "radius" option. At this point, I could press the Enter key to automatically enter the rest of the name.

In this case, code completion didn't save me much typing. But, it did make sure that I spelled the name of the variable correctly. This becomes more helpful as you begin working with longer names and more complex code.

In addition, code completion automatically enters opening and closing parentheses and braces whenever they're needed. This makes it easier to code functions like the main() function in this figure.

Code completion can also make it easy for you to enter values for string variables, which consist of a sequence of zero or more characters. If you type a double quotation mark to identify a string value, the code completion feature automatically adds the closing quotation mark and places the cursor between the two quotes. At this point, you can enter the text for the string.

If you experiment with the code completion feature, you'll quickly see when it helps you enter code more quickly and when it makes sense to enter the code yourself. In addition, you'll see that it helps you discover the functions that are available to you. This will make more sense as you learn more about C++ in the next few chapters.

As you enter C++ code, you may introduce syntax errors. A *syntax error* is any code that breaks the syntax rules of C++. If you enter code that contains a syntax error, the code won't compile. For example, if you enter an opening parenthesis without coding a closing parenthesis, your code has a syntax error and won't compile.

As you enter code into the code editor, Visual Studio displays syntax errors whenever it detects them. It does that by marking the error with a wavy red underline. If you position the mouse pointer over the code with the wavy red underline, Visual Studio displays a description of the error. This can provide valuable information that can help you fix the error. This feature is known as *error detection*, and it can help you find and fix errors before you even attempt to compile and run a program.

## Visual Studio's code editor with a code completion list
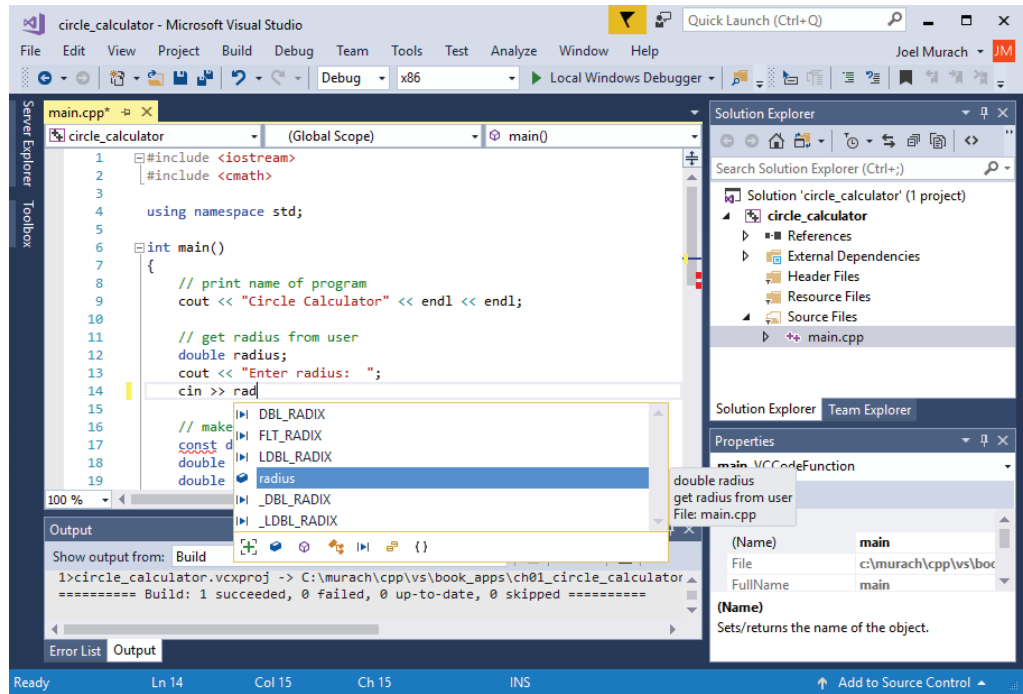


## How to use code completion

- When you enter code, Visual Studio often displays a list that attempts to help you complete the code entry. This is generally known as *code completion*, but Visual Studio refers to it as *IntelliSense*.

- If the code completion feature doesn't activate automatically, you can sometimes activate it manually by pressing Ctrl+Spacebar after entering one or more letters.

- To insert an item from a code completion list, you can use the Down and Up arrow keys on your keyboard to select the item and then press the Enter key.

- The code completion feature also helps you enter pairs of quotes, parentheses, braces, and so on. To do that, it enters the closing character immediately after you enter the opening character. For example, after you enter an opening quote, it enters the closing quote and places the cursor between the two quotes.

## How to use error detection

- A *syntax error* is any code that won't compile. If Visual Studio detects a syntax error as you enter code, it places a red wavy line under the statement. This is known as the *error detection* feature.

- To get more information about a syntax error, you can position the mouse pointer over the code with the red wavy underline.

