Nīlapatri

The Nilapatri C++ Foundation Guide

By Praajna Pattada

Nilapatri C++ Foundation Guide

Author: Praajna Pattada

Pages: 121

Publishing of First Edition: 22 May 2026

About

This book is a guide on C++ programming, primarily meant for those with little to basic knowledge of computer science and programming, as well as beginners. It explains the fundamentals of the C++ programming language and each concept elaborately.

The contents of this media are copyrighted by Nilapatri. Contents which are not owned by us belong to their respective owners and are credited accordingly. The unauthorized replication, printing or commercial usage, of the contents of this work is strictly prohibited without written permission from the author.

Nīlapatri

© Copyrights 2026 Praajna Pattada (praajna@nilapatri.com)

https://nilapatri.com

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Acknowledgements

श्रीनारायणाय नमः ।

This book has been written with the prime objective of explaining C++ in detail, including its advanced concepts, to beginners or those with moderate knowledge of programming. This book has been inspired by multiple websites and books, including but not limited to:

- A Complete Guide to Programming in C++ by Ulla Kirch-Prinz and Peter Prinz
- C++ Primer (5th Edition)
- The C++ Programming Language by Bjarne Stroustrup
- Learncpp.com

The primary reason for writing this book is due to the lack of a proper structured resource available easily, for learning C++, which is also absolutely free. While this book is definitely not a gold-standard resource, as I am merely a hobbyist and still a learner who is primarily a medico and not an engineer, it is hoped that it benefits all who need it and can provide a good foundation of programming.

I would like to express my immense gratitude to my Computer Science teachers, especially Mr. Rohan R. Abhijith, Mr. Dhruva Rao, Ms. Rajitha K., Mr. Sanath Jamadagni, Ms. Priyanka M. R. and Ms. Kavya Annaiya, who taught me programming and always inspired me to pursue it as my hobby, due to which I have been able to publish this book.

—Praajna Pattada

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Contents

Preface 6

Chapter 1: Introduction

- Basics of Computer Programming 7
- The C++ Programming Language 8
- Working of a Compiler 9
- Installing a Compiler/IDE 10
- GNU Compiler Collection (GCC)/MinGW-w64 11
- Microsoft Visual C++ 16
- Writing your first Program in C++ 19
- Ideal Programming Practices 23

Chapter 2: Fundamentals of C++ Programming

- Structure of a Program 24
- Keywords 27
- Variables 29
- Constants 32
- Operators 33
- Escape Sequences 34
- Input and Output using Variables 36
- Header Files 37
- Using Multiple Source Files 42
- Namespaces 44

Chapter 3: Other Types of Statements

- Declarations 46
- Functions 49
- Function Overloading 52
- Conditional Statements 54
- Loops 57
- Templates 62

Chapter 4: Strings, Compound Data Types and a Few Other Concepts

- Strings 64

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

- Type Conversion 68
- Macros 70
- Inline Functions 72
- Introduction to Compound Data Types 73
- References 75
- Pointers 77
- Arrays 79
- std::vector 80

Chapter 5: Object-Oriented Programming (OOP)

- Introduction 81
- Classes 82
- Structures and Unions 86
- Class Templates 88
- Enumerations 90
- Constructors and Destructors 92
- Operator Overloading 96

Chapter 6: A Few Miscellaneous Topics

- File Input and Output 98
- CTime 102
- Random Values 104
- Final Practice Question 107

Conclusion 108

Solutions 109

Index 120

About the Author 121

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Preface

There exist ample resources online about C++ programming. One might ask why it was necessary to write this e-book. As a programmer my childhood, I have always wanted to learn C++ since a long time. However, what was hard to find was a resource which is:

- Easy to understand
- Covered in-depth explanations of concepts
- Explained the 'Why?' of computer programs rather than just 'How?'
- Freely available

As someone who has a small hobby of computer programming and video game development, but studies medicine and writes books on philosophy, writing a book on this subject in which I had not been a great expert was a challenge I did not initially want to take up. However, seeing the lack of good beginner-friendly resources online, I decided that it was necessary. This book also acts as a reference library and covers the core concepts of C++, making it helpful for one to gain a foundation of the language and take up advanced learning further. It is hoped that it serves its purpose.

The examples in this book use G++ as the primary C++ compiler and are designed to run on Windows. While they can be adjusted to run on MacOS and Linux, it may be tedious for absolute beginners. It also assumes that the reader has basic knowledge of using software and computers, and already has a working C++ compiler, which is why elaborate instructions on the same are not provided.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Chapter 1: Introduction

Basics of Computer Programming

Today, all modern software work based on specific instructions given to a computer or any electronic device. A program is a set of instructions for a computer to perform an action. It is written in a standardized format, known as a 'programming language'. A computer requires explicit instructions to perform something. When a computer performs the actions instructed in a computer program, it is said to 'run' or 'execute' the program. This period is also known as 'runtime'. Primarily, computers understand instructions in binary codes, written in the form of zeroes and ones, known as 'machine language'. Each 0 or 1 is known as a 'binary digit' or 'bit'. Machine languages are called 'low-level languages'. The specific instructions in binary code can vary, depending on the hardware, making it tedious to write a program which can be universally run on any device. For this reason, we have high-level languages, such as C and C++, in which one can write a program using their standardized code and while running on a particular machine, the program gets converted to machine language. This is known as 'compilation' and the software which converts the code is known as a 'compiler' or 'interpreter'.

An interpreter is a basic form of software, which reads the program slowly and line-by-line, highlighting a single error at a time. Compilers are faster and read the entire program and detect all errors at once. A compiler produces an executable file, which is the final version of the program which is written in machine code and can be run on a computer. Unlike compilers, an interpreter provides the results quickly by immediately running the program, but does not compile the code.

Each line of instruction in a program, is known as a 'statement'. These provide specific instructions for the computer to provide a desired result. All these shall be explained elaborately in the upcoming chapters. Programs are written based on 'algorithms', which are sequences of instructions to achieve a certain result. Through formulating an algorithm, one use necessary statements to make a program to achieve the same algorithm.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

The C++ Programming Language

C++ was developed by Bjarne Stroustrup in 1979, as an extension of the programming language C. C had been one of the most advanced programming languages during those times and was used to develop a variety of software, including major operating systems. To make it more advanced, C++ was developed. Therefore, most code written in C is usually valid in C++. C++ introduced features such as Object-Oriented Programming (OOP), making it easier to develop advanced applications. Today, C++ powers the modern world, as most major applications such as operating systems, utility software, video games, audio and video applications, AI-related applications, etc., are all developed using C++. It continues to be an industry standard programming language till date.

C++ also has new updates, such as C++11 (in 2011), C++14 (in 2014), C++17 (in 2017) and C++23 (in 2023). C++11 has made multiple changes, due to which it is now considered the most basic form of C++, for most programmers.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Working of a Compiler

A compiler has the following major components:

- **Pre-Processor**: It gathers all the data and information required in a program and gets them ready to be compressed/included in a program. It removes the '`#include`' and '`#define`' statements, by adding the respective code to the program.
- **Compiler**: It is the core of the software, which has six components.
    - Lexical Analyser
    - Syntax Analyser/Parser
    - Semantic Analyzer
    - Intermediate Code Generator
    - Code Optimizer
    - Target Code Generator
- **Assembler**: It coverts the compiled program file specifically to a format which the computer can run, i.e., from assembly code to machine code. Its output file is called an 'object file'.
- **Linker**: The linker combines all created files to the main program file(s). It loads a variety of object files into a single file, to make it an executable file. It converts the code from relocatable code (with variables having undefined values) to absolute code (with defined values).

When you compile your program, it runs through these phases of processing and is finally converted to an executable file.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Installing a Compiler/IDE

To compile C++ programs on your computer and make executable files of the same, you will need a working C++ compiler. An Integrated Development Environment (IDE) is a software which includes a compiler, text editor and other tools to manage your program using a single application. Without an IDE, you will need to use a text editor like Notepad or Notepad++ to write your program, and run the compiler using command prompt, terminal or Windows PowerShell.

Some of the popular compilers/IDEs are:

- **GNU Compiler Collection (GCC)**: Includes G++, a popular C++ compiler. It is included in MinGW (Minimalist GNU for Windows), a version of GCC made for Windows. You can download it from its official website (https://mingw-w64.org).
- **Microsoft Visual Studio**: An IDE by Microsoft, which can run C++ programs using the compiler Visual C++, and can also be configured for developing applications in multiple other programming languages. It is available for Windows. (https://visualstudio.microsoft.com)
- **Clang**: A C++ compiler, popular among MacOS users
- **Codeblocks**: A simple IDE for C++, which uses GCC and is available for Windows, MacOS and Linux (https://www.codeblocks.org)
- **Dev C++**: An IDE for C++, which uses GCC and is available for Windows (https://www.dev-cpp.com)

Our website (https://nilapatri.com/cpp) also has links to an already prepared suite with MinGW, which one can download, if at all there are difficulties in downloading it from the official website or other sources. The instructions for setting it up will also be provided on the respective sites. Note that all programs in this book shall be tested using this compiler. For your own reference, you can also download the source code and compiled executables of the same on the following page of our website.

Elaborate instructions on installing a compiler have been skipped, as there are ample good tutorials online, for doing the same. A short tutorial has been provided for setting up MinGW and Visual Studio, in the upcoming sections.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

GNU Compiler Collection (GCC)/MinGW-w64

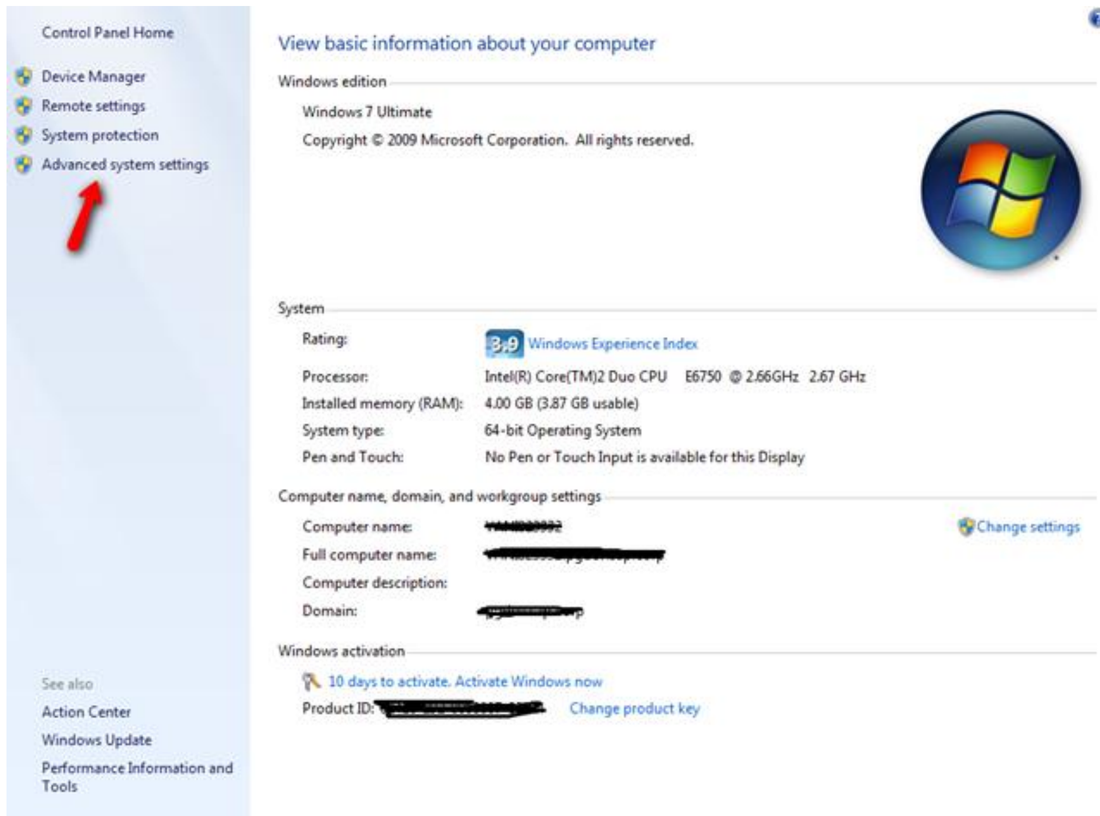
If you are a slightly advanced PC user or are comfortable working with command-line tools, GCC is an excellent option. All example programs and practice questions in this book are compiled using G++ on Windows. GCC is available on Windows, MacOS and Linux, though the instructions for setting it up and using it differs across platforms. This section contains a short set of instructions on installing GCC on Windows. You can visit the official website of MinGW (https://mingw-w64.org), a C++ development library for Windows, which is best for use with GCC. The easiest option to download and set up GCC is by downloading prepared suites of it, available online. Our website (https://nilapatri.com/cpp/) also has links to them. Visit https://winlabs.com/ for the latest versions which also include additional tools, such as GDB (a debugger). The download links look like this:

- GCC 15.2.0 (with **POSIX** threads) + MinGW-w64 14.0.0 (UCRT) - release 7
    - Win32 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
    - Win64 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
- GCC 15.1.0 (with **POSIX** threads) + MinGW-w64 13.0.0 (UCRT) - release 4
    - Win32 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
    - Win64 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
- GCC 15.1.0 (with **POSIX** threads) + MinGW-w64 13.0.0 (UCRT) - release 2
    - Win32 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
    - Win64 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
- GCC 15.1.0 (with **POSIX** threads) + MinGW-w64 12.0.0 (UCRT) - release 1
    - Win32 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
    - Win64 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
- GCC 15.1.0 (with **MCF** threads) + MinGW-w64 12.0.0 (UCRT) - release 1
    - Win32 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
    - Win64 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
- GCC 14.3.0 (with **POSIX** threads) + MinGW-w64 12.0.0 (UCRT) - release 1
    - Win32 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive
    - Win64 (without LLVM/Clang/LLD/LLDB): 7-Zip archive* | Zip archive

Another method is to use tools like MSYS2, as explained on the official website of MinGW, which may be harder for beginners. If you downloaded the individual components of GCC and MinGW, or a pre-written suite, you can make a folder for storing the files. Make sure that you have any archiving software, such as WinRAR, installed on your computer, and that the names of your chosen folder and its parent folders do not have any spaces. For example, use a folder such as 'C:\MinGW64\', which does can be accessed from the command line without any spaces in the names of the folder. You can then extract the files from the archive into this folder. The 'bin' folder contains the binary files of the compiler, which we will need to use to compile our programs.

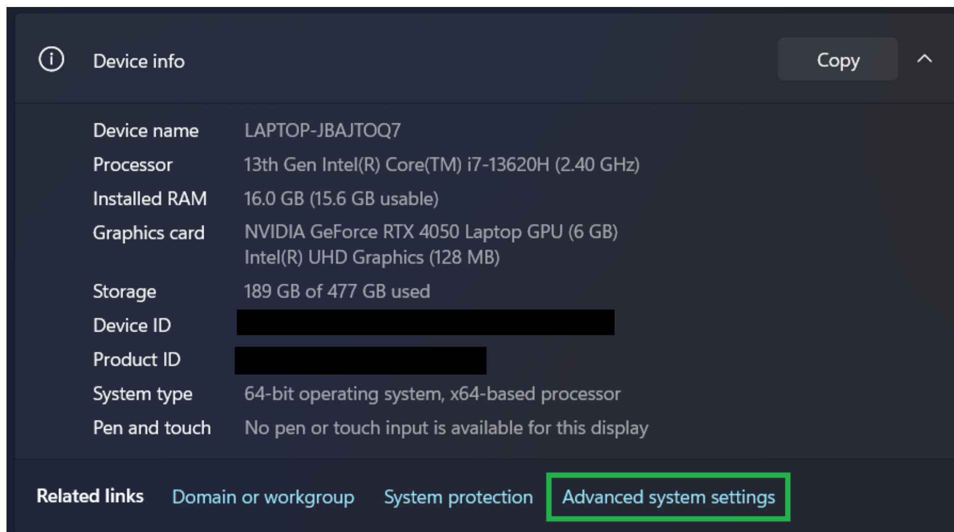
Now, you will need to add this directory to your system path environment variable. To do this (the procedure is almost the same for Windows XP/Vista/7/8/8.1/10/11), make sure you are using an Administrator account and simply open your system settings and go to 'System'. You will find an option such as 'Advanced System Settings'. An easier way to find this screen on any Windows operating system is by going to Windows explorer, right-clicking on the Computer/This PC folder and clicking 'Properties'. On Windows Vista and later, you can find the option on the sidebar.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)



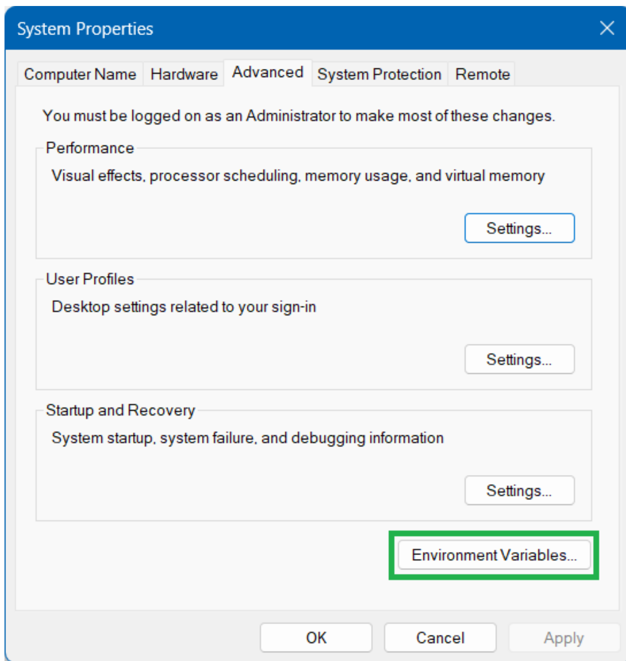
Picture Credit: nextofwindows.com

On Windows 10 and 11, the screen looks similar to this.

