

INTRODUCTION TO PROGRAMMING WITH C++



Jin He
Fresno City College

Introduction to Programming with C++

This text is disseminated via the Open Education Resource (OER) LibreTexts Project (<https://LibreTexts.org>) and like the thousands of other texts available within this powerful platform, it is freely available for reading, printing, and "consuming."

The LibreTexts mission is to bring together students, faculty, and scholars in a collaborative effort to provide an accessible, and comprehensive platform that empowers our community to develop, curate, adapt, and adopt openly licensed resources and technologies; through these efforts we can reduce the financial burden born from traditional educational resource costs, ensuring education is more accessible for students and communities worldwide.

Most, but not all, pages in the library have licenses that may allow individuals to make changes, save, and print this book. Carefully consult the applicable license(s) before pursuing such effects. Instructors can adopt existing LibreTexts texts or Remix them to quickly build course-specific resources to meet the needs of their students. Unlike traditional textbooks, LibreTexts' web based origins allow powerful integration of advanced features and new technologies to support learning.



LibreTexts is the adaptable, user-friendly non-profit open education resource platform that educators trust for creating, customizing, and sharing accessible, interactive textbooks, adaptive homework, and ancillary materials. We collaborate with individuals and organizations to champion open education initiatives, support institutional publishing programs, drive curriculum development projects, and more.

The LibreTexts libraries are Powered by [NICE CXone Expert](#) and was supported by the Department of Education Open Textbook Pilot Project, the California Education Learning Lab, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation nor the US Department of Education.

Have questions or comments? For information about adoptions or adaptations contact info@LibreTexts.org or visit our main website at <https://LibreTexts.org>.

This text was compiled on 05/15/2026

TABLE OF CONTENTS

Licensing

1: Chapter 1 Introduction to Computer Programming

- 1.1: Introduction to Computer Science
- 1.2: What Is a Computer? and What Is a Computer Program?
- 1.3: Compilers and the Apps We Need for Learning C++ Programming
- 1.4: Download and Install App for C++ Programming (PC Users)
- 1.5: Download and Install App for C++ Programming (Mac Users)
- 1.6: Save a Copy of Source File
- 1.7: Understanding the Code of the First C++ Program
- 1.8: Case Study- Using setw() to Format the Output

2: Chapter 2 Elementary Programming

- 2.1: Constants and Variables
- 2.2: Console Input and Output
- 2.3: Assignment Statements
- 2.4: Case Study- Sum of Numbers
- 2.5: Numerical Operators
- 2.6: Coding a Mathematical Formula
- 2.7: C++ Examples
- 2.8: Case Study- Find the Sum of Numbers

3: Chapter 3 Selection Statements

- 3.1: Relational Operators and Relational Statements
- 3.2: Selection Statement- if-statement
- 3.3: Case Study- Find the Largest Number
- 3.4: Case Study- Sorting Numbers
- 3.5: Selection Statement- if-else Statement
- 3.6: Random Number and Logical Operators
- 3.7: Conditional Expression
- 3.8: Case Study- Practice Multiplication
- 3.9: Switch Statements and Menu Driven Programs

4: Chapter 4 Characters and Strings

- 4.1: Character Data Type and ASCII Code
- 4.2: String Data Type

5: Chapter 5 Looping Statements

- 5.1: The Structure and The Logic of a while Loop Statement
- 5.2: Case Study- Find The Largest Number
- 5.3: Case Study- Find The Sum
- 5.4: Sentinel Control Loop and do-while Loop Statement
- 5.5: for Loop Statement
- 5.6: Nested Loop Statements

- [5.7: Case Study- Display a Rectangle](#)

[6: Chapter 6 C++ Functions](#)

- [6.1: Introduction to C++ Functions](#)
- [6.2: More Examples and Further Discussion about Functions](#)
- [6.3: Global, Local, and Static Variable](#)
- [6.4: Passing by Value vs Passing by Reference](#)
- [6.5: Function Overloading](#)

[7: Chapter 7 Arrays](#)

- [7.1: Introduction to Arrays](#)
- [7.2: Working with the Data Stored in Arrays](#)
- [7.3: Case Study- Sorting Data Stored in an Array](#)
- [7.4: User-Defined functions with an Array as its Input](#)
- [7.5: C-String and String Variables](#)

[8: Introduction to Object Oriented Programming](#)

- [8.1: Introduction to Object Oriented Programming \(PC Users\)](#)
- [8.2: Introduction to Object Oriented Programming \(Mac Users\)](#)
- [8.3: Class Constructors](#)
- [8.4: Case Study- C++ Class- StudentRecord](#)
- [8.5: Case Study- C++ Class- Bag](#)

[9: Pointers](#)

- [9.1: What is a Pointer?](#)
- [9.2: Pointer and Dynamic Memory Allocation](#)
- [9.3: Case Study- Array Copy](#)

[10: Recursive Functions](#)

- [10.1: Understand Recursive Functions](#)
- [10.2: Recursive Functions v.s. Loop Statements](#)
- [10.3: Using Array as Input of a Recursive Function](#)

[Index](#)

[Glossary](#)

[Detailed Licensing](#)

Licensing

A detailed breakdown of this resource's licensing can be found in [Back Matter/Detailed Licensing](#).

CHAPTER OVERVIEW

1: Chapter 1 Introduction to Computer Programming

- 1.1: Introduction to Computer Science
- 1.2: What Is a Computer? and What Is a Computer Program?
- 1.3: Compilers and the Apps We Need for Learning C++ Programming
- 1.4: Download and Install App for C++ Programming (PC Users)
- 1.5: Download and Install App for C++ Programming (Mac Users)
- 1.6: Save a Copy of Source File
- 1.7: Understanding the Code of the First C++ Program
- 1.8: Case Study- Using `setw()` to Format the Output

This page titled [1: Chapter 1 Introduction to Computer Programming](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

1.1: Introduction to Computer Science

Computer science is the study of computers and computational systems, focusing on the theory, design, and application of software and algorithms to solve problems. It involves a broad range of topics, from the theoretical principles of computation to the practical development of software, hardware, and artificial intelligence. The field applies principles from mathematics, engineering, and logic to design and build systems that process information and automate tasks.

The field of computer science (CS) is the study of computing, which includes all phenomena related to computers, such as the Internet. With foundations in engineering and mathematics, computer science focuses on studying algorithms. An algorithm is a sequence of precise instructions that enables computing. This includes components computers use to process information. By studying and applying algorithms, computer science creates applications and solutions that impact all areas of society. For example, computer science developed the programs that enable online shopping, texting with friends, streaming music, and other technological processes.

This page titled [1.1: Introduction to Computer Science](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

1.2: What Is a Computer? and What Is a Computer Program?

A computer is an electronic device that stores and processes data.

A computer system consists of both **hardware** and information stored on those hardware. Information stored on the computer hardware is often called **software**.

The hardware components of a computer system are the electronic and mechanical parts. The software components of a computer system are the data and the computer programs.

The major hardware components of a computer system are:

- Processor
- Main memory
- Secondary memory
- Input devices
- Output devices

For typical desktop computers, the processor, main memory, secondary memory, power supply, and supporting hardware are housed in a metal case. Many of the components are connected to the main circuit board of the computer, called the *motherboard*. The *power supply* supplies power for most of the components. Various input devices (such as the keyboard) and output devices (such as the monitor) are attached through connectors at the rear of the case.

You may or may not be familiar with those computer hardware components at this point, no worry, this book will not focus on how to build a computer, instead, we will focus on how to write computer programs. A computer is a machine. In order to operate this machine, we will need to write a set of instructions. Writing those computer instructions is called computer programming. A computer program is referred to a set of instructions that tell a computer what to do. For example, when we turn on a computer, the monitor of the computer is lighted up, and a screen display shows up:



All of these actions are controlled by a set of instructions, known as a computer program. Computers do not understand human languages, thus, those instructions must be written in languages computer can understand. C++ is one of those languages we will discuss in this book.

As a language, it has its language specified syntax rules. When we are writing a computer program, we must follow the syntax rules, otherwise, we can't communicate with the computer properly.

This page titled [1.2: What Is a Computer? and What Is a Computer Program?](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

1.3: Compilers and the Apps We Need for Learning C++ Programming

A C++ program is a set of computer statements, commands, and instructions that we will use to communicate with a computer. We can use a C++ program to do various things with a computer, such as conducting mathematical calculation, storing and managing large sets of experimental data.

We will write those statements, commands, and instructions in a language called C++. We need to save them into a file with **.cpp** as the file extension. Those statements, commands, and instructions tell the computer what we would like it to do. Following is an example of a C++ program that asks a computer to display a message "Good Morning!" on the screen of its monitor:

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

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

However, even the program were written in C++ language, the computer can't understand them directly, we will need to use a computer app, called "Compiler or "interpreter" to translate those statements, commands, and instructions from C++ language to a language that the computer can understand. Once the computer receives those instructions, it will perform the task. Then, we will use the reactions of a computer to tell if those statements, commands, and instructions were written

correctly. We normally observe the reactions of the computer through a window that opens on the screen of the computer monitor which is referred as **output window of the program**. Most of times, we also need a keyboard, a mouse, and a monitor to interact with a C++ program, such interactions are called programs' **input and output**.

To write a computer program, we will need to

1. Use a text editor to write the code for the program, which is set of statements, commands, and instruction and save it in a file with the extension .cpp which we called it as a **source file**.
2. Load the source file to a **compiler** (a computer app). The compiler will translate the program from C++ language to a language the computer can understand and also link all the resources the program will need. The compiler also does the error checking. This process is often referred as **building** the program.
3. Execute (run) the program and receive feedback from a computer.

IDE (Integrated Development Environment) is a computer application program (app) which allow us to write, build, and execute a C++ program. There are many IDE products on the market, the one we will be using in this book is *Microsoft Visual Studio Community 2022* . This app can be downloaded from Microsoft official website and it is free of charge.

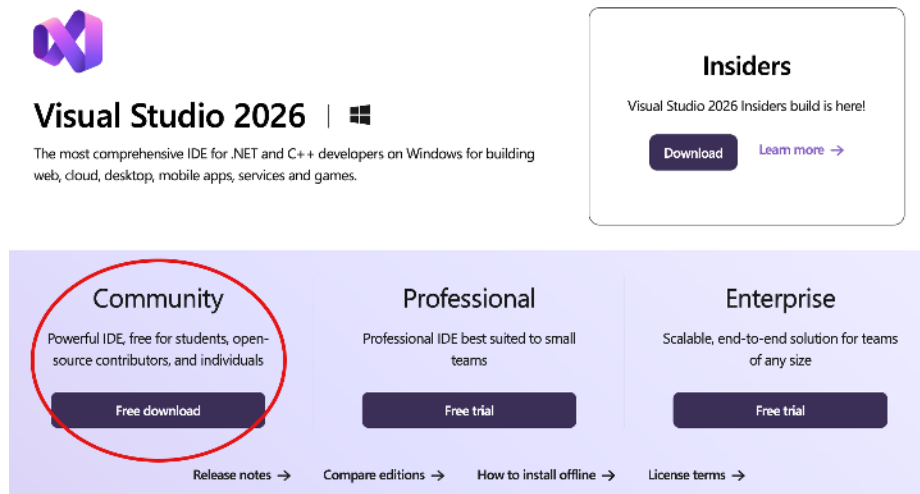
To able to write and run a C++ program, we will need to download and install this app in our PC computer first (for mac users, please seek help from your instructor).

This page titled [1.3: Compilers and the Apps We Need for Learning C++ Programming](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

1.4: Download and Install App for C++ Programming (PC Users)

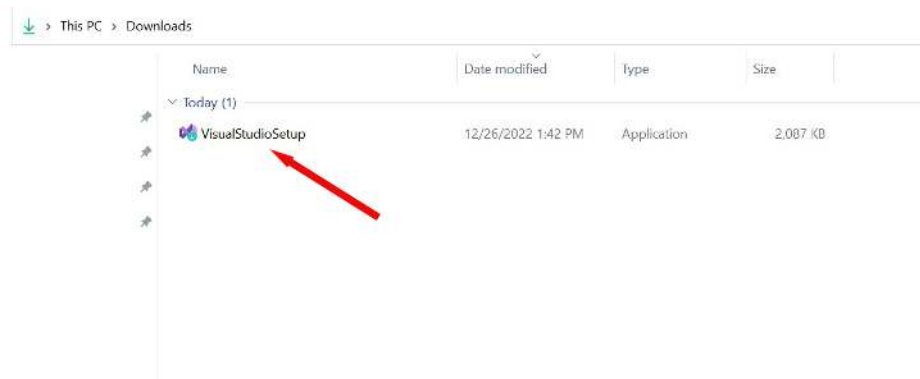
Step 1: Go to <https://visualstudio.microsoft.com/downloads>

You will see following screen:

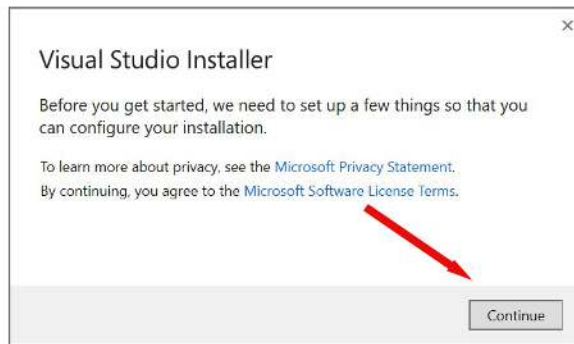


Select "Community" version, and click on "Free download".

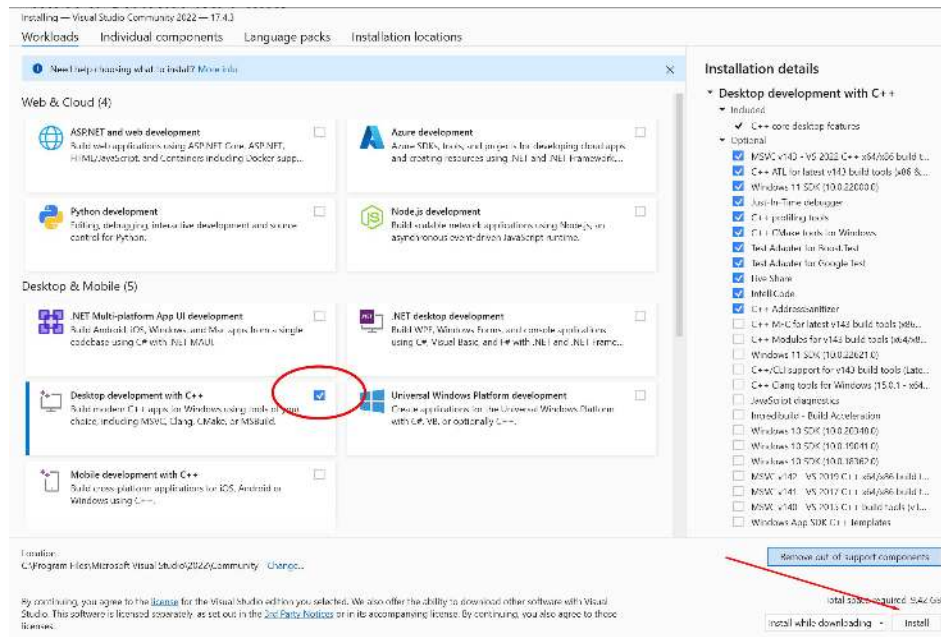
Step 2: Go to the Download Folder in your computer and run the following installer:



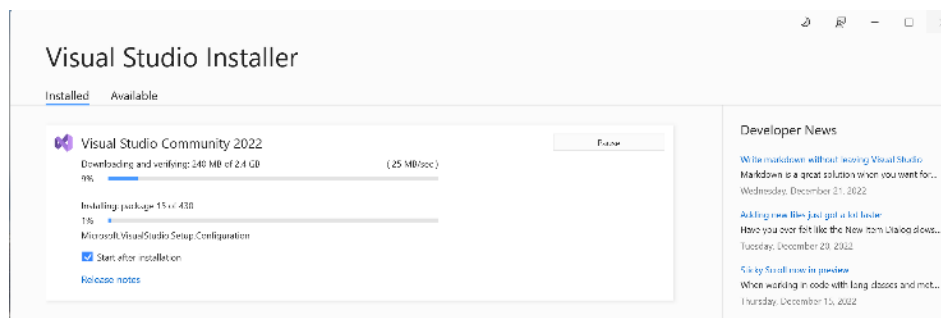
Double click on "VisualStudioSetup" in the Download Folder, in the following box, click on "Continue":



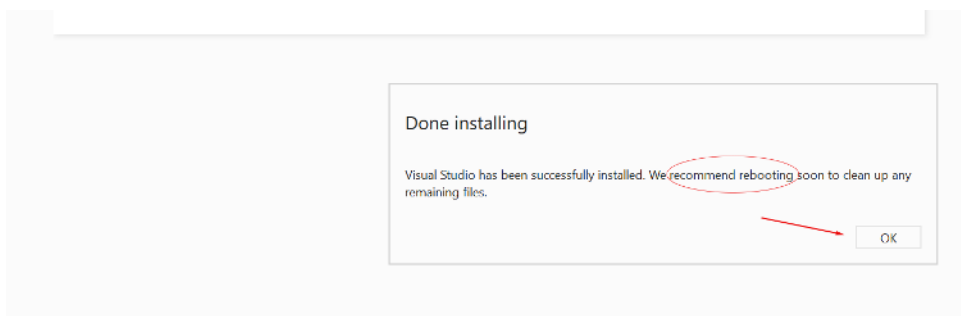
In following pop up window, check "Desktop development with C++", and leave the rest of boxes unchecked. Then click on "Install"



The installation process begins:



In the following window, click on OK and then restart your computer:



Congratulations! You have successfully installed Microsoft Visual Studio on your computer and ready to use it for coding a C++ program.

Get Started with Microsoft Visual Studio

Now, let's use the IDE we just installed to write, build, and run a simple C++ program.

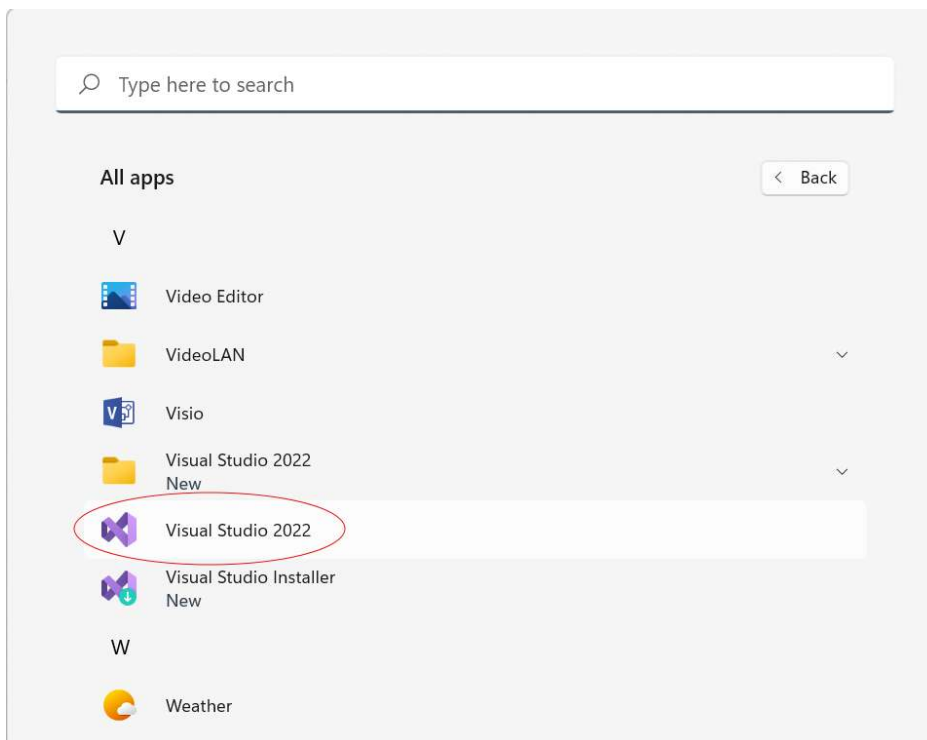
If you prefer watching video presentations for this topic, please study following video lecture:

Video Presentation 1.1 Getting Started with Microsoft Visual Studio

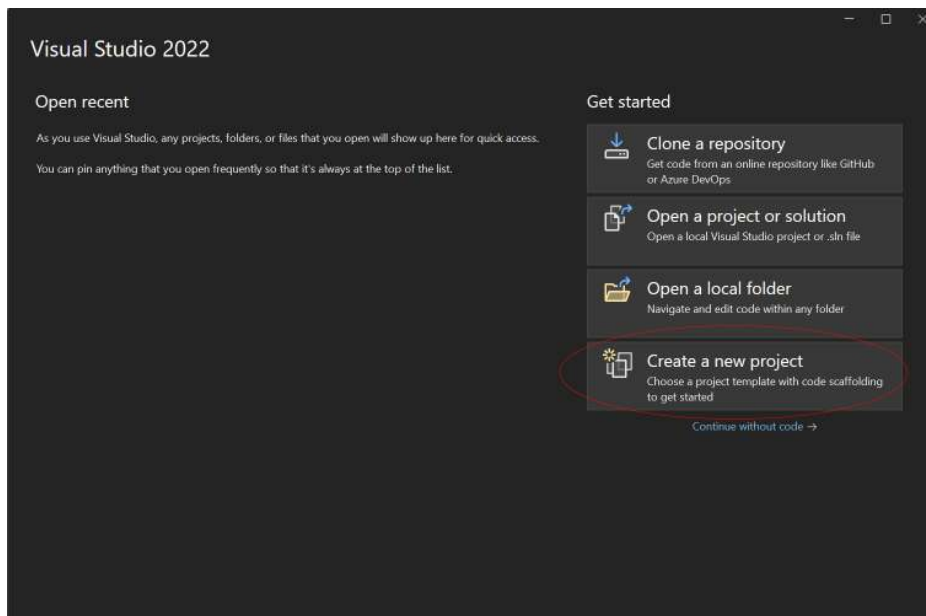
This video lecture is based on Community 2019, which is very similar to Community 2022

If you prefer the text and graphical presentation, please proceed.

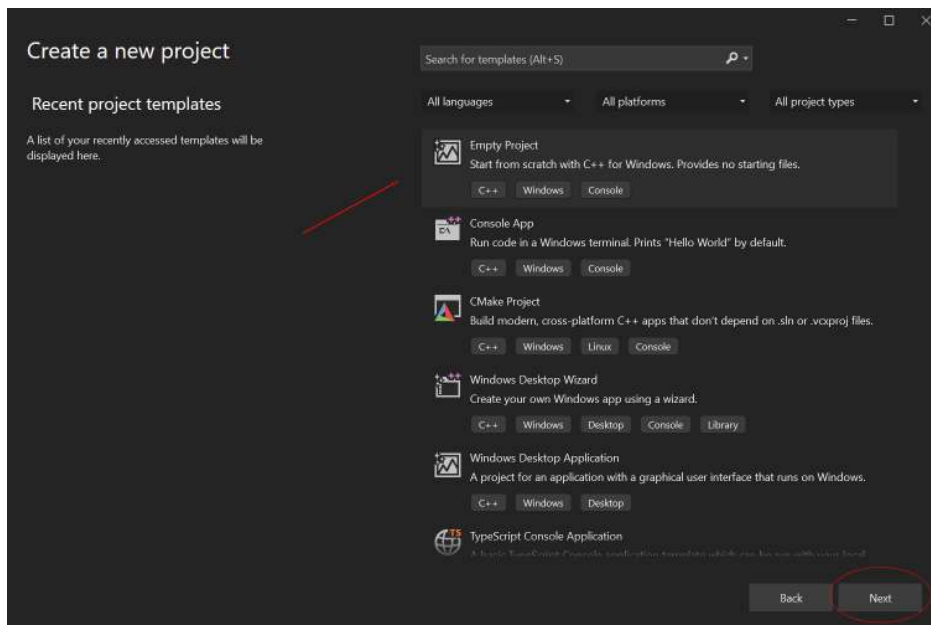
Step 1: Go to Start on your computer and launching the app (Now, it is Visual Studio 2026)



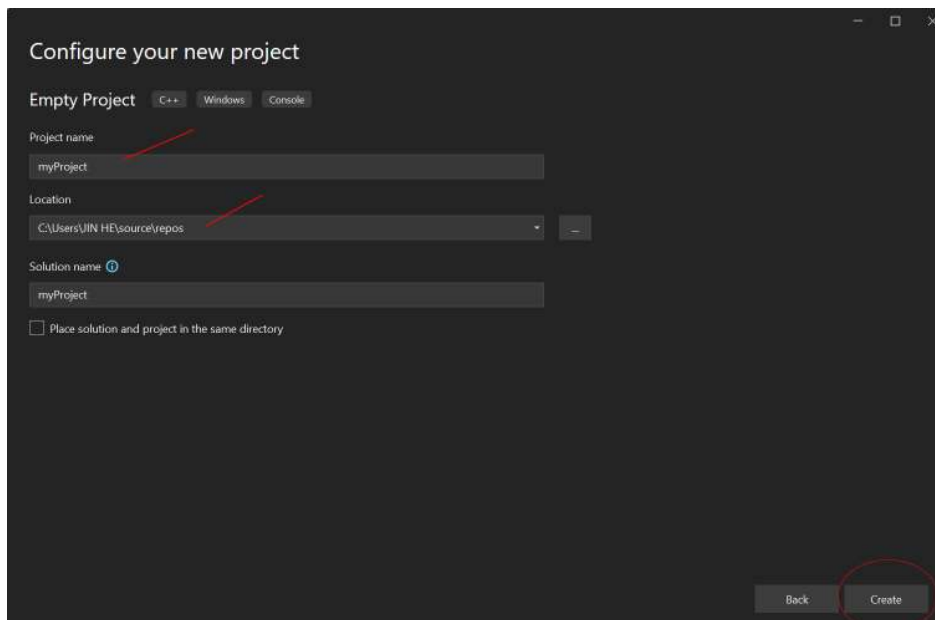
After the app is launched, we will see following screen (now it is Visual Studio 2026):



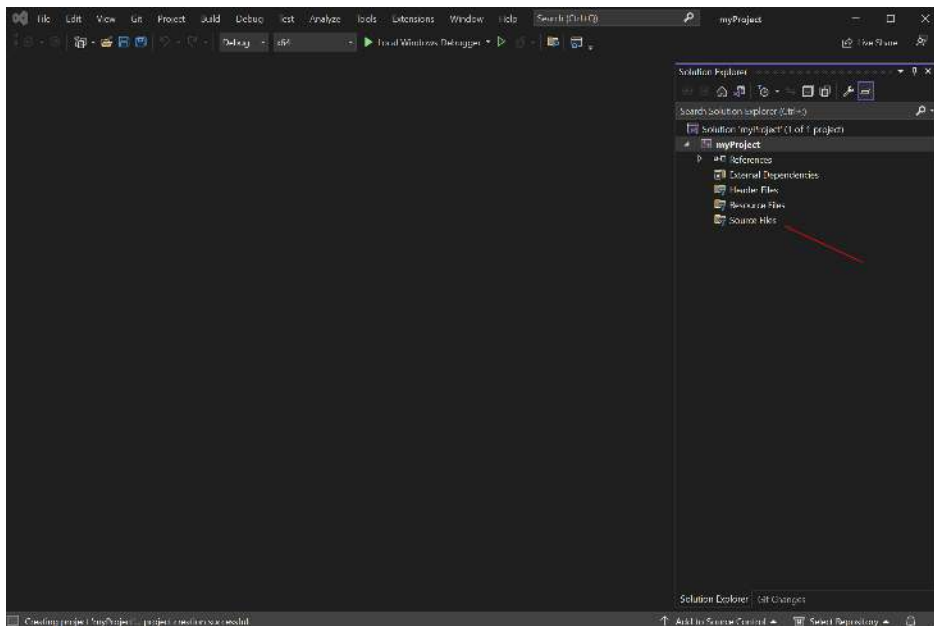
Step 2: Click on "Create a new project, then following screen appears:



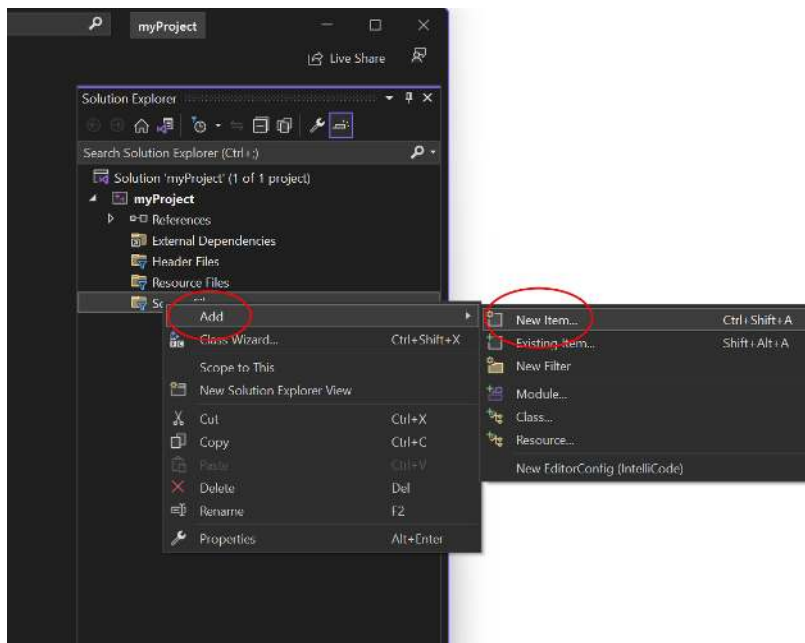
Step 3: Select "Empty Project" and then click on "Next". Now, following screen shows up:



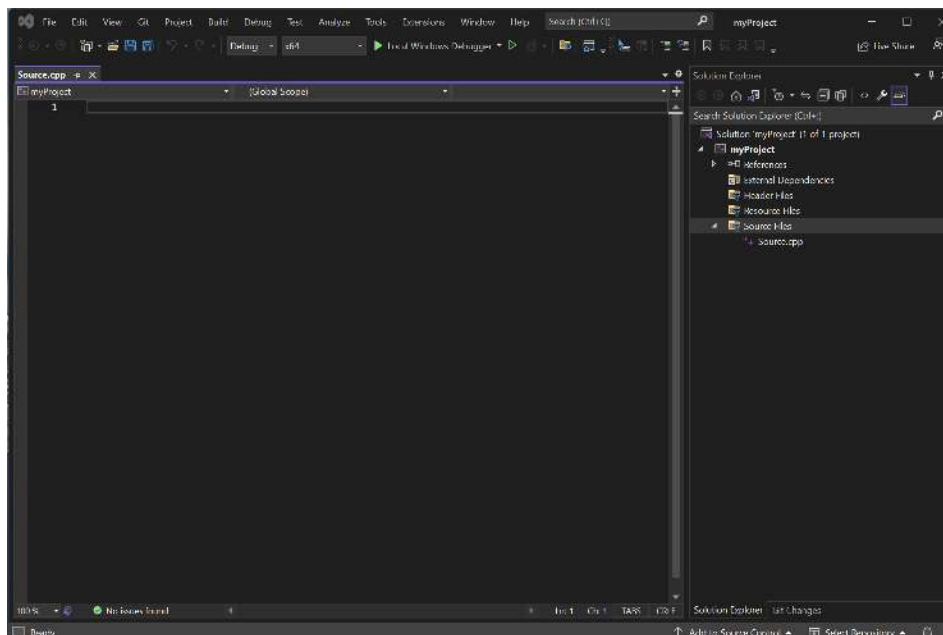
Step 4: Name your project (here I name my project as "myProject"). Please pay attention to the default location where the project is saved. You may change it to the location where you prefer such as "Desktop". Then click on "Create". We will see following screen:



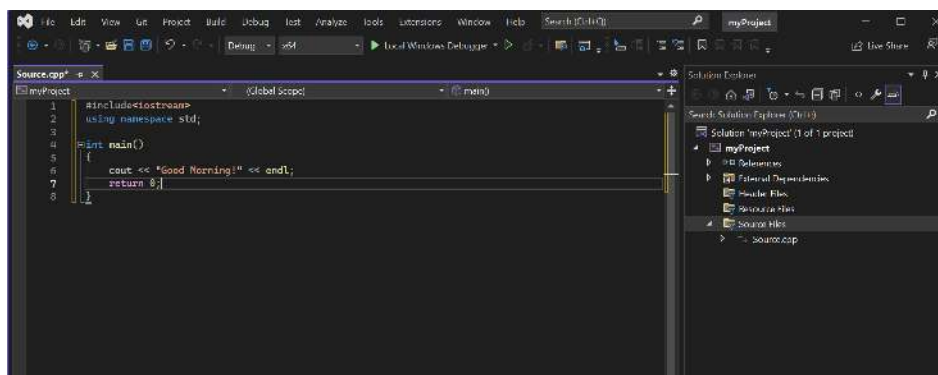
Step 5: Right click on "Source Files", then select "Add" and then select "New Items..." as shown below:



Now, a blank file is opened, and this is where we will write our C++ programs:

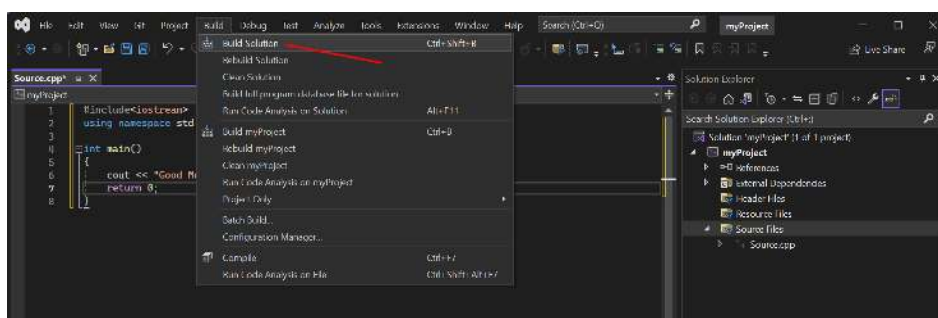


Step 6: Key in the code of a simple C++ program as shown below:

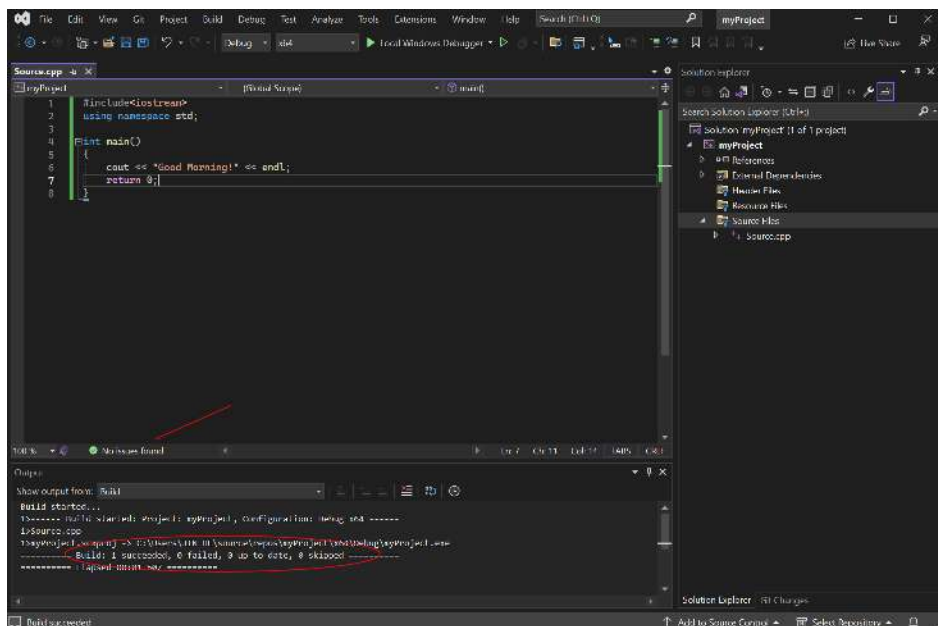


This program will display a message "Good Morning!" on the output screen of the program. A C++ program need to be built before it can be executed. When we say "Build a program" it means error-checking and linking all the computer resources which are required for the program to run.

Step 7: Click on "Build" and then select "Build Solution" as shown below:

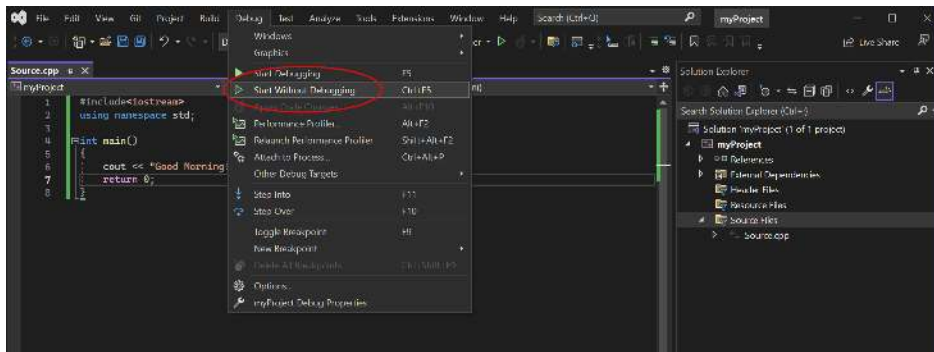


If there is no errors found, we should see following screen with those highlighted words:

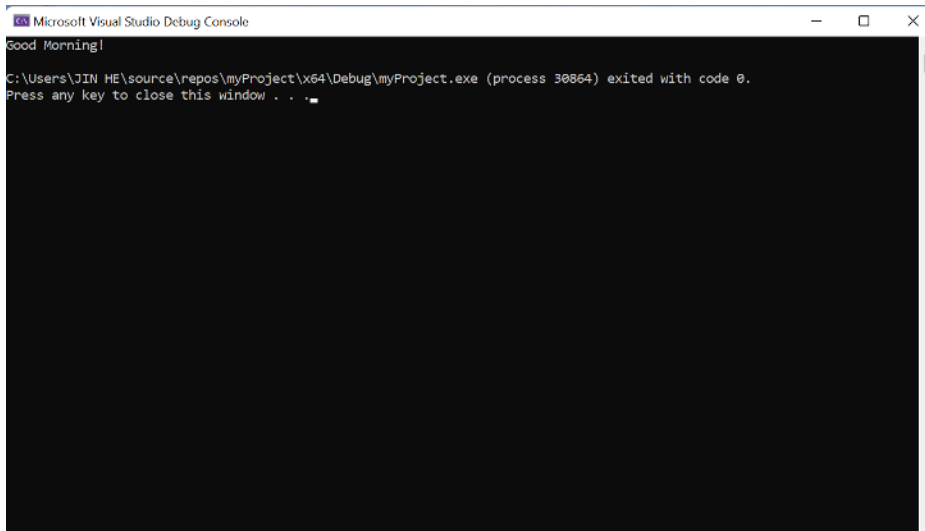


Now we are ready to execute the program.

Step 8: Click on "Debug" and then select "Start Without Debugging" as shown bellow:



Now, the program is being executed and an output window shows up. The message "Good Morning!" appears in the window:

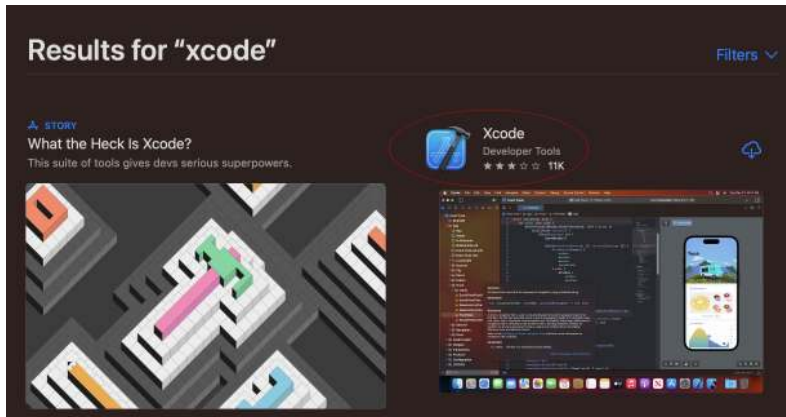


Congratulations! You just successfully wrote, built, and ran a C++ program with the IDE.

This page titled [1.4: Download and Install App for C++ Programming \(PC Users\)](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

1.5: Download and Install App for C++ Programming (Mac Users)

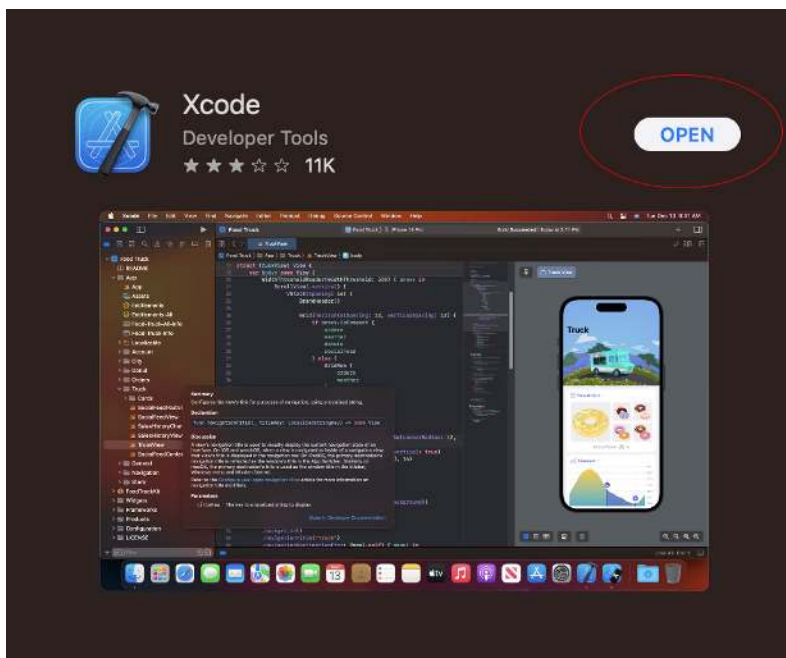
Go to App Store and search for "Xcode" . Download the app: (It may take about 11 GB of your computer memory spaces and it will take a while to download and install)



If have successfully installed Xcode on your Mac computer, congratulations!

Get Started with Xcode

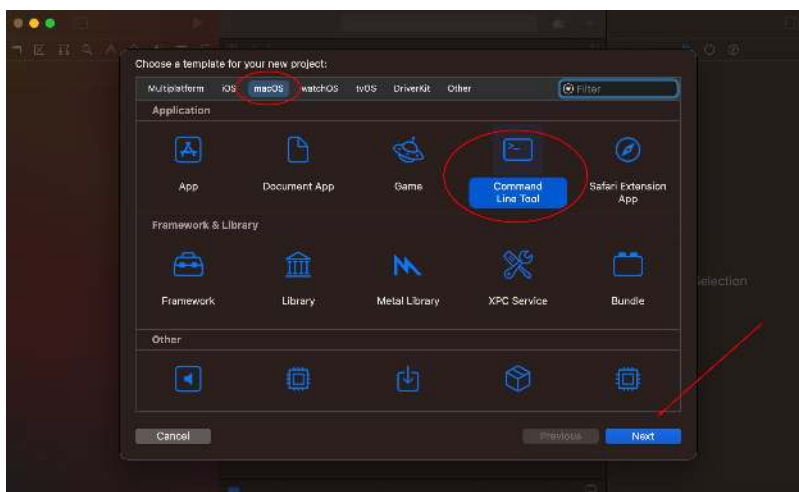
Step 1: Upon the completion of downloading and installing, launch the app (You may click on "Open" as shown below or go to the Launchpad on your computer to find the app and launch it from there)



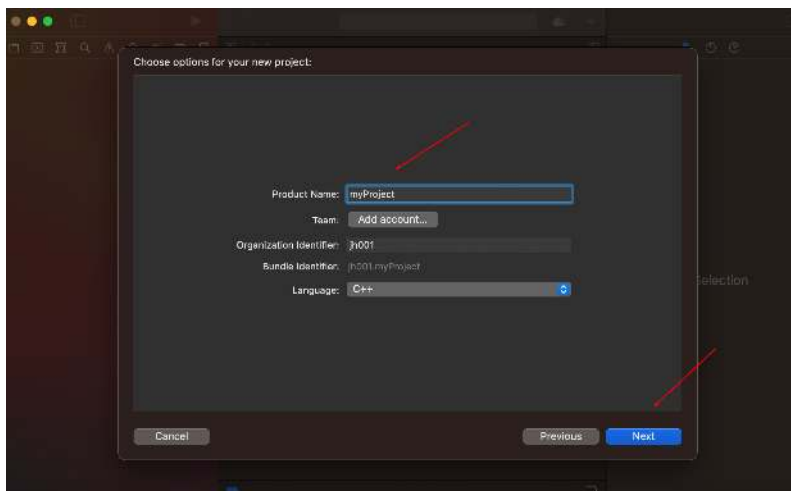
After the app was launched, you will see following window opened:



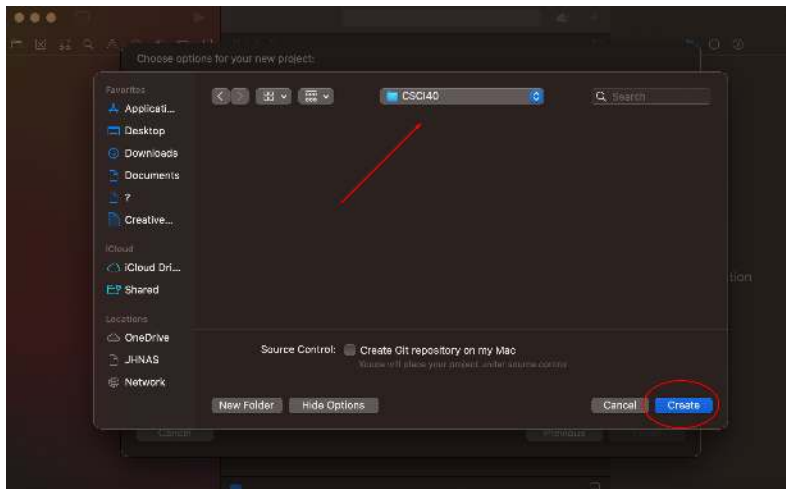
Step 2: Click on "Create a new Xcode project" as highlighted above. Then following window opens:



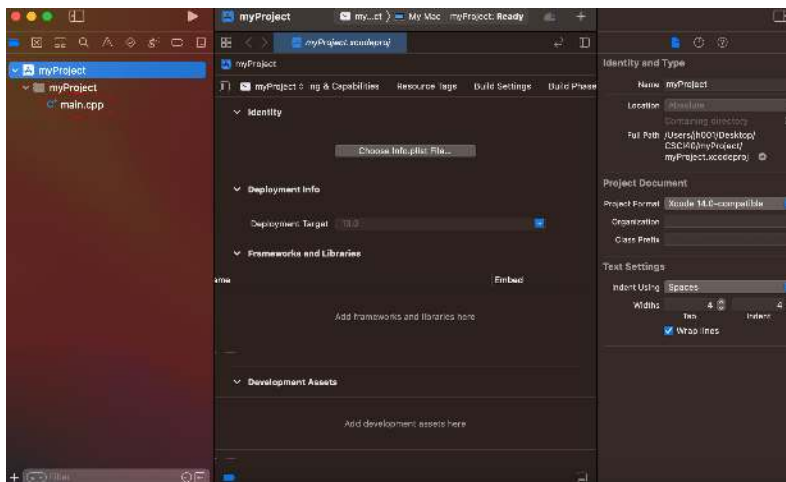
Step 3: Make sure "macOS" and "Command Line" are selected, and then click on "Next". Following window opens:



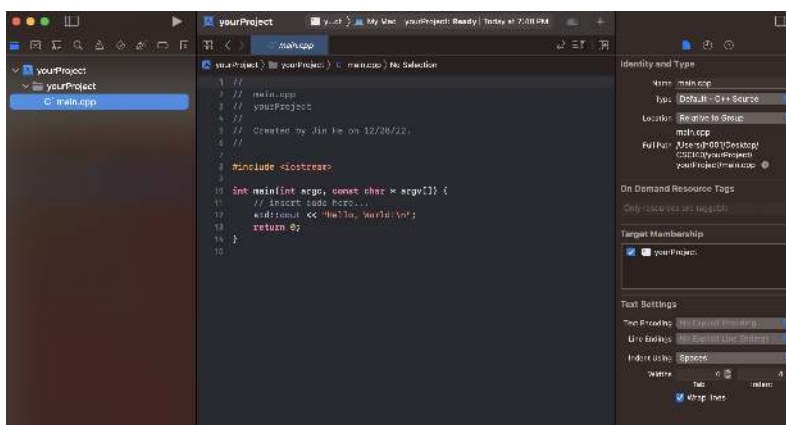
Step 4: Name your project. Here I name my project as myProject. Then click on "Next". We will see following screen:



Step 5: I create a folder "CSCI 40" on the desktop and navigate to this folder as pointed. It means, the project and associated files will be saved into this folder. Then click on "Create". Now, a project is created and project window will be opened as following:

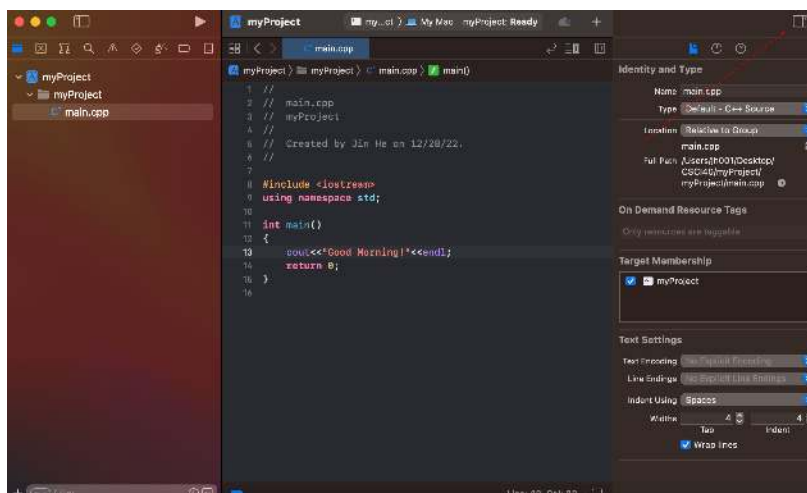


Click on "main.cpp", we will see following screen:



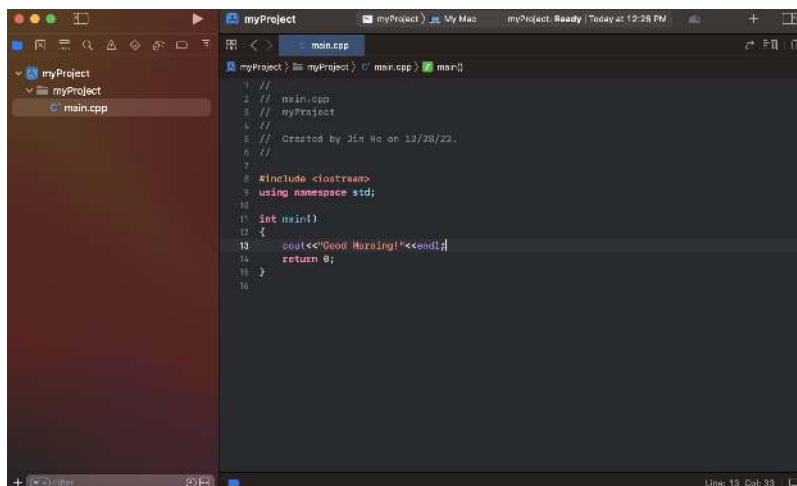
The source file "main.cpp" opens in the middle panel with preloaded code.

