C++ and Algorithmic Thinking for the Complete Beginner
Part 1 of 5
Kindle Edition

http://www.bouraspage.com

Warning and Disclaimer

This book is designed to provide information about learning "Algorithmic Thinking," mainly through the use of C++ programming language. Every effort has been taken to make this book compatible with all previous releases of C++, and it is almost certain to be compatible with any future releases of C++.

The information is provided on an "as is" basis. The authors shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the files that may accompany it.

This book is dedicated to
the memory of
my cousin, Remi.

Contents at a Glance

Preface **8**

Section 1 Introductory Knowledge **14**

Chapter 1 How a Computer Works *15*

Chapter 2 C++ *21*

Chapter 3 Software Packages to Install *23*

Review Questions in "Introductory Knowledge" *35*

Section 2 Getting Started with C++ **36**

Chapter 4 Introduction to Basic Algorithmic Concepts *37*

Chapter 5 Variables and Constants *51*

Chapter 6 Handling Input and Output *63*

Chapter 7 Operators *71*

Chapter 8 Trace Tables *85*

Chapter 9 Using NetBeans IDE *94*

Review Questions in "Getting Started with C++" *110*

Section 3 Sequence Control Structures **112**

Chapter 10 Introduction to Sequence Control Structures *113*

Chapter 11 Manipulating Numbers *122*

Chapter 12 Complex Mathematical Expressions *136*

Chapter 13 Exercises With a Quotient and a Remainder *142*

Chapter 14 Manipulating Strings *152*

Review Questions in "Sequence Control Structures" *167*

Index **168**

Table of Contents

Preface **8**
About the Authors 9
Acknowledgments 10
How This Book is Organized 10
Who Should Buy This Book? 10
Where to Find Answers to Review Questions and Exercises 11
How to Report Errata 11
Conventions Used in This Book 12

Section 1 Introductory Knowledge **14**

Chapter 1 How a Computer Works ***15***
1.1 Introduction 15
1.2 What is Hardware? 15
1.3 What is Software? 16
1.4 How a Computer Executes (Runs) a Program 16
1.5 Compilers and Interpreters 16
1.6 What is Source Code? 17
1.7 Review Questions: True/False 17
1.8 Review Questions: Multiple Choice 18

Chapter 2 C++ ***21***
2.1 What is C++? 21
2.2 What is the Difference Between a Script and a Program? 21
2.3 Why You Should Learn C++ 21
2.4 How C++ Works 22

Chapter 3 Software Packages to Install ***23***
3.1 Boost C++ Libraries 23
3.2 How to Set Up Boost C++ Libraries 23
3.3 Installing the C++ Compiler and the Debugger for Windows 23
3.4 NetBeans IDE 28
3.5 How to Set Up NetBeans IDE 28

Review Questions in "Introductory Knowledge" ***35***

Section 2 Getting Started with C++ **36**

Chapter 4 Introduction to Basic Algorithmic Concepts ***37***
4.1 What is an Algorithm? 37
4.2 The Algorithm for Making a Cup of Tea 37
4.3 Properties of an Algorithm 37
4.4 Okay About Algorithms. But What is a Computer Program Anyway? 38

4.5 The Party of Three! ... 38
4.6 The Three Main Stages Involved in Creating an Algorithm ... 38
4.7 Flowcharts ... 39
Exercise 4.7-1 Finding the Average Value of Three Numbers ... 41
4.8 What are "Reserved Words"? ... 42
4.9 What is the Difference Between a Statement and a Command? ... 43
4.10 What is Structured Programming? ... 43
4.11 The Three Fundamental Control Structures ... 43
Exercise 4.11-1 Understanding Control Structures Using Flowcharts ... 44
4.12 Your First C++ Program ... 45
4.13 What is the Difference Between Syntax Errors and Logic Errors? ... 46
4.14 Commenting Your Code ... 46
4.15 User-Friendly Programs ... 47
4.16 Review Questions: True/False ... 48
4.17 Review Questions: Multiple Choice ... 49

Chapter 5 Variables and Constants ... 51
5.1 What is a Variable? ... 51
5.2 What is a Constant? ... 53
5.3 How Many Types of Variables and Constants Exist in C++? ... 56
5.4 Rules for Naming Variables in C++ ... 56
5.5 Rules for Naming Constants in C++ ... 57
5.6 What Does the Phrase "Declare a Variable" Mean? ... 57
5.7 How to Declare Variables in C++ ... 58
5.8 How to Declare Constants in C++ ... 59
5.9 Review Questions: True/False ... 60
5.10 Review Questions: Multiple Choice ... 60
5.11 Review Exercises ... 62

Chapter 6 Handling Input and Output ... 63
6.1 Which Statement Outputs Messages and Results on a User's Screen? ... 63
6.2 How to Output Special Characters ... 65
6.3 Which Statement Lets the User Enter Data? ... 67
6.4 Review Questions: True/False ... 69
6.5 Review Questions: Multiple Choice ... 69

Chapter 7 Operators ... 71
7.1 The Value Assignment Operator ... 71
7.2 Arithmetic Operators ... 73
7.3 What is the Precedence of Arithmetic Operators? ... 74
7.4 Compound Assignment Operators ... 76
Exercise 7.4-1 Which C++ Statements are Syntactically Correct? ... 77
Exercise 7.4-2 Finding Variable Types ... 77

7.5 Incrementing/Decrementing Operators ... 78
7.6 String Operators ... 79
Exercise 7.6-1 Concatenating Names ... 80
7.7 Review Questions: True/False ... 80
7.8 Review Questions: Multiple Choice ... 81
7.9 Review Exercises ... 83

Chapter 8 Trace Tables ... 85
8.1 What is a Trace Table? ... 85
Exercise 8.1-1 Creating a Trace Table ... 86
Exercise 8.1-2 Swapping Values of Variables ... 87
Exercise 8.1-3 Swapping Values of Variables – A Second Approach ... 89
Exercise 8.1-4 Creating a Trace Table ... 90
Exercise 8.1-5 Creating a Trace Table ... 91
8.2 Review Questions: True/False ... 92
8.3 Review Exercises ... 92

Chapter 9 Using NetBeans IDE ... 94
9.1 Creating a New C++Project ... 94
9.2 Writing and Executing a C++ Program ... 96
9.3 What "Debugging" Means ... 100
9.4 Debugging C++ Programs with NetBeans IDE ... 101
9.5 Review Exercises ... 108

Review Questions in "Getting Started with C++" ... 110

Section 3 Sequence Control Structures ... 112

Chapter 10 Introduction to Sequence Control Structures ... 113
10.1 What is the Sequence Control Structure? ... 113
Exercise 10.1-1 Calculating the Area of a Parallelogram ... 113
Exercise 10.1-2 Calculating the Area of a Circle ... 114
Exercise 10.1-3 Calculating Fuel Economy ... 115
Exercise 10.1-4 Where is the Car? Calculating Distance Traveled ... 115
Exercise 10.1-5 Kelvin to Fahrenheit ... 117
Exercise 10.1-6 Calculating Sales Tax ... 118
Exercise 10.1-7 Calculating a Sales Discount ... 118
Exercise 10.1-8 Calculating the Sales Tax Rate and Discount ... 119
10.2 Review Exercises ... 120

Chapter 11 Manipulating Numbers ... 122
11.1 Introduction ... 122
11.2 Useful Mathematical Functions ... 123
Exercise 11.2-1 Calculating the Distance Between Two Points ... 130
Exercise 11.2-2 How Far Did the Car Travel? ... 132
11.3 Review Questions: True/False ... 133

11.4 Review Questions: Multiple Choice 133
11.5 Review Exercises 134

Chapter 12 Complex Mathematical Expressions 136
12.1 Writing Complex Mathematical Expressions 136
Exercise 12.1-1 Representing Mathematical Expressions in C++ 136
Exercise 12.1-2 Writing a Mathematical Expression in C++ 137
Exercise 12.1-3 Writing a Complex Mathematical Expression in C++ 137
12.2 Review Exercises 139

Chapter 13 Exercises With a Quotient and a Remainder 142
13.1 Introduction 142
Exercise 13.1-1 Calculating the Quotient and Remainder of Integer Division 142
Exercise 13.1-2 Finding the Sum of Digits 143
Exercise 13.1-3 Displaying an Elapsed Time 148
Exercise 13.1-4 Reversing a Number 150
13.2 Review Exercises 151

Chapter 14 Manipulating Strings 152
14.1 Introduction 152
14.2 The Position of a Character in a String 153
14.3 Retrieving an Individual Character From a String 153
Exercise 14.3-1 Displaying a String Backwards 154
14.4 Useful String Functions 155
Exercise 14.4-1 Switching the Order of Names 160
Exercise 14.4-2 Creating a Login ID 161
Exercise 14.4-3 Creating a Random Word 162
14.5 Review Questions: True/False 163
14.6 Review Questions: Multiple Choice 164
14.7 Review Exercises 166

Review Questions in "Sequence Control Structures" 167

Index 168

Preface

About the Authors

Aristides S. Bouras

Aristides[1] S. Bouras was born in 1973. During his early childhood, he discovered a love of computer programming. He got his first computer at the age of 12, a Commodore 64, which incorporated a ROM-based version of the BASIC programming language and 64 kilobytes of RAM!!!

He holds a degree in Computer Engineering from the Technological Educational Institute of Piraeus, and a degree in Electrical and Computer Engineering from the Democritus Polytechnic University of Thrace.

He worked as a software developer at a company that specialized in industrial data flow and labelling of products. His main job was to develop software applications for data terminals (originally in TALL and later in VB.NET language), as well as PC software applications for collecting and storing data on a Microsoft SQL Server.

He has developed many applications such as warehouse managing systems and websites for companies and other organizations. Nowadays, he works as a high school teacher. He mainly teaches courses in computer networks, programming tools for the Internet/intranets, and databases.

He is married to Loukia V. Ainarozidou and they have two children.

Loukia V. Ainarozidou

Loukia V. Ainarozidou was born in 1975. She got her first computer at the age of 13, an Amstrad CPC6128 with 128 kilobytes of RAM and an internal 3-inch floppy disk drive!!!

She holds a degree in Computer Engineering from the Technological Educational Institute of Piraeus, and a degree in Electrical and Computer Engineering from the Democritus Polytechnic University of Thrace.

She worked as a supervisor in the data logistics department of a company involved in the packaging of fruit and vegetables. Nowadays, she works as a high school teacher. She mainly teaches courses in computer networks, computer programming, and digital design.

She is married to Aristides S. Bouras and they have two children.

[1] Aristides (530 BC–468 BC) was an ancient Athenian statesman and general. The ancient historian Herodotus cited him as “the best and most honorable man in Athens.” He was so fair in all that he did that he was often referred to as “Aristides the Just.” He flourished in the early quarter of Athens’s Classical period and helped Athenians defeat the Persians at the battles of Salamis and Plataea.

Acknowledgments

If it weren't for Dr. Yannis T. Kappos, we may never have written this book. As a renowned author of technical books on AutoCAD, he inspired us to take a seat and start writing. We wish to express our enormous appreciation to him for generously spending his time answering all of our questions—even the foolish ones.

We would also like to extend our thanks, with particular gratefulness, to our friend and senior editor Victoria (Vicki) Austin for her assistance in copy editing. Without her, this book might not have reached its full potential. With her patient guidance and valuable and constructive suggestions, she helped us bring this book up to a higher level!

How This Book is Organized

The book you hold in your hands follows the spiral curriculum teaching approach, a method proposed in 1960 by Jerome Bruner, an American psychologist. According to this method, as a subject is being taught, basic ideas are revisited at intervals—at a more sophisticated level each time—until the reader achieves a complete understanding of the subject. First, the reader learns the basic elements without worrying about the details. Later, more details are taught and basic elements are mentioned again and again, eventually being stored in the brain's long term memory.

According to Jerome Bruner, learning requires the student's active participation, experimentation, exploration, and discovery. This book contains many examples, most of which can be practically performed. This gives the reader the opportunity to get his or her hands on C++ and become capable of creating his or her own programs.

Who Should Buy This Book?

This book is for anyone who wants to learn computer programming and knows absolutely nothing about it. Of course, if you are wondering whether this book is going to teach you how to create amazing applets or incredible desktop or mobile applications, the answer is "no"—that is a job for other books. So many books out there can teach you those skills in C++, Java, or C#. Many of them even claim that they can teach you in 24 hours! Don't laugh! They probably can do that, but all of them take one thing for granted—that the reader knows some basics about computer programming. None of those books, unfortunately, bothers to teach you the first thing that a novice programmer needs to learn, which is "Algorithmic Thinking."

Algorithmic Thinking involves more than just learning code. It is a problem solving process that involves learning *how to* code. With over 800 pages, and containing more than 300 solved and 400 unsolved exercises, over 450 true/false, 150 multiple choice, and 180 review questions (the solutions and the answers to which can be found on the Internet), this book is ideal for students, teachers, professors, novices or average programmers, or for anyone who wants to start learning or teaching computer programming using the proper conventions and techniques.

Where to Find Answers to Review Questions and Exercises

Answers to all of the review questions, as well as the solutions to all review exercises, are available free of charge on the Internet. You can download them from the following address:

http://www.bouraspage.com

How to Report Errata

Although we have taken great care to ensure the accuracy of our content, mistakes do occur. If you find a mistake in this book, either in the text or the code, we encourage you to report it to us. By doing so, you can save other readers from frustration and, of course, help us to improve the next version of this book. If you find any errata, please feel free to report them by visiting the following address:

http://www.bouraspage.com

Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website, and added to any existing list of errata.

Conventions Used in This Book

Following are some explanations on the conventions used in this book. "Conventions" refer to the standard ways in which certain parts of the text are displayed.

C++ Statements

This book uses plenty of examples written in C++ language. C++ statements are shown in a typeface that looks like this.

```
This is a C++ statement
```

Keywords, Variables, Functions, and Arguments Within the Text of a Paragraph

Keywords, variables, functions, and arguments are sometimes shown within the text of a paragraph. When they are, the special text is shown in a typeface different from that of the rest of the paragraph. For instance, `first_name = 5` is an example of a C++ statement within the paragraph text.

Words in Italics

You may notice that some of the special text is also displayed in italics. In this book, italicized words are general types that must be replaced with the specific name appropriate for your data. For example, the general form of a C++ statement may be presented as

```
void name ( type1 arg1, type2 arg2 )
```

In order to complete the statement, the keywords *name*, *type1*, *arg1*, *type2*, and *arg2* must be replaced with something meaningful. When you use this statement in your program, you might use it in the following form.

```
void display_rectangle ( int width, int height )
```

Three dots (...): an Ellipsis

In the general form of a statement you may also notice three dots (...), also known as an "ellipsis," following a list in an example. They are not part of the statement. An ellipsis indicates that you can have as many items in the list as you want. For example, the ellipsis in the general form of the statement

```
display_messages ( arg1, arg2, … )
```

indicates that the list may contain more than two arguments. When you use this statement in your program, your statement might be something like this.

```
display_messages ( message1, "Hello", message2, "Hi!" )
```

Square Brackets

The general form of some statements or functions may contain "square brackets" [ ], which indicate that the enclosed section is optional. For example, the general form of the statement

```
str.substr ( beginIndex [, endIndex ] )
```

indicates that the section [, *endIndex*] can be omitted.

The following two statements produce different results but they are both syntactically correct.

```
a = str.substr (3);
b = str.substr (3, 9);
```

The Dark Header

Most of this book's examples are shown in a typeface that looks like this.

project_31_2_3

```
#include <iostream>
using namespace std;
int main() {
  int a, b;

  a = 1;
  b = 2;
  cout << a + b;
  return 0;
}
```

The dark header **project_31_2_3** on top indicates the filename that you must open to test the program. All the examples that contain this header can be found free of charge on the Internet. You can download them from the following address

http://www.bouraspage.com

Notices

Very often this book uses notices to help you better understand the meaning of a concept. Notices look like this.

***Notice**: This typeface designates a note.*

Something Already Known or Something to Remember

Very often this book can help you recall something you have already learned (probably in a previous chapter). Other times, it will draw your attention to something you should memorize. Reminders look like this.

***Remember!** This typeface designates something to recall or something that you should memorize.*

Section 1

Introductory Knowledge

Chapter 1
How a Computer Works

1.1 Introduction

In today's society, almost every task requires the use of a computer. In schools, students use computers to search the Internet and to send emails. At work, people use them to make presentations, to analyze data, and to communicate with customers. At home, people use them to play games and to chat with other people all over the world. Of course, don't forget smartphones such as iPhones. They are computers as well!

Computers can perform so many different tasks because of their ability to be programmed. In other words, a computer can perform any job that a program tells it to. A program is a set of statements (often called instructions or "commands") that a computer follows in order to perform a specific task.

Programs (usually referred as "application software") are essential to a computer, because without them a computer is a dummy machine that can do nothing at all. The program actually tells the computer what to do and when to do it. The programmer or the software developer is the person who designs, creates, and tests computer programs.

This book introduces you to the basic concepts of computer programming using the C++ language.

1.2 What is Hardware?

The term "hardware" refers to all devices or components that make up a computer. If you have ever opened the case of a computer or a laptop you have probably seen many of its components, such as the microprocessor (CPU), the memory, and the hard disk. A computer is not a device but a system of devices that all work together. The basic components of a typical computer system are discussed here.

- **The Central Processing Unit (CPU)**

  This is the part of a computer that actually performs all the tasks defined in a program.

- **Main Memory (RAM – Random Access Memory)**

  This is the area where the computer holds the program (while it is being executed/run) as well as the data that the program is working with. All programs and data stored in this type of memory are lost when you shut down your computer or you unplug it from the wall outlet.

- **Secondary Storage Devices**

  This is usually the hard disk, and sometimes (but more rarely) the CD/DVD drive. In contrast to main memory, this type of memory can hold data for a longer period of time, even if there is no power to the computer. However,

programs stored in this memory cannot be directly executed. They must be transferred to a much faster memory; that is, the main memory.

- **Input Devices**

  Input devices are all those devices that collect data from the outside world and enter them into the computer for further processing. Keyboards, mice, and microphones are all input devices.

- **Output Devices**

  Output devices are all those devices that output data to the outside world. Monitors (screens) and printers are output devices.

1.3 What is Software?

Everything that a computer does is under the control of software. There are two categories of software: system software and application software.

System software is the program that controls and manages the basic operations. For example, it controls the internal operations of a computer, manages all devices connected to it, saves data, loads data, and allows other programs to be executed. Windows, Linux, Mac OS X, Android, and iOS are all examples of system software. Another term for this category of programs is "operating systems."

Application software refers to all the other programs that you use for your everyday tasks, such as browsers, word processors, notepads, games, and many more.

1.4 How a Computer Executes (Runs) a Program

When you turn on your computer, the main memory (RAM) is completely empty. The first thing the computer needs to do is to transfer the operating system from the hard disk to the main memory.

After the operating system is loaded to main memory, you can execute (run) any program (application software) you like. This is usually done by clicking, double clicking, or tapping the program's corresponding icon. For example, let's say you click on the icon of your favorite word processor. This action orders your computer to load (or copy) the word processing program from your hard disk to the main memory so the CPU can execute it.

***Remember**! Programs are stored on secondary storage devices such as hard disks. When you install a program on your computer, the program is actually copied to your hard disk. But when you execute a program, the program is copied (loaded) from your hard disk to the main memory, and that copy of the program is executed.*

***Notice**: The terms "run" and "execute" are synonymous.*

1.5 Compilers and Interpreters

Computers can execute programs that are written in a strictly defined computer language. You cannot write a program using a natural language such as English or Greek, because your computer won't understand you!

But what does a computer actually understand? A computer can understand a specific low-level language called the "machine language." In a machine language all

statements (or commands) are made up of zeros and ones. The following is an example of a program written in a machine language, that calculates the sum of two numbers.

```
0010 0001 0000 0100
0001 0001 0000 0101
0011 0001 0000 0110
0111 0000 0000 0001
```

Shocked? Don't worry, you are not going to write programs this way. Hopefully, no one writes computer programs this way anymore. Nowadays, all programmers write their programs in a high-level language and then they use a special program to translate them into a machine language. There are two types of programs that programmers use to perform translation: compilers and interpreters.

A compiler is a program that translates statements written in a high-level language into a separate machine language program. The machine language program can then be executed any time you wish. After the translation, the compiler is no longer required.

An interpreter is a program that simultaneously translates and executes the statements written in a high-level language. As the interpreter reads each individual statement in the program, it translates it into a machine language code and then directly executes it. This process is repeated for every statement in the program.

1.6 What is Source Code?

The statements (often called instructions or commands) that the programmer writes in a high-level language are called "source code" or simply "code." The programmer first types the source code into a program known as a code editor, and then uses either a compiler to translate it into a machine language program, or an interpreter to translate and execute it at the same time. NetBeans IDE is an example of an Integrated Development Environment (IDE) that enables programmers to both write and execute their source code. You will learn more about NetBeans IDE in Chapter 3.

1.7 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. Modern computers can perform so many different tasks because they have many gigabytes of RAM.
2. A computer can operate without a program.
3. A hard disk is an example of hardware.
4. Data can be stored in main memory (RAM) for a long period of time, even if there is no power to the computer.
5. Data is stored in main memory (RAM), but programs are not.
6. Speakers are an example of an output device.
7. Windows and Linux are examples of software.
8. A media player is an example of system software.

9. When you turn on your computer, the main memory (RAM) already contains the operating system.
10. When you open your word processing application, it is actually copied from a secondary storage device to the main memory (RAM).
11. In a machine language, all statements (commands) are a sequence of zeros and ones.
12. Nowadays, a computer cannot understand zeros and ones.
13. Nowadays, software is written in a language composed of ones and zeros.
14. Software refers to the physical components of a computer.
15. In a high-level computer programming language, the computer does not understand zeros and ones.
16. The compiler and the interpreter are software.
17. The compiler translates source code to an executable file.
18. The interpreter creates a machine language program.
19. After the translation, the interpreter is not required anymore.
20. Source code can be written using a simple text editor.
21. Source code can be executed by a computer without compilation or interpretation.
22. A program written in machine language requires compilation (translation).
23. A compiler translates a program written in a high-level language.

1.8 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. Which of the following is **not** computer hardware?
    a. a hard disk
    b. a DVD disc
    c. a sound card
    d. the main memory (RAM)
2. Which of the following is **not** a secondary storage device?
    a. a DVD reader/writer device
    b. a hard disk
    c. a USB flash drive
    d. RAM
3. Which one of the following operations can**not** be performed by the CPU?
    a. Transfer data to the main memory (RAM).
    b. Display data to the user.
    c. Transfer data from the main memory (RAM).
    d. Perform arithmetic operations.

4. A touch screen is
    a. an input device.
    b. an output device.
    c. both of the above
5. Which of the following is **not** software?
    a. Windows
    b. Linux
    c. iOS
    d. a video game
    e. a web browser
    f. All of the above are software.
6. Which of the following statements is correct?
    a. Programs are stored on the hard disk.
    b. Programs are stored on DVD discs.
    c. Programs are stored in main memory (RAM).
    d. All of the above are correct.
7. Which of the following statements is correct?
    a. Programs can be executed directly from the hard disk.
    b. Programs can be executed directly from a DVD disc.
    c. Programs can be executed directly from the main memory (RAM).
    d. All of the above are correct.
    e. None of the above is correct.
8. Programmers can**not** write computer programs in
    a. C++.
    b. natural language such as English, Greek, and so on.
    c. machine language.
9. A compiler translates
    a. a program written in machine language into a high-level language program.
    b. a program written in a natural language (English, Greek, etc.) into a machine language program.
    c. a program written in high-level language into a machine language program.
    d. none of the above
    e. all of the above
10. Machine language is
    a. a language that machines use to communicate with each other.

b. a language made up of numerical instructions that is used directly by a computer.

c. a language that uses English words for operations.

11. If two identical statements are one after the other, the interpreter

a. translates the first one and executes it, then it translates the second one and executes it.

b. translates the first one, then translates the second one, and then executes them both.

c. translates only the first one (since they are identical) and then executes it two times.

Chapter 2
C++

2.1 What is C++?

C++ is a widely used general-purpose computer programming language that allows programmers to create large-scale applications, embedded systems, operating system kernels, drivers, client-server applications, and many other types of software. C++ is an extension of its predecessor, the C programming language. The main difference between C++ and C is that C++ includes modern programming techniques such as Object Oriented Programming (OOP.) Moreover, in C++ it is possible to write code either in a "C style," or in "object-oriented style"—or even in both!

2.2 What is the Difference Between a Script and a Program?

Technically speaking, a script is *interpreted* whereas a program is *compiled*, but this is actually not their main difference. There is another small yet more important difference between them!

The main purpose of a script written in a scripting language such as JavaScript, or VBA (Visual Basic for Applications) is to control another application. So you can say that, in some ways JavaScript controls the web browser, and VBA controls a Microsoft Office application such as MS Word or MS Excel.

On the other hand, a program written in a programming language such as C++, C#, or Visual Basic executes independently of any other application. A program is compiled into a separate set of machine language instructions that can then be executed as stand-alone any time the user wishes.

***Notice**: Macros of Microsoft Office are scripts written in VBA. Their purpose is to automate certain functions within Microsoft Office.*

***Remember**! A script requires a hosting application in order to execute. A script cannot be executed as stand-alone.*

2.3 Why You Should Learn C++

C++ is what is known as a “high-level” computer language. The C++ coding style is quite easy to understand and it is very efficient on multiple platforms such as Windows, Linux, and Unix. C++ is a very flexible yet powerful language, making it most suitable for developing large-scale applications, embedded systems, operating system kernels, drivers, client-server applications, art applications, music players, or even video games.

C++ is everywhere! It is used on desktop computers, on laptops, and even in data centers costing millions of dollars. With millions of developers worldwide, C++ enables efficient development of exciting applications and services. There are billions—probably even trillions—of lines of code already written in C++ and your

possibilities for code reuse are huge! This is why so many companies and organizations prefer using C++ to any other programming language. This is also a very good reason why you should actually learn C++!

2.4 How C++ Works

Computers do not understand natural languages such as English or Greek, so you need a computer language such as C++ to communicate with them. C++ is a very powerful high-level computer language. The C++ compiler converts C++ language to a language that computers can actually understand, and that is known as the "machine language."

There are C++ compilers for just about every type of operating system. Thus you can compile your source code and create executable files, for example, for Windows, for Linux, or for OS X.

Chapter 3
Software Packages to Install

3.1 Boost C++ Libraries

Boost is a collection of libraries for the C++ programming language that provide support for tasks and structures such as event handling, string handling, memory handling, asynchronous input/output (I/O), and many more. It contains more than 140 libraries to help you increase the productivity of C++ without waiting for the next revision of the C++ standard.

In this book, some exercises and examples use a very small set of Boost libraries. However, if you need even more information you can visit:

http://www.boost.org/doc/libs/

3.2 How to Set Up Boost C++ Libraries

To install Boost C++ Libraries, you must download them, free of charge, from the following address:

http://www.boost.org

Click on the download link and select the corresponding file. Usually the filename of this download looks like this:

boost_X_XX_X.zip

where X_XX_XX is the version of Boost. When the download completes, unzip the file to your C: drive. This will create a folder named C:\boost_X_XX_X on your C: drive. Rename it to C:\boost.

3.3 Installing the C++ Compiler and the Debugger for Windows

This step is for Windows users only and describes how to install:

- g++, the GNU C++ compiler,
- gdb, the GNU debugger, and
- Make, a utility for compiling and linking projects.

In order to install these tools, you must first download Cygwin. Cygwin is a large collection of GNU and Open Source tools that provide computers than run on Windows with a functionality similar to a Linux distribution. Download it, free of charge, from the following address:

https://cygwin.com/install.html

Choose either the 32-bit or the 64-bit version, depending on the version of your operating system, and download the corresponding setup file. When the download is complete, run the installer.