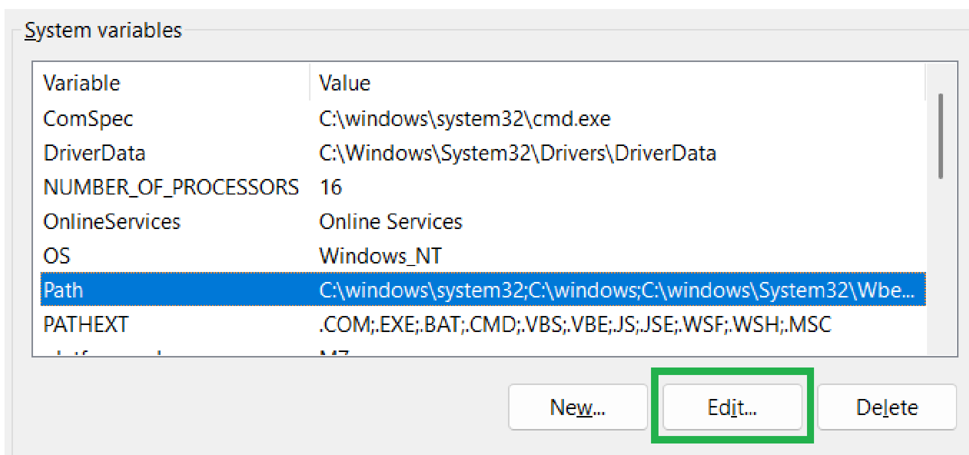


Once you click on it, the following popup window opens. Click on 'Environment Variables'.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

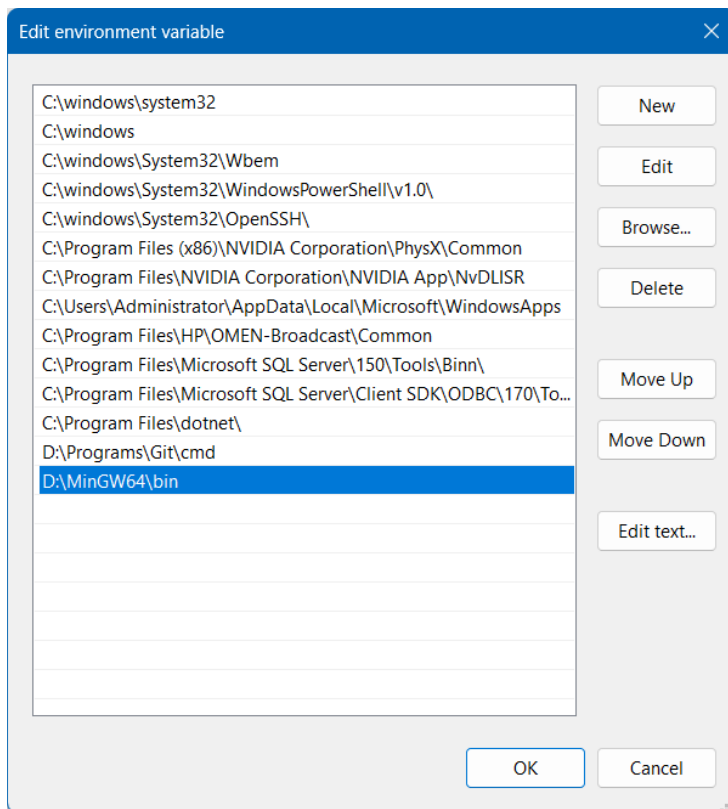


You will see a list of the environment variables. Note that these are system variables which Windows and other programs use to identify various directories or other information. Always modify them with caution. There will be two sections called 'User variables' (specific to the current user account) and 'System variables' (which are universal for all user accounts on the computer). Select 'Path' under the System variables section, and click 'Edit'.



On Windows 8.1 or earlier, you will have a small box with all directories in that variable, separated by a semi-colon (;). Press the end key to go to the end of the line and make sure the whole set of text ends with a semi-colon. If it does not, add a semi-colon and then type the path to the Binaries folder of GCC. On Windows 10 and above, you will have a dialog box instead, where you will need to click 'New' and type the directory to GCC.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)



In this example, MinGW is installed in a folder called 'MinGW64' on drive D:, which is why the binaries folder is 'D:\MinGW64\bin'. Press 'OK' and 'OK' again, to close all dialog boxes.

Now you have successfully added GCC to your environment variables. By doing this, a new window of command prompt or PowerShell, which may be accessing a different folder, can still access G++. To ensure that the compiler has installed correctly and you have also added its binary folder to the path variable correctly, open a new command prompt window and test the compiler. Do not change the directory using the 'cd' command. Simply type the following commands:

```
g++ --version
gcc --version
gdb --version
```

All three should provide information about the compiler version, as shown in the following screenshot. Note that some suites on do not include GDB, so the last command will not work if you downloaded only the core components of MinGW.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
Command Prompt

Microsoft Windows [Version 10.0.26200.8246]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Praajna>g++ --version
g++ (MinGW-W64 x86_64-ucrt-posix-seh, built by Brecht Sanders, r7) 15.2.0
Copyright (C) 2025 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


C:\Users\Praajna>gcc --version
gcc (MinGW-W64 x86_64-ucrt-posix-seh, built by Brecht Sanders, r7) 15.2.0
Copyright (C) 2025 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


C:\Users\Praajna>gdb --version
GNU gdb (GDB for MinGW-W64 x86_64, built by Brecht Sanders, r7) 17.1
Copyright (C) 2025 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

C:\Users\Praajna>
```

If you successfully obtained this screen, congratulations on setting up G++! Now, you can make your first program and see if it compiles too. Simply use the 'cd' (Change directory) command to go to the folder where your program is. If it is on another hard drive partition, first type the drive letter and a colon, and then press enter (for example, 'D:' for drive D). After switching to the folder with your program, remember the following syntax for the command to compile it:

```
g++ <Program source file(s).extension> -o <Executable Name>.exe
```

For example, if your program is 'HelloWorld.cpp', the command will be:

```
g++ HelloWorld.cpp -o HelloWorld.exe
```

If your program compiled successfully without errors or warnings, the command will complete without displaying anything and exit G++. Your compiled program's executable file will now be named 'HelloWorld.exe', which you can notice in the same folder. You can run it to see your program in action. If you are using Windows PowerShell instead of Command Prompt, you will need to type '`./HelloWorld.exe`' instead.

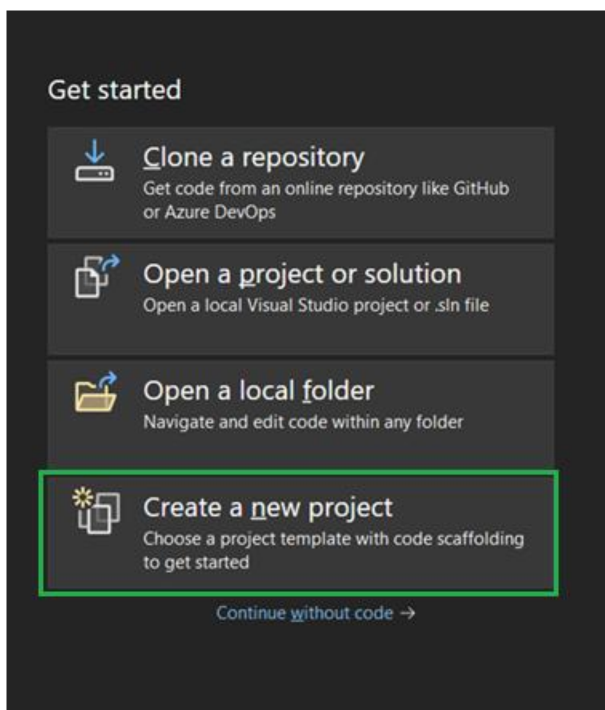
```
C:\Users\Praajna>D:

D:\>cd D:\Documents\CPlusPlus\Programs\ProgrammingExamples\Chapter1

D:\Documents\CPlusPlus\Programs\ProgrammingExamples\Chapter1>g++ HelloWorld.cpp -o HelloWorld.exe

D:\Documents\CPlusPlus\Programs\ProgrammingExamples\Chapter1>HelloWorld.exe
Hello, world!
Press any key to continue . . .

D:\Documents\CPlusPlus\Programs\ProgrammingExamples\Chapter1>
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Microsoft Visual C++

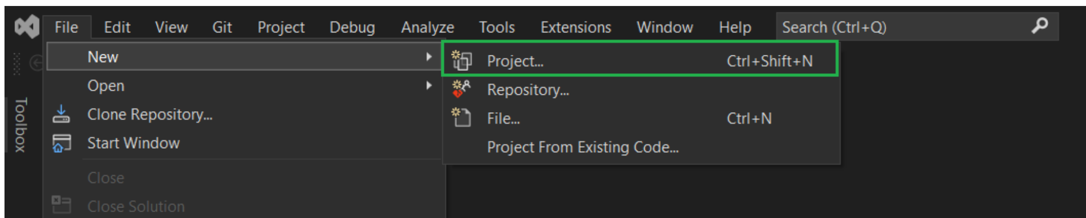
Visual C++, a C++ compiler by Microsoft, is a popular and industry-standard compiler for C++ development on Windows. It is especially convenient to use for beginners or those who may not have advanced technical knowledge, as it is included in Microsoft Visual Studio, a popular IDE by Microsoft. It has both paid and free editions. The Community edition (previously called 'Express') is free to use for smaller projects or individual developers. It is recommended to use at least Visual Studio 2017 or later versions, for the programs in this book. You can download it on Microsoft's official website for Visual Studio (https://visualstudio.microsoft.com), which has the latest version (2026, as of writing this book). Make sure to select the workload 'Desktop Development with C++', while installing Visual Studio and the SDK (Software Development Kit) for your respective operating system. Advanced PC users can also make an offline installer of the same using the installer or command prompt, for faster installation and to manually select any other specific workloads/individual components needed.

Once Visual Studio is installed, you will need to set it up for the first time and activate it, by signing in to your Microsoft account or creating one, if you do not have any. Then, you can start making C++ programs. On most IDEs, including Visual Studio, an application and its source code are organized into a 'project'. When you open Visual Studio, you may see a small window with the following options. Click on 'Create a new project'.

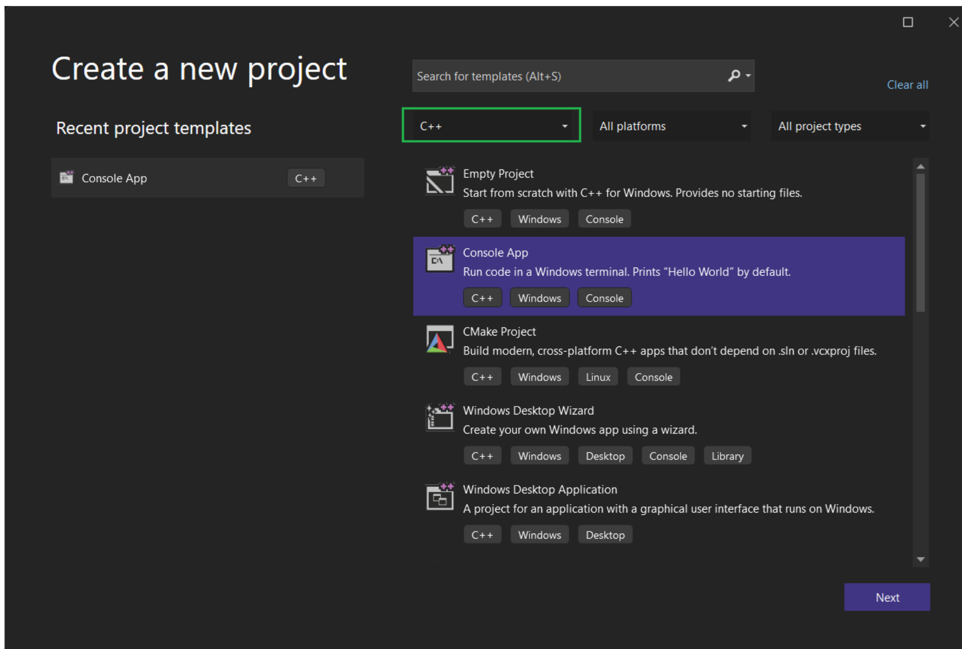


Alternatively, if you chose 'Continue without code' and the main window is already open, you will need to use the options from 'File' tab on the menu bar.

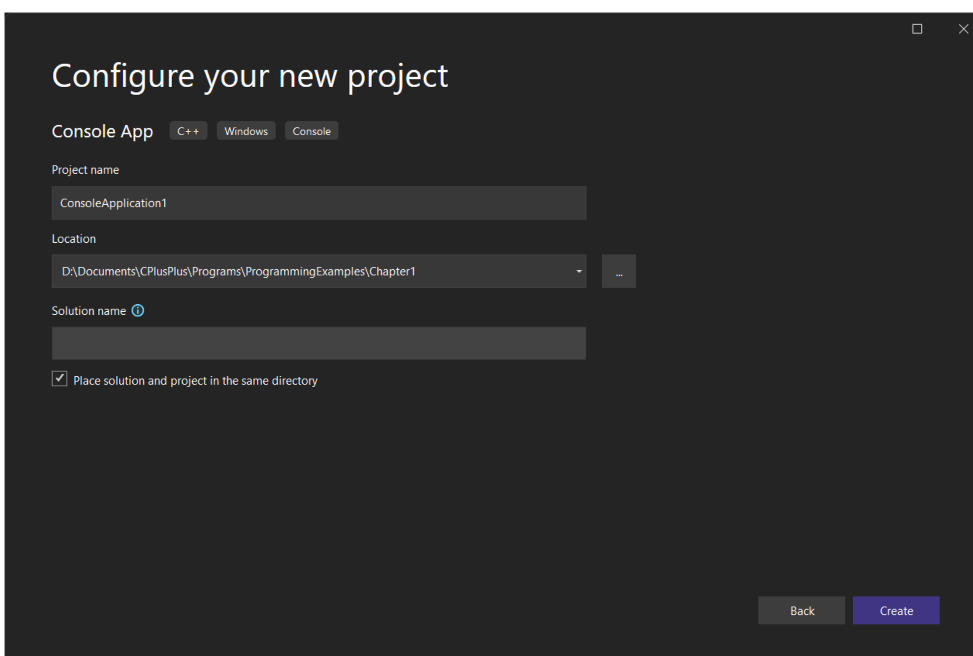
© Copyrights 2015-26 Nilapatri (Praajna Pattada)



When the next screen appears, select 'C++' in the first box on the top, to select the programming language. Select 'Console App' as the type of project, as all applications we shall be using in this project are console programs, as they run on the command-line.



You will then need to name your project and select the folder to store it in.



© Copyrights 2015-26 Nilapatri (Praajna Pattada)

You can select the option to keep the project and solution in the same directory (folder), since our programs shall not be using multiple project files. A solution in Visual Studio is a larger project file which organizes several other projects, if they are being used for a single application. Click on 'Create' to create your project. You will get a default "Hello World" program with pre-written code. You can erase all of the code and later write your own, as explained in the next section.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Writing your first Program in C++

Most programmers begin with a simple program which says, "Hello, world!" Thus, this book also contains the code for the same. Make a new file with the extension '.cpp'. We shall use the name '`HelloWorld.cpp`'. This is the standard extension for C++ program files. If file extensions are not visible on your computer, open 'File and Folder Options' and make sure that the option 'Hide extensions for known file types' is unchecked. In the file, write the following code. The line numbers are provided on the left column, for your convenience.

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello, world!";
6      return 0;
7  }
```

Compile the program and run it. If you are using G++ and open the executable (.exe) file manually, a command prompt window may open and close quickly. To watch the output properly, open command prompt, switch to the directory (folder) where your program's compiled executable is stored and then run it, as shown on page 15. For example, if the program is called 'HelloWorld.exe', type '`HelloWorld.exe`' (you can also omit the '.exe' extension) and press the enter key. The program will now display the words, "Hello, world!"

```
D:\Documents\CPlusPlus\Programs\ProgrammingExamples>g++ HelloWorld.cpp -o HelloWorld.exe

D:\Documents\CPlusPlus\Programs\ProgrammingExamples>HelloWorld.exe
Hello, world!

D:\Documents\CPlusPlus\Programs\ProgrammingExamples>
```

Now let us understand what this program says.

- The first line is a '`#include`' statement, which says that we shall be using the library for input and output, known as '`iostream`' (Input-Output stream).
- Line 3 declares and defines a `function`, known as 'main'. Line 4 opens its definition using a curly bracket.
- Line 5 declare that the program should provide the output saying "Hello World!" on the screen. This is done using the statement '`cout`' (pronounced 'C-out'), which is part of the namespace '`std`'. This statement belongs to the library called '`iostream`'.
- Line 6 says '`return 0`', implying that the program ends here and has run successfully. Zero is the code to indicate that the program terminated successfully.
- The last line, 7, uses the curly bracket to end the function '`main`'.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Let us now modify this program to include another feature. After the first include statement, add the other two statements shown below. Then, add the other lines accordingly.

```
1  // A program to say 'Hello, world!'
2  
3  #include <iostream>
4  #include <cstdlib> /* Can also use the C header 'stdlib.h' */
5  using namespace std;
6  
7  int main()
8  {
9      cout << "Hello, world!" << '\n';
10     system("PAUSE"); // Works only on Windows
11     return 0;
12 }
```

Compile the program and observe its output.

```
D:\Documents\CPlusPlus\Programs\ProgrammingExamples>g++ HelloWorld.cpp -o HelloWorld.exe

D:\Documents\CPlusPlus\Programs\ProgrammingExamples>HelloWorld.exe
Hello, world!
Press any key to continue . . .

D:\Documents\CPlusPlus\Programs\ProgrammingExamples>
```

Notice the differences here:

- The text within '/*' and '*/' is considered a comment in C. Comments are part of the code which are not considered a proper part of the program. They are ignored by the compiler and meant to be notes for the programmer or anyone else reading the code, to understand anything relevant to the context. In C++, two forward-slashes (//) imply that anything written after them in that entire line, is a comment. In C, you need to specify the start and end of a comment, using '/*' for opening it and '*/' for closing it.
- In the beginning, we used the statement '`using namespace std;`'. This implies that any statements within the namespace '`std`' can be used in this program. For this reason, we can only say '`cout`' instead of '`std::cout`'. A namespace is a group or category of specific statements, functions or variables. They are commonly used to organize the code of large programs.
- After `cout`, we used '\n', which ends the line and moves the cursor to the next line, before anything else is displayed on the screen. You can also use \n within the quotation marks, just after "Hello, world!" For example, the following line also performs the same function:

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
cout << "Hello, world!\n";
```

- An alternative to this is 'std::endl', which is not preferred as it requires iostream and its mechanism is comparatively slower.

- The additional statement, 'system("PAUSE")' instructs that the program must not close after its execution, but wait for the user to provide an input, by pressing a key. This prevents the issue of the command prompt window opening and closing in a fraction of a second, when you directly launch the executable file. For this to work, we need another library, 'cstdlib'. In C programs, you can use the library 'stdlib.h'. Since a valid C program also works in C++, you can try using the same and the code will run the same way.

Most statements, such as 'cout', end with a semicolon (;). This declares that the statement has finished.