Step 6: Modify the preloaded code as following:



You may close the right panel to expand the view of the source file by clicking the icon on the up right corner as pointed above.

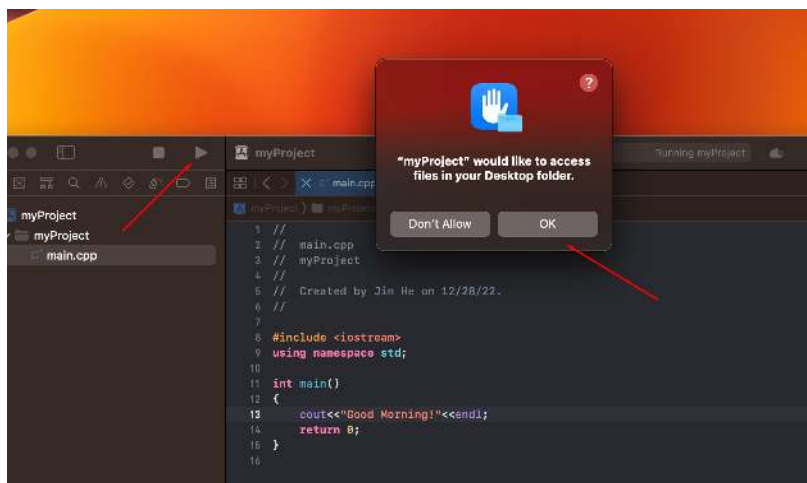
Now, the project window looks like below:



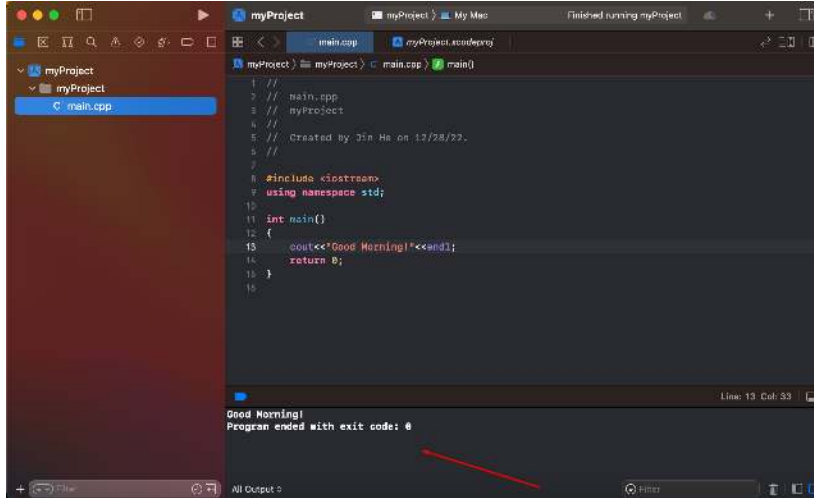
You may close the right panel to expand the view of the source file by clicking the icon on the up left corner.

Now, let's compile and run the program.

Step 7: To compile and run the program, click on the triangle as pointed below: (click on OK on the popup window)



After the program is being executed, a new panel shows up at the bottom of the project window. The output of the program is displayed inside of this panel. For this sample program we are running, the output of the program is a message: Good Morning! We refer this panel as Output Window.



To write another program, you may create a new project by following the steps shown above, or keep this project open and modify the code.

Congratulations! You have successfully written and execute a C++ program with Xcode on a Mac machine.

This page titled [1.5: Download and Install App for C++ Programming \(Mac Users\)](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

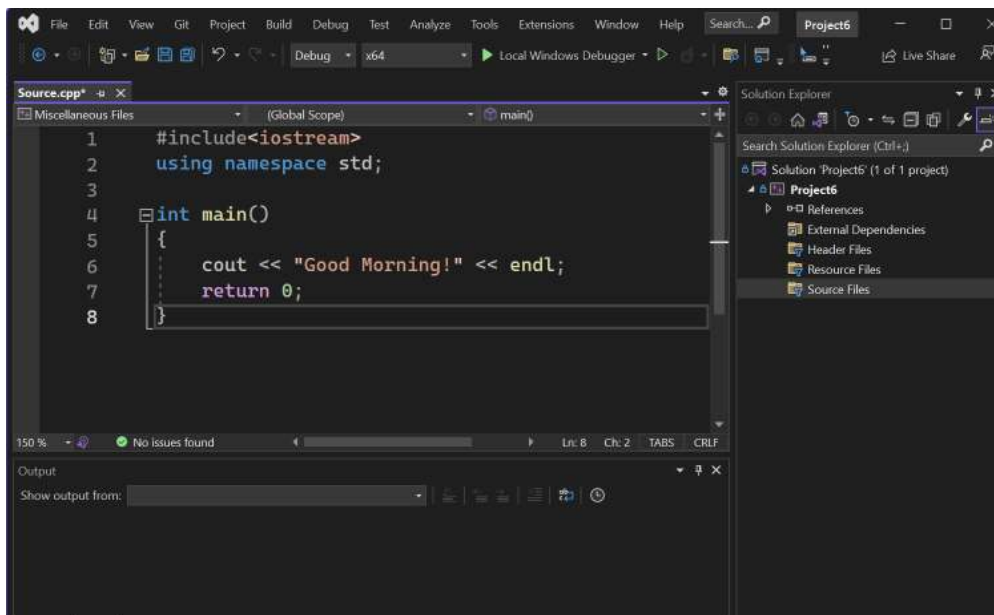
1.6: Save a Copy of Source File

If you would like to save all the C++ programs you wrote throughout the course, normally, you don't want to save the projects you created for each program you wrote. You will only need to save the source file of your program (The file with .cpp as its file extension instead of .sln)

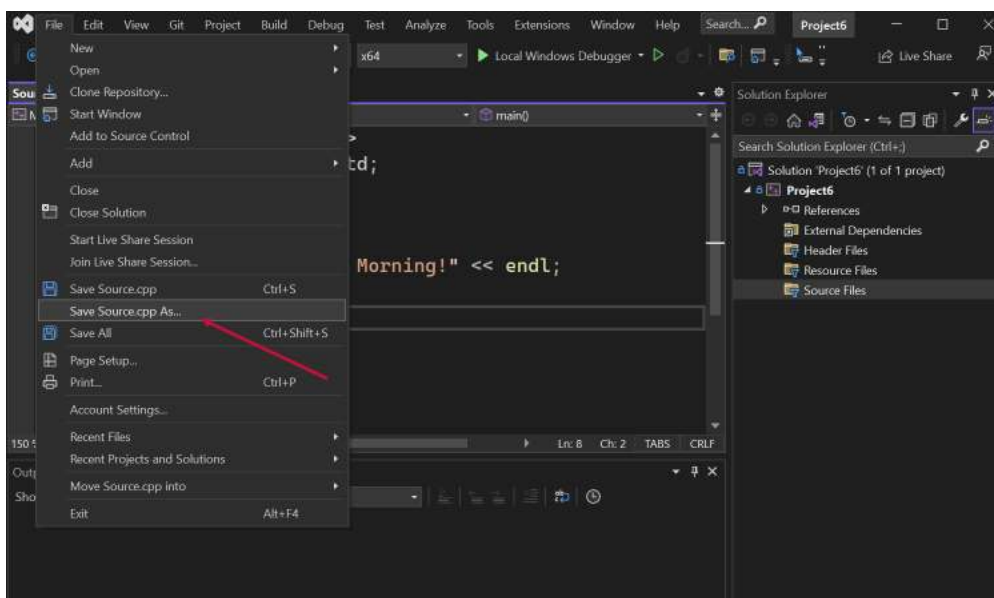
Although the source file of your program is located within the project you created for the program and you can find it somewhere inside of the project folder on your computer hard drive, it is recommended to simply save a copy of the source file of your program in a convenient location such as desktop, so you can easily find it and upload it for an assignment submission.

Here is the instructions:

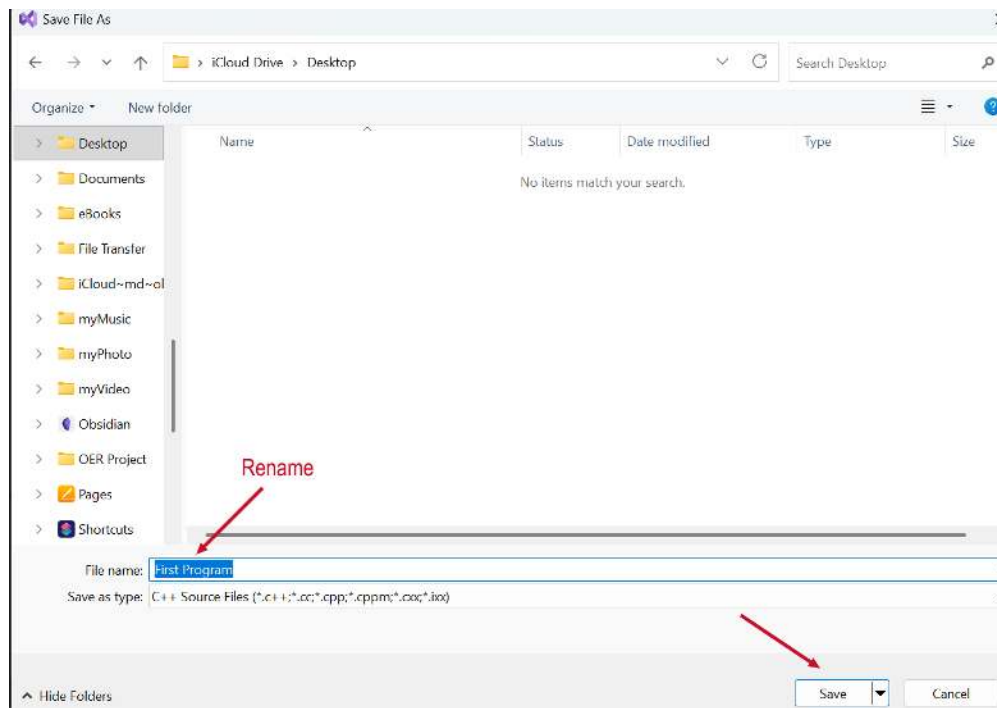
1. After you successfully execute the program, close the output window by press any key on the keyboard of your computer. The app's window remains open as following:



2. Now, lick on "File" on the Tool Bar and select "Save Source.cpp As..." shown as following:



3. After clicking on "Save Source.cpp As...", navigate to the location, such as "Desktop", you may rename the file and then click on "Save"



Now, a copy of your source file is saved in the location of your choice. Please note: this is a copy of the source file of your program. You can edit it, but the change you made in this file can't be synchronized with the source file inside of the project. In other words, if you execute the program inside of the project, you will not see the changes you made on the file you saved.

If you do want to run the file you saved, the best way to do it is to create a new project, open an empty source file in the new project. Select all the code in the saved file, copy and paste into the empty file in the new project. Then build and execute it in the new project.

This page titled [1.6: Save a Copy of Source File](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

1.7: Understanding the Code of the First C++ Program

Following is the code of our first C++ program:

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

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

Let's discuss this simple program. Please note, some of the concepts you may not fully understand at this point, but you will be getting a better understanding as we move along with the course.

Line 1: `#include<iostream>`

is a compiler *preprocessor directive* that tells the compiler to link the "*iostream library*" to this program, which is needed to support console input and output. Console input means to receive input from the keyboard and console output means to display output on the monitor screen.

C++ library contains pre-defined code for developing C++ programs. *iostream* is one of C++ library which is called a header file in C++ because it is usually included at the head of a program.

Line 2: `using namespace std;`

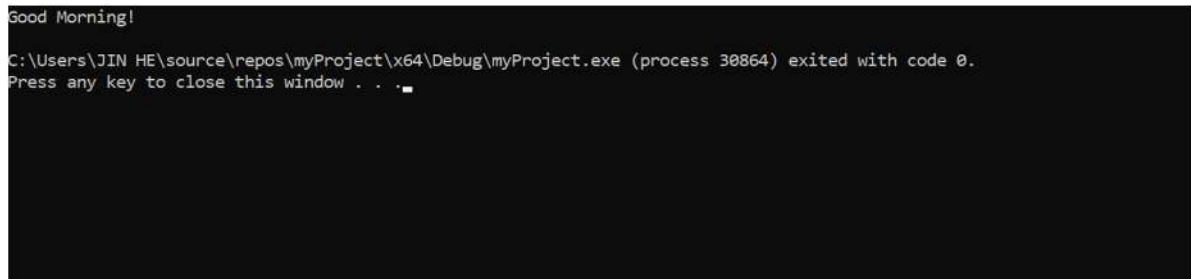
tells the compiler to use the standard namespace. Again, you may not understand what it means, it is ok, just place this line at the beginning of every program you are going to write. The reason we put this line here is we can use "cout" for displaying symbols or messages instead of using "std::cout"

We will need those two lines of the code at the beginning of every C++ program.

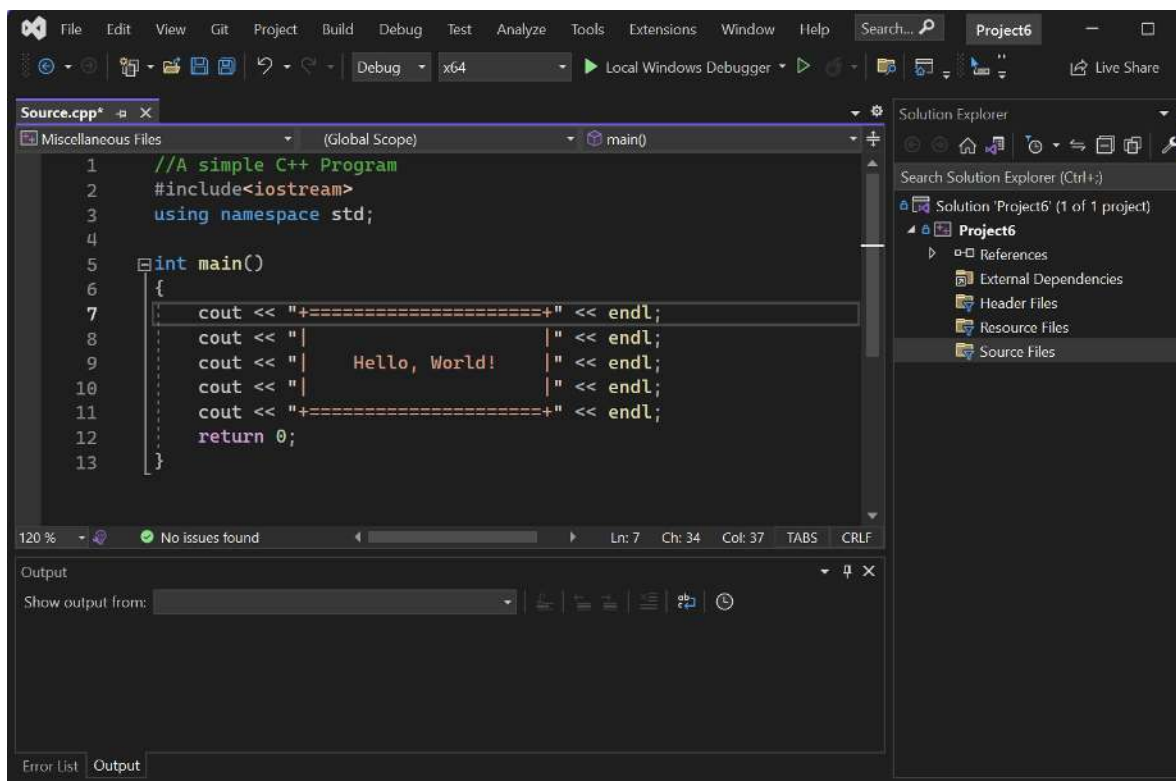
In C++, all the statements, instructions, and commands in a program will need to be placed inside a ***main() function***. A main() function starts with a word "int" and followed by blank space and "main()". After "main()", there is pair of braces, all the code, including statements, instructions, and comments, of your program should be placed between those braces:

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

Upon the completion of the program, the message "Good Morning!" will be displayed on the output window as following:



If we change the code between the braces in above program as following



```

1  //A simple C++ Program
2  #include<iostream>
3  using namespace std;
4
5  int main()
6  {
7      cout << "=====+" << endl;
8      cout << "          |" << endl;
9      cout << "    Hello, World!  |" << endl;
10     cout << "          |" << endl;
11     cout << "=====+" << endl;
12     return 0;
13 }

```

then upon the completion of the program, the display in the output window will be shown as following:

 preview

This program consists of six statements. The first five statements begin with "cout<<" , where **cout** (read as see-out) stands for "console output" and << is called "stream insertion operator". We call **cout<<** as a "see-out statements". At the end of each of those statements, we place "<<endl;" which we refer it as "end of line character" It tells the computer "This is the end of this line, if you want to continue to display something, please start at the beginning of the next line"

Anything that we would like to display on the output window, we will need to place them within a pair of quotation marks.

For example, the line 7 displays a string of characters: =====+ and the line 8 displays a string of blank spaces beginning with '|' and ending with a '|'. The line 9 displays a string beginning with '|' followed by a sequence of blank spaces and a phrase "Hollo World!" followed by another sequence of blank spaces, and ending with '|', and so on.

The last statement

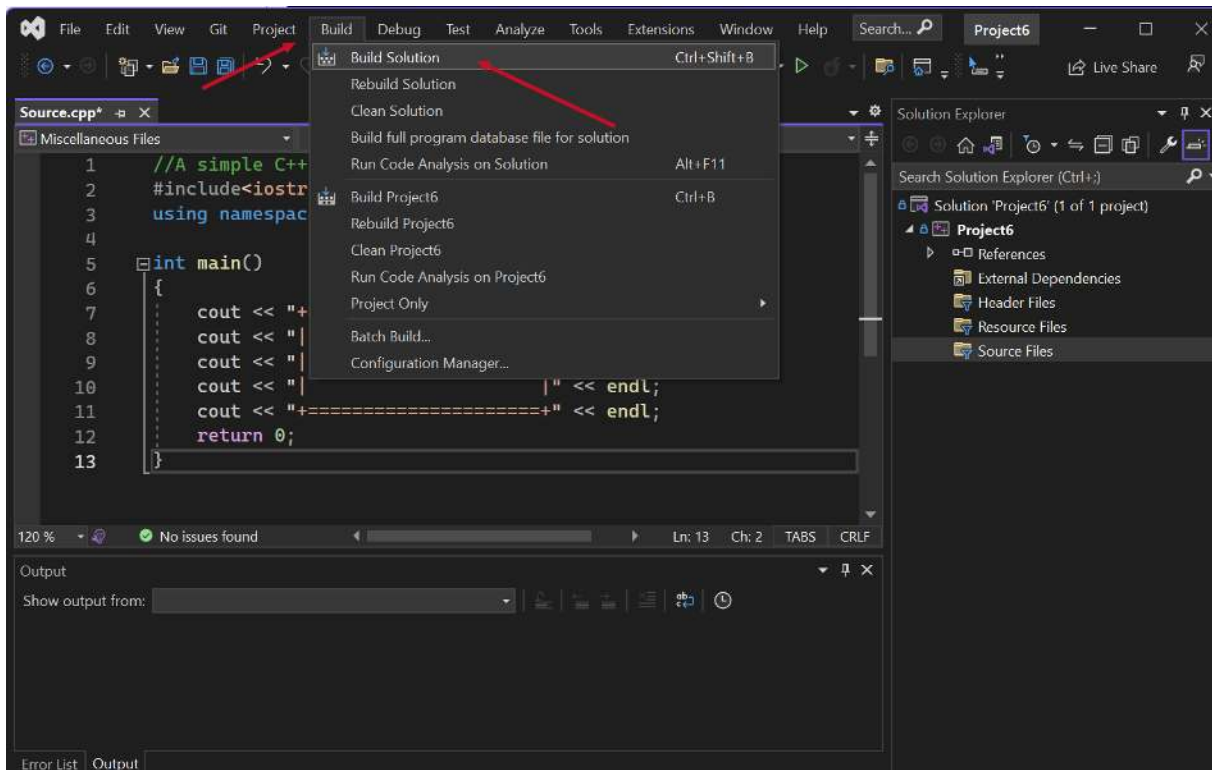
return 0;

is required at end of every program. This statement tells the computer that we have reached the end of the program.

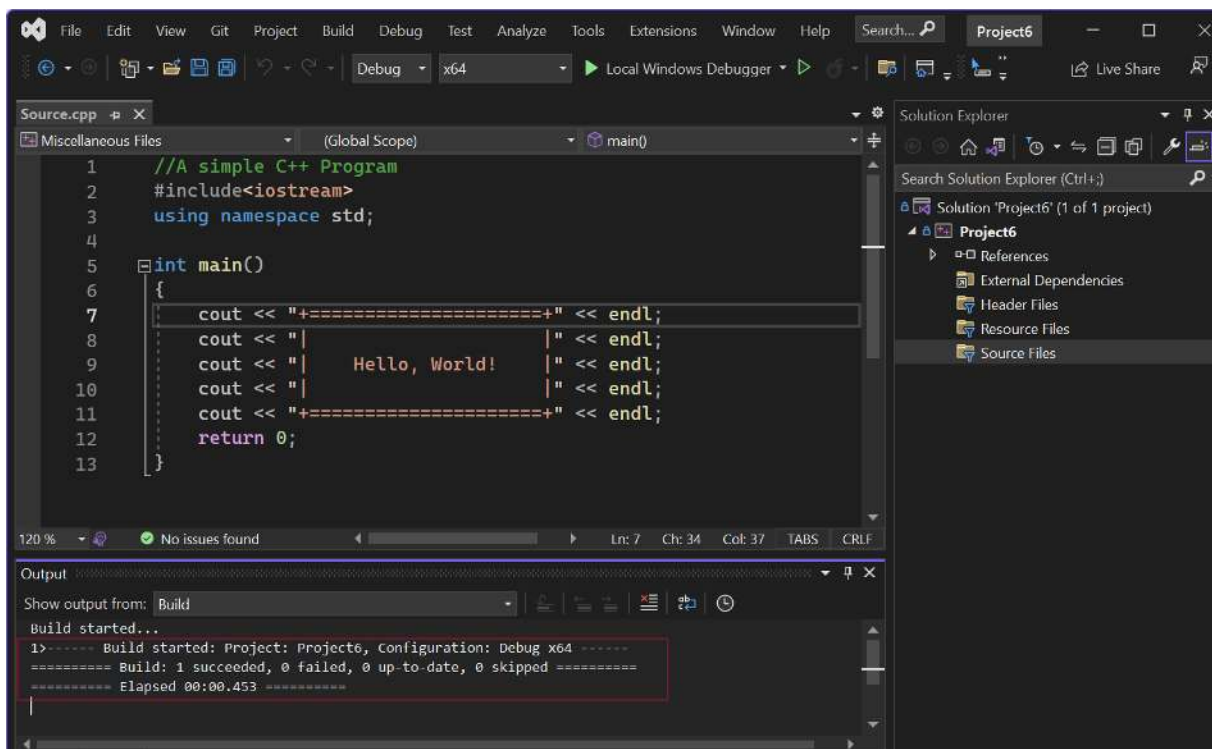
After we finish writing our program, the first thing we need to do is to "Build" the program. Building a program means

- Doing the error checking
- Translating the program to machine language so the computer can understand directly.
- Linking all the resources required to run the program

To build the program, click on "Build" and choose "Build Solution":

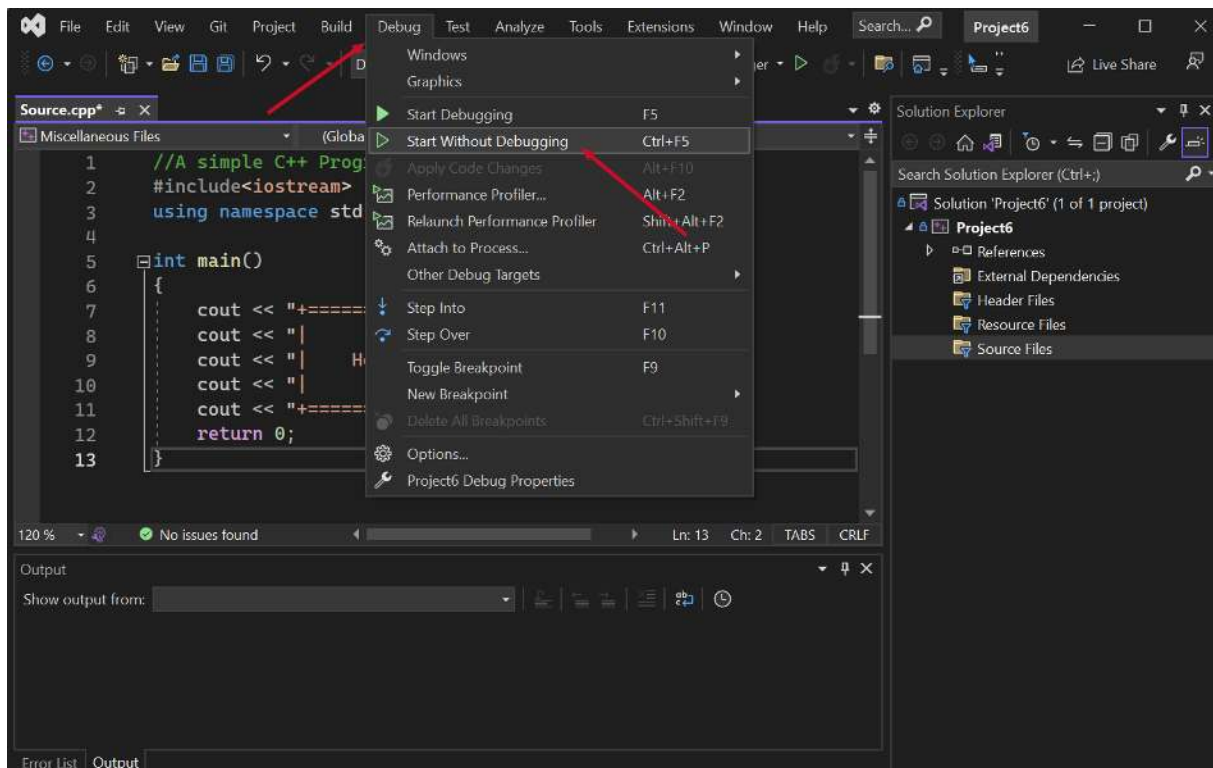


After successfully build the program:



we will need to send the program to computer CPU to have it executed (run).

To execute the program, click on "Debug" and choose "Start without debugging"



Upon the execution of the program, following output window opens and the output of the program shows on the screen:

 preview_mglj

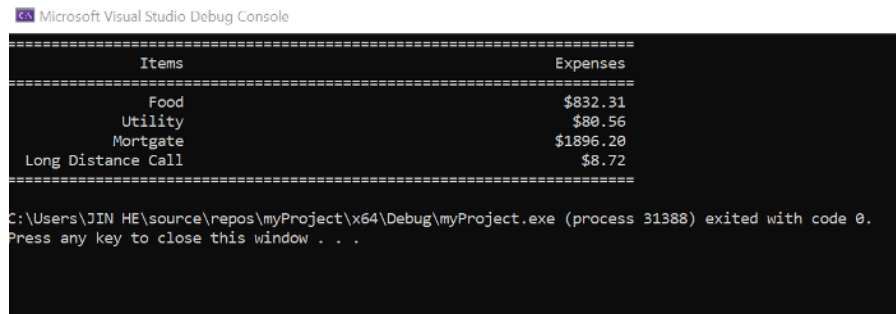
We can press any key to close this window and go back to the source file.

Congratulation! You are on your way to write more C++ programs.

This page titled [1.7: Understanding the Code of the First C++ Program](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

1.8: Case Study- Using setw() to Format the Output

Here we will discuss how to write a program that displays a table of monthly household expenses. This program will produce an output as following:



```

=====
Items                               Expenses
=====
Food                                $832.31
Utility                             $80.56
Mortgage                            $1896.20
Long Distance Call                   $8.72
=====

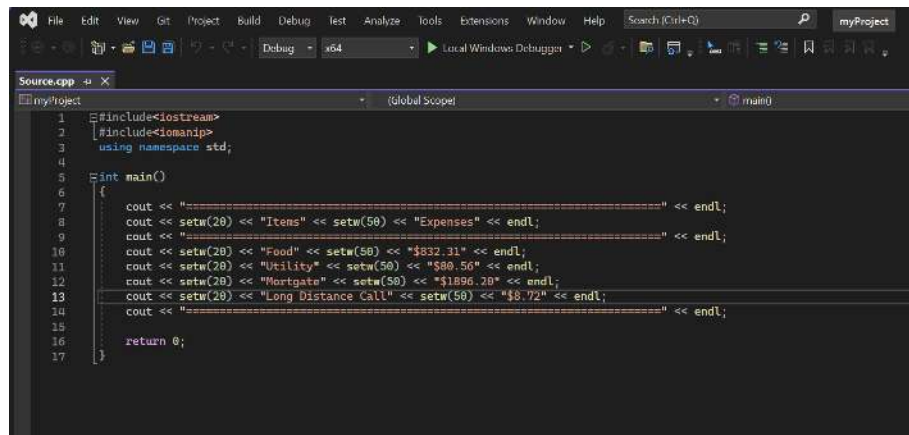
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 31388) exited with code 0.
Press any key to close this window . . .

```

Step 1: Launch the app (Visual Studio or Xcode) and create a new project and name it as "Align Output"

Step 2: Open a source file and clear all the preloaded code in the source file if there is any.

Step 3: Enter the following code into the source file:



```

1  #include<iostream>
2  #include<iomanip>
3  using namespace std;
4
5  int main()
6  {
7      cout << "===== " << endl;
8      cout << setw(20) << "Items" << setw(50) << "Expenses" << endl;
9      cout << "===== " << endl;
10     cout << setw(20) << "Food" << setw(50) << "$832.31" << endl;
11     cout << setw(20) << "Utility" << setw(50) << "$80.56" << endl;
12     cout << setw(20) << "Mortgage" << setw(50) << "$1896.20" << endl;
13     cout << setw(20) << "Long Distance Call" << setw(50) << "$8.72" << endl;
14     cout << "===== " << endl;
15
16     return 0;
17 }

```

Notes:

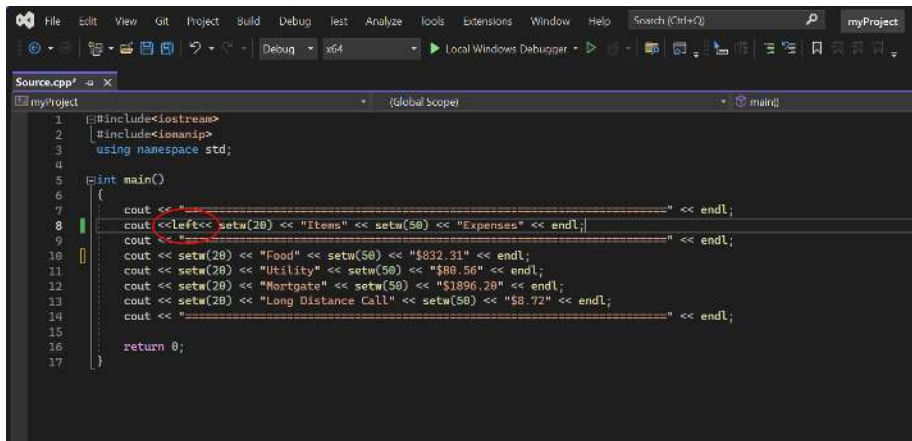
Line 2: `#include<iomanip>` is required for this program because the command `setw()` is used in line 8, 10, 11, 12, 13

The numbers used in each `setw()` command is for setting the width of a column. Since there are two columns in the table, we use `setw()` twice in the lines 8, 10, 11, 12, 13. In the first `setw()` we use number 20 for the width of the first column and the second `setw()` we use 50 for the width of the second column.

Step 4: Run the program and observe the output. Adjust code in line 7, 9, and 14 if the lines are not long enough to cover the table width by adding a few of '='s between the quotation marks or delete a few of '='s if the lines are too long

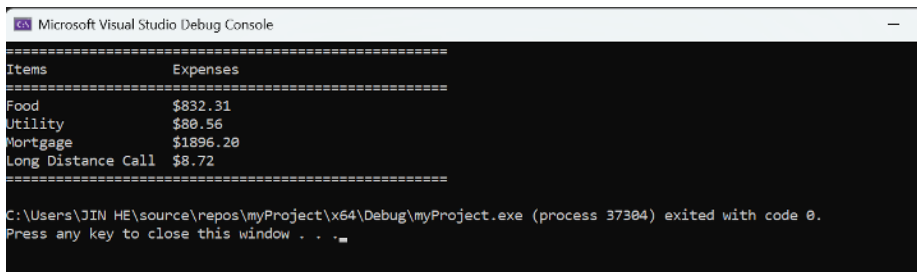
Step 5: Change the number in the first `setw()` to 15 and the number in the second `setw()` to 30 in line 8, 10, 11, 12, 13, then run the program and observe the output of the program

Step 6: Insert a word "left" into the line 8 as shown below:



```
1 #include<iostream>
2 #include<iomanip>
3 using namespace std;
4
5 int main()
6 {
7     cout << endl;
8     cout << left << setw(20) << "Items" << setw(50) << "Expenses" << endl;
9     cout << "=====" << endl;
10    cout << setw(20) << "Food" << setw(50) << "$832.31" << endl;
11    cout << setw(20) << "Utility" << setw(50) << "$80.56" << endl;
12    cout << setw(20) << "Mortgage" << setw(50) << "$1896.20" << endl;
13    cout << setw(20) << "Long Distance Call" << setw(50) << "$8.72" << endl;
14    cout << "=====" << endl;
15
16    return 0;
17 }
```

Step 7: Run the program again and observe the output of the program. Now the contents in each column are aligned to the left instead of right:



```
=====  
Items      Expenses  
=====  
Food       $832.31  
Utility     $80.56  
Mortgage   $1896.20  
Long Distance Call $8.72  
=====  
  
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 37304) exited with code 0.  
Press any key to close this window . . .
```

This page titled [1.8: Case Study- Using setw\(\) to Format the Output](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

CHAPTER OVERVIEW

2: Chapter 2 Elementary Programming

- [2.1: Constants and Variables](#)
- [2.2: Console Input and Output](#)
- [2.3: Assignment Statements](#)
- [2.4: Case Study- Sum of Numbers](#)
- [2.5: Numerical Operators](#)
- [2.6: Coding a Mathematical Formula](#)
- [2.7: C++ Examples](#)
- [2.8: Case Study- Find the Sum of Numbers](#)

This page titled [2: Chapter 2 Elementary Programming](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

2.1: Constants and Variables

Constants

A **constant** is a value that cannot be altered by the program during normal execution. Normally, a constant will be associated with a name when it is used in a program. For example, we can write a line of the code in a program to set up a constant which we can use throughout the program:

```
const double PI=3.1415926;
```

In programming, we often call a line of the code a **statement**. Each statement must be ended by a semi-colon. In this statement, *const* is called a key word which specifies the statement is for setting up a constant. As a key word, it is pre-defined and can not be changed to something else. *double* is also a key word which specifies that the constant we are setting up is a decimal number. In computer programming, data a program uses, receives, or outputs are categorized into different types: decimals (*double*), integers (*int*), characters (*char*), strings (*string*), or Boolean values (*bool*) and so on. For example, if we would like to write a statement that is setting up a constant with 100 as its value, we would write this statement as following:

```
const int MAX=100;
```

In above statements, the words *PI* and *MAX* are names we used for those constants in the program, and they are also called **identifiers**. Such constants are also called named constants. In C++, we use ALL CAPS for a named constant. As a coding person of a program, we can use any ALL CAPS string as the name of a constant. However, certain rules are applied:

- We normally don't begin an identifier with a digit, such as 5PGE.
- We normally don't use special character in an identifier, such as \$, or PG&E
- We cannot use blank space in an identifier, such as STUDENT ID. If you do want to use multiple words for an identifier, you may use STUDENT_ID

In program that contains above two statements, the values of *PI* and *MAX* can not be altered by the program during normal execution, which means if you write a statement:

```
PI=3.14;
```

or

```
MAX=200;
```

in the same program, then you can't successfully build and run the program, you will encounter syntax errors.

Variables

A variable in a C++ program is an identity that represents a unit in a computer memory for storing data. A variable must be associated with an identifier (a variable must be named before we can use it in a program. For example, if we would like to write a program that is for calculating the area of a rectangle and we would like to use two variables in this program for storing the length and the width of the rectangle, then we may write a statement that associates two identifiers with those two variables as following:

```
double length, width;
```

In this statement, the key word *double* was used to specify that both variables are used in this program for storing decimals. The word *length* and *width* are identifiers (name of the variables). Technically, we can use any string for the name of a variable, but certain rules are often apply:

- We don't use ALL CAPS for a variable identifier, such as LENGTH or WIDTH
- We normally don't capitalized the initial of an identifier, such as Length or Width
- We can't use a blank space in an identifier, such as test score, instead, we will use *test_score* or *testScore*
- We don't use special characters in a variable identifier, such as @userName
- We don't start an identifier with a digit, such as 20row, instead, we can use row20

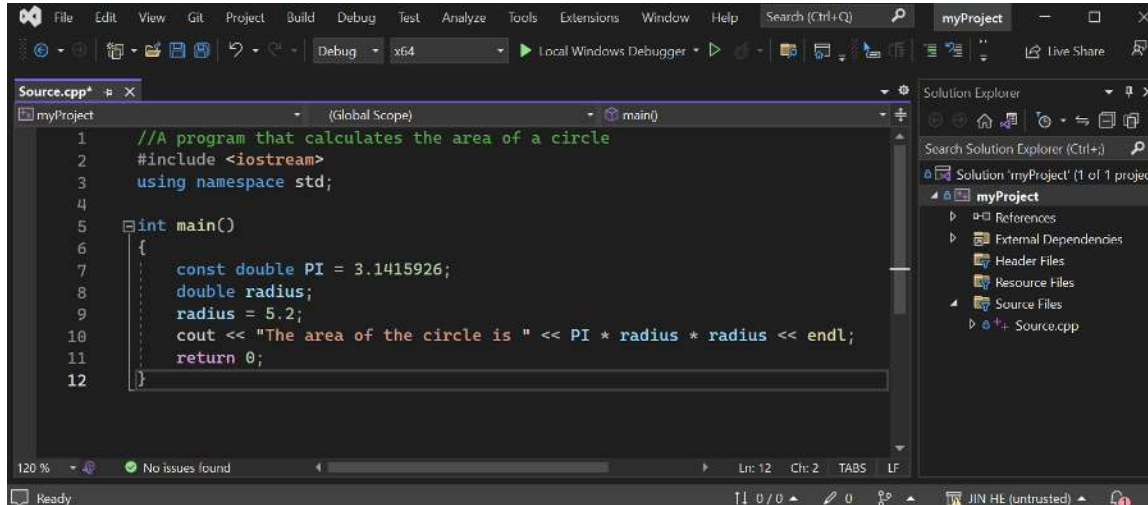
The statements used for associating variables to identifiers are called variable declarations. If a statement associates one or more variables to their identifiers, we say such statement "declares" those variables. Each variable must be declared before it can be used

in a program.

Now we discuss how to use constants and variables in a C++ program.

Example Write a program that calculates and displays the area of a circle with radius equals to 5

First we will need to create a project, and then add a source file to the project. You have to follow the steps we discuss in the [Section 1.5](#). Following is a screenshot of the program:



```

1 //A program that calculates the area of a circle
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     const double PI = 3.1415926;
8     double radius;
9     radius = 5.2;
10    cout << "The area of the circle is " << PI * radius * radius << endl;
11    return 0;
12 }

```

Line 1: This line is for documentation purpose. It must begin with `//` and anything follows `//` will be treated as documentation and will not be considered as the code of the program.

Line 2: This is a *compiler preprocessor directive* that tells the compiler to include the *iostream* library in this program, which is needed to support console input and output. C++ has many libraries which contain pre-defined code for developing C++ programs. The library like *iostream* is called a header file in C++, because it is usually included at the head of a program.

Line 3: this line of the code tells the compiler to use the standard namespace. (You may not understand what it means now, no worry, just put this line in every program you develop in this course)

Line 5: This line sets a block for the code of the program. It begins with the key word *int* followed by the identifier *main* with a pair of parentheses. All the code of the program should be placed within `{` and `}`. A program can't run without this building block. We call this building block the *main()* function. (We will discuss C++ functions in later chapter in more details) The *main()* function begins at line 5 and ends at line 12.

Within *main()* function,

Line 7: It is a statement that specifies a constant named as *PI* with 3.1415926 as its value

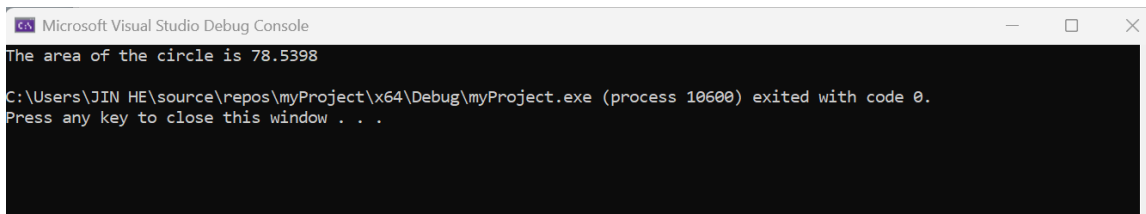
Line 8: This is a statement that declares a variable with identifier "radius" and specifies that this variable is for storing decimal data. Since radius of a circle is a measurement which are always in the form of fractions or decimals

Line 9: This is a so called "**assignment statement**", it assigns a value, in this case it is 5.2, to the variable *radius*

Line 10: This is an output statement, it displays the calculation result (the area of the circle) on the output screen. In this statement, `"cout<<"` will send the message "The area of the circle is " to the screen first and performs the calculation $PI * radius * radius$ and send the result to the screen through the second `"<<"` (which is called *stream insertion operator*). The result follows the message to provide a better user interface. At the end of this statement, `"endl"` is an end-of-line character and is sent to output window through the third stream insertion operator `"<<"` to tell the computer that if there is something else to display after this statement, the new display will be at the beginning of the next line.

Line 11: This statement should be placed at end of every *main()* function to exit the program. The value of 0 indicates that the program has terminated with a successful exit. Some compilers will work fine without this statement, but some may not. To play safe, it is good practice to place this statement at the end of *main()* function.

After coding the program, press Ctrl+F5 to build and execute the program. Following is a screenshot of the output window of the program:



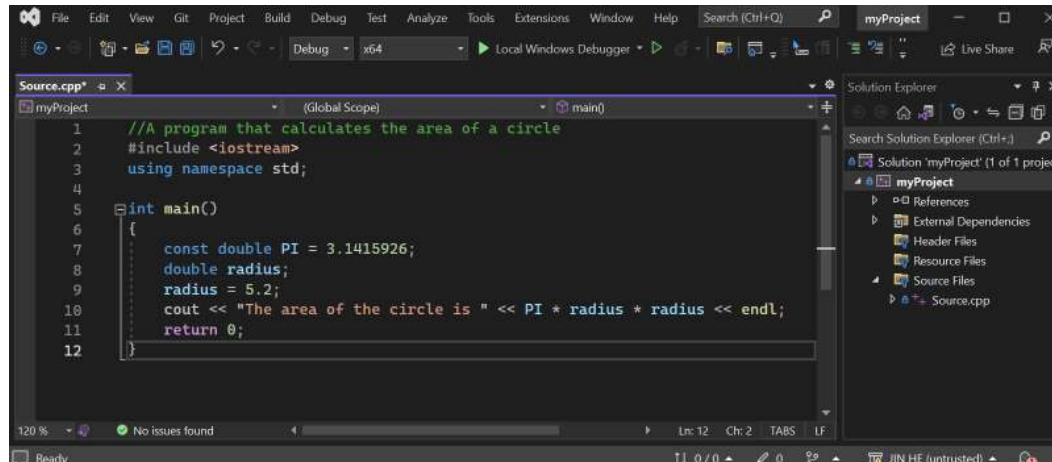
```
Microsoft Visual Studio Debug Console
The area of the circle is 78.5398
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 10600) exited with code 0.
Press any key to close this window . . .
```

From above discussions, we know identifiers are the names that identify elements in a program such as constants, variables, and functions. As an identifier, it is case sensitive. For example, abc and ABC are two different identifiers.

This page titled [2.1: Constants and Variables](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

2.2: Console Input and Output

Following is the sample program we discussed in the previous section:



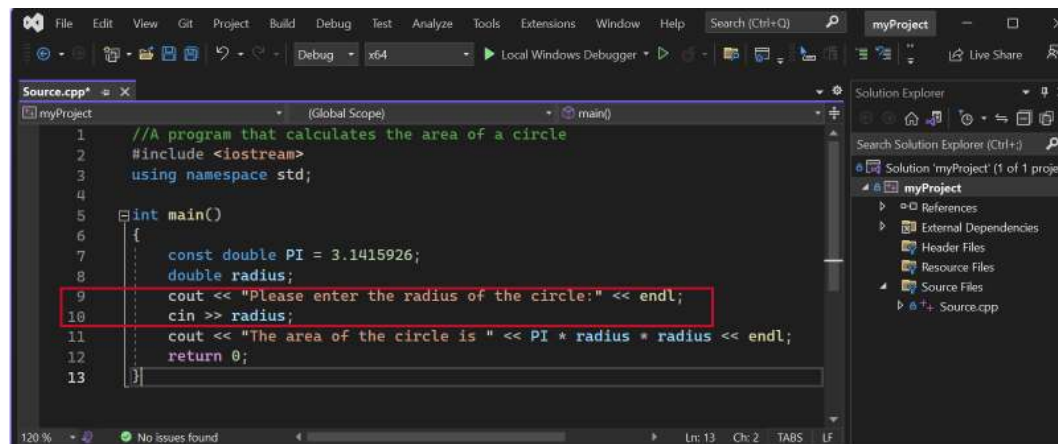
```

1 //A program that calculates the area of a circle
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     const double PI = 3.1415926;
8     double radius;
9     radius = 5.2;
10    cout << "The area of the circle is " << PI * radius * radius << endl;
11    return 0;
12 }

```

This program is fine as far as calculating the area of a circle. But, the problem is that this program can only calculate the area of a circle with its radius equal to 5.2. It can not be used for calculating the area of other circle with its radius other than 5.2 unless we change the code of the source file the program. We prefer to have a program that can be used to the area of a circle with any given radius without changing the code of the program.

Now, we modify above program by deleting line 9 in the original program and add line 9 and line 10 as following:



```

1 //A program that calculates the area of a circle
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     const double PI = 3.1415926;
8     double radius;
9     cout << "Please enter the radius of the circle:" << endl;
10    cin >> radius;
11    cout << "The area of the circle is " << PI * radius * radius << endl;
12    return 0;
13 }

```

Line 9: This is a console output statement, sometimes we simply call such statement a "see-out statement". It displays the message "Please enter the radius of the circle: " on the output screen served as prompt to let a user know what need to do.

Line 10: This is a console input statement. It uses the word "cin" read as "see-in" followed by "<<" which is called stream extraction operator. Following the stream extraction operator, is the variable we declared in line 8. This statement will pause the execution of the program:

```
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe
Please enter the radius of the circle:
```

and wait until a user of the program enter a numerical value through the keyboard and then press "Enter"

```
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe
Please enter the radius of the circle:
5
```

After the key "Enter" was pressed, then the numerical value the user entered will be assigned to the variable "radius" and the execution of the program proceed to line 11.

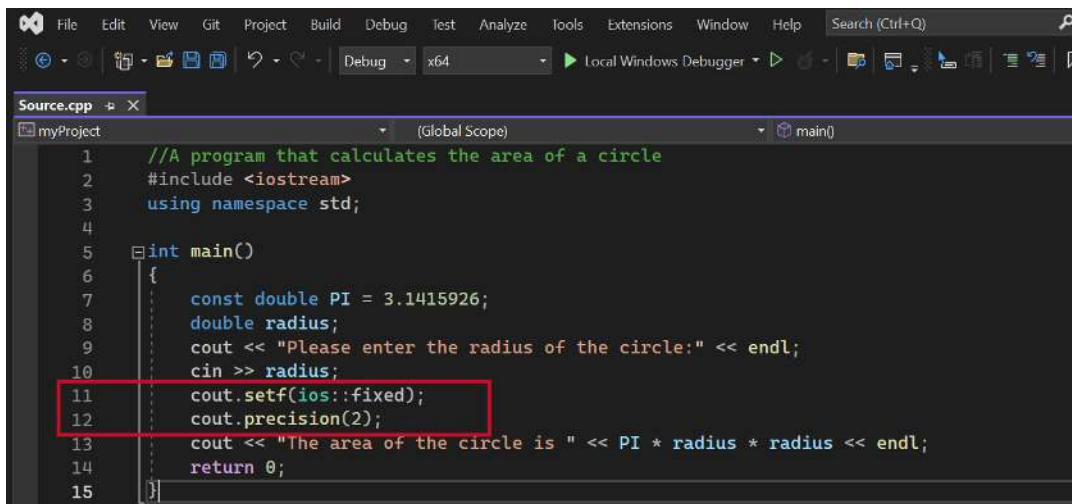
Line 11: Again, this is a "see-out" statement. In this statement, the first stream insertion operator "<<" send the message "The area of the circle is " to the screen, and the second "<<" send the result (are of the circle) to the screen and it is following the message to provide a better user's interface. The third << send the end of line character to the screen to tell the computer if there is something need to print, it will start from the beginning of the next line.