Click on the "Next" button to go through the installation screens. When you are prompted to choose a download source, choose "Install From Internet" as shown in **Figure 3–1**.

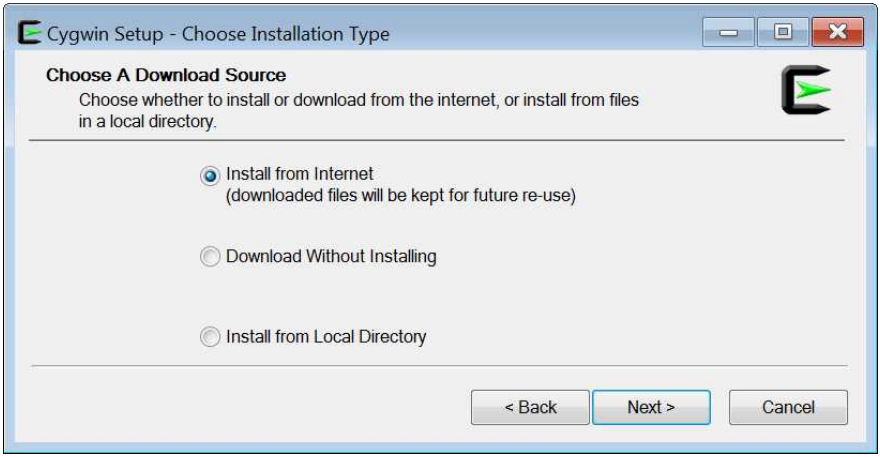


Figure 3–1 Choosing installation type

The next screen prompts you to select the installation folder. Leave the proposed folder, as shown in **Figure 3–2**.

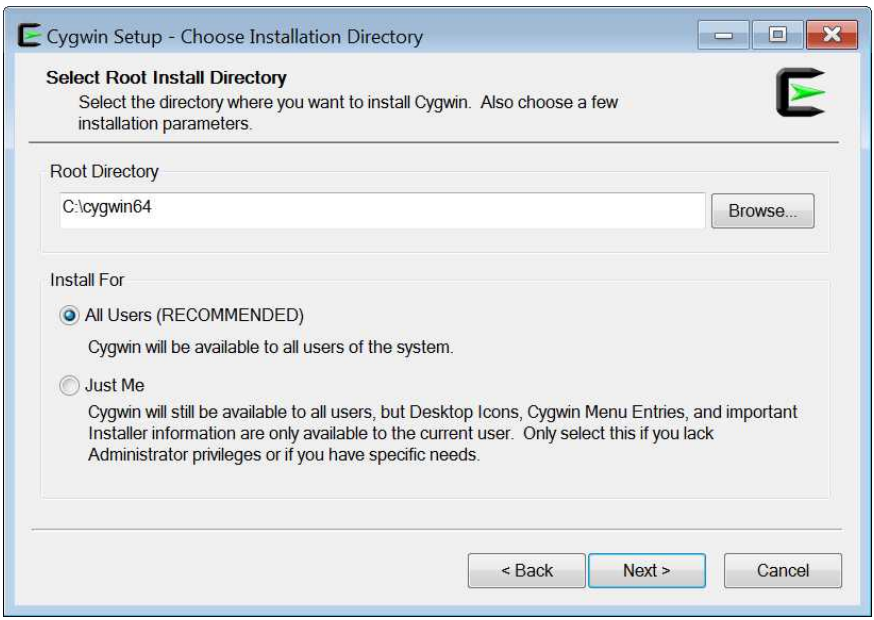


Figure 3–2 Choosing installation folder

The next screen prompts you to select the local package folder. Leave the proposed folder, as shown in **Figure 3–3**.

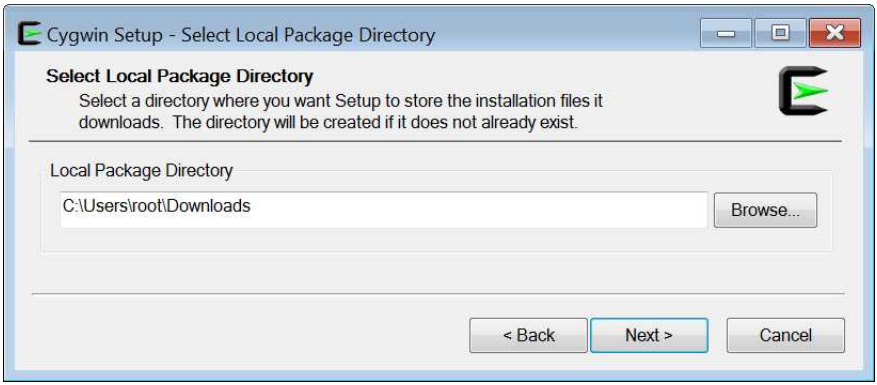


Figure 3–3 Choosing local package folder

The next screen prompts you to select the connection type. Make sure that the option "Direct connection" is selected, as shown in **Figure 3–4**.

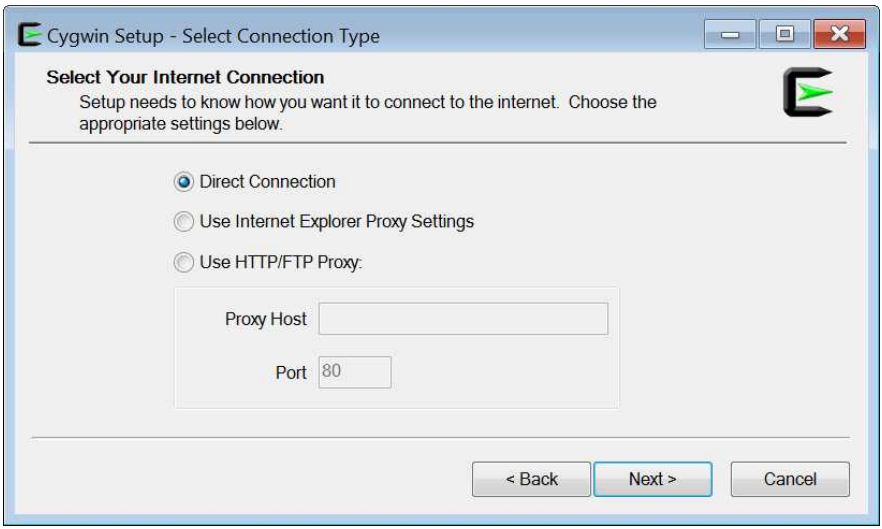


Figure 3–4 Choosing connection type

On the next screen choose the download site that is geographically the closest to you, as shown in **Figure 3–5.**

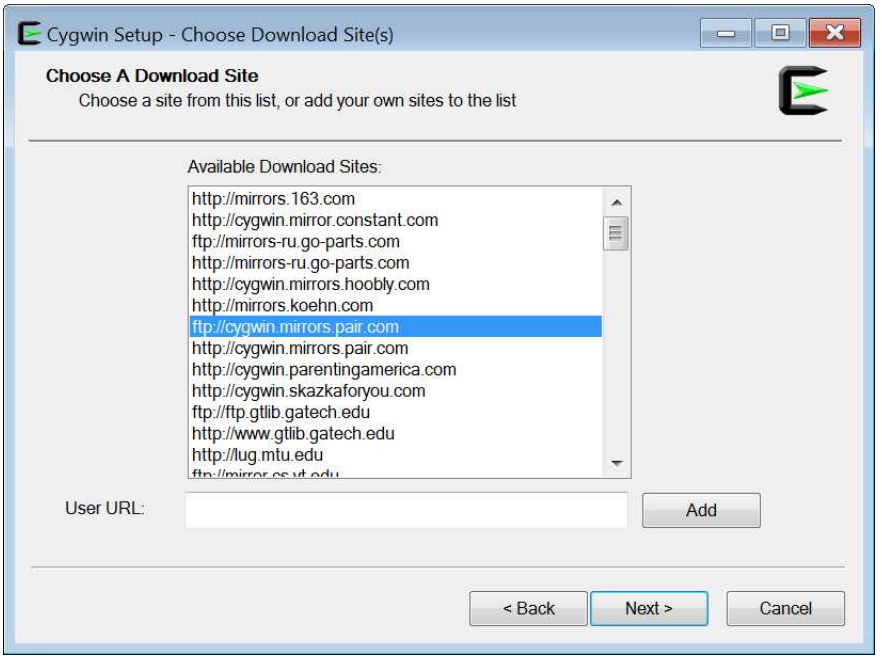


Figure 3–5 Choosing download site

After a brief delay, the next screen appears, asking you to specify the packages you want to install.

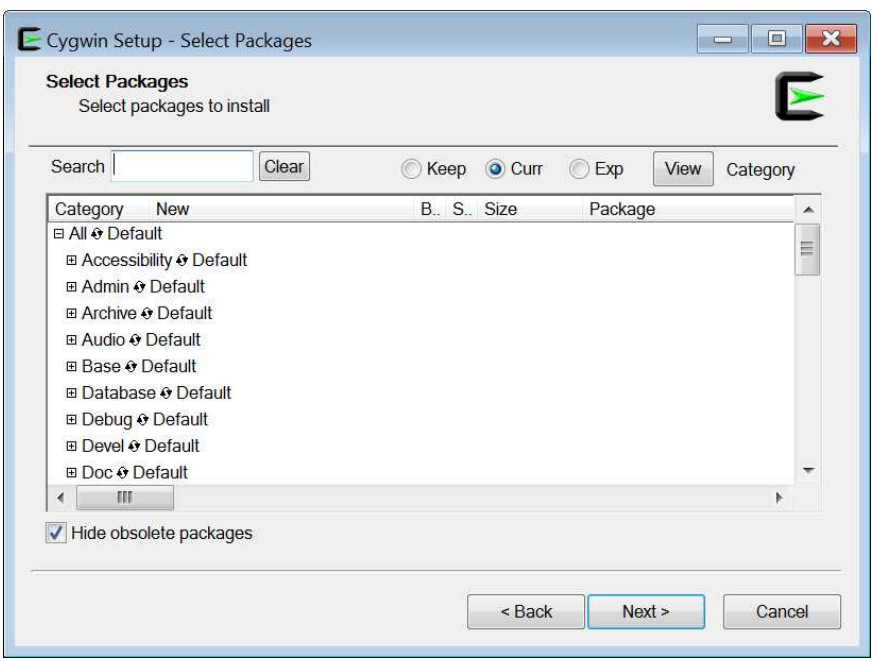


Figure 3–6 Choosing packages to install

Click the ⊞ next to Devel, to expose the developer packages and do the following. Scroll down until you see the “gcc-g++: GNU Compiler Collection (C++)” and click on the “Skip” setting for this package to select it. You may probably need to click on the “Skip” setting two or three times, until you select the most recent version. Scroll down and click the “Skip” setting for two more packages: for the “gdb: The GNU debugger” and for the “make: The GNU version of the ‘make’ utility” as shown in **Figure 3–7.**

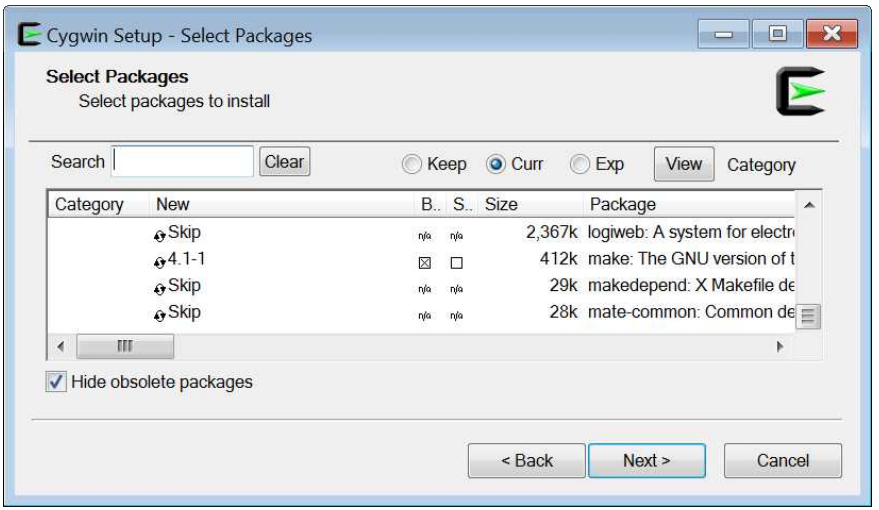


Figure 3–7 Choosing packages to install

With all required packages selected, you are finally ready. Click on the “Next” button. The next screen resolves dependencies. It recommends you to install some more required packages to satisfy dependencies. Leave the field “Select required packages” checked as shown in **Figure 3–8**.

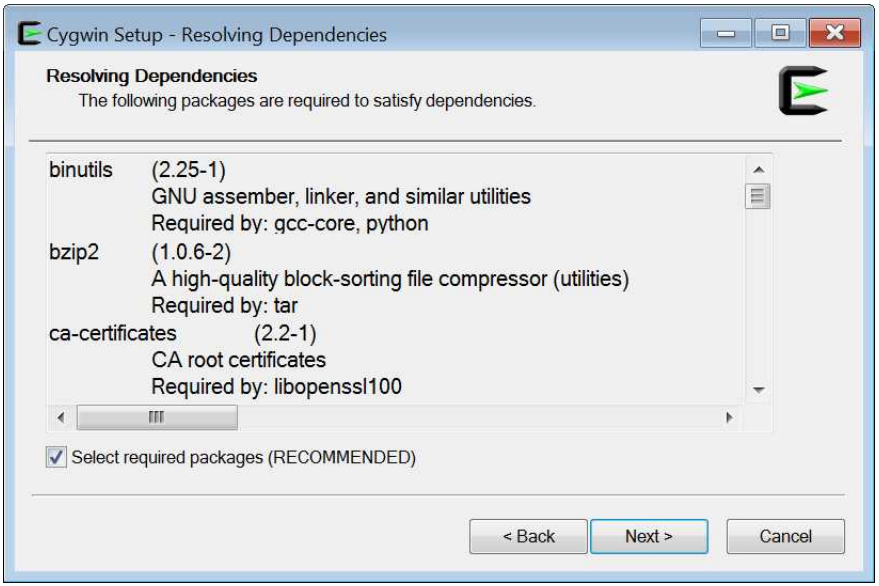


Figure 3–8 Resolving dependencies

Click on the "Next" button to begin the installation. When it finishes, click on the "Finish button" and an "Installation Complete" dialog appears; click on the OK button.

3.4 NetBeans IDE

NetBeans is an Integrated Development Environment (IDE) that provides a great set of tools for C++ and lets you easily write C++ programs. NetBeans is much more than a text editor. It can indent lines, match words and brackets, and highlight source code that is written incorrectly. It also provides automatic code, which means that as you type, it displays a list of possible completions.

The IDE also provides hints to help you analyze your code and find any potential problems. It even suggests some simple solutions to fix those problems.

You can use the NetBeans IDE not only to write but also to execute your programs directly from the IDE.

NetBeans IDE can be installed on all operating systems that support Java, from Windows to Linux to Mac OS X systems. NetBeans is free of charge and open source, and it has a large community of users and developers all around the world.

3.5 How to Set Up NetBeans IDE

In order to install NetBeans IDE, you must download it, free of charge, from the following address:

https://netbeans.org/downloads/index.html

On the NetBeans web page, there is a dropdown list that lets you choose the required platform (Windows, Linux, or Mac OS X). Choose the platform that matches yours. This book shows the process of installing NetBeans IDE on a Windows platform.

From all the "download bundles" available on netbeans.org, select and download the one called "All." When the download is complete, run the corresponding installer.

If you receive the message shown in **Figure 3–9**, it means that the Java SE Development Kit (JDK) was not found on your system. Please visit the displayed address to download and install the JDK before proceeding with the NetBeans IDE installation.

Figure 3–9 This message is displayed when the JDK is not found on your system.

On the first screen of the NetBeans IDE installer, you can keep and install all the proposed packs and runtimes or you can click on the "Customize" button and check only those proposed in **Figure 3–10**.

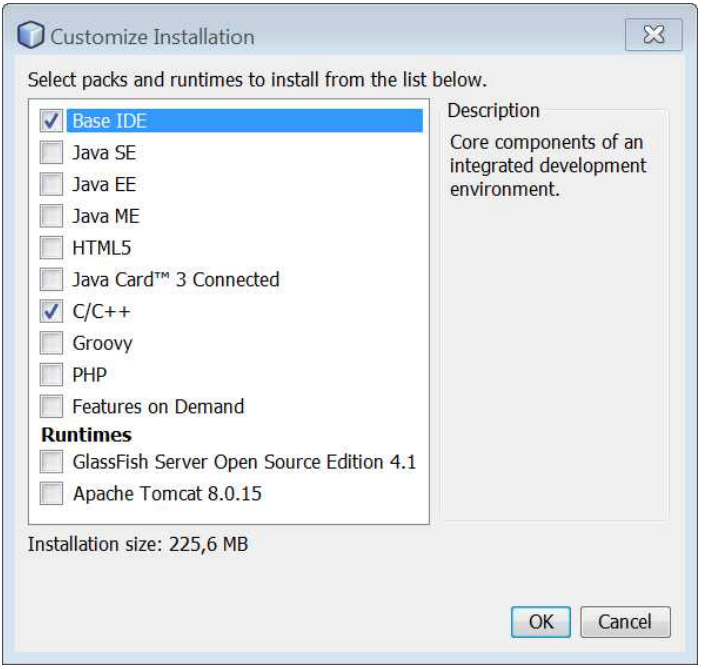


Figure 3–10 Selecting the packs and runtimes to install in the NetBeans IDE installer

Click on the "Next" button to go through the installation screens. When the license agreement shows up, you must read and accept the terms. The next screen prompts you to select the installation folder and the Java Environment. Leave the proposed folders as shown in **Figure 3–11**.

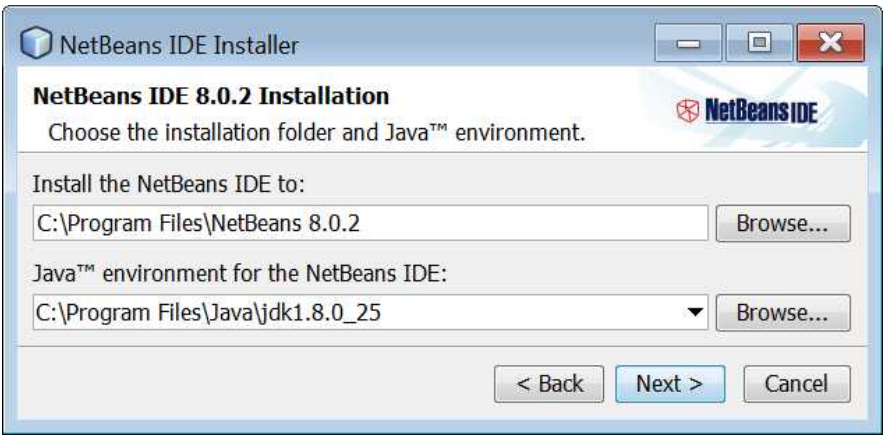


Figure 3–11 Selecting the installation folders in the NetBeans IDE installer

***Notice**: The installation folder and the Java environment proposed on your computer may differ from those in **Figure 3–11** depending on the versions of the NetBeans IDE or JDK that you have.*

On the next screen, leave the field "Check for Updates" checked, as shown in **Figure 3–12**. This option will automatically check for updates of installed plugins.

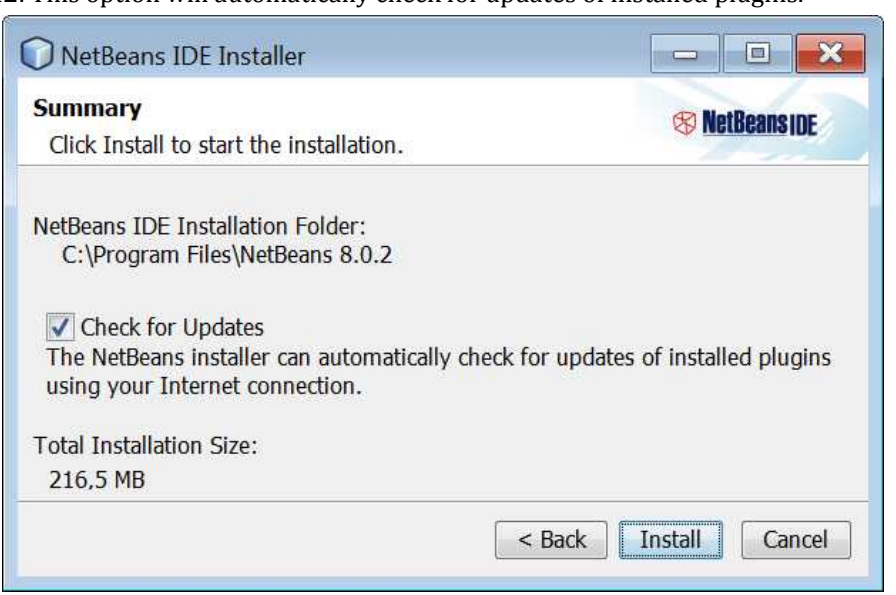


Figure 3–12 Selecting the box to check for updates

Click on the "Install" button.

When the installation process is complete, you can open the NetBeans IDE. The first screen that you will see is shown in **Figure 3–13**.

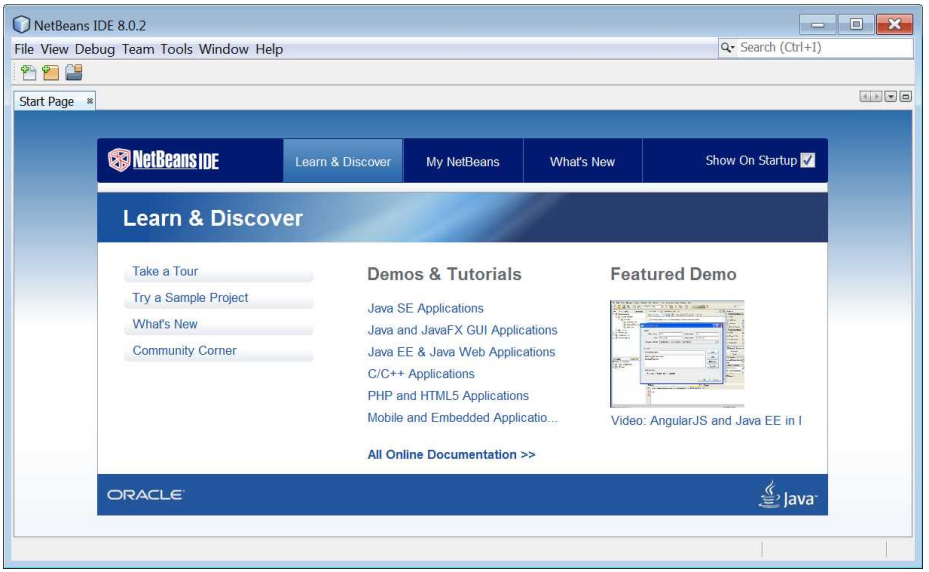


Figure 3–13 The NetBeans IDE start page

Now you need to configure NetBeans IDE. From the main menu select Tools→Options and in the popup window that appears, click on the C/C++ icon to open the "Options" dialog box of C/C++, as shown in **Figure 3–14**.

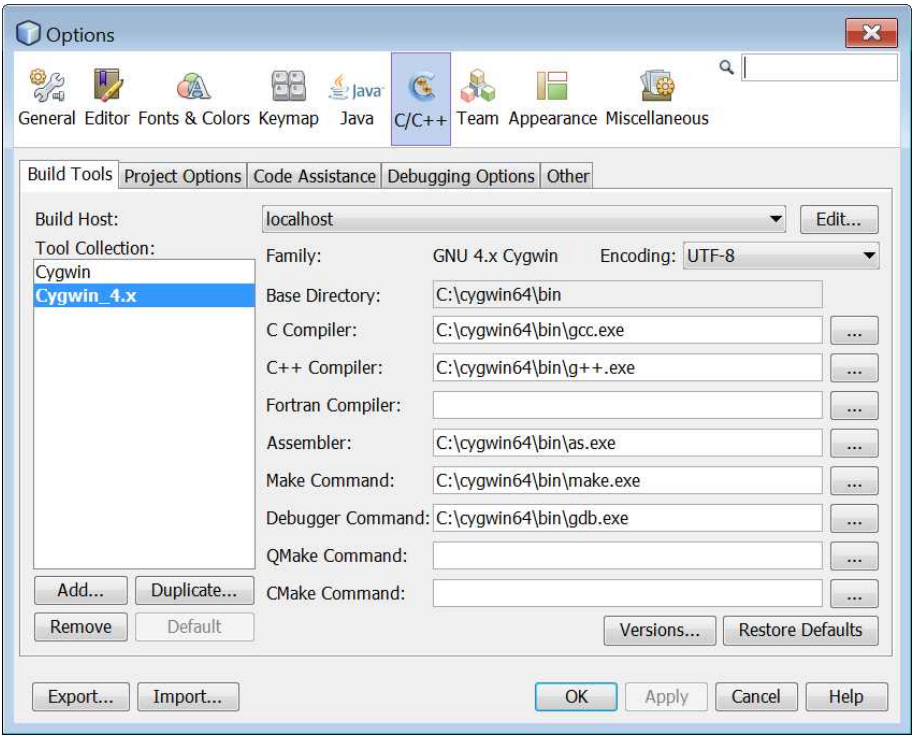


Figure 3–14 The "Options" dialog box of C/C++

On the "Code Assistance" tab, select the "C++ Compiler" tab, click on the "Add" button, and select the folder of the Boost C++ Library as shown in **Figure 3–15**.

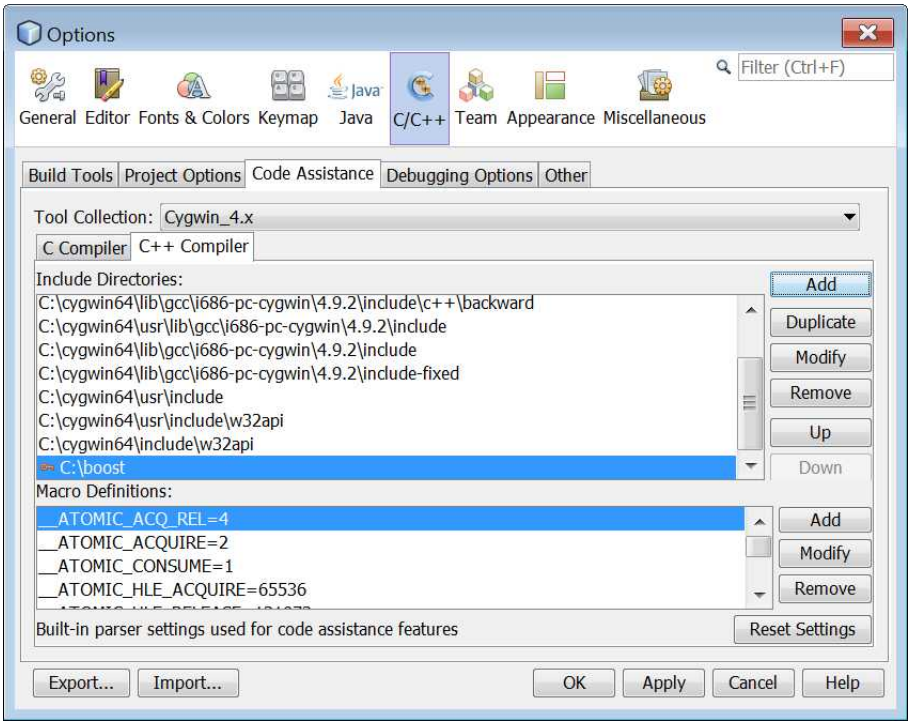


Figure 3–15 Adding the Boost C++ Library

One additional thing that this book does in order to save paper is to decrease the number of spaces per indent to two. You can also change this setting by first clicking on the Editor icon to open the Editor's "Options" dialog box, as shown in **Figure 3–16**.