Figure 1-9      How to use code completion and error detection

# How to create a new project

Figure 1-10 shows a procedure for using Visual Studio to create a new project that works with standard C++ code. To start, you display the New Project dialog and choose the type of project you want to create. In this dialog, you can select the Windows Console Application item as shown in this figure. In addition, you can enter a name and location for the project. In this figure, for example, the project name is "ch01_test" and its location is this folder:

`C:\murach\cpp\vs\book_apps`

If you install the source code for this book as described in appendix A, all of the Visual Studio programs presented in this book should be stored in this folder.

After completing the New Project dialog, you can click the OK button. This creates a new project that uses a precompiled header file named stdafx.h. This header file can improve performance of the compiler, especially if you're using many large libraries. However, it isn't standard, and it isn't necessary for the programs in this book, which don't use a lot of large libraries. As a result, we recommend setting the Precompiled Header property to "Not Using Precompiled Headers" as described in this figure. Then, you can use standard headers as shown throughout this book.

## Visual Studio's dialog for creating a new project
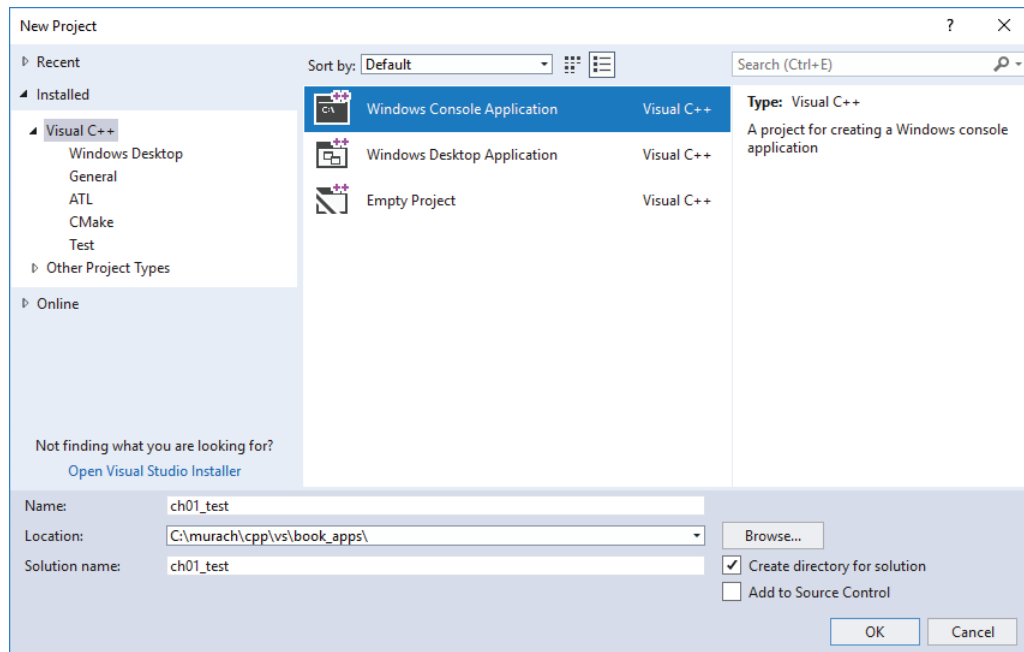


## Procedure

1. Click the New Project button in the toolbar or select the File→New→Project menu item to display the New Project dialog.

2. Select the Windows Console Application item, enter a name for the project, select a location for the project, and click the OK button to create the project.

3. Select the project in the Solution Explorer and then select the Project→Properties menu item to display the Property Pages dialog. Expand the C/C++ group, select the Precompiled Headers item, and set the Precompiled Header property to "Not Using Precompiled Headers".

4. Optionally, delete all unnecessary files related to the precompiled header (targetver.h, stdafx.h, and stdafx.cpp).

5. Optionally, rename the .cpp file that contains the main() function to main.cpp.

6. Open the file that contains the main() function and edit the code so it's like the code presented throughout this book.

Figure 1-10    How to create a new project (part 1of 2)

If you set the Precompiled Header property to "Not Using Precompiled Headers", you can use the Solution Explorer to delete all files related to the precompiled header that are no longer needed by the project. These include the targetver.h, stdafx.h, and stdafx.cpp files shown in this figure. Although deleting these files is optional, you can do that if you'd like to omit any unnecessary files. The easiest way to delete a file is to select it in the Solution Explorer and press the Delete key. Then, you can use the resulting dialog to permanently delete the file.

By default, Visual Studio creates a starting source code file that has the same name as the project. For the project shown in this figure, for example, Visual Studio created a source code file named ch01_test.cpp. However, I renamed this file to main.cpp to clearly show that this file contains the main() function for the program. The easiest way to rename a file is to select it in the Solution Explorer and press the F2 key. Then, you can enter a new name for the file. Alternately, you can right-click on the file and select the Rename item from the resulting menu.