Find final output window will be shown as following:

```
Microsoft Visual Studio Debug Console
Please enter the radius of the circle:
5
The area of the circle is 78.5398

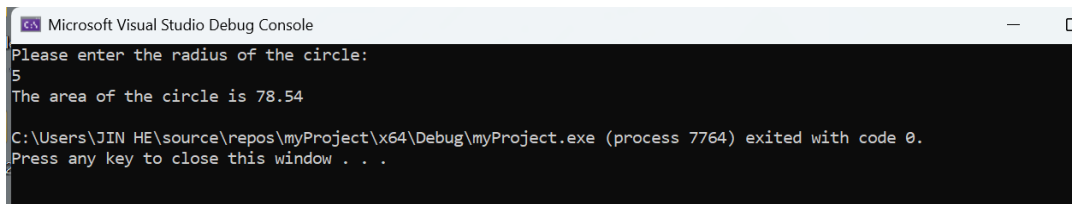
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 30960) exited with code 0.
Press any key to close this window . . .
```

If we only want to keep two digits after the decimal point, we can add two more lines of the code in the program shown as below:



```
1 //A program that calculates the area of a circle
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     const double PI = 3.1415926;
8     double radius;
9     cout << "Please enter the radius of the circle:" << endl;
10    cin >> radius;
11    cout.setf(ios::fixed);
12    cout.precision(2);
13    cout << "The area of the circle is " << PI * radius * radius << endl;
14    return 0;
15 }
```

now, the output of the program will be:



```
Microsoft Visual Studio Debug Console
Please enter the radius of the circle:
5
The area of the circle is 78.54
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 7764) exited with code 0.
Press any key to close this window . . .
```

This page titled [2.2: Console Input and Output](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

2.3: Assignment Statements

Assignment Operator And Assignment Statements

In the coding of C++ programs, the symbol '=' does not mean equal. It is an assignment operator. We use '=' to assign a numerical or a logical value to a variable. We also use it to assign a character or a string to a character variable or a string variable. For example, the following code in a C++ program

```
a = 5;
```

does not mean "The value of the variable *a* is equal to 5". What the code means is "Assigning integer 5 to the variable *a*". To be more specific, this code sends the integer 5 into one of the data storages located in the computer memory and this storage is represented by the variable *a* in this program. Such statement is called an **assignment statement**.

Let's take a look at following three lines of the code in a C++ program:

```
int a = 0;  
a = 5;  
a = 11;
```

Line 1: The first line of the code declares an integer variable *a* beginning with a word 'int'. It tells the computer, *a* is a variable that represents a data storage that can only be used for storing integer type of data, if you put a decimal number, say 2.5, in this storage, then the decimal part of the data will be lost, only the whole number part of this data, which is 2, is kept in the storage. This line also sends integer 0 into the data storage represented by the variable *a*. Each variable used in a program needs to be declared which means we need to put a line of code that specifies the data type of the variable following by the name of the variable. The first line of the code declares the variable *a*, and also initializes it by an integer 0.

Line 2: The second line of the code "assign" integer 5 into variable *a*. To be more specific, this line sends the integer 5 into the storage represented by the variable *a*. Since each data storage can only hold a single data, the original data, which is 0, will be overwritten by the new data after this line of code.

Line 3: The third line of the code sends the integer 11 to *a* and overwrites the data, 5, that was placed in the data storage previously. Therefore, the final value of the variable *a* after those lines of the code is 11.

An assignment statement can be a simple assignment:

```
int a;  
a=11;
```

An assignment statement can also be an assignment with numerical expressions:

```
int average;  
average=(12+3+7+8)/3;
```

An assignment statement can also be an assignment with an expression that contains numerical values as well as other identifiers. However, those identifiers in the expression must represent the variables that were previously declared and assigned values:

```
double price, tax, total;  
price=$129.99;  
tax: 0.085;  
total=price+price*tax;
```

Please note: If we use an assignment statement to assign a value or the value of an expression to a variable, the variable the values assigned to must be placed on the left side of assignment operator "=". For example, the statement

```
a=17;
```

is a correct statement, but

```
17=a;
```

is not.

Augmented Statements and Augmented Assignment Operators

Sometimes, the current value of a variable is used, modified, and then reassigned back to the same variable by using the code in the form of "a = a+5;" or "a += 5;" equivalently. Here the ' += ' is called an augmented operator, and the code "a = a + 5" is called an augmented statement. the code "a = a + 5;" and the code "a += 5;" are equivalent.

To help you understand this special C++ code, here are some examples:

```
int a = 5;  
a = a + 7;
```

After above two lines (or two statements), the value of **a** is 12. Here is why: In the second line, the code is executed from left, it takes the current value of **a** which is 5 (because we have put 5 into **a** in line 1) and add it to 7, the result is 12 which is put back into **a** by assignment operator '='.

Similarly, the following three lines of code change the value of **a** from 0 to 11 using two augmented statements based on the same logic explained above.

```
int a = 0;  
a = a + 7;  
a = a + 4;
```

Those three lines of code can be equally written with the augmented operator as

```
int a = 0;  
a += 7;  
a += 4;
```

Following table lists a few other augmented operators:

Operator	Name	Example	Equivalent
<code>+=</code>	Addition Assignment	<code>i+=5;</code>	<code>i=i+5;</code>
<code>-=</code>	Subtraction Assignment	<code>i-=5;</code>	<code>i=i-5;</code>
<code>*=</code>	Multiplication Assignment	<code>i*=5;</code>	<code>i=i*5;</code>
<code>/=</code>	Division Assignment	<code>i/=5;</code>	<code>i=i/5;</code>

Example

After following two lines of code, what is the value of the variable `s`?

```
int s = 8;  
s = s - 5; (or equivalently s -= 5;)
```

Answer: After above two lines of code, the value of the variable `s` is 3.

Here is why: The first line declares an integer variable `s` and assigns integer 8 into the variable. Then, on the right side of the statement, we subtract 5 from the value of `s`, which is 8, to produce the difference 3 and then assigns 3 back to the variable `s`. The original data in `s`, which was 8, now is overwritten by the new integer data 3.

Increment and Decrement Operators

Following code is a specific augmented statement which we often use in looping statement (We will discuss looping statements in Chapter 3):

```
a=a+1;
```

Each time this line is executed, the value of the variable `a` will be increased by 1. For example, after following four lines of code, the value of the variable `a` will be changed from 5 to 8:

```
int a = 5;  
a = a + 1;  
a = a + 1;  
a = a + 1;
```

Line 1: declares an integer variable, and assign integer 5 to it

Line 2: on the right side of the statement, the value of `a` is 5, we add 1 to it, the result is 6 and 6 will be sent to the same variable `a` to update the value of `a` from 5 to 6.

Line 3: on the right side of the statement, the value of `a` is 6, we add 1 to it, the result is 7 and 7 will be sent to the same variable `a` to update the value of `a` from 6 to 7.

Line 4: on the right side of the statement, the value of `a` is 7, we add 1 to it, the result is 8 and 8 will be sent to the same variable `a` to update the value of `a` from 7 to 8.

Thus, the augmented statement $a = a + 1$; is executed three time here, and it changes the value of a from 5 to 8 (the value of a is increased by 3)

Since such augmented statement is used very often, we create an equivalent statement:

$a++$;

or

$++a$;

in place of the statement $a = a + 1$;

Both $a++$; or $++a$; are equivalent to $a = a + 1$; However, there is a difference between $a++$; and $++a$;

For $a++$; the value of a is used and then change to $a + 1$.

For $++a$; the value of a changes to $a + 1$, and then is being used.

It seems very difficult to understand what we are talking about here. Let's use a couple of examples to explain.

Example

Let's exam following three lines of the code in a program:

```
int a=5, b = 12;
cout<<b+(a++)<<endl;
cout<<a<<endl;
```

- Line 1: We declare two integer variables a and b and also send two integers 5 and 12 into those two variables
- Line 2: Since use $a++$ here, we use the value of a , which is 5, and add it to the value of b , which is 12, to produce a sum of 17 and display the sum on the screen with the *cout* statement. Then change the value of a from 5 to 6;
- Line 3: Display the value of a on the screen, which is 6

Now, let's take a look at another example.

Example

```
int a=5, b = 12;
cout<<b+(++a)<<endl;
cout<<a<<endl;
```

- Line 1: We declare two integer variables a and b and also send two integers 5 and 12 into those two variables
- Line 2: Since use $++a$ here, we use the value of a , which is 5, and add it to the value of b , which is 12, to produce a sum of 17 and display the sum on the screen with the *cout* statement. Then change the value of a from 5 to 6;
- Line 3: Display the value of a on the screen, which is 6

This page titled [2.3: Assignment Statements](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

2.4: Case Study- Sum of Numbers

Example

Write a program that only uses two variables. The program asks a user to enter 5 integers and then calculate and display the sum of those numbers.

We begin this program by declaring two integer variables *s* and *a*, the variable *s* is for storing the sum which need to be initialized to 0 and the variable *a* is for storing each number the user enter.

```
myProject (Global Scope)
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int s = 0, a;
7  }
```

Now, we will use a *cout* statement to prompt the user to enter five integers at line 7, and then use a pair of following two statements:

```
cin >> a;
```

```
s = s + a;
```

the *cin* statement assigns a user input to the variable *a*, and the augmented statement "*s=s+a*;" adds the newly entered number to the value of *s* which is the sum of the numbers previously entered. We will use this pair of statements five times to add five integers the user entered from line 8 to line 17:

```
myProject (Global Scope)
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int s = 0, a;
7      cout << "Please enter five integers:" << endl;
8      cin >> a;
9      s = s + a;
10     cin >> a;
11     s = s + a;
12     cin >> a;
13     s = s + a;
14     cin >> a;
15     s = s + a;
16     cin >> a;
17     s = s + a;
18 }
```

Finally, use a *cout* statement to display the result with a message that signifies the output at line 18:

```
myProject (Global Scope)
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int s = 0, a;
7      cout << "Please enter five integers:" << endl;
8      cin >> a;
9      s = s + a;
10     cin >> a;
11     s = s + a;
12     cin >> a;
13     s = s + a;
14     cin >> a;
15     s = s + a;
16     cin >> a;
17     s = s + a;
18     cout << "The sum of the numbers you entered is " << s << endl;
19     return 0;
20 }
```

The output of the program is shown as following:

```
Microsoft Visual Studio Debug Console
Please enter five integers:
11
34
10
22
5
The sum of the numbers you entered is 82
C:\Users\JJIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 1920) exited with code 0.
Press any key to close this window . . .
```

Exercise

Write a program that only uses two variables. The program asks a user to enter 8 integers and then calculate and display the sum of those numbers.

This page titled [2.4: Case Study- Sum of Numbers](#) is shared under a [not declared](#) license and was authored, remixed, and/or curated by [Jin He](#).

2.5: Numerical Operators

Following table provides a list of commonly used numerical operators used in C++ programs. We have used the multiplication operator '*' in the previous sample program. We may need to use the rest of them in the future assignments. Please take a moment to get familiar with them, especially the integer division operator '/' and the modulus division operator '%'. Please study the notes that follow the table.

Operator	Name	Example	Result
+	Addition	34+2	36
-	Subtraction	12-7	5
*	Multiplication	15*3	75
/	Division	12/4	3
%	Modulus	20%3	2

Note 1:

By the time we reach adulthood, we normally think of division as resulting in an answer that might have a fractional part (a floating-point data type). This type of division is known as floating-point division. However, division, when both operands are of the integer data type, may act differently, and is called: **integer division**. Consider:

11 / 2

We may think the result is 5.5. But because both operands are of the integer data type the evaluation of the expression (or answer) would be 2 with no fractional part (it gets thrown away). Again, this type of division is called **integer division**.

Thus, following assignment statement will assign integer 2 to the variable a instead of 2.5

```
a = 5/2;
```

If we **do** want to send 2.5 to the variable **a**, we will need to write the code as following:

```
a = 5/2.0;
```

Pay special attention to following example:

```
a = 7/10;
```

This statement will send 0 to the variable **a**. 7 divided by 10 is 0.7, but since both numerator and the denominator are integers, the operator '/' will only result in 0 instead of 0.7. Again, if we do want to keep the digit after the decimal point, we will write the statement as

```
a = 7.0/10;
```

Note 2:

'%' is a modulus division operator. This operator produces a remainder instead of quotient. Consider following example:

```
a=11%4;
```

We know when we divide 4 into 11

$$\begin{array}{r} 2 \text{ r } 3 \\ 4 \overline{) 11} \\ \underline{- 8} \\ 3 \end{array}$$

the remainder is 3, thus the statement assign 3 to the variable a.

We don't have this operator in algebra. So, please take time to get familiar with this special operator. Here, I present a few examples to help you understand and get familiar with this operator:

- if $a = 20\%3$, then a is 2 (3 goes into 20 six times and the remainder is 2)
- if $a = 3\%7$, then a is 3 (7 goes into 3 zero times and the remainder is 3)
- if $a = 4\%10$, then a is 4 (10 goes into 4 zero times and the remainder is 4)
- if $2837\%10$, then a is 7 (10 goes into 2837 two hundred eighty-three times and the remainder is 7)

In C++ programming, some of the numerical calculation will be done by using build-in functions. For example, we don't have a key such as x^y on the computer keyboard, thus we use following function to perform exponential calculation:

`pow()`

For example, if want to calculate 2^4 and assign the result to a variable m , we will write following code:

```
int m;
```

```
m=pow(2,4)
```

`pow( $a$ ,  $b$ )` is an exponential operation function (we will discuss more about C++ functions in the future). This function takes two inputs a and b , and calculates the value of an algebraic expression a^b . For example, **`pow(2.0, 3)`** will produce the third power of 2. **`pow(3, 5)`** will produce the fifth power of 3 which is 243.

This page titled [2.5: Numerical Operators](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

2.6: Coding a Mathematical Formula

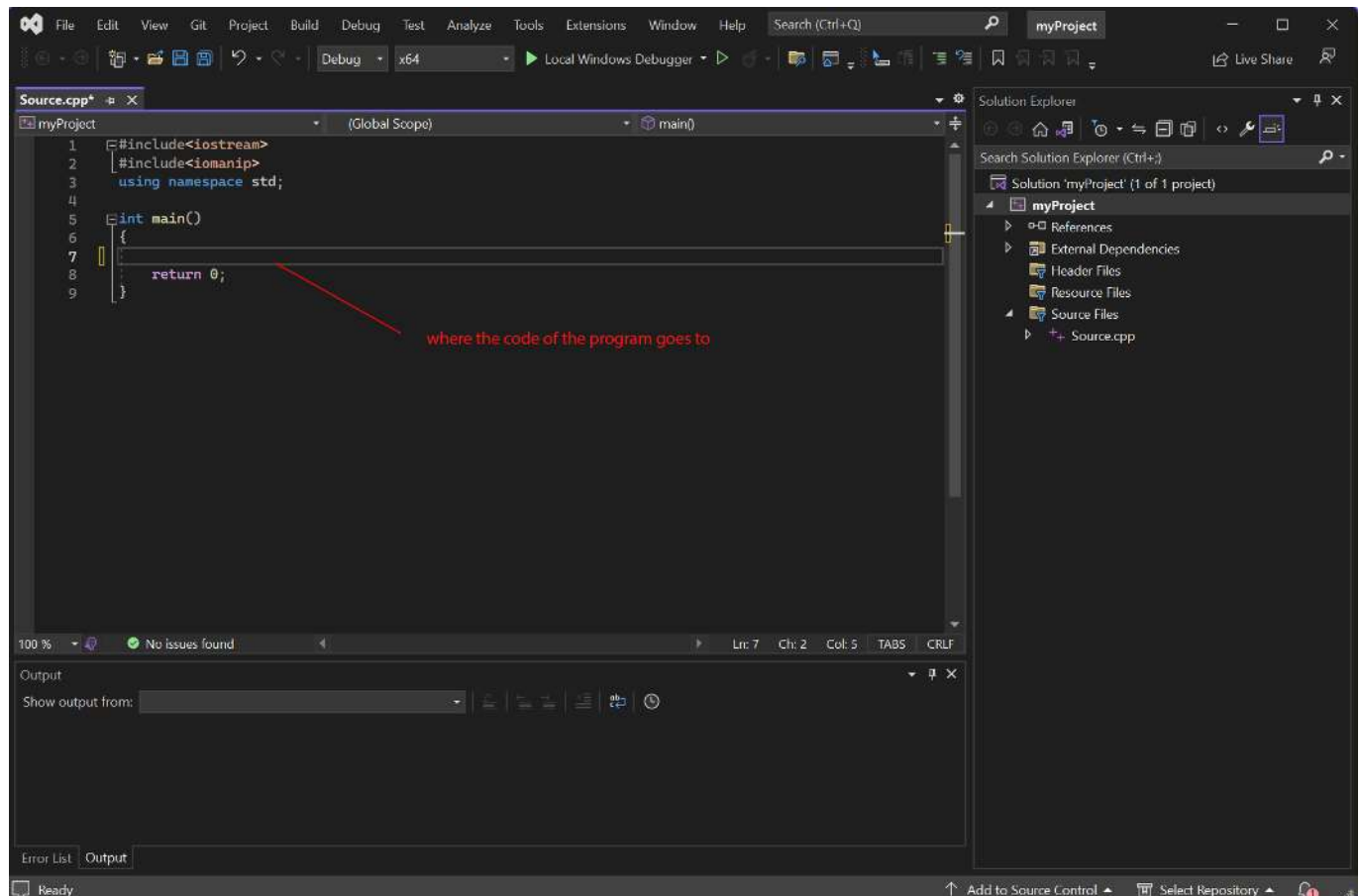
Example

Write a program that calculates the total amount money in a bank CD (Certificated Deposit) account based on the following formula:

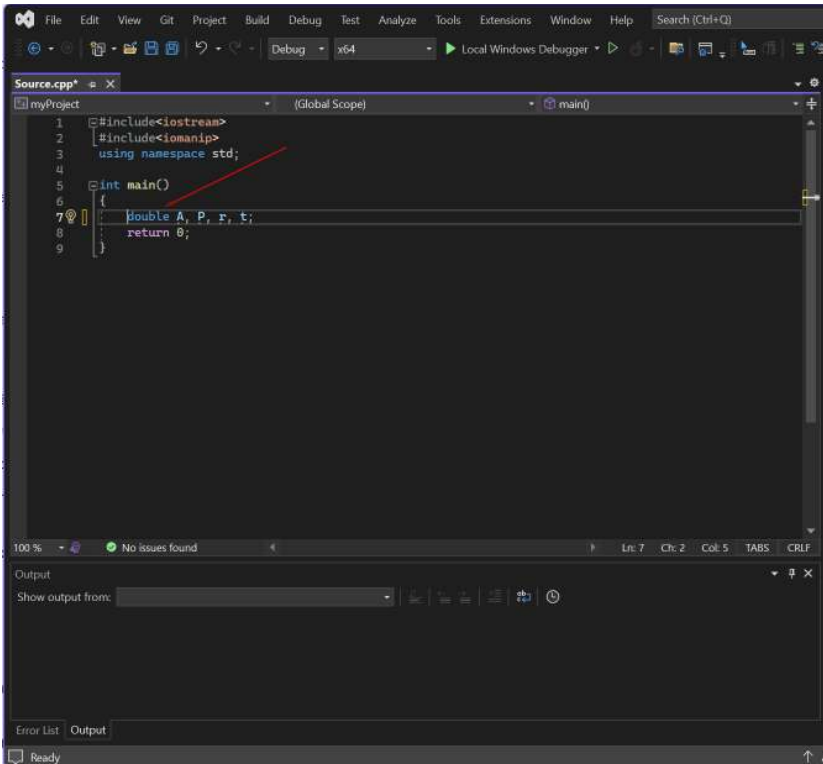
$$A = P(1 + \frac{r}{1200})^{12t} \quad (2.6.1)$$

This formula comes from algebra classes. There are four variables involved in this formula. All the variables on the left side of the equation are used for storing the data that come from a user, such as P , which is the principal amount (initial investment in a CD account), r , which is annual interest rate, and t , which is the term (The number of years). The variable on the right side of the equation, A is for storing the calculation result of the program. The challenge of this problem is how to code such complicated formula in a C++ program correctly.

Let's get started. First launch the app (Visual Studio or Xcode) and create an empty project. Within the project, open a source file. In the source file, put in the standard code for all the C++ programs and get ready for tackling the coding for this specific program:



We begin the coding from the inside of `main()`. The first line (line 7) is the code for declaring the variables we would like to use in the program: A , P , r , and t . Since the data this program is processing decimal in form, such as money, rate, and the number of years, we declare those variables by data type 'double' as following:

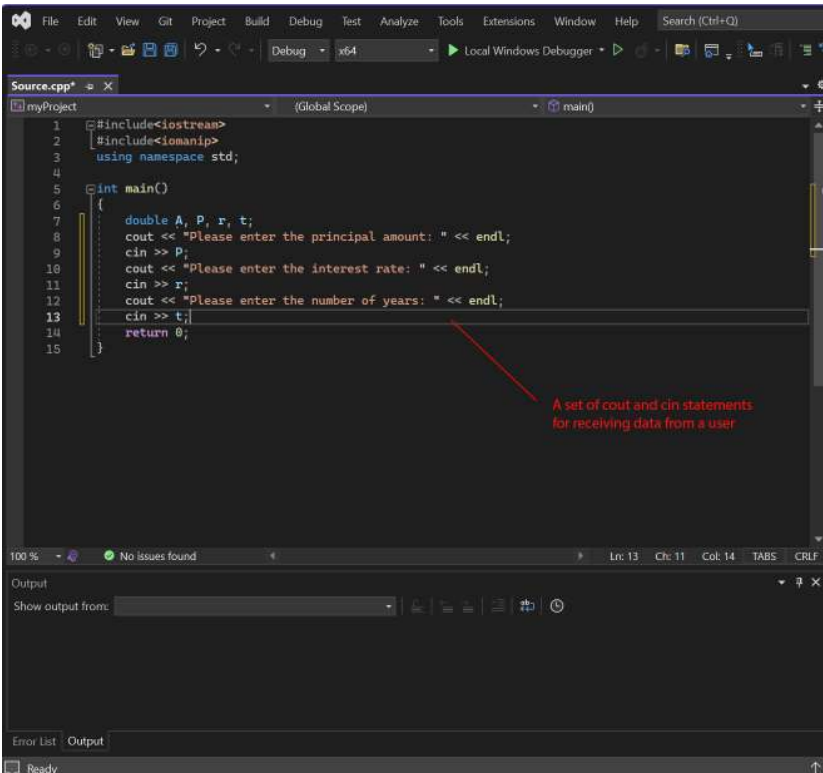


```

1  #include<iostream>
2  #include<iomanip>
3  using namespace std;
4
5  int main()
6  {
7      double A, P, r, t;
8      return 0;
9  }

```

The next thing is to use a set of *cout* and *cin* statements to receive data from a user:



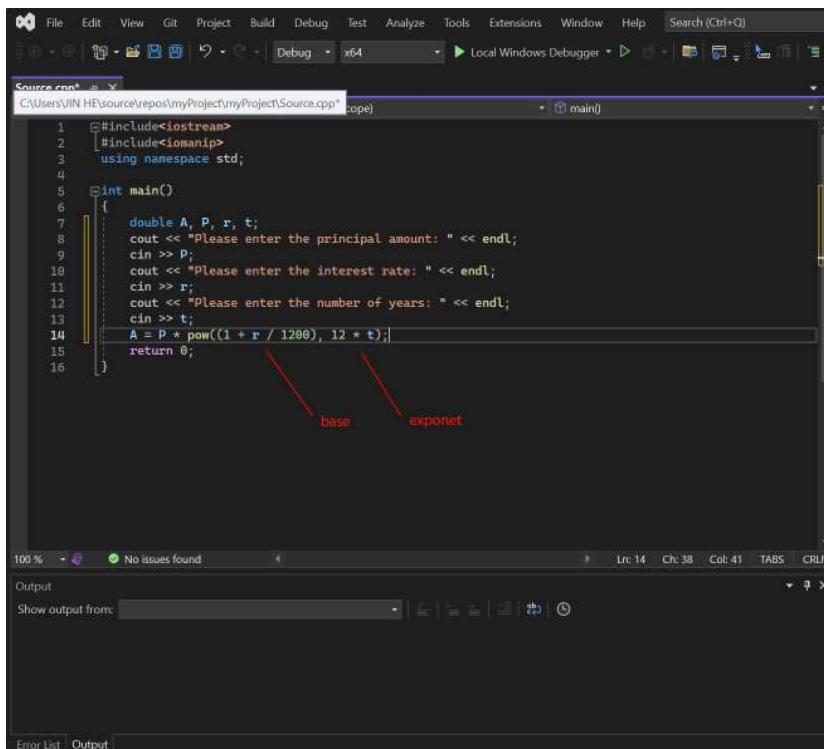
```

1  #include<iostream>
2  #include<iomanip>
3  using namespace std;
4
5  int main()
6  {
7      double A, P, r, t;
8      cout << "Please enter the principal amount: " << endl;
9      cin >> P;
10     cout << "Please enter the interest rate: " << endl;
11     cin >> r;
12     cout << "Please enter the number of years: " << endl;
13     cin >> t;
14     return 0;
15 }

```

A set of cout and cin statements for receiving data from a user.

As shown below, the next line (line 14) is the code of the formula. The program processes the data user entered based on the formula we coded and assign the result to the variable A. Here we use *pow()* for the exponential calculation:



```

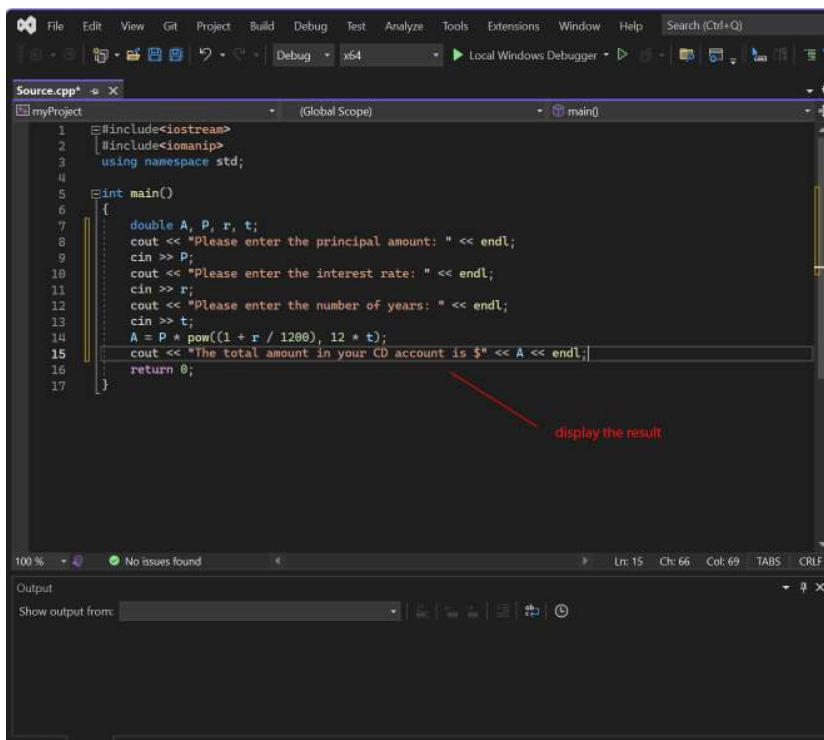
1  #include<iostream>
2  #include<iomanip>
3  using namespace std;
4
5  int main()
6  {
7      double A, P, r, t;
8      cout << "Please enter the principal amount: " << endl;
9      cin >> P;
10     cout << "Please enter the interest rate: " << endl;
11     cin >> r;
12     cout << "Please enter the number of years: " << endl;
13     cin >> t;
14     A = P * pow((1 + r / 1200), 12 * t);
15     return 0;
16 }

```

Annotations in the image:

- A red arrow points from the text "base" to the expression $(1 + r / 1200)$ in line 14.
- A red arrow points from the text "exponent" to the expression $12 * t$ in line 14.

As shown below, the code in line 15 displays the result on the output screen along with a message that signifies the number the program displays:



```

1  #include<iostream>
2  #include<iomanip>
3  using namespace std;
4
5  int main()
6  {
7      double A, P, r, t;
8      cout << "Please enter the principal amount: " << endl;
9      cin >> P;
10     cout << "Please enter the interest rate: " << endl;
11     cin >> r;
12     cout << "Please enter the number of years: " << endl;
13     cin >> t;
14     A = P * pow((1 + r / 1200), 12 * t);
15     cout << "The total amount in your CD account is $" << A << endl;
16     return 0;
17 }

```

Annotation in the image:

- A red arrow points from the text "display the result" to the output statement in line 15.

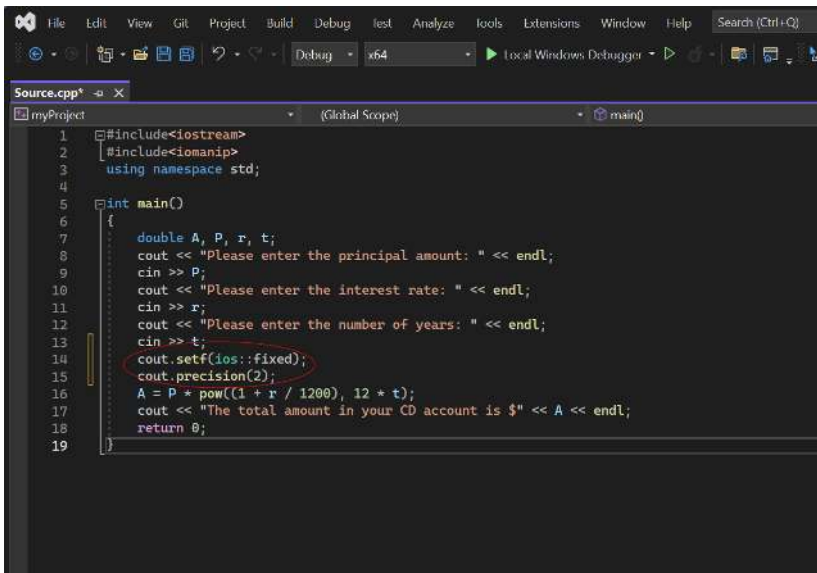
A sample run of this program will have an output screen shown as following:

```
Microsoft Visual Studio Debug Console
Please enter the principal amount:
5000
Please enter the interest rate:
1.7
Please enter the number of years:
3
The total amount in your CD account is $5261.42

C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 15732) exited with code 0.
Press any key to close this window . . .
```

If your program displays more than 2 digits after the decimal point, we can put following two lines of code before any one of the `cout` statement in the program

```
cout.setf(ios::fixed);
cout.precision(2);
```



```
Source.cpp* X
myProject (Global Scope) main()
1 #include<iostream>
2 #include<iomanip>
3 using namespace std;
4
5 int main()
6 {
7     double A, P, r, t;
8     cout << "Please enter the principal amount: " << endl;
9     cin >> P;
10    cout << "Please enter the interest rate: " << endl;
11    cin >> r;
12    cout << "Please enter the number of years: " << endl;
13    cin >> t;
14    cout.setf(ios::fixed);
15    cout.precision(2);
16    A = P * pow((1 + r / 1200), 12 * t);
17    cout << "The total amount in your CD account is $" << A << endl;
18    return 0;
19 }
```

This page titled [2.6: Coding a Mathematical Formula](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

2.7: C++ Examples

Overview

The following examples demonstrate data types, arithmetic operations, and input in C++.

Data Types

```
// This program demonstrates variables, literal constants, and data types.

#include <iostream>
#include <sstream>

using namespace std;

int main() {
    int i;
    double d;
    string s;
    bool b;

    i = 1234567890;
    d = 1.23456789012345;
    s = "string";
    b = true;
    cout << "Integer i = " << i << endl;
    cout << "Double d = " << d << endl;
    cout << "String s = " << s << endl;
    cout << "Boolean b = " << b << endl;
    return 0;
}
```

Output

```
Integer i = 1234567890
Real r = 1.23457
String s = string
Boolean b = 1
```

Discussion

Each code element represents:

- `//` begins a comment
- `#include <iostream>` includes standard input and output streams
- `#include <sstream>` includes standard string streams
- `using namespace std` allows reference to `string`, `cout`, and `endl` without writing `std::string`, `std::cout`, and `std::endl`.
- `int main()` begins the main function, which returns an integer value
- `{` begins a block of code

- `int i` defines an integer variable named `i`
- `;` ends each line of C++ code
- `double d` defines a double floating-point variable named `d`
- `string s` defines a string variable named `s`
- `bool b` defines a Boolean variable named `b`
- `i = , d = , s = , b =` assign literal values to the corresponding variables
- `cout` is standard output
- `<<` directs the next element to standard output
- `endl` ends the current line
- `return 0` returns the value 0 from `main`, indicating the main function completed successfully
- `}` ends a block of code

Arithmetic

```
// This program demonstrates arithmetic operations.

#include <iostream>
#include <sstream>

using namespace std;

int main() {
    int a;
    int b;

    a = 3;
    b = 2;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "a + b = " << a + b << endl;
    cout << "a - b = " << a - b << endl;
    cout << "a * b = " << a * b << endl;
    cout << "a / b = " << a / b << endl;
    cout << "a % b = " << a % b << endl;
    return 0;
}
```

Output

```
a = 3
b = 2
a + b = 5
a - b = 1
a * b = 6
a / b = 1
a % b = 5
```

Discussion

Each new code element represents:

- `+`, `-`, `*`, `/`, and `%` represent addition, subtraction, multiplication, division, and modulus, respectively.

Temperature

```
// This program converts an input Fahrenheit temperature to Celsius.
//
// References:
// https://www.mathsisfun.com/temperature-conversion.html
// https://en.wikibooks.org/wiki/C%2B%2B_Programming
#include <iostream>

using namespace std;

int main() {
    double fahrenheit;
    double celsius;

    cout << "Enter Fahrenheit temperature:" << endl;
    cin >> fahrenheit;

    celsius = (fahrenheit - 32) * 5 / 9;

    cout << fahrenheit << "° Fahrenheit is " << celsius << "° Celsius" << endl;

    return 0;
}
```

Output

```
Enter Fahrenheit temperature:
100
100° Fahrenheit is 37.7778° Celsius
```

Discussion

Each new code element represents:

- `cin >> fahrenheit` reads the next integer from standard input and assigns the value to the `fahrenheit` variable

References

- Wikiversity: Computer Programming

This page titled [2.7: C++ Examples](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

2.8: Case Study- Find the Sum of Numbers

Here, we study a sample program which performs a common programming task: Finding the Sum of Numbers.

Example

Write a program with only two variables. The program asks a user to enter 4 integers and produces the sum of those numbers.

Following is the code of the program, and we will walk you through it.

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int s=0,a;
7     cout<<"Please enter a number:"<<endl;
8     cin>>a;
9     s+=a;
10    cout<<"Please enter a number:"<<endl;
11    cin>>a;
12    s+=a;
13    cout<<"Please enter a number:"<<endl;
14    cin>>a;
15    s+=a;
16    cout<<"Please enter a number:"<<endl;
17    cin>>a;
18    s+=a;
19    cout<<"The sum of those numbers is "<<s<<endl;
20    return 0;
21 }
```

Line 6: We declare two variables *s* and *a*. We use variable *s* for storing the summation of the numbers a user enters and use variable *a* for storing the each number the user enters. The initial value of *s* should be set to 0.

Line 7 and Line 8: We use a set of *cout* and *cin* statements to ask a user to enter the first number from the console and save it in the variable *a*.

Line 9: This is an augmented statement. It adds the values of *s* and *a*, put the result back into the variable *s*. It is the same as $s=s+a$; At this point (Line 9) the variable *s* is holding the sum of the first number that a user entered.

Again, In Line 10 and Line 11 we use a set of *cout* and *cin* statements to ask the user to enter the second number from the console and save it in the variable *a*.

Line 12: Again, we use an augmented statement to add the values of *s* (which is the sum of the first number the user entered) and the value of *a* (which is the second number the user entered) and put the result back into the variable *s*. At this point, the value of *s* is the sum of the first and the second number the user entered.

Similarly, Line 13 - Line 15 adds the third number to the value of *s*, so the value of *s* is the sum of the first three numbers the user entered.

The code in Line 16 - Line 18, which is identical to the code in Line 13 - Line 15, adds the last number that the user entered to *s* to produce the sum of all 4 numbers the user entered.

Line 19: The *cout* statement displays the result on the output screen of the program.

Following is the screenshot of the output of the program:

```
Please enter a number:
2
Please enter a number:
5
Please enter a number:
1
Please enter a number:
4
The sum of those numbers is 12
Program ended with exit code: 0
```

Note: This program uses four groups of identical code, to produce the sum of 4 numbers a user enters. If want the program to produce a sum of 10 numbers a user enters, we can modify this program by copy and paste such code for additional 6 times. In later chapters, we will discuss how to use a looping statement to make the coding of this program more efficient.

This page titled [2.8: Case Study- Find the Sum of Numbers](#) is shared under a [not declared](#) license and was authored, remixed, and/or curated by [Jin He](#).

CHAPTER OVERVIEW

3: Chapter 3 Selection Statements

3.1: Relational Operators and Relational Statements

3.2: Selection Statement- if-statement

3.3: Case Study- Find the Largest Number

3.4: Case Study- Sorting Numbers

3.5: Selection Statement- if-else Statement

3.6: Radom Number and Logical Operators

3.7: Conditional Expression

3.8: Case Study- Practice Multiplication

3.9: Switch Statements and Menu Driven Programs

This page titled [3: Chapter 3 Selection Statements](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

3.1: Relational Operators and Relational Statements

In C++ programming, followings are commonly used relational operators:

Operator	Name	Example (radius=5)	Result
<	less than	radius<0	false
<=	less than or equal to	radius<=0	false
>	greater than	radius>0	true
>=	greater than or equal to	radius>=0	true
==	equal to	radius==5	true
!=	not equal to	radius!=5	false

We use above relational operators along with numbers, variables, expressions to form a relational statement such as followings:

$a > 5$

$i \leq 10$

$a \neq 0$

Each relational statement possesses a Boolean value, which is either true or false (1 or 0 equivalently) depending on the logical correctness of the statement. If a relational statement contains variables, the Boolean value of this statement usually depends on the value of the variable. For example, the value of the relational statement

$m > 5$

depends on the value of the variable m . If the value of m is 11, then the value of the statement is true or 1, if the value of m is 0, the value of this relational statement is false or 0.

In C++, a relational statement normally does not appear in a line of code all by itself such as

$a = 5 > 2;$

A relational statement is often used in a C++ program as a condition in a selection or a looping statement.

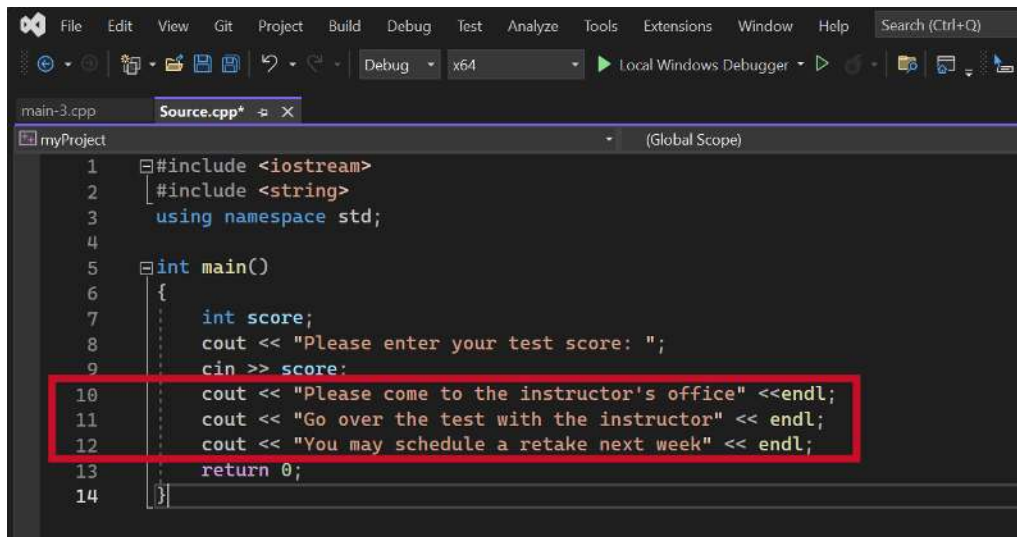
This page titled [3.1: Relational Operators and Relational Statements](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

3.2: Selection Statement- if-statement

Selection Statement: if-Statement

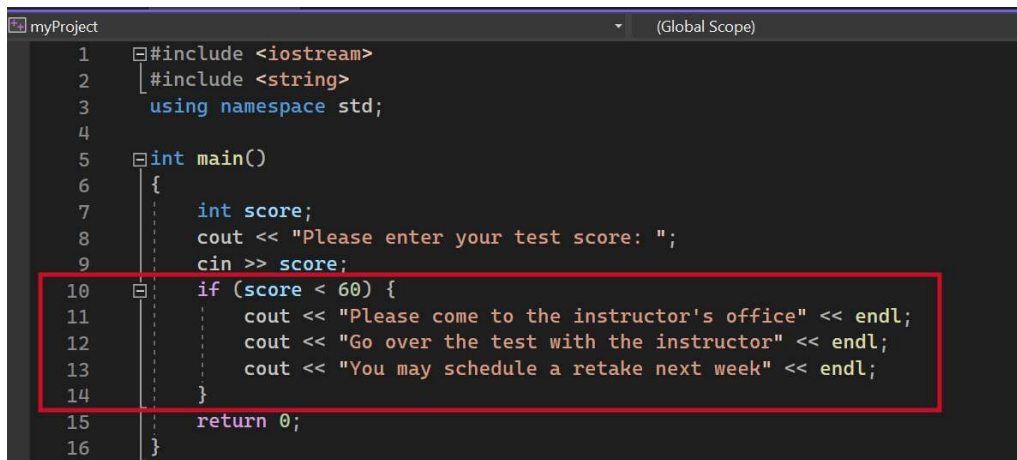
A C++ program consists of sequence of statements, and normally they will be executed one after the other sequentially. However, it is not always the case. Sometimes, a statement or a group of statements may or may not be executed depending on certain condition.

For example, following program contains a group of statements (line 10 to line 12):



```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     int score;
8     cout << "Please enter your test score: ";
9     cin >> score;
10    cout << "Please come to the instructor's office" << endl;
11    cout << "Go over the test with the instructor" << endl;
12    cout << "You may schedule a retake next week" << endl;
13    return 0;
14 }
```

If we run this program as the way it is, we will see the messages on the output screen regardless what the score we entered. If we only want the students with the test score below 60 to see those messages, we will use a selection statement in this program as shown below:



```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     int score;
8     cout << "Please enter your test score: ";
9     cin >> score;
10    if (score < 60) {
11        cout << "Please come to the instructor's office" << endl;
12        cout << "Go over the test with the instructor" << endl;
13        cout << "You may schedule a retake next week" << endl;
14    }
15    return 0;
16 }
```

In this program, we used an **"if" statement**, which is one of the **selection statements**, through line 10 to line 14. This selection statement starts at line 10 with a key word "if" followed by a relational statement `score<60` inside of a pair of parentheses. And a pair of braces { and } specifies a group of statements from line 11 to line 13. This group of statements will be executed if the the value of the relational statement `score<60` is true, which means the score a user enter is less than 60. Otherwise, nothing will be displayed on the output screen, which means the statements in line 11 to 13 will be skipped by the execution of the program.

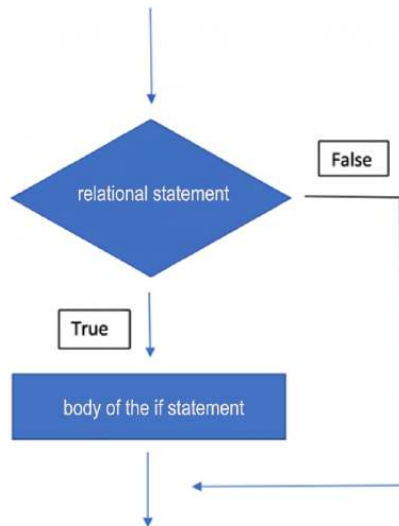
"if" Statement

The one we were using in above program is one of the selection statement in C++ language. Its structure is shown as following:

General structure of an "if" statement in C++:

```
if (relational statement) {  
    statement1;  
    statement2; <----- body of the if statement  
    .....  
    .....  
}  
Statement (s)
```

Following flow chart demonstrates the logic of an "if" statement:



When a computer executes an **if** statement in a program, it will first evaluate the value of the relational statement in the **if** statement, if the value of the value of the relational statement is true, then the computer will execute the statements in the body of the **if** statement, otherwise, the computer will skip the statements in the body of the **if** statement and move on the next statement that immediately follows the close brace } in the program.

Example

Write a program that asks a user to enter two integers and save them into two variables. Then switch the values of the two variables if the value of the first variable is greater than the value of the second variable, so when we display the values of those two variables in the order of first variable and the second variable, the numbers show up are always in an ascending order.

When we first see this programming problem, it seems intimidating. We just don't know what it is talking about and, of course, we don't know how to get started.

Don't worry, at least we can create an empty project and open a blank source file and that is a good start!

 preview_ila5

Read the question, the first statement of the question says: "Write a program that asks a user to enter two integers and save them into two variables". So, that means we will need two variables in this program. Since all the variables used in C++ program need to be declared, we should put a line of code that declares two integer variables. Yes, that is exactly what we need to do in this program:

 preview_cp61

We made some progress. It may not be perfect correct, but it is better than simply stare at screen and do nothing! In the first statement of the question, it also says: *"Ask a user to enter two integers"*, this tells us we should put a line of code like following:

```
cout<<"Please enter two integer:"<<endl;
```

because this line of code will display the message "Please enter two integers:" on the output screen and it can be seen by a user.

The first statement of the question also says to save the data a user entered into those variables we declared in line 6. Recall the statement we used in the example of calculating the area of a rectangle. There, we use

```
cin>>length;  
cin>>width;
```

to store the data user entered in the variables **length** and **width**. Now, we use similar statements to store the integer user entered into the variable **a** and **b**:

```
cin>>a;  
cin>>b;
```

Alternatively, those two statements can be written equivalently as following: (which uses a single line of the code instead of two)

```
cin>>a>>b;
```

we can use either one of them in our program to store the integers the user entered into the variable **a** and **b**:

 preview

We are making more progress!

Now, continue to read the question. Next phrase is *"Then switch the values of two variables"* Well, this may be a little bit difficult for you to grasp.

There are two data storage used in this program, one is represented by the variable **a** and the other is represented by the variable **b**, they are current used for storing the first and second integer entered by the user. We can't use following two lines of code for switching the values:

```
b = a;  
a = b;
```

The reason is when the program executes the first statement, the the value of **b** is replaced by the value of **a** and the original value of **b** is gone. The second statement just put the same data that was in **a** back to **a**. Thus, after above two statements, the values of both variables are the same. In other words, after above two lines of code, both **a** and **b** are holding the the data that user entered into **a**.

To switch the values between the variables **a** and **b**, we will need a third variable, say **t**. We will need to copy the data in **a** and save it in the variable **t** by following statement:

```
t = a;
```

then copy the value of **b** and put it into **a**:

```
a = b;
```

Now, the data in **a** is gone and replaced by a copy of the data in **b**. Since the data in the variable **a** has been saved in the variable **t**, we now copy the data in **t**, which is original data in **a**, and put it into the variable **b** by using following statement:

```
b = t;
```

So, by following three statements:

```
t = a; //copy the data in a to t
```

```
a = b; //copy the data in b to a
```

```
b = t; //copy the data in t to b
```

we successfully swapped the values of the variables **a** and **b**. However, when you read the question carefully, you will realize that we don't want to do the swapping for any integers user enter unconditionally. We only do the swapping when the first number a user entered is greater than the second number the user entered. That being said, we only need to execute above three statements under the condition: (**a > b**). Therefore, we will use an **if** statement to control the execution flow as following:

 preview_autm

The code shown above controls the execution of the switching under the condition (**a > b**) with an **if** statement beginning at line 10. Since we added an additional variable **t** in the program, we will need to go back to the line 7 and add it to the declaration statement. Now, the last thing we need to do in this program is to use a "cout" statement at line 15 to display the values of **a** and **b**.

 preview_kfge

Now, we have completed the coding of this program. When we run this program, for any two numbers a user entered, the program will always display those two numbers in an ascending order. Please test run this program on your computer.

Exercise

Write a program that asks a user to enter 6 integers. Among those integers, some of them are odd integers and some of them are even integers. The program uses only two variables to calculate and display the sum of those even integers user entered.

 [Download the solution](#)

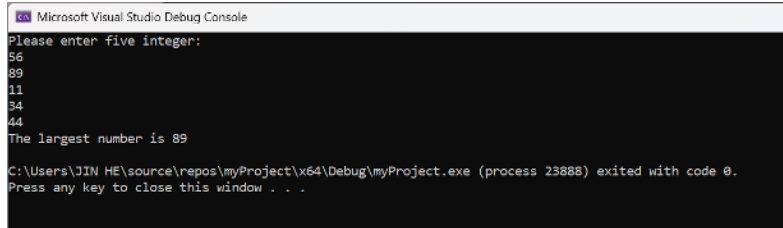
This page titled [3.2: Selection Statement- if-statement](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

3.3: Case Study- Find the Largest Number

Find the programming solution of following problem:

Use only two variables to write a C++ program that asks a user to enter 5 integers and then display the largest number that the user entered.

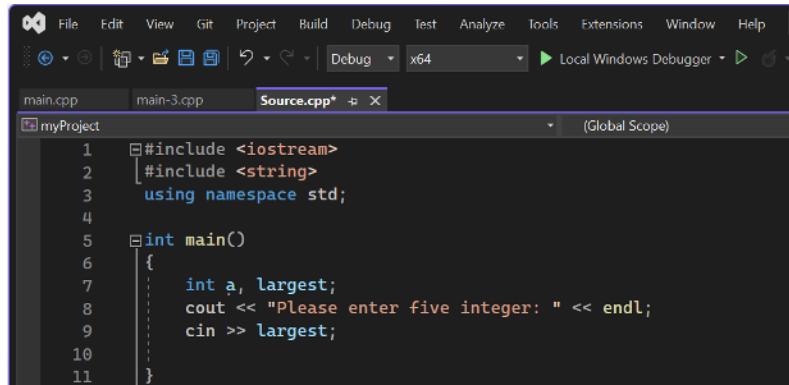
When we run this program, we should have following outputs:



```
Microsoft Visual Studio Debug Console
Please enter five integer:
56
89
11
34
44
The largest number is 89
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 23888) exited with code 0.
Press any key to close this window . . .
```

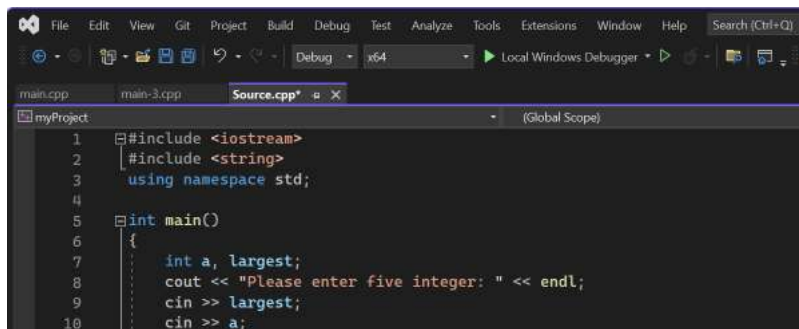
Now, let's code the program:

First, we declare two integer variables *a* and *largest* in line 7, and then, in line 8 and line 9, let a user to enter the first integer and assign this integer to the variable *largest* using a *cin* statement:



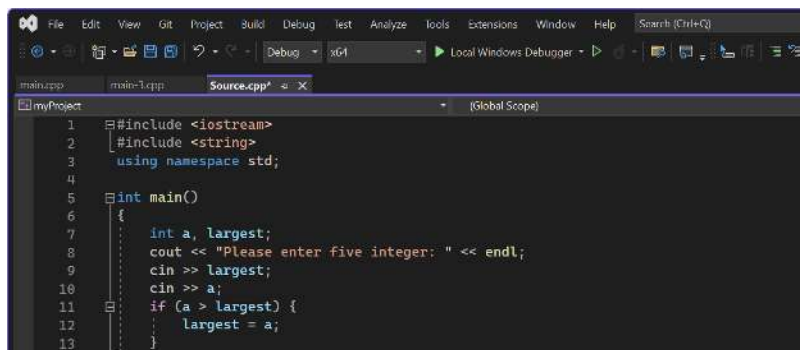
```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
main.cpp main-3.cpp Source.cpp* x64 Local Windows Debugger
myProject (Global Scope)
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     int a, largest;
8     cout << "Please enter five integer: " << endl;
9     cin >> largest;
10
11 }
```

Now, we use a *cin* statement, in line 10 to let the user to enter the second integer and assign it to the variable *a*:



```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q)
main.cpp main-3.cpp Source.cpp* x64 Local Windows Debugger
myProject (Global Scope)
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     int a, largest;
8     cout << "Please enter five integer: " << endl;
9     cin >> largest;
10    cin >> a;
```

This time, we compare the values of the variable *largest* and the variable *a*, if the value of the variable *a* is greater than the value of the variable *largest*, assign the value of *a* to the variable *largest*, otherwise, do nothing:

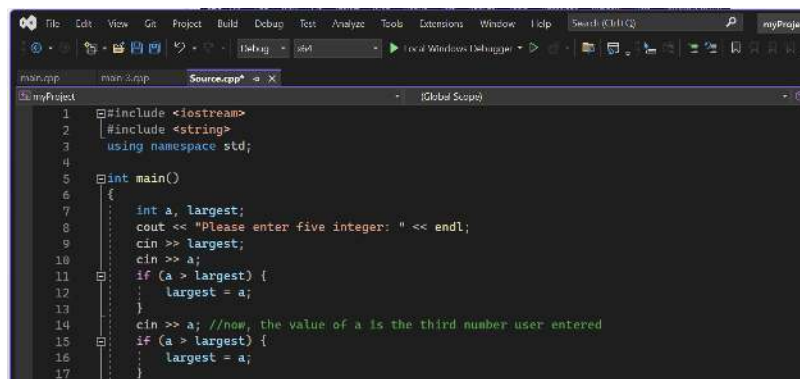


```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     int a, largest;
8     cout << "Please enter five integer: " << endl;
9     cin >> largest;
10    cin >> a;
11    if (a > largest) {
12        largest = a;
13    }

```

Now, in line 14, we use a `cin` again to let the user to enter the third integer and assign it to the variable `a`. Then, compare the third integer the user entered with the value of the variable `largest` by using another `if` statement. If the third number is greater than the value of the `largest`, replace the value of the variable `largest` by the third number, otherwise, do nothing:

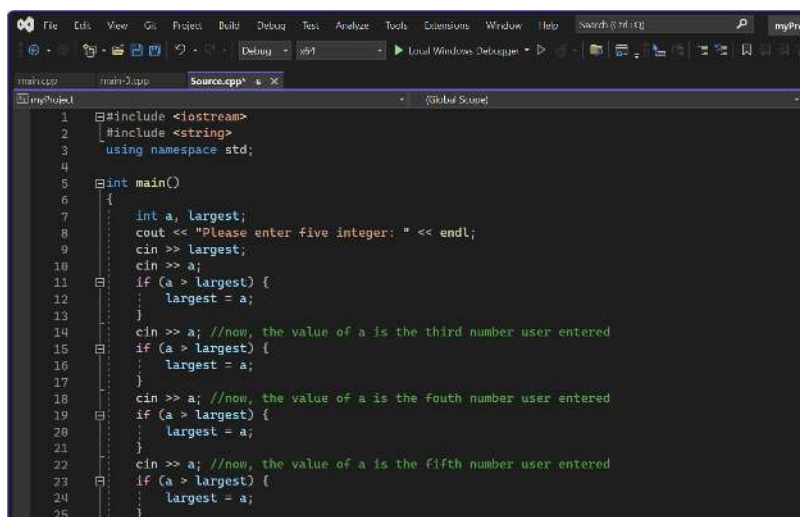


```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     int a, largest;
8     cout << "Please enter five integer: " << endl;
9     cin >> largest;
10    cin >> a;
11    if (a > largest) {
12        largest = a;
13    }
14    cin >> a; //now, the value of a is the third number user entered
15    if (a > largest) {
16        largest = a;
17    }

```

Please note: at this point, the variable `largest` is holding the largest of first three numbers user entered. Now, we repeat the code from line 10 to line 13 two more times:

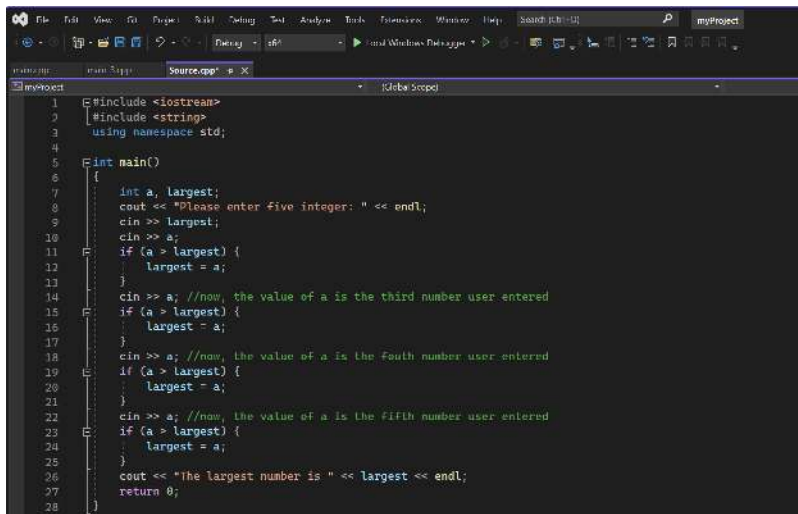


```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     int a, largest;
8     cout << "Please enter five integer: " << endl;
9     cin >> largest;
10    cin >> a;
11    if (a > largest) {
12        largest = a;
13    }
14    cin >> a; //now, the value of a is the third number user entered
15    if (a > largest) {
16        largest = a;
17    }
18    cin >> a; //now, the value of a is the fourth number user entered
19    if (a > largest) {
20        largest = a;
21    }
22    cin >> a; //now, the value of a is the fifth number user entered
23    if (a > largest) {
24        largest = a;
25    }

```

Because each time, we replace the value of the variable `largest` with a larger integer user entered, the variable `largest` will always hold the largest of the numbers user entered. After the program receives five integers, the value of the variable `largest` is the largest number user entered. Thus we use a `cout` statement to display it on the output screen:



```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     int a, largest;
8     cout << "Please enter five integer: " << endl;
9     cin >> largest;
10    cin >> a;
11    if (a > largest) {
12        largest = a;
13    }
14    cin >> a; //now, the value of a is the third number user entered
15    if (a > largest) {
16        largest = a;
17    }
18    cin >> a; //now, the value of a is the fourth number user entered
19    if (a > largest) {
20        largest = a;
21    }
22    cin >> a; //now, the value of a is the fifth number user entered
23    if (a > largest) {
24        largest = a;
25    }
26    cout << "The largest number is " << largest << endl;
27    return 0;
28 }
```

This page titled [3.3: Case Study- Find the Largest Number](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

3.4: Case Study- Sorting Numbers

Example

Write a program that asks a user to enter four integers and save them into four integer variables, and then re-arrange the values of those variables in a ascending order. (sorting numbers)

We begin this program by declaring four integer variables for storing user's input and one extra integer variable which we will be using later at line 7. We also use a set of *cout* and *cin* statements to allow a user to enter four integers and save them in the variable *a*, *b*, *c*, *d* at line 8 and 9:

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      int a, b, c, d, t;
8      cout << "Please enter four integers: " << endl;
9      cin >> a >> b >> c >> d;
10 }
```

Now, we use a set of three *if* statements to compare the values of each two adjacent variables. If the values of two variables are not in a ascending order, swap the values of those two variables, otherwise, do nothing:

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      int a, b, c, d, t;
8      cout << "Please enter four integers: " << endl;
9      cin >> a >> b >> c >> d;
10     if (a > b) {
11         t = a;
12         a = b;
13         b = t;
14     }
15     if (b > c) {
16         t = b;
17         b = c;
18         c = t;
19     }
20     if (c > d) {
21         t = c;
22         c = d;
23         d = t;
24     }
25 }
```

Each of those three *if* statements puts the larger number to the variable on the right. Thus, after those three *if* statements, the largest number is placed into the variable *d*, but the values of the first three variables are not necessarily in order.

Now, we will use two more *if* statements to compare the values of *a* and *b*, and *b* and *c*. If the values of two variables are not in a ascending order, swap the values of those two variables, otherwise, do nothing:

```

3   using namespace std;
4
5   int main()
6   {
7       int a, b, c, d, t;
8       cout << "Please enter four integers: " << endl;
9       cin >> a >> b >> c >> d;
10      if (a > b) {
11          t = a;
12          a = b;
13          b = t;
14      }
15      if (b > c) {
16          t = b;
17          b = c;
18          c = t;
19      }
20      if (c > d) {
21          t = c;
22          c = d;
23          d = t;
24      }
25      ////////////////
26      if (a > b) {
27          t = a;
28          a = b;
29          b = t;
30      }
31      if (b > c) {
32          t = b;
33          b = c;
34          c = t;
35      }
36      ////////////////
37  }

```

After those two if statements, the second largest number is placed into the variable *c*. However, the values of the variable *a* and *b* are not necessarily in an ascending order.

In the program shown in the following screen shot, We compare the values of the variables *a* and *b* with an if statement one more time (Line 37 - Line 41) to make sure the value of *a* is less than the value of *b*.

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      int a, b, c, d, t;
8      cout << "Please enter four integers: " << endl;
9      cin >> a >> b >> c >> d;
10     if (a > b) {
11         t = a;
12         a = b;
13         b = t;
14     }
15     if (b > c) {
16         t = b;
17         b = c;
18         c = t;
19     }
20     if (c > d) {
21         t = c;
22         c = d;
23         d = t;
24     }
25     ////////////////
26     if (a > b) {
27         t = a;
28         a = b;
29         b = t;
30     }
31     if (b > c) {
32         t = b;
33         b = c;
34         c = t;
35     }
36     ////////////////
37     if (a > b) {
38         t = a;
39         a = b;
40         b = t;
41     }
42     cout << "The values of the variable a, b, c, and d are now sorted in ascending order:" << endl;
43     cout << a << " " << b << " " << c << " " << d << endl;
44     return 0;
45 }

```

Following is output of a sample run of the program:

```
Microsoft Visual Studio Debug Console
Please enter four integers:
34 45 11 25
The values of the variable a, b, c, and d are now sorted in ascending order:
11 25 34 45
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 12764) exited with code 0.
Press any key to close this window . . .
```

Exercise

Write a program that asks a user to enter **six** integers and save them into four integer variables, and then re-arrange the values of those variables in a ascending order. (sorting numbers)

This page titled [3.4: Case Study- Sorting Numbers](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

3.5: Selection Statement- if-else Statement

In this section, we will discuss two-way selection statement: **if-else** statement.

Objective

- Understand the logic of the two-way selection statement
- Use if-else statement to solve simple programming problems

If we only need to decide whether to execute or to skip a statement or a group of statements, we normally use **if** statement we have discussed in the previous lecture. But if we need to make a selection between two statements or two groups of statements to execute, then a two-way selection statement: **if-else** statement will be more convenient.

Structure of an if-else Statement

```
if(relational Statement){
```

```
Statement 1;
```

```
Statement 2;
```

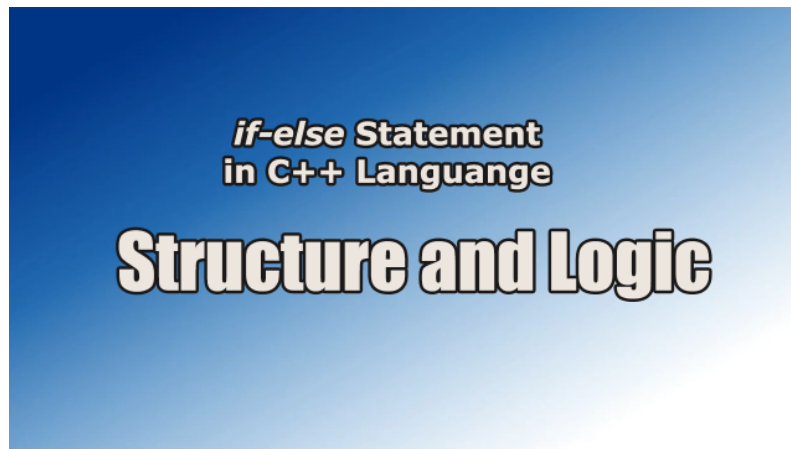
```
}else{
```

```
Statement 3;
```

```
Statement 4;
```

```
}
```

Following video explains the logic of such if-else statement:



Now, let's discuss how to use if-else statement in a C++ program.

Example

A company made a pay raise to its employees based on the following schedule:

Use formula 1:

new salary = old salary + old salary x 15%

if the old salary is less than \$5000.

Use formula 2:

new salary = old salary + old salary x 10%

if the old salary is \$5000 or more

If the old salary is \$5000 or more:

Write a program to allow an employee to check his/her new salary.

Following video demonstrates the coding of such program by using an *if-else* statement:



This page titled [3.5: Selection Statement- if-else Statement](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

3.6: Radom Number and Logical Operators

Random Number Function

In C++, the function **rand()** produces a random number between 0 and 32,767. If we want to get a random number between 0 and 9, we use **rand()%10**. Similarly, if we want to get a random number between 0 and 99, we use **rand()%100**. In general, the code **rand()%n** in a C++ program will generate a random number between 0 and n-1.

If you would like to random get a number between 1 and 10, you can use the code

```
rand()%10 + 1
```

If you want to random generate a number between 65 and 90, you can use the code

```
rand()%26+65
```

We will use a random number function in a C++ program discussed in Section 3.8: [Case Study: Practice Multiplication](#)

Logical Operators

Sometimes, in the C++ programming, we need to use relational statements that are formed by connecting two or more simple relational statements. The symbols or the words we use in C++ programming for connecting relational statements are called **logical operators**. The commonly used logical operators are

- **or**, this is also called a union operator. Alternatively, it can also be coded as || (two vertical bars with no space in between)
- **and**, this is also called an intersection operator. Alternatively, it can also be coded as && (shift key + 7, two of those with no space in between)
- **not**, this is also called a negation operator. Alternative, it can also be coded as ! (an exclamation mark)

In a C++ program, we use those logical operators to connect one or more simple relational statements to form complex relational statements.

Examples

- **(x > 5) or (x < -2)**
This is a relational statement formed by connecting two simple relational statements (x>5) and (x<-2) using connectives **or**. The truth value of this compound relational statement depends on the value of the variable x. If x is a number that is either greater than 5 or less than -2, then the relational statement is true. Otherwise, it is false. In general, if two relational statements are connected with the logical operator '**or**', then the compound relational statement is true as long as one of the two relational statements is true, and the compound relational statement is false only when both relational statements are false.
- **!(5==7)**
This is a relational statement formed by using the negation operator, which is true. Since 5==7 is false and the negation operator '!' negates the Boolean value of 5==7 (the word "negate" means switching the truth value of a relational statement)
- **!(a==5)**
This is also a relational statement formed by using the negation operator. The truth value of this relational statement depends on the value of the variable a. It is true if the value of the variable **a** is 5 and it will be false when the variable a actually equals 5.
- **(a>=200) and (a<=800)**
This is a relational statement formed by using the connective **and** connecting two simple relational statements (a>=200) and (a<=800). The truth value of this compound relational statement depends on the value of the variable **a**. If the value of the variable **a** falls between 200 and 800 with 200 and 800 included, then it is true. Otherwise, it is false. In general, if two relational statements are connected with the logical operator '**and**', then the compound relational statement is false as long as one of the two relational statements is false, and the compound relational statement is true only when both relational statements are true.

This page titled [3.6: Radom Number and Logical Operators](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

3.7: Conditional Expression

A **conditional expression** is a special type of selection statements.

Structure (syntax) of a conditional expression:

```
(relational statement)? x : y
```

Logic of this conditional expression:

If the relational statement is true, select the statement x, otherwise, select the statement y.

Example

```
cout << (x > y ? x : y) << endl;
```

Above C++ statement will check the truth value of the relational statement $x > y$. If it is true, then execute the code `cout<<x<<endl;` If it is false, then execute the code `cout<<y<<endl;`

Example

```
t = (x > y ? x : y);
```

Above C++ statement will check the truth value of the relational statement $x > y$. If it is true, then execute the code `t=x;` If it is false, then execute the code `t=y;`

Example

```
cout << (x%2==0 ? "x is even" : "x is odd");
```

Above C++ statement will check the truth value of the relational statement $x\%2==0$. If it is true, then execute the code `cout<<"x is even"<<endl;` If it is false, then execute the code `cout<<x is odd<<endl;`

Example

```
score =score +(score<=60? 10 : 5);
```

Above C++ statement will check the truth value of the relational statement $\text{score} \leq 60$. If it is true, then execute the code `score=score+10;` If it is false, then execute the code `score=score+5;`

This page titled [3.7: Conditional Expression](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

3.8: Case Study- Practice Multiplication

Example

Write a program that allow a user to practice on a multiplication problem. The program should meet following requirements:

1. Present a multiplication program with two random numbers between 1 and 100
2. Ask the user to provide the correct answer
3. Check the answer. If the answer is correct, provide the user another question and check the answer and give the feedback. If the answer is incorrect, provide the correct answer.

The output of this program should be (the 1st correct and the 2nd wrong):

```
Please provide the correct answer of the following question:
```

```
+-----+
|      17 x  8 = ?      |
+-----+
```

```
136
```

```
Great! It is correct. Please try another problem:
```

```
Please provide the correct answer of the following question:
```

```
+-----+
|      24 x 26 = ?      |
+-----+
```

```
1234
```

```
Sorry, the answer is incorrect and the correct answer is 624
```

The output of this program should be (the 1st wrong):

```
Please provide the correct answer of the following question:
```

```
+-----+
|      50 x 53 = ?      |
+-----+
```

```
1234
```

```
Sorry, the answer is incorrect and the correct answer is 2650
```

```
C:\Users\JIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 32080) exited with code 0.
Press any key to close this window . . .
```

The output of the program should be (the 1st correct, and the 2nd correct):

```
Please provide the correct answer of the following question:
```

```

+-----+
|      95 x  58 = ?      |
+-----+

5510
Great! It is correct. Please try another problem:
Please provide the correct answer of the following question:

+-----+
|      39 x  35 = ?      |
+-----+

1365
Great! It is correct. You are done with the practice.

C:\Users\JJIN HE\source\repos\myProject\x64\Debug\myProject.exe (process 32920) exited with code 0.
Press any key to close this window . . .

```

We will be using a random function and a nested if-else statement (an if-else statement inside of an if-else statement). Let's begin.

The first part of this program is coded as following:

```

myProject (Global Scope)
1  #include <iostream>
2  #include<iomanip>
3  using namespace std;
4
5  int main()
6  {
7      int a, b, product;
8      a = rand() % 100 + 1;
9      b = rand() % 100 + 1;
10     cout << "Please provide the correct answer of the following question:" << endl;
11     cout << endl;
12     cout << setw(10)<<"+"<<"-----> " << endl;
13     cout << setw(10) << "|" << setw(10) << a << setw(2) << "x" << setw(4) << b << setw(4) << "= ?" << setw(10) << "|" << endl;
14     cout << setw(10)<<"+"<<"----->+" << endl;
15     cout << endl;

```

Line 7: declares three integer variables

Line 8 - Line 9: use random function to generate two random numbers between 1 and 100 and assign them into variable *a* and *b*

Line 10 - Line 15: display the question. Use *setw()* to correctly align the question and the box

Now, we add the rest of the code of this program as following:

```

1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  int main()
6  {
7      int a, b, product;
8      a = rand() % 100 + 1;
9      b = rand() % 100 + 1;
10     cout << "Please provide the correct answer of the following question:" << endl;
11     cout << endl;
12     cout << setw(10) << "+" << "-----" << "+" << endl;
13     cout << setw(10) << "|" << setw(10) << a << setw(2) << "x" << setw(4) << b << setw(4) << "=" << "?" << setw(10) << "|" << endl;
14     cout << setw(10) << "+" << "-----" << "+" << endl;
15     cout << endl;
16     cin >> product;
17     if (product == a * b) {
18         cout << "Great! It is correct. Please try another problem:" << endl;
19         a = rand() % 100 + 1;
20         b = rand() % 100 + 1;
21         cout << "Please provide the correct answer of the following question:" << endl;
22         cout << endl;
23         cout << setw(10) << "+" << "-----" << "+" << endl;
24         cout << setw(10) << "|" << setw(10) << a << setw(2) << "x" << setw(4) << b << setw(4) << "=" << "?" << setw(10) << "|" << endl;
25         cout << setw(10) << "+" << "-----" << "+" << endl;
26         cout << endl;
27         cin >> product;
28         if (product == a * b) {
29             cout << "Great! It is correct. You are done with the practice." << endl;
30         }
31         else {
32             cout << "Sorry, the answer is incorrect and the correct answer is " << a * b << endl;
33         }
34     }
35     else {
36         cout << "Sorry, the answer is incorrect and the correct answer is " << a * b << endl;
37     }
38     return 0;
39 }

```

Line 16: a cin statement to take the user input and assign to the variable *product*

Line 17 - Line 37: It is an *if-else* statement which checks the user input of the first question

Line 18 - Line 32 (the code inside of red box): This is the body of *if*, which are code the program will be executing when the value of the relational statement "*product==a*b*" is true, which means the answer user entered is correct). This block of code provides the feedback (Line 18) and produce another question (Line 19 - Line 26). Inside of this block of the code, we use another *if-else* statement (Line 28 - Line 33) to check the answer of the second question and provide the feedback. Line 29 will be executed if the user enter the correct answer for the second question and Line 32 will be executed if the answer to the second question is wrong.

Line 36: This is the body of the *else* of the first *if-else* statement, which gives the user feed back and provide the correct answer of the first question. Line 36 will only be executed when the relational statement in Line 17 (which means the user did not get the correct answer of the first question)

In this program, we are using an *if-else* statement inside of the another *if-else* statement. In an *if-else* statement, if the body of *if* or the body of *else* also contains an *if-else* statement, we call such statement a **nested if-else** statement.

This page titled [3.8: Case Study- Practice Multiplication](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

3.9: Switch Statements and Menu Driven Programs

In this section, we will discuss another selection statement often used in a C++ program: The **switch statement**.

Statement Structure:

```
switch(a)
{
case 1:
statements;
case 2:
statements;
case 3:
statements;
-----
default:
statements;
};
```

Logic of the *switch* statement:

The statement will compare the value of the variable *a* with the integer that immediately follows the word *case*. If we have a match, then all the statements under the first word *case* as well as the statements under the rest of the cases including the statement under the *default* will be executed. If the value of the variable *a* does not equal the integer that follows the word *case*, the switch statement skips all the statements under the first case and compares the value of the variable *a* with the integer that follows the second word *case*. Again, if we have a match, then all the statements under the second *case* and the statement under the rest of cases will be executed including the statements under the *default*. If we don't have a match, the switch statement will skip the statements under the second *case*, and proceed to compare the value of *a* with the integer that follows the third word *case*, and so on. If the value of *a* does not match any integer that follows the word *case*, then the *switch* statement will skip the statements under all the cases and only execute the statements under the *default*.

Note:

1. There is no limit to the number of the cases that we can use in a *switch* statement
2. The integer used in a *switch* statement doesn't have to be started at 1 and doesn't need to be consecutive.
3. The value of the variable of *a* doesn't have to be integer, it can be a character. However, If the value of *a* is a character, then we have to use characters in place of integers in the switch statement.

Following is an example of using switch statement in a C++ program.

Example

Use "switch" statement to write a menu driven program that allow a user to selectively perform one of the following tasks:

1. Convert the temperature from Celsius to Fahrenheit
2. Display Next Month Appointments
3. Find My Grade

Additional Information:

1. The formula of converting the temperature from the Celsius to Fahrenheit is

$$\text{Fahrenheit} = (9/5.0) * \text{celsius} + 32$$

When a user selects this option, the program should ask user to enter the temperature in Celsius and then display the same temperature in Fahrenheit.

2. If a user opt to display his/her next month appointments, the program should display a list of his/her next month appointments

3. If a user opt to find his or her class grade, the program should first ask the user to enter four test scores, then calculate the sum.

Divide the sum by 4 and use the following scale to display the user's grade:

A: 90 or above, B: 80 - 89 C: 70 - 79, D: 60 - 69, F: 59 or below

Followings are the program's sample runs.

First run:

```
Please select one of the following:
+=====+
| 1. Celsius to Fahrenheit          |
| 2. Display Next Months Appointment |
| 3. Find My Grade                  |
| 4. Do Nothing                     |
+=====+
1
Enter a Degree in Celsius: 16
86.4 fahrenheit
Program ended with exit code: 0
```

The second run:

```
Please select one of the following:
+=====+
| 1. Celsius to Fahrenheit          |
| 2. Display Next Months Appointment |
| 3. Find My Grade                  |
| 4. Do Nothing                     |
+=====+
2
+=====+
|           April Appointments           |
+-----+
| 04/10      Work Meeting      3:00pm    |
| 04/15      Dentist Appointment 12:45pm  |
| 04/23      Doctor Appointment 11:15am   |
| 04/30      Work Meeting      4:15pm    |
+-----+
Program ended with exit code: 0
```

The third run:

```
Please select one of the following:
+=====+
| 1. Celsius to Fahrenheit          |
| 2. Display Next Months Appointment |
| 3. Find My Grade                  |
| 4. Do Nothing                     |
+=====+
3
Please enter 4 test scores (0-100) 78 64 91 82
C
Program ended with exit code: 0
```

The fourth run:

```
Please select one of the following:
+=====+
| 1. Celsius to Fahrenheit          |
| 2. Display Next Months Appointment |
| 3. Find My Grade                  |
| 4. Do Nothing                     |
+=====+
4
The User Chooses Not To Do Anything
Program ended with exit code: 0
```

Now, we begin the coding of this program:

First, we use a set of *cout* statements to display a menu to the user of this program as following:

```
Project5 (Global Scope)
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      cout << "===== " << endl;
7      cout << "  1. Celsius to Fahrenheit" << endl;
8      cout << "  2. Display Next Month Appointment" << endl;
9      cout << "  3. Find My Grade" << endl;
10     cout << "  4. Do Nothing" << endl;
11     cout << "===== " << endl;
12 }
```

Then we declare an integer variable *option* at the beginning of the *main()*, and use a set of *cout* and *cin* statements to prompt the user to provide an input:

```
Project5 (Global Scope)
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int option;
7      cout << "===== " << endl;
8      cout << "  1. Celsius to Fahrenheit" << endl;
9      cout << "  2. Display Next Month Appointment" << endl;
10     cout << "  3. Find My Grade" << endl;
11     cout << "  4. Do Nothing" << endl;
12     cout << "===== " << endl;
13     cout << "Please select a task you would like to perform or enter 4 if you don't want to do anything" << endl;
14     cin >> option;
15 }
```

Now, we set up a switch statement as following:

```
Project5 (Global Scope)
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int option;
7      cout << "===== " << endl;
8      cout << "  1. Celsius to Fahrenheit" << endl;
9      cout << "  2. Display Next Month Appointment" << endl;
10     cout << "  3. Find My Grade" << endl;
11     cout << "  4. Do Nothing" << endl;
12     cout << "===== " << endl;
13     cout << "Please select a task you would like to perform or enter 4 if you don't want to do anything" << endl;
14     cin >> option;
15     switch (option)
16     {
17     case 1:
18     :
19     case 2:
20     :
21     case 3:
22     :
23     case 4:
24     :
25     default:
26     :
27     };
28
29 }
```

We put the code that performs each task in the spaces after each case. For example, we put the code that performs temperature conversion in the space after the *case 1*:

```

1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int option;
7     cout << "===== " << endl;
8     cout << " 1. Celsius to Fahrenheit" << endl;
9     cout << " 2. Display Next Month Appointment" << endl;
10    cout << " 3. Find My Grade" << endl;
11    cout << " 4. Do Nothing" << endl;
12    cout << "===== " << endl;
13    cout << "Please select a task you would like to perform or enter 4 if you don't want to do anything" << endl;
14    cin >> option;
15    switch (option)
16    {
17        case 1:
18            double cel, fah;
19            cout << "Please enter the temperature in Celsius:" << endl;
20            cin >> cel;
21            fah = 9 / 5.0 * cel + 32;
22            cout << "The temperature is " << fah << "degree in Fahrenheit" << endl;
23            break;
24        case 2:
25
26        case 3:
27
28        case 4:
29
30    default:
31
32    };

```

If the value of the variable *option* is 1, then all the code under the *case 1* will be executed. By default, after all the statements in *case 1* are executed, all the statements under the rest of the cases will also be executed including the statements under the *default*. Thus, we put a *break* statement at line 23 to prevent the statements under the rest of the cases from being executed.

A screenshot of the completed program is shown as following:

```

Source.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int option;
7     cout << "===== " << endl;
8     cout << " 1. Celsius to Fahrenheit" << endl;
9     cout << " 2. Display Next Month Appointment" << endl;
10    cout << " 3. Find My Grade" << endl;
11    cout << " 4. Do Nothing" << endl;
12    cout << "===== " << endl;
13    cout << "Please select a task you would like to perform or enter 4 if you don't want to do anything" << endl;
14
15    std::cout << "std::cout\n";
16
17    case 1:
18        double cel, fah;
19        cout << "Enter a Degree in Celsius:" << endl;
20        cin >> cel;
21        fah = 9 / 5.0 * cel + 32;
22        cout << fah << "fahrenheit" << endl;
23        break;
24    case 2:
25        cout << "===== " << endl;
26        cout << "          April Appointments" << endl;
27        cout << "===== " << endl;
28        cout << " 04/10   Work Meeting      3:00 PM" << endl;
29        cout << " 04/15   Dentist Appointment  12:45 PM" << endl;
30        cout << " 04/23   Doctor Appointment   11:15 AM" << endl;
31        cout << " 04/30   Work Meeting         4:15 PM" << endl;
32        cout << "===== " << endl;
33        break;
34    case 3:
35        int a, b, c, d, s;
36        cout << "Please enter 4 test scores (0-100): ";
37        cin >> a >> b >> c >> d;
38        s = (a + b + c + d) / 4;
39        if (s >= 90 and s <= 100)
40            cout << "A" << endl;
41        if (s >= 80 and s <= 89)
42            cout << "B" << endl;
43        if (s >= 70 and s <= 79)
44            cout << "C" << endl;
45        if (s >= 60 and s <= 69)
46            cout << "D" << endl;
47        if (s >= 50 and s <= 59)
48            cout << "F" << endl;
49        break;
50    case 4:
51        cout << "The User Chooses Not To Do Anything" << endl;
52        break;
53    default:
54        cout << "Invalid input, please enter 1, 2, 3, or 4" << endl;
55    };
56    return 0;
57

```

If a user enter 3, then the program will skip the code under the *case 1* and *case 2* and executes the code under the *case 3*. After all the statements under the *case 3* are executed, the *break* statement at line 49 will prevent the code under the *case 4* and the statements under the *default* from being executed.

If a user enter an invalid input (an input other than 1, 2, 3, or 4), then only the statements under the *default* will be executed.

This page titled [3.9: Switch Statements and Menu Driven Programs](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

CHAPTER OVERVIEW

4: Chapter 4 Characters and Strings

[4.1: Character Data Type and ASCII Code](#)

[4.2: String Data Type](#)

This page titled [4: Chapter 4 Characters and Strings](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

4.1: Character Data Type and ASCII Code

In this section we will discuss how to store and process characters in a C++ program.

In a C++ program, if we want to use a variable *a* to store a single character, we will need following statement to declare such variable:

```
char a;
```

where 'char' is a character data type which indicates variable *a* is a variable that can be used for storing a single character.

In a C++ program, if we want to assign a single character into a variable, say assign character M into variable *a*, we may use following statement:

```
a = 'M';
```

Note: the character that you want to assign to a character variable must be placed within a pair of single quotation marks.

ASCII Code

Characters are not limited to upper case or lower case letters. A digit, a punctuation mark, even a blank space is also considered as a character. Characters also include those special symbols such as \$, %, &, +, * and so on.

Computers use a specific integer to represent each character, such integer is called ASCII code (American Standard Code for Information Interchange). Following table gives the ASCII codes of some of the characters we often use in a C++ program:

ASCII Code for Commonly Used Characters:

Characters	ASCII Code
'0' to '9'	48 to 57
'A' to 'Z'	64 to 90
'a' to 'z'	97 to 122

The ASCII code for '\$' is 36, '#' is 35, and the ASCII code for a blank space is 32 and so on. If you would like to find out the ASCII code for a specific character, say '%', this is what you can do:

1. Declare an integer variable

```
int a;
```

2. Put in a line of code as following:

```
a='%';
```

3. Using following *cout* statement to display the ASCII code of '%' on the screen:

```
cout<<a<<endl;
```

Why do we need to know the ASCII code for each character? Well, there are a lot of coding in computer programming that require the knowledge of ASCII code. Here we present a simple example that shows the usage of ASCII code in a C++ program.

Example

Write a program that provide a user a recommendation of five possible passwords that meet following requirements:

1. Exactly four characters
2. The first character must be a digit (0 - 9)
3. The second character must be a lower case letter
4. The third character must be an upper case letter
5. The last character must be a lower case letter
6. Five passwords have to be random generated instead of being hard coded into your program

Write a program that provide a user a recommendation of five possible passwords that meet following requirements:

1. Exactly four characters
2. The first character must be a digit (0 - 9)
3. The second character must be a lower case letter
4. The third character must be an upper case letter
5. The last character must be a lower case letter
6. Five passwords have to be random generated instead of being hard coded into your program

Before we begin the coding process, let's review random function and modulus division.

In C++, the function **rand()** produces a random number between 0 and 32,767. If we want to get a random number between 0 and 9, uses `rand()%10` because modulus division '%' returns the remainder. Any number is divided by 10, the remainder can only be one of the numbers between 0 and 9. For example, 12 is divided by 10, the remainder is 2; 15 is divided by 10, the remainder is 5; 19 is divided by 10, the remainder is 9; 20 is divided by 10, the remainder is 0; 21 is divided by 10, the remainder is 1, 22 is divided by 10, the remainder is 2 and so on. You can see, as long as the divisor is 10 the remainder can not be anything else except one of the numbers between 0 and 9.

Similarly, if we want to get a random number between 0 and 99, uses `rand()%100`.

Particularly, if we would like to have a random number between 0 and 25, we use `rand()%26`.

How about getting a lower case letter randomly? From above table, a lower case letter is represented by an integer between 97 and 122 with 97 and 122 included. To get a random lower case letter,

Step 1: declare character variable:

```
char a;
```

Step 2: use `rand()%26` to get a random number between 0 and 25

Step 3: get a random number between 97 and 122 by using `97+rand()%26`. According above table, this number can be used as an ASCII code to represent a lower case letter.

Step 4: store the number we got in Step 3 in the variable we declared in Step 1

We can get Step 2, 3, and 4 done with a single line of the code shown as following:

```
a=97+rand()%26;
```

We can display this random lower case letter by following `cout` statement:

```
cout<<a<<endl;
```

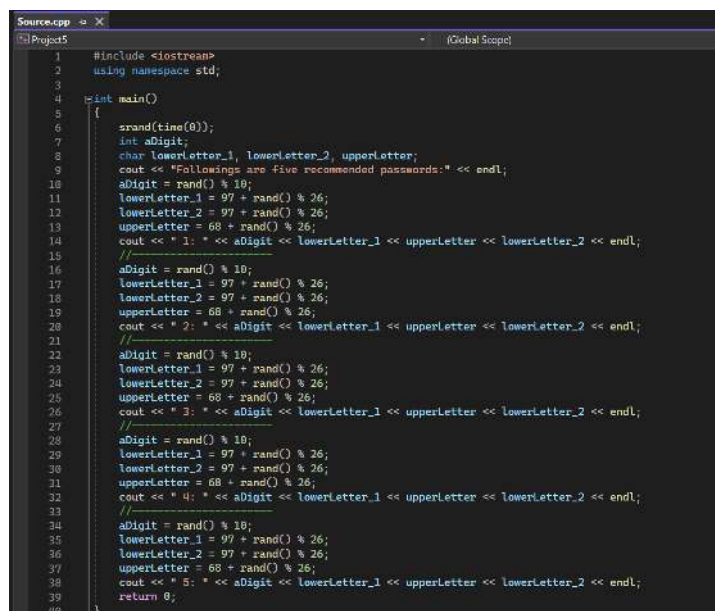
Similarly, we can get a random upper case letter and display it by using following statements:

```
char b;
```

```
b=65+rand()%26;
```

```
cout<<b<<endl;
```

Following is a screenshot of the completed program:



```
Source.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     srand(time(0));
7     int aDigit;
8     char lowerLetter_1, lowerLetter_2, upperLetter;
9     cout << "Following are five recommended passwords:" << endl;
10    aDigit = rand() % 10;
11    lowerLetter_1 = 97 + rand() % 26;
12    lowerLetter_2 = 97 + rand() % 26;
13    upperLetter = 68 + rand() % 26;
14    cout << "1: " << aDigit << lowerLetter_1 << upperLetter << lowerLetter_2 << endl;
15    //
16    aDigit = rand() % 10;
17    lowerLetter_1 = 97 + rand() % 26;
18    lowerLetter_2 = 97 + rand() % 26;
19    upperLetter = 68 + rand() % 26;
20    cout << "2: " << aDigit << lowerLetter_1 << upperLetter << lowerLetter_2 << endl;
21    //
22    aDigit = rand() % 10;
23    lowerLetter_1 = 97 + rand() % 26;
24    lowerLetter_2 = 97 + rand() % 26;
25    upperLetter = 68 + rand() % 26;
26    cout << "3: " << aDigit << lowerLetter_1 << upperLetter << lowerLetter_2 << endl;
27    //
28    aDigit = rand() % 10;
29    lowerLetter_1 = 97 + rand() % 26;
30    lowerLetter_2 = 97 + rand() % 26;
31    upperLetter = 68 + rand() % 26;
32    cout << "4: " << aDigit << lowerLetter_1 << upperLetter << lowerLetter_2 << endl;
33    //
34    aDigit = rand() % 10;
35    lowerLetter_1 = 97 + rand() % 26;
36    lowerLetter_2 = 97 + rand() % 26;
37    upperLetter = 68 + rand() % 26;
38    cout << "5: " << aDigit << lowerLetter_1 << upperLetter << lowerLetter_2 << endl;
39    return 0;
40 }
```

In C++ programming, there are some special characters called **Escape Sequences**. Following table shows some of those special characters, their notations, and the ASCII codes:

Escape Sequences

What is an escape sequence? In C++ programming, most of characters, such as digits, letters, and some of the special character, such as \$, &, #, can be embedded in a string of characters. For example, we can use following statements in a C++ program with no issues:

```
cout<<"The cost of the textbook is $250"<<endl;
cout<<"###Schedule###"<<endl;
```

However, not all the characters can be embedded in a string as those mentioned above. For example, the character `\` (backslash) can't be directly embedded in a string as we did with `#`:

The following statement will cause an error in your program:

```
cout<<"\\Schedule\\"<<endl;
```

In order to embed a backslash in a string, we need to use `\\` (put a backslash in front of it) instead of putting a backslash directly in the string. Thus, if we want to put three backslashes in front of "Schedule" and three backslashes after, we need to code above `cout` statement as following:

```
cout<<"\\Schedule\\"<<endl;
```

If you are not getting it, let's look another example.

Suppose we would like to use a `cout` statement to display following message:

```
cout<<"The word 'Textbook' means a book that is used for teaching a class"<<endl;
```

we will see an error message and the program can't be executed. The reason is the double quote " is one of the character that can't be directly embedded in a string. To correctly code above statement (to put a double quote in a string) we need to put a backslash in front of double quote as shown below:

```
cout<<"The word \"Textbook\" means a book that is used for teaching a class"<<endl;
```

In above two examples, \ is called an escape sequence of \, and \" is called the an escape sequence of ". To be more specific, if we want to put a backslash or a double quote in a string which we would like to display on the output screen with a *cout* statement, we can't put them in the string directly, we need to put their escape sequences in the string instead.

Some of the characters we use in C++ programming don't even have keys on the keyboard to allow us to put them into a string. For example, a linefeed character, which is used for indicating the end of line and the beginning of the next line. In C++, we use \n for such character. Following statement

```
cout<<"Fresno City College\nClovis Community College\nReedley College"<<endl;
```

will display

```
Fresno City College
Clovis Community College
Reedley College
```

and

```
cout<<"Fresno City College Clovis Community College Reedley College"<<endl;
```

will display

```
Fresno City College Clovis Community College Reedley College
```

Commonly Used Escape Sequences in C++

Escape Sequence	Name	ASCII Code
\b	Backspace	8
\t	Tab	9
\n	Linefeed	10
\r	Carriage Return	13
\	Backslash	92
\"	Double Quote	34

Casting between *char* and *int*

If we assign a character into an integer variable, the integer variable will receive the ASCII code of the character, for example

```
int a = 'A';
cout<< a <<endl;
```

above statements will display the ASCII code of letter A, which is 65, on the screen.

On the other hand, if we assign an integer into a character variable, the character variable will receive a character with the integer as its ASCII code, for example

```
char a = 90;
cout << a <<endl;
```

above statement will display **Z** on the screen.

Also, we can use following statement to see ASCII code of a given character, say, **&**

```
cout << static_cast<int>('&') <<endl;
```

or we can use following statement to see a character for a given ASCII code, say, **98**

```
cout << static_cast<char>(98) << endl;
```

above **static_cast** statement will convert a variable or a data from one data type to another data type.

For example, following statement will convert an integer data **15** from *int* to *double*:

```
static_cast<double>(15)
```

Character Functions

There is a list of functions that are working on characters, and we call them character functions.

Function	Description
isdigit(ch)	Returns true if the specified character is a digit
isalpha(ch)	Return true if the specified character is a letter
isalnum(ch)	Return true if the specified character is letter or a digit
islower(ch)	Return true if the specified character is a lower case letter
isupper(ch)	Return true if the specified character is an upper case letter
isspace(ch)	Return true if the specified character is a white-space character
tolower(ch)	Return the lowercase of the specified character
toupper(ch)	Return the uppercase of the specified character

We may not use those functions very often, but whenever we need them, we will know where to find them.

This page titled [4.1: Character Data Type and ASCII Code](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

4.2: String Data Type

When we need a variable in our program to store an **integer**, we use following statement to declare such variable

```
int a;
```

If we need a variable in our program to store a single **character**, we use following statement to declare such variable

```
char a;
```

Now, if we need a variable in our program to store a sequence of characters (a **string**), we use following statement to declare such variable

```
string a;
```

Note: We will need to add following line at the top of your program: (right below the line: `#include<iostream>`)

```
#include<string>
```

We can assign a string into such variable with following statement:

```
a = "Do Homework";
```

We can also let a user to enter a string to such variable with a `cin` statement

```
cin>>a;
```

Note: If the string contains a blank space such as "Do Homework", then above `cin` statement will only save the first word into variable `a`, and stop input at blank space. In other words, if we have following statements in our program:

```
string a;
```

```
cin>>a;
```

```
cout<<a<<endl;
```

if a user enter `Do Homework`, then the variable `a` only holds the string "Do" and "Homework" will not be saved into the variable `a`. So, when you display `a`, you only see `Do` on the screen.

If we want to save the entire line that a user enter (multiple words separated by white-spaces, we will need to use following statement

```
getline(cin, a);
```

Thus, if replace above code with

```
string a;
```

```
getline(cin, a);
```

```
cout<<a<<endl;
```

If a user enter `Do Homework`, the entire string will be saved into the variable `a` including a white-space. Now `cout<<a<<endl;` will display two words on the screen with a white-space in between. Please try it in your computer.

Once a string is saved into a string variable, we can use a `at()` function to access each individual character in the string.

For example

```
string a = "Do Homework";
```

```
cout << a.at(5)<<endl;
```

above statements will display the sixth character in the string "Do Homework". Similarly, `cout<<a.at(0)<<endl;` will display the first character in the string.

or you can simply use `a[5]` or `a[0]` in place of `a.at(5)` or `a.at(0)`:

```
string a = "Do Homework";  
cout << a[5]<<endl;
```

This page titled [4.2: String Data Type](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

CHAPTER OVERVIEW

5: Chapter 5 Looping Statements

5.1: The Structure and The Logic of a while Loop Statement

5.2: Case Study- Find The Largest Number

5.3: Case Study- Find The Sum

5.4: Sentinel Control Loop and do-while Loop Statement

5.5: for Loop Statement

5.6: Nested Loop Statements

5.7: Case Study- Display a Rectangle

This page titled [5: Chapter 5 Looping Statements](#) is shared under a [not declared](#) license and was authored, remixed, and/or curated by [Jin He](#).

5.1: The Structure and The Logic of a while Loop Statement

In this section, we begin the study on one of the important C++ statements: loop statement. When a programmer wants to repeatedly execute a statement or a group of statements, often, she/he would like to use a loop statement.

There are many different loop statements in C++ language, here we will begin with *while* loop statement.

We first introduce the structure of the *while* loop statement and explain its logic.

The Structure (syntax) of a *while* Loop Statement:

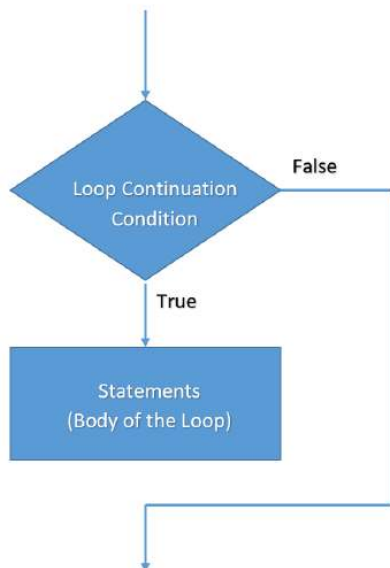
```
while ( looping-continuation-condition) {
```

```
Statement (s); // those statements are referred as the body of the loop
```

```
}
```

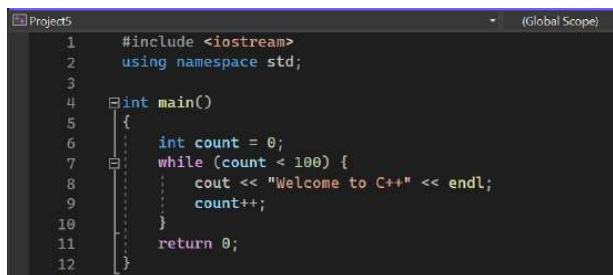
looping-continuation-condition is normally a relational statement which produces a Boolean value, either true (1) or false (0)

Following chart demonstrates the logic of a *while* loop statement:



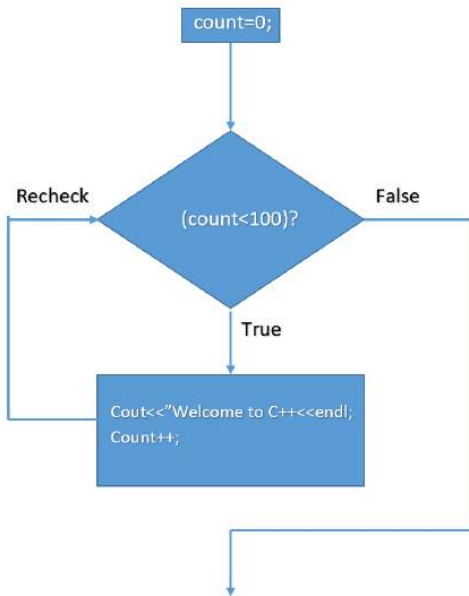
When a computer sees a while loop statement, it will first evaluate the loop-continuation-condition (normally it is a relational statement). If it is a true statement, then the computer will execute all the statements in the body of the loop. After the computer complete the last statement in the body of the loop, it will go back and re-evaluate the loop-continuation-condition. If the loop-continuation-condition is still a true statement, then execute all the statements in the body of the loop again. The computer will continue such loop until the loop-continuation-condition becomes a false statement, then computer will skip those statements in the body of the loop and move on to the next statement that follows the while looping statement.

Following is an example:



```
Project5 (Global Scope)
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int count = 0;
7      while (count < 100) {
8          cout << "Welcome to C++" << endl;
9          count++;
10     }
11     return 0;
12 }
```

Following chart shows the logic of above while loop statement:



In this case, the loop-continuation -condition is $(count < 100)$ and the true or false of this relational statement depends on the value of the variable *count*. Each time when the computer executes the statements in the body of the looping statement, the value of the variable *count* will be increased by 1. This looping statement will execute the statements in the body of the loop over and over again until the value of the variable *count* becomes 100, then the loop-continuation-condition becomes a false statement, and it breaks the loop. In this example, we use this loop statement execute the statement

```
cout<<"Welcome to C++!\n"<<endl;
```

100 time, which displays the message "Welcome to C++!" on the output screen 100 times. Here the variable *count* is called the looping variable.

If we want to execute certain statement(s) a certain number of times, we can always use above *while* loop statement to do so.

For example, if we want to execute the statement "*a=a+5;*" 11 times, we can use following statement to do so:

```
int a=0;
int count=0;
while (count<11){
a=a+5;
count++;
}
```

This while loop executes the statement "*a=a+5;*" 11 times and each time the statement add 5 to the value of *a* and put the result back to *a*. Thus, after this loop statement, the value of *a* is 55.

If we want to execute following three statements 5 times:

```
cout<<"Please enter an integer:"<<endl;
cin>>a;
s+=a;
```

we can place them in a while loop statement as following:

```
Project5 (Global Scope)
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int s = 0, a, i = 0;
7      while (i < 5) {
8          cout << "Please enter an integer:" << endl;
9          cin >> a;
10         s += a;
11         i++;
12     }
13     return 0;
14 }
```

In this program, the variable *i* is the looping variable. Each iteration of the while loop, the value of *i* is incremented by 1. This loop statement will repeatedly execute the statements in line 8, 9, 10 and line 11 until the value of *i* becomes 5. When the value of *i* becomes 5, the looping continuation condition, *i*<5, is a false statement, the loop stops. Thus, the line 8, 9 10 will be executed when *i*=0, 1, 2, 3, 4 with a total of five times.