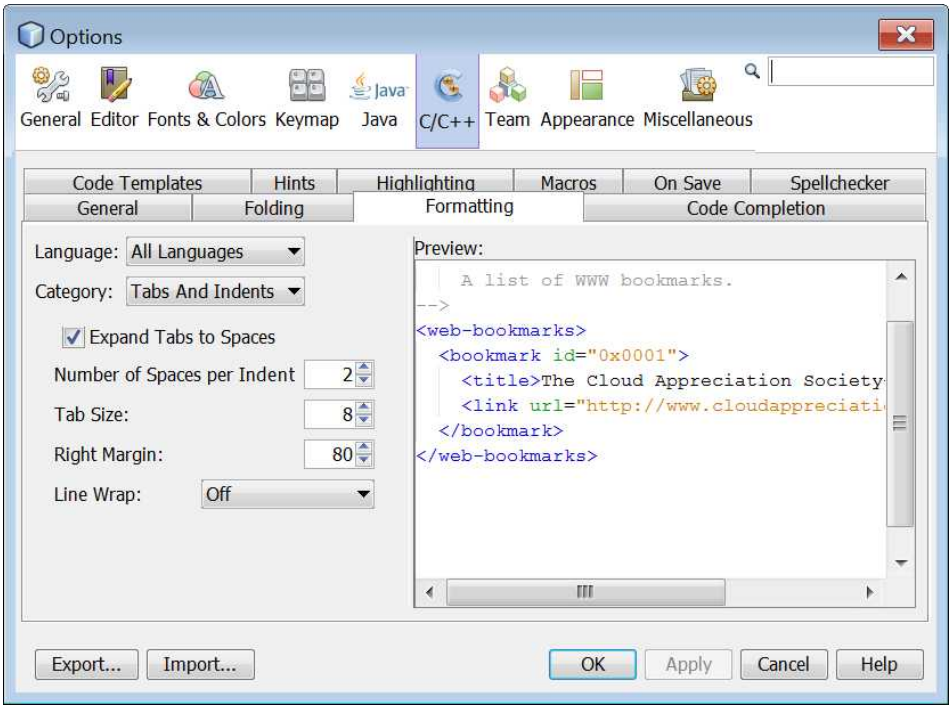


Figure 3–16 The Editor's "Options" dialog box

On the "Formatting" tab, set the field "Number of Spaces per Indent" to 2.

Click on the "OK" button.

NetBeans IDE has been configured properly! Now it's time to conquer the world of C++! In the upcoming chapters you will learn all about how to create C++ projects, how to write C++ programs, and so many tips and tricks useful in your first steps as a budding programmer!

Review Questions in “Introductory Knowledge”

Answer the following questions.

1. What is hardware?
2. List the five basic components of a typical computer system.
3. Which part of the computer actually executes the programs?
4. Which part of the computer holds the program and its data while the program is running?
5. Which part of the computer holds data for a long period of time, even when there is no power to the computer?
6. How do you call the device that collects data from the outside world and enters them into the computer?
7. List some examples of input devices.
8. How do you call the device that outputs data from the computer to the outside world?
9. List some examples of output devices.
10. What is software?
11. How many software categories are there, and what are their names?
12. A word processing program belongs to what category of software?
13. What is a compiler?
14. What is an interpreter?
15. What is meant by the term “machine language”?
16. What is source code?
17. What is C++?
18. What is the difference between a script and a program?
19. What are some of the possible uses of C++?
20. What is Boost?
21. What is NetBeans IDE?

Section 2

Getting Started with C++

Chapter 4
Introduction to Basic Algorithmic Concepts

4.1 What is an Algorithm?

An algorithm is a strictly defined finite sequence of well-defined statements (often called instructions or commands) that provides the solution to a problem or to a specific class of problems for any acceptable set of input values (if there are any inputs). In other words, an algorithm is a step-by-step procedure to solve a given problem. The term "finite" means that the algorithm should reach an end point and cannot run forever.

You can find algorithms everywhere in your real life, not just in computer science. For example, the process for preparing toast or a cup of tea can be expressed as an algorithm. Certain steps, in a certain order, must be followed in order to achieve your goal.

4.2 The Algorithm for Making a Cup of Tea

The following is an algorithm for making a cup of tea.

1. Put the teabag in a cup.
2. Fill the kettle with water.
3. Boil the water in the kettle.
4. Pour some of the boiled water into the cup.
5. Add milk to the cup.
6. Add sugar to the cup.
7. Stir the tea.
8. Drink the tea.

As you can see, there are certain steps that must be followed. These steps are in a specific order, even though some of the steps could be rearranged. For example, steps 5 and 6 can be reversed. You could add the sugar first, and the milk afterwards.

Please keep in mind that the order of some steps can probably be changed but you can't move them far away from where they should be. For example, you can't move step 3 ("Boil the water in the kettle.") to the end of the algorithm, because you will end up drinking a cup of iced tea (and not a warm one) which is totally different from your initial goal!

4.3 Properties of an Algorithm

An algorithm must satisfy the following properties:

- **Input**: The algorithm must have input values from a specified set.
- **Output**: The algorithm must produce the output values from a specified set of input values. The output values are the solution to a problem.

- **Finiteness**: For any input, the algorithm must terminate after a finite number of steps.
- **Definiteness**: All steps of the algorithm must be precisely defined.
- **Effectiveness**: It must be possible to perform each step of the algorithm correctly and in a finite amount of time. That is, its steps must be basic enough so that, for example, someone using a pencil and a paper could carry them out exactly, and in a finite amount of time. It is not enough that each step is definite (or precisely defined), but it must also be feasible.

4.4 Okay About Algorithms. But What is a Computer Program Anyway?

A computer program is nothing more than an algorithm that is written in a language that computers can understand, like C++, Java, C#, or Visual Basic.

A computer program cannot actually *make* you a cup of tea or cook your dinner, although an algorithm can guide you through the steps to do it yourself. However, programs can (for example) be used to calculate the average value of a set of numbers, or to find the maximum value among them. Artificial intelligence programs can even play chess or solve logic puzzles.

4.5 The Party of Three!

There are always three parties involved in an algorithm—the one that writes the algorithm, the one that executes it, and the one that uses or enjoys it.

Let's take an algorithm for preparing a meal, for example. Someone writes the algorithm (the author of the recipe book), someone executes it (probably your mother, who prepared the meal following the steps from the recipe book), and someone uses it (probably you, who enjoys the meal).

Now consider a real computer program. Let's take a video game, for example. Someone writes the algorithm in a computer language (the programmer), someone or something executes it (usually a laptop or a computer), and someone else uses it or plays with it (the user).

Be careful, because sometimes the terms "programmer" and "user" can be confusing. When you *write* a computer program, for that period of time you are "the programmer." However, when you *use* your own program, you are "the user."

4.6 The Three Main Stages Involved in Creating an Algorithm

Three main stages are involved in creating an algorithm: data input, data processing, and results output. The order is specific and cannot be changed.

Consider a computer program that finds the average value of three numbers. First, the program must prompt the user to enter the numbers (data input). Next, the program calculates the average value of the numbers (data processing). Finally, the program displays the result on the computer's screen (results output).

Let's take a look at these stages in more detail.

1. Prompt the user to enter a number.
2. Prompt the user to enter a second number.
3. Prompt the user to enter a third number.

Input

4. Calculate the sum of the three numbers.
5. Divide the sum by 3.

Processing

6. Display the result on the screen.

Output

In some rare situations, the input stage may be absent and the computer program may be composed of only two stages. For example, consider a computer program that is written to calculate the following sum.

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$$

In this example, the user enters no values at all because the computer knows exactly what to do. It must calculate the sum of the numbers 1 to 10.

Now let's take a look at the same example, slightly altered. Consider a computer program that is written to calculate the following sum.

$$1 + 2 + 3 + \cdots + N$$

Of course, this sum is not the same as the previous one. In this example, the user needs to input some data. The computer cannot decide by itself about the exact value of number N. This value must be entered by the user. Once the user enters a value for N, the computer can proceed to calculate the result. For example, if the user enters the number 5, the computer can then find the result of 1 + 2 + 3 + 4 + 5.

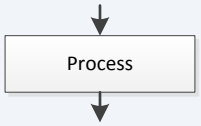
4.7 Flowcharts

A flowchart is a graphical method of presenting an algorithm, usually on paper. It is the visual representation of the algorithm's flow of execution. In other words, it visually represents how the flow of execution proceeds from one statement to the next until the end of the algorithm is reached. A flowchart cannot be entered directly into a computer as is. It must first be converted into a programming language such as C++.

The basic symbols that flowcharts use are shown in **Table 4–1**.

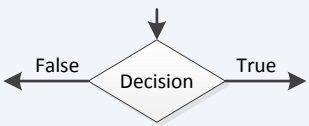
Table 4-1 Flowchart Symbols and Their Functions

	**Start/End**: Represents the beginning or the end of an algorithm. The Start symbol has one exit and the End symbol has one entrance.
	**Arrow**: Shows the flow of execution. An arrow coming from one symbol and ending at another symbol shows that control passes to the symbol that the arrow is pointing to. Arrows are always drawn as straight lines going up and down or sideways (never at an angle).



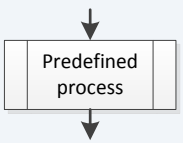
Process: Represents a process or mathematical (formula) calculation. The Process symbol has one entrance and one exit.

Data Input/Output: Represents the data input or the results output. In most cases, data comes from a keyboard and results are displayed on a screen. The Data input/output symbol has one entrance and one exit.



Decision: Indicates the point at which a decision is made. Based on a given condition (which can be true or false), the algorithm will follow either the right or the left path. The Decision symbol has one entrance and two (and always only two) exits.

Off-page connectors: Show continuation of a flowchart onto another page. They are used to connect segments on multiple pages when a flowchart gets too big to fit onto one sheet of paper. The outgoing off-page connector symbol has one entrance and the incoming off-page connector symbol has one exit.



Subprogram (predefined process): Depicts subprograms that are formally defined elsewhere, such as in a separate flowchart. The Predefined process symbol has one entrance and one exit.

An example of a flowchart is shown in **Figure 4–1.** The algorithm prompts the user to enter three numbers and then calculates their average value and displays it on the computer screen.

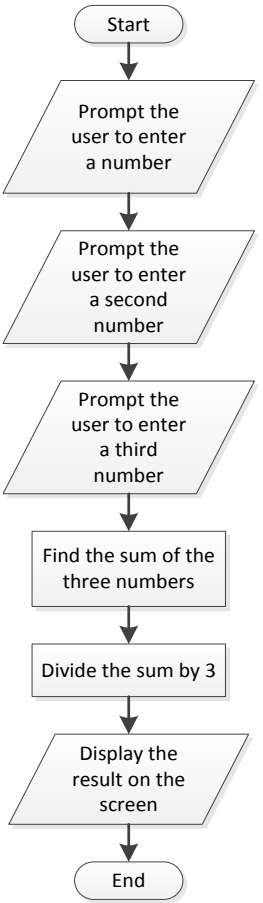


Figure 4–1 Flowchart for an algorithm that calculates and displays the average of three numbers

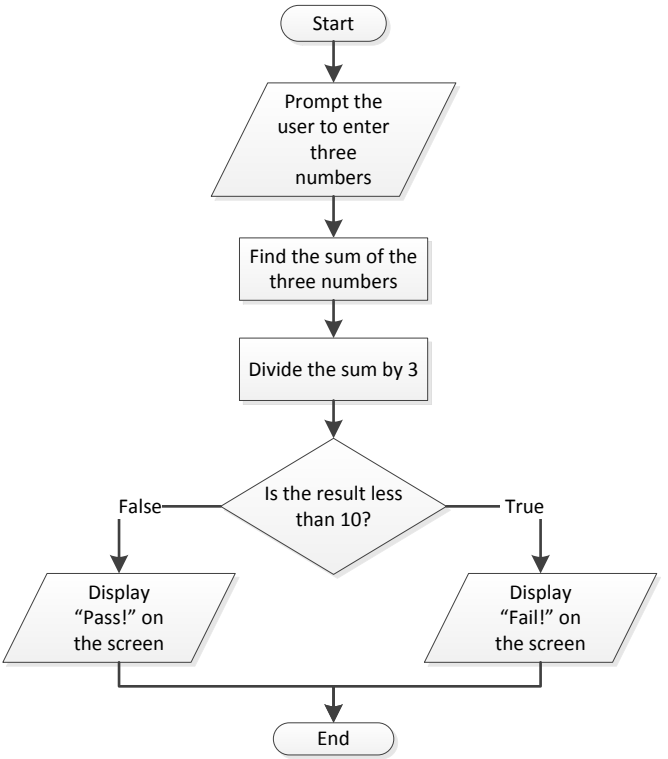
Remember! *A flowchart always begins and ends with a Start/End symbol!*

Exercise 4.7-1 *Finding the Average Value of Three Numbers*

Design an algorithm that calculates the average value of three numbers. Whenever the average value is below 10, a message "Fail!" should be displayed. Otherwise, if the average value is 10 or above, a message "Pass!" should be displayed.

Solution

In this problem, two different messages must be displayed, but only one can appear each time the algorithm is executed; the wording of the message depends on the average value. The flowchart for the algorithm is presented next.



Notice: *To save paper, you can prompt the user to enter all three numbers using one single oblique parallelogram.*

Remember! *A Decision symbol always has one entrance and two exit paths!*

Of course it is very soon for you to start creating your own algorithms. This particular exercise is quite simple and is presented in this chapter as an exception, just for demonstration purposes. You need to learn more before you start creating your own algorithms or even C++ programs. Just be patient! In a few chapters the big moment will come!

4.8 What are "Reserved Words"?

In a computer language, "reserved words" are all those words that have a strictly predefined meaning—they are reserved for special use and cannot be used for any other purpose. For example, the words `Start`, `End`, `Read`, and `Write` in flowcharts have a predefined meaning. They are used to represent the beginning, the end, the data input, and the results output, respectively.

Reserved words exist in all high-level computer languages as well. In C++, there are many reserved words such as `if`, `while`, `else`, and `for`. Their meanings are predefined, so these words cannot be used for any other purposes.

Notice*: Reserved words in a computer language are often called keywords.*

4.9 What is the Difference Between a Statement and a Command?

There is a big discussion on the Internet about whether there is, or is not, any difference between a statement and a command. Some people prefer to use the term "statement," and some others the term "command." For a novice programmer, there is no difference; both are instructions to the computer!

4.10 What is Structured Programming?

Structured programming is a software development method that uses modularization and structured design. This means that large programs are broken down into smaller modules and each individual module uses structured code, which means that the statements are organized in a specific manner that minimizes errors and misinterpretation. As its name suggests, structured programming is done in a structured programming language and C++ is one such language.

The structured programming concept was formalized in 1966 by Corrado Böhm[1] and Giuseppe Jacopini[2]. They demonstrated theoretical computer program design using sequences, decisions, and iterations.

4.11 The Three Fundamental Control Structures

There are three fundamental control structures in structured programming.

- **Sequence Control Structure**: This refers to the line-by-line execution, in which statements are executed sequentially, in the same order in which they appear in the program. They might, for example, carry out a series of read or write operations, arithmetic operations, or assignments to variables.
- **Decision Control Structure**: Depending on whether a condition is true or false, the decision control structure may skip the execution of an entire block of statements or even execute one block of statements instead of another.
- **Loop Control Structure**: This is a control structure that allows the execution of a block of statements multiple times until a specified condition is met.

If you didn't quite understand the deeper meaning of these three control structures, don't worry, because upcoming chapters will analyze them very thoroughly. Patience is a virtue. All you have to do for now is wait!

---

[1] Corrado Böhm (1923–) is a computer scientist known especially for his contribution to the theory of structured programming, and for the implementation of functional programming languages.

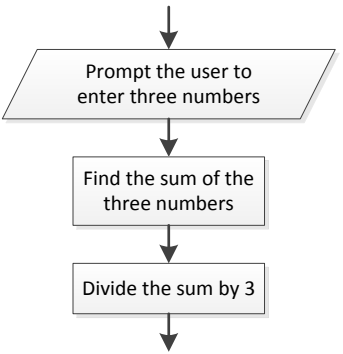
[2] Giuseppe Jacopini (1936–2001) was a computer scientist. His most influential contribution is the theorem about structured programming, published along with Corrado Böhm in 1966, under the title "Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules."
(Bohm, C., and Jacopini, G., "Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules," *Communications of the ACM* 9(5), (May, 1966), 366–371.)

Exercise 4.11-1 *Understanding Control Structures Using Flowcharts*

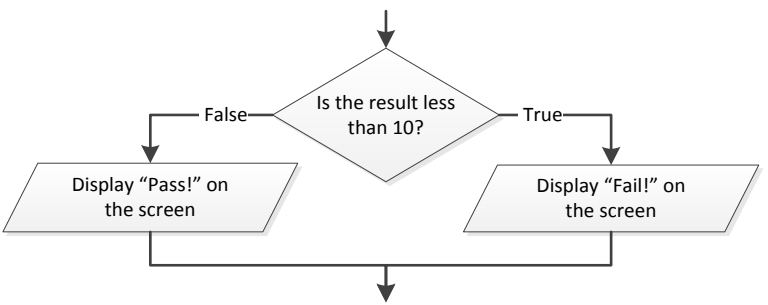
Using flowcharts, give an example for each type of control structure.

Solution

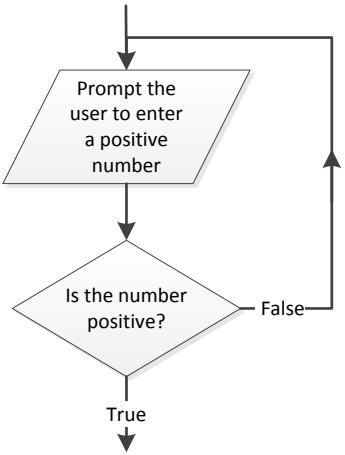
Example of a Sequence Control Structure



Example of a Decision Control Structure



Example of a Loop Control Structure



4.12 Your First C++ Program

Converting a flowchart to a computer language such as C++ results in a C++ program. A C++ program is nothing more than a text file including C++ statements. C++ programs can even be written in your text editor application! Keep in mind, though, that using NetBeans IDE to write C++ programs is a much better solution due to all of its included features that can make your life easier.

A C++ program must always contain a `main` function, as shown here.

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

int main() {

  // C++ code goes here

  return 0;
}
```

A C++ source code is saved on your hard disk with the default .cpp file extension.

Here is a very simple C++ program that displays just three messages on the screen.

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

int main() {

  cout << "Hello World!" << endl;
  cout << "Hallo Welt!" << endl;
  cout << "The End";
```

```
    return 0;
}
```

***Notice**: Please note that C++ requires that all statements be terminated with a semicolon.*

4.13 What is the Difference Between Syntax Errors and Logic Errors?

When programmers write code in a high-level language there are two types of errors that they might make: syntax errors and logic errors.

Syntax errors are mistakes such as misspelled keywords, a missing punctuation character, a missing bracket, or a missing closing parenthesis. Fortunately, NetBeans IDE detects these errors as you type and displays a red exclamation mark in front of the erroneous line. If you try to execute a C++ program that includes syntax errors, you will get error messages on your screen and the program won't be executed. You must correct all the errors and then try to execute the program again.

Logic errors are those errors that prevent your program from doing what you expected it to do. With logic errors you get no warning at all. Your code may compile and run but the result is not the expected one. Logic errors are the most difficult errors to detect. You must revisit your program thoroughly to determine where your error is. For example, consider a C++ program that prompts the user to enter three numbers, and then calculates and displays their average value. The programmer, however, made a typographical error; one of his or her statements divides the sum of the three numbers by 5, and not by 3 as it should. Of course the C++ program is executed as usual, without any error messages, prompting the user to enter three numbers and displaying a result, but obviously not the correct one! It is the programmer who has to find and correct the erroneously written C++ statement, not the computer or the compiler!

4.14 Commenting Your Code

When you write a small and easy program, anyone can understand how it works just by reading it line-by-line. However, long programs are difficult to understand, sometimes even by the same person who wrote them. Comments are extra information that can be included in a program to make it easier to read and understand. You can add explanations and other pieces of information, including:

- who wrote the program
- when the program was created or last modified
- what the program does
- how the program works

However, you should not over-comment. There is no need to explain every line of your program. Add comments only when a specific portion of your program is hard to follow.

In C++, you can add comments using one of the following methods:

- double slashes //……
- slash–asterisk, asterisk–slash delimiters /* …… */

The following program demonstrates how to use both types of commenting. Usually double slashes (//) are used for commenting one single line, whereas the slash–asterisk, asterisk–slash delimiters /* …… */ are used for commenting multiple lines at once.

```
#include <iostream>
using namespace std;
/*
  Created By Bouras Aristides
  Date created: 12/25/2003
  Date modified: 04/03/2008
  Description: This program displays some messages on the screen
*/

int main() {

  cout << "Hello Zeus!" << endl;  //display a message on the screen

  //display a second message on the screen
  cout << "Hello Hera!" << endl;

  /* display a third message on screen */  cout << "Γεια σας" << endl;

  //This is a comment        cout << "The End";

  return 0;
}
```

As you can see in the preceding program, you can add comments above a statement or at the end of it, but not in front of it. Look at the last statement, which is supposed to display the message "The End." This statement is never executed because it is considered part of the comment.

***Notice**: If you add comments using the delimiters /* …… */ in front of a statement, the statement is still executed. In the preceding example, the Greek message "Γεια σας", even though it is written next to some comments, is still executed. It is advisable, however, not to follow this writing style because it can make your code difficult to read.*

4.15 User-Friendly Programs

What is a "user-friendly" program? It's the one that the user considers a friend instead of an enemy, the one that can be used easily by a novice user.

If you want to write user-friendly programs you have to put yourself in the shoes of the user. Users want the computer to do their job their way, with a minimum of

effort. Hidden menus, imprecise labels and directions, and misleading error messages are all things that can make a program user-unfriendly!

The law that best defines user-friendly designs is the Law of Least Astonishment: "*The program should act in a way that least astonishes the user*." This law is also commonly referred to as the Principle of Least Astonishment (POLA).

4.16 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. The process for preparing a meal is actually an algorithm.
2. Algorithms are used only in computer science.
3. An algorithm can run forever.
4. In an algorithm, you can relocate a step in any position you wish.
5. An algorithm should produce the correct output values for just one set of input values.
6. Computers can play chess.
7. An algorithm can always become a computer program.
8. Programming is the process of creating a computer program.
9. There are always three parties involved in a computer program: the programmer, the computer, and the user.
10. The programmer and the user can sometimes be the same person.
11. It is possible for a computer program to output no results.
12. A flowchart is a computer program.
13. A flowchart is composed of a set of geometric shapes.
14. A flowchart is a method used to represent an algorithm.
15. You can design a flowchart without using any Start/End symbols.
16. You can design a flowchart without using any Process symbols.
17. You can design a flowchart without using any Data input/output symbols.
18. A flowchart should always include at least one Decision symbol.
19. In a flowchart, a Decision symbol can have one, two, or three exit paths, depending on the given problem.
20. Reserved words are all those words that have a strictly predefined meaning.
21. Structured programming includes structured design.
22. C++ is a structured computer language.
23. The basic principle of structured programming is that it includes only four fundamental control structures.
24. One statement, written ten times, is considered a loop control structure.
25. Decision control structure refers to the line-by-line execution.
26. A misspelled keyword is considered a logic error.
27. A C++ program can be executed even though it contains logic errors.

28. A missing semicolon at the end of a statement is considered a syntax error.
29. An exclamation mark at the end of a statement cannot prevent the whole C++ program from being executed.
30. One of the advantages of structured programming is that no errors are made while writing a computer program.
31. Logic errors are caught during compilation.
32. Syntax errors are the most difficult errors to detect.
33. A program that calculates the area of a triangle but outputs the wrong results contains logic errors.
34. When a program includes no output statements, it contains syntax errors.
35. A program should always contain comments.
36. If you add comments to a program, the computer can more easily understand it.
37. You can add comments anywhere in a program.
38. Comments are not visible to the users of a program.
39. A program is called user-friendly if it can be used easily by a novice user.
40. The acronym POLA stands for "Principle of Least Amusement."

4.17 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. An algorithm is a strictly defined finite sequence of well-defined statements that provides the solution to
    a. a problem.
    b. a specific class of problems.
    c. both of the above
2. Which of the following is **not** a property that an algorithm should satisfy?
    a. effectiveness
    b. fittingness
    c. definiteness
    d. input
3. A computer program is
    a. an algorithm.
    b. a sequence of instructions.
    c. both of the above
    d. none of the above
4. When someone prepares a meal, he or she is the
    a. "programmer."
    b. "user."
    c. none of the above

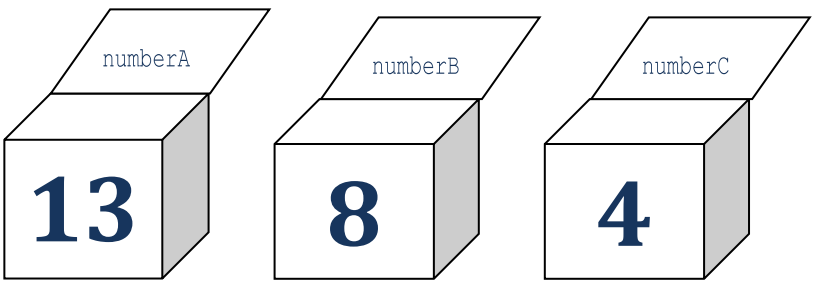
5. Which of the following does **not** belong in the three main stages involved in creating an algorithm?
    a. data production
    b. data input
    c. data output
    d. data processing
6. A flowchart can be
    a. presented on a piece of paper.
    b. entered directly into a computer as is.
    c. both of the above
7. A rectangle in a flowchart represents
    a. input/output.
    b. a processing operation.
    c. a decision.
    d. none of the above
8. Which of the following is/are control structures?
    a. a decision
    b. a sequence
    c. a loop
    d. All of the above are control structures.
9. Which of the following C++ statements contains a syntax error?
    a. `cout << "Hello Poseidon"`
    b. `cout << "It's me! I contain a syntax error!!!" << endl;`
    c. `cout << "Hello Athena" << endl;`
    d. none of the above
10. Which of the following `cout` statements is actually executed?
    a. `//cout << "Hello Apollo" << endl;`
    b. `/* cout << "Hello Artemis" << endl; */`
    c. `// This is executed // cout << "Hello Ares" << endl;`
    d. `/* This is executed */ cout << "Hello Aphrodite" << endl;`
    e. none of the above

Chapter 5
Variables and Constants

5.1 What is a Variable?

In computer science, a variable is a location in the computer's main memory (RAM) where you can store a value and change it as the program executes.

Picture a variable as a transparent box in which you can insert and hold one thing at a time. Because the box is transparent, you can also examine what it contains. Moreover, if you have two or more boxes you can give each box a unique name. For example, you could have three boxes, each containing a different number, and you could name the boxes `numberA`, `numberB`, and `numberC`.

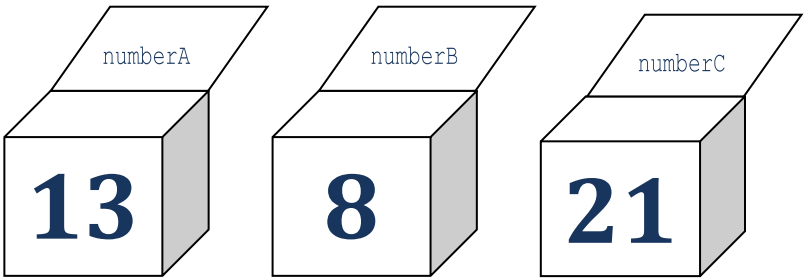


The boxes named `numberA`, `numberB` and `numberC` in the example contain the numbers 13, 8, and 4, respectively. Of course, you can examine or even alter the contained value of each one of these boxes at any time.