Now, note that the statement 'system("PAUSE");' is not considered ideal, as it uses a different and complicated library for a simple action. It is also specific to Windows and will not work on other operating systems. If you run the program within a modern IDE, it will usually remain open and say, 'Press any key to continue...' However, this is not always the case and since we do not want the program to close immediately, we can adopt a different strategy. Replace the 'system()' statements accordingly, as shown below. Instead of the library 'cstdlib', use 'limits'.

```
1   // A program to say 'Hello, world!'
2   
3   #include <iostream>
4   #include <limits>
5   using namespace std;
6   
7   int main()
8   {
9       cout << "Hello, world!" << '\n';
10      cin.clear();
11      cin.ignore(numeric_limits<streamsize>::max(), '\n'); /*
    Ignores any additional characters except a new line */
12      cin.get(); /* Get one more input from the user (i.e., wait
    till the user presses the enter key) */
13      return 0;
14  }
```

Now, this is a universal method of achieving the same. After the words, you can type anything and it will be ignored. You simply need to press enter to close the program. On some computers, you may need to press the enter key twice. Note that 'cin' (pronounced

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

'C-in', which is a statement for receiving input), 'numeric_limits' and 'streamsize' are all part of the 'std' namespace.

Also note that the most basic form of a C++ program is actually the following:

```
1  int main()
2  {
3      return 0;
4  }
```

A program only needs to have a main() function and preferably, use the return statement to signify the successful execution of the program. However, this program uses no libraries and has no other relevant statements to provide any visible input or output to the user. Certain modern compilers even consider the return 0; statement to be implicit, if it is not mentioned in the program.

While compiling your code on an IDE, you can also observe that there is an option called 'Build Configuration' or 'Target', with 'Debug' and 'Release' as the most common options. These are profiles, or collections of settings, which determine the platform your program targets. Debug is used to indicate that the program is still being tested and you are looking to identify any issues. Release is used to indicate that it is the final version of the program you would like to release publicly or for use by others.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Ideal Programming Practices

There are certain ideal programming practices to be followed, for best results:

- Always use a proper naming convention for your files or the components of your program.
- While learning, always manually type the code from this e-book on to your program file, as you will also become efficient in understanding and writing the code, rather than merely copying.
- Always initialize a variable while creating one, by giving it some value, such as 0, true/false, etc.
- Never ignore compiler warnings when you get them, even though your code would have compiled successfully. They indicate an issue in the method your program uses.
- Use a good font for your text editor, which can properly differentiate between ambiguous characters such as lowercase-L and uppercase-I, zero and 'O', etc. Consolas is a good example of such fonts.
- Always try to solve the questions/practice exercises provided in this book by yourself. Once you are done, verify your results with the solutions provided at the end.
- When the compilation of your code fails, relax and reexamine your code. The compiler's error logs provide important information on which part of your code has any issue. Make sure to rule out mistakes in your algorithm, spelling mistakes, etc.

With this in mind at all times, programming should be easier to learn, not only in the case of C++.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Chapter 2: Fundamentals of C++ Programming

Structure of a Program

A computer program must follow a standardized structure to be intelligible to a compiler, in order to be properly converted to machine code. In C++, a program must have a '`main`' function and include any necessary libraries or definitions in the beginning. The most basic part of a program is a statement. A **statement** is a line of code in a program, which provides a specific instruction. It is like a building block for a program. As said previously, most statements generally end with a semicolon (`;`). Each statement can be converted to machine code by a compiler to achieve specific results.

Larger applications can have multiple source files of C++ programs, which work together as a single unit. A C++ program is written in a **source file**, which generally can have the extensions .cpp, .cc, .cxx or .cp. A program can also use other files containing some relevant pre-defined code. These are known as '**Headers**'. They have the extension .h or .hpp, or can have no extension at all. In the previous chapter, we already used the header '`iostream`', which contains the definitions for the statement '`std::cout`' and other necessary statements to enable input and output.

Generally, a typical statement used in C++ can be any of the following:

- Declaration statement
- Jump statement
- Expression statement
- Compound statement
- Selection statement (conditional)
- Iteration statement (loop)
- Try blocks

A statement must be written using specific keywords. They are case-sensitive words recognized to be referring to specific instructions, by a compiler.

A **function** is a set of specific statements in a particular order. Each C++ program must have a function called '`main`', whose statements are executed automatically and sequentially, from top to bottom, upon running a program. After the last statement of the main function has been executed, the program ends.

An **identifier** refers to a unique name we use to refer to an entity such as a variable or function, in a program. For example, '`main`' is an identifier and the function with this name

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

is automatically executed in any C++ program. An identifier must always begin with a letter. It can only contain an underscore, number or letter. It should not contain a whitespace or be the same as a keyword. Identifiers are also case-sensitive and specific to an particular entity. Ideally, it is advisable to keep all their letters lowercase.

A **declaration** is a statement which declares that a particular entity exists and assigns an identifier for the same. It can also assign a certain value. For example, a statement which defines a variable, such as '`int a = 1;`'. Here, 'a' is the identifier.

A **definition** is a statement which defines a function or some action with respect to a particular identifier. It can also assign it a certain value. For example, the code within the curly brackets (`{ }`) of the `main` function in our first program, is a definition.

> ***The One-Definition Rule (ODR)****: It is a rule followed in C++ programming, which states that a single variable, function, etc., can be defined only once. A definition occurring within a specific scope does not violate this. (For example, a local variable called 'a' in a particular function does not conflict with another variable 'a' within the same program)*

An **object** is a part of the computer memory (RAM), where a particular value can be stored. A variable is a type of object.

An **escape sequence** is a set of characters which perform special functions. They begin with '\'. '\n' is an escape sequence to create a new line and '`\t`' is an escape sequence to create a tab-space.

A **whitespace** is a blank space in a C++ program. It is of vital importance in some contexts, as a small change can make a drastic difference. For example, when we define a variable called 'a', with 1 as its value, we should use the declaration statement '`int a = 1;`' and not something like '`inta=1;`', as the compiler understands it as a keyword '`inta`', instead of '`int`' (integer). But this is not applicable for text in quotation marks. Previously, we wrote a cout statement to display the text "Hello, world!" Had there been two or three spaces in between the words 'Hello' and 'world', the same extra spaces would have been displayed on the screen. Note that whitespaces outside of quotation marks do not count as a new line, even if they are written on separate lines. For example, in the following code, the screen will display the words, "This is line 1.This is line 2."

```
std::cout << "This is line 1.";
std::cout << "This is line 2.";
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

However, if one uses '`\n`' or '`endl`' after the first line, the two lines are displayed on two separate lines. If you use the following code, the two lines are displayed on the screen as two separate lines:

```
std::cout << "This is line 1." << '\n';
std::cout << "This is line 2." << '\n';
```

> *Note that comments and extra white-spaces are ignored by the compiler, as they are deleted by the lexical analyser, which is why they are meant only for easy understanding of the programmer or anyone else reading the code.*

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Keywords

Keywords refer to the specific case-sensitive words that C++ compilers understand as part of the programming language and to be referring to specific instructions. Like identifiers, keywords are also case-sensitive. As of the C++23 standards, the following is a list of all the keywords in C++:

`alignas`
`alignof`
`and`
`and_eq`
`asm`
`auto`
`bitand`
`bitor`
`bool`
`break`
`case`
`catch`
`char`
`char8_t` (since C++20)
`char16_t`
`char32_t`
`class`
`compl`
`concept` (since C++20)
`const`
`consteval` (since C++20)
`constexpr`
`constinit` (since C++20)
`const_cast`
`continue`
`co_await` (since C++20)
`co_return` (since C++20)
`co_yield` (since C++20)
`decltype`
`default`
`delete`
`do`
`double`
`dynamic_cast`
`else`
`enum`
`explicit`
`export`
`extern`
`false`
`float`
`for`
`friend`
`goto`
`if`
`inline`
`int`
`long`
`mutable`
`namespace`
`new`
`noexcept`
`not`
`not_eq`
`nullptr`
`operator`
`or`
`or_eq`
`private`
`protected`
`public`
`register`
`reinterpret_cast`
`requires (since C++20)`
`return`
`short`
`signed`
`sizeof`
`static`
`static_assert`
`static_cast`
`struct`
`switch`
`template`
`this`
`thread_local`
`throw`
`true`
`try`
`typedef`
`typeid`
`typename`
`union`
`unsigned`
`using`
`virtual`
`void`
`volatile`
`wchar_t`
`while`
`xor`
`xor_eq`

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

C++20 refers to the programming standard of C++, which was introduced in 2020. If your compiler does not support it or the option to use it is turned off, the keywords mentioned to be specific to C++20 will not work.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Variables/Data Types

A variable is a container which stores a certain value. Note that these are also categories of data types/fundamental types, which are attributes of an entity and define what kind of data it handles. These are also used for other entities, which you will learn through the course of the subsequent sections. They are of the following common types:

Type of Variable/Data Type		Minimum Size (Bytes)	Typical Size (Bytes)	Keyword	Possible Values and Range
Boolean		1	1	`bool`	true/false
Character (UTF-8)	UTF-8 Character	1	1	`char`	A single ASCII character, such as a letter, number, symbol or whitespace
	UTF-16 Character	2	2	`char16_t`	Supports UTF-16 characters
	UTF-32 Character	4	4	`char32_t`	Supports UTF-32 characters
	Wide Character	1	2/4	`wchar_t`	
Integer	Short integer	2	2	`short`	Any integer between -32768 to +32767
	Integer	2	4	`int`	Any integer between -32768 to +32767
	Long integer	4	4/8	`long`	Any integer between -2147483648 to +2147483647
	Long long integer	8	8	`long long`	Any integer up to 64 bits, i.e., -2^{63} to $(+2^{63} – 1)$
Floating-Point	Float (4 Bytes)	4	4	`float`	Any decimal number with up to six significant figures after decimals, for proper accuracy

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

	Double (8 Bytes)	8	8	`double`	A decimal number, accurate up to 15 decimals
	Long Double (10 Bytes)	8	8/12/16	`long double`	Accurate up to 19 digits
Pointer		4	4/8	`std::nullptr_t`	A reference to another variable

As said previously, an object is a value stored in the computer's memory. When we define a variable or some entity in our program with a certain data type, we allocate a certain amount of the computer's RAM to it. On most computers, which typically have 4 GB or more RAM nowadays, these are negligible. The smallest unit of computer memory is a bit, which can have two potential values – `0` or `1`. 8 bits make one byte and so, one byte can hold 256 different values. The formula to know the possible values of a certain number of bits is 2^n (where *n* refers to the number of bits).

To define a variable, one only needs to type the keyword and mention an identifier they like. If we were to define two integer variables a and b, with the value 1, we would use this:

```
int a = 1;
int b = 1;
```

Alternatively, since both are the same type of variable, we can also use the following:

```
int a = 1, b = 1;
```

Note that using only '`int a, b = 1;`' will set the value of 'b' to 1, but not set the value of 'a' to 1. If you simply use '`int a, b;`' or '`int a; int b;`', these variables will be uninitialized, as they do not have any initial value set. This can cause potential problems when running a program, as they can automatically be assigned random values by the computer, called 'garbage values'. Thus, it is always recommended to initialize any variable while declaring them. The first method we used here is called 'copy initialization'. Some other forms of initialization are as follows:

```
int a ( 1 ); // Direct initialization
int b { 1 }; // Direct-list initialization
int c {}; // Zero initialization
```

In the last line, though we do not specify a value for integer 'c', it does not take up a garbage value and is initialized to zero. Nonetheless, if you wish to use the value zero later in your program, it is best to specify it within the braces of c.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

By default, a variable when defined, is understood to be 'signed'. To make them unsigned, one needs to add the keyword '`unsigned`' before int, char, etc. By doing so, they can store only positive values and their range of values increases. For example, a signed int can store values from -32768 to +32767, while an unsigned int can store values from 0 to +65535. The basis for this nomenclature is that we require either a + or - sign to tell if a number is positive or negative. If there is no sign, it is assumed to be positive. So, the keyword `unsigned` changes the range of an integer variable to hold only positive values and not negative numbers. This makes it easier in some cases, for one to store positive values beyond the range of a typical short integer or other small variables, without using more memory to create a larger variable. For certain variables, trying to set their value to a number beyond their range will result in undefined behaviour, which can change their value to anything randomly. This is known as 'overflow'.

It is always recommended, as it is beneficial for typical programs, to avoid using unsigned integers and use only signed integers. There are few cases nonetheless, where using unsigned integers is absolutely required, such as Array indexing, which shall be explained subsequently in this book.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Constants

Certain values remain the same, regardless of the context, unlike variables. These are known as 'constants'. Certain variables may also store such fixed values, which will not change at any point of time, such as the value of Pi or some other value in your program which you want to be permanently fixed. Accidentally using the set operator (equals symbol) in any statement in your program will set a new value to the variable. To avoid this, one can define the variable itself as a 'named constant'. There are two types of constant values which we can use:

- Named constants, which have an identifier associated with it
- Literal constants, which are the constant values and have no associated identifier

In the following code, a new variable called 'pi' is created and its value is changed to 5.5.

```
double pi { 3.14159 };
pi = 5.5;
```

This obviously does not make sense, but is not considered an issue by the compiler. To avoid this for such obvious values, we can make the variable 'pi' a constant variable. To do this, one only needs to add the keyword '`const`' before it.

```
const double pi { 3.14159 };
```

Now, the variable 'pi' is a named constant. The numbers 3.14159 and 5 are literal constants. They do not have any associated identifiers, as they literally refer to those numerical values. Note that constant variables have to be initialized with their permanent values. It is not possible to change their values later nor leave them uninitialized.

Another form of creating constant variables is using the keyword `constexpr`. However, you must initialize these variables within your program itself with a constant expression, as the variable must have a fixed value during compilation itself. You cannot initialize them with the value of another variable, since it can be subject to change during execution of the program. You can also use the `return` value of a function or the value of another variable to initialize them, provided those functions and variables are also of the type `constexpr`. The key difference between entities which are `const` and `constexpr` is that the former can be initialized even at runtime, while the latter has to be initialized during compilation itself.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Operators

Operators are specific characters used to perform certain operations. The most common operators in C++ are as follows, some of them being the same as the commonly used mathematical operators:

- + for addition
- - for subtraction
- * for multiplication
- / for division
- = to set a value of a variable
- == to check if a value is equal to something
- != to check if a value is not equal to something
- > to check if a value is greater than something
- >= to check if a value is greater than or equal to something
- < to check if the value of a variable is lesser than something
- <= to check if a value is lesser than or equal to something
- ++ to increase the value of certain variables by 1
- -- to decrease the value of certain variables by 1

The following code creates a variable 'c' and sets its value to the sum of a and b, which would be 50. Note that it uses the set operator (=) and so, this is not considered to be initialized with the value of `a + b`. Rather, through this method, it is created as an uninitialized variable and its value is manually changed to the sum of a and b (i.e., 50).

```
int a { 10 };
int b { 40 };
int c = a + b;
```

This code displays the same on the screen.

```
std::cout << a + b;
```

Both of the following lines set the value of the variable to 100.

```
c = 100;
c = 10 * 10;
```

Through the course of this book, you shall use these operators frequently and also learn about others.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Escape Sequences

Escape sequences are certain codes which correspond to specific characters. They are useful to display certain characters, without altering the syntax of the program. We already discussed about the escape sequences \n and \t. Following is a list of various escape sequences in C++:

Escape Sequence	Code
Alert	\a
Backspace	\b
Form feed	\f
Line feed (new line)	\n
Carriage return	\r
Horizontal tab	\t
Vertical tab	\v
Backslash	\\
Single-quote (')	\'
Double-quote (")	\"
Question mark (?)	\?
String terminating character	\0
Numerical value of a character (up to 3 octal digits)	\ooo
Numerical value of a character (in hexadecimal digits)	\xhh

The following program provides the output shown below:

```
1   #include <iostream>
2   #include <limits>
3   using namespace std;
4   
5   int main()
6   {
7       cout << "These are a few of the escape sequences in C++" <<
    '\n';
8       cout << '\n' << "This is a new line" << '\n' << '\n';
9       cout << "This is a " << '\b' << "backspace." << '\n';
10      cout << "This is a " << '\t' << "horizontal tab" << '\n';
11      cout << "This is a " << '\v' << "vertical tab" << '\n';
12      cout << "This is a single quote: " << '\'' << '\n';
13      cout << "This is a double quote: " << '\"' << '\n';
14      cout << "This is a question mark: " << '\?' << '\n';
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
15        cin.clear();
16        cin.ignore(numeric_limits<streamsize>::max(), '\n');
17        cin.get();
18        return 0;
19    }
```

```
These are a few of the escape sequences in C++:

This is a new line

This is abackspace.
This is a       horizontal tab
This is a
          vertical tab
This is a single quote: '
This is a double quote: "
This is a question mark: ?
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Input and Output using Variables

Let us make a program to obtain a value from the user and display it and its products with 5 and 10.

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4  
5  int a {};
6  
7  int main()
8  {
9      cout << "Enter any integer: ";
10     cin >> a;
11     cout << '\n' << "You have entered " << a << '\n';
12     cout << a << " multiplied by five is " << a * 5 << '\n';
13     cout << a << " multiplied by ten is " << a * 10 << '\n';
14     cin.clear();
15     cin.ignore(numeric_limits<streamsize>::max(), '\n');
16     cin.get();
17     return 0;
18 }
```

Compile the program and upon running it, you will be asked to enter a number. Upon typing it and pressing enter, you will obtain the products of the number with 5 and 10. Note that if you try it with extremely large numbers, you will obtain incorrect values. If you enter any characters other than numbers, the program provides incorrect values. If you enter a decimal number, the decimal and the digits after it are ignored.

Now, if you wish to enable this program to also make the same calculations with decimal numbers, simply change the datatype of the variable 'a' from `int` to `float`.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Header Files

In C++, a header is a library or a separate set of pre-written code which can be included in a program. `iostream`, `cstdlib` and `limits` are the headers we have used till now, in our programs. Similarly, there are several other header files which are part of the standard library of C++. You can also find them in the directory of your compiler, usually named '`includes`'. To include a header in your program, you simply need to type `#include` and the name of the header file within double-quotes or the symbols '<' and '>'.

Header files are useful to split pieces of code which may not always be necessary for use in all of your programs, or to import them in multiple programs. You can create your own header files and add it to your project as well. If you are using an IDE, make sure to use the options to create a new file and select 'Header' for the file type. On simpler compilers, make sure both your source file and header file are in the same folder.

Let us take the following example. This program calculates the area of a rectangle.