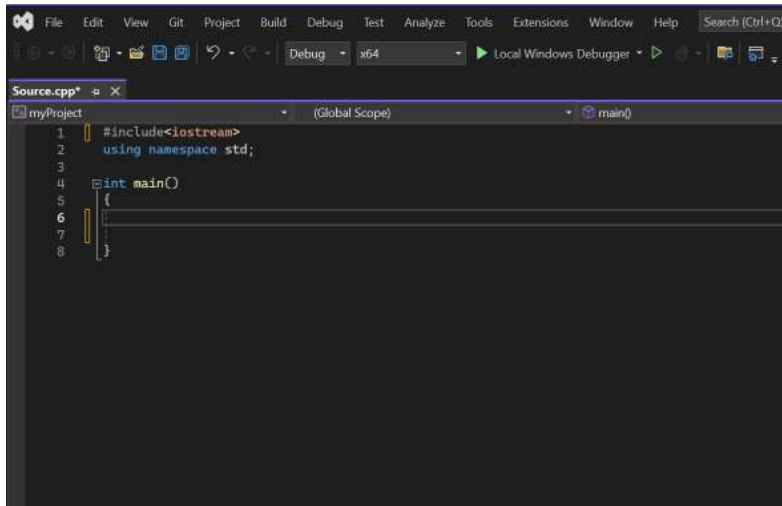
This page titled [5.1: The Structure and The Logic of a while Loop Statement](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

5.2: Case Study- Find The Largest Number

Find the largest number from a set of numbers is a common task in computer programming. A set of numbers, such as test scores from SAT or household incomes from a metropolitan area or average rainfalls in the last twenty years in Fresno County, can be used as inputs to a C++ program and the program will display the largest number in the set.

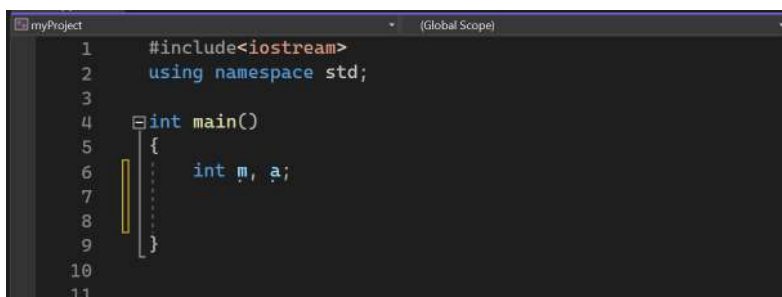
In this section, we will discuss how to code such program.

Again, we will need to create an empty project and open a blank source file. Then type in the code of initial setting:



```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6
7
8 }
```

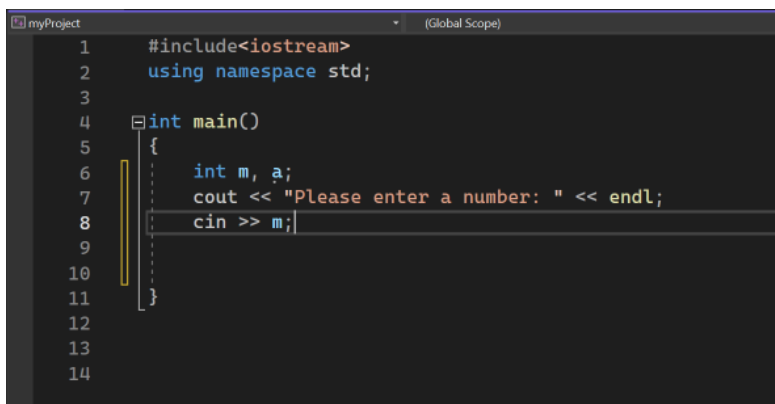
Declare two variables:



```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int m, a;
7
8
9
10
11 }
```

We use the variable *a* for storing the number a user enter, and use the variable *m* for holding the largest number among the numbers the user has entered.

If a user only enter one number, then this number will be the largest number, so we put this number into *m* directly:



```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int m, a;
7     cout << "Please enter a number: " << endl;
8     cin >> m;
9
10
11
12
13
14 }
```

If a user has two numbers to enter to the program, then we will put the second number into the variable *a* by using a set of *cout* and *cin* statements:

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int m, a;
7      cout << "Please enter a number: " << endl;
8      cin >> m;
9      cout << "Please enter a number:" << endl;
10     cin >> a;
11
12
13
14 }
```

When we put the second number in the variable *a*, we are not sure if the second number is greater than the first one. If it is, we should put it in the variable *m*, otherwise, we can just keep the first number in the variable *m*. We use an *if* statement to accomplish this:

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int m, a;
7      cout << "Please enter a number: " << endl;
8      cin >> m;
9      cout << "Please enter a number:" << endl;
10     cin >> a;
11     if (a > m)
12         m = a;
13
14
15 }
```

At this point, the variable *m* is holding the largest number the user entered so far. If a user just have two numbers to enter, then the we can stop the coding and put in a *cout* statement to display the value of the variable *m*, which is largest number from the set of the numbers that the user entered.

If a user has three numbers to enter to the program, then we will use another pair of *cout* and *cin* statements to place the third number user entered in the variable *a* as we did with the second number user entered:

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int m, a;
7      cout << "Please enter a number: " << endl;
8      cin >> m;
9      //----------------the second number-----
10     cout << "Please enter a number:" << endl;
11     cin >> a;
12     if (a > m)
13         m = a;
14     //----------------the third number-----
15     cout << "Please enter a number:" << endl;
16     cin >> a;
17     if (a > m)
18         m = a;
19 }
```

We are not sure if the third number user entered is the largest number user entered so far. We will need to compare. But we don't need to compare the third number with both the first and the second number user entered, all we need to do is to compare the third number with the value of the variable m because m is holding the largest among the first two numbers user entered.

If the third number is greater than the value of m , the third number will be the largest, then we will put third number into m , otherwise, we simply leave the m as it is, so the m is still holding the largest number among the three numbers user entered so far. Again, we will use an *if* statement to accomplish this:

 preview_id25

If a user has only three numbers to enter the program, then we can use a *cout* statement to display the value of the variable m at this point and we are done with program.

If a user has another number to enter, the fourth number, then we will repeat line 15 through line 18:

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int m, a;
7      cout << "Please enter a number: " << endl;
8      cin >> m;
9      //-----the second number-----
10     cout << "Please enter a number:" << endl;
11     cin >> a;
12     if (a > m)
13         m = a;
14     //-----the third number-----
15     cout << "Please enter a number:" << endl;
16     cin >> a;
17     if (a > m)
18         m = a;
19     //-----the fourth number-----
20     cout << "Please enter a number:" << endl;
21     cin >> a;
22     if (a > m)
23         m = a;
24 }
```

The logic will be the same: If the fourth number is greater than the value of the variable m , then the fourth number will be the largest number user entered so far, we will use it to replace the number in m , otherwise we will leave the value of the variable m unchanged. Either way, m is holding the largest number user entered.

Assume the user just need to enter four numbers, then we are done with input, and we can use a *cout* statement to display the value of m at line 24 and finish the coding as shown below:

 preview

If we take a close look at the code of this program, we realize this program actually runs a group of statements three time:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int m, a;
7     cout << "Please enter a number: " << endl;
8     cin >> m;
9     //-----the second number-----
10    cout << "Please enter a number:" << endl;
11    cin >> a;
12    if (a > m)
13        m = a;
14    //-----the third number-----
15    cout << "Please enter a number:" << endl;
16    cin >> a;
17    if (a > m)
18        m = a;
19    //-----the fourth number-----
20    cout << "Please enter a number:" << endl;
21    cin >> a;
22    if (a > m)
23        m = a;
24    cout << "The largest number you entered is " << m << endl;
25    return 0;
26 }
```

Now, we copy the code in one of the red boxes to the clipboard. Then delete the code in all three red boxes:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int m, a;
7     cout << "Please enter a number: " << endl;
8     cin >> m;
9
10    cout << "The largest number you entered is " << m << endl;
11    return 0;
12 }
13
14
15
```

Type the while loop we used in "Good Morning!" program:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int m, a;
7     cout << "Please enter a number: " << endl;
8     cin >> m;
9     int i = 0;
10    while (i < 3) {
11        cout << "Good Morning!" << endl;
12        i++;
13    }
14    cout << "The largest number you entered is " << m << endl;
15    return 0;
16 }
17
```

Delete the *cout* statement in line 11:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int m, a;
7     cout << "Please enter a number: " << endl;
8     cin >> m;
9     int i = 0;
10    while (i < 3) {
11
12        i++;
13    }
14    cout << "The largest number you entered is " << m << endl;
15    return 0;
16 }
```

Paste the code we have copied on the clipboard in the space where the statement `cout<<"Good Morning!"<<endl;` used to be (line 11)

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int m, a;
7     cout << "Please enter a number: " << endl;
8     cin >> m;
9     int i = 0;
10    while (i < 3) {
11        cout << "Please enter a number:" << endl;
12        cin >> a;
13        if (a > m)
14            m = a;
15        i++;
16    }
17    cout << "The largest number you entered is " << m << endl;
18    return 0;
19 }
20 }
```

Here, we use a while loop instead of three blocks of the same code, and it will work exactly the same.

However, this program only works if a user have four numbers to enter. If a user want to enter five numbers, then we have to change the looping condition from $i < 3$ to $i < 4$ to allow the user enter 5 numbers (first number goes to m directly and the rest of 4 numbers will be entered into a inside the loop). If a user has 15 numbers to enter, then we have to change the looping condition to $i < 14$. But, when we are coding this program, we don't know how many numbers a user would like to enter. If you put $i < 14$, what if a user wants to enter 20 numbers? If you put $i < 19$, what if a user wants to enter 6 numbers? Thus, putting in any specific number in the looping condition seems improper.

This what we are going to do:

Declare a new variable b and put $i < b$ for looping condition. Then put a set of `cout` and `cin` statements at the beginning of the program (line 7 and line 8) shown as below:

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int m,a,b;
7      cout << "How many data do you want to enter?" << endl;
8      cin >> b;
9      cout << "Please enter a number: " << endl;
10     cin >> m;
11     int i = 0;
12     while (i < b) {
13         cout << "Please enter a number:" << endl;
14         cin >> a;
15         if (a > m)
16             m = a;
17         i++;
18     }
19     cout << "The largest number you entered is " << m << endl;
20     return 0;
21 }
```

Now, this program will accommodate different needs of a user. It will first ask a user how many data he/she would like to enter, then use the user's input (the value of *b*) as part of looping condition to control the number of iterations and produce the largest number user entered.

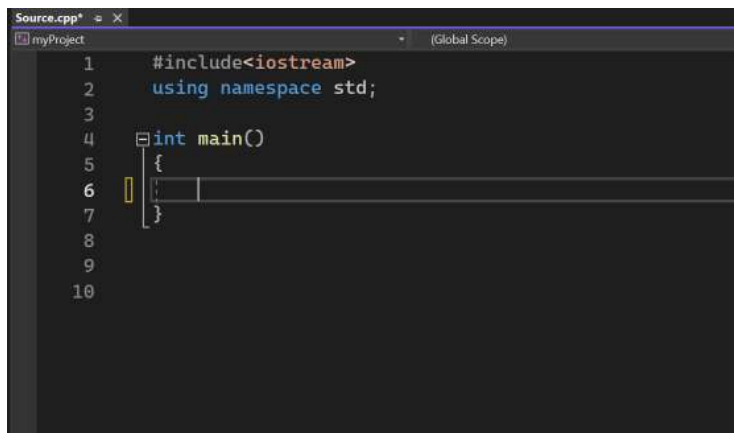
This page titled [5.2: Case Study- Find The Largest Number](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

5.3: Case Study- Find The Sum

Programming Task:

Write a program with a looping statement that allow a user to enter a set of integers and then produce (display) the sum of those numbers.

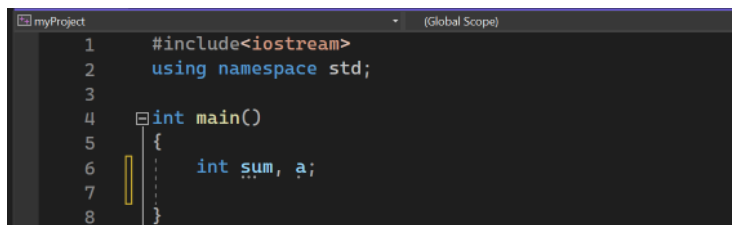
As always, we begin with an empty project with a blank source file opened and type in the code for the initial settings:



```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6
7  }
8
9
10
```

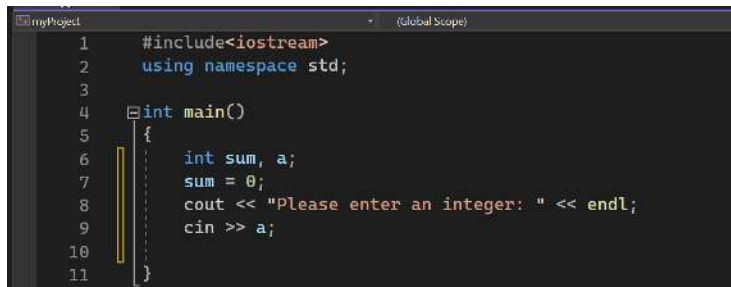
Step 1: Write a program that allows a user to enter 3 integers and display the sum of those numbers user entered.

We first declares two integer variables *sum* and *a*. We use the variable *sum* for storing the summation and use the variable *a* for storing the new number user entered.



```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int sum, a;
7
8  }
```

Next, we set the initial value of the variable *sum* to 0. Then use a set of *cout* and *cin* statements to allow a user to enter the first number into the program and save this number in the variable *a* as shown in the screenshot below:



```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int sum, a;
7      sum = 0;
8      cout << "Please enter an integer: " << endl;
9      cin >> a;
10
11 }
```

In the next line, we will put in a so called augmented statement: *sum = sum + a*; or *sum+=a*; this statement will take the value of *sum* and the value of *a*, add them up and put the result back to the variable *sum*. (Please review [section 2.3](#) for more information about augmented statement and augmented operators)

```
myProject (Global Scope)
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int sum, a;
7      sum = 0;
8      cout << "Please enter an integer: " << endl;
9      cin >> a;
10     sum = sum + a;
11
12 }
```

At this point, the variable *sum* in the program is holding the first number user entered. Now, we copy the code in line 8, 9, and 10 and paste them into the program after line 10 as shown in the red box below:

```
myProject (Global Scope)
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int sum, a;
7      sum = 0;
8      cout << "Please enter an integer: " << endl;
9      cin >> a;
10     sum = sum + a;
11     cout << "Please enter an integer: " << endl;
12     cin >> a;
13     sum = sum + a;
14
15 }
```

Now, line 11 asks the user to enter the second number. Line 12 is using *cin* statement to put the second number user entered in the variable *a*. The line 13 takes the value of the variable *sum* (it is holding the first number user entered) and the value of the variable *a* which is holding the second number user entered, adds them up and puts the result back to the variable *sum* (augmented statement).

At this point, the variable *sum* is holding the sum of the first and the second number user entered so far.

Now, we copy and paste those code in the red box above (line 11, 12 and 13) into the program right after the line 13:

```
myProject (Global Scope)
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int sum, a;
7      sum = 0;
8      cout << "Please enter an integer: " << endl;
9      cin >> a;
10     sum = sum + a;
11     cout << "Please enter an integer: " << endl;
12     cin >> a;
13     sum = sum + a;
14     cout << "Please enter an integer: " << endl;
15     cin >> a;
16     sum = sum + a;
17
18 }
```

In the program shown above, the line 14 is asking user to enter the third number, and the line 15 saves it in the variable *a*. Again, line 16 adds the value of *sum* (at this point the variable *sum* is holding the sum of the first two numbers user entered) and value of the variable *a*, puts the result back into the variable *sum*.

Thus, after the line 16 in this program, the variable *sum* is holding the sum of all three numbers user entered. What we need to do now is to put in a *cout* statement to display the sum of the three numbers user entered:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int sum, a;
7     sum = 0;
8     cout << "Please enter an integer: " << endl;
9     cin >> a;
10    sum = sum + a;
11    cout << "Please enter an integer: " << endl;
12    cin >> a;
13    sum = sum + a;
14    cout << "Please enter an integer: " << endl;
15    cin >> a;
16    sum = sum + a;
17    cout << "The sum of the numbers you entered is " << sum << endl;
18    return 0;
19 }
```

Now, we are done with this program. this is only the step 1. If we take a close look at this program, we can see this program runs the same group of statements three time:

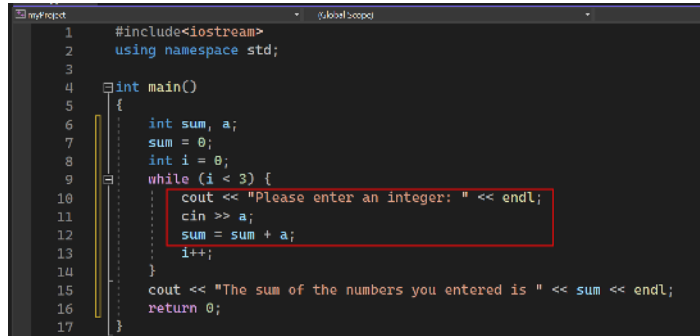
```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int sum, a;
7     sum = 0;
8     cout << "Please enter an integer: " << endl;
9     cin >> a;
10    sum = sum + a;
11    cout << "Please enter an integer: " << endl;
12    cin >> a;
13    sum = sum + a;
14    cout << "Please enter an integer: " << endl;
15    cin >> a;
16    sum = sum + a;
17    cout << "The sum of the numbers you entered is " << sum << endl;
18    return 0;
19 }
```

As we know, if a program needs to repeatedly execute a statement or a group of statements, we can use a looping statement. Thus, we can use a looping statement in place of those three groups of statements in the program. If you are not quite sure how to do it or if you are not quite sure what we are talking about. You can try to do following:

1. Copy the code in one of the red boxes to the clipboard
2. Delete the code in all three red boxes
3. Type in the code we used for displaying "Good Morning!" messages as shown below:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int sum, a;
7     sum = 0;
8     int i = 0;
9     while (i < 3) {
10         cout << "Good Morning!" << endl;
11         i++;
12     }
13     cout << "The sum of the numbers you entered is " << sum << endl;
14     return 0;
15 }
```

4. Delete line 10, and paste the code you copied to the clipboard in 1 at line 10:



```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int sum, a;
7     sum = 0;
8     int i = 0;
9     while (i < 3) {
10         cout << "Please enter an integer: " << endl;
11         cin >> a;
12         sum = sum + a;
13         i++;
14     }
15     cout << "The sum of the numbers you entered is " << sum << endl;
16     return 0;
17 }
```

This program is almost the one we would like to have for this assignment. However, it only calculates and displays the sum of three integers a user entered. It won't accommodate different needs of a user. For example, one user may need to find the sum of 5 integers and the other may need to calculate the sum of 20 integers and so on.

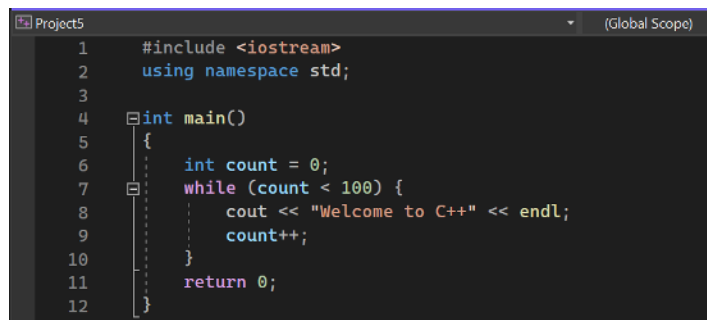
I will leave the final version of this program to you to complete. If you don't know how to complete the final version of this program, please study [section 5.2](#) where we were in a very similar situation and we solved this problem with an additional variable and a set of *cout* and *cin* statements.

This page titled [5.3: Case Study- Find The Sum](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

5.4: Sentinel Control Loop and do-while Loop Statement

Sentinel Control Loop

Most of the while loop statement used the examples were controlled by the values of a looping variable shown as below:



```

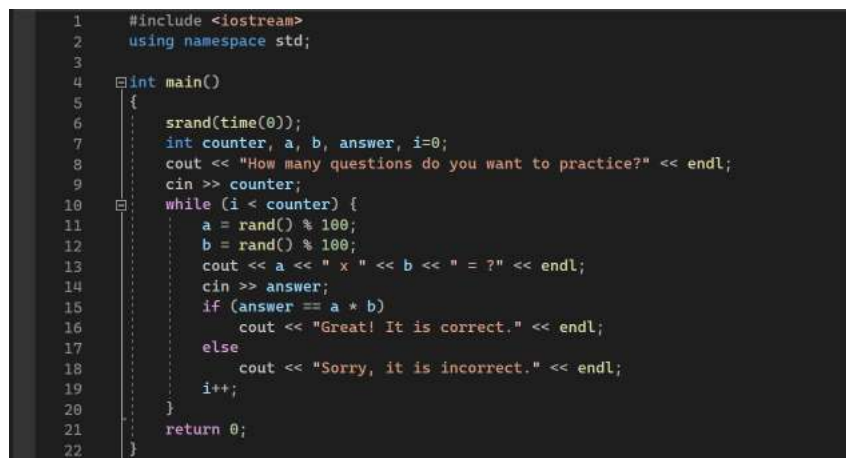
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int count = 0;
7      while (count < 100) {
8          cout << "Welcome to C++" << endl;
9          count++;
10     }
11     return 0;
12 }

```

In this program, we use a while loop statement to repeatedly execute the statement in line 8. The number of iterations controlled by the value of the looping variable *count*. When the value of the looping variable *count* reaches 100, the program exits the loop. Such looping statement is called a *counter control* loop statement. In a counter control loop statement, the number of iterations were coded into the program by the programmer and the user of the program has no control of the number of the iterations for such looping statement.

Another common technique for controlling a loop is to designate a special value, and use this value to allow a user to control the loop. Following is an example of using such technique for controlling a while loop statement:

Following is a screenshot of a program that helps a user to practice his/her multiplication skill:



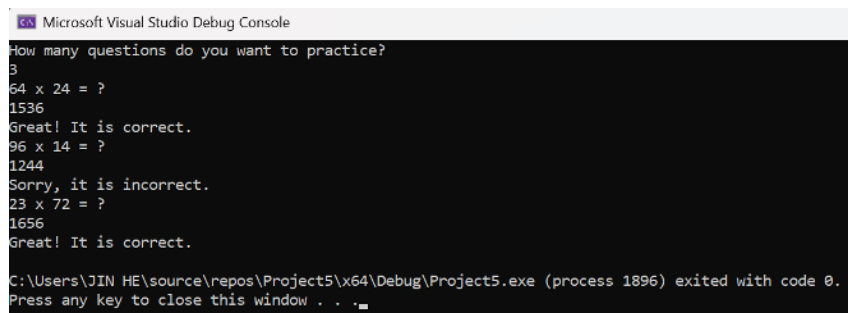
```

1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      srand(time(0));
7      int counter, a, b, answer, i=0;
8      cout << "How many questions do you want to practice?" << endl;
9      cin >> counter;
10     while (i < counter) {
11         a = rand() % 100;
12         b = rand() % 100;
13         cout << a << " x " << b << " = ?" << endl;
14         cin >> answer;
15         if (answer == a * b)
16             cout << "Great! It is correct." << endl;
17         else
18             cout << "Sorry, it is incorrect." << endl;
19         i++;
20     }
21     return 0;
22 }

```

In this program, we ask a user to enter the number of questions he/she would like to practice, and save the user's input into the variable *counter*, and use the value of this variable to control the number of iterations of the *while* loop.

Following is a screenshot of a sample run of this program:



```

Microsoft Visual Studio Debug Console
How many questions do you want to practice?
3
64 x 24 = ?
1536
Great! It is correct.
96 x 14 = ?
1244
Sorry, it is incorrect.
23 x 72 = ?
1656
Great! It is correct.
C:\Users\JIN HE\source\repos\Project5\x64\Debug\Project5.exe (process 1896) exited with code 0.
Press any key to close this window . . .

```

The user seems to have the control over the loop statement. However, this is still a counter control loop statement. Once a user enters the value for the variable *counter*, the user loses the control of the loop. For example, if a user enters 20, then the program will produce 20 questions, and the user can't do more or less unless the user forces to quit or restart the program.

Now, we revise the above program as follows:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      srand(time(0));
7      int a, b, answer, s=0;
8
9      while (s==0) {
10         a = rand() % 100;
11         b = rand() % 100;
12         cout << a << " x " << b << " = ?" << endl;
13         cin >> answer;
14         if (answer == a * b)
15             cout << "Great! It is correct." << endl;
16         else
17             cout << "Sorry, it is incorrect." << endl;
18         cout << "Try another problem?" << endl;
19         cout << "Enter 0 for Yes and Enter 1 for No" << endl;
20         cin >> s;
21     }
22     return 0;
23 }
```

Designate a variable as a sentinel

User's confirmation

In this program, we designated a variable *s* and initialized it to 0 and set the looping continuation condition to *s==0*. When *s==0* is true, we execute the statements in the body of the loop including a *cin* statement in line 20. This statement gives a user a chance to confirm whether the loop continues or stops. If a user enters 0 at line 20, then *s==0* is true, the statements in the body of the loop will be executed. If the user enters 1, *s==0* is false, the loop stops. In this program, *s* is a sentinel and such looping statement is called a sentinel control loop.

do-while Loop Statement

There are many different types of loop statements in C++ language. Here we discuss another loop statement: *do-while loop statement*.

A "do-while" loop is the same as a while loop, it allows a program to repeatedly execute a statement or a group of statements a certain number of times based on the status of a continuation condition.

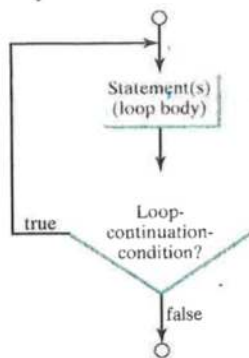
Here is the structure (syntax) of *do-while* loop statement:

do{

Statement (s); // the body of the loop

} while (loop-continuation-condition);

Following flow chart demonstrates the logic of the *do-while* loop statement:



When the computer sees a do-while loop, it will directly (without checking the loop-continuation-condition) go to the statements in the body of the loop. Upon the completion of the execution of the statements in the body of the loop, the computer will evaluate (check) the loop continuation condition. If the loop continuation condition is true, then the computer will go back to the body of the loop and execute those statements again, otherwise, the computer will move on to the next statement in your program that follows the *do-while* loop.

In other words, a regular while loop statement, the loop continuation condition will be checked first before the statements in the body of loop are executed. A do-while loop statement will execute the statements in the body of the loop first and then check the loop continuation condition. If the looping continuation condition is true, execute the body of the loop again, otherwise, exit the loop.

Following is a screenshot of a sample program that uses a *do-while* loop statement to display "Good Morning!" five times:

```

1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int i = 0;
7      do {
8          cout << "Good Morning!" << endl;
9          i++;
10     } while (i < 5);
11     return 0;
12 }
  
```

Of course, we can code the sample program with a regular *while* loop statement which will produce the same result. We put two programs below side by side:

<pre> 1 #include <iostream> 2 using namespace std; 3 4 int main() 5 { 6 int i = 0; 7 do { 8 cout << "Good Morning!" << endl; 9 i++; 10 } while (i < 5); 11 return 0; 12 } </pre>	<pre> 1 #include<iostream> 2 using namespace std; 3 4 int main() 5 { 6 int i = 0; 7 while (i < 5) { 8 cout << "Good Morning!" << endl; 9 i++; 10 } 11 return 0; 12 } </pre>
---	--

What we can see are

- *while* loop will check the looping continuation condition ***before*** the execution of the statements in the body of the loop
- *do-while* loop will check the looping continuation condition ***after*** the execution of the statements in the body of the loop.
- For any *do-while* loop, regardless of the loop continuation condition, the statements in the body of the loop will be executed ***at least once***. In above two programs, if we change the initial value of the looping variable *i* from 0 to 10, then the program on the left (the one that uses *do-while* loop) will display the message "Good Morning!" once, but the one on right (the one that uses *while* loop) will not display the message, not even once.

In following example, we demonstrate how to use a *do-while* loop statement to write a better menu drive program.

Example

Following screenshot is a menu driven program (we have discussed in [section 3.9](#)) that allows a user to selectively perform one of multiple tasks in a program:

```

1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int option;
7      cout << "=====+" << endl;
8      cout << "| 1. Celsius to Fahrenheit      |" << endl;
9      cout << "| 2. Display Next Month Appointment |" << endl;
10     cout << "| 3. Find My Grade                |" << endl;
11     cout << "| 4. Do Nothing                   |" << endl;
12     cout << "=====+" << endl;
13     cout << "Please select a task or enter 4 if you don't want to do anything." << endl;
14     cin >> option;
15     switch (option)
16     {
17     case 1:
18         double cel, fah;
19         cout << "Enter the temperature in Celsius:" << endl;
20         cin >> cel;
21         fah = 9 / 5.0 * cel + 32;
22         cout << "It is " << fah << " in Fahrenheit" << endl;
23         break;
24     case 2:
25         cout << "=====+" << endl;
26         cout << "|           April Appointments           |" << endl;
27         cout << "|-----|-----|-----|-----|" << endl;
28         cout << "| 04/10   Work Meeting       1:00 PM    |" << endl;
29         cout << "| 04/15   Dentist Appointment 12:45 PM   |" << endl;
30         cout << "| 04/23   Doctor Appointment 11:15 PM   |" << endl;
31         cout << "| 04/30   Work Meeting       4:15 PM    |" << endl;
32         cout << "=====+" << endl;
33         break;
34     case 3:
35         int a, b, c, d, s;
36         cout << "Please enter 4 test scores (0-100):" << endl;
37         cin >> a >> b >> c >> d;
38         s = (a + b + c + d) / 4;
39         if (s >= 90 and s <= 100)
40             cout << "A" << endl;
41         if (s >= 80 and s <= 89)
42             cout << "B" << endl;
43         if (s >= 70 and s <= 79)
44             cout << "C" << endl;
45         if (s >= 60 and s <= 69)
46             cout << "C" << endl;
47         if (s >= 0 and s <= 59)
48             cout << "F" << endl;
49         break;
50     case 4:
51         cout << "The user choose not to do anything" << endl;
52         break;
53     default:
54         cout << "Invalid input, please enter 1, 2, 3, or 4" << endl;
55     };
56     return 0;
57 }

```

This menu driven program is fine, but it only allows a user to perform one of the tasks in the program once. If the user would like to perform another task in the program or to perform the same task the second time, the user has to launch the program again because the program is terminated after a task is completed.

Now, we revise this menu driven program by using a *do-while* loop statement which allow a user to go back to the menu after the selected task is completed so the user can perform any task in the program and perform any one of the tasks as many times as user wants without re-launching the program.

Following is the screenshot of this program:

```

1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int option;
7      do {
8          cout << "=====+" << endl;
9          cout << "| 1. Celsius to Fahrenheit      |" << endl;
10         cout << "| 2. Display Next Month Appointment |" << endl;
11         cout << "| 3. Find My Grade                 |" << endl;
12         cout << "| 4. Do Nothing                    |" << endl;
13         cout << "=====+" << endl;
14         cout << "Please select a task or enter 4 if you don't want to do anything." << endl;
15         cin >> option;
16         switch (option)
17         {
18             case 1:
19                 double cel, fah;
20                 cout << "Enter the temperature in Celsius:" << endl;
21                 cin >> cel;
22                 fah = 9 / 5.0 * cel + 32;
23                 cout << "It is " << fah << " in Fahrenheit" << endl;
24                 break;
25             case 2:
26                 cout << "=====+" << endl;
27                 cout << "|          April Appointments          |" << endl;
28                 cout << "|-----+-----+-----+-----+|" << endl;
29                 cout << "| 04/10  Work Meeting      1:00 PM    |" << endl;
30                 cout << "| 04/15  Dentist Appointment 12:45 PM  |" << endl;
31                 cout << "| 04/23  Doctor Appointment 11:15 PM  |" << endl;
32                 cout << "| 04/30  Work Meeting      4:15 PM    |" << endl;
33                 cout << "=====+" << endl;
34                 break;
35             case 3:
36                 int a, b, c, d, s;
37                 cout << "Please enter 4 test scores (0-100):" << endl;
38                 cin >> a >> b >> c >> d;
39                 s = (a + b + c + d) / 4;
40                 if (s >= 90 and s <= 100)
41                     cout << "A" << endl;
42                 if (s >= 80 and s <= 89)
43                     cout << "B" << endl;
44                 if (s >= 70 and s <= 79)
45                     cout << "C" << endl;
46                 if (s >= 60 and s <= 69)
47                     cout << "C" << endl;
48                 if (s >= 0 and s <= 59)
49                     cout << "F" << endl;
50                 break;
51             case 4:
52                 cout << "The user choose not to do anything" << endl;
53                 break;
54             default:
55                 cout << "Invalid input, please enter 1, 2, 3, or 4" << endl;
56         };
57     } while (option!=4);
58     return 0;
59 }

```

This program will not check anything before it displays the menu. After the menu was displayed, it asked the user to make a selection by entering 1, 2, 3, or 4. The switch statement then responded the user's input. When the task the user selected was completed, the program exits the switch statement and then check the loop continuation condition which is "*option!=4*" this relational statement is false only when the value of the variable *option* is 4. For any other value, the looping continuation condition is true, thus, another iteration of the loop begins.

This page titled [5.4: Sentinel Control Loop and do-while Loop Statement](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

5.5: for Loop Statement

"for" Loop Statement

Besides *while* or *do-while* loop statements, there are other types of looping statements. Another popular looping statement is a *for* loop statement.

The structure (syntax) of a *for* loop statement:

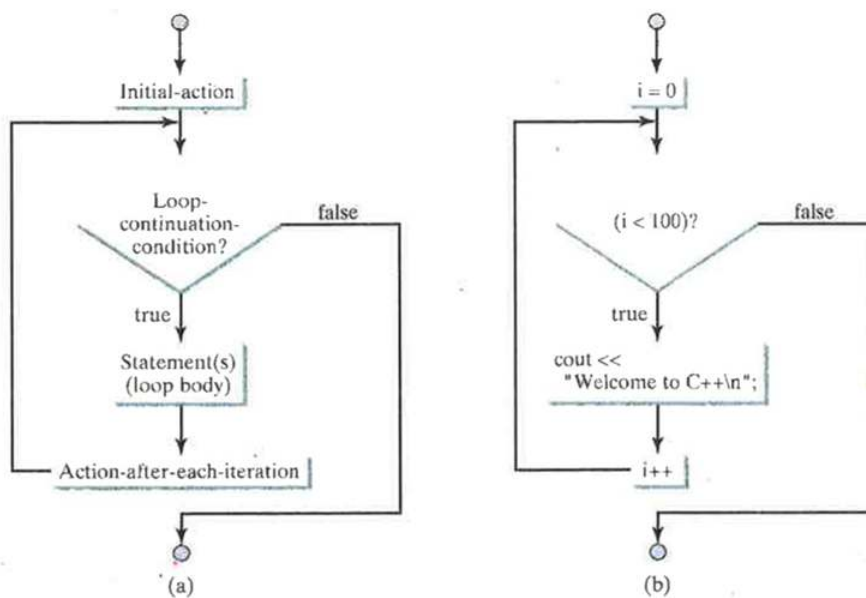
```
for (int i = initial_Value; i < end_Value; the statement that changes the value of i) {
    Statement (s); //body of the loop
}
```

Example

Following is a *for* loop statement which displays message "Welcome to C++" on the output screen 100 times.

```
for(int i = 0; i < 100; i++){
    cout << "Welcome to C++\n";
}
```

Following flow charts demonstrate the logic of this *for* loop statement:



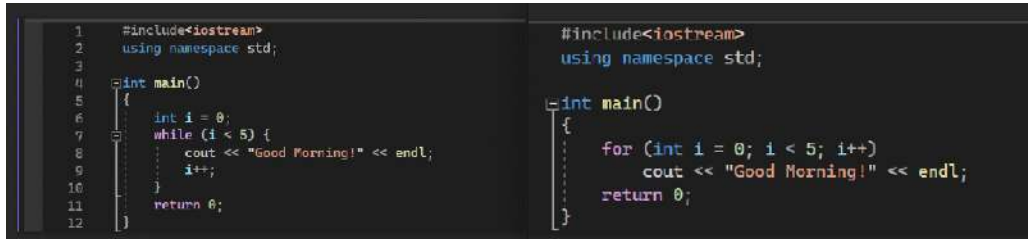
When the computer sees a *for* loop statement, it will initialize the variable *i* to 0, then evaluates the loop continuation condition *i* < 100, if it is a true statement, the computer goes to the body of the loop and executes those statements in the body of the loop. Upon the completion of the execution of the statements in the body of the loop, the computer executes the statement *i*++; to add 1 to the value of *i* and then re-check the loop continuation condition, if the loop continuation condition is still a true statement, then executes the statements in the body of the loop again, otherwise, move on to the next statement in the program that follows the *for* loop statement.

A *while* loop and a *for* loop are interchangeable. In other words, a statement or a group of statements can be repeatedly executed by either a *while* loop statement or a *for* loop statement. To help you to understand the logic of a *for* loop statement, we solve the same problem with a *while* loop and with a *for* loop and put two programs side by side.

Example

Write a program that displays a message "Good Morning! 5 times.

Following are two program that solve this problem, one uses a *while* loop and the other uses a *for* loop:



```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int i = 0;
7     while (i < 5) {
8         cout << "Good Morning!" << endl;
9         i++;
10    }
11    return 0;
12 }
```

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

int main()
{
    for (int i = 0; i < 5; i++)
        cout << "Good Morning!" << endl;
    return 0;
}
```

For this problem, the code of the program that is using for loop statement seems more compact.

This page titled [5.5: for Loop Statement](#) is shared under a [not declared](#) license and was authored, remixed, and/or curated by [Jin He](#).

5.6: Nested Loop Statements

A nested loop statement consists of an outer loop and one or more inner loops. In other words, nested loops are those loop statements for which the statements in the body of the loop also contains loop statements.

Example

```
for(int i = 0; i < 3; i++){  
    for(int j = 0; j < 5; j++){  
        cout << i << j << " ";  
    }  
    cout << endl;  
}
```

This nested looping statement has an outer loop:

```
for(int i = 0; i < 3; i++){  
}
```

The blue color coded are two statements which form the body of this outer loop:

```
for(int j = 0; j < 5; j++){  
    cout << i << j << " ";  
}  
cout << endl;
```

The first statement in the outer loop is also a *for* loop statement

The second statement in the outer loop simply moves the cursor to the beginning of the next line.

The program will repeatedly execute those two statements three times when the value of *i* takes the value of 0, 1, and 2

When *i* = 0, the program executes

```
for(int j = 0; j < 5; j++){  
    cout << 0 << j << " ";  
}
```

and

```
cout << endl;
```

The first *for* loop statement displays following numbers on the screen and the second *cout* statement moves the cursor to the beginning of the next line.

00 01 02 03 04

When the outer loop completes the execution of those two statements, it changes the value of its looping variable *i* from 0 to 1, and then evaluates the outer loop continuation condition which is *i* < 3.

Since *i* = 1, the outer loop continues and executes following two statements again, and this time, the value of *i* is 1 in the inner loop

```
for(int j = 0; j < 5; j++){  
    cout << 1 << j << " ";  
}
```

and

```
cout << endl;
```

Now, the program displays following numbers on the screen:

10 11 12 13 14

Then, the computer goes back to the outer loop to change the value of i from 1 to 2, and evaluates the loop continuation condition of the outer loop.

When $i = 2$, the computer executes following two statements again:

```
for(int j = 0; j < 5; j++){  
    cout << 2 << j << " ";  
}
```

and

```
cout << endl;
```

Displays following numbers on the screen:

20 21 22 23 24

Then, the computer goes back to the outer loop to change the value of i from 2 to 3, and evaluates the loop continuation condition of the outer loop.

When $i = 3$, the loop continuation condition of the outer loop becomes a false statement, the computer skip the statements in the body of the outer loop and move on to the next statement in the program. Now, following is the output of the program at this point:

00 01 02 03 04

10 11 12 13 14

20 21 22 23 24

Of cause, it is not necessary to use a nested loop statement to display those numbers on the screen, here we just use it as a sample code to help you to understand the logic of a nested looping statement.

=====

Now let's take a look another example.

What is output of following nested for loop?

```
for(int i=1;i<=6;i++){  
    for(int j=0;j<i;j++){  
        cout<<"* ";  
    }  
    cout<<endl;  
}
```

Now let us see how above nested loop works:
In the outer loop, first $j=1$
The program executes following two statements:

```
for(int j=0;j<1;j++){  
    cout<<"* "  
    cout<<endl;
```

which print a single asterisk and starts a new line

*

Then $j=2$
The program executes following two statements:

```
for(int j=0;j<2;j++){  
    cout<<"* "  
    cout<<endl;
```

which print out two asterisks and starts a new line, now the output is

*
**

Now i=3

The program executes following two statements:

```
for (int j=0; j<3; j++)  
    cout<<"* "
```

```
cout<<endl;
```

which print out three asterisks and starts a new line

*

*

*

When i=4, the program will print out

*

*

*

*

When i=5, the program will print out

*

*

*

*

*

*

When i=6, the program will print out

*

*

*

*

*

*

*

When i=7, the looping condition "i<=6" failed, the program skips following two statement and exit the outer for loop.

```
for (int j=0; j<3; j++)
```

```
    cout<<"* "
```

```
cout<<endl;
```

=====

Above examples help you to be able to read a nested loop, so you can understand how a nested loop works. Now we try to write a nested loop statement to solve a programming problem.

Now let us write a program that prints following triangle:

*

**

Let us first write a program without using nested loop

Step 1: print out three blank spaces and one asterisk by using following looping statements:

```
for (int k=0; k<3; k++)
```

```
    cout<<" ";
```

```
for (int k=0; k<1; k++)
```

```
    cout<<"*";
```

```
cout<<endl;
```

Step 2: using following statements to print out two blank spaces and two asterisks:

```
for (int k=0; k<2; k++)
```

```
    cout<<" ";
```

```
for (int k=0; k<2; k++)
```

```
    cout<<"*";
```

```
cout<<endl;
```

Step 3: using following statements to print out one blank spaces and three asterisks:

```
for(int k=0;k<1;k++)
    cout<<" ";
for(int k=0;k<3;k++)
    cout<<"*";
cout<<endl;
```

Step 4: using following statements to print out no blank space and four asterisks:

```
for(int k=0;k<0;k++)
    cout<<" ";
for(int k=0;k<4;k++)
    cout<<"*";
cout<<endl;
```

In each step, the red number in the first looping statement changes from 3 to 0, and the red number in the second looping statement changes from 1 to 4, now let us replace the red number in the second looping statement by i and replace the red number in the first looping statement by $4 - i$

```
for(int k=0;k<4-i;k++)
    cout<<" ";
for(int k=0;k<i;k++)
    cout<<"*";
cout<<endl;
```

When $i=1$, we have two looping statements in step 1

When $i=2$, we have two looping statements in step 2

When $i=3$, we have two looping statements in step 3

When $i=4$, we have two looping statements in step 4

Now, we write another loop to vary the value of i from 1 to 4:

```
for(int i=1;i<=4;i++) {
    for(int k=0;k<4-i;k++)
        cout<<" ";
    for(int k=0;k<i;k++)
        cout<<"*";
    cout<<endl;
}
```

So, the outer loop changes the value of the variable i in each of the inner loops to produce four groups of the statements we mentioned in Step 1, 2, 3, and 4 which produce the output of the triangle we expected.

The last example may not be easy to understand, you may need to study it a couple of times to grasp the idea of the code.

This page titled [5.6: Nested Loop Statements](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

5.7: Case Study- Display a Rectangle

Example

Write a program that displays a rectangle as following:

```
*****
* *
* *
* *
* *
*****
```

We first write a program as following with a nested loop statement:

There are two statements inside of outer loop. One of them is a *for* loop statement:

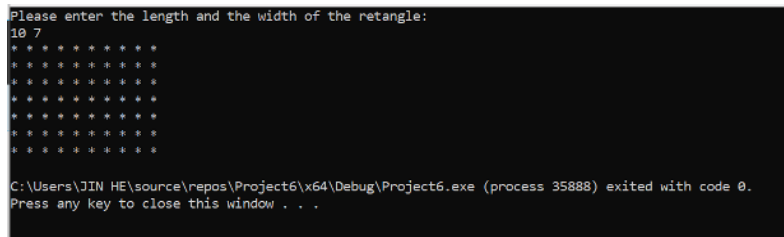
```
for(int j=0;j<length;j++){
cout<<"* ";
}
```

If a user enter 10 for the *length* and 7 for the *width*, then above loop statement will display 10 asterisks in a row. the second statement in the outer loop is

```
cout<<endl;
```

This statement will move the cursor to the beginning of the next line, so the next print will start at the beginning of next line.

The outer loop will execute those two statement 7 times, Thus the program will display 7 rows of the asterisks with 10 asterisks in each row:



However, this is not what we want. We only want the asterisks around the rectangle, but not inside of the rectangle. In other words, we only want the first row (when $i=0$) and the last row (when $i=width-1$) to be full rows of asterisks, from the second row ($i=1$) to the second to the last row ($i=width-2$), we only want the first column ($j=0$) and the last column ($j=length-1$) to be an asterisk and the rest of those column (from $j=2$ to $j=length-2$) to white spaces.

Now, we revise above program by adding an *if-else* statement shown as following:

```
Project6 (Global Scope)
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int length, width;
7      cout << "Please enter the length and the width of the rectangle:" << endl;
8      cin >> length >> width;
9      for (int i = 0; i < width; i++) {
10         for (int j = 0; j < length; j++) {
11             if (i == 0 or i == width - 1 or j == 0 or j == length - 1)
12                 cout << "*" ;
13             else
14                 cout << " ";
15         }
16         cout << endl;
17     }
18     return 0;
19 }
```

The output of this revised program is shown as following:

```
Please enter the length and the width of the rectangle:
10 7
* * * * *
*           *
*           *
*           *
*           *
* * * * *

C:\Users\JIN HE\source\repos\Project6\x64\Debug\Project6.exe (process 9836) exited with code 0.
Press any key to close this window . . .
```

If you would like to check if you actually understand above program, please try to write a program that displays a rectangle as following:

```
Please enter the length and the width of the rectangle:
10 7
* * * * *
* * * * *
* *       *
* *       *
* *       *
* * * * *
* * * * *

C:\Users\JIN HE\source\repos\Project6\x64\Debug\Project6.exe (process 9872) exited with code 0.
Press any key to close this window . . .
```

This page titled [5.7: Case Study- Display a Rectangle](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

CHAPTER OVERVIEW

6: Chapter 6 C++ Functions

[6.1: Introduction to C++ Functions](#)

[6.2: More Examples and Further Discussion about Functions](#)

[6.3: Global, Local, and Static Variable](#)

[6.4: Passing by Value vs Passing by Reference](#)

[6.5: Function Overloading](#)

This page titled [6: Chapter 6 C++ Functions](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

6.1: Introduction to C++ Functions

If you go back to take a look of the C++ programs you have written for this class, you will find the source file of each C++ program contains a single `main()` function which is in following form:

```
int main()
{
    //The code of your program
}
```

All the codes you wrote for the program were placed into this function between '{' and '}' Each C++ program's source file will have one and only one such function.

There are also other C++ functions you can use in a C++ program. Those "other" functions are generally classified into two categories.

Preloaded or Build-in Functions

We have used such functions in Study Section 1, such as `pow()`, which is one of the build-in functions that calculates the power of a number. For example, you may include following line in your program:

```
cout<<pow(2,5)<<endl;
```

this statement will calculate the fifth power of 2 which is 32 and displays it on the screen. In this case, 2 and 5 are called the **inputs** of the function. Each time when you call such function in your program you will need to provide such inputs, and they can be any numbers you choose and function will produce the correct calculation result accordingly.

`setw()` is another example of build-in functions, which is used in our program for formatting the output of our program. This function also requires an input, which is an integer.

User-Defined Functions

A user-defined function is a function we, as a programmer, will write in our program for certain purpose.

For build-in functions, we don't need to worry about the code of those functions, in other words, we don't need to know how those functions were written and how they were preloaded into the program. All we need to know is the names of those functions and how to use them in our programs. In this study section we will not focus our discussion on those functions, instead, we will learn how to write and use a user-defined functions.

A C++ function, regardless it is a preloaded/build-in function or it is a user-defined function, is a group of statements, commands that perform a certain task with the inputs to produce outputs. In a program, we can repeatedly use those functions as often as needed.

In this section, we will discuss:

1. How to name a function
2. How to specify the type of function's input and output. The output of a function can be a number, a character, or a string. If the output of a function is a number, a character, or a string, the function must have a so called "return statement" We will discuss it in examples.
3. How to write the code for a function so the function will perform the tasks specified.
4. How use those functions in our program.

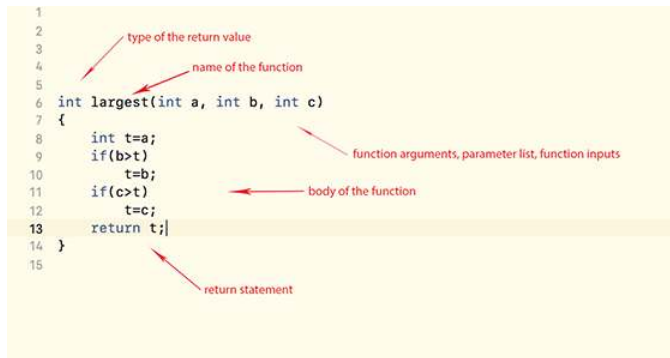
Now, let start with the structure of a C++ Function (Syntax of a C++ function):

```
Type-of-the-Return-Value nameOfFunction (Formal Parameters)
{
    statement (s);
}
```

In C++ programming, a function is normally referring to a group of statements, which perform a certain programming tasks, for example, a group of statements that calculates a person's body mass index (BMI).

A C++ function receives its input values through 'Formal Parameters', and use a set of C++ statements to perform certain task. If the task produces a value (it may be a numerical value, character value, Boolean value, or a string value), the function uses a return statement to return such value to the caller of the function. (Here the caller is referring to a statement in *main()* that contains the name of a function, we will discuss the details of the calling a function.)

Following shows an example of a C++ function:



```
1
2
3
4
5
6 int largest(int a, int b, int c)
7 {
8     int t=a;
9     if(b>t)
10        t=b;
11     if(c>t)
12        t=c;
13     return t;
14 }
15
```

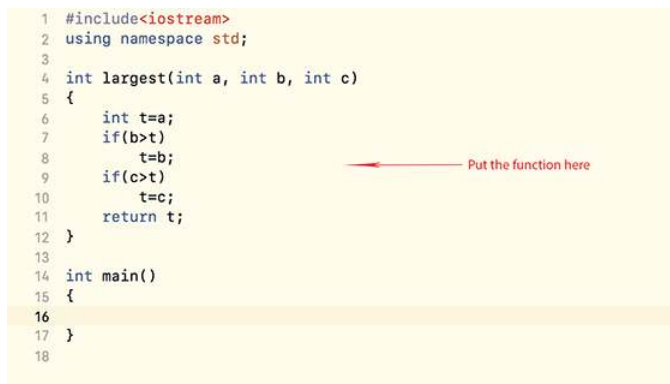
Annotations in the diagram:

- Line 6: *type of the return value* (points to 'int')
- Line 6: *name of the function* (points to 'largest')
- Line 6: *function arguments, parameter list, function inputs* (points to '(int a, int b, int c)')
- Lines 8-13: *body of the function* (points to the code block between '{' and '}')
- Line 13: *return statement* (points to 'return t;')

Line 6 is called the header of the function, which specifies the type of the return value of the function, the name of the function, and the formal parameters of the function. This function has three integer parameters, which is for holding the input values of the function.