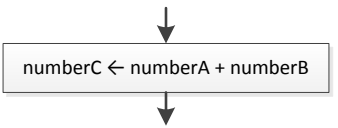
Now, let's say that someone asks you to find the sum of the values of the first two boxes and then store the result in the last box. The steps you must follow are:

1. Take a look at the first two boxes and examine the contained values.
2. Use your CPU (this is your brain) to calculate the sum (the result).
3. Insert the result in the last box. However, since each box can contain only one single value at a time, the value 4 is actually replaced by the number 21.

The boxes now look like this.



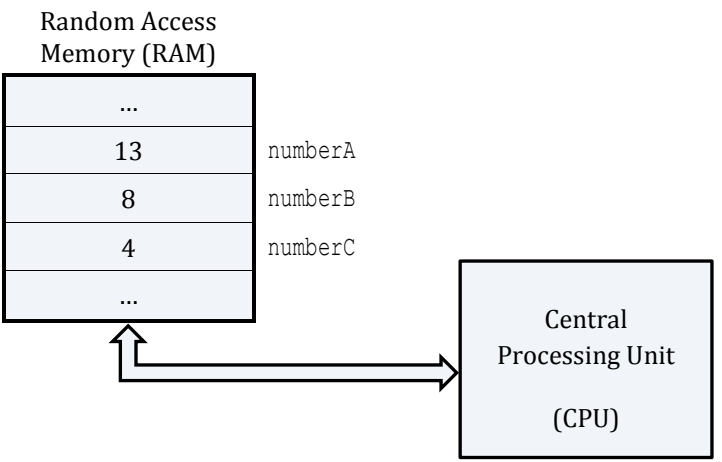
In a flowchart, the action of storing a value in a variable is represented by a left arrow



This action is usually expressed as "Assign a value, or the result of an expression, to a variable." The left arrow is called the "value assignment operator."

***Notice**: Please note that this arrow always points to the left. You are not allowed to use right arrows. Also, on the left side of the arrow only one single variable should exist.*

In real computer science, the three boxes are actually three individual regions in main memory (RAM), named `numberA`, `numberB` and `numberC`.



When a program instructs the CPU to execute the statement

```
numberC ← numberA + numberB
```

it follows the same three-step process as in the previous example.

1. The number 13 is transferred from the RAM's region named `numberA` to the CPU.

   The number 8 is transferred from the RAM's region named `numberB` to the CPU.

   (This is the first step, in which you examined the values contained in the first two boxes.)

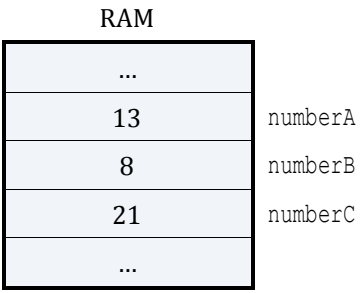
2. The CPU calculates the sum of 13 + 8.

   (This is the second step, in which you used your brain to calculate the sum, or result.)

3. The result, 21, is transferred from the CPU to the RAM's region named `numberC`, replacing the existing number 4.

   (This is the third step, in which you inserted the result in the last box.)

After execution, the RAM looks like this.



***Remember**! While a C++ program is running a variable can hold various values, but only one value at a time. When you assign a value to a variable, this value remains stored until you assign a new value. The old value is lost.*

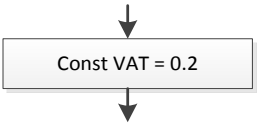
A variable is one of the most important elements in computer science because it helps you interact with data stored in the main memory (RAM). Soon, you will learn all about how to declare and use variables in C++.

5.2 What is a Constant?

Sometimes you may need to use a value that cannot change while the program is running. Such a value is called a "constant." In simple terms, you could say that a constant is a locked variable. This means that when a program begins to run and a value is assigned to the constant, nothing can change the value of the constant while the program is running. For example, in a financial program an interest rate can be declared as a constant.

A descriptive name for a constant can also improve the readability of your program and help you avoid errors. For example, let's say that you are using the value 3.14159265 (but not as a constant) at many points throughout your program. If you make an error when typing the number, this will produce the wrong results. But, if this value is given a name, any typographical error in the name is detected by C++, and you are notified with an error message.

In a flowchart, you can represent the action of setting a constant equal to a value with the equals sign, (=).



***Notice**: This book uses the reserved word* `Const` *to distinguish a constant from a variable.*

Consider an algorithm that lets the user enter the prices of three different products and then calculates and displays the 20% Value Added Tax (known as VAT) for each product. The flowchart in **Figure 5–1** shows this process when no constant is used.

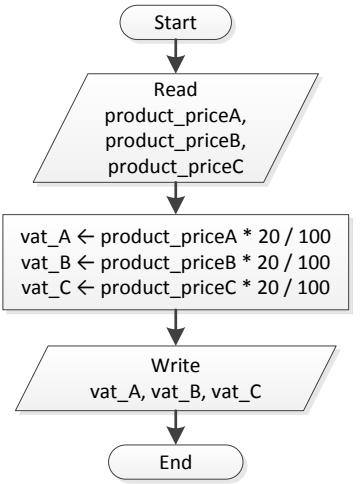


Figure 5–1 Calculating the 20% VAT for three products without the use of a constant

Even though this algorithm is absolutely correct, the problem is that the author used the 20% VAT (20/100) three times. If this were an actual computer program, the CPU would be forced to calculate the result of the division (20/100) three individual times.

***Notice**: Generally speaking, division and multiplication are time consuming operations that must be avoided when possible.*

A much better solution would be to use a variable, as shown in **Figure 5–2**. This reduces the number of division and multiplication operations and also decreases the potential for typographical errors.

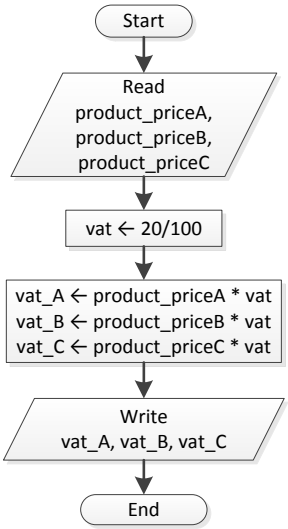


Figure 5–2 Calculating the 20% VAT for three products using a variable, `vat`

This time the division (20/100) is calculated only once, and then its result is used to calculate the VAT of each product.

But even now, the algorithm (which might later become a computer program) isn't perfect; `vat` is a variable and any programmer could accidentally change its value.

The ideal solution would be to change the variable `vat` to a constant `VAT`, as shown in **Figure 5–3**.

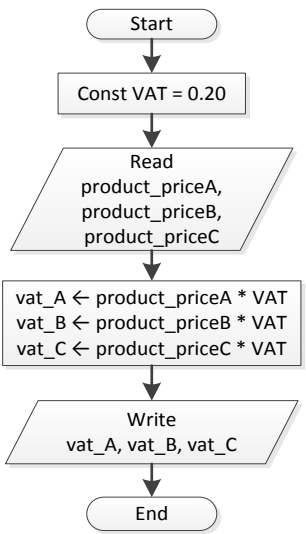


Figure 5–3 Calculating the 20% VAT for three products using a constant, `VAT`

***Notice**: Please note that when a constant is declared, the equals symbol (=) is used instead of the left arrow.*

This last solution is the best choice for many reasons.

- No one, including the programmer, can change the value of `VAT` just by accidentally writing a statement such as `VAT ← 0.60` in any position of the program.
- The potential for typographical errors is minimized.
- The number of multiplication and division operations is kept as low as possible.
- If one day the finance minister decides to increase the Value Added Tax from 20% to 22%, the programmer needs to change just one line of code!

5.3 How Many Types of Variables and Constants Exist in C++?

Many different types of variables and constants exist in all computer languages. The reason for this diversity is the different types of data each variable or constant can hold. Most of the time, variables and constants hold the following types of data.

- **Integers**: Integer values are positive or negative numbers without any fractional part, such as 5, 100, 135, -25, and -5123.
- **Reals**: Real values are positive or negative numbers that include a fractional part, such as 5.14, 7.23, 5.0, 3.14, and -23.78976. Real values are also known as "floats."
- **Booleans**[1]: Boolean variables or constants can hold only one of two values: `true` or `false`.
- **Characters**: Characters are alphanumeric values (always enclosed in single or double quotes) such as 'a', 'c', "Hello Zeus", "I am 25 years old", or "Peter Loves Jane For Ever". A sequence of characters is also known as a "string"!!! (Probably the word "string" makes you visualize something wearable, but unfortunately it's not. Please keep your dirty precious mind focused on computer science!)

5.4 Rules for Naming Variables in C++

Certain rules must be followed when assigning names to your variables.

- The name of a variable can contain only Latin characters (English uppercase or lowercase characters), numbers, and the underscore character (_). Examples of variable names are `firstName`, `last_name1`, and `age`.
- Variable names are case sensitive, which means there is a distinct difference between uppercase and lowercase characters. For example, `myVAR`, `myvar`, `MYVAR`, and `MyVar` are actually four different variables.

[1] George Boole (1815–1864) was an English mathematician, philosopher, and logician. He is best known as the architect of what is now called Boolean logic (Boolean algebra), the basis of the modern digital computer.

- No space characters are allowed. If a variable is described by more than one word, you can use the underscore character (_) between the words. For example, the variable name `student age` is wrong. Instead, you might use `student_age`, or even `studentAge`.
- A valid variable name can start with a letter, or an underscore. Numbers are allowed, but they cannot be used at the beginning of the variable name. For example the variable name `1student_name` is not properly written. Instead, you might use something like `student_name1` or `student1_name`.
- A variable name is usually chosen in a way that describes the meaning and the role of its contained data. For example, a variable that holds a temperature value might be named `temperature`, `temp`, or even `t`.

5.5 Rules for Naming Constants in C++

Certain rules must be followed when assigning names to your constants.

- The name of a constant can contain only Latin characters (English uppercase or lowercase characters), numbers, and the underscore character (_). Moreover, even though lowercase letters are permitted, it is advisable to use only uppercase letters. This helps you to visually distinguish constants from variables. Examples of constant names are `VAT` and `COMPUTER_NAME`.
- Constant names are case sensitive, which means there is a distinct difference between uppercase and lowercase characters. For example, the `myCONST`, `myconts`, `MYCONST`, and `MyConst` are actually four different constants.
- No space characters are allowed. If a constant is described by more than one word, you can use the underscore character (_) between the words. For example, the constant name `COMPUTER NAME` is wrong. Instead, you might use `COMPUTER_NAME`, or even `COMPUTERNAME`.
- A valid constant name can start with a letter, or an underscore. Numbers are allowed, but they cannot be used at the beginning of the constant name. For example, the constant name `1COMPUTER_NAME` is not properly written. Instead, you might use something like `COMPUTER_NAME1` or `COMPUTER1_NAME`.
- A constant name is usually chosen in a way that describes the meaning and the role of its contained data. For example, a constant that holds the Value Added Tax might be named `VAT`, or `VALUE_ADDED_TAX`.

5.6 What Does the Phrase "Declare a Variable" Mean?

Declaration is the process of reserving a portion in main memory (RAM) for storing the contents of a variable. In many high-level computer languages (including C++), the programmer must write a specific statement to reserve that portion in the RAM. In most cases, they even need to specify the variable type so that the compiler or the interpreter knows exactly how much space to reserve.

Here are some examples showing how variables are declared in different high-level computer languages.

Declaration Statement	High-level Computer Language
Dim sum as Integer	Visual Basic
int sum;	C++, C#, Java, and many more
sum: Integer;	Pascal, Delphi
var sum;	Javascript

5.7 How to Declare Variables in C++

C++ is a strongly typed programming language. This means that each variable must have a specific data type associated with it. For example, a variable can hold an integer, a real, or a character. In C++ some of the primitive data types are: bool, int, float, double, and char. Which one to use depends on the given problem! Moreover, several of the basic types can be modified using one or more of these type modifiers:

- unsigned
- short
- long

To be more specific:

- type **bool** can hold only two possible values: that is, `true` or `false`
- type **int** can hold an integer between -2^{31} and $+2^{31} - 1$
- type **unsigned int** can hold an integer between 0 and $+2^{32} - 1$
- type **short int** can hold an integer between -2^{15} and $+2^{15} - 1$
- type **unsigned short int** can hold an integer between 0 and $+2^{16} - 1$
- type **float** can hold a real of single precision
- type **double** can hold a real of double precision
- type **char** can hold a single character.

In many computer languages, there is one more variable type called “string”, which can hold a sequence of characters. These sequences of characters, or “strings” are usually enclosed in double or single quotes, such as “Hello Zeus”, “I am 25 years old”, and so on. C++ also supports strings, but keep in mind that a string in C++ is not a primitive data type. Without going into detail, a string in C++ is declared the same way as you declare a primitive data type such as int, byte, or double, but internally C++ stores and handles them in a quite different way.

To declare a variable, the general form of the C++ statement is

```
type name [ = value];
```

where

- *type* can be bool, int, short int, unsigned short int, string and so on
- *name* is a valid variable name
- *value* can be any valid initial value

Below are some examples of how to declare variables.

```
int number1;
bool found;
string first_name;
string student_name;
```

In C++ you can declare and directly assign an initial value to a variable. The next code fragment

```
int num = 5;
string name = "Hera";
char favorite_character = 'w';
```

is equivalent to

```
int num;
string name;
char favorite_character;

num = 5;
name = "Hera";
favorite_character = 'w';
```

Notice: *In C++, you can represent the action of assigning a value to a variable by using the equals (=) sign. This is equivalent to the left arrow in flowcharts.*

Notice: *Please note that in C++ you assign a value to a variable of type string using double quotes (" "), but you assign a value to a variable of type char using single quotes (' ').*

Last but not least, you can declare many variables of the same type on one line by separating them with commas.

```
double x, y, z;
long int w = 3, u = 2;
```

5.8 How to Declare Constants in C++

You can declare constants in C++ using the keyword `const`.

```
const type name = value;
```

The following examples show how to declare constants in C++.

```
const double VAT = 0.22;
const int NUMBER_OF_PLAYERS = 25;
```

```
const string FAVORITE_SONG = "We are the world";
const char FAVORITE_CHARACTER = 'w';
```

***Notice**: Please note that C++ requires that all statements be terminated with a semicolon (;).*

Once a constant is defined, its value cannot be changed while the program is running.

5.9 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A variable is a location in the computer's secondary storage device.
2. For a value assignment operator in a flowchart, you can use either a left or a right arrow.
3. A variable can change its content while the program executes.
4. A constant can change its content while the program executes.
5. The value 10.0 is an integer.
6. A Boolean variable can hold only one of two values.
7. The value “10.0” enclosed in double quotes is a real value.
8. In computer science, a string is something that you can wear.
9. The name of a variable can contain numbers.
10. A variable can change its name while the program executes.
11. The name of a variable cannot be a number.
12. The name of a constant must always be a descriptive one.
13. The name `student name` is not a valid variable name.
14. The name `student_name` is a valid constant name.
15. In C++, the name of a constant can contain uppercase and lowercase letters.
16. In C++, there is no need to declare a variable.
17. In C++, you must always declare at least one constant.

5.10 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. A variable is a place in
    a. a hard disk.
    b. a DVD disc.
    c. a USB flash drive.
    d. all of the above
    e. none of the above
2. A variable can hold
    a. one value at a time.
    b. many values at a time.

   c. all of the above
   d. none of the above
3. Using constants in a C++ program
   a. helps programmers to completely avoid typographical errors.
   b. helps programmers to avoid using division and multiplication.
   c. all of the above
   d. none of the above
4. Which of the following is an integer?
   a. 5.0
   b. -5
   c. “5”
   d. none of the above is an integer.
5. A Boolean variable can hold the value
   a. one.
   b. zero.
   c. `true`.
   d. “true”.
   e. none of the above
6. In C++, strings are
   a. enclosed in single quotes.
   b. enclosed in double quotes.
   c. both of the above
7. Which of the following is **not** a valid C++ variable?
   a. `city_name`
   b. `cityName`
   c. `city-name`
8. You can define a constant by using the keyword `const`. Once a constant is defined,
   a. it can never be changed.
   b. it can be changed using the keyword `const` again.
   c. Both of the above are correct.

5.11 Review Exercises

Complete the following exercises.

1. Match each element from the first column with one element from the second column.

Value	Data Type
1. “True”	a. Boolean
2. 123	b. Real
3. `False`	c. String
4. 10.0	d. Integer

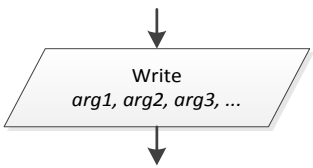
2. Match each element from the first column with one element from the second column.

Data	Data Type
1. The name of a person	a. Boolean
2. The age of a person	b. Real
3. The result of the division 5/2	c. Integer
4. Is it black or is it white?	d. String

Chapter 6
Handling Input and Output

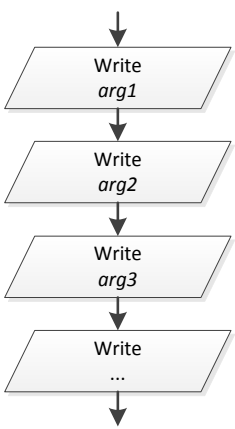
6.1 Which Statement Outputs Messages and Results on a User's Screen?

A flowchart uses the oblique parallelogram and the reserved word "Write" to display a message or the final results to the user's screen.



where *arg1, arg2*, and *arg3* can be variables, expressions, or even strings enclosed in double quotes.

The oblique parallelogram that you have just seen is equivalent to



In C++, you can achieve the same result by using the statement

```
cout << arg1 << arg2 << arg3 << …  ;
```

or the equivalent sequence of statements.

```
cout << arg1;
cout << arg2;
cout << arg3;
…
```

***Remember**! If you want to display a string on the screen, the string must be enclosed in double quotes.*

The following C++ program:

```
#include <iostream>
using namespace std;
int main() {
  int a;

  a = 5 + 6;
  cout << "The sum of 5 and 6 is " << a;
  return 0;
}
```

displays the message shown in **Figure 6–1**.

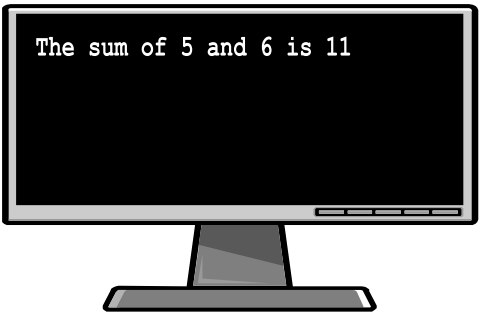


Figure 6–1 A string displayed on the screen

***Notice**: Please note the space inserted at the end of the first string in C++, just after the word "is." If you remove it, the number 11 will get too close to the last word and the output on the screen will be*

The sum of 5 and 6 is11

You can also calculate the result of a mathematical expression directly in a `cout` statement. The following statement:

```
#include <iostream>
using namespace std;
int main() {
  cout << "The sum of 5 and 6 is " << 5 + 6;
  return 0;
}
```

displays exactly the same message as the statements in **Figure 6–1**.

***Notice**: Please note that C++ requires that all statements be terminated with a semicolon.*

6.2 How to Output Special Characters

Look carefully at the following example:

```
#include <iostream>
using namespace std;
int main() {
  cout << "Good Morning";
  cout << "Good Evening";
  cout << "Good Night";
  return 0;
}
```

Although you may believe that these three messages are displayed one under the other, the actual output result is shown in **Figure 6–2**.

Figure 6–2 The output result displays on one line

In order to output a "line break" you must put the special sequence of characters `\n` after every sentence.

```
#include <iostream>
using namespace std;
int main() {
  cout << "Good Morning\n";
  cout << "Good Evening\n";
  cout << "Good Night\n";
  return 0;
}
```

or use the `endl` manipulator as follows

```
#include <iostream>
using namespace std;
int main() {
  cout << "Good Morning" << endl;
  cout << "Good Evening" << endl;
  cout << "Good Night" << endl;
  return 0;
}
```

The output result now appears in **Figure 6–3**.

Figure 6–3 The output result now displays line breaks

Keep in mind that the same result can also be accomplished with one single statement.

```
cout << "Good Morning\nGood Evening\nGood Night\n";
```

Another interesting sequence of characters is the `\t` which can be used to output a "tab stop." The tab character (`\t`) is useful for aligning output.

```
#include <iostream>
using namespace std;
int main() {
  cout << "Good Morning\t";
  cout << "Good Evening\n";
  cout << "Good Afternoon\t";
  cout << "Good Night";
  return 0;
}
```

The output result now appears in **Figure 6–4**.

Figure 6–4 The output result now displays tabs

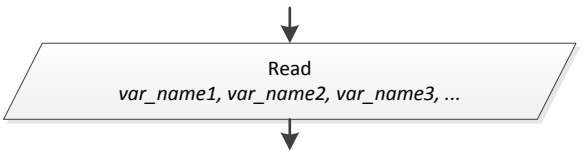
Of course, the same result can be accomplished with one single statement.

```
cout << "Good Morning\tGood Evening\nGood Afternoon\tGood Night";
```

6.3 Which Statement Lets the User Enter Data?

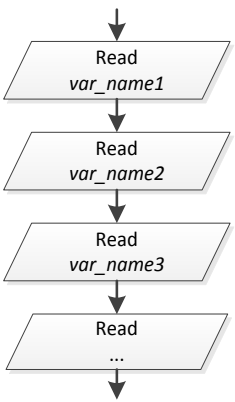
Do you recall the three main stages involved in creating an algorithm or a computer program? The first stage was the "data input" stage, in which the computer lets the user enter data such as numbers, their name, their address, or their year of birth.

A flowchart uses the oblique parallelogram and the reserved word "Read" to let a user enter his or her data.



where *var_name1*, *var_name2*, and *var_name3* must be variables only.

The oblique parallelogram that you have just seen is equivalent to



Notice: *When a* `Read` *statement is executed, the flow of execution is interrupted until the user has entered all the data. When data entry is complete, the flow of execution continues to the next statement. Usually data are entered from a keyboard.*

In C++, data input is accomplished using the following C++ statements

```
//Read a number or a string (without spaces or tabs) from the keyboard
cin >> var_name;

//Read a string that includes spaces and tabs from the keyboard
getline(cin, var_name_str);
```

where

- *var_name* can be a variable of any type.
- *var_name_str* can be any variable of type `string`.

***Notice**: Please note that* `cin` *considers space characters as terminating characters when reading a string from the keyboard. Thus, reading a string with* `cin` *means to always read a single word, not a phrase or an entire sentence. To read an entire line you can use the* `getline()` *function.*

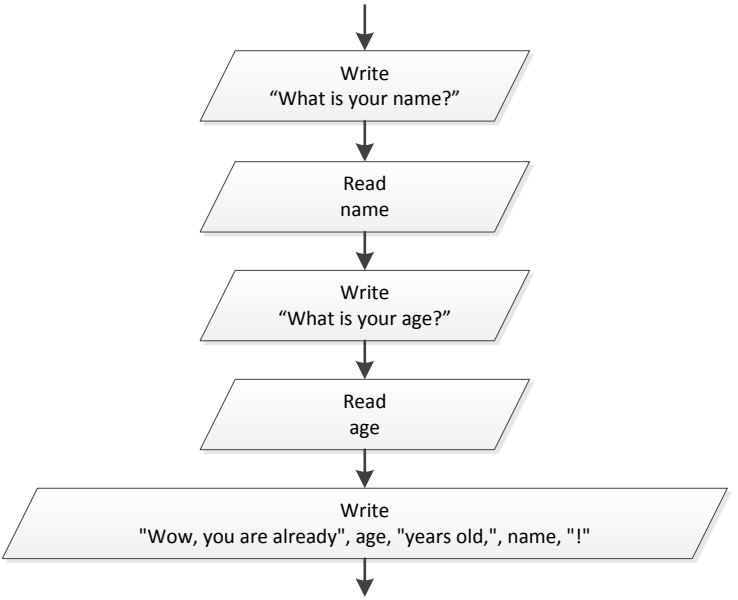
The statement `cin` can also be used to request more than one value. The following code fragment lets the user enter his or her name and age.

```
string name;
int age;

cin >> name >> age;

cout << "Wow, you are already" << age << "years old," << name << "!");
```

However, it could be even better if, before each data input, a "prompt" message is displayed. This makes the program more user-friendly. For example, look at the following program.

```
string name;
int age;

cout << "What is your name? ";
cin >> name;

cout << "What is your age? ";
cin >> age;

cout << "Wow, you are already" << age << "years old," << name << "!");
```

The corresponding flowchart fragment looks like this.



Now when a user executes your program he or she knows exactly what to enter.

***Notice**: In this book there is a subtle difference between the words “prompts” and “lets.” When the wording of an exercise says “Write a C++ program that **prompts** the user to enter…” this means that you must include a prompt message before every data input. On the other hand, when the wording of an exercise says “Write a C++ program that **lets** the user enter…” this means that you are not actually required to include a prompt message; that is, it is not wrong to include one but you don’t have to!*

6.4 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. In C++, the word `cin` is a reserved word.
2. The `cout` statement can be used to display a message or the content of a variable.
3. When the `cin` statement is executed, the flow of execution is interrupted until the user has entered a value.
4. One single statement, `cin`, can be used to enter multiple data values.
5. Before data input, a prompt message must always be displayed.

6.5 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. The statement `cout << "Hello"` displays
    a. the word `Hello` (without the double quotes).

   b. the word `"Hello"` (including the double quotes).
   c. the content of the variable `Hello`.
   d. none of the above

2. The statement `cout << "Hello\nHermes"` displays
   a. the message `Hello Hermes`.
   b. the word `Hello` in one line and the word `Hermes` in the next one.
   c. the message `HelloHermes`.
   d. the message `Hello\nHermes`.
   e. none of the above

3. The statement `cin >> data1_data2`
   a. lets the user enter a value and assigns it to variable `data1`. Variable `data2` remains empty.
   b. lets the user enter a value and assigns it to variable `data1_data2`.
   c. lets the user enter two values and assigns them to variables `data1` and `data2`.
   d. none of the above

Chapter 7
Operators

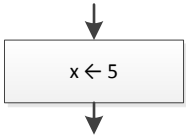
7.1 The Value Assignment Operator

The most commonly used operator in C++ is the value assignment operator (=). For example, the C++ statement

```
x = 5;
```

assigns a value of 5 to variable x.

As you read in Chapter 5, this is equivalent to the left arrow used in flowcharts.



Probably the left arrow used in a flowchart is more convenient and clearer than the value assignment operator (=) sign because it shows in a more graphical way that the value or the result of an expression on the right is assigned to a variable on the left.

Be careful! The (=) sign is not equivalent to the one used in mathematics. In mathematics, the expression `x = 5` is read as *"x is equal to 5."* However, in C++ the expression `x = 5` is read as *"assign the value 5 to x"* or *"set x equal to 5."* They look the same but they act differently!

In mathematics, the following two lines are equivalent:

```
x = y + z
y + z = x
```

The first one can be read as "x is equal to the sum of y and z" and the second one as "the sum of y and z is equal to x."

On the other hand, in C++, the following two statements are definitely **not** equivalent. In fact, the second one is considered wrong!

```
x = y + z;
y + z = x;
```

The first statement seems quite correct. It can be read as "*Set x equal to the sum of y and z*" or "*Assign the sum of y and z to x.*"

But what about the second one? Think! Is it possible to assign the value of `x` to `y + z`? The answer is obviously a big "NO!"

***Remember**! In C++, the variable on the left side of the (=) sign represents a region in main memory (RAM) where a value can be stored. Thus, on the left side only one single variable must exist! However, on the right side there can be a number, a variable, a string, or even a complex mathematical expression.*

In **Table 7–1** you can find some examples of value assignments.