When you create a new project, Visual Studio generates some starting code for you. In this figure, for example, Visual Studio generated some of the code for the main() function. However, it also generated some extra code related to the precompiled headers that was unnecessary. As a result, I deleted this code and added some standard C++ code that provides for writing output to the console. At this point, running this program would display a message of "Success!" on the console, which is a good starting point for a new program.

## A new project in Visual Studio with some starting source code
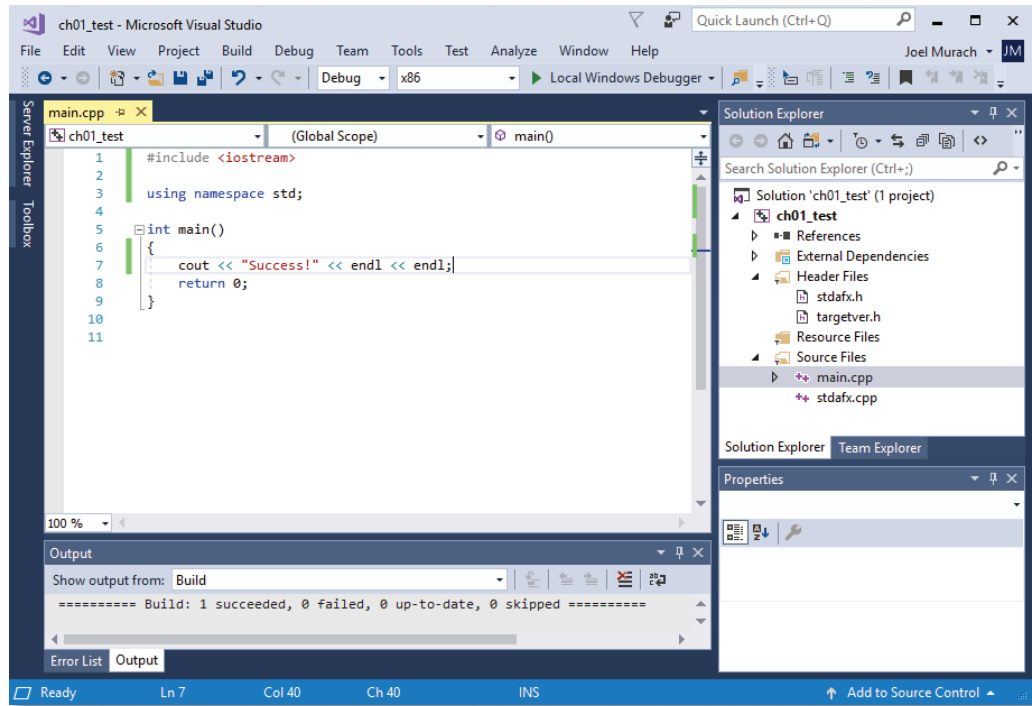


## How to rename and delete files

- To rename a file, you can select it in the Solution Explorer, press F2, and enter a new name for the file.
- To delete a file, you can select it in the Solution Explorer, press the Delete key, and use the resulting dialog to confirm the deletion.

## Description

- By default, Visual Studio creates a starting source code file that has the same name as the project. However, it's a common practice to use the name main.cpp for this file, as shown by the programs in this book.
- By default, Visual Studio creates a new project that uses a precompiled header file named stdafx.h. This header file can improve performance of the compiler, especially if you're using many large libraries. However, it isn't standard, and it isn't necessary for the programs in this book, which don't use a lot of large libraries.
- If you set the Precompiled Header property to "Not Using Precompiled Headers" as described in part 1 of this figure, you can use standard headers as shown throughout this book. In addition, you can optionally use the Solution Explorer to delete all files related to the precompiled header, including targetver.h, stdafx.h, and stdafx.cpp.

Figure 1-10    How to create a new project (part 2 of 2)

# How to use Xcode for macOS development

If you read the first six topics in this chapter and you're using a Mac, you're ready to start learning how to use the Xcode IDE to develop programs that run on macOS. In particular, you're ready to learn how to open and run any of the source code provided by this book. You can download this source code as described in appendix B of this book.

## How to open a project and work with source code

Figure 1-11 begins by showing Xcode with source code for a C++ project open in its *code editor*. In Xcode, a *project* is a folder that contains the folders and files for a program. In this figure, for example, the project is named circle_calculator, and it contains a folder named circle_calculator that contains the main.cpp file that stores the code for the program.

To open a project in Xcode, you can follow the procedure shown in this figure. For this book, the Xcode projects for the applications presented are in this folder:

```
/murach/cpp/xcode/book_apps
```

Once you open the project, you can view the source code for the project in the code editor by clicking on its file.