Line 8 - Line 13: Those statements are called the body of the function. Those statements place the largest input into the variable *t* and return it.

There are many different ways to place a function in your project. Here, we place this function in the source file of our program as shown down below:



```
1 #include<iostream>
2 using namespace std;
3
4 int largest(int a, int b, int c)
5 {
6     int t=a;
7     if(b>t)
8         t=b;
9     if(c>t)
10        t=c;
11     return t;
12 }
13
14 int main()
15 {
16
17 }
18
```

Annotation in the diagram:

- Line 4-12: *Put the function here* (points to the `largest` function definition)

Once we place the function in the source file of the program, we can use such function in our program as following:

```

14 int main()
15 {
16     int x, y, z;
17     cout<<"Please enter any 3 integers:"<<endl;
18     cin>>x>>y>>z;
19     cout<<"The largest number is "<<largest(x,y,z)<<endl;
20     return 0;
21 }
22

```

Please enter any 3 integers:
4 11 8
The largest number is 11
Program ended with exit code: 0

The program send the return value of the function to the screen

The function is called in the program

The program passes 3 integers to the function and the function returns the largest integer to the main() function

Sample run of the program

In this example, we can see, a user-defined function is like a self-built programming tool, you can use such tool in our program wherever it is needed, and we can use it in our program as many time as we like, it brings convenience and clarity. Of course, we can put the code of this user-defined function in the main() and still can find the largest number. But, we will have to repeatedly type in those code if we need to perform such task multiple times.

From now on, when we are asked to write a function to perform a certain programming task, we will need to write a separate function which include the name of the function, type of output, type and number of the inputs, which is called the "header of the function". We also need to provide the code for such function so the function will correctly produce (return) the desired output.

Example

Write a function that receives a person's body weight and body height and returns the body mass index (BMI)

We have done this problem before. We used following code (statements) to perform such task:

```

(Global Scope)
#include<iostream>
using namespace std;

int main()
{
    double weight, height, kilos, meters, bmi;
    cout << "Please Enter the body weight in pounds: ";
    cin >> weight;
    cout << "Please Enter the height in inches: ";
    cin >> height;
    kilos = weight * 0.45359237;
    meters = height * 0.0254;
    bmi = kilos / (meters * meters);
    cout << "The BMI index is: " << bmi << endl;
    return 0;
}

```

We did this problem by placing all the code directly in the main() function, which was fine and it did the job. Now, we would like to revise this program by using a user-defined function that does the same job.

Step 1: Type the following function structure between the line "using namespace std;" and the line "int main()"

Type-of-the-Return-Value **nameOfFunction** (Formal Parameters)

```

{
}

```

Step 2: Copy all the statements in the main() function into the function structure as shown below:

Type-of-the-Return-Value **nameOfFunction** (Formal Parameters)

```
{  
  
double weight, height, kilos, meters, bmi;  
cout << "Please Enter the body weight in pounds: ";  
cin >> weight;  
cout << "Please Enter the height in inches: ";  
cin >> height;  
kilos = weight * 0.45359237;  
meters = height * 0.0254;  
bmi = kilos / (meters * meters);  
cout << "The BMI index is: " << bmi << endl;  
return 0;  
}
```

We will need to make some changes, otherwise, it will not be working.

Step 3: Delete those *cout* and *cin* statements. This function will not taking the body weight and the body height directly from console. Instead it will take those data through the function's arguments. Delete the variables *weight* and *height* as well. Now the function will look like following:

Type-of-the-Return-Value **nameOfFunction** (Formal Parameters)

```
{  
  
double kilos, meters, bmi;  
  
kilos = w * 0.45359237;  
meters = h * 0.0254;  
bmi = kilos / (meters * meters);  
cout << "The BMI index is: " << bmi << endl;  
return 0;  
}
```

Step 4: Replace "Type-of-the Return-Value" by "double"; Replace "nameOfFunction" by "BMI"; Replace "Formal Parameters" by "double weight, double height". The function will look like following:

```
double BMI (double weight, double height)  
{  
  
double kilos, meters, bmi;  
  
kilos = w * 0.45359237;  
meters = h * 0.0254;  
bmi = kilos / (meters * meters);  
  
return 0;  
}
```

here we specify that the number this function produces is a decimal number. We name this function as BMI. We setup two arguments for this function, through those arguments, the function will receives a person's weight and height.

Step 5: Change the last statement in the function from "return 0;" to "return bmi;"

```
double BMI (double weight, double height)  
{  
  
double kilos, meters, bmi;
```

```
kilos = w * 0.45359237;  
meters = h * 0.0254;  
bmi = kilos / (meters * meters);
```

```
return bmi;
```

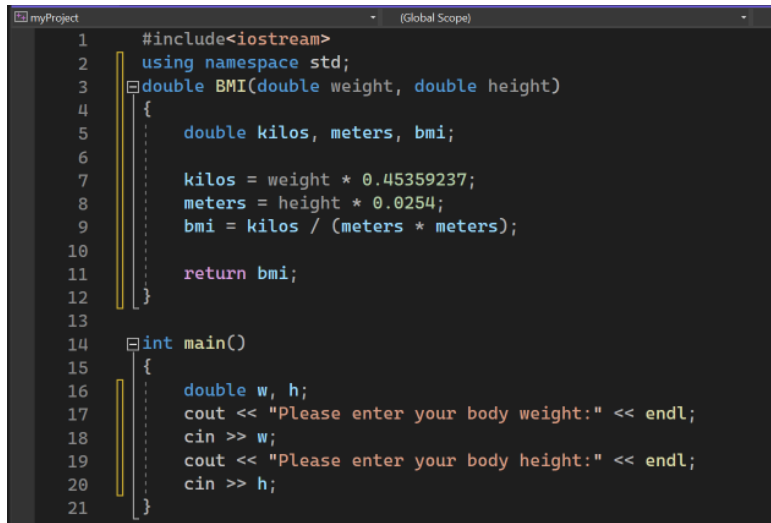
```
}
```

Now, we have completed this function.

We demonstrate how the function is used in the *main()* for calculating a person's body mass index.

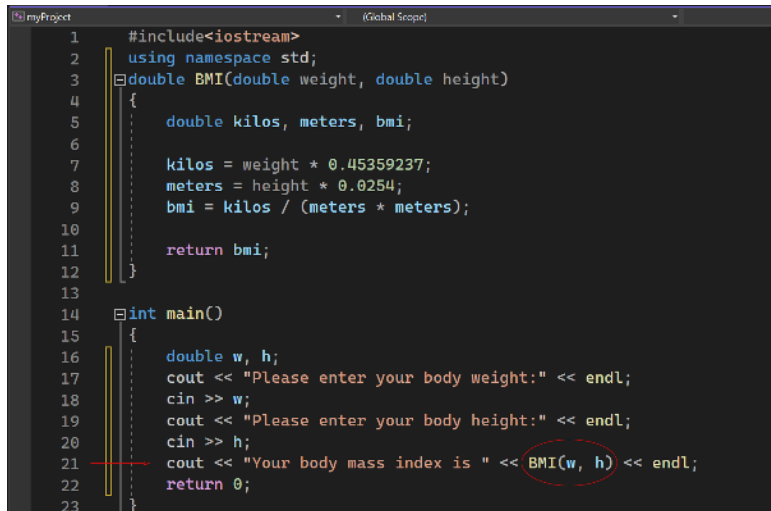
Step 6: Delete all the code in *main()* and

- Declare two double variables *w* and *h* for storing a person's weight and height in Line 16
- Use two pairs of *cout* and *cin* statements to ask a user to enter a person's weight and height and save them in the variable *w* and *h* in Line 17 - Line 20:



```
1  #include<iostream>  
2  using namespace std;  
3  double BMI(double weight, double height)  
4  {  
5      double kilos, meters, bmi;  
6  
7      kilos = weight * 0.45359237;  
8      meters = height * 0.0254;  
9      bmi = kilos / (meters * meters);  
10  
11     return bmi;  
12 }  
13  
14 int main()  
15 {  
16     double w, h;  
17     cout << "Please enter your body weight:" << endl;  
18     cin >> w;  
19     cout << "Please enter your body height:" << endl;  
20     cin >> h;  
21  
22 }
```

We add two more lines in *main()* to complete this program:



```
1  #include<iostream>  
2  using namespace std;  
3  double BMI(double weight, double height)  
4  {  
5      double kilos, meters, bmi;  
6  
7      kilos = weight * 0.45359237;  
8      meters = height * 0.0254;  
9      bmi = kilos / (meters * meters);  
10  
11     return bmi;  
12 }  
13  
14 int main()  
15 {  
16     double w, h;  
17     cout << "Please enter your body weight:" << endl;  
18     cin >> w;  
19     cout << "Please enter your body height:" << endl;  
20     cin >> h;  
21     cout << "Your body mass index is " << BMI(w, h) << endl;  
22     return 0;  
23 }
```

We add a *cout* statement at line 21. The statement displays the banner "Your body mass index is " followed by the number produced by the *return* statement inside of the function *BMI*. *BMI(w,h)* is the function call. The function receives the body weight and height entered by the user, through the *w* and *h*, processes them with the statements we coded into the function, and then produces the result as the output of the function and sends it to *main()* which is the caller of the function *BMI()* in this program.

Now, many of you may ask: "Why do we just calculate body mass index in *main()* as we did before? Why do we have to go through all those troubles to make a user defined function?"

First, we hide all the detail code of the calculation from *main()* to make *main()* look very clean and neat. Second, we can use BMI() for calculation anywhere we want in *main()* without re-typing the code. There are many other reasons and benefits of using a user defined function in C++ programming, and we will see them in the future.

This page titled [6.1: Introduction to C++ Functions](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

6.2: More Examples and Further Discussion about Functions

Example

Write a user-defined function with the name *sumBetween()*. This function receives two integers as its inputs and produces a sum of all the integers between those two inputs. For example, if the function receives 2 and 5, then *sumBetween(2, 5)* will produce a return value of 2+3+4+5, which is 14.

Step 1: In the structure of a C++ function shown below:

```
Type-of-the-Return-Value nameOfFunction (Formal Parameters)
{
    statement (s);
}
```

We replace "Type-of-the-Return-Value" by "int", replace "nameOfFunction" by "sumBetween", replace "Formal Parameters" by "int a, int b" shown in the following screen shot:

```
1  #include<iostream>
2  using namespace std;
3  int sumBetween(int a, int b)
4  {
5      ...
6  }
7  int main()
8  {
9      ...
10 }
```

Step 2: Write the code for the function shown in the following screen shot:

```
1  #include<iostream>
2  using namespace std;
3  int sumBetween(int a, int b)
4  {
5      int s = 0;
6      if (a > b) {
7          int t;
8          t = a;
9          a = b;
10         b = t;
11     }
12     for (int i = a; i <= b; i++) {
13         s += i;
14     }
15     return s;
16 }
17 int main()
18 {
19     ...
20 }
```

Line 5 - Line 15 are the code of the function *sumBetween*:

Line 5: we declare an integer variable *s* for storing the summation

Line 6 - Line 11: this *if* statement is to ensure the value of *a* is less than the value of *b*. (if the value of *a* is greater, then switch the value between *a* and *b*)

Line 12 - Line 14: use a for loop statement to add all the numbers between *a* and *b*

Line 15: this return statement returns the summation.

Step 3: write the code in the `main()` to test the function `sumBetween()`:

```
1  #include<iostream>
2  using namespace std;
3  int sumBetween(int a, int b)
4  {
5      int s = 0;
6      if (a > b) {
7          int t;
8          t = a;
9          a = b;
10         b = t;
11     }
12     for (int i = a; i <= b; i++) {
13         s += i;
14     }
15     return s;
16 }
17 int main()
18 {
19     int startingNum, endNum;
20     cout << "Please enter the first number:" << endl;
21     cin >> startingNum;
22     cout << "Please enter the second number:" << endl;
23     cin >> endNum;
24     cout << "The sum of all the numbers between " << startingNum << " and " << endNum << " is ";
25     cout << sumBetween(startingNum, endNum) << endl;
26     return 0;
27 }
```

A test run of this program produces following output:

```
Please enter the first number:
2
Please enter the second number:
6
The sum of all the numbers between 2 and 6 is 20

C:\Users\JIN HE\source\repos\Project6\x64\Debug\Project6.exe (process 16488) exited with code 0.
Press any key to close this window . . .
```

Most of functions produce a data as its output, such as the function we created in above program. As the sum of the numbers between *a* and *b*, The function `sumBetween(a,b)` produces an integer. Such number will be made available in the `main()`. We may display this number or use it for further calculation in other part of `main()`.

As an output of a function, it is not necessarily a numerical value, a character, or a string, it may be a display. Such display could be a number, a message, or a graph. In following example, we will discuss one of those functions.

Example

Write a user-defined function that receives an integer as its input and display a triangle accordingly.

Write a user-defined function that receives an integer as its input and display a triangle accordingly.

First, we open an empty project and create a source file with initial settings. We begin the coding of this user-defined function between "using namespace std;" and "int main()" in the source file:

```

1 #include<iostream>
2 using namespace std; use void for the type of the return value of the function
3
4 void displayTriangle(int base)
5 {
6     for(int i=base;i>0;i--){
7         for(int j=0;j<i;j++){
8             cout<<"* ";
9             cout<<endl;
10        }
11    }
12 }
13 int main()

```

- For a function that does not produce a data and only display something, we put "void" in front of the name of the function
- We name this function by "displayTriangle"
- We set up the argument of this function by an integer parameter "int base". The function will receive an integer as its input through parameter "base"
- Line 6 - 10 is a nested looping statement which displays a triangle with character '*'
- This function does not need a return statement.

Following screenshot shows how such function is used in **main()**. Please note, we don't need a **cout** statement to call such function. We simply call this function by its name. The bottom portion of the screenshot shows the output of this program, which is a display produced by the function **displayTriangle()**

```

1 #include<iostream>
2 using namespace std; use void for the type of the return value of the function
3
4 void displayTriangle(int base)
5 {
6     for(int i=base;i>0;i--){
7         for(int j=0;j<i;j++){
8             cout<<"* ";
9             cout<<endl;
10        }
11    }
12 }
13 int main()
14 {
15     int x;
16     cout<<"Please enter a integer: "<<endl;
17     cin>>x;
18     cout<<"The triangle is displayed as following:\n";
19     displayTriangle(x);
20
21     return 0; function is called directly by its name
22     no cout statement is needed.
23 }

```

```

Please enter a integer:
7
The triangle is displayed as following:
*****
*****
*****
*****
*****
*****
*****
Program ended with exit code: 0

```

Following example shows a user-defined function that produces a display of a menu. This function does not even require any inputs. The argument of the function contains 0 parameter. Thus, follow the name of the function, we have a pair of parentheses with nothing in it.

Example

```
Source.cpp  +  X
Project1    -  (Global Scope)
1  #include<iostream>
2  using namespace std;
3
4  void Menu()
5  {
6      cout << "+=====+\n";
7      cout << "| 1. Display a do do list   |\n";
8      cout << "| 2. Practice Multiplication   |\n";
9      cout << "| 3. Calculate Body Mass Index |\n";
10     cout << "| 4. Do nothing                |\n";
11     cout << "+=====+\n";
12 }
13
14 int main()
15 {
16     Menu();
17     return 0;
18 }
```

Example

Write a function that receives two integers and produces the greatest common factor as the output of this function. We may name this function as *gcf()*. For example, if this function receives 15 and 10, then *gcf(15,10)* will produce the common factor: 5

This function will be coded as following:

```
1  #include<iostream>
2  using namespace std;
3  int gcf(int a, int b)
4  {
5      int m = 0, commonFactor = 1;
6      if (a < b) {
7          m = a;
8      }
9      else {
10         m = b;
11     }
12     for (int i = 1; i <= m; i++) {
13         if (a % i == 0 and b % i == 0)
14             commonFactor = i;
15     }
16     return commonFactor;
17 }
18 int main()
19 {
20     int first, second;
21     cout << "Please enter the first number:" << endl;
22     cin >> first;
23     cout << "Please enter the second number:" << endl;
24     cin >> second;
25     cout << "The greatest common factore is " << gcf(first, second) << endl;
26     return 0;
27 }
```

Line 6 - Line 11: This *if* statement set *m* to the smaller number the function receives.

Line 12 - Line 15: This *for* loop statement check all the numbers from 1 to *m* to see if any one of them can evenly divide into both *a* and *b* and store such number into the variable *commonFactor*. If there are more than one number between 1 and *m* that can evenly divide into both *a* and *b*, then the loop statement will replace the smaller one by the larger one.

Line 16: return the largest *commonFactor*.

Following is a screenshot of a test run of this program:

```
Please enter the first number:
18
Please enter the second number:
42
The greatest common factore is 6

C:\Users\JIN HE\source\repos\Project6\x64\Debug\Project6.exe (process 27012) exited with code 0.
Press any key to close this window . . .
```

[6.2: More Examples and Further Discussion about Functions](#) is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

6.3: Global, Local, and Static Variable

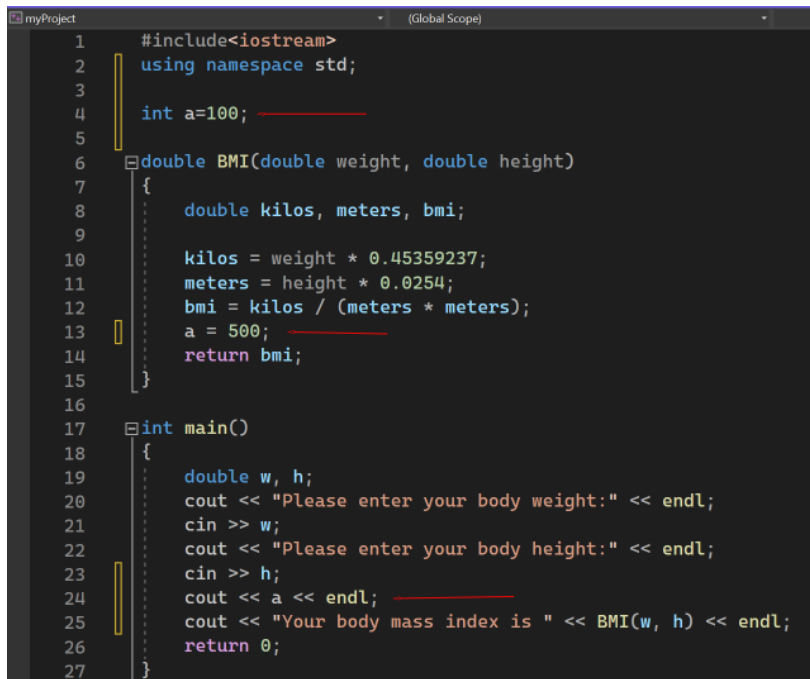
Global Variable

If a variable is declared outside of all the functions including `main()`, then it is a **global** variable. A global variable can be used inside of any user defined functions including `main()`. In other words, a global variable represents a "public" memory cell which is accessible from any user-defined function including `main()`.

In following program, at Line 4, a global variable `a` is declared. This variable is declared outside of all the user-defined functions as well as outside of `main()`. It is also initialized to 100.

We can use this variable `a` inside of the user-defined function `BMI()` at Line 13, which is assigned to 500.

We can also use this variable in `main()` at Line 24, where we use a `cout` statement to display its value on the screen.



```
1  #include<iostream>
2  using namespace std;
3
4  int a=100;
5
6  double BMI(double weight, double height)
7  {
8      double kilos, meters, bmi;
9
10     kilos = weight * 0.45359237;
11     meters = height * 0.0254;
12     bmi = kilos / (meters * meters);
13     a = 500;
14     return bmi;
15 }
16
17 int main()
18 {
19     double w, h;
20     cout << "Please enter your body weight:" << endl;
21     cin >> w;
22     cout << "Please enter your body height:" << endl;
23     cin >> h;
24     cout << a << endl;
25     cout << "Your body mass index is " << BMI(w, h) << endl;
26     return 0;
27 }
```

If we run this program, on the output screen, we will see 100 followed by the someone's body mass index.

But, in the main, if we place the statement `"cout<<a<<endl;"` after `"cout<<"Your body mass index is "<<BMI(w,h)<<endl;"` at Line 25 shown down below:

```

1  #include<iostream>
2  using namespace std;
3
4  int a=100;
5
6  double BMI(double weight, double height)
7  {
8      double kilos, meters, bmi;
9
10     kilos = weight * 0.45359237;
11     meters = height * 0.0254;
12     bmi = kilos / (meters * meters);
13     a = 500;
14     return bmi;
15 }
16
17 int main()
18 {
19     double w, h;
20     cout << "Please enter your body weight:" << endl;
21     cin >> w;
22     cout << "Please enter your body height:" << endl;
23     cin >> h;
24     cout << "Your body mass index is " << BMI(w, h) << endl;
25     cout << a << endl;
26     return 0;
27 }

```

Now we will see someone's body mass index followed by 500 because the global variable `a` was changed to 500 in the user-defined function `BMI()`. Thus, please note, we don't recommend to use a global variable unless we have to. The reason is the value of a global variable is not reliable since all the functions can access it and can make a change to it.

Please test run above two programs to see the difference.

Local Variable

A local variable is a variable that is declared inside of a function. Such variable can only be used inside of the function where it was declared. The life of a local variable is the same as the life of the function where it is declared. It means after the function call, all the variables that was declared inside of the function along with the data stored in them are all gone.

Example

In following program, the variables in the user-defined function `BMI()`: `kilos`, `meters`, `bmi` are local variables. The variables in `main()`: `w`, `h` are also local variables.

Variables `kilos`, `meters`, and `bmi` can only be used in the function `BMI()`. Variables `w` and `h` can only be used in `main()`.

```

1  #include<iostream>
2  using namespace std;
3  double BMI(double weight, double height)
4  {
5      double kilos, meters, bmi;
6
7      kilos = weight * 0.45359237;
8      meters = height * 0.0254;
9      bmi = kilos / (meters * meters);
10
11     return bmi;
12 }
13
14 int main()
15 {
16     double w, h;
17     cout << "Please enter your body weight:" << endl;
18     cin >> w;
19     cout << "Please enter your body height:" << endl;
20     cin >> h;
21     cout << "Your body mass index is " << BMI(w, h) << endl;
22     return 0;
23 }

```

Static Variable

A static variable is a variable that preserves its value after the function call is completed. We will use following sample programs to explain what "preserve its value" means.

```
1 #include<iostream>
2 using namespace std;
3
4 void functionA()
5 {
6     int a=5;
7     a+=10;
8     cout<<a<<endl;
9 }
10
11 void functionB()
12 {
13     static int a=5;
14     a+=10;
15     cout<<a<<endl;
16 }
17
18 int main()
19 {
20     cout<<"functionA() is called three times:\n";
21     functionA();
22     functionA();
23     functionA();
24     cout<<"functionB() is called three times:\n";
25     functionB();
26     functionB();
27     functionB();
28
29     return 0;
30 }
31
```

```
functionA() is called three times:
15
15
15
functionB() is called three times:
15
25
35
Program ended with exit code: 0
```

In *functionA()*, variable *a* was declared as a local variable and was initialized to 5. When the function was first called, then the statement "*a*+=10;" changes the value of this variable to 15, and the statement "*cout << a << endl*;" displays the value of *a* on the screen, it was the first 15 on the screen.

When the first function call was completed, the memory cell represented by the variable *a* is discarded and went back to the memory reservoir to be ready for re-use.

When the function *functionA()* was called again (the second time), the function allocates a piece of memory for variable *a* and initialize it to 5. The function repeat the same process: change the value of *a* to 15 and display it on the screen, and that was the second 15 on the screen.

Each time when the function *functionA()* was called, it allocates a new memory cell for the local variable *a* and initialize it to 5, change it to 15, and display it on the screen. That why we are seeing three 15's on the screen when we are calling *functionA()* three time in a roll.

In the *functionB()*, *a* was declared as a static variable by placing "**static**" in front of "int". In the function, the static variable *a* was also initialized to 5. then the function use statement "*a*+=10;"to change the value of this variable to 15, and use statement "*cout << a << endl*;" to display the value of *a* on the screen, it was the fourth 15 on the screen.

Then the function *functionB()* was called again (the second time), the variable *a* in the function represented the same memory cell as the variable *a* did in the first function call. Since the data in this memory cell had been changed from 5 to 15 by the first function call, in the second function call of this function, the statement "*a*+=10;" changed the value of this variable from 15 to 25, and the statement "*cout << a << endl*;" displays the value of *a* on the screen, it was 25 on the screen.

then the function *functionB()* was called again (the third time), the variable *a* in the function represented the same memory cell as the variable *a* did in the first function and it was also the memory cell that *a* represented in the second function call. Since the data in this memory cell had been changed from 5 to 15 and from 15 to 25 by the first and the second function call, in the third function

call of this function, the statement `"a+=10;"` changed the value of this variable from 25 to 35, and the statement `"cout << a <<endl;"` displays the value of `a` on the screen, it was 35 on the screen.

To summarize above discussion, please keep in mind:

1. A local variable represents a new memory cell each time when the function is called.
2. A static variable represents the same memory cell each time when the function is call, and this memory cell was allocated to the function when the function was called the first time.

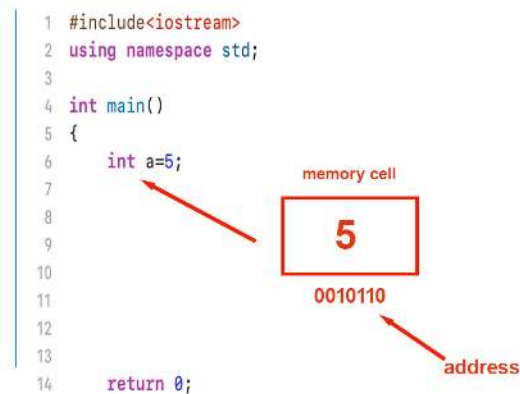
In short, a static variable preserves the value when the function is called, and a local variable doesn't.

6.3: Global, Local, and Static Variable is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

6.4: Passing by Value vs Passing by Reference

Passing by value and passing by reference is a difficult concept to grasp. Before we begin the discussion, we need to review the concept about variables, memory cells.

It is not just in C++, in all the computer programming language, we use variables in our coding. A variable represents a memory cell. When we declare a variable in a computer program, we assign a memory cell to the variable. When we initialize the variable, we place a data into the memory cell that the variable represents. Once we declare a variable in our program, we can store data, retrieve the data, as well as modify the data in the memory cell through the variable we declared. Each memory cell in the computer also has an address which is a sequence of integers.

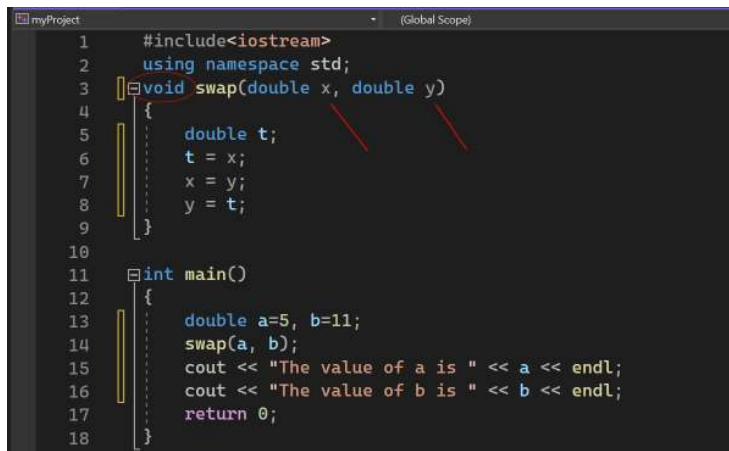


In following example, we will write a user-defined function. We will use this function to swap the value of two variables in the `main()`. In this example, we will go a little bit deeper into a user-defined functions and the variables, memories used in those functions.

Example

Write a user-defined function that can be used to in `main()` to swap the values of two variables in `main()`.

Following is the first draft of the function and a `main()` that contains the code for testing the function:



```

1  #include<iostream>
2  using namespace std;
3  void swap(double x, double y)
4  {
5      double t;
6      t = x;
7      x = y;
8      y = t;
9  }
10
11 int main()
12 {
13     double a=5, b=11;
14     swap(a, b);
15     cout << "The value of a is " << a << endl;
16     cout << "The value of b is " << b << endl;
17     return 0;
18 }

```

- The user-defined function `swap()` does not produce any data, so it was declared as `void`

- x and y are parameters used for receiving values of two variables in `main()`
- Line 5 declares a double variable t which will serve as temporary storage.
- Line 6 copies the value of x into the variable t
- Line 7 assigns the value of y to x
- Line 8 copies the value of t , which is the value of the original value of x , into the variable y

Now, x is holding the original value of y and y is holding the original value of x . The values of the variables x and y are swapped.

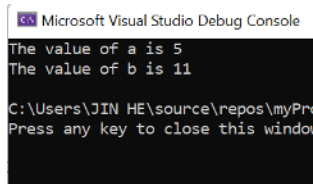
In the `main()`:

- line 13 declares two variables a , b and initializes them to 5 and 11.
- line 14 is calling the user-defined function `swap()` and passes two variables a and b to this function.
- line 15 and 16 display the value of a and b . If the function `swap()` is working correctly, then on the output screen, we should see:

The value of a is 11

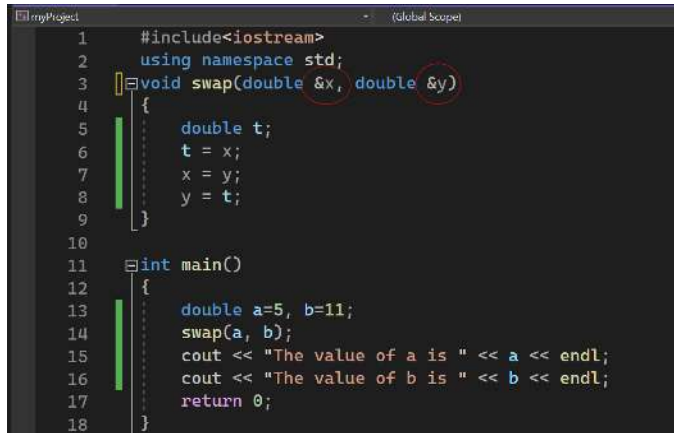
The value of b is 5

But, from the output of this program:



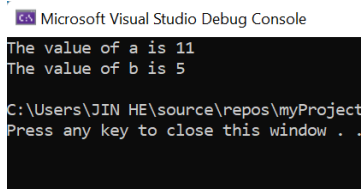
we can see that the value of a is still 5 and the value of b is still 11. Obviously, the user-defined function `swap()` is not working correctly.

We will provide a fix first, and then dig out the detail to find out why the original function did not work.



As you can see, the fix is easy: all we did is to put an `&` symbol in front of each parameter x and y of the function.

Now, if we run the program, we can see:



The values of the variable a and b are swapped by the function `swap()` in `main()`.

Why the original version of `swap()` was not working?

If we use the original version of the `swap()` in `main()`, when it receives two variables `a` and `b` at Line 14, the function allocates three memory cells: one represented by `t`, one represented by `x`, and one represented by `y` respectively. The function copies the value of the variable `a` into `x` and copies the value of `b` into `y`. Then the function use following three statements to swap the values in `x` and `y`:

```
t=x;
x=y;
y=t;
```

Now, the `x` is holding the value of `b` and `y` is holding the value of `a`. However, both `x` and `y` are local memory cells that belong to the function `swap()`. What the function `swap()` did is it swapped the values between those two local memory cells. Those two memory cells are not the same memory cells represented by variable `a` and `b` in the `main()`, thus, the function `swap()` did not swap the values in `a` and `b`, the value of `a` and `b` in `main()` remain unchanged.

In short, in this program, the variables `a`, `b`, `x`, `y` are representing four memory cells, two of them belong to `main()` and two of them belong to `swap()`. The function `swap()` copies the value of `a` and `b` into its local memory `x` and `y` and then swap the values between `x` and `y`. The function did not swap the value of `a` and `b` in `main()`.

Copying the value of the variable the function receives into a local memory cell is referred as "**Passing by value**", which is fine for most of the functions, but it won't work on the function `swap()` because the purpose of this function is to swap the values of two variables in `main()`.

By putting an `'&'` character in front of `x` and `y` in the argument of `swap()`, we change the game.

The second version of `swap()`, instead of copying the value of `a` and `b` into its local memory cells, let `x` and `y` represent the memory cells that allocated to the variable `a` and `b` in `main()`. Now, both `x` and `a` are representing the same memory cell and so are `y` and `b`. Thus, when the `swap()` uses following statement to swap the values in `x` and `y`:

```
t=x;
x=y;
y=t;
```

it also swaps the values in `a` and `b` in `main()`.

By placing `'&'` character in front of a parameter of a function, we allow this parameter, which is also a local variable, to directly represent the memory cell represented by the variable the function receives. This is so called "**Passing by reference**".

Most of user-defined functions use a `return` statement to send a calculation result to the caller function such as the functions we coded in earlier sections: `largest()`, `BMI()`, and `sumBetween()`, `gcf()` and so on. But, sometimes, a user-defined function may produce multiple calculation results. It will be easy to just display those data inside of the user-defined function. What if we need to pass them to the caller function?

Example

Write a user-defined function that uses the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

to calculate the solutions of a quadratic equation:

$$ax^2 + bx + c = 0$$

and return the calculation results to the caller function.

Following is a partially completed code of the function:

```

1 #include<iostream>
2 using namespace std;
3
4 double solution(double a, double b, double c)
5 {
6     double s1, s2;
7     s1=(-b+sqrt(b*b-4*a*c))/(2*a);
8     s2=(-b-sqrt(b*b-4*a*c))/(2*a);
9     return s1;
10 }
11 int main()
12 {
13
14     return 0;
15 }

```

This function correctly calculated the solutions of a quadratic equation by using the values of the coefficients of the function a , b , and c and the quadratic formula. However, it only returned one of the solutions of the equation to the caller function. We can't use more than one return statement in a function and we can't use a return statement to return more than one result either.

In order to return the second solution of the equation, we revise the code of the function and finish the `main()` to demonstrate how the caller function receive multiple outputs from a user-defined function:

```

1 #include<iostream>
2 using namespace std;
3
4 double getSolution(double a, double b, double c, double &solution2)
5 {
6     double s1, s2;
7     s1=(-b+sqrt(b*b-4*a*c))/(2*a);
8     s2=(-b-sqrt(b*b-4*a*c))/(2*a);
9     solution2=s2;
10    return s1;
11 }
12 int main()
13 {
14     double a, b, c, solution_1, solution_2;
15     cout<<"Please enter the values of coefficients:"<<endl;
16     cin>>a>>b>>c;
17     solution_1=getSolution(a,b,c,solution_2);
18     cout<<"The first solution of the equation is "<<solution_1<<endl;
19     cout<<"The second solution of the equation is "<<solution_2<<endl;
20     return 0;
21 }

```

In Line 4, we add an extra parameter `solution2` with `&` in front of it. Now, the variable `solution2` in `getSolution()` and the variable `solution_2` in `main()` represent the same memory cell that is declared for `solution_2` in `main()`. Therefore, the statement in Line 9 assigns the second solution of the equation to variable `solution_2` in `main()`.

In `main()`, Line 17, we call the function `getSolution()` and receive the first solution of the equation from the return value of the function and receive the second solution through the variable `solution_2`.

Following is the screenshot of the test run of this program:

```

Please enter the values of coefficients:
1 -5 -6
The first solution of the equation is 6
The second solution of the equation is -1
Program ended with exit code: 0

```

This page titled [6.4: Passing by Value vs Passing by Reference](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

6.5: Function Overloading

In C++, we may write different functions that have the same name. For example, we may write two functions, one of them receives two inputs and returns the largest number the functions receives, and the other receives 3 inputs and also returns the largest number the function receives. Instead of calling those two functions with different name such as `largest2()` and `largest3()`, we can name both of them as `largest()`.

Following is a sample program with a sample run:

```
1 #include<iostream>
2 using namespace std;
3
4 int largest(int a, int b)
5 {
6     if(a>=b)
7         return a;
8     else
9         return b;
10 }
11 int largest(int a, int b, int c)
12 {
13     int t=a;
14     if(b>t)
15         t=b;
16     if(c>t)
17         t=c;
18     return t;
19 }
20 int main()
21 {
22     int x,y,z;
23     cout<<"Please enter two integers:\n";
24     cin>>x>>y;
25     cout<<"The largest number is "<<largest(x,y)<<endl;
26     cout<<"Please enter three integers:\n";
27     cin>>x>>y>>z;
28     cout<<"The largest number is "<<largest(x,y,z)<<endl;
29     return 0;|
30 }
31
```

In the `main()` function, it seems the `main()` function is calling the same function twice, one with two inputs and the other with three inputs. Actually, the `main()` function is calling two different functions. How the computer know which function it is calling? The computer will tell which function it is calling by the number of the inputs. If we call the function `largest()` by providing two inputs, then the computer knows we are calling the first `largest()`; If we call the function `largest()` by providing three inputs, then the computer knows we are calling the second `largest()`.

The computer can also tell which function it calls based on the type of the inputs. Let us take a look of following sample program:

In this program, we have two functions that have the same name: `average()`. The first function receives two double type of the inputs and displays the average of those two inputs. The second function receives two character type of the inputs and displays the average of those two inputs symbolically.

```
1 #include<iostream>
2 using namespace std;
3
4 void average(double a, double b)
5 {
6     cout<<"The average is "<<(a+b)/2<<endl;
7 }
8 void average(char a, char b)
9 {
10     cout<<"The average is "<<"("<<a<<" + "<<b<<")/2"<<endl;
11 }
12 int main()
13 {
14     double x,y;
15     char r,s;
16     cout<<"Please enter two integers:\n";
17     cin>>x>>y;
18     average(x,y);
19     cout<<"Please enter two letters:\n";
20     cin>>r>>s;
21     average(r,s);
22     return 0;
23 }
24
```

```
Please enter two integers:
3 5
The average is 4
Please enter two letters:
m n
The average is (m + n)/2
Program ended with exit code: 0
```

In the *main()* function, when two double type of inputs are provided during a function call, the computer will invoke the first function and display the numerical average of the inputs; when two character type of inputs are provided, the computer will invoke the second function and display the symbolic average of the inputs.

Those two functions are called overloading functions. Overloading functions have the same name, but must either have the different number of the inputs or have the different type of the inputs. If two functions have the same name, the same number of inputs, and the same type of the inputs, then the computer will be confused and can't tell the difference of those two functions.

This page titled [6.5: Function Overloading](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

CHAPTER OVERVIEW

7: Chapter 7 Arrays

7.1: Introduction to Arrays

7.2: Working with the Data Stored in Arrays

7.3: Case Study- Sorting Data Stored in an Array

7.4: User-Defined functions with an Array as its Input

7.5: C-String and String Variables

This page titled [7: Chapter 7 Arrays](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

7.1: Introduction to Arrays

What is an Array?

In C++ programming, an array is referring to a sequence of memory cells declared in a program. It is an efficient way of storing a large set of data for the program to process. For example, if we are going to write a program that processes the data collected from a survey of 500 households' annual incomes, the first thing that we need to do is to store the data in our program, so we will need to allocate 500 memory cells to our program and place the data into those memory cells. Normally, we allocate memory cells by declaring variables. But, to declare 500 variables for storing 500 households' annual income seems not very practical. Very often, we will be using an array for such task.

In this case, instead of declaring 500 variables, we use following statement in the program to declare an array:

```
double income[500];
```

This statement allocates 500 memory cells to the program, we can use those memory cells to store the data collected. In this statement, 'double' indicates the memory cells can be used for storing the decimal data. 'income' is the name of the array we use in this program, it should reflect the nature of the data stored in the array. '500' indicates the total number of memory cells allocated.

Each memory cell in an array is called an element. Therefore, the array we declare above has 500 elements. and each element in this array is represented by following notations from the beginning to the end respectively:

```
income[0], income[1], income[2], ..., income[498], income[499]
```

The integer inside of the square bracket in each notation is called **index**. So, the index of the first element is 0 and the index of the last element is 499

It will be helpful to visualize an array with following diagram:



How to store the data into an Array in a C++ Program?

As variables, arrays need to be declared before it can be used in a program. Declaring an array is the same as declaring a variable. We need to specify

- data type of the element in the array
- the name of the array
- the number of elements in the array

For example, following statement can be used in a program to declare an integer array with 100 elements:



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

int main()
{
    int myArray[100];
}
```

The diagram shows three red arrows pointing to the code: one to 'int' labeled 'Data Type', one to 'myArray' labeled 'Array Name', and one to '100' labeled 'Number of Elements'.

Now we are going to discuss how to store the data in an array. For example, if we want to write a program that asks a user to enter 5 integers and save them in five memory cells, we may declare 5 variables in our program and use a **cin** statement to let a user to enter 5 integers and display those integers on the screen as following:

```
int main()
{
    int a, b, c, d, e;
    cout << "Please enter five integers: " << endl;
    cin >> a >> b >> c >> d >> e;
    cout << a << endl;
    cout << b << endl;
```

```
cout<< c <<endl;
cout<< d <<endl;
cout<< e <<endl;

return 0;
}
```

with an array, we now can do exact the same thing as following:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int a[5];
7     cout<<"Please enter five integers:"<<endl;
8     cin>>a[0]>>a[1]>>a[2]>>a[3]>>a[4];
9     cout<<a[0]<<endl;
10    cout<<a[1]<<endl;
11    cout<<a[2]<<endl;
12    cout<<a[3]<<endl;
13    cout<<a[4]<<endl;
14    return 0;
15 }
16
```

Line 6 declares an integer array with five memory cells. And those memory cells are represented by the symbols: a[0], a[1], a[2], a[3], a[4].

In stead of using five variables, in this program, we use a[0], a[1], a[2], a[3], a[4] to represent five integer memory cells and we can declare them with a single statement:

```
int a[5];
```

We can visualize this array with following diagram:



The array has five elements and the index of the first element is 0 and the index of the last element is 4.

Line 8: `cin>>a[0]>>a[1]>>a[2]>>a[3]>>a[4];` is a `cin` statement that takes the user's inputs and save them into those elements in the array we declared. More importantly, this statement can be replaced by following looping statement:

```
for(int i=0; i<5; i++){
    cin>>a[i];
}
```

Such replacement is especially important when we are working with a large set of integers.

Also, following statements in the program

```
cout<< a[0] <<endl;
cout<< a[1] <<endl;
cout<< a[2] <<endl;
cout<< a[3] <<endl;
cout<< a[4] <<endl;
```

can be replaced by following looping statement:

```
for(int i=0; i<5; i++){
    cout<<a[i]<<endl;
}
```

Now, the program can be revised as following with two looping statements:

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

int main()
{
    int a[100];
    cout << "Please enter 5 integers:" << endl;
    for (int i = 0; i < 5; i++)
        cin >> a[i];
    for (int i = 0; i < 5; i++)
        cout << a[i] << endl;
    return 0;
}
```

As you can see, using an array in a program is not only for avoiding using a large set of variables. More impurely, all the memory cells we used are indexed so we can deploy looping statements easily.

As we can initialize a variable by using following statement:

```
int a = 5;
```

We can also initialize arrays as followings:

```
int a[6] = {11, 21, 15, 10, 7, 5, 10};
```

```
double b[3] = {2.23, 3.11, 5.6};
```

```
string myArray[4] = {"Spring", "Summer", "Fall", "Winter"};
```

If we want to initialize all the elements in an array to be the same value, say 0, we can use following statement:

```
int a[100] = {0};
```

This statement will initialize all 100 elements in the array *a* to 0.

As we mentioned earlier, the main reason of using arrays in a program for storing data is that the memory cells are numerically labeled by indexes.

Example

In a program, we declare an integer array and initialized each element of the array as following:

```
int testScores[15] = {78, 65, 34, 55, 72, 68, 52, 48, 80, 60, 71, 43, 66, 76, 57};
```

If we want to add 5 extra points to every score that are stored in this array, instead of using following 15 statements:

```
testScore[0] += 5;
testScore[1] += 5;
testScore[2] += 5;
testScore[3] += 5;
testScore[4] += 5;
testScore[5] += 5;
testScore[6] += 5;
testScore[7] += 5;
testScore[8] += 5;
testScore[9] += 5;
testScore[10] += 5;
testScore[11] += 5;
testScore[12] += 5;
testScore[13] += 5;
```

```
testScore[14]+=5;
```

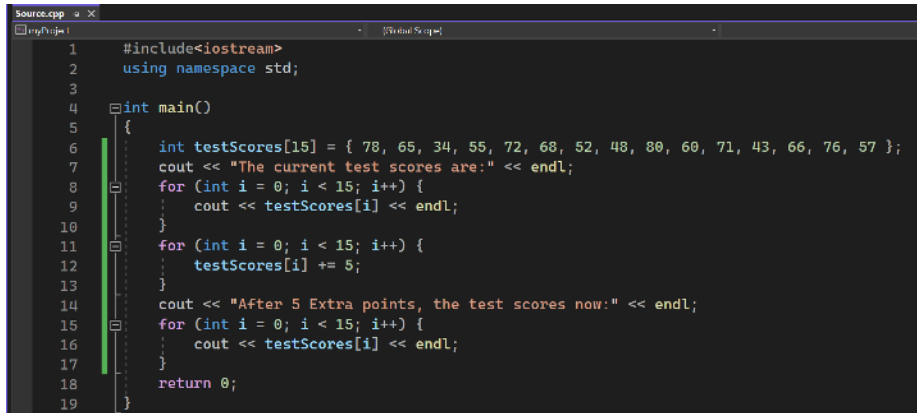
Note: $\text{testScore}[i] += 5$ is equivalent to $\text{testScore}[i] = \text{testScore}[i] + 5$; Such statement add integer 5 to the value of $\text{testScore}[i]$ and put the summation back to $\text{testScore}[i]$, where i represents index which runs from 0 to 14.

We can use a looping statement (shown as following) to replace above 14 augmented statements.

```
for(int i=0; i<15;i++)  
testScore[i]+=5;
```

In this looping statement, $\text{testScore}[i] += 5$; will be executed 15 times, the value of i starts at 0 and ends at 14. This looping statement works exactly as those 15 statements we had earlier.

Please test run following program:



```
1  #include<iostream>  
2  using namespace std;  
3  
4  int main()  
5  {  
6      int testScores[15] = { 78, 65, 34, 55, 72, 68, 52, 48, 80, 60, 71, 43, 66, 76, 57 };  
7      cout << "The current test scores are:" << endl;  
8      for (int i = 0; i < 15; i++) {  
9          cout << testScores[i] << endl;  
10     }  
11     for (int i = 0; i < 15; i++) {  
12         testScores[i] += 5;  
13     }  
14     cout << "After 5 Extra points, the test scores now:" << endl;  
15     for (int i = 0; i < 15; i++) {  
16         cout << testScores[i] << endl;  
17     }  
18     return 0;  
19 }
```

This page titled [7.1: Introduction to Arrays](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

7.2: Working with the Data Stored in Arrays

Once a collection of data are stored into an array in a C++ program, there are many things we may need to do with those data. Here we cover some basic tasks.

1. Display and calculate the sum of the data that are stored in the array

Following is a sample program that asks a user to enter a set of numbers and save them into an array. Then the program

- displays the data in the array on the screen
- calculate and display the sum of those data in the array

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     double a[5], s=0;
7     cout<<"Please enter 5 numbers:"<<endl;
8     for(int i=0;i<5;i++)
9         cin>>a[i];
10    for(int i=0;i<5;i++)
11        s+=a[i];
12    cout<<"The sum is "<<s<<endl;
13    return 0;
14 }
```

2. Find and Display the Largest Data Stored in the Array

In following sample program, we code a sample array into the program for testing purpose. We use a while loop statement to identify the largest number in the array and then display it on the screen. Please note: We don't alter the data in each element of the array while we are conduct the calculation.

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     double a[5]={2, -1, 5, 3, 1};
7     double m=a[0];
8     for(int i=0;i<5;i++)
9         if(a[i]>m)
10            m=a[i];
11    cout<<"The largest data is "<<m<<endl;
12    return 0;
13 }
```

3. Negative Number Counts in an Array

In this sample program, we ask a user to enter 20 integers and save them into an integer array. Then the program find the total negative number counts in the array and replace those negative numbers in the array by 0.

Here is the code:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int a[20],count=0;
7     cout<<"Please enter 20 integers:"<<endl;
8     for(int i=0;i<20;i++)
9         cin>>a[i];
10    for(int i=0;i<20;i++)
11        cout<<a[i]<<endl;
12    for(int i=0;i<20;i++){
13        if(a[i]<0){
14            count++;
15            a[i]=0;
16        }
17    }
18    cout<<"The total negative numbers in the array is "<<count<<endl;
19    for(int i=0;i<20;i++)
20        cout<<a[i]<<endl;
21
22    return 0;
23 }
```

4. Formatting The Display of the Data Stored in an Array

In this program, we use `rand()` to randomly generate 80 integers from 0 to 99 and store them in an integer array. Then we display those integers in the output screen with 10 integers in a row.

Here is the code:

```
1 #include<iostream>
2 #include<iomanip>
3 using namespace std;
4
5 int main()
6 {
7     int a[80];
8     srand((int)time(0));
9     for(int i=0;i<80;i++)
10        a[i]=rand()%100;
11    for(int i=0;i<80;i++){
12        if(i%10==0)
13            cout<<endl;
14        cout<<setw(5)<<a[i];
15    }
16    cout<<endl;
17    return 0;
18 }
```

Line 9 and Line 10: Store 80 random numbers in the array *a*.

Line 11 - Line 15: Use a for loop to display 80 numbers in the array *a*. The if statement in Line 12 ensure each time when the value of the looping variable *i* is a multiple of 10 a new line begins.

The output of the program is

```
51 98 11 97 75 91 78 15 26 8
67 73 35 93 66 87 71 82 47 75
6 98 35 19 85 23 18 76 60 77
62 61 94 54 9 96 83 27 95 95
73 87 87 20 81 34 36 30 74 22
35 78 23 67 72 68 60 46 6 3
62 64 54 76 84 1 77 75 83 81
51 36 9 43 30 51 23 1 69 56
Program ended with exit code: 0
```

This page titled [7.2: Working with the Data Stored in Arrays](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

7.3: Case Study- Sorting Data Stored in an Array

Sorting numbers in an array is a common programming task and there are many different ways of doing it. The way we are discussing here is often referred as "Bubble Sort".