Table 7-1 Examples of Value Assignments

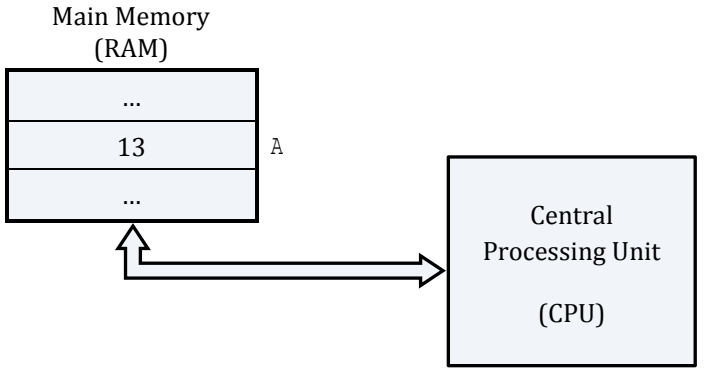
`a = 9`	Assign a value of 9 to variable `a`.
`b = c`	Assign the content of variable `c` to variable `b`.
`d = "Hello Zeus"`	Assign the string *Hello Zeus* to variable `d`.
`d = a + b`	Calculate the sum of the contents of variables `a` and `b` and assign the result to variable `d`.
`b = a + 1`	Calculate the sum of the content of variable `a` and 1 and assign the result to variable `b`. Please note that the content of variable `a` is not altered.
`a = a + 1`	Calculate the sum of the content of variable `a` and 1 and assign the result back to variable `a`. In other words, increase variable `a` by one.

Confused about the last one? Are you thinking about your math teacher right now? What would he/she say if you had written `a = a + 1` on the blackboard? Can you personally think of a number that is equal to the number itself plus one? Are you nuts? This means that 5 is equal to 6 and 10 is equal to 11!

Obviously, things are different in computer science. The statement `a = a + 1` is absolutely acceptable! It instructs the CPU to retrieve the value of variable `a` from main memory (RAM), to increase the value by one, and to assign the result back to variable `a`. The old value of variable `a` is replaced by the new one.

Still don't get it? Let's take a look at how the CPU and main memory (RAM) cooperate with each other in order to execute the statement `A ← A + 1` (this is the same as the statement `a = a + 1` in C++).

Let's say that there is a region in memory, named `A` and it contains the number 13



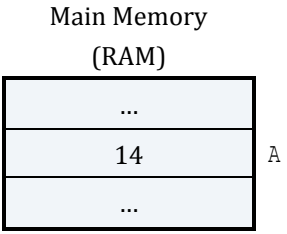
When a program instructs the CPU to execute the statement

```
A ← A + 1
```

the following procedure is executed:

- the number 13 is transferred from the RAM's region named `A` to the CPU;
- the CPU calculates the sum of 13 and 1; and
- the result, 14, is transferred from the CPU to the RAM's region named `A` replacing the existing number, 13.

After execution, the RAM looks like this.



7.2 Arithmetic Operators

Just like every high-level programming language, C++ supports almost all types of arithmetic operators.

Arithmetic Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder after integer division)

The first four operators are straightforward and need no further explanation.

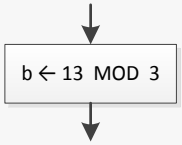
The modulus operator (%) returns the remainder of an integer division, which means that

```
a = 13 % 3
```

assigns a value of 1 to variable a.

Notice: *In C++, the modulus operator (%) cannot be used with floating-point numbers. For example, the operation 13.4 % 4 cannot be performed.*

Notice*: Please keep in mind that flowcharts are a loose method used to represent an algorithm. Although the use of the modulus (%) operator is allowed in flowcharts, this book uses the commonly accepted MOD operator instead! For example, the C++ statement* `b = 13 % 3` *is represented in a flowchart as*



In mathematics, as you already know, you are allowed to use parentheses as well as braces and brackets.

$$y = \frac{5}{2}\left\{3 + 2\left[4 + 7\left(7 - \frac{4}{3}\right)\right]\right\}$$

However, in C++ there is no such thing as braces and brackets. Parentheses are all you have; therefore, the same expression should be written using parentheses instead of braces or brackets.

```
y = 5 / 2 * (3 + 2 * (4 + 7 * (7 - 4 / 3)));
```

One other thing that is legal in mathematics but not in C++ is that you can skip the multiplication operator and write `3x`, meaning *"3 times x."* In C++, however, you must always use an asterisk anywhere a multiplication operation exists. This is one of the most common mistakes novice programmers make when they write mathematical expressions in C++.

7.3 What is the Precedence of Arithmetic Operators?

Arithmetic operators follow the same precedence rules as in mathematics, and these are: multiplication and division are performed first, addition and subtraction are performed afterwards.

Higher Precedence  Lower precedence	Arithmetic Operators
	*, /, %
	+, -

When multiplication and division exist in the same expression, and since both are of the same precedence, they are performed left to right (the same way as you read), which means that the expression

```
y = 6 / 3 * 2
```

is equivalent to $y = \frac{6}{3} \cdot 2$, and assigns a value of 4 to variable `y`, (division is performed before multiplication).

If you want, however, the multiplication to be performed before the division, you can use parentheses to change the precedence, that is:

```
y = 6 / (3 * 2)
```

which is equivalent to $y = \frac{6}{3 \cdot 2}$

Notice: *C++ programs can be written using your computer's text editor application, but keep in mind that it is not possible to write fractions in the form of* $\frac{6}{3}$ *or* $\frac{4x+5}{6}$*. Forget it! There is no equation editor in a text editor! All fractions must be written on one single line. For example,* $\frac{6}{3}$ *should be written as 6/3, and* $\frac{4x+5}{6}$ *should be written as (4*x+5)/6.*

The order of operations can be summarized as follows:

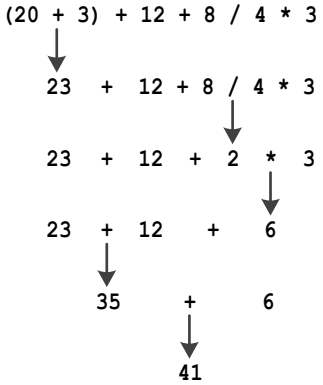
1. Any operations enclosed in parentheses are performed first.
2. Next, any multiplication and division operations are performed from left to right.
3. In the end, any addition and subtraction operations are performed from left to right.

So, in statement `y = (20 + 3) + 12 + 8 / 4 * 3;`

the operations are performed as follows:

1. 20 is added to 3, yielding a result of 23.
2. 8 is divided by 4, yielding a result of 2. This result is then multiplied by 3, yielding a result of 6.
3. 23 is added to 12, yielding a result of 35. This result is then added to 6, yielding a final result of 41.

Next is the same sequence of operations presented in a more graphical way.



7.4 Compound Assignment Operators

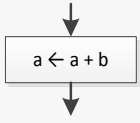
C++ offers a special set of operators known as compound assignment operators, which can help you write code faster.

Operator	Description	Example	Equivalent to
+=	Addition assignment	`a += b`	`a = a + b`
-=	Subtraction assignment	`a -= b`	`a = a - b`
*=	Multiplication assignment	`a *= b`	`a = a * b`
/=	Division assignment	`a /= b`	`a = a / b`
%=	Modulus assignment	`a %= b`	`a = a % b`

Looking at the *"Equivalent to"* column, it becomes clear that same result can be achieved by just using the classic assignment (=) operator. So the question that arises here is *why do these operators exist?*

The answer is simple: It's a matter of convenience. Once you start using them, your life finds a different meaning!

***Notice**: Please keep in mind that flowcharts are a loose method to represent an algorithm. Although the use of compound assignment operators is allowed in flowcharts, this book uses only the commonly accepted operators shown in the "Equivalent to" column. For example, the C++ statement* `a += b` *is represented in a flowchart as*



Exercise 7.4-1 *Which C++ Statements are Syntactically Correct?*

Which of the following C++ assignment statements are syntactically correct?

i. `a = -10;`
ii. `10 = b;`
iii. `a_b = a_b + 1;`
iv. `a = "COWS";`
v. `a = COWS;`
vi. `a + b = 40;`
vii. `a = 3 b;`
viii. `a = "true";`
ix. `a = true;`
x. `a %= 2;`
xi. `a += 1;`
xii. `a =* 2;`

Solution

i. **Correct**. It assigns the integer value -10 to variable `a`.
ii. **Wrong**. On the left side of the assignment operator, only variables can exist.
iii. **Correct**. It increases variable `a_b` by one.
iv. **Correct**. It assigns the string (the text) `COWS` to variable `a`.
v. **Correct**. It assigns the content of constant `COWS` to variable `a`.
vi. **Wrong**. On the left side of the assignment operator, only one variable can exist.
vii. **Wrong**. It should have been written as `a = 3 * b`.
viii. **Correct**. It assigns the string `true` to variable `a`.
ix. **Correct**. It assigns the value `true` to variable `a`.
x. **Correct**. This is equivalent to `a = a % 2`.
xi. **Correct**. This is equivalent to `a = a + 1`
xii. **Wrong**. It should have been written as `a *= 2` (which is equivalent to `a = a * 2`)

Exercise 7.4-2 *Finding Variable Types*

What is the type of each of the following variables?

i. `a = 15;`
ii. `width = "10 meters";`
iii. `b = "15";`
iv. `temp = 13.5;`
v. `b = true;`
vi. `b = "true";`

Solution

i. The value 15 belongs to the set of integers, thus the variable `a` is an integer.
ii. The "*10 meters*" is text, thus the `width` variable is a string.
iii. The "15" is text string, thus the `b` variable is a string.
iv. The value 13.5 belongs to the set of real numbers, thus the variable `temp` is real.
v. The value `true` is Boolean, thus variable `b` is a Boolean.
vi. The value `true` is a text, thus variable `b` is a string.

7.5 Incrementing/Decrementing Operators

Adding 1 to a number, or subtracting 1 from a number, are both so common in computer programs that C++ incorporates a special set of operators to do this. C++ supports two types of incrementing and decrementing operators:

- pre-incrementing/decrementing operators
- post-incrementing/decrementing operators

Pre-incrementing/decrementing operators are placed before the variable name. Post-incrementing/decrementing operators are placed after the variable name. These four types of operators are shown here.

Operator	Description	Example	Equivalent to
Pre-incrementing	Increment a variable by one	`++a;`	`a = a + 1`
Pre-decrementing	Decrement a variable by one	`--a;`	`a = a - 1`
Post-incrementing	Increment a variable by one	`a++;`	`a = a + 1`
Post-decrementing	Decrement a variable by one	`a--;`	`a = a - 1`

Once again, looking at the *"Equivalent to"* column, it becomes clear that you can achieve the same result by just using the classic assignment operator (=). However, it is much more sophisticated and productive to use these new operators. In the beginning you may hate them, but as the years pass you are definitely gonna love them!

Let's see an example with a pre-incrementing operator,

```
a = 5;
++a;                //This is equivalent to a = a + 1
b = a;

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

and another one with a post-incrementing operator.

```
a = 5;
a++;                //This is equivalent to a = a + 1
b = a;

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

In both examples a value of 6 is assigned to variable `b`! So, where is the catch? Are these two examples equivalent? The answer is "yes," but only in these two examples. In other cases the answer will likely be "no." There is a small difference between them.

Let's spot that difference! The rule is that a pre-increment/decrement operator performs the increment/decrement operation first and then delivers the new value.

A post-increment/decrement operator delivers the old value first and then performs the increment/decrement operation. Look carefully at the next two examples.

```
a = 5;
b = ++a;
cout << a << endl;          //Outputs: 6
cout << b << endl;          //Outputs: 6
```

and

```
a = 5;
b = a++;
cout << a << endl;          //Outputs: 6
cout << b << endl;          //Outputs: 5
```

In the first example, variable `a` is incremented by one and then its new value is assigned to variable `b`. In the end, both variables contain a value of 6.

In the second example, the value 5 of variable `a` is assigned to variable `b`, and then variable `a` is incremented by one. In the end, variable `a` contains a value of 6 but variable `b` contains a value of 5!

Notice: *The double slashes after the* `cout` *statements indicate that the text that follows is a comment; thus, it is never executed.*

7.6 String Operators

There are two operators that can be used to concatenate (join) strings.

Operator	Description	Example	Equivalent to
+	Concatenation	`a = "Hi" + " there";`	
+=	Concatenation assignment	`a += "Hello";`	`a = a + "Hello";`

Notice*: Joining two separate strings into a single one is called "concatenation."*

The following example displays "What's up, dude?"

```
#include <iostream>
using namespace std;
int main() {
  string a, b, c;

  a = "What's ";
  b = "up, ";
  c = a + b;
  c += "dude?";

  cout << c;
  return 0;
}
```

Exercise 7.6-1 *Concatenating Names*

Write a C++ program that prompts the user to enter his or her first and last name (assigned to two different variables) and then joins them together in a single string (concatenation).

Solution

The C++ program is shown here.

```
#include <iostream>
using namespace std;
int main() {
  string first_name, last_name, full_name;

  cout << "Enter first name: ";
  cin >> first_name;
  cout << "Enter last name: ";
  cin >> last_name;

  full_name = first_name + " " + last_name;
  cout << full_name;
  return 0;
}
```

***Notice**: Please note the extra space character added between the first and last name*

7.7 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. The statement `x = 5` can be read as *"Variable `x` is equal to 5."*
2. The value assignment operator assigns the result of an expression to a variable.
3. A string can be assigned to a variable only by using the `cin` statement.
4. The statement `5 = y` assigns value 5 to variable `y`.
5. On the right side of a value assignment operator an arithmetic operator must always exist.
6. On the left side of a value assignment operator more than one variable can exist.
7. You cannot use the same variable on both sides of a value assignment operator.
8. The statement `a = a + 1` decrements variable `a` by one.
9. In C++, the word `MOD` is a reserved word.
10. The statement `x = 0 % 5` assigns a value of 5 to variable `x`.
11. The operation `5 % 0` is not possible.

12. Division and multiplication have the higher precedence among the arithmetic operators.
13. When division and multiplication operators co-exist in an expression, multiplication operations are performed before division.
14. The expression `8 / 4 * 2` is equal to 1.
15. The expression `4 + 6 / 6 + 4` is equal to 9.
16. The expression `a + b + c / 3` calculates the average value of three numbers.
17. The statement `a += 1` is equivalent to `a++`.
18. The statement `a = "true"` assigns a Boolean value to variable `a`.
19. The statement `a = 2 a` doubles the content of variable `a`.
20. The statements `a++` and `++a` are completely equivalent.
21. The statement `a = a++ - a--;` always assigns a value of -1 to variable `a`.
22. The statement `a = "George" + " Malkovich"` assigns value `GeorgeMalkovich` to variable `a`.
23. The following code fragment satisfies the property of definiteness.

```
int a, b;
double x;
cin >> a;
cin >> b;
x = a / (b - 7);
cout << x << endl;
```

7.8 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. Which of the following C++ statements assigns a value of 10 to variable `a`?
   a. `10 = a;`
   b. `a ← 10;`
   c. `a = 100 / 10;`
   d. none of the above
2. The statement `a = b` can be read as
   a. assign the content of variable `a` to variable `b`.
   b. variable `b` is equal to variable `a`.
   c. assign the content of variable `b` to variable `a`.
   d. none of the above
3. The expression `0 % 10 + 2` is equal to
   a. 7.
   b. 2.
   c. 12.

d. none of the above

4. Which of the following C++ statements is syntactically correct?
   a. `a = 4 * 2 y - 8 / (4 * q);`
   b. `a = 4 * 2 * y - 8 / 4 * q);`
   c. `a = 4 * 2 * y - 8 / (4 */ q);`
   d. none of the above
5. Which of the following C++ statements increments variable `a` by one and then assigns its value to variable `b`?
   a. `b = a + 1;`
   b. `b = a++;`
   c. `b = ++a;`
   d. all of the above
6. Which of the following C++ statements assigns value `George Malkovich` to variable `a`?
   a. `a = "George" + " " + "Malkovich";`
   b. `a = "George" + " Malkovich";`
   c. `a = "George " + "Malkovich";`
   d. all of the above
7. The following code fragment

```
x = 2;
x++;
```

   does **not** satisfy the property of
   a. finiteness.
   b. definiteness.
   c. effectiveness.
   d. none of the above
8. The following code fragment

```
int a;
double x;

cin >> a;
x = 1 / a;
```

   does **not** satisfy the property of
   a. finiteness.
   b. input.
   c. effectiveness.
   d. none of the above

7.9 Review Exercises

Complete the following exercises.

1. Which of the following C++ assignment statements are syntactically correct?

   i. `a ← a + 1;`
   ii. `a += b;`
   iii. `a b = a b + 1;`
   iv. `a = a++;`
   v. `a = hello;`
   vi. `a = 40";`
   vii. `a = b · 5;`
   viii. `a =+ "true";`
   ix. `fdadstwsdgfgw = 1;`
   x. `**a;`

2. What is the type of each of the following variables?

   i. `a = "false";`
   ii. `w = false;`
   iii. `b = "15 meters";`
   iv. `weight = "40";`
   v. `b = 13.0;`
   vi. `b = 13;`

3. Match each element from the first column with one element from the second column.

Operation	Result
i. 1 / 2	a. 100
ii. 1 / 2 * 2	b. 0.25
iii. 0 % 10 * 10	c. 0
iv. 10 % 2 + 7	d. 0.5
	e. 7
	f. 1

4. What displays on the screen after executing each of the following code fragments?

   i.
   ```
   a = 5;
   b = a * a + 1;
   cout << b++ << endl;
   ```

   ii.
   ```
   a = 9;
   b = a / 3 * a;
   cout << ++b << endl;
   ```

5. What displays on the screen after executing each of the following code fragments?

   i.
   ```
   a = 5;
   a = a++;
   cout << a << endl;
   ```

   ii.
   ```
   a = 5;
   a = ++a;
   cout << a << endl;
   ```

6. What is the result of each of the following operations?

   i. `21 % 5`
   ii. `10 % 2`
   iii. `11 % 2`
   iv. `10 % 6 % 3`
   v. `0 % 3`
   vi. `100 / 10 % 3`

7. What displays on screen after executing each of the following code fragments?

i.
```
a = 5;
b = 2;
c = a % ++b;
d = b % (a + b);
cout << c;
count "*" << d << endl;
```

ii.
```
a = 0.4;
b = 8;
a += 0.1;
c = a * b % b;
cout << c << endl;
```

8. Calculate the result of the expression `a % b` for the following cases.

i. `a = 20, b = 3`
ii. `a = 15, b = 3`
iii. `a = 22, b = 3`
iv. `a = 0, b = 3`
v. `a = 3, b = 0`
vi. `a = 2, b = 2`

9. Calculate the result of the expression

```
b * (a % b) + a / b
```

for each of the following cases.

i. `a = 10, b = 5`
ii. `a = 10, b = 4`

10. What displays on the screen after executing the following code fragment?

```
a = "My name is";
a += " ";
a = a + "George Malkovich";
cout << a << endl;
```

11. Fill in the gaps in each of the following code fragments so that they display a value of 5.

i.
```
a = 2;
a = a - …… ;
cout << a << endl;
```

ii.
```
a = 4;
b = a * 0.5;
b += a;
a = b - …… ;
cout << a << endl;
```

12. What displays on the screen after executing the following code fragment?

```
city = "California";
California = city;
cout << city << ", " << California << endl;
```

Chapter 8
Trace Tables

8.1 What is a Trace Table?

A trace table is a technique used to test algorithms or computer programs for logic errors that occur while the algorithm or program executes.

The trace table simulates the flow of execution. Statements are executed step by step, and the values of variables change as an assignment statement is executed.

Trace tables are typically used by novice programmers to help them visualize how a particular algorithm or program works. Trace tables can also help advanced programmers detect logic errors.

A typical trace table is shown here.

Step	Statement	Notes	variable1	variable2	variable3
1					
2					
...					

Let's see a trace table in action! For the following C++ program, a trace table is created to determine the values of the variables in each step.

```
#include <iostream>
using namespace std;
int main() {
  int x, y, z;

  x = 10;
  y = 15;
  z = x * y;
  z++;
  cout << z;
  return 0;
}
```

The trace table for this program is shown below. Notes are optional, but they help the reader to better understand what is really happening.

Step	Statement	Notes	x	y	z
1	`x = 10`	The value 10 is assigned to variable `x`.	**10**	?	?
2	`y = 15`	The value 15 is assigned to variable `y`.	10	**15**	?
3	`z = x * y`	The result of the product `x * y` is	10	15	**150**

		assigned to z.			
4	z++	Variable z is incremented by one.	10	15	**151**
5	cout << z	The value 151 is displayed.			

Exercise 8.1-1 *Creating a Trace Table*

What result is displayed when the following program is executed?

```
#include <iostream>
using namespace std;
int main() {
  string Ugly, Beautiful, Handsome;

  Ugly = "Beautiful";
  Beautiful = "Ugly";
  Handsome = Ugly;
  cout << "Beautiful";
  cout << Ugly;
  cout << Handsome;
  return 0;
}
```

Solution

Let's create a trace table to find the output result.

Step	Statement	Notes	Ugly	Beautiful	Handsome
1	Ugly = "Beautiful"	The string "Beautiful" is assigned to the variable Ugly.	**Beautiful**	?	?
2	Beautiful = "Ugly"	The string "Ugly" is assigned to the variable Beautiful.	Beautiful	**Ugly**	?
3	Handsome = Ugly	The value of variable Ugly is assigned to the variable Handsome.	Beautiful	Ugly	**Beautiful**
4	cout << "Beautiful"	The string "Beautiful" is displayed.			
5	cout << Ugly	The string "Beautiful" is displayed.			
6	cout << Handsome	The string "Beautiful" is displayed.			

Exercise 8.1-2 *Swapping Values of Variables*

Write a C++ program that lets the user enter two values, in variables `a` *and* `b`*. At the end of the program, the two variables should swap their values. For example, if variables* `a` *and* `b` *contain the values 5 and 7 respectively, after swapping their values, variable* `a` *should contain the value 7 and variable* `b` *should contain the value 5!*

Solution

The following program, even though it may seem correct, is erroneous and doesn't really swap the values of variables `a` and `b`!

```
#include <iostream>
using namespace std;
int main() {
  int a, b;

  cin >> a;
  cin >> b;

  a = b;
  b = a;

  cout << a << endl;
  cout << b << endl;
  return 0;
}
```

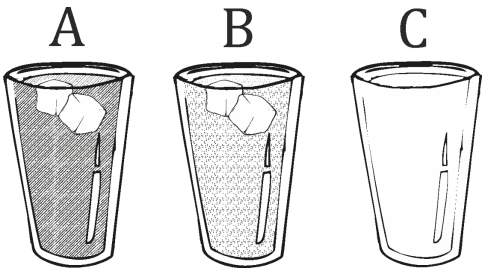
Let's see why! Suppose the user enters two values, 5 and 7. The trace table is shown here.

Step	Statement	Notes	a	b
1	`cin >> a`	User enters the value 5	**5**	?
2	`cin >> b`	User enters the value 7	5	**7**
3	`a = b`	The value of variable `b` is assigned to variable `a`. Value 5 is lost!	**7**	7
4	`b = a`	The value of variable `a` is assigned to variable `b`	7	**7**
5	`cout << a << endl`	The value 7 is displayed		
6	`cout << b << endl`	The value 7 is displayed		

Oops! Where is the value 5?

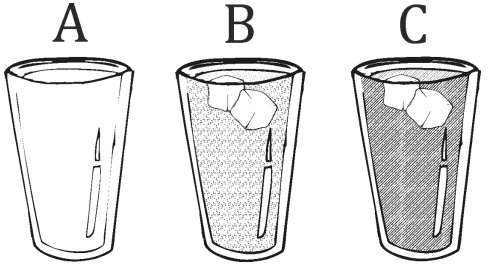
The solution wasn't so obvious after all! So, how do you really swap values anyway?

Consider two glasses: a glass of orange juice (called glass A), and a glass of lemon juice (called glass B). If you want to swap their content, all you must do is find and use one extra empty glass (called glass C).

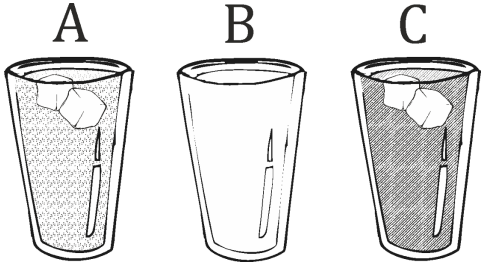


The steps that must be followed are:

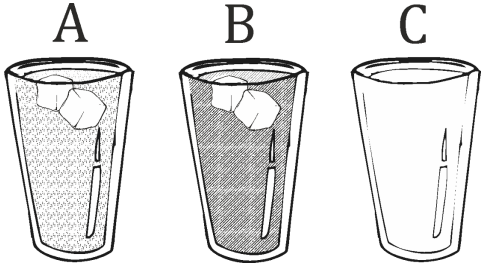
1. Empty the contents of glass A (orange juice) into glass C.



2. Empty the contents of glass B (lemon juice) into glass A.



3. Empty the contents of glass C (orange juice) into glass B.



Swapping completed successfully!

You can follow the same steps to swap the contents of two variables in C++.

```
#include <iostream>
```

```
using namespace std;
int main() {
  int a, b, c;

  cin >> a;
  cin >> b;

  c = a;   //Empty the contents of glass A (orange juice) into glass C
  a = b;   //Empty the contents of glass B (lemon juice) into glass A
  b = c;   //Empty the contents of glass C (orange juice) into glass B

  cout << a << endl;
  cout << b << endl;
  return 0;
}
```

Notice: *Please note the double slashes after each of the three assignment statements. The text after double slashes is considered a comment and is never executed.*

Exercise 8.1-3 *Swapping Values of Variables – A Second Approach*

Write a C++ program that lets the user enter two numeric values, in variables `a` *and* `b`*. In the end, the two variables should swap their values.*

Solution

Since the variables contain only numeric values, you can use the following C++ program.

```
#include <iostream>
using namespace std;
int main() {
  int a, b;

  cin >> a;
  cin >> b;

  a = a + b;
  b = a - b;
  a = a - b;

  cout << a << endl;
  cout << b << endl;
  return 0;
}
```

Notice*: Compared to previous exercise, the only major disadvantage of this method is that it cannot swap the contents of alphanumeric variables (strings).*

Exercise 8.1-4 *Creating a Trace Table*

Create a trace table to determine the values of the variables in each step of the C++ program for three different executions.

The input values for the three executions are: (i) 0.3, (ii) 4.5, and (iii) 10.

```
#include <iostream>
using namespace std;
int main() {
  double a, b, c;

  cin >> b;
  c = 3;
  c = c * b;
  a = 10 * c % 10;

  cout << a;
  return 0;
}
```

Solution

i. For the input value of 0.3, the trace table looks like this.

Step	Statement	Notes	a	b	c
1	cin >> b	User enters value 0.3	?	**0.3**	?
2	c = 3		?	0.3	**3**
3	c = c * b		?	0.3	**0.9**
4	a = 10 * c % 10		**9**	0.3	0.9
5	cout << a	The value 9 is displayed			

ii. For the input value of 4.5, the trace table looks like this.

Step	Statement	Notes	a	b	c
1	cin >> b	User enters value 4.5	?	**4.5**	?
2	c = 3		?	4.5	**3**
3	c = c * b		?	4.5	**13.5**
4	a = 10 * c % 10		**5**	4.5	13.5
5	cout << a	The value 5 is displayed			

iii. For the input value of 10, the trace table looks like this.

Step	Statement	Notes	a	b	c
1	cin >> b	User enters value 10	?	**10**	?

2	`c = 3`		?	10	**3**
3	`c = c * b`		?	10	**30**
4	`a = 10 * c % 10`		**0**	10	30
5	`cout << a`	The value 0 is displayed			

Exercise 8.1-5 *Creating a Trace Table*

Create a trace table to determine the values of the variables in each step when a value of 3 is entered.

```
#include <iostream>
using namespace std;
int main() {
  double a, b, c, d;

  cin >> a;

  b = a + 10;
  a = b * (a - 3);
  c = 3 * b / 6;
  d = c * c;
  d--;

  cout << d;
  return 0;
}
```

Solution

For the input value of 3, the trace table looks like this.