Before you open a source code file in the code editor, you may need to expand the folder for the project and the folder that contains the file for the source code. To expand a folder, you can click the triangle to its left. In this figure, for example, the circle_calculator folder has been expanded, and the source code file named main.cpp has been opened. This file contains the source code that contains the main() function for the Circle Calculator program described earlier in this chapter.

When you open a source code file in the code editor, it uses different colors for different language elements to make it easier for you to recognize the C++ syntax. In addition, Xcode provides standard File and Edit menus and keyboard shortcuts that let you save and edit the source code. For example, you can press Command+S to save your source code, and you can use standard menu items and keyboard shortcuts to cut, copy, and paste code.

When you're done working with a project, you can close it. To do that, you can select the Close item from the File menu. However, if you open another project, Xcode automatically opens another window for that project. Then, you will have multiple Xcode windows open.

### Xcode with source code displayed in its code editor
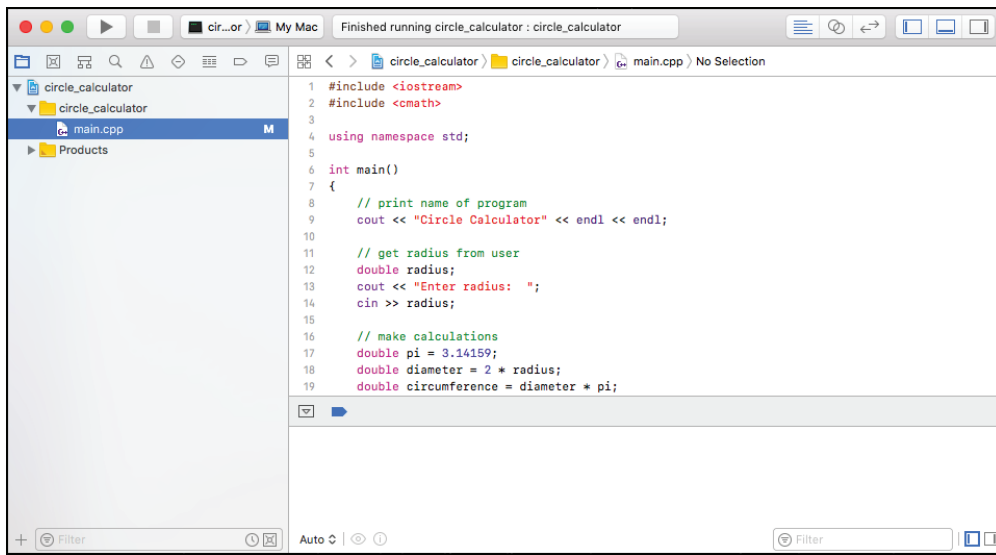


## How to open and close a project

- To open a project, press Command+O or select the File→Open menu item. Then, use the dialog that's displayed to locate and select the project file (.xcodeproj) and click the Open button.
- To close a project, select the File→Close Project menu item.

## How to open, edit, and save a source code file

- To open a source code file in the *code editor*, click on it in the Navigator. Before you do that, you may need to expand the folder for the project by clicking on the triangle to its left, and you may need to expand the folder that contains the source files by clicking on the triangle to its left.
- To edit a source code file, you can use normal editing techniques to enter new code or edit existing code.
- To save a source code file, press Command+S or select the File→Save menu item.

## Description

- An Xcode *project* consists of a folder that contains the folders and files for a program.

Figure 1-11    How to open an existing project and work with its source code

# How to compile and run a project

Figure 1-12 shows how to compile and run a project. One easy way to run a project is to press Command+R. Then, if the project needs to be compiled, Xcode automatically compiles (builds) the project and runs its main() function. This feature is known as *automatic compilation*.

When you use Xcode to run a console program, it uses a Console window like the one shown in this figure. Then, the program uses this window to display output to the user and get input from the user. In this figure, for example, the program starts by displaying the name of the program on the console. Then, it prompts the user to enter a radius. In this figure, the user entered a radius of 7 by typing 7 and pressing Enter. Next, the program displays the diameter, circumference, and area of the circle. After that, the program displays a message of "Bye!" to indicate that the program has ended, and Xcode displays some extra information that includes the exit code that the program returned when it ended.