```
1   #include <iostream>
2   #include <limits>
3   using namespace std;
4
5   float length { 0.0 };
6   float width { 0.0 };
7
8   int main()
9   {
10      cout << "Enter the length of a rectangle: ";
11      cin >> length;
12      cout << '\n' << "Enter the width of a rectangle: ";
13      cin >> width;
14      cout << '\n' << "The length is " << length << '\n';
15      cout << "The width is " << width << '\n';
16      cout << "The area is " << (length * width) << '\n';
17      cin.clear();
18      cin.ignore(numeric_limits<streamsize>::max(), '\n');
19      cin.get();
20      return 0;
21  }
```

While this program works, the code may become confusing as more functions and other complex features are added. One can make the code simpler by making a header file specifically with those. For this example, let us make a separate function to calculate the area, called `CalculateArea()`, which multiplies the length and width. The program would be like this:

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4  
5  float length { 0.0 };
6  float width { 0.0 };
7  
8  float CalculateArea(float length, float width)
9  {
10     return (length * width);
11 }
12 
13 int main()
14 {
15     cout << "Enter the length of a rectangle: ";
16     cin >> length;
17     cout << '\n' << "Enter the width of a rectangle: ";
18     cin >> width;
19     cout << '\n' << "The length is " << length << '\n';
20     cout << "The width is " << width << '\n';
21     cout << "The area is " << CalculateArea(length, width) <<
   '\n';
22     cin.clear();
23     cin.ignore(numeric_limits<streamsize>::max(), '\n');
24     cin.get();
25     return 0;
26 }
```

This mechanism is useful for complex and larger projects, where frequently used code, especially functions or other data types can be stored in a header file and accessed by other multiple programs, as required.

Observe the variables `length` and `width` here. `CalculateArea()` is a function which multiples two float values which are given to it, when calling the function, and returns their product as a float value. The values provided to that function are also named `length` and `width`, but note that they are <u>local variables specific to that function</u>. They do not conflict with the main variables in our program, which have the same names. Now, this same code can be simplified by making a separate header with this function. Make a new file with the same name and give it the extension `.h`. Since in our example, we have named the source file '`RectangleArea.cpp`', we shall use the same name for the header, '`RectangleArea.h`'.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

RectangleArea.h

```
1   float CalculateArea(float length, float width)
2   {
3       return (length * width);
4   }
```

RectangleArea.cpp

```
1   #include <iostream>
2   #include <limits>
3   #include "RectangleArea.h"
4   using namespace std;
5   
6   float length { 0.0 };
7   float width { 0.0 };
8   
9   int main()
10  {
11      cout << "Enter the length of a rectangle: ";
12      cin >> length;
13      cout << '\n' << "Enter the width of a rectangle: ";
14      cin >> width;
15      cout << '\n' << "The length is " << length << '\n';
16      cout << "The width is " << width << '\n';
17      cout << "The area is " << CalculateArea(length, width) <<
    '\n';
18      cin.clear();
19      cin.ignore(numeric_limits<streamsize>::max(), '\n');
20      cin.get();
21      return 0;
22  }
```

Compile and run your program, and it should provide the same results. Now, ideally, you must use 'Header Guards' with your program. These is an additional feature which prevents erroneously redefining a certain code in larger programs with multiple header or source files. For example, let us suppose we have another header in our same program, 'CalculateArea.h', which has the following line:

```
#include "RectangleArea.h"
```

Now suppose you use the following code in the beginning of your source file:

```
#include <iostream>
#include <limits>
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
#include "RectangleArea.h"
#include "CalculateArea.h"
```

The compiler understands the rest of the code to be as follows:

```
5   using namespace std;
6
7   float CalculateArea(float length, float width)
8   {
9       return (length * width);
10  } /* Obtained from RectangleArea.h */
11
12  float CalculateArea(float length, float width)
13  {
14      return length * width;
15  } /* Obtained from CalculateArea.h */
16
17  float length { 0.0 };
18  float width { 0.0 };
19
20  int main()
21  {
22      cout << "Enter the length of a rectangle: ";
23      cin >> length;
24      cout << '\n' << "Enter the width of a rectangle: ";
25      cin >> width;
26      cout << '\n' << "The length is " << length << '\n';
27      cout << "The width is " << width << '\n';
28      cout << "The area is " << CalculateArea(length, width) <<
    '\n';
29      cin.clear();
30      cin.ignore(numeric_limits<streamsize>::max(), '\n');
31      cin.get();
32      return 0;
33  }
```

This is an obvious violation of the one-definition rule and causes a compilation error. To prevent this, we can modify the header files a little, by adding a header guard. Change the code to the following, in both header files:

```
1   #ifndef AREA_CALCULATOR
2   #define AREA_CALCULATOR
3
4   float CalculateArea(float length, float width)
5   {
6       return (length * width);
7   }
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
8
9   #endif
```

The name 'AREA_CALCULATOR' is an identifier for the header guard. This can be any name, preferably in capital letters. Here, the header file verifies if the same identifier (AREA_CALCULATOR) has been defined elsewhere and if not, it defines it and uses the code within it, till #endif.

Now even if you use both #include "RectangleArea.h" and #include "CalculateArea.h" in your code, there will be no errors, as the definition in the second header will be ignored. Remember to always use header guards as much as possible, to prevent the possibility of such errors.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Using Multiple Source Files

Larger applications use multiple source files and not a single one, to simplify their code. The main program contains the main() function, which acts like a central hub or focal point, calls other functions from the other program files and controls all other actions, as required. Let us take a simple example.

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4  
5  void DisplayText()
6  {
7      cout << "This program has run successfully." << '\n';
8      cin.clear();
9      cin.ignore(numeric_limits<streamsize>::max(), '\n');
10     cin.get();
11 }
12 
13 int main()
14 {
15     DisplayText();
16     return 0;
17 }
```

`DisplayText()` is a function (with the datatype `void`, as it does not return any information), which displays the text that the program ran successfully and waits for the user’s input. The `main` function simply executes this program, which is essentially the same as the same statements being written in the `main` function itself.

Now, we can split this application into multiple program files, as an example. Make a new file by any name ending with '.cpp' and keep it in the same folder. If you are using an IDE, simply go to the folder named 'Source' or wherever your source file is, right-click or use other options to create a new file and select the options to create a new 'Source file'. On IDEs, multiple source files are included in your application automatically, while compiling. In the new file (which we shall name '`Function.cpp`'), add the following code:

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4  
5  void DisplayText()
6  {
7      cout << "This program has run successfully." << '\n';
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
8       cin.clear();
9       cin.ignore(numeric_limits<streamsize>::max(), '\n');
10      cin.get();
11  }
```

Now, in your main program, write only the following code:

```
1   void DisplayText();
2   
3   int main()
4   {
5       DisplayText();
6       return 0;
7   }
```

Note that in your main program, you still have to declare the function '`DisplayText()`', to tell the compiler that the definition for the same can be found elsewhere and to prevent the compiler from reading the main program file first and generating errors due to being unable to find the definition for the same. Here, the `DisplayText()` function is defined elsewhere and all the code for the same is in the other file. On G++ and similar compilers, you will need to use the following command to compile your program:

```
g++ Program.cpp Function.cpp -o MultifileProject.exe
```

> *Alternatively, for larger projects with a lot of source files, simply use '`*.cpp`', instead of naming each file. This selects all the files in that folder which have the extension `.cpp`.*

Run the program and you can observe that it displays the mentioned text and waits for your input.

```
This program has run successfully.
|
```

This is the same mechanism for using multiple source files in your projects. Note that for certain objects which are common to multiple programs or need to be defined properly, using custom header files may be more convenient, compared to using multiple source files. Using multiple source files can enable one to split the code of large programs, while using multiple headers helps one to declare and define certain entities like variables, functions, etc., which can be reused later. The need to do so depends on your type of project.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Namespaces

As mentioned previously, a namespace is a category of statements, especially declarations and definitions. The most useful function of a namespace is that it can be used to classify different entities with similar names. It provides a scope to certain identifiers. For example, we already used the namespace 'std' in our previous programs, which contains definitions for cin, cout, etc. If we simply use the statements cin or cout, they would not work, because they are defined within the namespace std. So, one must use either 'std::' before mentioning the statements or else mention that the namespace is in use, by stating 'using namespace std;'. Now, look at the following program:

```
1  #include <iostream>
2  #include <limits>
3  
4  namespace CustomNamespace
5  {
6      int a { 1 };
7  }
8  
9  namespace SecondNamespace
10 {
11     int a { 4 };
12 }
13 
14 int main()
15 {
16     std::cout << "The first variable \'a\' has the value " <<
   CustomNamespace::a << '\n';
17     std::cout << "The second variable \'a\' has the value " <<
   SecondNamespace::a << '\n';
18     std::cin.clear();
19     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
20     std::cin.get();
21     return 0;
22 }
```

Notice that we have two variables 'a', with the values 1 and 4. However, despite having the same identifier, there is no error (called 'naming collision' or 'naming conflict'). This is because one variable is defined within the scope of the namespace called CustomNamespace and another is defined within SecondNamespace. You can use a similar practice to declare and define other entities too, such as functions, classes and other compound data types, within specific namespaces.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Note that in this program, to prevent unnecessary complications and as it is not a good practice, we have omitted the statement `using namespace std;`, which is why we have manually specified `std::` before using its defined functions and objects. When you run the program, you get the following output:

```
The first variable 'a' has the value 1
The second variable 'a' has the value 4
```

Practice Question 1: Modify this program such that instead of using the values 1 and 4 respectively for the variables 'a', you ask the user to enter two values (which can be even decimals) and then display them on the screen.

Refer to page 109 for the solution.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Chapter 3: Other Types of Statements

Declarations

By now, you should be well-aware of declarations, as you have already used them multiple times in your programs. A declaration is a statement which declares something. One can notice that is one of the most frequently used statements in a typical C++ program. To recap, the following are declarations:

```
float a {};
int b {};
void DisplayText();
```

The following is a definition for the function `DisplayText()`:

```
void DisplayText()
{
    cout << "This program has run successfully." << '\n';
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cin.get();
}
```

A definition is also a type of declaration, which defines a program. There are certain rules with declarations, which must be kept in mind at all times. Let us now get back to our previous program, where we made a function to display some text on the screen:

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4
5  void DisplayText()
6  {
7      cout << "This program has run successfully." << '\n';
8      cin.clear();
9      cin.ignore(numeric_limits<streamsize>::max(), '\n');
10     cin.get();
11 }
12
13 int main()
14 {
15     DisplayText();
16     return 0;
17 }
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Did you notice that we defined the function `DisplayText()`, before the `main()` function? What would happen if `main()` occurred before `DisplayText()`, as shown below?

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4
5  int main()
6  {
7      DisplayText();
8      return 0;
9  }
10
11 void DisplayText()
12 {
13     cout << "This program has run successfully." << '\n';
14     cin.clear();
15     cin.ignore(numeric_limits<streamsize>::max(), '\n');
16     cin.get();
17 }
```

This code will not compile at all and provide an error, because the compiler always reads the code from top to bottom and finds errors at every line. It will notice that the `main()` function calls a function called `DisplayText()`, which does not even exist in the code it has read so far. Therefore, it is important to make declarations of any variable, function, class, etc., before using them in the lines below them.

But what would be an effective solution to the problem above? What if we wish to define the code for `DisplayText()` elsewhere, but still want to define `main()` first? The solution is actually simple and the same as the strategy used while using multiple source files, as explained on page 42. Simply add a declaration for `DisplayText()` before `main()`:

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4
5  void DisplayText();
6
7  int main()
8  {
9      DisplayText();
10     return 0;
11 }
12
13 void DisplayText()
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
14 {
15     cout << "This program has run successfully." << '\n';
16     cin.clear();
17     cin.ignore(numeric_limits<streamsize>::max(), '\n');
18     cin.get();
19 }
```

These are known as '**forward declarations**'. They are highly useful for cases where you may not want to define a class, function or some other entity initially but still mention them before they are defined. This also prevents violation of the ODR. This already tells the compiler that there is a function by the name `DisplayText()`, which is not defined yet, but will be. Note that if you do not define the function later in your code, the program again fails to compile, as there is not even a blank definition for it.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Functions

A **function** is a set of statements which can be executed, as and when desired. To execute a function, one needs to call the function by mentioning its identifier. We have already used multiple functions of the data types `float`, `int` and `void`. The data type of a function depends upon what value it returns. In our previous program where we used multiple files, as explained on pages 41 and 42, the `main()` function is the caller and performs a function call for the function `DisplayText()`. The following is the syntax of a function.

```
datatype identifier(arguments)
{
    // Definition/Scope
    // Here, one can write the statements which one would want the
function to execute
    // At the end, use the 'return' statement to return some value to
the caller, if the function’s datatype is not 'void'
}
```

`void` is a data type which simply indicates that we do not want it to return any value to the caller. It will simply perform the actions mentioned in its scope. While calling the function, it is always necessary to use brackets for the function call to work. Also remember that any variables or other definitions specifically within the scope of a function are local variables and are specific to that function.

Let us go back to our previous program on page 38, where we used a custom header file and created a function for the same, with the data type `float`.

```
float CalculateArea(float length, float width)
{
    return (length * width);
}
```

Here, `CalculateArea` is the identifier and its arguments are the float variables `length` and `width`. Our program already had two variables by the same names, but these were not in conflict with the ODR, as the variables here are only defined within the scope of this function. The function understands `length` and `width` to be its own local variables whose values are provided by the caller of the function. In `RectangleArea.cpp`, we called this function through the `main()` function. While calling a function, we need to specify values for the arguments/parameters of the function, in the same sequence they are mentioned in the definition of the function. For this reason, in our previous program, we specified the length entered by the user and then the width entered by the user as the values of `length` and `width` for the `CalculateArea()` function. Then, this function simply multiplies the

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

values it is given while being called, through the `cout` statement. In `RectangleArea.cpp`, the variables `length` and `width` were global variables, as they were defined outside any function or class. Similarly, if they had been defined within the `main()` function, they would also be local variables specific to `main()`.

If no arguments are required for a function, the brackets need to be left blank. Remember that while calling a function, it is necessary to mention the empty brackets or specify some values for its variables (if there are any arguments and they do not have a default value). Otherwise, the function call will simply not work. Look at the following program:

```
int ASampleFunction(int a, int b, int c = 5)
{
    return (a + b + c);
}

int SecondFunction(int a = 1, int b = 2, int c = 5)
{
    return (a + b + c);
}

int ThirdFunction(int a = 1, int b, int c) /* This function
itself is incorrectly declared, as variables to the right of a
variable with a default value should also have default values,
but only a has a default value */
{
    return (a + b + c);
}

int main()
{
    int a = ASampleFunction(10, 10, 11); // Correct function call
- returns the value '31', by overriding the default value of c
(i.e., 5) with 11
    int b = ASampleFunction(10, 12); // Incorrect, as you cannot
mention values only for the variables on the right and skip the
ones on the left
    int c = ASampleFunction(, 12); // Incorrect, as you cannot
mention values only for the variables on the right and skip the
ones on the left
    int d = SecondFunction(1, 1, 1); // Correct, as it returns
'3' and overrides the values of all three variables with 1
    int e = SecondFunction(); // Correct, as the function uses
the default values of its variables and returns 8 (1 + 2 + 5)
    int f = SecondFunction(1, 3); // Correct, as the function
uses the default values of variable c (i.e., 5) and returns 9 (1
+ 3 + 5)
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
    int g = SecondFunction(5); // Correct, as the function uses the default values of b and c (i.e., 2 and 5) and returns 12 (5 + 2 + 5)
    int h = SecondFunction(1, , 3); // Incorrect, as you cannot skip specifying a value for a parameter (i.e., b), but specify a value for any parameter after it
}
```

You can observe here that all four functions use the variables a, b and c, but again, they do not violate the ODR, as they are all local variables defined only within the scope of their respective functions. The variables a, b and c declared in `main()`, `ASampleFunction()`, `SecondFunction()` and `ThirdFunction()` all differ from each other and are considered different variables. They are known only within the scope of their respective functions and available only to the statements within those functions. If your program needs to access a variable across different functions, you will need to make them global variables instead of local, similar to how we previously declared multiple variables outside any functions.

Also note that <u>you cannot make a function call to `main()` in any function and you cannot make functions nested in other functions</u>. The following shows such an example of these mistakes:

```
void Function1()
{
    void Function2()
    {

    }
}

void Function3()
{
    main();
}
```

Functions are useful when you use certain code repeatedly, as they can help simplify a program by using such code only once but ensuring that you can repeatedly perform the same actions whenever required, with just a function call.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Function Overloading

There may be cases where for our convenience, we would like to have multiple functions with the same identifier, but involve different data types. C++ supports this without any naming collisions. This is known as 'Function overloading'.

Let us see an example of this. We should make a program which asks a user to provide two numbers and then multiplies them and displays their product first and then their product converted to an integer. For the sake of example, instead of simply using a variable to convert the final product directly to `int`, let us use two separate functions for obtaining their products as decimal numbers and as integers. It would be convenient to name them the same. Since their datatype differs, it is possible to use the same name without any conflicts at all. While calling a function, the correct function is identified based on the following:

- Type of data being given while calling the function
- Number of parameters used to call the function (if there is a difference in them)

The following is an example program for this:

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4  
5  float a {};
6  float b {};
7  float c {};
8  int d {};
9  
10 float Multiply(float x, float y)
11 {
12     cout << '\n' << "Multiplying decimal values.";
13     return (x * y);
14 }
15 
16 int Multiply(int x, int y, bool isInteger)
17 {
18     cout << '\n' << "Multiplying integer values.";
19     return (x * y);
20 }
21 
22 int main()
23 {
24     cout << "Enter any number to multiply: ";
25     cin >> a;
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

26	`    cout << "Enter another number: ";`
27	`    cin >> b;`
28	`    c = Multiply(a, b); // Calls the function of datatype float, as there are only two arguments and c is also a float`
29	`    d = Multiply(a, b, true); // Calls the function of datatype int, since this has three arguments and so, the decimal values are converted to integers, ignoring the digits after the decimal point.`
30	`    cout << '\n' << "You have entered " << a << " and " << b << '\n';`
31	`    cout << "Their product is" << '\n' << c << " (as a decimal value)" << '\n' << d << " (as an integer value)" << '\n';`
32	
33	`    cin.clear();`
34	`    cin.ignore(numeric_limits<streamsize>::max(), '\n');`
35	`    cin.get();`
36	`    return 0;`
37	`}`

This will be its output, upon entering a decimal number.

```
Enter any number to multiply: 10.7
Enter another number: 5.5

Multiplying decimal values.
Multiplying integer values.
You have entered 10.7 and 5.5
Their product is
58.85 (as a decimal value)
50 (as an integer value)
```

The bool variable has been used in the `int` function to make it have three arguments, which is helpful in differentiating the two functions despite having the same identifier. You can also notice how the function of datatype `int` has considered the numbers to be 5 and 10, since integers omit any digits after the decimal point, thereby making a major difference in both the products obtained.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Conditional Statements

Conditional statements are those statements which check a given condition and execute specified actions accordingly. They are of two major types:

- If-Else if-Else
- Switch

If statements are the most commonly used, as they have a simple syntax and can be used for a variety of conditions with almost any variable. Let us suppose we have a program with two variables, a and b. The following code will ensure that their values are explained correctly, depending on which one is greater or if they are equal:

```
if (a >= b)
{
    cout << "The value of a is greater than or equal to that of
b.";
} else
{
    cout << "The value of a is lesser than that of b.";
}

if (a > b)
{
    cout << " The value of a is greater than that of b.";
} else if (a == b)
{
    cout << "The value of a is equal to that of b.";
} else
{
    cout << "a is lesser than b.";
}
```

In the first If and else blocks, the program simply checks if the values of a and b are equal or if a's value is greater than b's value. If not, it simply mentions that a is lesser than b. However, the next statement uses two different possibilities, using the `else if` statement. If the first condition is not true, the program moves to the scope of `else if`. When both are not true, the program executes the actions of the statements mentioned in the scope of `else`. Now, look at this code:

```
if (a > b)
{
    cout << "The value of a is greater than that of b.";
} else if (a == b)
{
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
    cout << "The value of a is equal to that of b.";
}
```

While this is also correct, if at all the variable a has a lesser value than that of b, the program performs no action at all and moves forward, ignoring the entire set of conditional statements, since there is no `else` statement. Note that you can make a chain of conditions by using a series of blocks of `else if` statements and terminating them with `else`, to specify what is to be done if none of them are true.