Step	Statement	Notes	a	b	c	d
1	`cin >> a`	User enters value 3	**3**	?	?	?
2	`b = a + 10`		3	**13**	?	?
3	`a = b * (a - 3)`		**0**	13	?	?
4	`c = 3 * b / 6`		0	13	**6.5**	?
5	`d = c * c`		0	13	6.5	**42.25**
6	`d--`		0	13	6.5	**41.25**
7	`cout << d`	The value 41.25 is displayed				

8.2 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A trace table is a technique for testing a computer.
2. Trace tables help a programmer find errors in a computer program.
3. You cannot write a computer program without first creating its corresponding trace table.
4. In order to swap the values of two integer variables, you always need an extra variable.

8.3 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the C++ program for three different executions.

   The input values for the three executions are: (i) 3, (ii) 4, and (iii) 1.

```
#include <iostream>
using namespace std;
int main() {
  int a, b, c, d;

  cin >> a;

  a = (a + 1) * (a + 1) + 6 / 3 * 2 + 20;
  b = a % 13;
  c = b % 7;
  d = a * b * c;
  cout << a << ", " << b << ", " << c << ", " << d;
  return 0;
}
```

2. Create a trace table to determine the values of the variables in each step of the C++ program for two different executions.

   The input values for the two executions are: (i) 3, 4; and (ii) 4, 4

```
#include <iostream>
using namespace std;
int main() {
  double a, b, c, d, e;

  cin >> a;
  cin >> b;

  c = a + b;
  d = 1 + a / b * c + 2;
  e = c + d;
  c += d + e;
```

```
    e--;
    d -= c + d % c;
    cout << c << ", " << d << ", " << e;
    return 0;
}
```

Chapter 9
Using NetBeans IDE

9.1 Creating a New C++Project

So far, you have learned some really good basics about C++ programs. Now it's time to learn how to enter programs into the computer, execute them, see how they perform, and see how they display the results.

The first thing you must do is create a new C++ project. The NetBeans IDE provides a wizard to help you do that. Start the NetBeans IDE, and from its main menu select "File→New Project…" as shown in **Figure 9–1**.

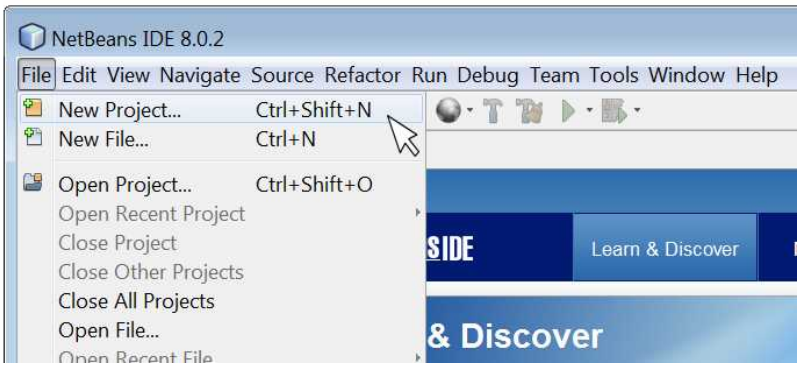


Figure 9–1 Starting a new C++ project from NetBeans IDE

The "New Project" wizard will appear, and it will guide you through the steps that follow.

1. On the "New Project" screen of the wizard, under "Choose Project," select "C++ Application" as shown in **Figure 9–2**. Then, click on the "Next" button.

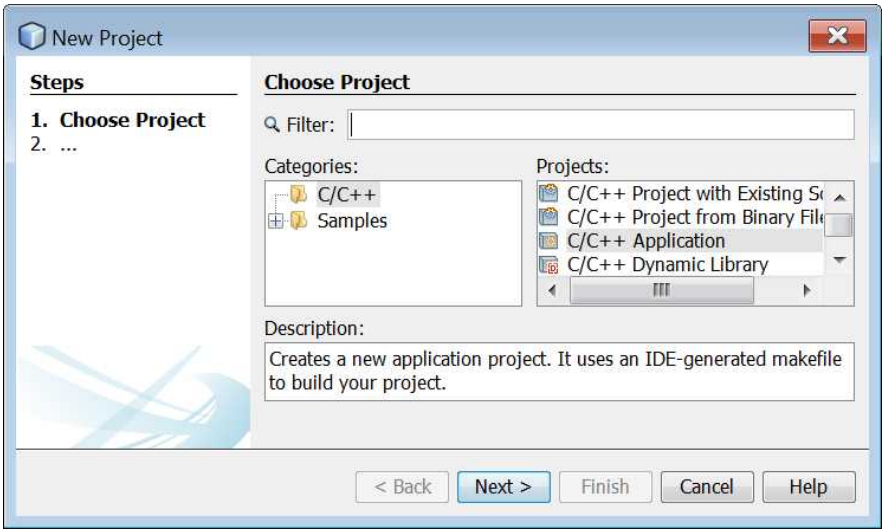


Figure 9–2 Selecting the C++ Application under Projects

2. You will see the "New C++ Application" screen of the wizard. Under "Name and Location" , in the "Project Name" field, type "testingProject" as shown in **Figure 9–3**. Leave all other fields unchanged. Then, click on the "Next" button.

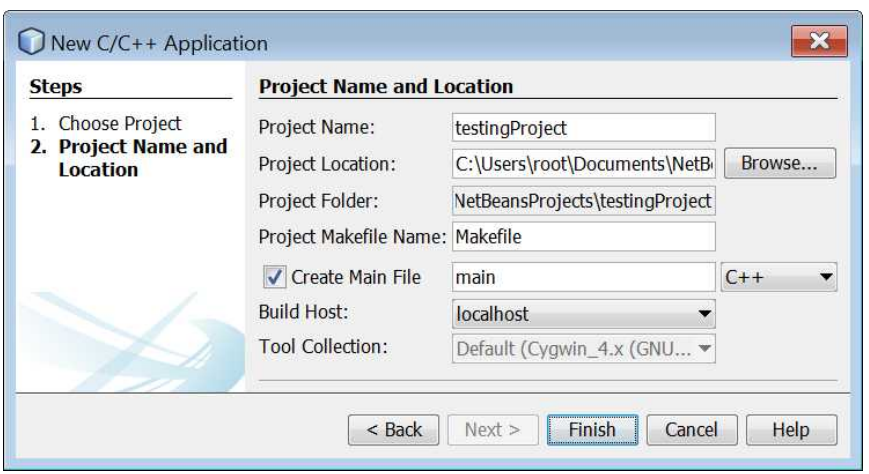


Figure 9–3 Selecting the name and location of a new C++ project

3. Click on the "Finish" button.

The project is created and opened in your NetBeans IDE. You should see the following components (see **Figure 9–4**):

- the Projects window, which contains a tree view of the components of the projects, including source files, libraries that your code may depend on, and so on. Find and double click on the testingProject→Source Files→main.cpp file.

- the Source Editor window with the file called "main.cpp" that you just opened. In this file you can write your C++ code. Of course, one single project can contain many such files.

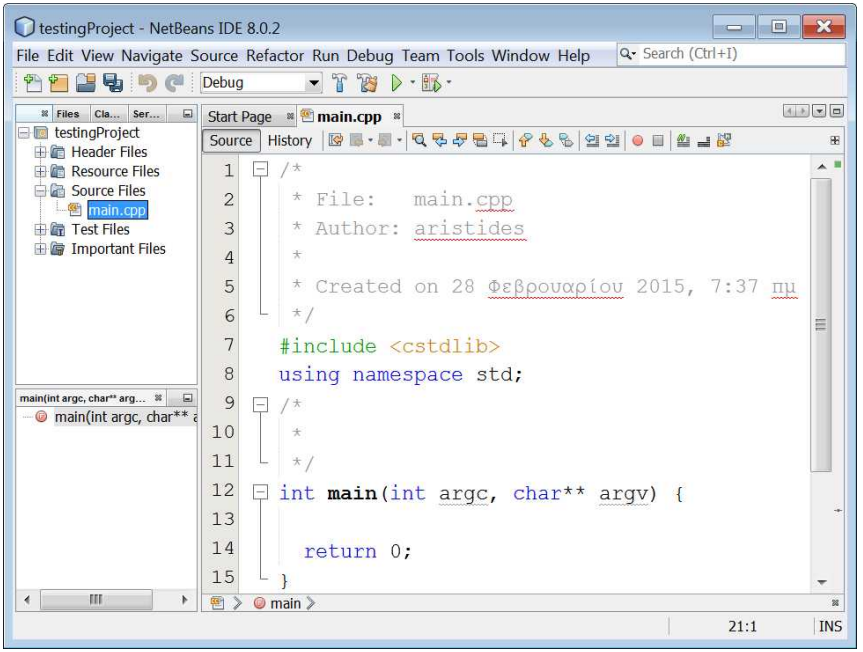


Figure 9–4 Viewing the "Projects" and "Source Editor" windows in NetBeans IDE

9.2 Writing and Executing a C++ Program

You have just seen how to create a new C++ project. Let's now write the following (terrifying, and quite horrifying!) C++ program and try to execute it.

```
#include <iostream>
using namespace std;
int main() {
  cout << "Hello World";
  return 0;
}
```

In window "main.cpp", delete everything and type only the first character from the `#include` statement by hitting the "#" key on your keyboard. A popup window appears, as shown in **Figure 9–5**. This window contains all available C++ statements, and other items that begin with the character "#."

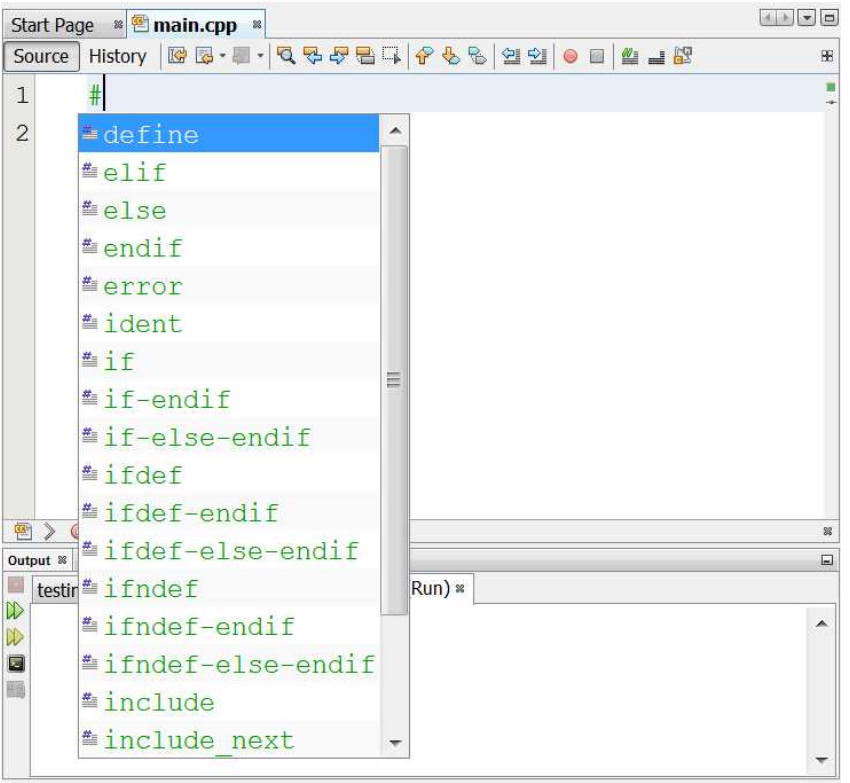


Figure 9–5 The popup screen in the Source Editor window

You can highlight a selection by using the up and down arrow keys on your keyboard.

Type the second character "i" from the `#include` statement. Now the options have become fewer as shown in **Figure 9–6**.

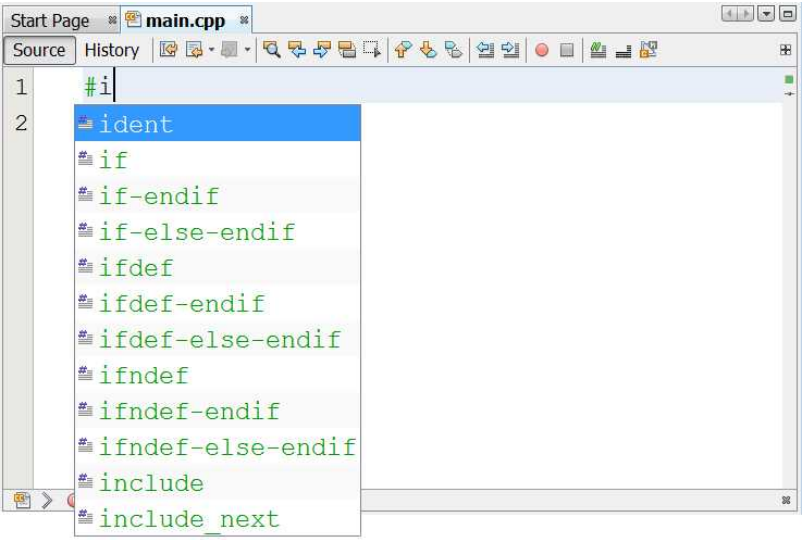


Figure 9–6 The popup screen in the Source Editor window

Select the option `#include` and hit "Enter ↵" key. The statement is automatically entered into your program. Complete the statement by writing `<iostream>`. Then, continue typing the rest of the C++ program (as shown in **Figure 9–7**). Your NetBeans IDE should look like this.

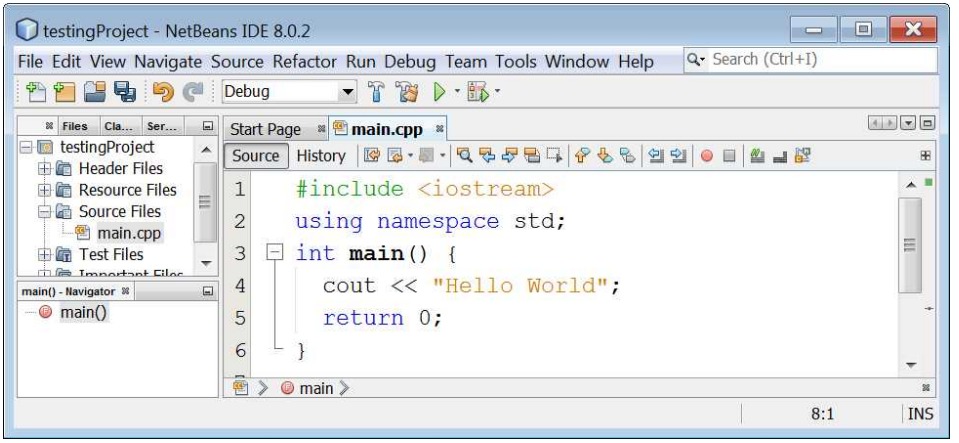


Figure 9–7 Entering a C++ program in the "main.cpp" file

Now let's try to execute the program! From the main menu, select "Run→Run Project" or hit the F6 key. The C++ program executes and the output is displayed in the "Output" window as shown in **Figure 9–8**.

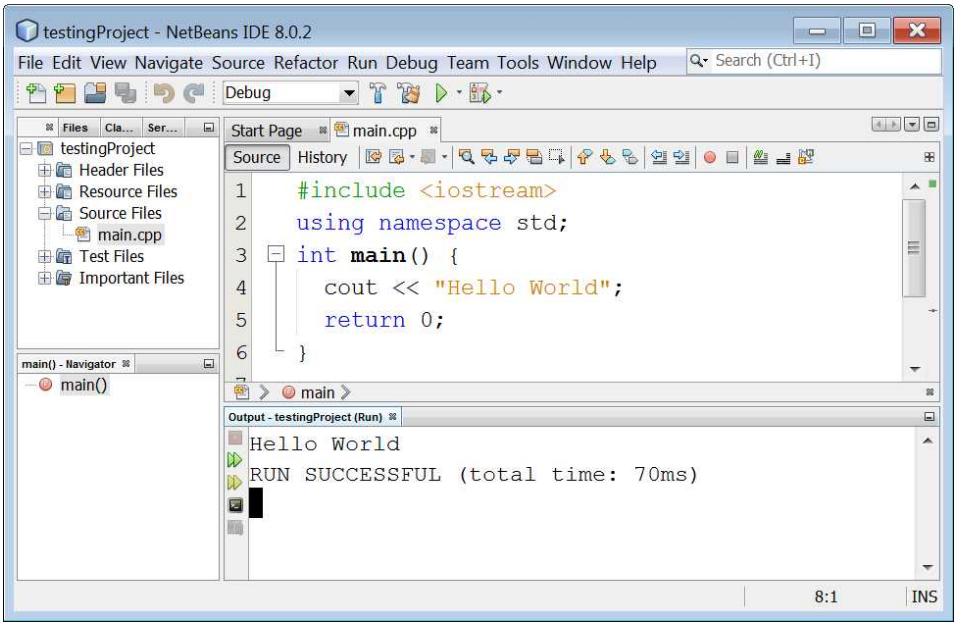


Figure 9–8 Viewing the results of the executed program in the Output window

Congratulations! You have just written and executed your first C++ program!

Now let's write another C++ program, one that prompts the user to enter his or her name. Type the following C++ statements into the NetBeans IDE and hit F6 to execute the project.

project_9_2

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

int main() {
  string name;

  cout << "Enter your name: ";
  cin >> name;
  cout << "Hello " << name << endl;
  cout << "How do you do?" << endl;

  return 0;
}
```

***Remember**! You can execute a project by selecting "Run →Run Project" from the main menu, or by hitting the F6 key.*

Once you execute the program, the message "Enter your name" is displayed in the "Output" window. The program waits for you to enter your name, as shown in **Figure 9–9**.

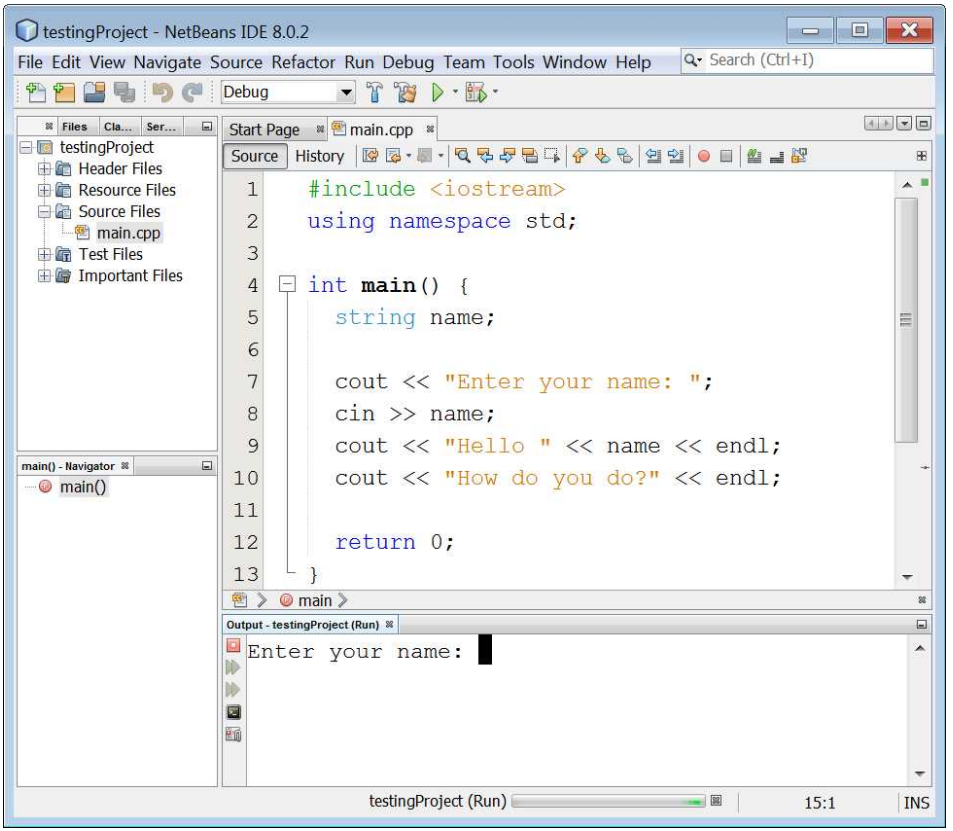


Figure 9–9 Viewing a prompt in the Output window

Type your name and hit the "Enter ↵" key. Once you do that, your computer continues executing the rest of the statements. When execution finishes, the final output is as shown in **Figure 9–10**.

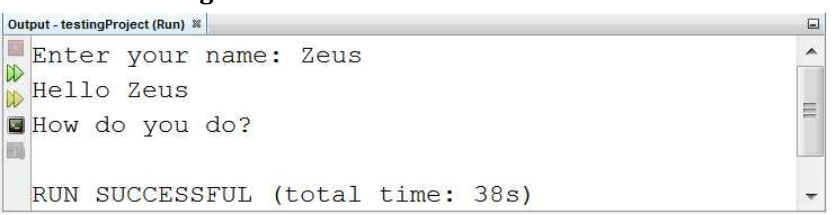


Figure 9–10 Responding to the prompt in the Output window

9.3 What "Debugging" Means

Debugging is the process of finding and reducing the number of defects (bugs) in a computer program, in order to make it perform as expected.

There is a myth about the origin of the term "debugging." In 1940, while Grace Hopper[1] was working on a Mark II Computer at Harvard University, her associates discovered a bug (a moth) stuck in a relay (an electrically operated switch). This bug was blocking the proper operation of the Mark II computer. So, while her associates where trying to remove the bug, Grace Hopper remarked that they were "debugging" the system!

9.4 Debugging C++ Programs with NetBeans IDE

As you already know, when someone writes code in a high-level language there are two types of errors that he or she might make—syntax errors and logic errors. NetBeans IDE provides all the necessary tools to help you debug your programs and find the errors.

Debugging syntax errors

Fortunately, NetBeans IDE detects syntax errors while you are typing (or when you are trying to run the project) and underlines them with a wavy red line as shown in **Figure 9–11.**

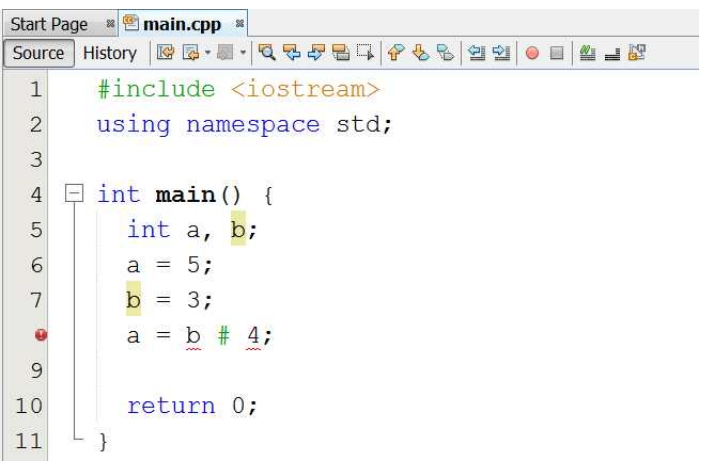


Figure 9–11 In the NetBeans IDE, syntax errors are underlined with a wavy red line

All you have to do is correct the corresponding error and the red line will disappear at once. However, if you are not certain about what is wrong with your code, you can just place your mouse cursor on the red exclamation mark. NetBeans IDE will try to help you by showing a popup window with a brief explanation of the error, as shown in **Figure 9–12**.

[1] Grace Murray Hopper (1906–1992) was an American computer scientist and US Navy admiral. She was one of the first programmers of the Harvard Mark I computer, and developed the first compiler for a computer programming language known as A–0 and later a second one, known as B–0 or FLOW-MATIC.

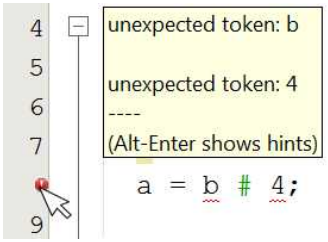


Figure 9–12 The NetBeans IDE shows an explanation of a syntax error

Debugging logic errors by executing programs step by step

Compared to syntax errors, logic errors are more difficult to find. Since the NetBeans IDE cannot spot and underline logic errors, you are all alone! Let's look at following C++ program, for example. It prompts the user to enter two numbers and calculates and displays their sum. However, it contains a logic error!

project_9_4a

```
#include <iostream>
using namespace std;
int main() {
  double s, a, b, S;

  s = 0;
  cout << "Enter 1st value: ";
  cin >> a;
  cout << "Enter 2nd value: ";
  cin >> b;
  S = a + b;
  cout << "The sum is: " << s;
  return 0;
}
```

If you type this program into the NetBeans IDE, you will notice that there is not even one exclamation mark indicating any error. If you run the program, however, and enter two values, 5 and 2, you can see for yourself that even though the sum of 5 and 2 is 7, NetBeans IDE insists that it is zero, as shown in **Figure 9–13**.

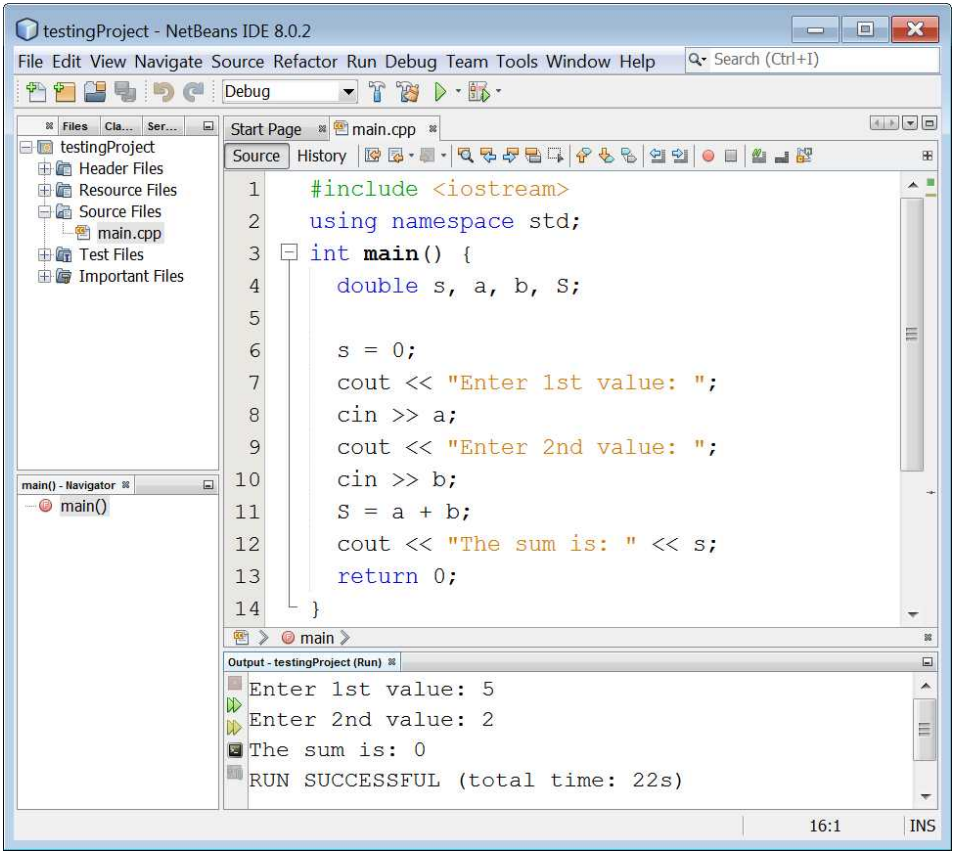


Figure 9–13 Viewing the result of a logic error in the Output window

What the heck is going on? Of course for an expert programmer, correcting this error would be a piece of cake. But for you, a novice one, even though you are trying hard you find nothing wrong. So, where is the error?

Sometimes human eyes get so tired that they cannot see the obvious. So, let's try to use some magic! Let's try to execute the program step by step using the debugger. This gives you the opportunity to observe the flow of execution and take a closer look at the current values of variables in each step.

Place the cursor at the line where the statement `s = 0` exists and hit the CTRL+F8 key combination. A pink rectangle appears in the gray margin and the corresponding line has pink background highlighting it as shown in **Figure 9–14**.