## Xcode after compiling and running a console application
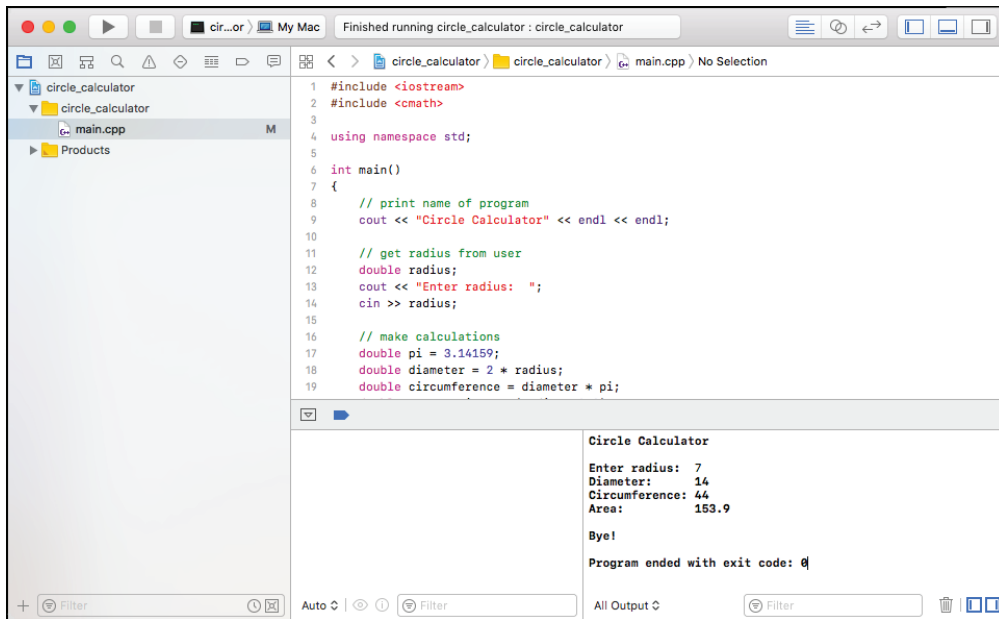


## Description

- To run a project, press Command+R or select the Project→Run menu item.
- When you run a project, Xcode automatically compiles it. As a result, you don't need to manually compile a project before you run it. This is known as the *automatic compilation* feature.
- When you run a project that writes data to the console, that data is displayed in the Console window in the lower right corner of the Xcode window.
- To display or hide the Console window, you can click the icon on the bottom right corner of the Xcode window.
- When you run a project that requests input from the console, the Console window pauses to accept the input. Then, you can click in the Console window, type the input, and press the Enter key.
- In addition to displaying output and accepting input, the Console window can display other information. For example, the Console window automatically displays a message when the program finishes running, and it can display errors that are encountered when an application is run.

Figure 1-12    How to compile and run a project

# How to use code completion and error detection

Figure 1-13 shows how to use the *code completion* feature. This feature helps you avoid typing mistakes, and it makes it easier to enter code. In this figure, for example, I started to enter the name of a variable. After I entered "r", Xcode displayed a list that included "radius", which is the name of the variable that I wanted to enter. So, I pressed the Down Arrow key to highlight the "radius" option. At this point, I could press the Enter key to automatically enter the rest of the name.

In this case, code completion didn't save me much typing. But, it did make sure that I spelled the name of the variable correctly. This becomes more helpful as you begin working with longer names and more complex code.

In addition, code completion sometimes enters opening and closing parentheses and braces whenever they're needed. This makes it easier to code functions like the main() function in this figure.

If you experiment with the code completion feature, you'll quickly see when it helps you enter code more quickly and when it makes sense to enter the code yourself. In addition, you'll see that it helps you discover the functions that are available to you. This will make more sense as you learn more about C++ in the next few chapters.

As you enter C++ code, you may introduce syntax errors. A *syntax error* is any code that breaks the syntax rules of C++. If you enter code that contains a syntax error, the code won't compile. For example, if you enter an opening parenthesis without coding a closing parenthesis, your code has a syntax error and won't compile.

As you enter code into the code editor, Xcode displays syntax errors whenever it detects them. It does that by displaying an error icon and message to the right of the line of code. In this figure, for example, you can see an error icon and message to the right of the line of code that I was entering. If you click on the error icon, Xcode displays more information about the error. This can provide valuable information that can help you fix the error. This feature is known as *error detection*, and it can help you find and fix errors before you even attempt to compile and run a program. In addition, for some errors, the extra information may include a suggested fix and a Fix button that you can click on to automatically implement this fix.