> *The keyword `continue` performs a similar function if used within a conditional block, by instructing the program to move on with the rest of the code.*

Another point to note is that if there is only a single statement for each condition, the statements `if`, else if and else statements can be used without making them into blocks, or in other words, without using curly brackets.

```
cin << a;
cin << b;

if (a > b)
    cout << " The value of a is greater than that of b.";
else if (a == b)
    cout << "The value of a is equal to that of b.";
else
    cout << "The value of a is lesser than that of b.";

return 0;
```

In this case, the program will execute only the first statement immediately after `if`, `else if` and `else`.

Now, another type of conditional statement is `switch`. It is convenient for cases where a limited set of possibilities of results are possible and you want specific results for a specific value of a variable. The following program uses the `switch` statement:

```
1   #include <iostream>
2   #include <limits>
3   using namespace std;
4
5   int a {};
6
7   int main()
8   {
9       cout << "Enter any integer lesser than or equal to 5: ";
10      cin >> a;
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
11
12      switch(a)
13      {
14          case 1:
15              cout << '\n' << "You have entered 1." << '\n';
16              break;
17
18          case 2:
19              cout << '\n' << "You have entered 2." << '\n';
20              break;
21
22          case 3:
23              cout << '\n' << "You have entered 3." << '\n';
24              break;
25
26          case 4:
27              cout << '\n' << "You have entered 4." << '\n';
28              break;
29
30          case 5:
31              cout << '\n' << "You have entered 5." << '\n';
32              break;
33
34          default:
35              cout << '\n' << "You have entered some other value."
    << '\n';
36              break;
37      }
38
39      cin.clear();
40      cin.ignore(numeric_limits<streamsize>::max(), '\n');
41      cin.get();
42      return 0;
43  }
```

Here, the switch statement checks for the mentioned possible values of a specified variable. You cannot use operators here, unlike If-Else statements. The `break;` statement signals the program to exit the series of checking for all the conditions. If you do not specify it, the program continues checking for the remaining conditions. If all of them do not match the given value, it refers to the `default` case and executes its statements.

If a conditional statement is within a function, you can also break the chain by using the `return` statement with the required value.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Loops

A **loop** is a set of statements which are executed repeatedly, as long as a given condition is true. There are three major types of loops in C++:

- For loops
- While loops
- Do-While loops

For loops are slightly more advanced among these. Their syntax is as follows:

```
for (initialization of a variable; condition; statement which is
executed after each round of the loop)
{
    // Statements to be executed repeatedly
}
```

A good example of a for loop is this:

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4  
5  int main()
6  {
7      for (int i = 1; i <= 10; i++)
8      {
9          cout << "This is round " << i << " of this loop." <<
   '\n';
10     }
11     cout << '\n' << "The loop has exited.";
12 
13     cin.clear();
14     cin.ignore(numeric_limits<streamsize>::max(), '\n');
15     cin.get();
16     return 0;
17 }
```

This program produces the following output:

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
This is round 1 of this loop.
This is round 2 of this loop.
This is round 3 of this loop.
This is round 4 of this loop.
This is round 5 of this loop.
This is round 6 of this loop.
This is round 7 of this loop.
This is round 8 of this loop.
This is round 9 of this loop.
This is round 10 of this loop.

The loop has exited.
```

The loop repeats the same statement over and over again, for a total of ten times, checking if the condition is satisfied each time. It stops when it is no longer the case. In the parameters of our loop, we first made a new local variable called 'i' and assigned it the value 1. Then, we specified the condition which must be satisfied for the loop to run, i.e., i should be less than or equal to 10. Finally, we mentioned a statement which is called after each execution, i.e., to increase the value of i by 1.

Now, in the parameters of the loop, if you change `i++` to `i = 11`, you will get the following output:

```
This is round 1 of this loop.

The loop has exited.
```

Initially, the value of `i` is set to 1 and the condition of `i` being lesser than 11 is satisfied. The `cout` statement is executed and then the value of `i` is immediately set to 11, which breaks the loop, as the condition is no longer satisfied.

A `while` loop has a simple syntax:

```
while (condition)
{
    // Statements to be executed repeatedly
}
```

The following is an example of a while loop:

```
1  #include <iostream>
2  #include <limits>
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
3  using namespace std;
4
5  int a {};
6  int i { 1 };
7
8  int main()
9  {
10     cout << "How many times do you want this loop to repeat?
   (Minimum 1 and maximum 10): ";
11     cin >> a;
12
13     if (a >= 1 && a <= 10)
14     { }
15     else if (a > 10)
16     {
17         a = 10;
18     }
19     else
20     {
21         a = 1;
22     }
23
24     while (i <= a)
25     {
26         cout << "This is round " << i << " of this loop." <<
   '\n';
27         i++;
28     }
29
30     cout << '\n' << "The loop has exited.";
31
32     cin.clear();
33     cin.ignore(numeric_limits<streamsize>::max(), '\n');
34     cin.get();
35     return 0;
36 }
```

When you run this program, it asks the user to provide a number between 1 to 10 and executes the same loop in the previous program the number of times specified by the user. Here, we have also added a set of `if`, `else if` and `else` blocks to change the value to either 1 or 10, if the user provides anything lesser or greater than those respectively, and to default to 1, if no proper integer is provided.

Remember that you can also a `for` loop with only a single condition and no other parameters, similar to a `while` loop, if you do not wish to create and initialize a variable or

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

have a specific statement (incrementing or decrementing the value of the variable) repeated. Simply leave the first and/or third arguments blank.

```
for (; i <= 10;)
{
    // Do something as long as the value of i is less than or equal to 10
    // Note that this loop will freeze your program and consume more and more memory, if you mention some statement which consumes memory, since there is no statement here which changes the value of i and alters the condition of i being lesser than or equal to 10
}
```

Additionally, you can configure a `while` loop to increment or decrement variables, if you like. In the previous program, try modifying it by removing the variable '`i`'. Change the loop to the following and add a line before it:

```
cout << "Counting..." << '\n';

while (a--)
{
    cout << "This is round " << a << " of this loop." << '\n';
}
```

Now, when you run the program, you get the following output:

```
How many times do you want this loop to repeat? (Minimum 1 and maximum 10): 10
Counting...
This is round 9 of this loop.
This is round 8 of this loop.
This is round 7 of this loop.
This is round 6 of this loop.
This is round 5 of this loop.
This is round 4 of this loop.
This is round 3 of this loop.
This is round 2 of this loop.
This is round 1 of this loop.
This is round 0 of this loop.

The loop has exited.
```

A do-`while` loop has a similar syntax:

```
do
{
    // Statements to be executed repeatedly or at least once
} while (condition);
```

Unlike a `while` loop, a do-`while` loop executes the statements once and then checks the condition. If the condition is true, it executes the specified actions a second time and so on.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

If the condition is false, it executes the statements once and terminates the loop. Note that a do-while loop must end with a semicolon.

In the following example, though the condition is false, the loop still gets executed once and then does not get executed again.

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4  
5  int main()
6  {
7      int i { 4 };
8  
9      do
10     {
11         cout << "Executing the loop." << '\n';
12     }  while (i <= 2);
13 
14     cout << '\n' << "The loop has exited.";
15 
16     cin.clear();
17     cin.ignore(numeric_limits<streamsize>::max(), '\n');
18     cin.get();
19     return 0;
20 }
```

```
Executing the loop.

The loop has exited.
```

Practice Question 2:

Make a simple calculator program which asks the user what operation they would like to perform (addition/subtraction/multiplication/division) or if they would like to exit the program. It must then ask for two numbers, perform the operation, display the result and then return to the same choices in the beginning, asking whether to perform any operation or exit the program.

Hint: First, think of an appropriate algorithm for achieving this. Use loops, conditional statements and functions.

Refer to page 110 for the solution.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Templates

Templates are statements or sets of statements which are written with an undefined data type, acting as templates of any given data type. In the previous section on function overloading, we looked at cases where we may have different functions of the same name. However, there might be cases where we may have a fixed set of code for a specific task, but the data type of the function may vary. For instance, in our example on function overloading, we had two functions of different data types, namely `int` and `double`, which did the exact same thing – multiplying two given values. What if we wanted to use more data types, depending on the user's input? What if we created a single function with the instruction of multiplying the given values, and simply defined the data type, as and when required? This can be achieved using function templates. There are similar templates even for classes, which we shall discuss subsequently. To understand a function template, look at the following function template based on the `Multiply()` function in the previous section:

```
template <typename A>
A Multiply(A x, A y)
{
    return (x * y);
}
```

Here, we use an example type 'A', named for our own convenience. But what A represents is not defined till we call the function elsewhere. Then, this creates a function of the specific data type. This is called 'function template instantiation'. When you call the same function again with arguments of the same datatype, the function which has already been instantiated is used. If a different one is used, however, a new function of the new datatype is created. Generally, a single capital letter is preferred for the name of the template. Note that since identifiers and keywords are always case-sensitive, the identifier A does not come in conflict with the variable 'a'.

Now, in the `main()` function, change the lines to set the values of c and d to the following:

```
c = Multiply<double>(a, b);
d = Multiply<int>(a, b);
```

Here, we first instantiate a function of datatype `double` and then instantiate a new function using the same template, using datatype `int`. Note that this will provide errors if you try to use string values (a concept explained in Chapter 4). You must ensure not to call a function template using an incompatible data type.

Templates can also be made for variables, using the following syntax:

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
template <typename T>
T var1 = T(1);
```

This creates a variable called 'var1', of data type 'T'. This can be better understood using the following example, which makes a variable for the constant Pi:

```
1   #include <iostream>
2   #include <limits>
3   using namespace std;
4   
5   template <typename A> constexpr A pi = A(3.1415926535897932385L);
    // 'L' is a generic suffix for 'long', which is determined by the
    type of variable
6   
7   int main()
8   {
9       cout << "Five-digit value of Pi: " << pi<float> << '\n';
10      cout << "Integer value of Pi: " << pi<int> << '\n';
11  
12      cin.clear();
13      cin.ignore(numeric_limits<streamsize>::max(), '\n');
14      cin.get();
15      return 0;
16  }
```

This program provides the following output:

```
Five-digit value of Pi: 3.14159
Integer value of Pi: 3
```

Here, it creates a `float` variable first and then an `int` variable, as `cout` is executed.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Chapter 4: Strings, Compound Data Types and a Few Other Concepts

Strings

Strings are a special type of variables which store a set of characters at once, in the form of text. While they work as variables, they are actually a class (which shall be explained in the subsequent chapters) and not a data type. Similar to literals, any text you use in a program is called a 'string literal' (such as "Hello, world!" in our first program). To use strings in your program, you need to include the header 'string', by using `#include <string>` and making an object (with the class `string`), by using `std::string`.

Look at the following program:

```
1  #include <iostream>
2  #include <string>
3  #include <limits>
4  using namespace std;
5
6  string name {};
7  string age {};
8
9  int main()
10 {
11     cout << "Enter your full name: ";
12     cin >> name;
13     cout << "Enter your age: ";
14     cin >> age;
15     cout << "Your name is " << name << " and your age is " << age
   << '\n';
16     cin.clear();
17     cin.ignore(numeric_limits<streamsize>::max(), '\n');
18     cin.get();
19     return 0;
20 }
```

Here, we obtain the user's name and age as strings, and then display them. But the output may surprise you.

```
Enter your full name: Ramesh C.B.
Enter your age: Your name is Ramesh and your age is C.B.
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

A disadvantage of `char` variables is that they store only a single character and if the user provides multiple characters (such as 'ab', '12', '125', 'abcd', etc.) instead of only one, the program keeps them in a queue and uses them immediately as inputs for the subsequent `cin` statements. Similarly, strings consider an input only till the first whitespace and the rest of the words are considered a separate value entirely. For this reason, this program simply considers the first name and uses the user's last name as an input for their age, as it does not consider spaces as a part of the value entered.

To solve this problem, simply use the following line instead of `cin`:

```
std::getline(std::cin >> std::ws, name);
```

```
1   #include <iostream>
2   #include <string>
3   #include <limits>
4   using namespace std;
5
6   string name {};
7   string age {};
8
9   int main()
10  {
11      cout << "Enter your full name: ";
12      getline(cin >> ws, name);
13      cout << "Enter your age: ";
14      getline(cin >> ws, age);
15      cout << "Your name is " << name << " and your age is " << age
    << '\n';
16      cin.clear();
17      cin.ignore(numeric_limits<streamsize>::max(), '\n');
18      cin.get();
19      return 0;
20  }
```

Now, the program will work as expected.

```
Enter your full name: Praajna Pattada
Enter your age: 21 years
Your name is Praajna Pattada and your age is 21 years
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

The `std::getline()` function ensures that it takes up an entire line of characters as a single response and a single input to a string object. `std::ws` is an input manipulator, which ensures that all whitespaces in the beginning of the user's response are not considered.

Practice Question 3:

Go back to practice question 2 and verify your program with its solution, if you haven't done so already. Run the program and you can notice that the same issue occurs if you enter multiple numbers in the main menu, instead of anything between 1 to 5. For example, if you type 12 and press enter, it considers the user to have chosen addition (1) and also considers the first number to be 2. It only asks the user to enter the second number.

```
What would you like to do? Enter the respective number and press Enter.

1: Addition
2: Subtraction
3: Multiplication
4: Division
5: Quit this program

12

Enter the first number: Enter the second number: 5

The sum of 2 and 5 is
7
```

Can you solve this issue to ensure that if anything other than 1 to 5 is entered in the beginning, the program simply considers it an invalid selection?

Hint: There are two possible solutions.

- While setting the value of a `char` variable from a string, use '`[0]`' to get the first character of that string.
- To convert a number to a string literal, use the function `std::to_string()`. For example, to convert the number 1 to a string, use `std::to_string(1)`.

Refer to page 113 for the solution.

To know the number of characters in a string, use the function `length()`, which is a part of class `std::string`. Go back to the previous program which asks the user's name and age. Add the following line just after line 15:

```
cout << "Your name is " << name.length() << " letters long." << '\n';
```

Compile and run the program to see this function in action.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
Enter your full name: Praajna
Enter your age: 21 years
Your name is Praajna and your age is 21 years
Your name is 7 letters long.
```

In C++17, a new feature has been introduced to make it easy to display the contents of a string. While displaying the contents of a string, like we did in our program using `cout`, the contents of the string are copied twice into the computer's memory – once to initialize the string's contents and then to display it on the screen. C++17 has a new class called '`std::string_view`'. To use this, you need to use the statement '`#include <string_view>`'. You can initialize a string_view in the same manner as a string, but it only provides read-only access to its contents.

```
std::string text1 { "Hi." };
std::string_view text2 { text1 };
std::string_view text3 { "Hi." }; // These three lines are correct ways of initialization

text1 = text2; // Will not work, as you cannot directly convert an object of class std::string_view to std::string
text1 = static_cast<std::string>(text2); // Will work, as this function casts a string_view to a string
```

To keep the value of a string_view constant, you can also use the keyword '`constexpr`' before it. For procedures requiring read-only access to any text, `string_view` is always preferable.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Type Conversion

Look at some of our previous programs carefully. The text displayed on the screen are characters and a string is a set of characters, as we learnt. However, at times, we directly used `cout` to display the values of numerical variables, such as `int`, `float` and `double`. How did the numbers get converted to text? C++ programs are designed to automatically perform **type conversion**, where a data type of a value in a variable is changed as required. Type conversion does not change the value of the original variable itself, but copies it wherever required and in the process, converts it to the required new type.

In the previous section, in the hint to practice question 3, the following function was mentioned:

```
std::to_string();
```

This is a perfect example of type conversion, which converts the values of variables like `int`, `float`, etc., to a `string`, which is treated as text instead of numbers. In the following lines of code, we create an integer variable and then set the value of a string to the integer.

```
int a { 1 };
std::string text_a {};
text_a = std::to_string(a);
```

Here, the number 1 gets converted to a string, which is interpreted as text which reads '1'. For this reason, in the solution of practice question 3, we mentioned that it is necessary to use quotation marks when the value is read from a `string` or `char`.

Now, in the initial example we mentioned on the top of this page, we explained how C++ automatically converts the numbers in certain variables directly to a string of characters and displays them on the screen. This is an example of implicit type conversion. There is no need to use the `std::to_string()` function in cases like these. In certain cases, explicit type conversion becomes necessary. In one approach of practice question 3, you can notice that the compiler would produce errors if you did not use the `to_string()` function and tried to convert the integer values to a string, in the blocks of `if-else if` statements, because type conversion is necessary.

In many cases, an operator called '`static_cast`' will be useful. It is written in this manner:

```
static_cast<>()
```

First, the new data type required must be specified within the angle brackets (< and >). Then, the value to be converted or the variable which already contains the value must be specified in the parentheses. Look at the following example:

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
1  #include <iostream>
2  #include <string>
3  #include <limits>
4  
5  int main()
6  {
7      float a { 1.75 };
8      int b { };
9      b = static_cast<int>(a);
10     std::cout << "Variable b has the value " << b << '\n'; // Displays the number '1', as int cannot store decimal numbers
11     char c { 49 }; // 49 is the ASCII code for the number '1'
12     std::cout << c << '\n'; // Displays the number '1'
13     std::cout << static_cast<int>(c) << '\n'; // Converts the value of char c, i.e., 49, to an int, and so, this displays the number '49' on the screen
14 
15     std::cin.clear();
16     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
17     std::cin.get();
18     return 0;
19 }
```

The output is as expected:

```
Variable b has the value 1
1
49
```

Note that we generally do not need to use static_cast for converting decimal values to integers, as seen in our previous examples, where the integer variables simply ignore the decimals and the digits after them. However, certain compilers provide warnings about the loss of data which occurs and may not compile the code, if you have enabled the option of treating warnings like errors.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Macros

Macros are custom substitutes for certain words, as defined by the user. They are checked by the pre-processor. Let us look at the following example. A, Hello, DISPLAY, CLEAR and RETURN are examples of macros.

```
1  #define A 50.03F
2  #define Hello "Hello, world!"
3  #define DISPLAY(x, y) (std::cout << "The two values provided are
   " << x << " and " << y << '\n')
4  #define CLEAR (std::cin.clear())
5  #define RETURN return 0;
6  
7  #include <iostream>
8  #include <limits>
9  
10 int main()
11 {
12     std::cout << Hello << '\n';
13     std::cout << "The value of variable A is " << A << '\n';
14     DISPLAY(4, 5);
15     CLEAR;
16     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
17     std::cin.get();
18     RETURN
19 }
```

This program provides the following output:

```
Hello, world!
The value of variable A is 50.03
The two values provided are 4 and 5
```

Similar to variables, macros are custom substitutes for certain statements, including constant values. They are defined similar to header guards, using the keyword #define. When you define a macro, the pre-processor replaces all its occurrences throughout the program with its definition. You can also notice that the macros CLEAR and RETURN are used to replace the respective statements themselves, with RETURN also including the semicolon at the end. Macros can also use parameters, as shown in the macro DISPLAY, which has the parameters x and y.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

You can also use the same macros in different source files, by defining them in a single header file and then including it in the required source files. For the previous program, you can make the following header:

Macros.h

```
1  #define A 50.03F
2  #define Hello "Hello, world!"
3  #define DISPLAY(x, y) (std::cout << "The two values provided are
   " << x << " and " << y << '\n')
4  #define CLEAR (std::cin.clear())
5  #define RETURN return 0;
```

Then simply use the following line in your main source file:

```
#include "Macros.h"
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Inline Functions

Previously, we discussed about various types of functions. However, there are cases where using a function may slow down your program, especially if you call it quite often. A function occupies a specific stack of memory and when called, the return address of the caller is stored in memory, which is used for the program to be diverted back to the main code, after executing the code in the function's definition. To prevent this, especially for smaller functions which are called frequently, we can use inline functions. Simply add the keyword `inline` while declaring the function.