```
3  int main() {
4    double s, a, b, S;
5
   s = 0;
7    cout << "Enter 1st value: ";
8    cin >> a;
9    cout << "Enter 2nd value: ";
```

Figure 9–14 Using the debugger in the NetBeans IDE

Now, start the debugger by selecting "Debug→Debug Project" from the main menu or by hitting the CTRL+F5 key combination. By doing this, you enable the debugger and more icons are displayed on the toolbar.

The program counter (the green arrow ⇨ icon in the gray margin) stops at the first line of the program as shown in **Figure 9–15**.

```
3  int main() {
4    double s, a, b, S;
5
   s = 0;
7    cout << "Enter 1st value: ";
8    cin >> a;
9    cout << "Enter 2nd value: ";
```

Figure 9–15 Using the debugger in the NetBeans IDE

***Notice**: The program counter shows you, at any time, which statement is the next to be executed.*

Click on the "Step Over"  icon on the toolbar. This action executes the statement `s = 0;` and the program counter moves to the second C++ statement, which is the next to be executed.

From the main menu, select "Window→Debugging→Variables". In the window that appears, you can watch all the variables declared in main memory (RAM) and the value they contain during each step of execution as shown in **Figure 9–16**.

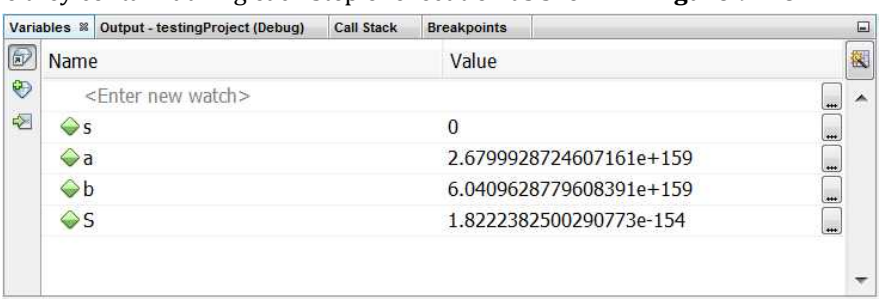


Figure 9–16 All declared variables are displayed in the Variables window of the debugger

***Notice**: Please note that, currently, only variable `s` contains a meaningful value.*

Click on the "Step Over" toolbar icon again. The second statement is executed. You can go back to the "Output" window and see the output result (see **Figure 9–17**).

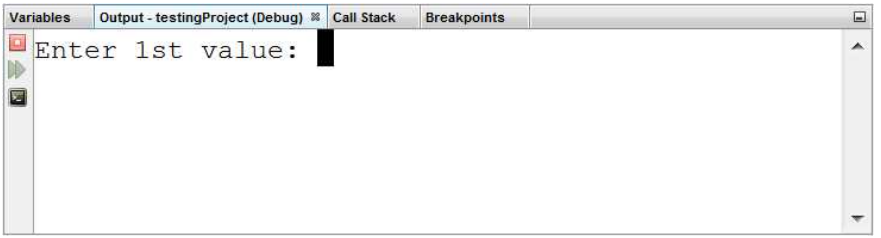


Figure 9–17 Viewing a prompt in the Output window

Click on the "Step Over" toolbar icon again. The third C++ statement is executed and the program waits for you to enter a value. Place the cursor inside the "Output" window, type the value 5, and hit the "Enter ↵" key.

The program counter moves to the fourth C++ statement. Click on the "Step Over" toolbar icon twice. This action executes the fourth and fifth C++ statements. Place the cursor inside the "Output" window, type the value 2, and hit the "Enter ↵" key.

The program counter moves to the sixth C++ statement. Click on the "Step Over" toolbar icon once again. The statement `S = a + b` is executed. What you expected here was the sum of 5.0 and 2.0, which is 7.0, to be assigned to variable `s`. Instead of this, the value 7.0 is assigned to the variable `S` (with a capital S), as shown in **Figure 9–18**.

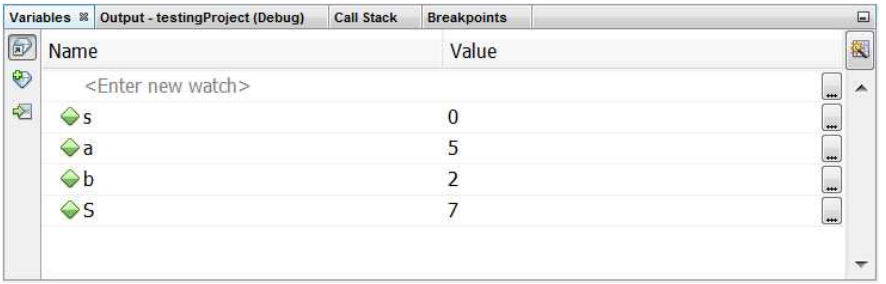


Figure 9–18 Variables and their values are displayed in the Variables window of the debugger

Now it becomes more obvious to you! You mistakenly declared two variables in the main memory (RAM), the variable `s` and the variable `S` with a capital S. So, when the flow of execution goes to the last statement, `cout << "The sum is: " << s` the value 0 instead of the value 7 is displayed.

You have just found the error! Click on the "Finish Debugger Session" toolbar icon to cancel execution, correct the error by changing variable `S` to `s`, and you are ready! You just performed your first debugging! Re-execute the program and you will now see that it calculates and displays the sum correctly.

Debugging logic errors by adding breakpoints

Debugging step by step has a big disadvantage. You have to click on the "Step Over" toolbar icon again and again until you reach the position where the error might be. Can you imagine yourself doing this in a large program?

For large programs there is another approach. If you suspect that the error is somewhere at the end of the program there is no need to debug all of it right from the beginning. You can add a marker (called a "breakpoint") where you think that the error might be, execute the program and when flow of execution reaches that breakpoint, the flow of execution will pause automatically. You can then take a closer look at the current values of the variables at the position where the program was paused.

***Notice**: When the program pauses, you have two options for resuming the execution: you can add a second breakpoint somewhere below in the program, click on the "Continue" toolbar icon, and allow the program to continue execution until that new breakpoint; or, you can just use the "Step Over" toolbar icon and execute the program step by step thereafter.*

The next C++ program prompts the user to enter two values and calculates their average value.

However, the program contains a logic error. When the user enters the values 10 and 12, the value 16, instead of 11, is displayed.

project_9_4b

```
#include <iostream>
using namespace std;
int main() {
  double a, b, average;

  cout << "Enter 1st value: ";
  cin >> a;
  cout << "Enter 2nd value: ";
  cin >> b;
  average = a + b / 2;
  cout << "The average value is: " << average;
  return 0;
}
```

You suspect that the problem is somewhere at the end of the program. However, you do not want to debug the entire program, but just the portion in which the error might be. So, let's try to add a breakpoint at the `average = a + b / 2` statement (see **Figure 9–19**). There are two ways to do this: you can click in the left gray margin on the corresponding line number, or you can place the cursor at the line of interest and hit the CTRL+F8 key combination.

```
7    cin >> a;
8    cout << "Enter 2nd value: ";
9    cin >> b;
     average = a + b / 2;
11   cout << "The average value is: " << average;
```

Figure 9–19 Adding a breakpoint to a program

***Notice**: You know that a breakpoint has been set when the pink rectangle appears in the gray margin and the corresponding line has pink background highlighting it.*

Hit CTRL+F5 to start the debugger. Enter the values 10 and 12 when requested in the output window. You will notice that just after you enter the second number and hit the "Enter ↵" key, the flow of execution pauses at the breakpoint. (You will see that the pink breakpoint highlighting is replaced by the green highlighting of the program counter.)

***Remember**! You can debug a project by selecting "Debug→Debug Project" from the main menu or by hitting the CTRL+F5 key combination.*

Now you can take a closer look at the current values of the variables. Variables `a` and `b` contain the values 10.0 and 12.0 respectively, as they should, so there is nothing wrong with the data input, as shown in **Figure 9–20**.

Figure 9–20 Viewing the current values of the variables in the Variables window

Click on the "Step Over" toolbar icon once. The statement `average = a + b / 2` executes and the main memory (RAM) now contains the following values (see **Figure 9–21**).

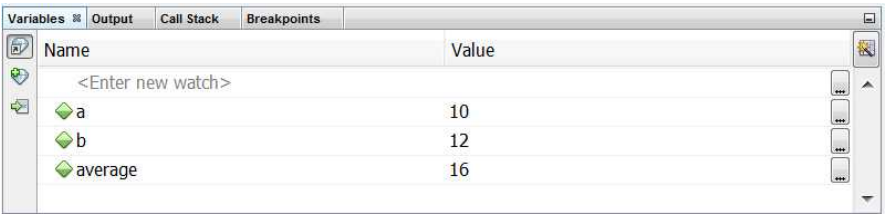


Figure 9–21 Viewing the current values of the variables in the Variables window

There it is! You just found the statement that erroneously assigns a value of 16.0, instead of 11.0, to the variable `average`! And now comes the difficult part; you should consider why this happens!

After two days of thinking, it becomes obvious! You had just forgotten to enclose `a + b` inside parentheses; thus, only the variable `b` was divided by 2. Click on the "Finish Debugger Session"  toolbar icon, remove all breakpoints, correct the error by enclosing `a + b` inside parentheses and you are ready! Re-execute the program and see now that it calculates and displays the average value correctly.

***Notice**: You can remove a breakpoint the same way you added it: click in the left gray margin on the corresponding line number, or place the cursor at the line that contains a breakpoint and hit the CTRL+F8 key combination.*

9.5 Review Exercises

Complete the following exercises.

1. Type the following C++ program into the NetBeans IDE and execute it step by step. Determine why it doesn't calculate correctly the sum of 1 + 3 + 5.

```
#include <iostream>
using namespace std;
int main() {
  int SS, S1, S3, S5, S;

  SS = 0;
  S1 = 1;
  S3 = 3;
  S5 = 5;

  S = S1 + S3 + SS;

  cout << S;
  return 0;
}
```

2. Create a trace table to determine the values of the variables in each step of the C++ program for two different executions. Then, type the program in the NetBeans IDE, execute it step by step, and confirm the results.

   The input values for the two executions are: (i) 5, 5; and (ii) 4, 8.

```
#include <iostream>
using namespace std;
int main() {
  double a, b, c, d, e;

  cin >> a;
  cin >> b;

  c = a + b;
  d = 5 + a / b * c + 2;
  e = c - d;
  c -= d + c;
```

```
  e--;
  d -= c + a % c;

  cout << c << ", " << d << ", " << e;
  return 0;
}
```

3. Create a trace table to determine the values of the variables in each step of the C++ program for three different executions. Then, type the program in the NetBeans IDE, execute it step by step, and confirm the results.

   The input values for the three executions are: (i) 0.50, (ii) 3, and (iii) 15.

```
#include <iostream>
using namespace std;
int main() {
  int a;
  double b, c;

  cin >> b;

  c = 5;
  c = c * b;
  a = 10 * c % 10;

  cout << a;
  return 0;
}
```

Review Questions in "Getting Started with C++"

Answer the following questions.

1. What is an algorithm?
2. Give the algorithm for making a cup of coffee.
3. What are the five properties of algorithms?
4. Can an algorithm execute forever?
5. What is a computer program?
6. What are the three parties involved in an algorithm?
7. What are the three stages that make up a computer program?
8. Can a computer program be made up of two stages?
9. What is a flowchart?
10. Can a flowchart be entered into a computer as is?
11. What are the basic symbols that flowcharts use?
12. What is meant by the term "reserved words"?
13. What is structured programming?
14. What are the three fundamental control structures of structured programming?
15. Give an example of each control structure using flowcharts.
16. Can a programmer write C++ programs in a text editor?
17. What is a syntax error? Give one example.
18. What is a logic error? Give one example.
19. What type of error is caused by a misspelled keyword, a missing punctuation character, or the incorrect use of an operator?
20. Why should a programmer add comments in his or her code?
21. Why should a programmer write user-friendly programs?
22. What does the acronym POLA stand for?
23. What is a variable?
24. How many variables can exist on the left side of the left arrow in flowcharts?
25. In which part of a computer are the values of the variables stored?
26. What is a constant?
27. How can constants be used to help programmers?
28. Why should a computer program avoid division and multiplication operations whenever possible?
29. What are the four fundamental types of variables and constants in C++?
30. What does the phrase "declare a variable" mean?
31. How do you declare a variable in C++? Give an example.

32. How do you declare a constant in C++? Give an example.
33. What symbol is used in flowcharts to display a message?
34. What special sequence of characters is used in C++ to output a "line break"?
35. What symbol is used in flowcharts to let the user enter data?
36. Which symbol is used in C++ as a value assignment operator, and how is it represented in a flowchart?
37. Which arithmetic operators does C++ support?
38. What is a modulus operator, and how is it usually represented in flowcharts?
39. Summarize the rules for the precedence of arithmetic operators.
40. What compound assignment operators does C++ support?
41. What incrementing/decrementing operators does C++ support?
42. What string operators does C++ support?
43. What is a trace table?
44. What are the benefits of using a trace table?
45. Describe the steps involved in swapping the contents (either numeric or alphanumeric) of two variables.
46. Two methods for swapping the values of two variables have been proposed in this book. Which one is better, and why?
47. What does the term "debugging" mean?
48. Describe the way in which NetBeans IDE helps you find syntax errors.
49. Describe the ways in which NetBeans IDE helps you find logic errors.

Section 3

Sequence Control Structures

Chapter 10
Introduction to Sequence Control Structures

10.1 What is the Sequence Control Structure?

"Sequence control structure" refers to the line-by-line execution by which statements are executed sequentially, in the same order in which they appear in the program. They might, for example, carry out a series of read or write operations, arithmetic operations, or assignments to variables.

The sequence control structure is the simplest of the three fundamental control structures that you learned about in Chapter 4. The following program shows an example of C++ statements that are executed sequentially.

project_10_1

```
#include <iostream>
using namespace std;
int main() {
  double a, b;

  cout << "Enter a number: ";
  cin >> a;

  b = a * a;

  cout << "The square of " << a << " is " << b;
  return 0;
}
```

Exercise 10.1-1 *Calculating the Area of a Parallelogram*

Write a C++ program that calculates and displays the area of a parallelogram.

Solution

The area of a parallelogram can be calculated using the following formula:

$$Area = Base \times Height$$

In this exercise, the user enters values for *Base* and *Height* and the program calculates and displays the area of the parallelogram. The solution to this problem is shown here.

project_10_1_1

```
#include <iostream>
using namespace std;
int main() {
```

```
  double area, base, height;

  cout << "Enter the length of Base: ";
  cin >> base;
  cout << "Enter the length of Height: ";
  cin >> height;

  area = base * height;

  cout << "The area of the parallelogram is " << area;
  return 0;
}
```

Exercise 10.1-2 *Calculating the Area of a Circle*

Write a C++ program that calculates and displays the area of a circle.

Solution

The area of a circle can be calculated using the following formula:

$$Area = \pi \cdot Radius^2$$

The value of π is a known quantity, which is 3.14159. Therefore, the only value the user has to enter is a value for *Radius*. The solution to this problem is shown here.

project_10_1_2a

```
#include <iostream>
using namespace std;
int main() {
  double area, radius;

  cout << "Enter the length of Radius: ";
  cin >> radius;

  area = 3.14159 * radius * radius;

  cout << "The area of the circle is " << area;
  return 0;
}
```

A much better approach would be with to use a constant, `PI`.

project_10_1_2b

```
#include <iostream>
using namespace std;
const double PI = 3.14159;

int main() {
  double area, radius;
```

```
  cout << "Enter the length of Radius: ";
  cin >> radius;
  area = PI * radius * radius;
  cout << "The area of the circle is " << area;
  return 0;
}
```

***Notice**: Please note that the constant `PI` is declared outside of the function `main`.*

Exercise 10.1-3 *Calculating Fuel Economy*

In the United States, a car's fuel economy is measured in miles per gallon, or MPG. A car's MPG can be calculated using the following formula:

$$MPG = \frac{miles\ driven}{gallons\ of\ gas\ used}$$

Write a C++ program that prompts the user to enter the total number of miles he or she has driven and the gallons of gas used. Then the program should calculate and display the car's MPG.

Solution

This is quite a simple case. The user enters the total number of miles he or she has driven and the gallons of gas used, and then the program should calculate and display the car's MPG.

project_10_1_3

```
#include <iostream>
using namespace std;
int main() {
  double gallons, miles_driven, mpg;

  cout << "Enter miles driven: ";
  cin >> miles_driven;
  cout << "Enter gallons of gas used: ";
  cin >> gallons;

  mpg = miles_driven / gallons;

  cout << "Your car's MPG is: " << mpg;
  return 0;
}
```

Exercise 10.1-4 *Where is the Car? Calculating Distance Traveled*

A car starts from rest and moves with a constant acceleration along a straight horizontal road for a given time. Write a C++ program that prompts the user to enter

the acceleration and the time the car traveled, and then calculates and displays the distance traveled. The required formula is

$$S = u_o + \frac{1}{2}at^2$$

where

- ***S*** is the distance the car traveled, in meters (m)
- u_o is the initial velocity (speed) of the car, in meters per second (m/sec)
- ***t*** is the time the car traveled, in seconds (sec)
- ***a*** is the acceleration, in meters per second^2 (m/sec^2)

Solution

Since the car starts from rest, the initial velocity (speed) u_0 is zero. Thus, the formula becomes

$$S = \frac{1}{2}at^2$$

and the C++ program is

project_10_1_4

```
#include <iostream>
using namespace std;
int main() {
  double S, a, t;

  cout << "Enter acceleration: ";
  cin >> a;
  cout << "Enter time traveled: ";
  cin >> t;

  S = 0.5 * a * t * t;

  cout << "Your car traveled " << S << " meters";
  return 0;
}
```

Exercise 10.1-5 *Kelvin to Fahrenheit*

Write a C++ program that converts a temperature value from degrees Fahrenheit[1] to its degrees Kelvin[2] equivalent. The required formula is

$$1.8 \times Kelvin = Fahrenheit + 459.67$$

Solution

The formula given cannot be used in your program as is. In a computer language such as C++, it is not permitted to write

```
1.8 * kelvin = fahrenheit + 459.67
```

Remember! *In the position on the left side of the (=) sign, only one single variable may exist. This variable is actually a region in RAM where a value can be stored.*

The program converts degrees Fahrenheit to degrees Kelvin. The value for degrees Fahrenheit is a known value and it is given by the user, whereas the value for degrees Kelvin is what the C++ program should calculate. So, you need to solve for Kelvin. After a bit of work, the formula becomes

$$Kelvin = \frac{Fahrenheit + 459.67}{1.8}$$

and the C++ program is shown here.

project_10_1_5

```
#include <iostream>
using namespace std;
int main() {
  double fahrenheit, kelvin;

  cout << "Enter a temperature in Fahrenheit: ";
  cin >> fahrenheit;

  kelvin = (fahrenheit + 459.67) / 1.8;

  cout << "The temperature in Kelvin is " << kelvin;
  return 0;
}
```

[1] Daniel Gabriel Fahrenheit (1686–1736) was a German physicist, engineer, and glass blower who is best known for inventing both the alcohol and the mercury thermometers, and for developing the temperature scale now named after him.
[2] William Thomson, 1st Baron Kelvin (1824–1907), was an Irish-born British mathematical physicist and engineer. He is widely known for developing the basis of absolute zero (the Kelvin temperature scale), and for this reason a unit of temperature measure is named after him. He discovered the Thomson effect in thermoelectricity and helped develop the second law of thermodynamics.

Exercise 10.1-6 *Calculating Sales Tax*

An employee needs a program to enter the before-tax price of a product and calculate its final price. Assume a value added tax (VAT) of 19%.

Solution

The sales tax can be easily calculated. You must multiply the before-tax price of the product by the sales tax rate. Be careful—the sales tax is not the final price, but only the tax amount.

The after-tax price can be calculated by adding the initial before-tax price and the sales tax that you calculated beforehand.

In this program you can use a constant named `VAT` for the sales tax rate.

project_10_1_6

```
#include <iostream>
using namespace std;
const double VAT = 0.19;

int main() {
  double price_after_tax, price_before_tax, sales_tax;

  cout << "Enter the before-tax price of a product: ";
  cin >> price_before_tax;

  sales_tax = price_before_tax * VAT;
  price_after_tax = price_before_tax + sales_tax;

  cout << "The after-tax price is: " << price_after_tax;
  return 0;
}
```

***Notice**: Please note that the constant `VAT` is declared outside of the function `main`.*

Exercise 10.1-7 *Calculating a Sales Discount*

Write a C++ program that prompts the user to enter the price of an item and the discount offered as a percentage (on a scale of 0 to 100). The program should then calculate and display the new price.

Solution

The discount amount can be easily calculated. You must multiply the before-discount price of the product by the discount value and then divide it by 100. The division is necessary since the user enters a value for the discount on a scale of 0 to 100. Be careful—the result is not the final price but only the discount amount.

The final after-discount price can be calculated by subtracting the discount amount that you calculated beforehand from the initial before-discount price.

project_10_1_7

```
#include <iostream>
using namespace std;
int main() {
  int discount;
  double discount_amount, price_after_discount, price_before_discount;

  cout << "Enter the price of a product: ";
  cin >> price_before_discount;

  cout << "Enter the discount offered (0 - 100): ";
  cin >> discount;

  discount_amount = price_before_discount * discount / 100;
  price_after_discount = price_before_discount - discount_amount;

  cout << "The price after discount is: " << price_after_discount;
  return 0;
}
```

Exercise 10.1-8 *Calculating the Sales Tax Rate and Discount*

Write a C++ program that prompts the user to enter the before-tax price of an item and the discount offered as a percentage (on a scale of 0 to 100). The program should then calculate and display the new price. Assume a sales tax rate of 19%.

Solution

This exercise is just a combination of the previous two exercises!

project_10_1_8

```
#include <iostream>
using namespace std;
const double VAT = 0.19;

int main() {
  int discount;
  double discount_amount, price_after_discount, price_after_tax;
  double price_before_discount, sales_tax;

  cout << "Enter the price of a product: ";
  cin >> price_before_discount;

  cout << "Enter the discount offered (0 - 100): ";
  cin >> discount;
```

```
    discount_amount = price_before_discount * discount / 100;
    price_after_discount = price_before_discount - discount_amount;

    sales_tax = price_after_discount * VAT;
    price_after_tax = price_after_discount + sales_tax;

    cout << "The discounted after-tax price is: " << price_after_tax;
    return 0;
}
```

10.2 Review Exercises

Complete the following exercises.

1. Write a C++ program that prompts the user to enter values for base and height, and then calculates and displays the area of a triangle.
2. Write a C++ program that prompts the user to enter two angles of a triangle, and then calculates and displays the third angle.

   Hint: The sum of the measures of the interior angles of any triangle is 180 degrees
3. Write a C++ program that lets a student enter his or her grades from four tests, and then calculates and displays the average grade.
4. Write a C++ program that prompts the user to enter a value for radius, and then calculates and displays the perimeter of a circle.
5. Write a C++ program that prompts the user to enter the charge for a meal in a restaurant, and then calculates and displays the amount of a 10% tip, 7% sales tax, and the total of all three amounts.
6. A car starts from rest and moves with a constant acceleration along a straight horizontal road for a given time in seconds. Write a C++ program that prompts the user to enter the acceleration (in m/sec^2) and the time traveled (in sec) and then calculates the distance traveled. The required formula is

$$S = u_o + \frac{1}{2}at^2$$

7. Write a C++ program that prompts the user to enter a temperature in degrees Fahrenheit, and then converts it into its degrees Celsius[3] equivalent. The required formula is

$$\frac{C}{5} = \frac{F - 32}{9}$$

[3] Anders Celsius (1701–1744) was a Swedish astronomer, physicist, and mathematician. He founded the Uppsala Astronomical Observatory in Sweden and proposed the Celsius temperature scale, which takes his name.

8. The Body Mass Index (BMI) is often used to determine whether a person is overweight or underweight for his or her height. The formula used to calculate the BMI is

$$BMI = \frac{weight \cdot 703}{height^2}$$

   Write a C++ program that prompts the user to enter his or her weight (in pounds) and height (in inches), and then calculates and displays the user's BMI.

9. Write a C++ program that prompts the user to enter the subtotal and gratuity rate (on a scale of 0 to 100) and then calculates the tip and total. For example if the user enters 30 and 10, the C++ program should display "Tip is $3.00 and Total is $33.00".

10. An employee needs a program to enter the before-tax price of three products and then calculate the final after-tax price of each product, as well as their average value. Assume a value added tax (VAT) of 20%.

11. An employee needs a program to enter the after-tax price of a product, and then calculate its before-tax price. Assume a value added tax (VAT) of 20%.

12. Write a C++ program that prompts the user to enter the initial price of an item and the discount offered as a percentage (on a scale of 0 to 100), and then calculates and displays the final price and the amount of money saved.

13. Write a C++ program that prompts the user to enter the electric meter reading in kilowatt-hours (kWh) at the beginning and end of a month. The program should calculate and display the amount of kWh consumed and the amount of money that must be paid given a cost of each kWh of $0.06 and a value added tax (VAT) of 20%.

14. Write a C++ program that prompts the user to enter two numbers, which correspond to current month and current day of the month, and then calculates the number of days until the end of the year. Assume that each month has 30 days.

Chapter 11
Manipulating Numbers

11.1 Introduction

Just like every high-level programming language, C++ provides many functions (methods) that can be used whenever and wherever you wish. Functions are nothing more than a block of statements packaged as a unit that has a name and performs a specific task.

To better understand functions, let's take Heron's[1] iterative formula that calculates the square root of a positive number.

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{y}{x_n}\right)$$

where

- ***y*** is the number for which you want to find the square root
- x_n is the *n*-th iteration value of the square root of *y*

Please don't be disappointed! No one at the present time calculates the square root of a number this way. Fortunately, C++ includes a function for that purpose! This function, which is actually a small subprogram, has been given the name `sqrt` and the only thing you have to do is call it by its name and it will do the job for you. Function `sqrt` probably uses Heron's iterative formula, or perhaps a formula of another ancient or modern mathematician. The truth is that you don't really care! What really matters is that `sqrt` gives you the right result! An example is shown here.

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

int main() {
  double x, y;

  cin >> x;
  y = sqrt(x);
  cout << y;
```

[1] Heron of Alexandria (c. 10–c. 70 AD) was an ancient Greek mathematician, physicist, astronomer, and engineer. He is considered the greatest experimenter of ancient times. He described the first recorded steam turbine engine, called an "aeolipile" (sometimes called a "Hero engine"). Heron also described a method of iteratively calculating the square root of a positive number. Today, though, he is known best for the proof of "Heron's Formula" which finds the area of a triangle from its side lengths.

```
  return 0;
}
```

***Notice**: Please note that the function `sqrt()` is defined in the library `cmath`. Therefore, in order to use the function `sqrt`, you must include the library `cmath` using the statement `#include <cmath>` at the beginning of your program.*

Even though C++ supports many mathematical functions, this chapter covers only those absolutely necessary for this book's purpose. However, if you need even more information you can visit

http://www.cplusplus.com/reference/cmath/

***Notice**: C++ mathematical functions are used whenever you need to calculate a square root, the sine, the cosine, an absolute value, and so on.*

11.2 Useful Mathematical Functions

Absolute value

```
abs( number )
```

This returns the absolute value of *number*. It is defined in the `cmath` library.