To help you to understand the logic of the code of a "Bubble Sort" program, we begin with one of the sample programs we had in the [Section 3.4](#):

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main()
6  {
7      int a, b, c, d, t;
8      cout << "Please enter four integers: " << endl;
9      cin >> a >> b >> c >> d;
10     if (a > b) {
11         t = a;
12         a = b;
13         b = t;
14     }
15     if (b > c) {
16         t = b;
17         b = c;
18         c = t;
19     }
20     if (c > d) {
21         t = c;
22         c = d;
23         d = t;
24     }
25     ///////////////////////////////////////////////////
26     if (a > b) {
27         t = a;
28         a = b;
29         b = t;
30     }
31     if (b > c) {
32         t = b;
33         b = c;
34         c = t;
35     }
36     ///////////////////////////////////////////////////
37     if (a > b) {
38         t = a;
39         a = b;
40         b = t;
41     }
42     cout << "The values of the variable a, b, c, and d are now sorted in ascending order:" << endl;
43     cout << a << " " << b << " " << c << " " << d << endl;
44     return 0;
45 }
```

This program asks a user to enter four integers, and save them into four integer variables: *a*, *b*, *c*, and *d*. Then it uses the first three *if* statements (Line 10 - Line 24) to make sure the largest value is put into the variable *d*. Each one of those three *if* statements compares the values of each pair of two consecutive variables (the first *if* statement compares the values of *a* and *b*, the second *if* statement compares the values of *b* and *c*, and the third *if* statement compares the values of *c* and *d*). In each pair, if the value of the first variable is greater than the value of the second variable, then swap the values among those two variable to make sure the larger value is always stored the second variable of the pair. Thus, after those three *if* statements, the largest value will be in the variable *d*.

Then, this program uses the second group of two *if* statements (Line 26 - Line 35) to ensure the second largest number is stored in the variable *c*.

Finally, this program uses the last *if* statement (Line 37 - Line 41) to make sure the third largest number is in *b* and the smallest number is in *a*.

Now, we revise above program by replacing variables *a*, *b*, *c*, *d* with an array of four elements, here, *a*=*a*[0], *b*=*a*[1], *c*=*a*[2], and *d*=*a*[3]. The revise program is shown as following:

```

1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      int a[4], t;
7      cout << "Please enter four integers: " << endl;
8      cin >> a[0] >> a[1] >> a[2] >> a[3];
9      if (a[0] > a[1]) {
10         t = a[0];
11         a[0] = a[1];
12         a[1] = t;
13     }
14     if (a[1] > a[2]) {
15         t = a[1];
16         a[1] = a[2];
17         a[2] = t;
18     }
19     if (a[2] > a[3]) {
20         t = a[2];
21         a[2] = a[3];
22         a[3] = t;
23     }
24     ////////////////
25     if (a[0] > a[1]) {
26         t = a[0];
27         a[0] = a[1];
28         a[1] = t;
29     }
30     if (a[1] > a[2]) {
31         t = a[1];
32         a[1] = a[2];
33         a[2] = t;
34     }
35     ////////////////
36     if (a[0] > a[1]) {
37         t = a[0];
38         a[0] = a[1];
39         a[1] = t;
40     }
41     cout << "The values of the array are sorted in ascending order:" << endl;
42     cout << a[0] << " " << a[1] << " " << a[2] << " " << a[3] << endl;
43     return 0;
44 }

```

Following is a screen shot of a sample run of above program:

```

Please enter four integers:
7 4 11 5
The values of the array are sorted in ascending order:
4 5 7 11

C:\Users\JIN HE\source\repos\Project6\x64\Debug\Project6.exe (process 27216) exited with code 0.
Press any key to close this window . . .

```

We can replace the first three *if* statements (Line 9 - Line 23) by the following loop statement:

```

for(int i=0; i<3;i++){
    if(a[i]>a[i+1]){
        t=a[i];
        a[i]=a[i+1];
        a[i+1]=t;
    }
}

```

and replace the second two *if* statements (Line 25 - Line 34) by the following loop statement:

```

for(int i=0; i<2;i++){
    if(a[i]>a[i+1]){
        t=a[i];
        a[i]=a[i+1];
        a[i+1]=t;
    }
}

```

and then replace the last *if* statement (Line 36 - Line 40) by the following loop statement:

```
for(int i=0; i<1;i++){
if(a[i]>a[i+1]){
t=a[i];
a[i]=a[i+1];
a[i+1]=t;
}
```

Those three loop statements are almost identical except the looping continuation condition: $i < 3$; $i < 2$; $i < 1$; Thus, we can repeatedly execute those loop statements with another loop statement by changing the value of the looping variable from 3 to 1 as shown below:

```
for(int k=3;k>0;i--){
for(int i=0; i<k;i++){
if(a[i]>a[i+1]){
t=a[i];
a[i]=a[i+1];
a[i+1]=t;
}
}
}
```

Following is the final version of this program:

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      const int length = 4;
7      int a[length], t;
8      cout << "Please enter "<<length<<" integers: " << endl;
9      for (int i = 0; i < length; i++) {
10         cin >> a[i];
11     }
12     for (int k = length-1; k > 0; k--) {
13         for (int i = 0; i < k; i++) {
14             if (a[i] > a[i + 1]) {
15                 t = a[i];
16                 a[i] = a[i + 1];
17                 a[i + 1] = t;
18             }
19         }
20     }
21     cout << "The values of the array are sorted in ascending order:" << endl;
22     cout << a[0] << " " << a[1] << " " << a[2] << " " << a[3] << endl;
23     return 0;
24 }
```

If you would like to use this program to sort an array with different number of elements, simply change the value of the variable *length*.

This page titled [7.3: Case Study- Sorting Data Stored in an Array](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

7.4: User-Defined functions with an Array as its Input

In this section, we discuss how to create a user-defined function that uses an array as its input.

Example

Write a user-defined function that receives an integer array as its input. The function produces the sum of the numbers in the array as its output.

Find the sum of all the numbers in an array is not a new programming task. We have done it in [Section 7.2](#). But we did it in the *main()*. Now, we will discuss how to get it done in a user-defined function.

Here is the code of this function along with a *main()* for testing purpose:

```
1  #include<iostream>
2  using namespace std;
3
4  int getSum(int a[], int length)
5  {
6      int s=0;
7      for(int i=0;i<length;i++){
8          s+=a[i];
9      }
10     return s;
11 }
12 int main()
13 {
14     int sampleArray[12]={3,-1,2,5,11,8,4,2,-7,10,-5,1};
15     cout<<"The sum is "<<getSum(sampleArray,12)<<endl;
16     return 0;
17 }
```

The function *getSum()* receives an integer array along with the array's length and calculates and returns the sum of the integers in the array it receives. Please note: when we specify one of the input is an array, we use a pair of square brackets with no number in between, thus, the function can take an array with various length. Also notice, when we use this function in the *main()* in Line 15, we only pass the name of the array along with the length of the array.

Example

Write a user-defined function that receives an integer array as its input. The function checks the array to see if there is a negative number in it. If there is one, then return a Boolean value *true*, otherwise return *false*.

Following is the code of the function along with *main()* for testing purpose:

```

1  #include<iostream>
2  using namespace std;
3
4  bool containNegative(int a[], int length)
5  {
6      for (int i = 0; i < length; i++) {
7          if (a[i] < 0)
8              return true;
9      }
10     return false;
11 }
12
13 int main()
14 {
15     int array_1[5] = { 4, 2, 11, 5, -1 };
16     int array_2[6] = { 3, 1, 8, 3, 9, 1 };
17     if (containNegative(array_1, 5)) {
18         cout << "This array contains a negative number." << endl;
19     }
20     else {
21         cout << "This array does not contain a negative number." << endl;
22     }
23     if (containNegative(array_2, 6)) {
24         cout << "This array contains a negative number." << endl;
25     }
26     else {
27         cout << "This array does not contain a negative number." << endl;
28     }
29     return 0;
30 }

```

From this example, we can see that the function can work with arrays with different lengths.

Example

Write a user-defined function that receives an integer array as its input. The function sorts the numbers in the array it receives in a ascending order.

We wrote a program that sort the numbers in an array in an ascending order in the previous section ([Section 7.3](#)). But we did it in `main()`. Now, we will create a user-defined function that will do the same job.

Following is the code of the function along with `main()` for testing purpose:

```

1  #include<iostream>
2  using namespace std;
3
4  void sort(int a[], int length)
5  {
6      int t;
7      for (int k = length-1; k > 0; k--) {
8          for (int i = 0; i < k; i++) {
9              if (a[i] > a[i+1]) {
10                 t = a[i];
11                 a[i] = a[i+1];
12                 a[i+1] = t;
13             }
14         }
15     }
16 }
17
18 int main()
19 {
20     int array_1[5] = { 4, 2, 11, 5, -1 };
21     int array_2[6] = { 3, 1, 8, 3, 9, 1 };
22     sort(array_1, 5);
23     sort(array_2, 6);
24     cout << "The arrays are sorted as following:" << endl;
25     cout << "array-1:" << endl;
26     for (int i = 0; i < 5; i++) {
27         cout << array_1[i] << " ";
28     }
29     cout << endl;
30     cout << "array-2:" << endl;
31     for (int i = 0; i < 6; i++) {
32         cout << array_2[i] << " ";
33     }
34     cout << endl;
35     return 0;
36 }

```

Line 4: Since this function only sort the array it receives, no return value is needed, so we declare such function as void.

Line 6: We only need to declare a variable `t` in the function for swapping purpose. The value of the variable length is coming from the second input of the function. The code of the nested looping statement is directly copy and paste from the previous program in the Section 7.3.

In `main()` we call this function twice in Line 22 and Line 23 with different arrays as their inputs.

The output of this program:

```
The arrays are sorted as following:
array-1:
-1 2 4 5 11
array-2:
1 1 3 3 8 9

C:\Users\JIN HE\source\repos\Project6\x64\Debug\Project6.exe (process 25652) exited with code 0.
Press any key to close this window . . .
```

This page titled [7.4: User-Defined functions with an Array as its Input](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

7.5: C-String and String Variables

String Variable and C-String

In a C++ program, a string of characters can be stored in either a string variable or in a character array.

To store a string of characters in a string variable or in a c-string (a character array), we can use either an assignment statement in the program or use a *cin* or *getline()* statement.

Following is a screen shot of a program. In this program, a string "Homework" is stored in a string variable *s* in line 7 and in an array of characters *a* in line 8 respectively. In either case, the string is stored in an array one character per element.

 undefined

At the end of the array, a special character '\0' is placed into the array to indicating the end of the string.

The program uses a while looping statement to display the characters in each string on the screen with one character per line. We can see that the end of string character, \0, is used for terminating the looping statement.

```
1 #include<iostream>
2 #include<string>
3 using namespace std;
4
5 int main()
6 {
7     string s="Homework";
8     char a[]="Homework";
9
10    int i=0;
11    while(s[i]!='\0'){
12        cout<<s[i]<<endl;
13        i++;
14    }
15    cout<<"=====\n";
16    i=0;
17    while(a[i]!='\0'){
18        cout<<a[i]<<endl;
19        i++;
20    }
21    cout<<"=====\n";
22    i=0;
23
24    return 0;
25 }
26
```

The output of the program is:

```
H
o
m
e
w
o
r
k
=====
H
o
m
e
w
o
r
k
=====
```

In this program, the strings "Homework" are placed into string variable *s* and character array *a* with the code of the program. It is convenient for demonstration or testing purposes. But, if we want to write a program to save a string that is entered by a user, then we will need to revise this program as following:

```
1 #include<iostream>
2 #include<string>
3 using namespace std;
4
5 int main()
6 {
7     string s;
8     char a[100];
9     cout<<"Please enter a string: "<<endl;
10    getline(cin,s);
11    cout<<"Please enter a string: "<<endl;
12    cin.getline(a,100);
13    int i=0;
14    while(s[i]!='\0'){
15        cout<<s[i]<<endl;
16        i++;
17    }
18    cout<<"====="<<endl;
19    i=0;
20    while(a[i]!='\0'){
21        cout<<a[i]<<endl;
22        i++;
23    }
24    return 0;
25 }
```

The statement in Line 10: `getline(cin, s)` allows a user to input a string into the string variable `s`. The statement in Line 12: `cin.getline(a, 100)` allows a user to input a string into the character array `a`.

Please note: to use a string variable in a program, we need to add a line of code `"#include<string>"` at the beginning of the source code as we did in above sample program in Line 2.

Example

Write a user-defined function that use a c-string as its input. The function returns the counts of a specified character in the array.

Following is the code of the function along with a `main()` that shows how the function is used in a C++ program:

```
1 #include<iostream>
2 using namespace std;
3
4 int charCount(char a[], char key)
5 {
6     int i=0, counter=0;
7     while(a[i]!='\0'){
8         if(a[i]==key)
9             counter++;
10        i++;
11    }
12    return counter;
13 }
14
15 int main()
16 {
17     char str[100],ch;
18     cout<<"Please enter a string:"<<endl;
19     cin.getline(str,100);
20     cout<<"Please enter an character:"<<endl;
21     cin>>ch;
22     cout<<"The count of character "<<ch<<" is "<<charCount(str,ch)<<endl;
23     return 0;
24 }
25
```

Line: 10 Col: 13

```
Please enter a string:
I am student in Computer Science
Please enter an character:
e
The count of character e is 4
Program ended with exit code: 0
```

Line 4: Since the function returns the count of the occurrences of a character, the type of the return value of the function should be *int*. We name the function as. The function needs two inputs: *char a[]* and *char key*. The first one is a character array and the second one is a single character.

Line 6 - Line 12: The body of the function. It uses a *while* loop to check each character in the array against the value of the variable *key*. Increases the value of the variable *counter* if there is a match. The looping continuation condition *a[i]!='\0'* ensure the loop stops when all the characters in the array were checked.

Line 22: The function was called with a string stored in the array *str* and a character stored in the variable *ch* and the return value of the function was displayed in the output of the program. The string the user entered is "*I am s tudent in Computer Science*" and the count of the character '*e*' is 4.

This page titled [7.5: C-String and String Variables](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

CHAPTER OVERVIEW

8: Introduction to Object Oriented Programming

- [8.1: Introduction to Object Oriented Programming \(PC Users\)](#)
- [8.2: Introduction to Object Oriented Programming \(Mac Users\)](#)
- [8.3: Class Constructors](#)
- [8.4: Case Study- C++ Class- StudentRecord](#)
- [8.5: Case Study- C++ Class- Bag](#)

This page titled [8: Introduction to Object Oriented Programming](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

8.1: Introduction to Object Oriented Programming (PC Users)

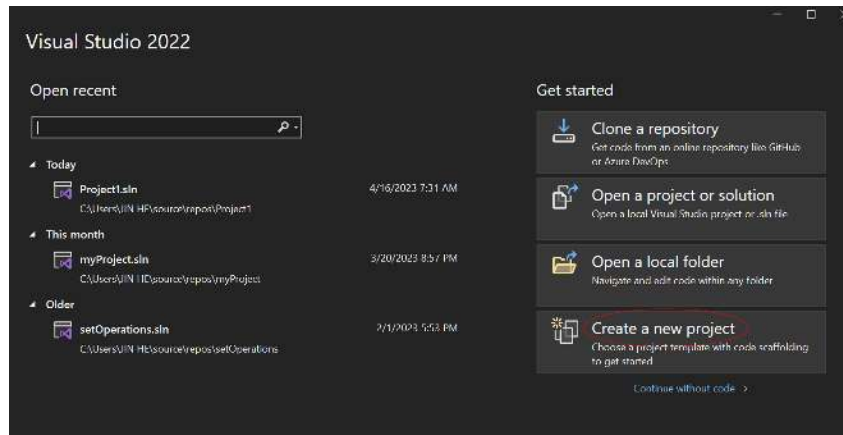
Object-oriented programming (OOP) is a computer programming approach that uses classes and objects. In object-oriented programming, we first create classes and then use classes to create objects. Such objects possess certain features and functions that can be used in a program.

As an example, here we create a class called *Rectangle* and then we use this class to create objects in a program.

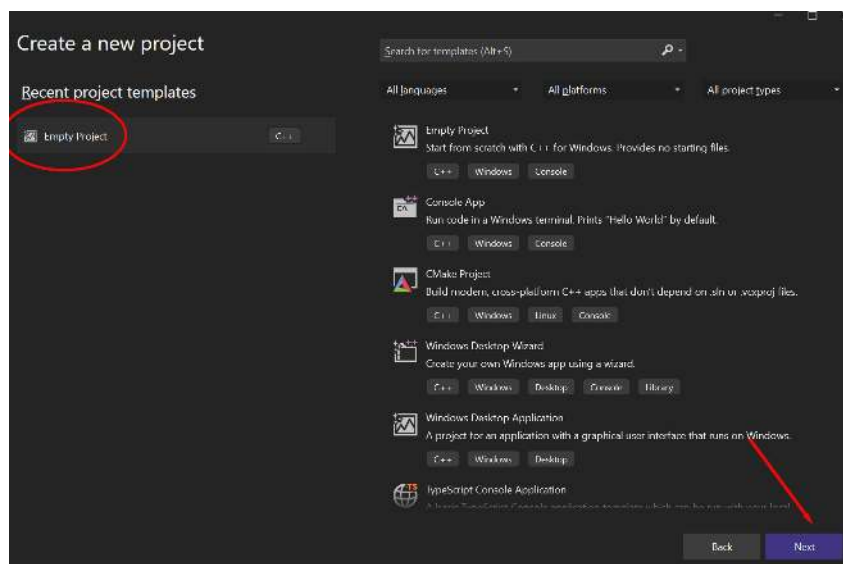
To create a class, we will need to open a header file (a file with .h as its file extension)

To create a header file:

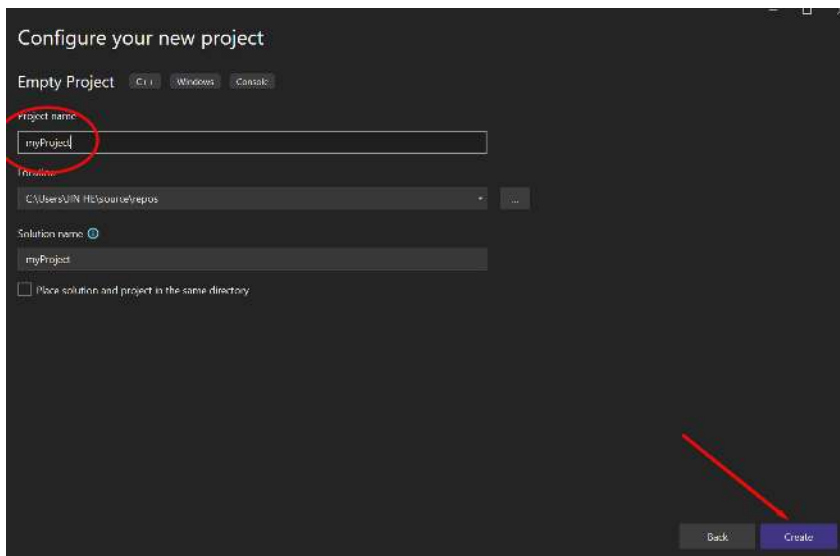
1. Open a project



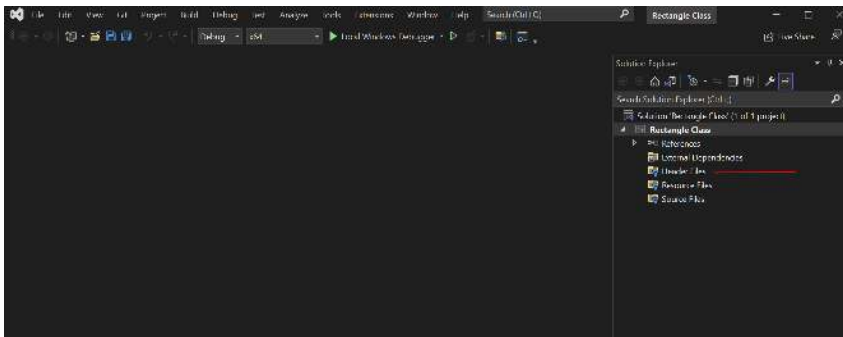
Select "Empty Project" and then click on "Next":



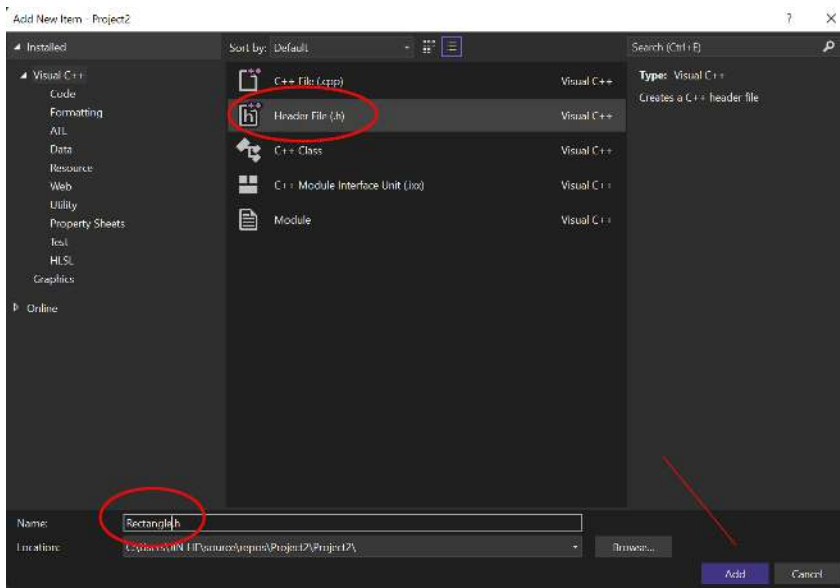
Name the project, and then click on "Create"



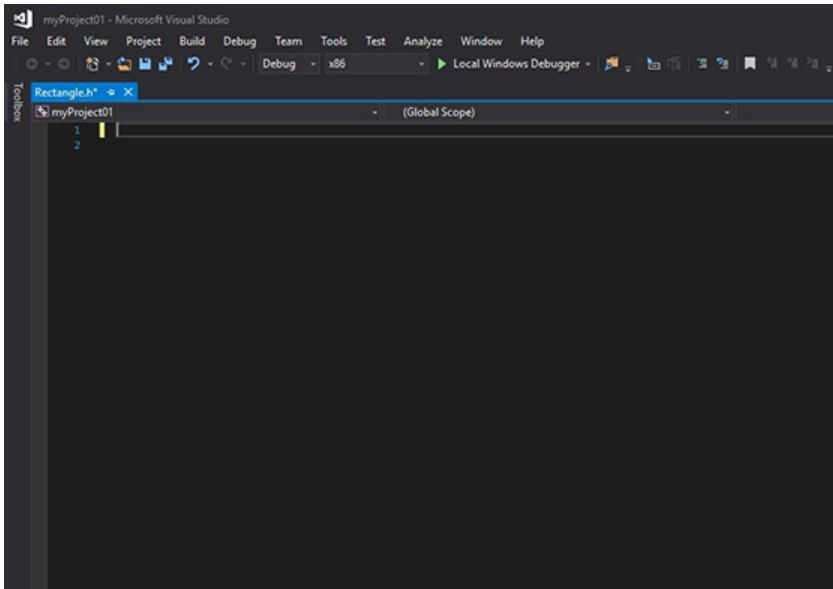
when a project is open, In Solution Explorer, right click on Header File (.h) -> Add -> New Item



Following window opens. In this window, select Header File (.h) and name the file as *Rectangle*, then click on "Add":



Now, a blank header file, named as *Rectangle.h*, is open as following: (Delete the preloaded code if there is one)



Inside of this header file we will write a C++ class for creating rectangle objects. Now, following the steps below:

Step 1: Name the class as Rectangle and separate code of the class with two sections, private and public shown as follows:

```
1 #include<iostream>
2 using namespace std;
3
4 class Rectangle
5 {
6     private:
7
8     public:
9
10 };
```

Step 2: Declare two data members under the private section and declare six member functions in the public section as shown below:

```
1 #include<iostream>
2 using namespace std;
3
4 class Rectangle
5 {
6     private:
7         double length;
8         double width;
9     public:
10         void setLength(double len);
11         void setWidth(double wid);
12         double getLength();
13         double getWidth();
14         double getArea();
15         void display();
16 };
```

Two data members specify the attributes of a rectangle and member functions specify the functionalities of the class *Rectangle*. This portion of the class is called **class interface** or **class definition**.

Now, we continue the header file with the implementations of those member functions specified in the class interface. Implementing a member function is to code the function based on its functionality. For example, the first member function is for setting the numerical value for the length of a rectangle, we implement this function as follows:

```
1 #include<iostream>
2 using namespace std;
3
4 class Rectangle
5 {
6 private:
7     double length;
8     double width;
9 public:
10    void setLength(double len);
11    void setWidth(double wid);
12    double getLength();
13    double getWidth();
14    double getArea();
15    void display();
16 };
17 //----Implementation-----
18 void Rectangle::setLength(double len)
19 {
20     length=len;
21 }
```

This function has no return value, we specify the function with *void*. This function is one of the member functions of the class *Rectangle*, we place the code *Rectangle::* in front of the header of the function. The function has only one input specified by *double len*. This function contains a single line of the code: *length=len*; which is setting the value of the *length* to the input value of the function *len*.

Similarly, we implement the rest of the member functions as following:

```
1 #include<iostream>
2 using namespace std;
3
4 class Rectangle
5 {
6 private:
7     double length;
8     double width;
9 public:
10    void setLength(double len);
11    void setWidth(double wid);
12    double getLength();
13    double getWidth();
14    double getArea();
15    void display();
16 };
17 //----Implementation-----
18 void Rectangle::setLength(double len)
19 {
20     length=len;
21 }
22 void Rectangle::setWidth(double wid)
23 {
24     width=wid;
25 }
26 double Rectangle::getLength()
27 {
28     return length;
29 }
30 double Rectangle::getWidth()
31 {
32     return width;
33 }
34 double Rectangle::getArea()
35 {
36     return length*width;
37 }
38 void Rectangle::display()
39 {
40     for(int i=0;i<width;i++){
41         for(int j=0;j<length;j++){
42             if(i==0 or i==width-1 or j==0 or j==length-1){
43                 cout<<"* ";
44             }
45             else{
46                 cout<<" ";
47             }
48         }
49         cout<<endl;
50     }
51 }
```

In following program, we demonstrate how to use the class *Rectangle* to create objects and how to call the member functions of the class *Rectangle*.

```
1 #include<iostream>
2 #include"Rectangle.h"
3 using namespace std;
4
5 int main()
6 {
7     Rectangle myRec;
8     myRec.setLength(10);
9     myRec.setWidth(7);
10    myRec.display();
11    cout<<"The area of the rectangle myRec is "<<myRec.getArea()<<endl;
12    return 0;
13 }
```

Line 2: `#include"Rectangle.h"` links the header file *Rectangle.h* to the program.

Line 7: Here we use the class *Rectangle* to create an object *myRec*. In other words, *myRec* is an instance of the class *Rectangle*. *myRec* posses all the functions we specified in the class definition.

Line 8 and Line 9: We are calling *setLength()* and *setWidth()* functions that come with the instance *myRec* to set the numerical values for the *length* and the *width* of *myRec*.

Line 10: We are calling the member function of *myRec.display()* to visualize the instance *myRec*.

Line 11: We are calling member function *getArea()* to calculate and return the value of the area of the instance *myRec*.

Following is the screen shot of a test run of the program:

```
* * * * *
*           *
*           *
*           *
*           *
*           *
* * * * *
The area of the rectangle myRec is 70
Program ended with exit code: 0
```

This page titled [8.1: Introduction to Object Oriented Programming \(PC Users\)](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

8.2: Introduction to Object Oriented Programming (Mac Users)

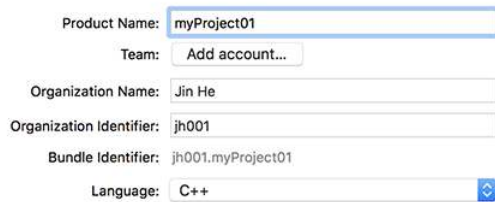
Object-oriented programming (OOP) is a computer programming approach that uses classes and objects. In object-oriented programming, we first create classes and then use classes to create objects. Such objects possess certain features and functions that can be used in a program.

As an example, here we create a class called *Rectangle* and then we use this class to create objects in a program.

To create a class, we will need to open a header file (a file with .h as its file extension)

To create a header file:

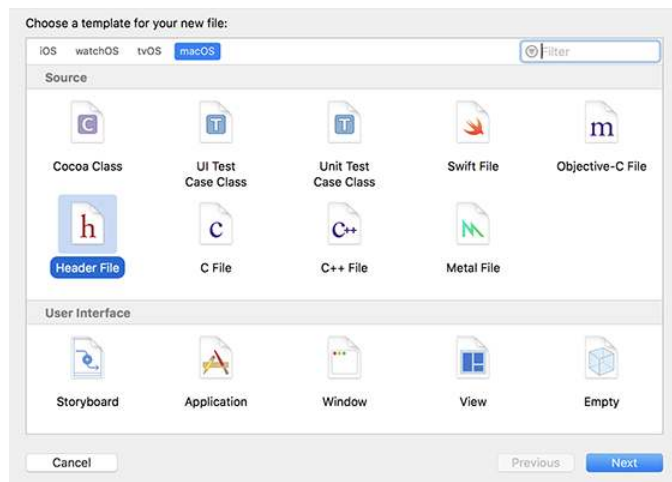
1. Create a project



The image shows the Xcode Project Creation dialog. The 'Product Name' field is filled with 'myProject01'. The 'Team' field has a button 'Add account...'. The 'Organization Name' field is filled with 'Jin He'. The 'Organization Identifier' field is filled with 'jh001'. The 'Bundle Identifier' field is filled with 'jh001.myProject01'. The 'Language' dropdown menu is set to 'C++'.

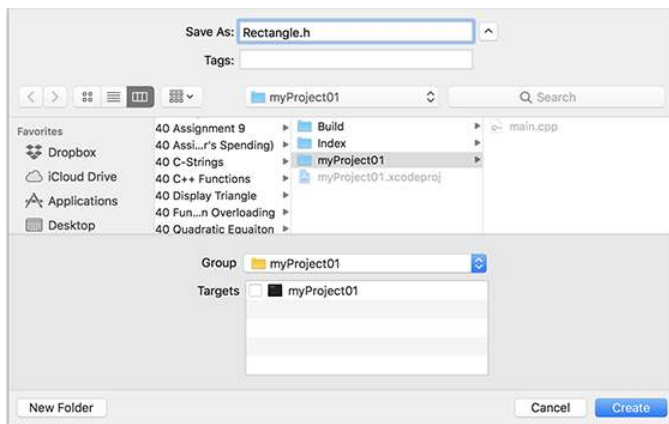
After the project is created: Select File -> New -> File from Template...

Select "Header File" and then click on "Next"



Save As: *Rectangle.h*

Select the folder where the project was created and then click on "Create" (make sure the header file *Rectangle.h* is saved in the project folder)



Now, a blank header file, named as *Rectangle.h*, is created with preloaded code as following:

```

1 //
2 // Rectangle.h
3 // myProject01
4 //
5 // Created by Jin He on 4/19/19.
6 // Copyright © 2019 Jin He. All rights reserved.
7 //
8
9 #ifndef Rectangle_h
10 #define Rectangle_h
11
12
13 #endif /* Rectangle_h */
14

```

Delete the preloaded code, and begin with code of the class *Rectangle*:

Step 1: Name the class as *Rectangle* (Line 4) and separate code of the class with two sections, *private* (Line 6) and *public* (Line 8) shown as following:

```

1 #include<iostream>
2 using namespace std;
3
4 class Rectangle
5 {
6 private:
7
8 public:
9
10 };
11

```

Step 2: Declare two data members under the private section (Line 7 and Line 8) and declare six member functions in the public section (Line 10 - Line 15) as shown below:

```

1 #include<iostream>
2 using namespace std;
3
4 class Rectangle
5 {
6 private:
7     double length;
8     double width;
9 public:
10    void setLength(double len);
11    void setWidth(double wid);
12    double getLength();
13    double getWidth();
14    double getArea();
15    void display();
16 };

```

Two data members specify the attributes of a rectangle and member functions specify the functionalities of the class *Rectangle*. This portion of the class is called **class interface** or **class definition**.

Now, we continue the header file with the implementations of those member functions specified in the class interface. Implementing a member function is to code the function based on its functionality. For example, the first member function is for setting the numerical value for the length of a rectangle, we implement this function as follows:

```
1  #include<iostream>
2  using namespace std;
3
4  class Rectangle
5  {
6  private:
7      double length;
8      double width;
9  public:
10     void setLength(double len);
11     void setWidth(double wid);
12     double getLength();
13     double getWidth();
14     double getArea();
15     void display();
16 };
17 //-----Implementation-----
18 void Rectangle::setLength(double len)
19 {
20     length=len;
21 }
```

This function has no return value, we specify the function with *void*. This function is one of the member functions of the class *Rectangle*, we place the code *Rectangle::* in front of the header of the function. The function has only one input specified by *double len*. This function contains a single line of the code: *length=len;* which is setting the value of the *length* to the input value of the function *len*.

Similarly, we implement the rest of the member functions as following:

```
1 #include<iostream>
2 using namespace std;
3
4 class Rectangle
5 {
6 private:
7     double length;
8     double width;
9 public:
10    void setLength(double len);
11    void setWidth(double wid);
12    double getLength();
13    double getWidth();
14    double getArea();
15    void display();
16 };
17 //-----Implementation-----
18 void Rectangle::setLength(double len)
19 {
20     length=len;
21 }
22 void Rectangle::setWidth(double wid)
23 {
24     width=wid;
25 }
26 double Rectangle::getLength()
27 {
28     return length;
29 }
30 double Rectangle::getWidth()
31 {
32     return width;
33 }
34 double Rectangle::getArea()
35 {
36     return length*width;
37 }
38 void Rectangle::display()
39 {
40     for(int i=0;i<width;i++){
41         for(int j=0;j<length;j++){
42             if(i==0 or i==width-1 or j==0 or j==length-1){
43                 cout<<"* ";
44             }
45             else{
46                 cout<<"  ";
47             }
48         }
49         cout<<endl;
50     }
51 }
```

In following program, we demonstrate how to use the class *Rectangle* to create objects and how to call the member functions of the class *Rectangle*. (Please make sure to place the header file *Rectangle.h* in the project folder)

```
1 #include<iostream>
2 #include"Rectangle.h"
3 using namespace std;
4
5 int main()
6 {
7     Rectangle myRec;
8     myRec.setLength(10);
9     myRec.setWidth(7);
10    myRec.display();
11    cout<<"The area of the rectangle myRec is "<<myRec.getArea()<<endl;
12    return 0;
13 }
```

Line 2: `#include"Rectangle.h"` links the header file *Rectangle.h* to the program.

Line 7: Here we use the class *Rectangle* to create an object *myRec*. In other words, *myRec* is an instance of the class *Rectangle*. *myRec* posses all the functions we specified in the class definition.

Line 8 and Line 9: We are calling *setLength()* and *setWidth()* functions that come with the instance *myRec* to set the numerical values for the *length* and the *width* of *myRec*.

Line 10: We are calling the member function of *myRec.display()* to visualize the instance *myRec*.

Line 11: We are calling member function *getArea()* to calculate and return the value of the area of the instance *myRec*.

Following is the screen shot of a test run of the program:

```
* * * * *
*           *
*           *
*           *
*           *
*           *
* * * * *
The area of the rectangle myRec is 70
Program ended with exit code: 0
```

This page titled [8.2: Introduction to Object Oriented Programming \(Mac Users\)](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

8.3: Class Constructors

What is a class constructor and what is it for?

To answer those questions, we have to go back to "variable" first. Given a `main()` function as following:

```
int main()
{
    int a;
    a = 5;
    .....
}
```

In the program, the statement "`int a;`" declares an integer variable ***a***, and the statement "`a = 5;`" assigns an initial value 5 to the variable ***a***. We use two statements to declare and initialize an integer variable in this program. Actually, we can use a single statement to declare and initialize a variable at the same time, such as "`int a = 5;`"

Similarly, in the following program:

```
int main()
{
    Rectangle rectangleA;
    rectangleA.setLength(7);
    rectangleA.setWidth(5);
    .....
}
```

the statement "`Rectangle rectangleA;`" declares an instance of the class `Rectangle`, and then the statements : "`rectangleA.setLength(7);`" and "`rectangleA.setWidth(5);`" set the initial values for the *length* and the *width* of the `rectangleA`. Now the question is "Can we use a single statement that declares and initializes the values of the *length* and the *width* of an object of rectangle like we did with the integer variable?"

The answer is "Yes", such statement should be looked like "`Rectangle rectangleA(7,5);`" However, we can't use such statement in our program now unless we add a special member function in the definition of the class `Rectangle`. We specify such function in the definition of the class `Rectangle` by using following statement:

```
Rectangle(double l, double w);
```

and then code this function in implementation section as following:

```
Rectangle::Rectangle(double l, double w)
{
    length=l;
    width=w;
}
```

```
1 //Header File for Class Rectangle
2 #ifndef Rectangle_h
3 #define Rectangle_h
4
5 #include<iostream>
6 using namespace std;
7 class Rectangle
8 {
9 private:
10     double length;
11     double width;
12 public:
13     Rectangle(double l, double w);
14     void setLength(double l);
15     void setWidth(double w);
16     double getLength();
17     double getWidth();
18     void display();
19 };
20
21 //Implementations
22 Rectangle::Rectangle(double l, double w)
23 {
24     length = l;
25     width = w;
26 }
27 void Rectangle::setLength(double l)
28 {
29     length = l;
30 }
```

The reason why we say it is a special member function is because of

1. The name of this function must be the same as the name of the class
2. Even this function has no return value, we do not place *void* in front of the name of the function.
3. This function can only be invoked when we declare an object of *Rectangle*.

After add this member function to the header file *Rectangle.h* we can use following statement to declare (create) an object of *Rectangle*/an instance of the class *Rectangle* and initialize the *length* and the *width* of this rectangle at the same time:

```
Rectangle rectangleA(7,5);
```

This special member function is called "**Class Constructor**" To be more specific, we call this function "**Non-default Constructor**"

In the following program, we use this statement at Line 7 to declare an instance of the class *Rectangle* called *rectangleA* and initialize the *length* and the *width* of *rectangleA* to 7 and 10,

```
Source.cpp* x Rectangle.h*
myProject01 (Global Scope)
1 #include<iostream>
2 #include"Rectangle.h"
3 using namespace std;
4
5 int main()
6 {
7     Rectangle rectangleA(7, 11);
8     rectangleA.display();
9     system("pause");
10    return 0;
11 }
```

Following is the screen shot of a test run of the program:

```
* * * * *
*           *
*           *
*           *
*           *
*           *
* * * * *
The area of the rectangle myRec is 70
Program ended with exit code: 0
```

We can see the length and the width of the object *rectangleA* were initialized by 7 and 11 without using *setLength()* and *setWidth()* functions.

But once we add this member function to the header file, we can't use following statement in our program anymore:

Rectangle rectangleB;

which means we can't just declare an object without provide the initial values for the *length* and the *width* of the rectangle, and sometime it is inconvenient.

If we do want the statement "*Rectangle rectangleB;*" to be a valid statement, in other words, if we do want to allow to declare an object of *Rectangle* without providing the initial values for the length and the width of the rectangle, we can add another special member function, which is called "**Default Constructor**". This function will allow the the *length* and the *width* of a rectangle to take default values.

We specify such function in the definition of the class *Rectangle* by using following statement:

Rectangle();

and add following function definition into the header file in the implementation section:

```
Rectangle::Rectangle()
{
    length=5;
    width=5;
}
```

where 5 is the default value for the length and the width, and it can be set by you with any value you prefer.

```

1  #include<iostream>
2  using namespace std;
3
4  class Rectangle
5  {
6  private:
7      double length;
8      double width;
9  public:
10     Rectangle();
11     Rectangle(double l, double w);
12     void setLength(double len);
13     void setWidth(double wid);
14     double getLength();
15     double getWidth();
16     double getArea();
17     void display();
18 };
19 //-----Implementation-----
20 Rectangle::Rectangle()
21 {
22     length=5;
23     width=5;
24 }
25 Rectangle::Rectangle(double l, double w)
26 {
27     length=l;
28     width=w;
29 }
30 void Rectangle::setLength(double len)
31 {
32     length=len;
33 }
34 void Rectangle::setWidth(double wid)
35 {
36     width=wid;
37 }
38 double Rectangle::getLength()
39 {
40     return length;
41 }
42 double Rectangle::getWidth()
43 {
44     return width;
45 }
46 double Rectangle::getArea()
47 {
48     return length*width;
49 }
50 void Rectangle::display()
51 {
52     for(int i=0;i<width;i++){
53         for(int j=0;j<length;j++){
54             if(i==0 or i==width-1 or j==0 or j==length-1){
55                 cout<<"* ";
56             }
57             else{
58                 cout<<" ";
59             }
60         }
61         cout<<endl;
62     }
63 }

```

default constructor

non-default constructor

In the following program, we can declare two instances of the class Rectangle at Line 7:

```

1  #include<iostream>
2  #include"Rectangle.h"
3  using namespace std;
4
5  int main()
6  {
7      Rectangle rectangleA, rectangleB(10,7);
8      cout<<"The area of rectangleA is "<<rectangleA.getArea()<<endl;
9      cout<<"It looks as following:"<<endl;
10     rectangleA.display();
11     cout<<"===== "<<endl;
12     cout<<"The area of rectangleB is "<<rectangleB.getArea()<<endl;
13     cout<<"It looks as following:"<<endl;
14     rectangleB.display();
15     return 0;

```

We declared *rectangleA* without providing initial value for its length and its width, thus *rectangleA* used the default constructor for setting its length to 5 and its width to 5. On the other hand, we declared *rectangleB* with initial values of its length and its width provided, thus *rectangleB* used its non-default constructor to set the length and the width to 10 and 7 respectively.

Following is a screenshot of a test run of this program:

```
The area of rectangleA is 25
It looks as following:
* * * * *
*       *
*       *
*       *
*       *
* * * * *
=====
The area of rectangleB is 70
It looks as following:
* * * * * * * * *
*               *
*               *
*               *
*               *
*               *
* * * * * * * * *
Program ended with exit code: 0
```

This page titled [8.3: Class Constructors](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

8.4: Case Study- C++ Class- StudentRecord

Here we would like to discuss another example of C++ Class: *StudentRecord*.

The class *StudentRecord* is for creating an object of student record. As a student record, the object may contain following information:

Student last name
Student first name
Student identification number
Student GPA record

Of course, a student record may contain many other information, for simplicity, here we only consider a student record that contains above information.

We will specify four data members in the private section of the definition of the class *StudentRecord*. We will use string variables for holding student's last name, first name, and the id number. We use double variable for holding the GPA record:

```
class StudentRecord
{
private:
string lastName;
string firstName;
string idNumber;
double GPA;

public:
-----
};
```

In the public section, we will first need a constructor which will initialize the values of the data members:

The default constructor:

```
StudentRecord();
```

The non-default constructor:

```
StudentRecord(string last, string first, string iD, double gpa);
```

Then we will need a pair of set and get. functions for each of the data members:

```
void setLastName(string last);
string getLastName();

void setLastFirstName(string first);
string getFirstName();

void setID(string iD);
string getID();

void setGPA(string last);
double getGPA();
```

Finally, we will need a member function that will display the content of a student record:

```
void display();
```

The definition of the class *StudentRecord* is shown as following:

```
1 //
2 // StudentRecord.h
3 // Created by J He on 2019/4/11.
4 // Copyright © 2019 J He. All rights reserved.
5 //
6
7 #ifndef Header_h
8 #define Header_h
9
10 #include<string>
11 #include<iostream>
12 #include<iomanip>
13 using namespace std;
14
15 class StudentRecord
16 {
17 private:
18     string lastName;
19     string firstName;
20     string idNumber;
21     double GPA;
22 public:
23     //Constructors
24     StudentRecord();
25     StudentRecord(string last,string first,string id,double gpa);
26     //Set and Get Functions
27     void setLastName(string last);
28     string getLastName();
29     void setFirstName(string first);
30     string getFirstName();
31     void setID(string id);
32     string getID();
33     void setGPA(double gpa);
34     double getGPA();
35     //Utility Functions
36     void display();
37 };
38
```

Now will need to implement those constructors and member functions. "To implement " means to put C++ statements in those functions to tell what they need to do when they were called in the source file of a program.

First, let us look the default constructor. The default constructor will be called when we create an object of *StudentRecord* without providing any initial values for the data members. The function will simply assign the default values for each data member:

```
StudentRecord::StudentRecord()
{
    lastName="None";
    firstName="None";
    idNumber="----";
    GPA=0;
}
```

Then, we will look at non-default constructor. The non-default constructor will set the input values to each of the data members of the class:

```
StudentRecord::StudentRecord(string last, string first, string id, double gpa)
{
    lastName=last;
    firstName=first;
    idNumber=id;
    GPA=gpa;
}
```

Followings are the set and get functions for each data member:

```
void StudentRecord::setLastName(string last)
{
    lastName=last;
}
```

```
string StudentRecord::getLastName()
{
    return lastName;
}

void StudentRecord::setFirstName(string first)
{
    firstName=first;
}

string StudentRecord::getFirstName()
{
    return firstName;
}

void StudentRecord::setID(string iD)
{
    idNumber=iD;
}

string StudentRecord::getID()
{
    return idNumber;
}

void StudentRecord::setGPA(double gpa)
{
    GPA=gpa;
}

double StudentRecord::getGPA()
{
    return GPA;
}
```

Finally, we look at the display function. This function simply displays each data member of the class on the screen:

```
void StudentRecord::display()
{
    cout<<"===== "<<endl;
    cout<<setw(20)<<"Student Last Name:"<<setw(10)<<lastName<<endl;
    cout<<setw(20)<<"Student First Name:"<<setw(10)<<firstName<<endl;
    cout<<setw(20)<<"Student ID:"<<setw(10)<<idNumber<<endl;
    cout<<setw(20)<<"Student GPA:"<<setw(10)<<GPA<<endl;
    cout<<"===== "<<endl;
}
```

We will place above implementations in the header file *StudentRecord.h* below the class interface:

```
//Implementation of Member Functions
StudentRecord::StudentRecord()
{
    lastName="None";
    firstName="None";
    idNumber="-----";
    GPA=0;
}

StudentRecord::StudentRecord(string last, string first, string iD, double gpa)
{
    lastName=last;
    firstName=first;
    idNumber=iD;
    GPA=gpa;
}

void StudentRecord::setLastName(string last)
{
    lastName=last;
}

string StudentRecord::getLastName()
{
    return lastName;
}

void StudentRecord::setFirstName(string first)
{
    firstName=first;
}

string StudentRecord::getFirstName()
{
    return firstName;
}
```

```
void StudentRecord::setID(string iD)
{
    idNumber=iD;
}

string StudentRecord::getID()
{
    return idNumber;
}

void StudentRecord::setGPA(double gpa)
{
    GPA=gpa;
}

double StudentRecord::getGPA()
{
    return GPA;
}

void StudentRecord::display()
{
    cout<<left<<endl;
    cout<<"===== "<<endl;
    cout<<setw(30)<<"Student Last Name:"<<setw(10)<<lastName<<endl;
    cout<<setw(30)<<"Student First Name:"<<setw(10)<<firstName<<endl;
    cout<<setw(30)<<"Student ID:"<<setw(10)<<idNumber<<endl;
    cout<<setw(30)<<"Student GPA:"<<setw(10)<<GPA<<endl;
    cout<<"===== "<<endl;
}

#endif /* Header_h */
```

Since we use "cout" and set() functions in our implementations, will need to place following lines at the beginning of the header file:

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

After we correctly create the header file with above class in the project, we can open a source file and use the class *StudentRecord* to create objects in the main() function:

```
1 #include<iostream>
2 #include"StudentRecord.h"
3 using namespace std;
4
5 int main()
6 {
7     StudentRecord recordA;
8     StudentRecord recordB("He", "Jin", "0000",3.0);
9     recordA.display();|
10    recordB.display();
11
12    return 0;
13 }
```

Line 7: We declare an instance of the class *StudentRecord*, *recordA* ,without providing initial values. The default constructor will be called here.

Line 8: We declare an instance of the class *StudentRecord*, *recordB*, with the initial values provided

Line 9 and Line 10: We call the member function *display()* of each object to display the *recordA* and *recordB* in the output window.

Following is the screenshot of a sample run of the program:

```
=====
Student Last Name:      None
Student First Name:    None
Student ID:            -----
Student GPA:           0
=====
=====
Student Last Name:      He
Student First Name:    Jin
Student ID:            0000
Student GPA:           3
=====
```

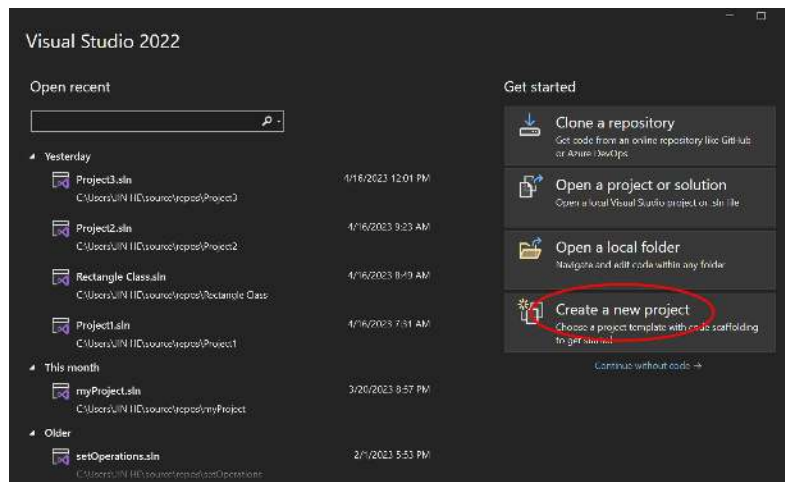
This page titled [8.4: Case Study- C++ Class- StudentRecord](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

8.5: Case Study- C++ Class- Bag

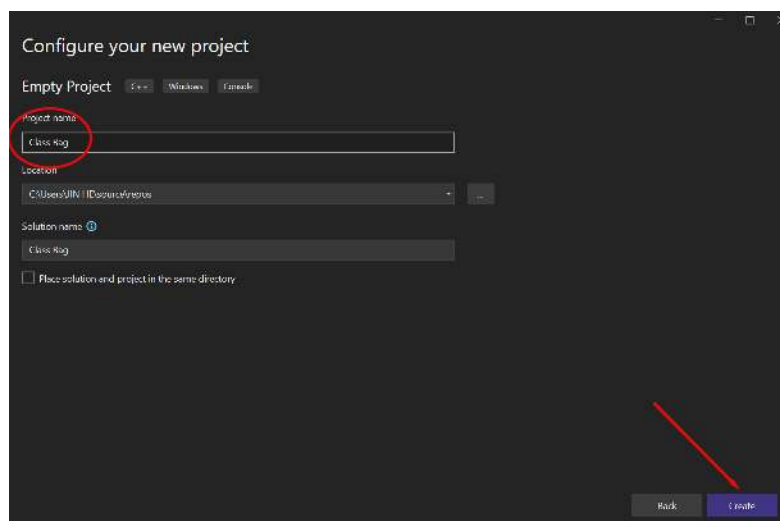
In this section, we will walk through another example of C++ class: *Bag*. This is a more practical class. We will also demonstrate how to use this class in a program to create an object and how to use the object for solving some programming problem.