```
inline int Function1(int a, int b)
{
    return (a + b);
}
```

Inline functions are ideal for functions with small amounts of code, which are defined in a single source file. They cannot be declared elsewhere and should not contain a large number of statements. If they are larger, certain compilers simply ignore the word `inline` and provide warnings. Each time an inline function is called, the function call is simply replaced with the statements of the function.

Inline functions work with a mechanism similar to that of macros. They work as an alternative to macros with parameters. However, instead of all references to them being replaced with specific code, an inline function acts like an ordinary function which does not interrupt the normal flow of the program.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Introduction to Compound Data Types

So far, you should have a good idea of using the various types of datatypes. However, let us suppose we have a scenario where we want to store multiple pieces of information and organize sets of it to store large amounts of data. If we take the example of modern video games, especially if you have used advanced game engines and tried to develop one, you would observe that the various functions in a game are performed by different 'objects', as explained previously, with regard to object-oriented programming. Several attributes, such as location, physics, etc., are common to almost all objects, such as the player's character, enemies and other objects in the scene.

In such scenarios, we would prefer to have a 'super-variable' which contains a set of multiple data types and can help simplify our task of easily making various other instances of data types with the same set of multiple attributes at once. Previously, we defined an 'object' as a location in the computer's memory where information is stored. However, in object-oriented programming, an object is precisely defined as a self-contained unit which performs specific functions. In other words, instead of having our program focus on performing the tasks in the `main()` function, one by one, we can split the code into different sets of compound data types and functions, which act simultaneously and independently, while being interlinked to one another. This is achieved using OOP. This is exactly how several modern applications work, including video games. In a video game, you can have an object managing the player's character and have other objects to control enemies and obstacles, simultaneously. Instead of a single task being done at a time, this enables one to make complex applications which perform multiple tasks simultaneously.

Primarily, this is achieved using compound data types, which are a fundamental requirement and the unit of OOP, enabling one to create 'objects'. In C++, the following are the compound data types:

- **Functions**
- **C-style Arrays**
- **Pointer types**
    - Pointer to object
    - Pointer to function
- **Pointer to member types**
    - Pointer to data member
    - Pointer to member function
- **Reference types**
    - L-value references

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

    - R-value references
- **Enumerated types**
    - Unscoped enumerations
    - Scoped enumerations
- **Class types**
    - Structs
    - Classes
    - Unions

We shall look at these in the upcoming sections.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

References

A **reference** is a variable which stores information about another variable and its location in the computer memory, but does not store its value nor is an independent variable. In other words, think of a variable like a sheet of paper with some information written on it. A reference is another object with information about where to locate that paper. It does not say much about the same information in that paper, but only helps us know where it is.

A reference is defined using the ampersand character (&). For example, `int&` and `double&` respectively indicate an l-value reference to an integer and a double.

```
int a {}; // An integer variable 'a'
int b { 5 };
int& lref { a }; // An L-value reference to 'a'

cout << a << '\n'; // Displays the value of 'a' (i.e., 0)
cout << &a << '\n'; // Displays the memory address of 'a'

a = 1; // Changes the value of a to 1
lref = 4; // Also changes the value of a to 4, by finding the
variable referenced here
lref = b; // Changes the value of a to 5, by changing the value
of the referenced variable (a) to the value of b
```

When a reference is initialized, it is said to be bound to the concerned object, function, etc. It cannot be changed after initialization. For this reason, the last line simply changes the value of a to 5, but does not bind `lref` to the variable b.

The data type of a reference must also match the data type of the entity it is bound to. If a reference is being bound to a constant variable, it must also be initialized using the keyword `const`. An L-value reference cannot be bound to an R-value. In C++, an expression which refers to a memory address of an object is known as an 'L-value'. All other values are known as 'R-values'.

References are useful when we pass large amounts of data, but do not want to make copies of it. Consider many of our previous examples where we made functions with `int`, `float` or `double` variables as their arguments. While calling those functions, we would make copies of the values we would use as their arguments. However, if we use complex entities, such as strings, which are a class, it would use more memory. A better approach would be to use a reference instead. Consider the following program:

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
1  #include <iostream>
2  #include <string>
3  #include <limits>
4  
5  void displayText(std::string& text)
6  {
7      std::cout << "The text you have entered is: " << text <<
   '\n';
8      std::cout << "The memory address of the string is: " << &text
   << '\n';
9  }
10 
11 int main()
12 {
13     std::string Text {};
14     std::cout << "Enter a line of text: ";
15     std::cin >> Text;
16     std::getline(std::cin >> std::ws, Text);
17 
18     std::cin.clear();
19     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
20     std::cin.get();
21     return 0;
22 }
```

This gives the following output:

```
Enter a line of text: Hello, world!
The text you have entered is: Hello, world!
The memory address of the string is: 0x155d9ffae0
```

Here, we do not make new temporary copies of the line "Hello, world!," while transferring it across functions. Rather, the function `displayText()` simply receives a reference to the string `Text`, from which its value is obtained and displayed on the screen.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Pointers

Pointers are expressions which represent both the address and type of an object. Pointers are quite similar to L-value references. They use the dereference operator (*) instead of the address-of operator (&). The dereference operator indicates the value stored at the referenced memory address. It is used to access the object referenced by a pointer. Pointers perform a function highly similar to L-value references. The following lines are two alternate methods of making a pointer called 'p':

```
int* p; // OR
int *p;
```

After creating a pointer, you must reference it to a memory address. If we suppose we have an int variable 'a', we can use the following to make 'p' a pointer of 'a':

```
p = &a;
```

Now, let us modify our previous program to use a pointer instead. It shall be as follows:

```
1   #include <iostream>
2   #include <string>
3   #include <limits>
4
5   void displayText(std::string* text)
6   {
7       std::cout << "The text you have entered is: " << *(text) <<
    '\n'; // Get the value stored at the memory address of the given
    pointer (i.e., p)
8       std::cout << "The memory address of the string is: " << text
    << '\n'; // Display the memory address of the given pointer
9   }
10
11  int main()
12  {
13      std::string Text {};
14      std::string* p { &Text };
15      std::cout << "Enter a line of text: ";
16      std::getline(std::cin >> std::ws, Text);
17      displayText(p);
18
19      std::cin.clear();
20      std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
    '\n');
21      std::cin.get();
22      return 0;
    }
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Its output is also similar:

```
Enter a line of text: Howdy, pal!
The text you have entered is: Howdy, pal!
The memory address of the string is: 0x96793ff820
```

As shown here, both pointers and references can also be used as parameters in a function.

Unlike references, a pointer can be assigned to point to a different object, by using the set operator (=). A pointer can have a value called 'null' (by setting its value to 'nullptr'). This means that the pointer is set to not point to any object at all, and is blank. You can try this by modifying the declaration of 'p' in the previous program, on line 14, by replacing it with these two lines:

```
std::string* p { nullptr }; // Creates a new pointer 'p', which
points to no object
p = &Text; // Sets the pointer to point to the string 'Text'
```

If a pointer is left blank during initialization, its value is set to nullptr.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Arrays

Arrays are sets of multiple values of a single data type, stored in consecutive memory addresses. They are like lists of different values. They are useful when we want to store multiple values of data and use lesser code to access them whenever required. In C, strings are actually arrays of characters. The syntax of an array is as follows:

```
datatype name[size];
```

For example,

```
float Array1[10]; // A float array with ten different values
```

To access a specific value within an array, one has to reference its position/serial number, called the 'index', in the array, using the symbols `[` and `]`. For example, '`Array1[0]`' or '`Array1[9]`' respectively refer to the first and last values of `Array1`. The index always starts from zero and ends with a number less than the total number of values. You can manually initialize an array by specifying all the values for their entities:

```
int IntegerArray[4] = { 1, 4, 0, 8 };
```

Note that an array must be initialized with a fixed length (also called 'size'), to determine how many values it can hold and how much memory should be allocated to it. For example, `IntegerArray` here with a capacity of 4 values, occupies about 8 (4 x 2) bytes (refer to the table on page 29). You can also not specify its size during initialization, if you already specify an initial value. For example, the following code is also valid and automatically sets the size to 4 values.

```
int IntegerArray[] = { 1, 4, 0, 8 };
```

Similarly, you can use C-style strings, which are nothing but arrays of the data type `char`, such as the following:

```
char Name[] = { "Sanjeev" };
```

In the computer memory, it is stored in a manner like this:

```
char Name[8] = { 'S', 'a', 'n', 'j', 'e', 'e', 'v', '\0' };
```

Here, the size is automatically set to 8 bytes (including the terminating null character). You can keep a higher size to allow more characters to be stored.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

std::vector

While arrays are useful, they have a major catch – one cannot change their size as they like, after they have been created. They must already be allocated a fixed amount of memory during runtime. This problem is solved by the class `std::vector`. It works similar to arrays and is a class of the namespace `std`. Suppose we want to find out the average of ten numbers. We can make an array or a vector out of them and use smaller lines of code for any tasks requiring all of them or even a specific value from the array. Now, look at the following program:

```
1   #include <iostream>
2   #include <vector>
3   #include <limits>
4
5   int main()
6   {
7       std::vector<double> Data1 { 1, 50.3, 103.4, 17.5, 28, 0 };
8       double Total { static_cast<double>(Data1.size()) }; // We
    initially get the value as a typedef or std::size_t, which is a
    class whose type is defined as per the usage of its values
9       double AverageValue { (Data1[0] + Data1[1] + Data1[2] +
    Data1[3] + Data1[4] + Data1[5]) / Total };
10      std::cout << "The given values are: " << Data1[0] << ", " <<
    Data1[1] << ", " << Data1[2] << ", " << Data1[3] << ", " <<
    Data1[4] << " and " << Data1[5] << "." << '\n';
11      std::cout << "Their average is: " << AverageValue << '\n';
12
13      std::cin.clear();
14      std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
    '\n');
15      std::cin.get();
16      return 0;
17  }
```

This program gives the following output:

```
The given values are: 1, 50.3, 103.4, 17.5, 28 and 0.
Their average is: 33.3667
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Chapter 5: Object-Oriented Programming (OOP)

Introduction

As we discussed on page 70, compound data types form the building blocks of object-oriented programming. The primary reason for the growth of the popularity of C++ to be used to create most complex applications is due to the feature of OOP. Through object-oriented programming, one can create multiple objects with similar or different features. Instead of only one set of code being executed at a time, objects work as different instances of a particular set of code, which run simultaneously and can perform different tasks. The objects can still be interlinked with each other and access each other's data. This is perhaps, one of the greatest advantages of C++, which has enabled the existence of most modern programs. Till now, the programs written use procedural programming, where the program follows a fixed procedure and does not use objects, which act as independent units.

In the upcoming sections of this book, we shall begin to explore concepts of OOP elaborately.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Classes

Classes are user-defined data types which contain their own custom members, such as variables or functions. These are called 'data members'. A class can be said to be the building block of an object and works like a broad category for objects. An object is derived from a class and works as its class is written. For example, if we consider a laptop and a desktop as two objects, their class would be 'computer'. A class contains the definitions of an object and its various attributes.

Usually, a class is written using a custom header file and a source file, with the same name and different extensions (.h and .cpp). A class also has members which can be `private` or `public`. Any members which belong to the former category are accessible only to other members of the same class. The members of the latter category can be accessed by any objects. This is how an object derived from a class works as an independent unit within a program. Typically, it is recommended to use a capital letter for the first letter of the name of a class, and use lower-case letters for the names of any members within a class.

Let us look at how a class works, using an example. Create the following files and add the code given here. You can use any specifications for the theoretical computers in question here.

`Computer.h`

```
1  #ifndef _COMPUTER_
2  #define _COMPUTER_
3
4  #include <iostream>
5  #include <string>
6
7  class Computer
8  {
9      private:
10         std::string serialno { };
11         std::string product_id { };
12         bool isPortable;
13
14     public:
15         std::string name {};
16         std::string CPU {};
17         int RAM {};
18         std::string RAM_units {};
19         std::string Manufacturer {};
20         std::string OS {};
21         std::string deviceType {};
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
22          void show_information();
23  };
24
25  #endif
```

Computer.cpp

```
1   #include "Computer.h"
2
3   void Computer::show_information()
4   {
5       std::cout << "Details of computer named \'" << name << "\':"
    << '\n' << '\n';
6       std::cout << "This computer runs " << OS << " as its
    operating system." << '\n';
7       std::cout << "This computer's processor is " << CPU << "." <<
    '\n';
8       std::cout << "This computer has " << RAM << " " << RAM_units
    << " of Random Access Memory." << '\n';
9
10      if (deviceType == "Laptop")
11          isPortable = true;
12      else
13          isPortable = false;
14
15      if (isPortable)
16          std::cout << "This computer is a laptop and is portable."
    << '\n';
17      else
18          std::cout << "This computer is a desktop. It is not
    portable." << '\n';
19
20      std::cout << "______________________" << '\n' << '\n';
21  }
```

Main.cpp

```
1   #include "Computer.h"
2   #include <limits>
3
4   int main()
5   {
6       Computer Laptop, Desktop; // Creates two objects of the type
    'Computer'
7
8       Laptop.name = "Laptop";
9       Laptop.CPU = "Intel Core i7";
10      Laptop.OS = "Windows 11";
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
11      Laptop.RAM = 16.0f; // 'f' is a suffix used in certain C++
   programs to indicate a floating-point value, similar to 'l' for
   long
12      Laptop.RAM_units = "Gigabytes";
13      Laptop.deviceType = "Laptop";
14      Laptop.show_information(); // Calls the function from our
   'Laptop' object
15
16      Desktop.name = "Desktop";
17      Desktop.CPU = "Intel Core i5";
18      Desktop.OS = "Windows 10";
19      Desktop.RAM = 8.0f;
20      Desktop.RAM_units = "Gigabytes";
21      Desktop.deviceType = "Desktop";
22      Desktop.show_information();
23
24      std::cin.clear();
25      std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
26      std::cin.get();
27      return 0;
28  }
```

Compile both source files into a single program. When you run the program, you will obtain the following output:

```
Details of computer named 'Laptop':

This computer runs Windows 11 as its operating system.
This computer's processor is Intel Core i7.
This computer has 16 Gigabytes of Random Access Memory.
This computer is a laptop and is portable.
----------------------

Details of computer named 'Desktop':

This computer runs Windows 10 as its operating system.
This computer's processor is Intel Core i5.
This computer has 8 Gigabytes of Random Access Memory.
This computer is a desktop. It is not portable.
----------------------
```

Here, note that `Main.cpp` contains the actual usage of the class `Computer`, as it creates two objects of that class and uses them accordingly. Alternatively, you can also skip making a separate header file for your class and simply include the code directly in the `Computer.cpp` file. You can even use a single `.cpp` source file itself instead of multiple, by

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

simply keeping the `main()` function also in the same file. However, it is more convenient to use a separate source file (and header file) for a class, as it can be implemented in multiple other source files as and when required, creating distinct objects.

Did you also notice that we accessed and modified the `bool` variable `isPortable` only from `Computer.cpp`, but not `Main.cpp`? This is because they are private members and cannot be accessed by other objects or functions outside their own class.