## Xcode's editor with a code completion list
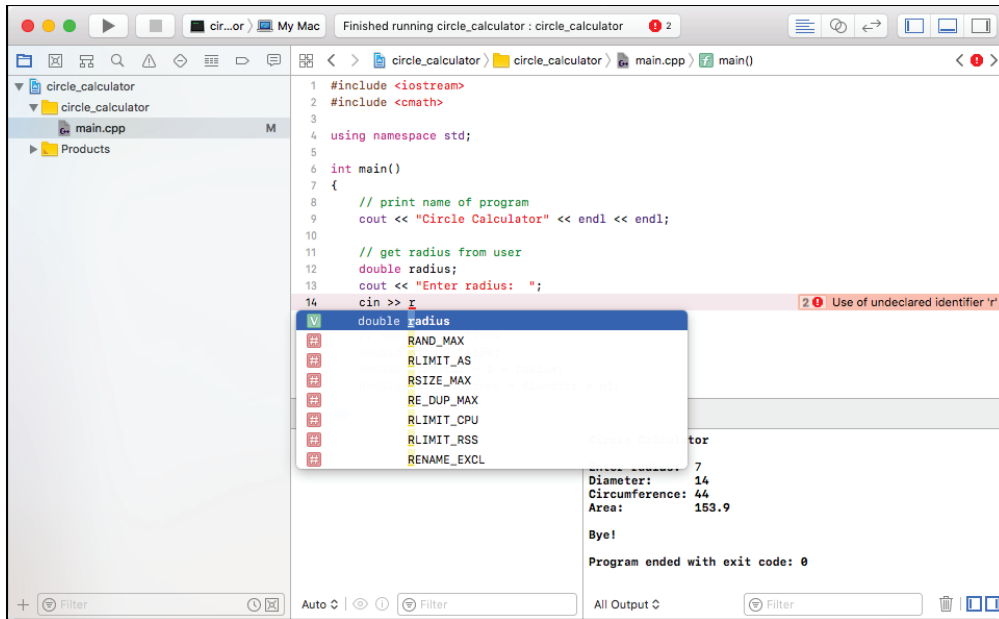


## How to use code completion

- When you enter code, Xcode often displays a list that attempts to help you complete the code entry. This is generally known as *code completion*.

- To insert an item from a code completion list, you can use the Down and Up arrow keys on your keyboard to select the item and then press the Enter key.

## How to use error detection

- A *syntax error* is any code that won't compile. If Xcode detects a syntax error as you enter code, it displays an error icon and message to the right of the statement. This is known as the *error detection* feature.

- To get more information about a syntax error, you can click the error icon. Sometimes the information that's displayed includes a Fix button that will implement a suggested fix.

Figure 1-13    How to use code completion and error detection

# How to create a new project

Figure 1-14 shows a procedure for using Xcode to create a new project for a console application. To start, you display the first dialog for creating a new project and choose the type of project you want to create. In this dialog, you can select the macOS and Command Line Tool options as shown in this figure.

In the second dialog, you must enter a name for the project in the Product Name field. In this figure, for example, the project name is "ch01_test". In addition, you often need to select C++ as the language for the project from the Language drop-down list. If necessary, you can also specify a name for your organization and a unique identifier for it. In this example, I used our website address (murach.com) backwards (com.murach), because that's a common way to uniquely identify an organization. Then, if you ever distribute the program, the organization identifier and the product name will be combined to create a unique bundle identifier as shown in this dialog.

In the third dialog (not shown here), you can specify the location for the project. If you install the source code for this book as described in the appendix, all of the Xcode programs presented in this book should be stored in this folder:

`/Documents/murach/cpp/xcode/book_apps`

After you specify the location, you can click the Create button to create the project. Then, Xcode creates a folder that corresponds with the project name, and it creates some additional files that it uses to configure the project.