Let's get started on writing this class.

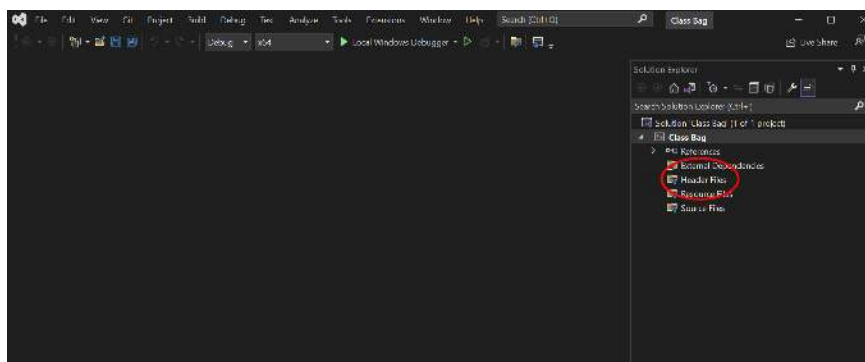
Step 1: Open an project:



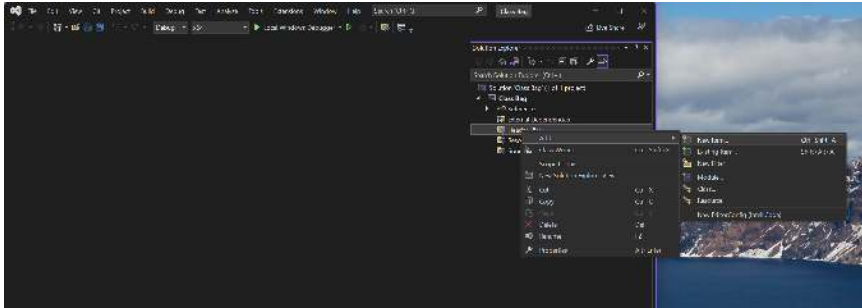
Name the project and then click on "Create":



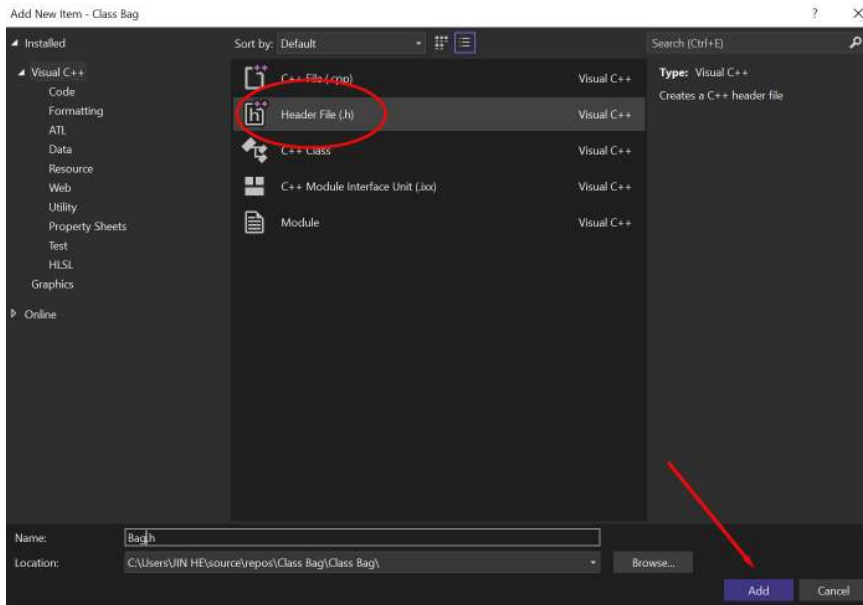
Step 2: Create a header file for the class Bag. In the solution explorer, right click on "Header Files"



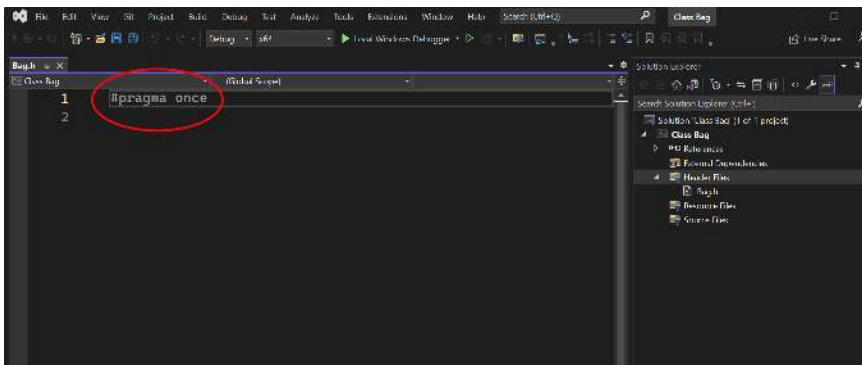
Select Add->New Item...



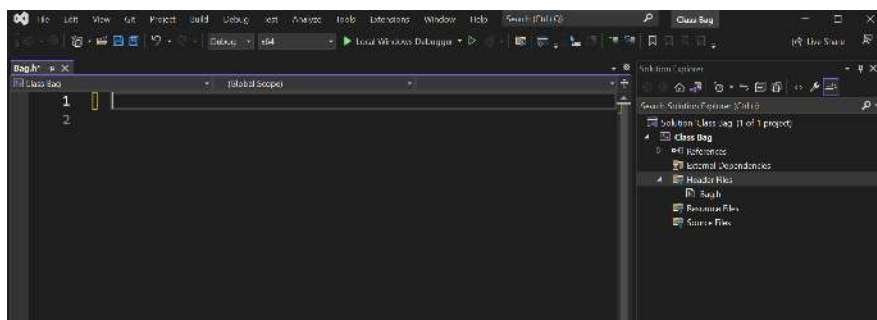
Select "Header File (.h), then rename the file as "Bag.h", then click on "Add"



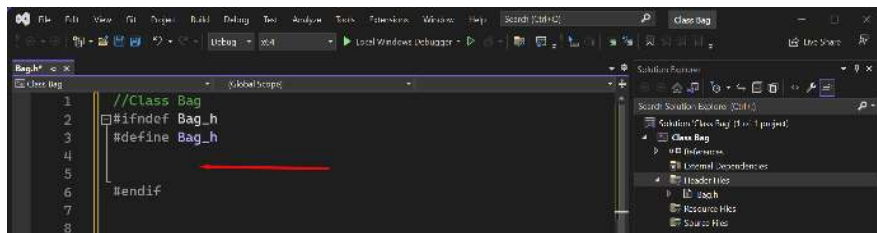
A header file is open. Delete the preload code:



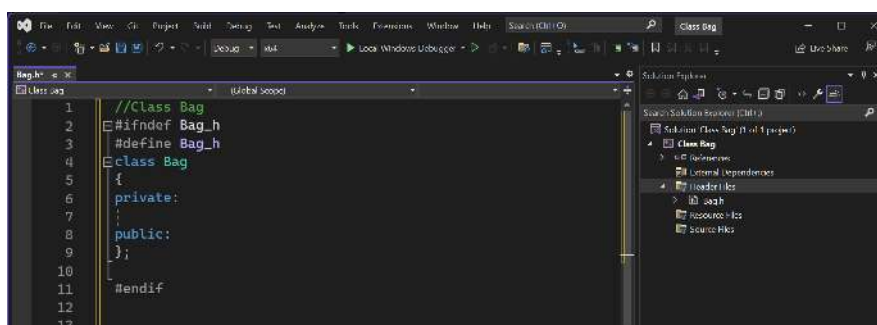
Now, we have a blank header file created with name "Bag.h":



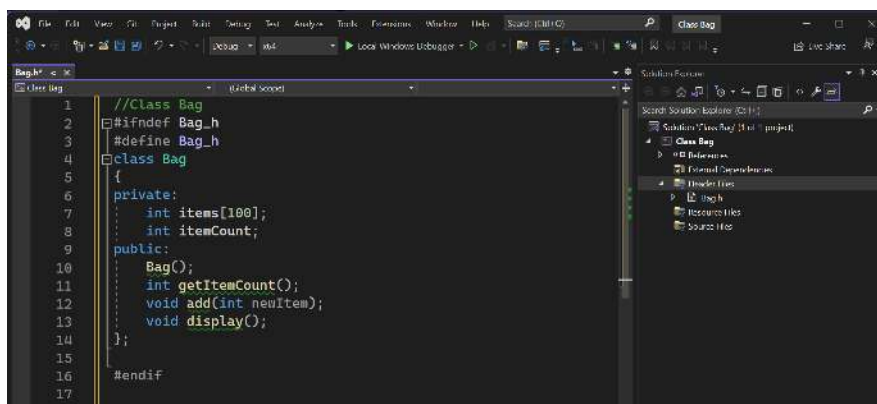
Step 3: Coding for the class Bag. Setup the class Bag with following three lines of code. The code of the class Bag will be placed in between line "#define Bag_h" and the line "#endif"



We name the class as *Bag* and separate the code of the class Bag into two sections: *private* and *public*:



Declare two data members in the private section and declare four member functions (including a default constructor) in the public section:



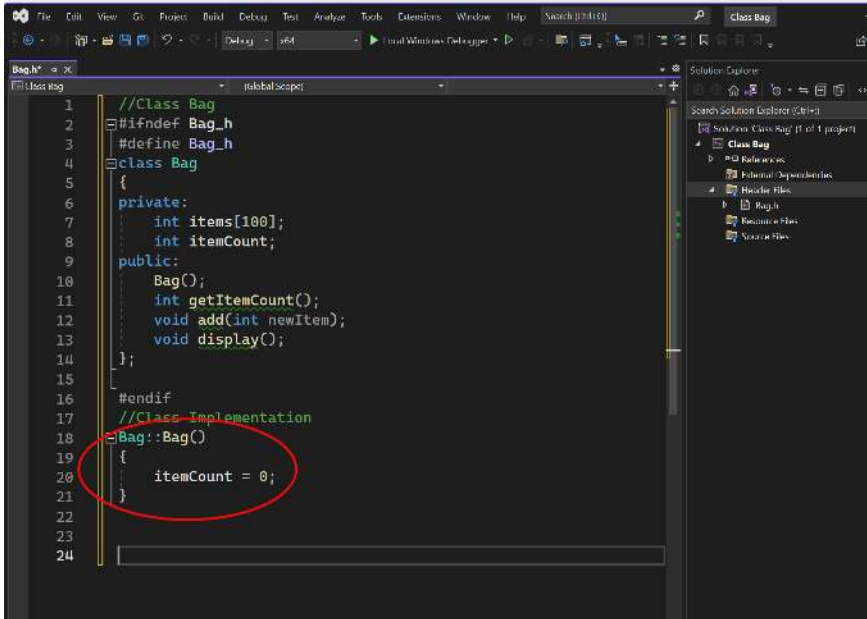
This is the class interface of the class Bag.

Line 7: The first data member is an integer array with 100 elements.

Line 8: The second data member is an integer variable for storing the counts of the data in the array *items*. We will be using this data member in the following member functions.

Step 4: Coding the class constructor.

The code of the constructor of the class Bag is given as following:



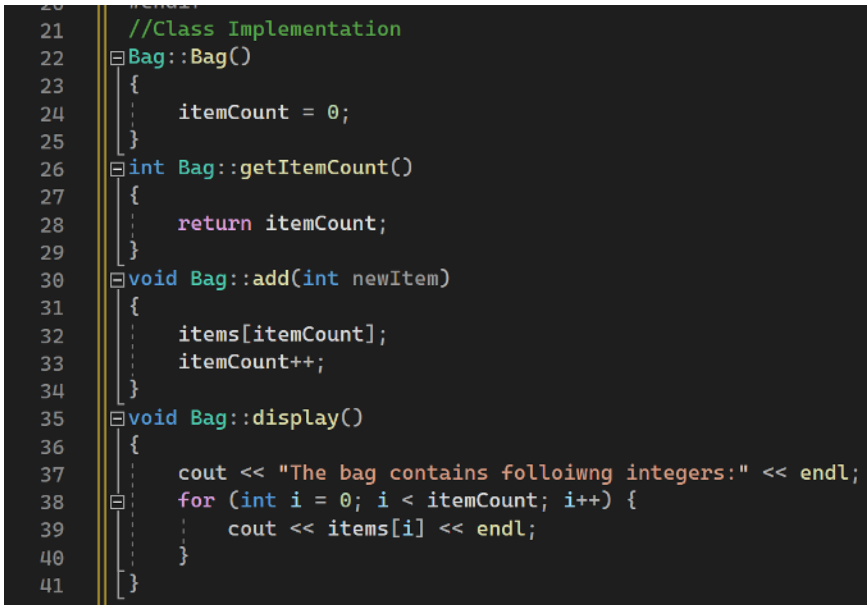
```

1 //Class Bag
2 #ifndef Bag_h
3 #define Bag_h
4 class Bag
5 {
6 private:
7     int items[100];
8     int itemCount;
9 public:
10    Bag();
11    int getItemCount();
12    void add(int newItem);
13    void display();
14 };
15
16 #endif
17 //Class Implementation
18 Bag::Bag()
19 {
20     itemCount = 0;
21 }
22
23
24

```

This is a default constructor, it sets the value of the data member *itemCount* to 0. It is called when an instance of the class Bag is created.

And the rest of the member functions will be coded as following:



```

21 //Class Implementation
22 Bag::Bag()
23 {
24     itemCount = 0;
25 }
26 int Bag::getItemCount()
27 {
28     return itemCount;
29 }
30 void Bag::add(int newItem)
31 {
32     items[itemCount];
33     itemCount++;
34 }
35 void Bag::display()
36 {
37     cout << "The bag contains folloiwng integers:" << endl;
38     for (int i = 0; i < itemCount; i++) {
39         cout << items[i] << endl;
40     }
41 }

```

Line 26 - Line 29: The member function *getItemCount()* returns the value of the data member *itemCount*

Line 30 - Line 34: The member function *add()* adds a new data item to the array *items*, which means adding a new item into the bag. The function puts the new item at the end of the array *items*.

We put all those code in the header file *Bag.h*

Now, we discuss how to add additional member functions to this class.

Example

We want to add a member function called `contains()`. This function will receive an integer and check if this integer is in the bag. If it is in the bag, return `true`, otherwise return `false`.

Step 1: Add function declaration in the class interface as following:

```
1 //Class Bag
2 #ifndef Bag_h
3 #define Bag_h
4
5 #include<iostream>
6 using namespace std;
7
8 class Bag
9 {
10 private:
11     int items[100];
12     int itemCount;
13 public:
14     Bag();
15     int getItemCount();
16     void add(int newItem);
17     void display();
18     bool contains(int anItem);
19 };
20
21 #endif
22 //Class Implementation
23 Bag::Bag()
24 {
25     itemCount = 0;
26 }
27 int Bag::getItemCount()
28 {
```

Step 2: Go to the bottom of the header file and set up the function as following:

```
13 public:
14     Bag();
15     int getItemCount();
16     void add(int newItem);
17     void display();
18     bool contains(int anItem);
19 };
20
21 #endif
22 //Class Implementation
23 Bag::Bag()
24 {
25     itemCount = 0;
26 }
27 int Bag::getItemCount()
28 {
29     return itemCount;
30 }
31 void Bag::add(int newItem)
32 {
33     items[itemCount];
34     itemCount++;
35 }
36 void Bag::display()
37 {
38     cout << "The bag contains following integers:" << endl;
39     for (int i = 0; i < itemCount; i++) {
40         cout << items[i] << endl;
41     }
42 }
43 bool Bag::contains(int anItem)
44 {
45 }
46
47
```

Step 3: Using a looping statement to check the integers in the bag to see if there is a match. We use `i < itemCount` as the looping continuation condition to stop the loop when we have checked all the data that are in the array and no need to check the rest part of the array that has no data in it. If we got a match (`items[i] == anItem`) then break the loop and return Boolean value `true` to complete the function call. Upon the completion of the looping statement, if no match is found, then return Boolean value `false` and terminate the function:

```

19 };
20
21 #endif
22 //Class Implementation
23 Bag::Bag()
24 {
25     itemCount = 0;
26 }
27 int Bag::getItemCount()
28 {
29     return itemCount;
30 }
31 void Bag::add(int newItem)
32 {
33     items[itemCount];
34     itemCount++;
35 }
36 void Bag::display()
37 {
38     cout << "The bag contains folloiwng integers:" << endl;
39     for (int i = 0; i < itemCount; i++) {
40         cout << items[i] << endl;
41     }
42 }
43 bool Bag::contains(int anItem)
44 {
45     for (int i = 0; i < itemCount; i++) {
46         if (items[i] == anItem)
47             return true;
48     }
49     return false;
50 }
51

```

Step 4: Write a source file for testing the function:

```

1 #include<iostream>
2 #include"Bag.h"
3 using namespace std;
4
5
6
7 int main()
8 {
9     Bag myBag;
10    for (int i = 0; i < 10; i++)
11        myBag.add(rand() % 100);
12    myBag.display();
13    //===Testing contains() function===//
14    if (myBag.contains(60))
15        cout << "Yes, it is in the bag!" << endl;
16    else {
17        cout << "No, it is not in the bag." << endl;
18    }
19    return 0;
20 }

```

Line 7: Use the class Bag to create an instance *myBag*. The default constructor will set the variable *itemCount* of *myBag* to 0.

Line 8 - Line 9: We use a *for* loop statement and the random function *rand()* to randomly generate 10 integers from 0 to 99 and put them into the object *myBag*.

Line 10: Call the member function of *myBag* to display the data we just put into the bag.

Line 12 - Lind 16: Use an if-else statement to check the return value of the member function *myBag.contains()* to see if the integer 60 is one of numbers in the object *myBag*.

Please code this program into your computer and do a test run.

This page titled [8.5: Case Study- C++ Class- Bag](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

CHAPTER OVERVIEW

9: Pointers

[9.1: What is a Pointer?](#)

[9.2: Pointer and Dynamic Memory Allocation](#)

[9.3: Case Study- Array Copy](#)

9: Pointers is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

9.1: What is a Pointer?

Introduction to C++ Pointer

In C++ programming, a pointer is a special variable that can only be used for holding an address of a memory cell.

What is an address?

In a C++ program, when you use following statement to declare a variable:

```
int a;
```

or

```
char a;
```

you allocate a piece of memory cell on the computer memory card to your program so you can store a data (either an integer or a character) in such memory cell.

There are three types of information that are associated with each memory cell:

1. the name (such as *a* in above example)
2. content (the data stored in the memory cell, sometimes it may not have a content, such as the one in above example)
3. address which is the physical address of such memory cell to indicate where the memory cell is located on the computer memory card.

A memory cell on a computer memory card is like a building in a city. Each building in a city have three type of information:

1. The name of the building (such as Walmart, CVS and so on)
2. The content (what is inside of the building)
3. Street address of the building (such as 1101 E. University Ave)

A pointer is a variable that is specifically for storing the address of memory cell used in a computer program.

Now, let us get familiar with pointer.

In a program, we may use following statement to declare pointer variable:

```
int *p;
```

where *p* is the name of the pointer, it represents a memory cell that can be used for storing the address of an integer data. Similarly,

```
char *ch;
```

declares a pointer *ch* that represents a memory cell which can be used for storing the address of a character data

Once a pointer is declared in a program, we can use it to store the address of a variable:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int *ptr, a=5;
7     ptr=&a;
8     cout<<"The address of the variable a is "<<ptr<<endl;
9     return 0;
10 }
```

Line 6: we declare an integer pointer *ptr* and a regular variable *a* with the initial value 5.

Line 7: '&' is an extract operator which extracts the address of the variable *a* and the statement, *ptr=&a*; assignd the address of *a* to the pointer *ptr*.

Line 8: this statement displays the address of the variable *a*:

```
The address of the variable a is 0x16fdfee9c
Program ended with exit code: 0
```

Once a point hold an address of a variable in a program, such as *ptr* is holding the address of the variable *a* in above program, then **ptr* is same as the variable *a*, which means **ptr* and *a* represent the same memory cell. We use following program to explain this:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int *ptr, a=5;
7     ptr=&a;
8     *ptr=15;
9     cout<<"The value of the variable a is "<<a<<endl;|
10    return 0;
11 }
```

Since we assign the address of *a* to the pointer *ptr* in Line 7, the *ptr* is holding the address of the variable *a*, thus, **ptr* is the same as the variable *a*. In Line 8, we assign integer 15 to **ptr*, this is the same as assigning 15 to the variable *a*. Therefore, when we use a *cout* statement to display the value of the variable *a*, we should see its value has been changed to 15, following is a screenshot of the output of the program:

```
The value of the variable a is 15
Program ended with exit code: 0
```

To help you to better understand points, let's study following sample program:

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6     int *ptr1, *ptr2, a=5, b=20;
7     ptr1=&a;
8     ptr2=&b;
9     ptr2=ptr1;
10    *ptr2=10;
11    cout<<"The value of the variable a is "<<a<<endl;
12    cout<<"The value of the variable b is "<<b<<endl;
13    return 0;
14 }
```

Line 6: We declared two pointers and two variables and initialized the variables to 5 and 20 respectively.

Line 7 and Line 8: We assigned the addresses of the variables *a* and *b* to pointer *ptr1* and *ptr2* respectively.

Line 9: We copy the address in the pointer *ptr1* into pointer *ptr2*, thus, at this point, both *ptr1* and *ptr2* are holding the address of the variable *a* (both *ptr1* and *ptr2* are pointing to *a*). Therefore both **ptr1* and **ptr2* are the same as *a*.

Line 10: We assigned integer 10 to **ptr2*, which is the same as assigned 10 to *a*.

Line 11: Display the value of *a* on the output screen, which is 10

Line 12: Display the value of b on the output screen, which is 20, because after Line 9, $ptr2$ is no longer pointing to b , instead, it is pointing to a . Therefore, the statement in Line 10 did not alter the value of the variable b , instead, it alter the value of a from 5 to 10, that is why the value of b is remained as 20. Following is the screenshot of output of the program:

```
The value of the variable a is 10
The value of the variable b is 20
Program ended with exit code: 0
```

This page titled [9.1: What is a Pointer?](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

9.2: Pointer and Dynamic Memory Allocation

Why do we need a pointer? Why do we need to store the address of a memory cell? There are no simple answers to those questions. We may need to take next programming class to fully understand the power of C++ pointers. Here, we just discuss one of the applications of pointers.

Memory Dynamical Allocation

In a C++ program, we often need to declare a variable or an array for storing data.

```
int a;
```

```
int myArray[1000];
```

No matter where we put those declaration statements in the program, the memory cell that *a* or *myArray* represent will be locked to the program as soon as the program is built even the program may not use those memory cells until the end of the program. If this program runs for days or weeks, then those memory cells will be locked the whole time and no other program or apps can use them, this is going to be a waste of computer resources. Is there any way that we can wait until we actually need those memory cells to declare/allocate them? The answer is YES. We can use following statements to dynamically allocate a single memory cell or an array of memory cells:

```
int *p;
```

```
p = new int;
```

or

```
int *p;
```

```
p = new int[1000];
```

We first need to declare a pointer *p*, then use a "new-statement", *p = new int*; this statement **dynamically** allocate an integer memory cell to the program and send the address of this memory cell to the pointer *p*. So pointer *p* is holding the address of this memory cell. And this won't happen until the *new-statement* is executed. If we don't need this memory cell toward to the end of the program, we can use such *new-statement* to allocate a memory cell at the time when we need it.

Once *p* is holding the address of this memory, we can use following statement to assign an integer into this memory:

```
*p = 20;
```

or we can use following statement to display the integer in the memory:

```
cout<<*p<<endl;
```

Similarly, We can also use a *new-statement* to dynamically allocate an array of memory cells to a program:

```
int *p;
```

```
p = new int[1000];
```

We can even allow a user to determine the size of the array:

```
int length;
```

```
int *p;
```

```
cout<<"Please enter the length of the array: ";
```

```
cin>>length;
```

```
p = new int[length];
```

Once we use a pointer and a new-statement to allocate an array, the address of the first element of the array will be assigned into the pointer. We can use the name of the pointer as the name of the array, for example, once we use above statements to declare an integer array, the address of the first element is assigned into the pointer *p*, and we can use *p* as the name of the array, thus we can use following looping statement to store data into this array:

```
for(int i=0; i< length; i++){  
    cout<<"Please enter an integer:\n";  
    cin>>p[i];  
}
```

We can also use following statement to display the data stored in the array:

```
for(int i=0; i< length; i++){  
    cout<<p[i]<<endl;  
}
```

9.2: Pointer and Dynamic Memory Allocation is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

9.3: Case Study- Array Copy

In this study case, we demonstrate another application of pointer in C++ programming.

Example

Write a function that receives an character array (a C-string) through its parameter. The function makes a copy of the array it receives and then changes all the characters in the copy to their upper case counterparts. The function returns the copy of the array without altering the data in the original array it receives.

Here is the code of the function along with *main()*, which is for testing purposes:

```
1 #include<iostream>
2 using namespace std;
3
4 char *capitalize(char a[])
5 {
6     int length=0;
7     int i=0;
8     while(a[i]!='\0'){
9         length++;
10        i++;
11    }
12    char *copy=new char[length];
13    for(int i=0;i<length;i++){
14        copy[i]=a[i];
15        copy[i]=toupper(copy[i]);
16    }
17    return copy;
18 }
19
20 int main()
21 {
22     char a[]="I am student in Computer Science";
23     char *cap;
24     cap=capitalize(a);
25     cout<<"The original string is:"<<endl;
26     cout<<a<<endl;
27     cout<<"The string the function returns is:"<<endl;
28     cout<<cap<<endl;
29     return 0;
30 }
```

Line 4: This is the header of the function. We specify the return of the function is a character pointer by placing '*' in front of the name of the function. The parameter of the function is *char a[]*, which indicates the input of the function is a character array (a C-string).

Line 6 - Line 11: We use a while loop to count the number of the characters in the array the function receives which is array *a[]*.

Line 12: Here we declare a character pointer *copy* and use a new-statement to dynamically allocate another array with length being the same as *a[]*.

Line 13 - Line 16: We use a for loop statement to copy each character in *a[]* into the new array *copy[]* (Line 14) and at the same time, change each character in the array *copy[]* to their upper case counterparts (Line 15).

Line 17: The function returns the array *copy*.

Line 20 - Line 30 is *main()* where we test the function *capitalize()*.

Line 22: We create a sample string in *main()*

Line 23: We declare a character pointer *cap*

Line 24: The function is called and the function receives array *a* as its input. The function makes a copy of the array *a* and changes all the characters in array *copy* to their upper case counterparts. The function return the array *copy* (Actually, the function returns

the address of the first element of the array *copy*, and we put the return address into the pointer *cap*, thus *cap* becomes the name of the array the function returns.

Following the screenshot of a test run of this program:

```
The original string is:  
I am student in Computer Science  
The string the function returns is:  
I AM STUDENT IN COMPUTER SCIENCE  
Program ended with exit code: 0
```

This page titled [9.3: Case Study- Array Copy](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

CHAPTER OVERVIEW

10: Recursive Functions

[10.1: Understand Recursive Functions](#)

[10.2: Recursive Functions v.s. Loop Statements](#)

[10.3: Using Array as Input of a Recursive Function](#)

This page titled [10: Recursive Functions](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

10.1: Understand Recursive Functions

A recursive function in C++ or other programming languages is a function that calls itself to form a recursion, which is an extremely powerful problem-solving technique. Problem that at first appear to be quite difficult often have a simple recursive solution.

Like other problem-solving technique, recursion breaks a problem into several smaller problems. But the difference about recursion is that those smaller problems are exactly the same type as the original problem, thus, the way to solve those smaller problem is going to be the same as the way to solve the original problem. And also, those smaller problems are themselves breaking into even smaller problems of the same type.

For example, if we want to find the largest number in a set of 5 numbers, we can break this problem into following smaller problems:

- Find the largest in a set of 4 of those 5 numbers
- Find the larger number between the largest number we found in those 4 numbers and the other number in the original set, the result will be the largest number of those five numbers in the original set

And the problem "Find the largest in a set of 4 of those 5 numbers can be further broken down to two even smaller problem

- Find the largest in a set of 3 of those 4 numbers
- Find the larger number between the largest number we found in those 3 numbers and the other number in the 4-number set, the result will be the largest number of those 4 numbers

We can continue this:

- Find the largest in a set of 2 of those 3 numbers
- Find the larger number between the largest number we found in those 2 numbers and the other number in the 3-number set, the result will be the largest number of those 3 numbers

and

- Find the largest in a set of 1 of those 2 numbers
- Find the larger number between the largest number we found in those 1 numbers and the other number in the 2-number set, the result will be the largest number of those 2 numbers

Since the largest number in a set of 1 number is the number itself, we have had the solution of the last problem. Then we pass the solution to the previous problem and we can easily find the largest number in a 2-number set by comparing two numbers

Continue to pass the largest number found in the 2-number set to the problem with 3-number set, we can also find the largest number in the 3-number set by comparing two numbers (one is the largest number in the 2-number set and the other one is the third number in the 3-number set.

Continue to pass the largest number found in the 3-number set to the problem with 4-number set, we can also find the largest number in the 4-number set by comparing two numbers (one is the largest number in the 3-number set and the other one is the fourth number in the 4-number set.

Continue to pass the largest number found in the 4-number set to the problem with 5-number set, we can also find the largest number in the 5-number set by comparing two numbers (one is the largest number in the 4-number set and the other one is the fifth number in the 5-number set. And this is solution of the original problem.

If you feel very confusing about the recursive technique we used in solving this problem, let me help you to understand it with following scenario:

Assume you have four brothers and sisters, they are age 15, 10, 8, and 5. Now, you got five boxes with different numbers of candies in each box. You would like to find out which box that has the most candies in it.

Of course, you can do the job all by yourself by opening each boxes and count the candies in the each box. But, you don't want to do it all by yourself, you would like to ask your brothers and sisters to help you out.

You keep one box of candies and do the counting for this box and pass the rest of four boxes to your 15-year old brother and let him to find out which box that contains the most of candies in those four boxes.

Your 15-years old brother can do the job by himself by opening those four boxes and count the candies in each box, but he doesn't want to either, he, like you, would like to ask the rest of brothers and sisters for help

He keep one box of candies and do the counting and pass the rest of three boxes to his 10-years old sister and let her to find out which box that contains the most candies.

Again, the 10-years old sister keeps only one box and do the counting and pass the rest of two boxes to her 8-years old brother and let him to find out which box that has the most candies.

Now, 8-years old brother also just keep one box for counting and pass the other box to his 5-years old sister and let her to find out which box that has the most candies.

The 5-years old sister doesn't need to do anything to find out which box has the most candies because she only has one box. But she is a nice girl, she still count the number of candies in the box and tell the her 8-year brother: this box has the most candies and the number of the candies in this box is so on so.

The 8-years old brother got the answer from 5-years old sister and compare the number got from his sister with the number in his hand, pass the box that has more candies along with the number of candies in the box to his 10-years old sister

The 10-years old sister got the answer from 8-years old brother and compare the number got from her younger brother with the number in his hand, pass the box that has more candies along with the number of candies in the box to her 15-years old brother.

The 15-years old brother got the answer from 10-years old sister and compare the number got from his sister with the number in his hand, pass the box that has more candies along with the number of candies in the box to you

Now, all you need to do is comparing the number in your hand with the number on the box you receive from your 15-years old brother, you will be easily find out the box that contains the most candies in those five boxes you receive originally.

The original job was broken into five smaller jobs. Each of those smaller jobs only needs to conducting a single box counting and a single comparison except the job that passed to the youngest sister.

The way we solve this candy counting problem is called recursion. We break a bigger problem into five smaller and easier problems of the same type.

However, understanding recursion is one thing and being able to use this technique in coding this another thing.

This page titled [10.1: Understand Recursive Functions](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

10.2: Recursive Functions v.s. Loop Statements

In theory, any programming problem that can be solved with a looping statement can be solved with a recursive function and vice versa, because both techniques are recursion in nature. We begin our recursive function study by finding the recursive function solution of a programming problem that we have had a solution with a looping statement.

Example

Write a function that receives a positive integer as its input. The function asks a user to enter a set of numbers, and the counts of numbers in the set is based on the integer value the function receives. For example, if the integer the function receives is 10, then the function will ask a user to enter 10 numbers, if the integer value the function receives is 5, then the function will ask a user to enter 5 numbers and so on. Then the function returns the sum of those numbers the user entered.

First, we provide the solution of this problem by using a looping statement:

```
1 #include<iostream>
2 using namespace std;
3
4 int getSum(int c)
5 {
6     int s=0,a;
7     int i=0;
8     while(i<c){
9         cout<<"Please enter a number:"<<endl;
10        cin>>a;
11        s+=a;
12        i++;
13    }
14    return s;
15 }
16
17 int main()
18 {
19     int cap,sum;
20     cout<<"How many numbers do you want to enter?"<<endl;
21     cin>>cap;
22     sum=getSum(cap);
23     cout<<"The sum of those numbers is "<<sum<<endl;
24     return 0;
25 }
26 }
```

and a sample run of this program has following output:

```
How many numbers do you want to enter?
5
Please enter a number:
2
Please enter a number:
5
Please enter a number:
3
Please enter a number:
2
Please enter a number:
3
The sum of those numbers is 15
Program ended with exit code: 0
```

Now, let's create a recursive version of this program. Following is the code of the recursive function along with a *main()* for testing purposes.

```
1  #include<iostream>
2  using namespace std;
3
4  int getSum(int c)
5  {
6      int s;
7      if(c==1){
8          cout<<"Please enter a number:"<<endl;
9          cin>>s;
10         return s;
11     }
12     else{
13         cout<<"Please enter a number:"<<endl;
14         cin>>s;
15         return s+getSum(c-1);
16     }
17 }
18
19 int main()
20 {
21     int cap,sum;
22     cout<<"How many numbers do you want to enter?"<<endl;
23     cin>>cap;
24     sum=getSum(cap);
25     cout<<"The sum of those numbers is "<<sum<<endl;
26     return 0;
27 }
```

The recursive logic:

If the integer value the function receives is 1, then the function only asks a user to enter 1 number and return it as the sum.

If integer value the function receives more than 1, then the function also asks the user to enter 1 number, and call the function itself *getSum()* letting this function to ask the user to enter the rest of the numbers and return the sum of those numbers. When the function receives the return value of the function call *getSum()*, which is the sum of the rest of *c-1* numbers the user entered, adds the number the user entered in this function, *s*, to the return value of the function *getSum()* to produce the sum of all the numbers user entered and return it (Line 15). As we can see, both functions produce the same solution:

```
How many numbers do you want to enter?
5
Please enter a number:
2
Please enter a number:
5
Please enter a number:
3
Please enter a number:
2
Please enter a number:
3
The sum of those numbers is 15
Program ended with exit code: 0
```

Example

Write a function that receives a positive integer, say *n*, then the function displays all the integers from 1 to *n*.

Again, we first provide a solution by using a looping statement:

```
1 #include<iostream>
2 using namespace std;
3
4 void display(int n)
5 {
6     for(int i=1;i<=n;i++)
7         cout<<i<<endl;
8 }
9
10 int main()
11 {
12     int n;
13     cout<<"Please enter a positive integer:"<<endl;
14     cin>>n;
15     cout<<"======"<<endl;
16     display(n);
17     return 0;
18 }
```

The output of this program:

```
Please enter a positive integer:
10
=====
1
2
3
4
5
6
7
8
9
10
Program ended with exit code: 0
```

Now, let's create a recursive version of this function:

```
1 #include<iostream>
2 using namespace std;
3
4 void display(int n)
5 {
6     if(n==1)
7         cout<<1<<endl;
8     else{
9         display(n-1);
10        cout<<n<<endl;
11    }
12 }
13
14 int main()
15 {
16     int n;
17     cout<<"Please enter a positive integer:"<<endl;
18     cin>>n;
19     cout<<"======"<<endl;
20     display(n);
21     return 0;
22 }
```

The output of this program:

```
Please enter a positive integer:
10
=====
1
2
3
4
5
6
7
8
9
10
Program ended with exit code: 0
```

Both functions provide identical solutions.

Logic:

If the integer the function receives is 1 ($n=1$), then just display this number which is 1.

If the integer the function receives is greater than 1 ($n>1$), then call the function itself, *getSum($n-1$)* at Line 9, this function displays the number from 1 to $n-1$, and then, display the last number, which is n at line 10.

This page titled [10.2: Recursive Functions v.s. Loop Statements](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

10.3: Using Array as Input of a Recursive Function

Example

Write a recursive function that receives an array of integers and returns the largest number in the array the function receives.

Step 1: Write the header of this function:

```
1  #include<iostream>
2  using namespace std;
3
4  int getLargest(int aArray[], int length)
5  {
6  ...
7  }
```

Step 2: Code the function with the base case: when length=1.

If the length=1, that means the array the function receives has only one element which is aArray[0] and the number in this element is the largest number, so function should return it.

```
1  #include<iostream>
2  using namespace std;
3
4  int getLargest(int aArray[], int length)
5  {
6      if (length == 1)
7          return aArray[0];
```

Step 3: Recursion

If the array the function receives has more than one elements, then the function will keep the last one by using the statement in line 9: `int a = aArray[length-1];` and pass the rest of the numbers in the array to the next function (like pass the rest of boxes to the next sibling). Since all the functions will do exact the same thing:

- keep the last number in the array it receives and pass the rest of the numbers in the array to the next function
- receives the solution from the next function and compare with the number it keeps and return the largest

Therefore, all the functions will poses the same code, they are the same function: `getLargest()`. The statement in line 10: `"int b = getLargest(aArray, length-1);"` passes the rest of the numbers in the array to the function `getLargest()`

```
1  #include<iostream>
2  using namespace std;
3
4  int getLargest(int aArray[], int length)
5  {
6      if (length == 1)           keep the last number in the array
7          return aArray[0];
8      else {
9          int a = aArray[length - 1];
10         int b = getLargest(aArray, length - 1);
11         if (a > b)
12             return a;
13         else
14             return b;          pass the rest of numbers to the next function
15     }
16 }
```

The statement in Line 10 also receives the return value from the next function call and put it into the variable `b`

Line 11 - Line 14: compare the return value received from the next function call with the number the function kept and return the larger value which is the largest number in the array.

How many function calls does this process made? It depends on the number of elements in the array. For example, if the original array has 5 numbers, then find the largest number in the array will use five function calls or use the same function five times (just like finding the box that has the most candies among five candy boxes needs five siblings). Beginning from the second function,

each function call will receive one less items from the array and keep one of them and pass the rest of the items to the next function and wait for result that comes back from the next function call. After receive the result, compare and then returns the larger one.

The return value of first function is the solution of the original problem.

We can test run this function with following program:

```
1  #include<iostream>
2  using namespace std;
3
4  int getLargest(int aArray[], int length)
5  {
6      if (length == 1)
7          return aArray[0];
8      else {
9          int a = aArray[length - 1];
10         int b = getLargest(aArray, length - 1);
11         if (a > b)
12             return a;
13         else
14             return b;
15     }
16 }
17
18 int main()
19 {
20     int myArray[10] = { 4, 6, 1, 7, 2, 5, 12, 10, 3, 2 };
21     cout << "The numbers in the array are:" << endl;
22     for (int i = 0; i < 10; i++) {
23         cout << myArray[i] << endl;
24     }
25     cout << "The largest number is " << getLargest(myArray, 10) << endl;
26     return 0;
27 }
```

Let's try to use this recursive technique to solve another problem.

Example

Write a recursive function that receives an array of integers and returns the sum of the numbers in the array the function receives.

Step 1: Step 1: Write the header of this function:

```
1  #include<iostream>
2  using namespace std;
3
4  int getSum(int aArray[], int length)
5  {
6      ...
7  }
```

Step 2: Code the function with the base case: when length=1.

If the length=1, that means the array the function receives has only one element which is aArray[0] and the number in this element is the sum of the array, so function should return it.

```
int getSum(int aArray[], int length)
{
    if (length == 1)
        return aArray[0];
    ...
}
```

Step 3: Recursion

If the array the function receives has more than one elements, then the function will keep the last one by using the statement in line 9: `int a = aArray[length-1];` and pass the rest of the numbers in the array to the next function. Since all the functions will do exact the same thing:

- keep the last number in the array it receives and pass the rest of the numbers in the array to the next function
- receives the solution from the next function which is the sum of the rest of the numbers in the array it passed to the next function and return the sum of the number it kept and the number it received from the next function

Therefore, all the functions will poses the same code, they are the same function: *getSum()*. The statement in line 10: "*int b = getSum(aArray, length-1);*" passes the rest of the numbers in the array to the function *getSum()*

```
1  #include<iostream>
2  using namespace std;
3
4  int getSum(int aArray[], int length)
5  {
6      if (length == 1)
7          return aArray[0];
8      else {
9          int a = aArray[length - 1];
10         int b = getSum(aArray, length - 1);
11         return a + b;
12     }
13 }
```

The statement in line 10 also receives the solution from the next function and put it into the variable *b*

Line 11: returns the sum of the number the function kept and the number the received from line 9 which is the sum of the rest of the numbers it passed to the next function.

How many functions does this process use? It depends on the number of elements in the array. For example, if the original array has 5 numbers, then find the sum of the numbers in the array will use five functions or use the same function five times (just like finding the box that has the most candies among five candy boxes needs five siblings, this time, instead of finding the box that has the most candies, we find the total number of candies in all five boxes). Beginning from the second function, each function will receive one less items from the array and keep one of them and pass the rest of the items to the next function and wait for result that comes back from the next function. After receive the result, add the number the function kept and return the sum

The return value of first function is the solution of the original problem.

We can test run this function with following program:

```
1  #include<iostream>
2  using namespace std;
3
4  int getSum(int aArray[], int length)
5  {
6      if (length == 1)
7          return aArray[0];
8      else {
9          int a = aArray[length - 1];
10         int b = getSum(aArray, length - 1);
11         return a + b;
12     }
13 }
14 int main()
15 {
16     int myArray[10] = { 4, 6, 1, 7, 2, 5, 12, 10, 3, 2 };
17     cout << "The numbers in the array are:" << endl;
18     for (int i = 0; i < 10; i++) {
19         cout << myArray[i] << endl;
20     }
21     cout << "The largest number is " << getSum(myArray, 10) << endl;
22     return 0;
23 }
```

Example

Write a recursive function that receives a positive integer *n* and returns the sum of all the integers from 1 to *n*. For example, if the function receives an integer 5, then the function will return 15

If we use above "recursive thinking", the recursive function can be easily coded. When the function receives an integer *n*, we can consider the function receives a sequence of integers from 1, 2, 3, ..., *n*. The function will keep the last number in the sequence and pass the rest of numbers to the next function. The wait for the next function to return the solution, which is going be the sum of the numbers from 1 to *n*-1. Then the function add the it kept, which is *n*, and return the sum which will be sum of all the numbers in the sequence. Of course, the next won't actually do the entire calculation, it will also kept the last number in the sequence and pass the

rest of numbers in the sequence to its next function. The only function that won't pass the sequence is the last function because the sequence the this function receives has only one number. This function simply return it.

```
1  #include<iostream>
2  using namespace std;
3
4  int sumUp(int n)
5  {
6      if (n == 1)
7          return 1;
8      else {
9          int a = n;
10         int b = sumUp(n-1);
11         return a + b;
12     }
13 }
```

If the number the function receives is 1, then the sum of the integer from 1 to 1 is 1. Thus the function returns 1

If the n is greater than 1, then the function keeps the last number which is n and put it in the variable a in line 9 and passes the rest of the numbers to the next function in line 10. Once the function get the result from the next function, put it in the variable b.

The function return the sum of the number it kept and the result from the next function in line 11.

This page titled [10.3: Using Array as Input of a Recursive Function](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

Glossary

Sample Word 1 | Sample Definition 1

Detailed Licensing

Overview

Title: [Introduction to Programming with C++ \(Jin He\)](#)

Webpages: 75

Applicable Restrictions: Noncommercial

All licenses found:

- [CC BY-NC-SA 4.0](#): 81.3% (61 pages)
- [Undeclared](#): 17.3% (13 pages)
- [CC BY-SA 4.0](#): 1.3% (1 page)

By Page

- [Introduction to Programming with C++ \(Jin He\)](#) - [CC BY-NC-SA 4.0](#)
 - [Front Matter](#) - [CC BY-NC-SA 4.0](#)
 - [TitlePage](#) - [CC BY-NC-SA 4.0](#)
 - [InfoPage](#) - [CC BY-NC-SA 4.0](#)
 - [Table of Contents](#) - [Undeclared](#)
 - [Licensing](#) - [CC BY-NC-SA 4.0](#)
 - [1: Chapter 1 Introduction to Computer Programming](#) - [CC BY-NC-SA 4.0](#)
 - [1.1: Introduction to Computer Science](#) - [CC BY-NC-SA 4.0](#)
 - [1.2: What Is a Computer? and What Is a Computer Program?](#) - [CC BY-NC-SA 4.0](#)
 - [1.3: Compilers and the Apps We Need for Learning C++ Programming](#) - [CC BY-NC-SA 4.0](#)
 - [1.4: Download and Install App for C++ Programming \(PC Users\)](#) - [CC BY-NC-SA 4.0](#)
 - [1.5: Download and Install App for C++ Programming \(Mac Users\)](#) - [CC BY-NC-SA 4.0](#)
 - [1.6: Save a Copy of Source File](#) - [CC BY-NC-SA 4.0](#)
 - [1.7: Understanding the Code of the First C++ Program](#) - [CC BY-NC-SA 4.0](#)
 - [1.8: Case Study- Using setw\(\) to Format the Output](#) - [CC BY-NC-SA 4.0](#)
 - [2: Chapter 2 Elementary Programming](#) - [CC BY-NC-SA 4.0](#)
 - [2.1: Constants and Variables](#) - [CC BY-NC-SA 4.0](#)
 - [2.2: Console Input and Output](#) - [CC BY-NC-SA 4.0](#)
 - [2.3: Assignment Statements](#) - [CC BY-NC-SA 4.0](#)
 - [2.4: Case Study- Sum of Numbers](#) - [Undeclared](#)
 - [2.5: Numerical Operators](#) - [CC BY-NC-SA 4.0](#)
 - [2.6: Coding a Mathematical Formula](#) - [CC BY-NC-SA 4.0](#)
 - [2.7: C++ Examples](#) - [CC BY-SA 4.0](#)
 - [2.8: Case Study- Find the Sum of Numbers](#) - [Undeclared](#)
 - [3: Chapter 3 Selection Statements](#) - [CC BY-NC-SA 4.0](#)
 - [3.1: Relational Operators and Relational Statements](#) - [CC BY-NC-SA 4.0](#)
 - [3.2: Selection Statement- if-statement](#) - [CC BY-NC-SA 4.0](#)
 - [3.3: Case Study- Find the Largest Number](#) - [CC BY-NC-SA 4.0](#)
 - [3.4: Case Study- Sorting Numbers](#) - [CC BY-NC-SA 4.0](#)
 - [3.5: Selection Statement- if-else Statement](#) - [CC BY-NC-SA 4.0](#)
 - [3.6: Radom Number and Logical Operators](#) - [CC BY-NC-SA 4.0](#)
 - [3.7: Conditional Expression](#) - [CC BY-NC-SA 4.0](#)
 - [3.8: Case Study- Practice Multiplication](#) - [CC BY-NC-SA 4.0](#)
 - [3.9: Switch Statements and Menu Driven Programs](#) - [CC BY-NC-SA 4.0](#)
 - [4: Chapter 4 Characters and Strings](#) - [CC BY-NC-SA 4.0](#)
 - [4.1: Character Data Type and ACII Code](#) - [CC BY-NC-SA 4.0](#)
 - [4.2: String Data Type](#) - [CC BY-NC-SA 4.0](#)
 - [5: Chapter 5 Looping Statements](#) - [Undeclared](#)
 - [5.1: The Structure and The Logic of a while Loop Statement](#) - [CC BY-NC-SA 4.0](#)
 - [5.2: Case Study- Find The Largest Number](#) - [CC BY-NC-SA 4.0](#)
 - [5.3: Case Study- Find The Sum](#) - [CC BY-NC-SA 4.0](#)
 - [5.4: Sentinel Control Loop and do-while Loop Statement](#) - [CC BY-NC-SA 4.0](#)
 - [5.5: for Loop Statement](#) - [Undeclared](#)
 - [5.6: Nested Loop Statements](#) - [CC BY-NC-SA 4.0](#)

- 5.7: Case Study- Display a Rectangle - *CC BY-NC-SA 4.0*
- 6: Chapter 6 C++ Functions - *CC BY-NC-SA 4.0*
 - 6.1: Introduction to C++ Functions - *CC BY-NC-SA 4.0*
 - 6.2: More Examples and Further Discussion about Functions - *Undeclared*
 - 6.3: Global, Local, and Static Variable - *Undeclared*
 - 6.4: Passing by Value vs Passing by Reference - *CC BY-NC-SA 4.0*
 - 6.5: Function Overloading - *CC BY-NC-SA 4.0*
- 7: Chapter 7 Arrays - *CC BY-NC-SA 4.0*
 - 7.1: Introduction to Arrays - *CC BY-NC-SA 4.0*
 - 7.2: Working with the Data Stored in Arrays - *CC BY-NC-SA 4.0*
 - 7.3: Case Study- Sorting Data Stored in an Array - *CC BY-NC-SA 4.0*
 - 7.4: User-Defined functions with an Array as its Input - *CC BY-NC-SA 4.0*
 - 7.5: C-String and String Variables - *CC BY-NC-SA 4.0*
- 8: Introduction to Object Oriented Programming - *CC BY-NC-SA 4.0*
 - 8.1: Introduction to Object Oriented Programming (PC Users) - *CC BY-NC-SA 4.0*
 - 8.2: Introduction to Object Oriented Programming (Mac Users) - *CC BY-NC-SA 4.0*
 - 8.3: Class Constructors - *CC BY-NC-SA 4.0*
 - 8.4: Case Study- C++ Class- StudentRecord - *CC BY-NC-SA 4.0*
 - 8.5: Case Study- C++ Class- Bag - *CC BY-NC-SA 4.0*
- 9: Pointers - *Undeclared*
 - 9.1: What is a Pointer? - *CC BY-NC-SA 4.0*
 - 9.2: Pointer and Dynamic Memory Allocation - *Undeclared*
 - 9.3: Case Study- Array Copy - *CC BY-NC-SA 4.0*
- 10: Recursive Functions - *CC BY-NC-SA 4.0*
 - 10.1: Understand Recursive Functions - *CC BY-NC-SA 4.0*
 - 10.2: Recursive Functions v.s. Loop Statements - *CC BY-NC-SA 4.0*
 - 10.3: Using Array as Input of a Recursive Function - *CC BY-NC-SA 4.0*
- Back Matter - *Undeclared*
 - Index - *Undeclared*
 - Glossary - *Undeclared*
 - Detailed Licensing - *Undeclared*