The dot operator (`.`) is called the 'class member access operator', as it enables you to access the members of an object of a certain class. The assignment operator (=) also works for objects of the same class. For example, if we want all the values of the members of our object `Laptop` to be applied to those of the object `Desktop`, we would have to use the following code:

```
Desktop = Laptop;
```

If you create a pointer to a class, the class member access operator would be '`->`', instead of dot. We could use the following in our previous program:

```
Computer* ComputerPtr { &laptop };

// You can now use this..
(*ComputerPtr).show_information();
// ..or this
ComputerPtr->show_information();
```

Remember that as said previously, you can use custom namespaces to declare and define a class or any other compound data type.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Structures and Unions

Structures and Unions are data types which are almost identical to classes, with some variations in their features. Structures and unions are available in C as well. They are respectively declared and defined using the keywords `struct` and `union`. Consider the following program as an example, where we also use a single file for all code.

```
1  #include <iostream>
2  #include <string>
3  #include <limits>
4  
5  struct Date
6  {
7      int Day {};
8      std::string Month {};
9      int Year {};
10     void ShowDate()
11     {
12         std::cout << "This program\'s compilation date is " <<
   Day << " " << Month << " " << Year << "." << '\n';
13     }
14 };
15 
16 int main()
17 {
18     Date Today; // Creates a new object of the struct 'Date'
19 
20     // Set the date of making this program here
21     Today.Day = 8;
22     Today.Month = "May";
23     Today.Year = 2026;
24     Today.ShowDate();
25 
26     std::cin.clear();
27     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
28     std::cin.get();
29     return 0;
30 }
```

The program now displays the given date:

```
This program's compilation date is 8 May 2026.
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

The primary difference between a struct and a class is that all the members of a `struct` are public, while those of a `class` are private by default, except when the keyword `public` is used while declaring them. Structures are compatible with C, while classes are not so and the latter are the basic units of OOP.

A union is similar to these in function, with a similar syntax. The key difference is that all members of a union share the same memory location and the amount of memory to be used is determined by the largest member of the union. Look at the following example:

```
union UserAccount
{
    char Username[15] {};
    std::string Password {};
}
```

To create an instance of the above union, use the following code:

```
union UserAccount Account1;
strcpy(Account1.Username, "Ramesh");
strcpy(Account1.Password, "Temporary-password");
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Class Templates

Previously, we had discussed about function templates and variable templates. Similarly, it is possible to create templates for structures and classes. Consider the following example:

```
1   #include <iostream>
2   #include <string>
3   #include <cstring> // Required for the function strcpy()
4   #include <limits>
5   
6   template <typename A>
7   class TemplateClass
8   {
9       public:
10          A variable1 {};
11          A variable2 {};
12          char Name[15] {};
13          void DisplayValues();
14  };
15  
16  template <typename A>
17  void TemplateClass<A>::DisplayValues()
18  {
19      std::cout << "The variables of " << Name << " have the following values:" << '\n' << '\n';
20      std::cout << variable1 << '\n' << variable2 << '\n' << "_________________" << '\n';
21  }
22  
23  int main()
24  {
25      TemplateClass<float> Object1;
26      TemplateClass<int> Object2;
27  
28      strcpy(Object1.Name, "Object 1");
29      Object1.variable1 = 1.01f;
30      Object1.variable2 = 2.0f;
31      Object1.DisplayValues();
32  
33      strcpy(Object2.Name, "Object 2");
34      Object2.variable1 = 4;
35      Object2.DisplayValues();
36  
37      std::cin.clear();
38      std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
39      std::cin.get();
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
40      return 0;
41  }
```

Here, we create two objects, `Object1` and `Object2`. Their ambiguous data types are respectively initialized as `float` and `int`. We then set the values of their member variables to the numbers 1.01 and 2, in `Object1`, and 4 and 0, in `Object2`. So, the program provides the following output:

```
The variables of Object 1 have the following values:

1.01
2
-----------------
The variables of Object 2 have the following values:

4
0
-----------------
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Enumerations

Enumerations/Enumerated types are compound data types with a set of named constants. These constant values are known as 'enumerators'. An enumeration must be fully defined properly before it can be used in a program. Unscoped enumerations are declared using the keyword enum. Look at the following example:

```
enum Fruit
{
    mango, /* 0 */
    orange, /* 1 */
    apple, /* 2 */
    banana, /* 3 */
    grapes, /* 4 */
};

enum Switch {
    on, /* 0 */
    off, /* 1 */
};
```

In `int main()`, the following code can be used:

```
Fruit fruit1 { mango };
Fruit fruit2 { apple };

Switch switch1 { on };
Switch switch2 { off };
```

This uses two unscoped enumerations and two objects, `fruit1` and `fruit2`, are created using it as their data type. The name of an enumeration/enumerated type should start with a capital letter. While enumerations need not be named, leaving them unnamed should be strictly avoided.

In these enumerations, the numbers in the comments are used as indices for the enumerators. The following example explains this:

```
1  #include <iostream>
2  #include <string>
3  #include <limits>
4
5  enum Switch {
6      off, /* 0 */
7      on, /* 1 */
8  };
9
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
10 std::string_view getStatus(Switch exampleSwitch)
11 {
12     switch(exampleSwitch)
13     {
14         case off: return "off";
15         case on: return "on";
16         default: return "error";
17     }
18 }
19
20 int main()
21 {
22     std::cout << "Enter the state of a switch (0 for off, 1 for
   on): ";
23     int switchState {};
24     std::cin >> switchState;
25     if (switchState < 0 || switchState > 1)
26     {
27         std::cout << "Error! You have entered an invalid input."
   << '\n';
28     } else {
29         Switch mySwitch { static_cast<Switch>(switchState) };
30         std::cout << "The switch is " << getStatus(mySwitch) <<
   '\n';
31     }
32
33     std::cin.clear();
34     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
35     std::cin.get();
36     return 0;
37 }
```

The output works as it should:

```
Enter the state of a switch (0 for off, 1 for on): 1
The switch is on
```

Here, we created a function called `getStatus()`, which returns a value which is a `string_view` and uses an enumeration of type `Switch` as its parameter. Then, after receiving an input from the user, we use a `switch` statement display the value as a `string_view`.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Constructors and Destructors

Let us go back to our previous program on displaying the date, and change the structure to a class.

```
1  #include <iostream>
2  #include <string>
3  #include <limits>
4  
5  class Date
6  {
7      public:
8          int Day {};
9          std::string Month {};
10         int Year {};
11         void ShowDate()
12         {
13             std::cout << "This program\'s compilation date is " << Day << " " << Month << " " << Year << "." << '\n';
14         }
15 };
16 
17 int main()
18 {
19     // This is called 'Aggregate Initialization', which initializes the members of a struct or class in the order they are declared.
20     Date Today { 10, "May", 2026 };
21     Today.ShowDate();
22 
23     std::cin.clear();
24     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
25     std::cin.get();
26     return 0;
27 }
```

As stated previously, the members of a class are private by default and those of a structure are public by default. This program works, because `int main()` can access the members of the class `Date`, which are public. This class is called an 'aggregate class'. But what if the class members were private? In such a case, aggregate initialization would not work at all.

To solve this problem, we have a simple solution – using a **constructor**, which is a function which initializes all members of a non-aggregate class. The following example shows the usage of a constructor:

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```cpp
#include <iostream>
#include <string>
#include <limits>

class Date
{
    private:
        int Day {};
        std::string Month {};
        int Year {};
        void ShowDate()
        {
            std::cout << "This program\'s compilation date is " << Day << " " << Month << " " << Year << "." << '\n';
        }

    public:
        // This is a constructor, which takes up three arguments
        Date(int a, std::string m, int y)
        {
            Day = a;
            Month = m;
            Year = y;
            std::cout << "Constructor called." << '\n';
            ShowDate();
        }
};

int main()
{
    // This is called 'Aggregate Initialization', which initializes the members of a struct or class in the order they are declared.
    Date Today { 10, "May", 2026 };

    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::cin.get();
    return 0;
}
```

Once again, the program works as expected, even though all members are private, except the constructor.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
Constructor called.
This program's compilation date is 10 May 2026.
```

Constructors do not have any data type and must be simply named with the identifier of the same class.

Now, when a non-aggregate class is destroyed, a function called a 'destructor' is called. To create a destructor, simply make a function using the same identifier of the class, like a constructor, but use the tilde symbol (~) at the beginning. A class can only have a single constructor and a single destructor.

Look at the following program, where we use a constructor and a destructor:

```
1   #include <iostream>
2   #include <string>
3   #include <limits>
4   
5   class SimpleObj
6   {
7       private:
8           char ID {};
9           void ShowID()
10          {
11              std::cout << "Object ID: " << ID << '\n';
12          }
13  
14      public:
15          SimpleObj(char id)
16          {
17              ID = id;
18              std::cout << "Constructor called for object with ID "
    << ID << "." << '\n';
19              ShowID();
20          }
21  
22          ~SimpleObj()
23          {
24              std::cout << "Destructor called for object with ID "
    << ID << "." << '\n';
25          }
26  };
27  
28  int main()
29  {
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
30      SimpleObj Obj1 { 'A' };
31      {
            SimpleObj Obj2 { 'B' };
        }  // 'Obj2' goes out of scope here, so its destructor is
   called.
32
33      std::cin.clear();
34      std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
35      std::cin.get();
36      return 0;
37  }
```

The program will provide the following output:

```
Constructor called for object with ID A.
Object ID: A
Constructor called for object with ID B.
Object ID: B
Destructor called for object with ID B.
```

Note that the destructor was not called for `Obj1` with the ID A, since the main() function already leads the program to terminate before the destructor is called.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Operator Overloading

Operator overloading refers to making custom functions which are called using existing operators. Similar to function overloading, where we created different functions with the same identifier, we can use existing operators to perform specific actions for a specific class, in C++. Note that one cannot create custom operators. Operator overloading also works only in relation to a specific class. To create an overloaded operator, one needs to create a `friend` function. These are not members of a class, but work in relation to it.

```
1  #include <iostream>
2  #include <string>
3  #include <limits>
4  
5  class BankAccount
6  {
7      private:
8          int balance;
9  
10     public:
11         // Constructor for the class
12         BankAccount(int initialBalance) : balance { initialBalance } {}
13 
14         std::string accountName {};
15         friend BankAccount operator+(const BankAccount& acc1, const BankAccount& acc2); // Declaring the operator+ function as a friend of the BankAccount class allows it to access the private members
16         void displayBalance();
17 };
18 
19 void BankAccount::displayBalance()
20 {
21     std::cout << "The current balance of " << accountName << " is " << balance << "." << '\n';
22 }
23 
24 // This is a function to overload the '+' operator for the BankAccount class. It takes two BankAccount objects as parameters and returns a new BankAccount object as the result of the addition.
25 BankAccount operator+(const BankAccount& acc1, const BankAccount& acc2)
26 {
27     // Add the given balances of the two BankAccount objects and return a new BankAccount object with the resulting balance.
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
28      return (acc1.balance + acc2.balance);
29  }
30  
31  int main()
32  {
33      BankAccount Account1 { 8000 }; // Initializing Account1 with
    a balance of 10000
34      Account1.accountName = "Account 1";
35      Account1.displayBalance();
36      BankAccount Account2 { 2000 }; // Initializing Account2 with
    a balance of 2000
37      Account2.accountName = "Account 2";
38      Account2.displayBalance();
39  
40      // Using the overloaded '+' operator to add the balances of
    Account1 and Account2, we create a new BankAccount object called
    'NewAccount'
41      BankAccount NewAccount { Account1 + Account2 };
42      NewAccount.accountName = "New Account";
43      NewAccount.displayBalance();
44  
45      std::cin.clear();
46      std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
    '\n');
47      std::cin.get();
48      return 0;
49  }
```

Compile the program and you can observe how well operator overloading works here:

```
The current balance of Account 1 is 8000.
The current balance of Account 2 is 2000.
The current balance of New Account is 10000.
```

Note that the following operators can be overloaded in C++, using similar mechanisms:

```
+ - * / % ++ -- == != < <= > >= && || ! = op= & | ^ ~ << >> () [] & *
-> , new delete
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Chapter 6: A Few Miscellaneous Topics

File Input and Output

The headers `fstream`, `ifstream` and `ofstream` can be used to read data from and write data to files. The following program demonstrates the same:

```
1   #include <iostream>
2   #include <fstream>
3   #include <string>
4   #include <limits>
5
6   int ReadFromFile(std::string fileName)
7   {
8       std::ifstream inf{ fileName };
9       if (!inf)
10      {
11          // Display an error and exit
12          std::cerr << "The file " << fileName << " cannot be opened for viewing.\n";
13          return 0; // We use a custom return value to indicate success or failure of the function. In this case, 1 indicates success and 0 indicates failure.
14      }
15
16      std::string Input{};
17      // Executes as long as there are lines to read in the file
18      while (std::getline(inf, Input)) // You can use 'inf >> Input' as the condition, to read word by word, but using 'std::getline(inf, Input)' allows you to read the file line by line, which is often more useful for text files.
19      {
20          std::cout << Input << '\n';
21      }
22
23      return 1;
24  }
25
26  int WriteToFile(std::string fileName)
27  {
28      std::ofstream outf{ fileName };
29
30      // If the output file stream cannot be opened for writing
31      if (!outf)
32      {
33          // Display an error and exit
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
34          std::cerr << "The file " << fileName << " cannot be
    opened for writing.\n";
35          return 0;
36      }
37  
38      // Write some text to the file
39      outf << "Hello, world!\n";
40      outf << "This file has been written using a C++ program.\n";
41      outf << "If you can see this, it means that the program has
    successfully written data to this file.\n";
42      return 1;
43  }
44  
45  int main()
46  {
47      std::string Filename {};
48      std::string UserInput {};
49      char selection {};
50      int status {};
51  
52      do {
53          std::cout << "What would you like to do? Enter a number
    to select the operation:" << '\n' << '\n';
54          std::cout << "1. Read from file" << '\n';
55          std::cout << "2. Write to file" << '\n';
56          std::cout << "3. Exit" << '\n' << '\n';
57          std::getline(std::cin >> std::ws, UserInput);
58          selection = UserInput[0];
59  
60          if (selection == '1')
61          {
62              std::cout << "Enter the filename: ";
63              std::getline(std::cin >> std::ws, Filename);
64              status = ReadFromFile(Filename);
65  
66              if (status == 1)
67              {
68                  std::cout << '\n' << '\n' << "File read
    successfully." << '\n';
69              }
70  
71              std::cout << "__________" << '\n' << '\n';
72          } else if (selection == '2')
73          {
74              std::cout << "Enter the filename: ";
75              std::getline(std::cin >> std::ws, Filename);
76              status = WriteToFile(Filename);
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

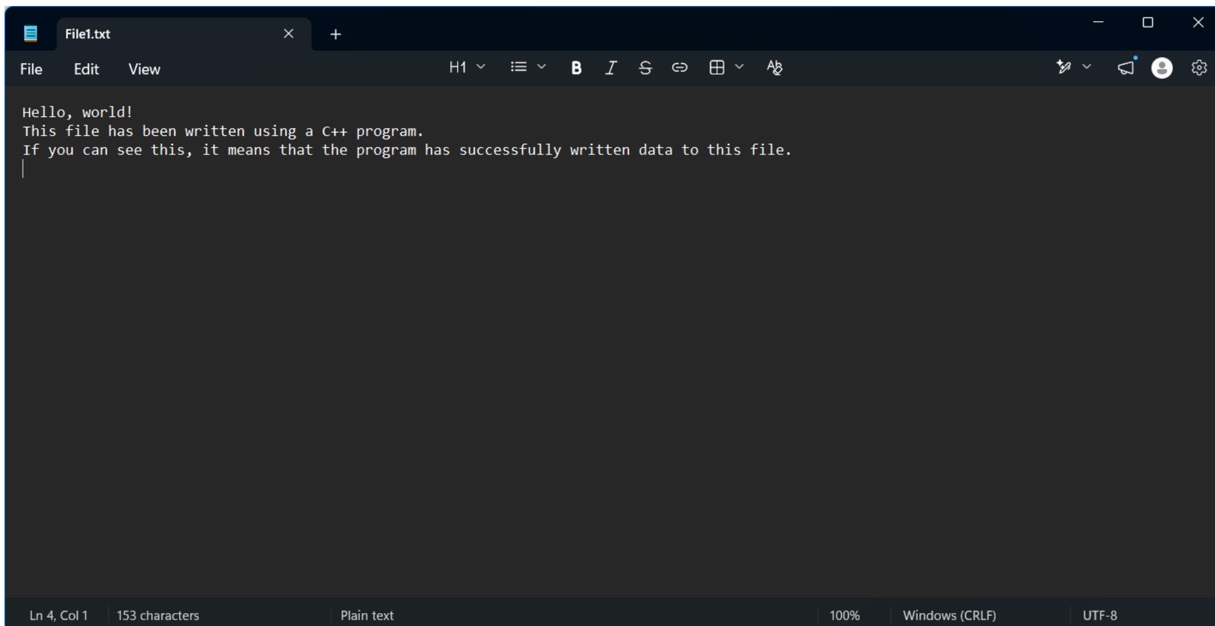
```
77
78                  if (status == 1)
79                  {
80                      std::cout << "File written successfully." <<
    '\n';
81                  }
82
83                  std::cout << "__________" << '\n' << '\n';
84              } else if (selection == '3')
85              {
86                  std::cout << "Exiting program..." << '\n';
87                  std::cout << "__________" << '\n';
88                  continue;
89              } else
90              {
91                  std::cout << "Invalid selection. Please try again."
    << '\n';
92                  std::cout << "__________" << '\n' << '\n';
93              }
94          } while (selection != '3');
95
96          std::cin.clear();
97          std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
    '\n');
98          std::cin.get();
99          return 0;
100     }
```

When you run it, you can use the options to create a new file.

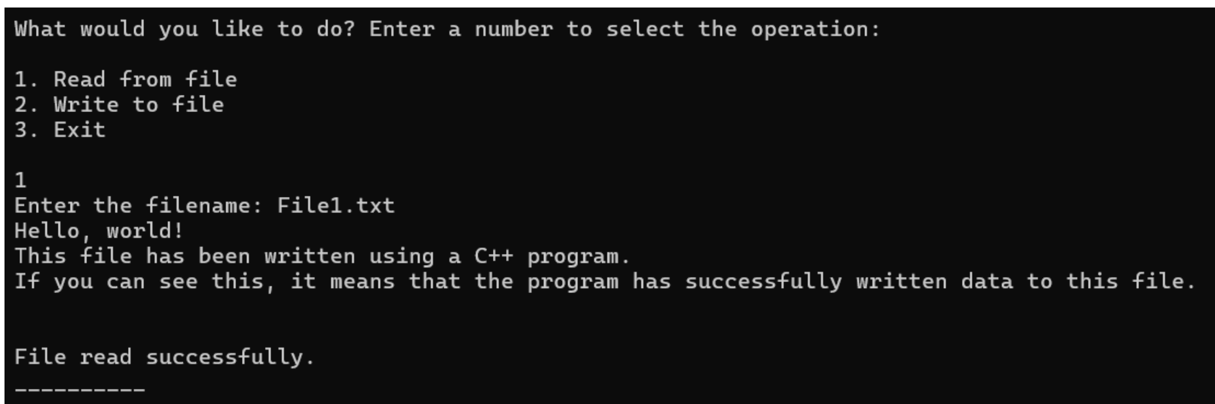
```
What would you like to do? Enter a number to select the operation:

1. Read from file
2. Write to file
3. Exit

2
Enter the filename: File1.txt
File written successfully.
----------
```

A file of the same name will now be created in the same directory as your program.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)



Now, if you use the options to read data, you will get the following:



© Copyrights 2015-26 Nilapatri (Praajna Pattada)

C-Time

Previously, we had made a program to display the date and time, as we set. However, what if we wish to display the current date and time, by checking the computer's clock? C-Time is a C++ header which enables us to achieve this. Look at the following program:

```
1   #include <iostream>
2   #include <ctime>
3   #include <limits>
4
5   int main()
6   {
7       struct tm* Time;
8       time_t CurrentTime;
9       time(&CurrentTime);
10      Time = localtime(&CurrentTime);
11
12      std::cout << "Today is " << Time->tm_mday << "/" << Time->tm_mon + 1 << "/" << Time->tm_year + 1900 << "." << '\n';
13      std::cout << "The time is " << Time->tm_hour << ":" << Time->tm_min << ":" << Time->tm_sec << '\n';
14
15      std::cin.clear();
16      std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
17      std::cin.get();
18      return 0;
19  }
```

This program will now display the time in a 24-hour format and use numbers for the date.