## Xcode's dialogs for creating a new project
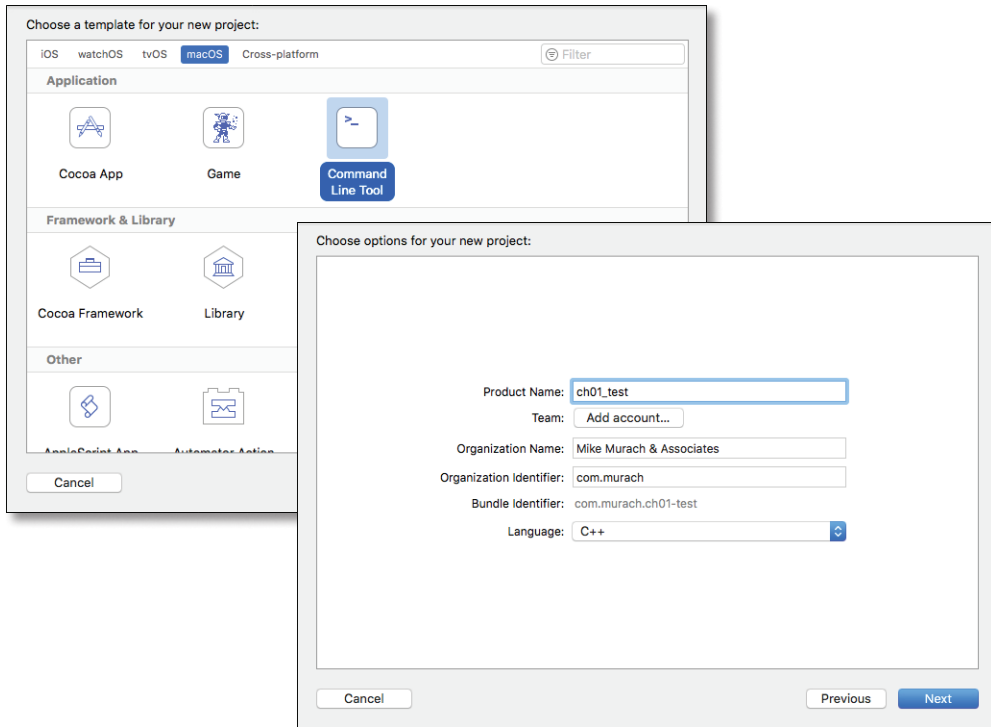


### Procedure

1.  Select the File→New→Project menu item to display the first dialog for creating a new project.

2.  In the first dialog, select the macOS operating system, select the Command Line Tool item, and click the Next button.

3.  In the second dialog, enter a product name, select the C++ language, and click the Next button. If necessary, enter an organization name and an identifier for the organization that uniquely identifies it.

4.  In the third dialog, navigate to the folder you want to use to store your project, and click the Create button to create the folder for the project.

Figure 1-14      How to create a new project (part 1 of 2)

When you create a new project, Xcode generates some starting code for you. For example, Xcode generated some of the code for the main() function shown in this figure. However, it also generated some extra code that was unnecessary. As a result, I edited this file so it only contains the starting code shown in this figure. At this point, running this program would display a message of "Success!" on the console, which is a good starting point for a new program.

**A new project in Xcode with some starting source code**
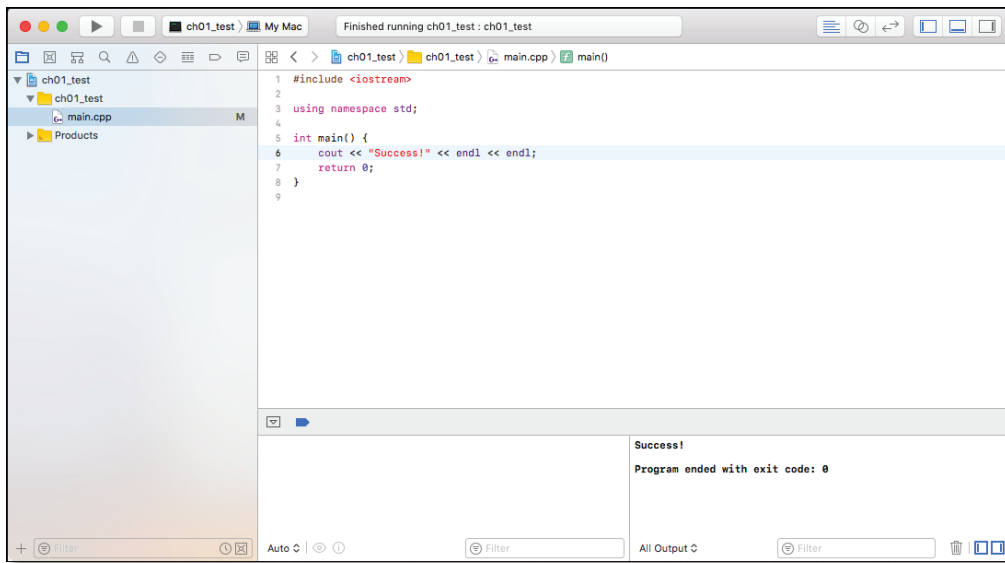


**Procedure (continued)**

5. In the Xcode window, expand the folder for the project if necessary. Then, click on the main.cpp file to show the main() function that was generated for the project.

6. Edit the generated code so it's like the code presented throughout this book.

**Description**

• By default, Xcode generates some starting code for you. You can edit this code so it contains C++ code like the code shown throughout this book.

Figure 1-14    How to create a new project (part 2 of 2)

# Perspective

In this chapter, you were introduced to C++ programming. In addition, you learned how to use an IDE such as Visual Studio or Xcode to create and run a C++ program on Windows or macOS. With that as background, you're ready to learn how to write your own C++ programs. But first, I recommend that you familiarize yourself with the IDE of your choice by doing the exercises at the end of this chapter.

# Terms

International Organization for
   Standardization (ISO)
application (app)
program
desktop application
GUI application
graphical user interface (GUI)
console application
console
command prompt
terminal
source code
machine language
low-level language
high-level language
preprocessor

compiler
object code
linker
runtime library
cross compiling
Integrated Development
   Environment (IDE)
code editor
Visual Studio solution
project
automatic compilation
code completion
IntelliSense
syntax error
error detection

# Summary

- Since 1998, C++ has been standardized by the *International Organization for Standardization* (*ISO*).

- An *application*, or *app*, is computer software that performs a task or a related set of tasks. Applications can also be referred to as *programs*, even though one application may actually consist of many related programs.