```
Today is 10/5/2026.
The time is 22:43:36
```

Note that the header `ctime` has the following variables:

```
int tm_sec; // 0 - 59(60)
int tm_min; // 0 - 59
int tm_hour; // 0 - 23
int tm_mday; // Day of month: 1 - 31
int tm_mon; // Month: 0 - 11 (0 refers to January)
int tm_year; // Years since 1900
int tm_wday; // Weekday: 0 - 6 (0 refers to Sunday)
int tm_yday; // Day of year: 0 – 365
int tm_isdst; // Flag for summer-time
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Practice Question 4: Using your knowledge, make a new program which displays the date and time in the following format, with a 12-hour format for the time:

```
Today is 10 May 2026 (Sunday)
The time is 10:59:22 PM
```

Refer to page 116 for the solution.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Random Values

Generating random values is quite useful for certain algorithms, especially in games. Computers do not have the ability to generate truly random numbers, as the functionality of any software involves predictable results. Generally, programs use pseudo-random number generation to emulate randomness and picking a value out of them. `cstdlib` has a set of functions which can be used to generate random numbers. It has two major functions – `srand()` and `rand()`. The syntaxes for them are as follows:

```
void srand(unsigned int seed); // Takes a variable for 'seeding', to start a sequence
int rand(); // Returns a random number as an integer
```

The following program takes up four random numbers and displays them on the screen:

```
1   #include <iostream>
2   #include <cstdlib>
3   #include <limits>
4
5   int main()
6   {
7       unsigned int seed {};
8       int a {}, b {}, c {}, d {};
9       std::cout << "Please enter any random number: ";
10      std::cin >> seed;
11      srand(seed);
12      a = rand();
13      b = rand();
14      c = rand();
15      d = rand();
16      std::cout << '\n' << "Here are four random numbers generated based on this:" << '\n' << a << '\n' << b << '\n' << c << '\n' << d << '\n';
17
18      std::cin.clear();
19      std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
20      std::cin.get();
21      return 0;
22  }
```

Run the program and observe the results.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
Please enter any random number: 1

Here are four random numbers generated based on this:
41
18467
6334
26500
```

Notice that you can actually predict the “randomness,” as the numbers remain the same for a particular seed value provided.

As an alternative, you can use a value from the date and time to seed the random sequence of numbers. For example, we could use the number of seconds from the current time, using `ctime`.

```
1  #include <iostream>
2  #include <cstdlib> // Can also use 'stdlib.h'
3  #include <ctime>   // Can also use 'time.h'
4  #include <limits>
5  
6  int main()
7  {
8      int a {}, b {}, c {}, d {};
9      unsigned int seed {};
10     struct tm* Time;
11     time_t CurrentTime;
12     time(&CurrentTime);
13     Time = localtime(&CurrentTime);
14     seed = Time->tm_sec; // Use seconds as seed
15 
16     srand(seed);
17     a = rand();
18     b = rand();
19     c = rand();
20     d = rand();
21 
22     std::cout << '\n' << "Here are four random numbers generated
   based on this:" << '\n' << a << '\n' << b << '\n' << c << '\n' <<
   d << '\n';
23 
24     std::cin.clear();
25     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
26     std::cin.get();
27     return 0;
28 }
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Alternatively, instead of lines 10 to 16 in the previous program, you can use the following code to obtain the time in seconds:

```
long long seconds;
time (&seconds);
srand ((unsigned)seconds);
```

To get a random number within a range of ten numbers (from 0 to 9), you can use the following code:

```
(rand() % 10)
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Final Practice Question

Practice Question 5:

As we approach the end of this book, here is the final practice question, which is commonly recommended for those with good knowledge of C++ programming. Make a number-guessing game. The computer chooses a random number in a range, such as 1 to 10. The user has three or four chances to guess the number correctly. If the guess is wrong, the program must mention that the guess was lesser or greater than the number in question. If all attempts are exhausted without the correct answer, the user is considered to have lost the game. If the user guesses it at any time, the user wins. The user must then be asked if the game should be repeated or be closed.

Use your knowledge of the various types of statements and the libraries you have used so far. Compare your program with the solution on page 118.

Hint: Use the following code to clear the input buffer, after repeated attempts:

```
std::cin.sync();
std::cin.clear();
```

Use loops and conditional statements, wherever required. Make variables to store the random number, the user's input and the number of attempts.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Conclusion

By now, this book should have provided you a good foundation in C++ programming. Remember that the best method of learning is to develop the capacity to explain something and by practically using it. As a popular quote says, “If you can't explain it to a six-year-old, you don't understand it.”

While this book is not a resource to obtain highly advanced programming knowledge or a gold-standard replacement for other resources, it intends to provide a good foundation for programmers. You should be able to use the same skills for other similar programming languages and developing applications. You will also find more clarity in learning even other programming languages.

We recommend advancing your knowledge of programming through other books and resources. Following are a few useful tips:

- Read the books ' A Complete Guide to Programming in C++' (by Ulla Kirch-Prinz and Peter Prinz) and 'C++ Primer' (5th Edition). The former is an excellent resource for advanced knowledge of C++ programming and also works as a reference library.
- If you are curious to know elaborate details of C++, read Bjarne Stroustrup’s book 'The C++ Programming Language'.
- If you are interested in game development, you can apply your programming skills easily and learn it, especially by taking up online courses for game engines such as Unity or Unreal Engine.

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Solutions

Chapter 2

Practice Question 1:

```
1  #include <iostream>
2  #include <limits>
3  
4  namespace CustomNamespace
5  {
6      float a {};
7  }
8  
9  namespace SecondNamespace
10 {
11     float a {};
12 }
13 
14 int main()
15 {
16     std::cout << "Enter any number: ";
17     std::cin >> CustomNamespace::a;
18     std::cout << '\n' << "Enter another number: ";
19     std::cin >> SecondNamespace::a;
20     std::cout << '\n' << "The first variable \'a\' has the value
   " << CustomNamespace::a << '\n';
21     std::cout << "The second variable \'a\' has the value " <<
   SecondNamespace::a << '\n';
22     std::cin.clear();
23     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
24     std::cin.get();
25     return 0;
26 }
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Chapter 3

Practice Question 2:

```
1  #include <iostream>
2  #include <limits>
3  using namespace std;
4  
5  char UserInput {}; // A variable to store the user's response
6  double a {};
7  double b {};
8  double result {};
9  
10 /* A function which presents options to the user. Called at the
   beginning of the program, like a main menu. */
11 void Ask()
12 {
13     cout << "What would you like to do? Enter the respective
   number and press Enter." << '\n';
14     cout << '\n' << "1: Addition" << '\n' << "2: Subtraction" <<
   '\n' << "3: Multiplication" << '\n' << "4: Division"  << '\n' <<
   "5: Quit this program" << '\n' << '\n';
15     cin >> UserInput;
16 }
17 
18 void ReceiveInput()
19 {
20     cout << '\n' << "Enter the first number: ";
21     cin >> a;
22     cout << "Enter the second number: ";
23     cin >> b;
24 }
25 
26 void Add(double x, double y)
27 {
28     result = x+y;
29     cout << '\n' << "The sum of " << a << " and " << b << " is "
   << '\n' << result << '\n';
30     cout << "---------------------" << '\n';
31 }
32 
33 void Subtract(double x, double y)
34 {
35     result = x-y;
36     cout << '\n' << b << " subtracted from " << a << " is " <<
   '\n' << result << '\n';
37     cout << "---------------------" << '\n';
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
38 }
39 
40 void Multiply(double x, double y)
41 {
42     result = x*y;
43     cout << '\n' << "The product of " << a << " and " << b << " 
   is " << '\n' << result << '\n';
44     cout << "----------------------" << '\n';
45 }
46 
47 void Divide(double x, double y)
48 {
49     result = x/y;
50     cout << '\n' << a << " divided by " << b << " is " << '\n' <<
   result << '\n';
51     cout << "----------------------" << '\n';
52 }
53 
54 int main()
55 {
56     do {
57         Ask();
58 
59        /* It is necessary to use quotation marks while mentioning
   the numbers, to compare them with the response of the user */
60         if (UserInput == '1')
61         {
62             ReceiveInput();
63             Add(a, b);
64         } else if (UserInput == '2') {
65             ReceiveInput();
66             Subtract(a, b);
67         } else if (UserInput == '3') {
68             ReceiveInput();
69             Multiply(a, b);
70         } else if (UserInput == '4') {
71             ReceiveInput();
72             Divide(a, b);
73         } else if (UserInput != '5') {
74             cout << "You have not entered a valid selection.
   Please try again." << '\n';
75             cout << "----------------------" << '\n';
76         }
77     } while (UserInput != '5');
78 
79     cout << "Thank you for using this calculator." << '\n' << "--
   --------------------" << '\n';
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
80
81      std::cin.clear();
82      std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
    '\n');
83      std::cin.get();
84      return 0;
85  }
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Chapter 4

Practice Question 3:

There are two possible solutions here. A major portion of the code need not to be changed.

(A) Create a string to first take the user's input and get only its first character. Use `#include <string>` in the beginning of the program. Then, modify the `Ask()` function as follows:

```
void Ask()
{
    std::cout << "What would you like to do? Enter the respective
number and press Enter." << '\n';
    std::cout << '\n' << "1: Addition" << '\n' << "2:
Subtraction" << '\n' << "3: Multiplication" << '\n' << "4:
Division"  << '\n' << "5: Quit this program" << '\n' << '\n';
    std::string Input {};
    std::getline(std::cin >> std::ws, Input);
    UserInput = Input[0]; // Get only the first character from
the string 'Input'
}
```

Now, the program will only consider the first character, as shown in the following example. Since '1' is the code for addition, it ignores anything after that and considers the response to be 1.

```
What would you like to do? Enter the respective number and press Enter.

1: Addition
2: Subtraction
3: Multiplication
4: Division
5: Quit this program

1a

Enter the first number: 10
Enter the second number: 5

The sum of 10 and 5 is
15
----------------------
```

(B) Change `UserInput` from a variable of datatype `char` to an object of class `std::string`. Use `#include <string>` in the beginning of the program. Then, modify the third line in the `Ask()` function:

```
   #include <string>

3  std::string UserInput {};
4  double a {};
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
5   double b {};
6   double result {};
7
8   /* A function which presents options to the user. Called at the
    beginning of the program, like a main menu. */
9   void Ask()
10  {
11      std::cout << "What would you like to do? Enter the respective
    number and press Enter." << '\n';
12      std::cout << '\n' << "1: Addition" << '\n' << "2:
    Subtraction" << '\n' << "3: Multiplication" << '\n' << "4:
    Division"  << '\n' << "5: Quit this program" << '\n' << '\n';
13      std::getline(std::cin >> std::ws, UserInput);
14  }
```

Now, go to the `main()` function and change the conditions in the do-`while` loop and the `if-else if` blocks accordingly.

```
do {
    Ask();

    if (UserInput == std::to_string(1))
    {
        ReceiveInput();
        Add(a, b);
    } else if (UserInput == std::to_string(2)) {
        ReceiveInput();
        Subtract(a, b);
    }
    } else if (UserInput == std::to_string(3)) {
        ReceiveInput();
        Multiply(a, b);
    } else if (UserInput == std::to_string(4)) {
        ReceiveInput();
        Divide(a, b);
    } else if (UserInput != std::to_string(5)) {
        std::cout << "You have not entered a valid selection.
Please try again." << '\n';
        std::cout << "----------------------" << '\n';
    }
} while (UserInput != std::to_string(5));
```

The function `std::to_string()` converts a given number to a string literal, so that it can be compared with a string variable directly, using an appropriate operator like == or `!=`. Note that in approach (B), we check if the user’s input is exactly a number between 1 to 5. While approach (A) checked only if the first character is a number between 1 to 5 and

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

ignores any other characters entered after that by the user, approach (B) will consider the response invalid even if the user enters a proper number first and adds any extra characters. The following example output of program (B) should make it clear.

```
What would you like to do? Enter the respective number and press Enter.

1: Addition
2: Subtraction
3: Multiplication
4: Division
5: Quit this program

1a
You have not entered a valid selection. Please try again.
----------------------
```

In approach (A), the program would consider this a valid response and ask the users the numbers to be added, as it would be interpreted as 1 (i.e., the user has chosen addition).

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Chapter 6

Practice Question 4:

```
1   #include <iostream>
2   #include <string>
3   #include <ctime>
4   #include <limits>
5   
6   std::string_view GetMonth(int month)
7   {
8       switch (month)
9       {
10          case 0: return "January";
11          case 1: return "February";
12          case 2: return "March";
13          case 3: return "April";
14          case 4: return "May";
15          case 5: return "June";
16          case 6: return "July";
17          case 7: return "August";
18          case 8: return "September";
19          case 9: return "October";
20          case 10: return "November";
21          case 11: return "December";
22          default: return "Invalid Month";
23      }
24  }
25  
26  std::string_view GetDay(int day)
27  {
28      switch (day)
29      {
30          case 0: return "Sunday";
31          case 1: return "Monday";
32          case 2: return "Tuesday";
33          case 3: return "Wednesday";
34          case 4: return "Thursday";
35          case 5: return "Friday";
36          case 6: return "Saturday";
37          default: return "Invalid Day";
38      }
39  }
40  
41  int main()
42  {
43      struct tm* Time;
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
44      time_t CurrentTime;
45      time(&CurrentTime);
46      Time = localtime(&CurrentTime);
47      int Hour { Time->tm_hour };
48
49      std::cout << "Today is " << Time->tm_mday << " " <<
    GetMonth(Time->tm_mon) << " " << Time->tm_year + 1900 << " (" <<
    GetDay(Time->tm_wday) << ")" << '\n';
50      if (Hour > 12)
51      {
52          std::cout << "The time is " << (Hour - 12) << ":" <<
    Time->tm_min << ":" << Time->tm_sec << " PM" << '\n';
53      } else {
54          std::cout << "The time is " << Time->tm_hour << ":" <<
    Time->tm_min << ":" << Time->tm_sec << " AM" << '\n';
55      }
56
57      std::cin.clear();
58      std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
    '\n');
59      std::cin.get();
60      return 0;
61  }
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Practice Question 5:

```
1   #include <iostream>
2   #include <string>
3   #include <ctime>
4   #include <limits>
5   
6   int main()
7   {
8       int number{}, attempts{}, guess{};
9       bool guessed{ false };
10      char status{'1'}; // A variable to check if the player wants to repeat the game or exit.
11      struct tm *Time;
12      time_t CurrentTime;
13      time(&CurrentTime);
14      Time = localtime(&CurrentTime);
15      int seconds{Time->tm_sec};
16      srand(seconds); // Seeding the random number generator with the current second.
17  
18      std::cout << "----------------------------------------" << '\n' << "Welcome to the Number-Guessing Game!" << '\n';
19      std::cout << '\n' << "The computer will select a random number between 1 to 10. You have three attempts to guess it. If you guess it correctly within them, you win. Otherwise, the system wins." << '\n' << '\n';
20  
21      do {
22          guessed = false;
23          attempts = 0;
24          number = (rand() % 10) + 1; // Generate a random number between 1 and 10.
25          std::cout << '\n' << "----------------------------------------" << '\n';
26          std::cout << "A random number has been chosen between 1 to 10. Enter your guess." << '\n' << '\n';
27  
28          while (!guessed && attempts < 3)
29          {
30              std::cin.sync(); // Clear input buffer
31              std::cin.clear();
32              std::cout << "Attempt " << (attempts + 1) << ": ";
33              std::cin >> guess;
34              attempts++;
35  
36              if (guess >= 1 && guess <= 10)
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

```
37             {
38                 if (guess == number)
39                 {
40                     guessed = true;
41                 } else if (guess < number)
42                 {
43                     std::cout << "Your guess is low." << '\n';
44                 } else if (guess > number)
45                 {
46                     std::cout << "Your guess is high." << '\n';
47                 }
48             } else {
49                     std::cout << "Invalid input. Please enter a
   number between 1 and 10." << '\n';
50             }
51         } // End of the guessing loop
52 
53         if (guessed)
54         {
55             std::cout << "Congratulations! You guessed the number
   correctly." << '\n';
56         } else {
57             std::cout << "Sorry, you've used all your attempts.
   The correct number was " << number << "." << '\n';
58         }
59 
60         std::cout << "Would you like to play again? If yes, enter
   1. Enter 0 to exit the game." << '\n';
61         do {
62             std::cin.get(status);
63         } while (status != '1' && status != '0');
64     } while (status == '1');
65 
66     std::cout << '\n' << "---------------------------------------
   -" << '\n';
67     std::cout << "Thank you for playing!" << '\n';
68 
69     std::cin.clear();
70     std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
   '\n');
71     std::cin.get();
72     return 0;
73 }
```


© Copyrights 2015-26 Nilapatri (Praajna Pattada)

Index

A

Array p. 79

C

Class p. 82

- Template p. 88

Compiler p. 7,

- Working p. 9
- Installation p. 10

Constants p. 32
CTime p. 102

D

Data type p. 29
Declaration p. 25
Definition p. 25

E

Enumeration p. 90
Escape Sequence p. 34

F

Function p. 49

- Overloading p. 52
- Template p. 62

FStream p. 98

G

Garbage value p. 30

H

Header p. 37

I

Identifier p. 24
IOStream p. 19

K

Keywords p. 27

M

Macros p. 70

N

Namespace p. 44

O

Operator p. 33

P

Pointer p. 73, 77

R

Reference pp. 73, 75

S

Structure

- of a Program p. 24
- Data type p. 86

V

Variable p. 29

© Copyrights 2015-26 Nilapatri (Praajna Pattada)

About the Author

Praajna Pattada Hari Kumaara Varma is an aspiring family physician and author on spirituality, from Karnataka, India. He is also a developer of video games and websites, a cartoonist and story writer. He started his hobby of developing websites and video games at the age of 10 years and started studying spirituality and learning about different Hindu philosophies at 15 years of age. He founded Achyuta Bhakti Deets in 2022, to propagate spirituality and philosophy in a way which is practically applicable and benefits the present-day society. Its objectives also include propagating the knowledge of and encouraging learning Samskrta, the Itihaasas, Puraanas, Aagamas and Vaishnava philosophy, especially Madhva Siddhaanta, among all Saattvika-minded people regardless of caste, creed, sex, race and any such identities.

In 2015, he founded 'Nilapatri', his first official website, initially as a platform for selling video games made using Scratch. Currently, Nilapatri hosts a variety of software, including online keyboards for typing in Samskrta and other Indian languages. He developed his first major video game at the age of 17 years and has been a learner of various computer programming languages, especially C++ and C#.

॥ श्रीकृष्णार्पणमस्तु ॥

© Copyrights 2015-26 Nilapatri (Praajna Pattada)