- A *desktop application* is an application that runs on your computer. A desktop application can use a *graphical user interface* (*GUI*) or the *console* to display output and get user input.

- Applications that use a command line window known as the console to interact with the user are known as *console applications*.

- A *high-level language* such as C++ is relatively easy for humans to read and understand.

- A *low-level language* such as *machine language* is hard for humans to read and understand but efficient for a computer to use.

- A *compiler* is a program that translates a high-level language to a low-level language.

- The C++ compiler translates C++ source code into machine language that consists of the 1s and 0s that can be run directly by an operating system.

- An *Integrated Development Environment* (*IDE*) can make working with C++ easier by providing code completion, error detection, automatic compilation, and a debugger.

- Visual Studio and Xcode are two of the most commonly used IDEs for C++ development.

- A Visual Studio *solution* can contain one or more *projects* where each project is a folder that contains the folders and files for a program.

- An Xcode *project* is a folder that contains the folders and files for a program.

- The main() function of a program is the starting point of the program.

## Before you do the exercises in this book

Before you do any of the exercises in this book, you need to have an IDE installed on your system. You also need to download the folders and files for this book from our website and install them on your system. For complete instructions, please refer to appendix A for Windows or appendix B for macOS.

## Exercise 1-1    Open and run two projects

This exercise guides you through the process of opening two existing projects, reviewing the code, and building and running the projects. It also has you introduce a syntax error to see what happens.

### Open the Circle Calculator project and review its code

1. Use your IDE to open the Circle Calculator project or solution in this folder:
   `book_apps\ch01_circle_calculator`

2. Open the main.cpp file.

3. Review the code to see that it contains two #include directives for the iostream and cmath header files, a using directive for the std namespace, and a main() function that accepts a radius from the console, calculates the diameter, radius, and circumference, and writes the output to the console. Note that the main() function ends by returning a value of 0.

### Run the Circle Calculator project

4. Run the project, building it if necessary. (If you're using Visual Studio on Windows and you get an error that indicates that the Windows SDK was not found, use the information in figure 1-8 to fix it and then run the project again.)

5. When prompted, enter a radius. Then, review the output that's written to the console.

### Introduce a syntax error

6.  Remove the semicolon (;) from the first statement in the main() function to see the errors that are detected.

7.  Run the project to see what happens.

8.  Add the semicolon back and then run the project again. When you're done, close the project.

### Open the Guess Number project and review its code

9.  Use your IDE to open the Guess Number project in this folder:
    `book_apps\ch04b_guess_number`

10. Open the main.cpp file and review its code. Note that its code is structured similarly to the code for the Circle Calculator program, but it uses some statements that you haven't learned about yet.

### Run the Guess Number project

11. Run the project, building it if necessary. (If you're using Visual Studio on Windows, you may need to change the Windows SDK version for the project.)

12. Enter input when prompted, and review the output that's written to the console. When you're done, close the project.

## Exercise 1-2    Create a new project

This exercise guides you through the process of creating a new project that displays a message on the console.

1.  Use your IDE to create a project named ch01_success_message in the ex_starts folder.

2.  If you're using Visual Studio, modify the project so it doesn't use precompiled headers. Then, delete all the files related to precompiled headers that are no longer necessary.

3.  If you're using Visual Studio, rename the ch01_success_message.cpp file to main.cpp.

4.  Open the main.cpp file and modify the generated code so it looks like this:

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

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

As you enter this code, use code completion whenever it's helpful.

5.  Run the application and view the output. It should display a message that says, "Success!" When you're done, close the project.

# How to become a C++ programmer

The easiest way is to let **Murach's C++ Programming** be your guide! So if you've enjoyed this chapter, I hope you'll get your own copy of the book today. You can use it to:

*Mike Murach, Publisher*

- Teach yourself how to code, test, and debug C++ programs by using modern features of the language that help you learn more quickly and easily than ever before

- Master professional skills like how to work with I/O streams, files, strings, vectors, functions, the STL, arrays, C strings, exceptions, structures, classes, pointers, templates, and generic programming

- Learn classic C++ techniques for working with legacy code and embedded systems

- Develop C++ programs that combine the best procedural practices with the best object-oriented practices

- Pick up new skills whenever you want or need to by focusing on material that's new to you

- Look up coding details or refresh your memory on forgotten details when you're in the middle of developing a C++ program

- Loan to your colleagues who will be asking you more and more questions about C++

To get your copy, you can order online at **www.murach.com** or call us at 1-800-221-5528 (toll-free in the U.S. and Canada). And remember, when you order directly from us, this book comes with my personal guarantee:

### 100% Guarantee

*You must be satisfied. Each book you buy directly from us must outperform any competing book or course you've ever tried, or send it back within 60 days for a full refund…no questions asked.*

Thanks for your interest in Murach books!