Example

project_11_2a

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  int a, b;

  a = -5;
  b = abs(a);

  cout << abs(a) << endl;          //outputs: 5
  cout << b << endl;               //outputs: 5
  cout << abs(-5.2) << endl;       //outputs: 5.2
  cout << abs(5.2) << endl;        //outputs: 5.2
  return 0;
}
```

Cosine

```
cos( number )
```

This returns the cosine of *number*. The value of *number* must be expressed in radians. It is defined in the `cmath` library.

Example

project_11_2b

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  double a, p;

  p = 3.14159265;
  a = cos(2 * p);
  cout << a << endl;              //outputs: 1
  return 0;
}
```

Integer value

```
(int)number
```

This returns the integer value of *number*.

Example

project_11_2c

```
#include <iostream>
using namespace std;
int main() {
  double a = 5.4;

  cout << (int)a << endl;         //outputs: 5
  cout << (int)34 << endl;        //outputs: 34
  cout << (int)34.9 << endl;      //outputs: 34
  cout << (int)-34.999 << endl;   //outputs: -34
  return 0;
}
```

***Notice**: In computer science, type casting is a way of converting a variable of one data type into another. Please note that* `(int)` *is not a function. It is just a way in C++ to turn a real into an integer. Also note that if a real contains a fractional part, that part is lost during conversion.*

Real value

```
(double)number
```

This returns the *number* as real.

Example

project_11_2d

```
#include <iostream>
using namespace std;
int main() {
```

```
  int a = 5;

  cout << (double)a << endl;       //outputs: 5.0
  cout << (double)34 << endl;      //outputs: 34.0
  cout << (double)-34 << endl;     //outputs: -34.0
  cout << a / 2 << endl;           //outputs: 2
  cout << (double)a / 2 << endl;   //outputs: 2.5
  return 0;
}
```

***Notice**: In computer science, type casting is a way of converting a variable of one data type into another. Please note that* `(double)` *is not a function. It is just a way in C++ to turn an integer into a real.*

***Notice**: In C++, the result of the division of two integers is always an integer. Thus, in the expression* `a / 2`*, since variable* `a` *and number 2 are integers, the results is always an integer. As shown in the last statement, if you wish a result of type real you need to add the* `(double)` *casting operator in front of at least one of the operands of the division.*

Moreover, the statement

```
cout << (double)a / 2 << endl;
```

is equivalent to

```
cout << a / (double)2 << endl;
```

and to

```
cout << (double)a / (double)2 << endl;
```

and of course to

```
cout << a / 2.0 << endl;
```

They all output the value of 2.5.

Pi

```
M_PI
```

This returns the value of π. It is defined in the `cmath` library.

Example

project_11_2e

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  cout << M_PI << endl;    //outputs: 3.14159
  cout.precision(20);      //Set the maximum number of digits to display
  cout << M_PI << endl;    //outputs: 3.1415926535897931159
  return 0;
```

```
}
```

***Notice**: Please note that `M_PI` is a constant, not a function. This is why no parentheses are used.*

***Notice**: The statement `cout.precision(20)` sets the maximum number of digits that the statement `cout` displays.*

Power

```
pow( number, exp )
```

This returns the result of *number* raised to the power of *exp*. It is defined in the `cmath` library.

Example

project_11_2f

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  double a, b, c;

  a = 2;
  b = 3;
  cout << pow(a, b) << endl;      //outputs: 8.0

  c = pow(3, 2);
  cout << c << endl;              //outputs: 9.0
  return 0;
}
```

***Notice:** Function `pow()` can be used to calculate the square root of a number as well. It is known from mathematics that $\sqrt[z]{X}=X^{\frac{1}{z}}$. So, you can write `y = pow(x, 1/2)` to calculate the square root of `x` or you can even write `y = pow(x, 1/3)` to calculate the cube root of `x`, and so on!*

Random

```
rand()
```

This returns a pseudo-random integer in the range between 0 and RAND_MAX, where RAND_MAX is a constant of a value of at least 32767. Function `rand()` is defined in the `cstdio` library.

Example

project_11_2g

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

```
int main() {
  //output a random integer between 0 and RAND_MAX
  cout << rand()<< endl;

  //output a random integer between 0 and RAND_MAX
  cout << rand()<< endl;

  //output a random integer between 0 and RAND_MAX
  cout << rand()<< endl;

  return 0;
}
```

Pseudo-random numbers appear to be random but actually they are not! If you execute the previous program you will find that each time you execute it, you get exactly the same sequence of "random" numbers.

This is the same as getting an already mixed up deck of cards and picking the first card in order, then the second one, then the third one, and so on. Then you put the cards back on the deck in the same order as they were before picking them. Then you start again and you pick the first card in order, then the second one, then the third one, and so on. Obviously these three cards are the same as those that you picked previously. So what you need to do here is to always mix the cards up before you start picking them.

In C++ you need to do the same. You need to "mix the cards up." To do so, you can use the `srand(time(NULL))` statement at the beginning of your program as shown in the example that follows.

Example

project_11_2h

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;
int main() {

  srand (time(NULL));

  //output a random integer between 0 and RAND_MAX
  cout << rand() << endl;

  //output a random integer between 0 and RAND_MAX
  cout << rand() << endl;

  //output a random integer between 0 and RAND_MAX
  cout << rand() << endl;

  return 0;
```

```
}
```

Notice: *Please note that function* `time()` *is defined in the library* `ctime`.

If you want a pseudo-random integer between *minimum* and *maximum* you can use the following pattern

minimum + rand() % (*maximum* - *minimum* + 1)

project_11_2i

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;
int main() {

  srand (time(NULL));

  //output a random integer between 2 and 10
  cout << 2 + rand() % (10 - 2 + 1) << endl;

  //output a random integer between 20 and 25
  cout << 20 + rand() % 6 << endl;

  return 0;
}
```

Notice*: Random numbers are widely used in computer games. For example, an "enemy" may show up at a random time or move in random directions. Moreover, random numbers are used in simulation programs, in statistical programs, in computer security to encrypt data, and so on.*

Round

```
round( number )
```

This returns the closest integer. It is defined in the `cmath` library.

If you want the rounded value of *number* to a specified *precision*, you can use the following formula:

round(*number* * pow(10, *precision*)) / pow(10, *precision*)

Example

project_11_2j

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  double a, y;
```

```
  a = 5.9;
  cout << round(a) << endl;                    //outputs:6
  cout << round(5.4) << endl;                  //outputs:5

  a = 5.312;
  y = round(a * pow(10, 2)) / pow(10, 2);
  cout << y << endl;                           //outputs:5.31

  a = 5.315;
  y = round(a * pow(10, 2)) / pow(10, 2);
  cout << y << endl;                           //outputs:5.32

  //outputs: 2.345
  cout << round(2.3447 * 1000) / 1000;
  return 0;
}
```

Sine

```
sin( number )
```

This returns the sine of *number*. The value of *number* must be expressed in radians. Function `sin()` is defined in the `cmath` library.

Example

project_11_2k

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  double a;

  a = sin(3 * M_PI / 2);
  cout << a << endl;                           //outputs: -1.0
  return 0;
}
```

Square root

```
sqrt( number )
```

This returns the square root of *number*. It is defined in the `cmath` library.

Example

project_11_21

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  cout << sqrt(9) << endl;        //outputs: 3
```

```
  cout << sqrt(2) << endl;          //outputs: 1.41421
  cout.precision(15);
  cout << sqrt(2) << endl;          //outputs: 1.4142135623731
  return 0;
}
```

Remember! *The statement* `cout.precision(15)` *sets the maximum number of digits that the statement* `cout` *displays.*

Tangent

```
tan( number )
```

This returns the tangent of *number*. The value of *number* must be expressed in radians. Function `tan()` is defined in the `cmath` library.

Example

project_11_2m

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  double a;

  a = tan(10);
  cout << a << endl;                    //outputs: 0.648361
  return 0;
}
```

Exercise 11.2-1 *Calculating the Distance Between Two Points*

Write a C++ program that prompts the user to enter the coordinates (x, y) of two points and then calculates the straight line distance between them. The required formula is

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Solution

In this exercise, you need to use two functions: the `pow()` function, which returns the result of a number raised to a power; and the `sqrt()` function, which returns the square root of a number.

To simplify things, the terms $(x_1 - x_2)^2$ and $(y_1 - y_2)^2$ are calculated individually and the results are assigned to two temporary variables.

The C++ program is shown here.

project_11_2_1a

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

```
int main() {
  double d, x1, x2, x_temp, y1, y2, y_temp;

  cout << "Enter coordinates for point A: ";
  cin >> x1;
  cin >> y1;

  cout << "Enter coordinates for point B: ";
  cin >> x2;
  cin >> y2;

  x_temp = pow(x1 - x2, 2);
  y_temp = pow(y1 - y2, 2);

  d = sqrt(x_temp + y_temp);

  cout << "Distance between points: " << d;
  return 0;
}
```

Now let's see another approach.

You should realize that it is actually possible to nest one function within another. Doing that, the result of the nested function is used as an argument for the outer function. This is a writing style that most programmers prefer to follow because it can save a lot of code lines. Of course, if you nest too many functions, no one will be able to understand your code. A nesting of up to four levels is quite acceptable.

The C++ program that uses nested functions is shown here.

project_11_2_1b

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  double d, x1, x2, y1, y2;

  cout << "Enter coordinates for point A: ";
  cin >> x1;
  cin >> y1;

  cout << "Enter coordinates for point B: ";
  cin >> x2;
  cin >> y2;

  d = sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));

  cout << "Distance between points: " << d;
  return 0;
```

```
}
```

Exercise 11.2-2 *How Far Did the Car Travel?*

A car starts from rest and moves with a constant acceleration along a straight horizontal road for a given distance. Write a C++ program that prompts the user to enter the acceleration and the distance the car traveled and then calculates the time traveled. The required formula is

$$S = u_o + \frac{1}{2}at^2$$

where

- ***S** is the distance the car traveled, in meters (m)*
- ***u_o** is the initial velocity (speed) of the car, in meters per second (m/sec)*
- ***t** is the time the car traveled, in seconds (sec)*
- ***a** is the acceleration, in meters per second^2 (m/sec^2)*

Solution

Since the car starts from rest, the initial velocity (speed) u_0 is zero. Thus, the formula becomes

$$S = \frac{1}{2}at^2$$

Now, if you solve for time, the final formula becomes

$$t = \sqrt{\frac{2S}{a}}$$

In C++, you can use the `sqrt()` function, which returns the square root of a number.

project_11_2_2

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  double S, a, t;

  cout << "Enter acceleration: ";
  cin >> a;
  cout << "Enter distance traveled: ";
  cin >> S;

  t = sqrt(2 * S / a);

  cout << "Your car traveled for " << t << " seconds";
  return 0;
}
```

11.3 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. In general, functions are small subprograms that solve small problems.
2. Every programmer must use Heron's iterative formula to calculate the square root of a positive number.
3. The `abs()` function returns the absolute position of an item.
4. The statement `(int)3.59` returns a result of 3.6.
5. The `M_PI` constant is equal to 3.14.
6. The statement `pow(2, 3)` returns a result of 9.
7. The `rand()` function can also return negative random numbers.
8. There is a 50% possibility that the statement `y = rand() % 2` will assign a value of 1 to variable `y`.
9. The statement `round(3.59)` returns a result of 4.
10. To calculate the sinus of 90 degrees, you have to write `y = sin (M_PI / 2)`.
11. The statement `y = sqrt(-2)` is valid.
12. The following code fragment satisfies the property of definiteness.

```
double a, b, x;

cin >> a;
cin >> b;
x = a * sqrt(b);
cout << x << endl;
```

11.4 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. Which of the following calculates the result of the variable `a` raised to the power of 2?
   a. `y = a * a;`
   b. `y = pow(a, 2);`
   c. `y = a * a / a * a;`
   d. all of the above
2. What is the value of the variable `y` when the statement `y = abs(+5.2)` is executed?
   a. -5.2
   b. -5
   c. 0.2
   d. 5.2
   e. 5

f. none of the above

3. Which of the following calculates the sinus of 180 degrees?
   a. `sin(180)`
   b. `sin(M_PI)`
   c. all of the above
   d. none of the above
4. What is the value of the variable `y` when the statement `y = (int)(5/2)` is executed?
   a. 2.5
   b. 3
   c. 2
   d. 0.5
5. What is the value of the variable `y` when the statement `y = pow(sqrt(4), 2)` is executed?
   a. 4
   b. 2
   c. 8
   d. 16
6. What is the value of the variable `y` when the statement `y = round(5.2)/2.0` is executed?
   a. 2
   b. 2.5
   c. 2.6
   d. none of the above

11.5 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the C++ program for two different executions.

   The input values for the two executions are: (i) 9, and (ii) 4.

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  double a;
  int b, c;

  cin >> a;

  a += 6 / sqrt(a) * 2 + 20;
```

```
  b = (int)round(a) % 4;
  c = b % 3;

  cout << a << ", " << b << ", " << c;
  return 0;
}
```

2. Create a trace table to determine the values of the variables in each step of the C++ program for two different executions.

   The input values for the two executions are: (i) -2, and (ii) -3

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
  int a, b, c;

  cin >> a;

  b = (int)abs(a) % 4 + (int)pow(a, 4);
  c = b % 5;

  cout << b << ", " << c;
  return 0;
}
```

3. Write a C++ that prompts the user to enter an angle θ in radians and then calculates and displays the angle in degrees. It is given that $2\pi = 360^{o}$.
4. Write a C++ program that calculates the hypotenuse of a right-angled triangle, given its two right angle sides A and B. It is known from the Pythagorean[2] theorem that

$$hypotenuse = \sqrt{A^2 + B^2}$$

5. Write a C++ program that prompts the user to enter the angle θ (in degrees) of a right-angled triangle and the length of its adjacent side, and then calculates the length of the opposite side. It is known that $2\pi = 360^{o}$, and

$$\tan(\theta) = \frac{Opposite}{Adjacent}$$

[2] Pythagoras of Samos (c. 571–c. 497 BC) was a famous Greek mathematician, philosopher, and astronomer. He is best known for the proof of the important Pythagorean theorem. He was an influence for Plato. His theories are still used in mathematics today.

Chapter 12
Complex Mathematical Expressions

12.1 Writing Complex Mathematical Expressions

In Chapter 7 you learned all about arithmetic operators but little about how to use them and how to write your own complex mathematical expressions. In this chapter, you are going to learn how easy is to convert mathematical expressions to C++ statements.

> ***Remember!*** *Arithmetic operators follow the same precedence rules as in mathematics, which means that multiplication and division are performed before addition and subtraction. Moreover, when multiplication and division co-exist in the same expression, and since both are of the same precedence, these operations are performed left to right.*

Exercise 12.1-1 *Representing Mathematical Expressions in C++*

Which of the following C++ statements correctly represent the following mathematical expression?

$$x = \frac{1}{10 + z}27$$

i. `x = 1 * 27 / 10 + z`

ii. `x = 1 · 27 / (10 + z)`

iii. `x = 27 / 10 + z`

iv. `x = 27 / (10 + z)`

v. `x = (1 / 10 + z) * 27`

vi. `x = 1 / ((10 + z) * 27)`

vii. `x = 1 / ( 10 + z) * 27`

viii. `x = 1 / ( 10 + z) / 27`

Solution

i. **Wrong**. Since the multiplication and the division are performed before the addition, this is equivalent to $x = \frac{1}{10}27 + z$.

ii. **Wrong**. An asterisk should have been used for multiplication.

iii. **Wrong**. Since the division is performed before the addition, this is equivalent to $x = \frac{27}{10} + z$.

iv. **Correct**. This is equivalent to $x = \frac{27}{10 + z}$.

v. **Wrong**. Inside parentheses, the division is performed before the addition. This is equivalent to $x = \left(\frac{1}{10} + z\right)27$.

vi. **Wrong**. Parentheses are executed first and this is equivalent to $x = \frac{1}{(10+z)27}$.

vii. **Correct**. Division is performed before multiplication (left to right). The term $\frac{1}{10+z}$ is calculated first and then, the result is multiplied by 27.

viii. **Wrong**. This is equivalent to $= \frac{\frac{1}{10+z}}{27}$

Exercise 12.1-2 *Writing a Mathematical Expression in C++*

Write a C++ program that calculates the mathematical expression

$$y = 10\,x - \frac{10 - z}{4}$$

Solution

First, you must distinguish between the data input and the output result. Obviously, the output result is assigned to y and the user must enter values for x and z. The solution for this exercise is shown here.

project_12_1_2

```
#include <iostream>
using namespace std;
int main() {
  double x, y, z;

  cout << "Enter value for x: ";
  cin >> x;
  cout << "Enter value for z: ";
  cin >> z;

  y = 10 * x - (10 - z) / 4;

  cout << "The result is: " << y;
  return 0;
}
```

Exercise 12.1-3 *Writing a Complex Mathematical Expression in C++*

Write a C++ program that calculates the mathematical expression

$$y = \frac{5\dfrac{3x^2 + 5x + 2}{7w - \dfrac{1}{z}} - z}{4\dfrac{3 + x}{7}}$$

Solution

Oops! Now the expression is more complex! In fact, it is much more complex! So, let's take a look at a quite different approach.

The main idea is to break the complex expression into smaller, simpler expressions and assign each sub-result to temporary variables. In the end, you can build the original expression out of all these temporary variables! This approach is presented here.

project_12_1_3a

```
#include <iostream>
using namespace std;
int main() {
  double denominator, nominator, temp1, temp2, temp3, w, x, y, z;

  cout << "Enter value for x: ";
  cin >> x;
  cout << "Enter value for w: ";
  cin >> w;
  cout << "Enter value for z: ";
  cin >> z;

  temp1 = 3 * x * x + 5 * x + 2;
  temp2 = 7 * w - 1 / z;
  temp3 = (3 + x) / 7;
  nominator = 5 * temp1 / temp2 - z;
  denominator = 4 * temp3;

  y = nominator / denominator;

  cout << "The result is: " << y;
  return 0;
}
```

You may say, "*Okay, but I wasted so many variables and as everybody knows, each variable is a portion of main memory. How can I write the original expression in one single line and waste less memory?*"

This job may be a piece of cake for an advanced programmer, but what about you? What about a novice programmer?

The next method will help you write even the most complex mathematical expressions without any syntax or logic errors! The rule is very simple. "*After breaking the complex expression into smaller, simpler expressions and assigning each sub-result to temporary variables, start backwards and replace each variable with its assigned expression. Be careful though! When you replace a variable with an expression, you must always enclose the expression in parentheses!*"

Confused? Don't be! It's easier in action. Let's try to rewrite the previous C++ program. Starting backwards, replace variables `nominator` and `denominator` with their assigned expressions. The result is

```
y = (5 * temp1 / temp2 - z) / (4 * temp3);
```

nominator denominator

***Notice**: Please note the extra parentheses added.*

Now you must replace variables `temp1`, `temp2`, and `temp3` with their assigned expressions, and the one-line expression is complete!

```
y = (5 * (3 * x * x + 5 * x + 2)/(7 * w - 1 / z) - z)/(4 * ((3 + x) / 7));
```

temp1 temp2 temp3

It may look scary at the end but it wasn't that difficult, was it?

The C++ program can now be rewritten

project_12_1_3b

```
#include <iostream>
using namespace std;
int main() {
  double w, x, y, z;

  cout << "Enter value for x: ";
  cin >> x;
  cout << "Enter value for w: ";
  cin >> w;
  cout << "Enter value for z: ";
  cin >> z;

  y = (5 * (3 * x * x + 5 * x + 2)/(7 * w - 1 / z) - z)/(4 * ((3 + x) / 7));

  cout << "The result is: " << y;
  return 0;
}
```

12.2 Review Exercises

Complete the following exercises.

1. Match each element from the first table with one **or more** elements from the second table.

Expression
i. `5 / pow(x, 2) * y + pow(x,3)`
ii. `5 / (pow(x, 3) * y) + pow(x, 2)`

	Expression
a.	`5 * y / pow(x, 2) + pow(x, 3)`
b.	`5 * y / x * x + pow(x, 3)`
c.	`5 / (x * x * x * y) + x * x`
d.	`5 / (x * x * x) * y + x * x`
e.	`5 * y / (x * x) + x * x * x`
f.	`1 / (x * x * x * y) * 5 + x * x`
g.	`y / (x * x) * 5 + pow(x, 3)`
h.	`1 / (x * x) * 5 * y + x / x * x * x`

2. Write the following mathematical expressions in C++ using one line of code for each.

i. $y = \frac{(x+3)^{5w}}{7(x-4)}$

ii. $y = \sqrt[5]{\left(3x^2 - \frac{1}{4}x^3\right)}$

iii. $y = \frac{\sqrt{x^4-2x^3-7x^2+x}}{\sqrt[3]{4\left(7x^4-\frac{3}{4}x^3\right)(7x^2+x)}}$

iv. $y = \frac{x}{x-3(x-1)} + \left(x\sqrt[5]{x-1}\right)\frac{1}{(x^3-2)(x-1)^3}$

v. $y = \left(\sin\left(\frac{\pi}{3}\right) - \cos\left(\frac{\pi}{2}w\right)\right)^2$

vi. $y = \frac{\left(\sin\left(\frac{\pi}{2}x\right)+\cos\left(\frac{3\pi}{2}w\right)\right)^3}{\left(\tan\left(\frac{2\pi}{3}w\right)-\sin\left(\frac{\pi}{2}x\right)\right)^{\frac{1}{2}}} + 6$

3. Write a C++ program that prompts the user to enter a value for *x* and then calculates and displays the result of the following mathematical expression.

$$y = \sqrt{x}(x^3 + x^2)$$

4. Write a C++ program that prompts the user to enter a value for *x* and then calculates and displays the result of the following mathematical expression.

$$y = \frac{7x}{2x + 4(x^2 + 4)}$$

Suggestion: Try to write the expression in one line of code.

5. Write a C++ program that prompts the user to enter a value for *x* and *w* and then calculates and displays the result of the following mathematical expression.

$$y = \frac{x^{x+1}}{\left(\tan\left(\frac{2w}{3}+5\right) - \tan\left(\frac{x}{2}+1\right)\right)^3}$$

Suggestion: Try to write the expression in one line of code

6. Write a C++ program that prompts the user to enter a value for `x` and `w` and then calculates and displays the result of the following mathematical expression.

$$y = \frac{3+w}{6x - 7(x+4)} + \left(x\sqrt[5]{3w+1}\right)\frac{5x+4}{(x^3+3)(x-1)^7}$$

Suggestion: Try to write the expression in one line of code.

7. Write a C++ program that prompts the user to enter a value for `x` and `w` and then calculates and displays the result of the following mathematical expression.

$$y = \frac{x^x}{\left(\sin\left(\frac{2w}{3}+5\right) - x\right)^2} + \frac{(\sin(3x)+\mathrm{w})^{x+1}}{\left(\sqrt{7w}\right)^{\frac{3}{2}}}$$

Suggestion: Try to write the expression in one line of code

8. Write a C++ program that prompts the user to enter the lengths of all three sides A, B, and C, of a triangle and then calculates and displays the area of the triangle. You can use Heron's formula, which has been known for nearly 2,000 years!

$$Area = \sqrt{S(S-A)(S-B)(S-C)}$$

where S is the semi-perimeter

$$S = \frac{A+B+C}{2}$$

Chapter 13
Exercises With a Quotient and a Remainder

13.1 Introduction

What types of problems might require the use of the quotient and the remainder of an integer division? There is no simple answer to that question! However, quotients and remainders can be used to:

- split a number into individual digits
- examine if a number is odd or even
- convert an elapsed time (in seconds) to hours, minutes, and seconds
- convert an amount of money (in USD) to a number of $100 notes, $50 notes, $20 notes, and such
- determine if a number is a palindrome
- count the number of digits within a number
- determine how many times a specific digit occurs within a number

Of course, these are some of the uses and certainly you can find so many others. Next you will see some exercises that make use of the quotient and the remainder of integer division.

Exercise 13.1-1 *Calculating the Quotient and Remainder of Integer Division*

Write a C++ program that prompts the user to enter two integers and then calculates the quotient and the remainder of the integer division.

Solution

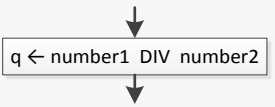
Since C++ doesn't actually incorporate an arithmetic operator that calculates the integer quotient, you can use the `(int)` casting operator to achieve the same result.

project_13_1_1

```
#include <iostream>
using namespace std;
int main() {
  int number1, number2, q, r;

  cout << "Enter first number: ";
  cin >> number1;
  cout << "Enter second number: ";
  cin >> number2;
```

```
    q = (int)(number1 / number2);
    r = number1 % number2;
    cout << "Integer Quotient: " << q << "\nInteger Remainder: " << r;
    return 0;
}
```

***Notice**: In flowcharts, in order to calculate the quotient of an integer division, you can use the popular* `DIV` *operator. An example is shown here.*



***Notice**: In C++, the result of the division of two integers is always an integer. Thus, in the statement* `q = (int)(number1 / number2)`*, since variables* `number1` *and* `number2` *are integers, the* `(int)` *casting operator is redundant. However, it is a good practice to keep it there just for improved readability.*

Exercise 13.1-2 *Finding the Sum of Digits*

Write a C++ program that prompts the user to enter a four-digit integer and then calculates the sum of its digits.

Solution

What you should keep in mind here is that the statement

```
cin >> number;
```

in the beginning of the program assigns the given four-digit integer to one single variable, `number`, and not to four individual variables.

So, first you must split the integer into its four digits and assign each digit to a separate variable. Then you can calculate the sum of these four variables and get the required result. There are two approaches available.

First Approach

Let's try to understand the first approach using an arithmetic example. Take the number 6753, for example.

First digit = 6	The first digit can be isolated if you divide the given number by 1000 to get the integer quotient `digit1 = (int)(6753 / 1000)`
Remaining digits = 753	The remaining digits can be isolated if you divide the given number by 1000 to get the integer remainder `r = 6753 % 1000`

Second digit = 7	The second digit can be isolated if you divide the remaining digits by 100 to get the integer quotient `digit2 = (int)(753 / 100)`
Remaining digits = 53	The remaining digits are now `r = 753 % 100`
Third digit = 5	The third digit can be isolated if you divide the remaining digits by 10 to get the integer quotient `digit3 = (int)(53 / 10)`
Fourth digit = 3	The last remaining digit, which happens to be the fourth digit, is `digit4 = 53 % 10`

The C++ program that solves this algorithm is shown here.

project_13_1_2a

```
#include <iostream>
using namespace std;
int main() {
  int digit1, digit2, digit3, digit4, number, r, sum;

  cout << "Enter a four-digit integer: ";
  cin >> number;

  digit1 = (int)(number / 1000);
  r = number % 1000;

  digit2 = (int)(r / 100);
  r = r % 100;

  digit3 = (int)(r / 10);
  digit4 = r % 10;

  sum = digit1 + digit2 + digit3 + digit4;
  cout << sum;
  return 0;
}
```

The trace table for the program that you have just seen is shown here.

Step	Statement	Notes	number	digit1	digit2	digit3	digit4	r	sum
1	`cout << "Enter a …`	The message “Enter a …” is displayed							

2	`cin >> number`	User enters 6753	**6753**	?	?	?	?	?	?
3	`digit1 = (int)(number / 1000)`		6753	**6**	?	?	?	?	?
4	`r = number % 1000`		6753	6	?	?	?	**753**	**?**
5	`digit2 = (int)(r / 100)`		6753	6	**7**	?	?	753	?
6	`r = r % 100`		6753	6	7	?	?	**53**	**?**
7	`digit3 = (int)(r / 10)`		6753	6	7	**5**	?	53	?
8	`digit4 = r % 10`		6753	6	7	5	**3**	53	?
9	`sum = digit1+digit2+…`		6753	6	7	5	**3**	53	**18**
10	`cout << sum`	Value 18 is displayed							

To further help you, there is also a general purpose C++ program that can be used to split any given integer. Since the length of your program depends on the number of digits, `N`, all you have to do is write `N-1` pairs of statements.

```
cout << "Enter an N-digit integer: ";
cin >> number;

digit1 = (int)(number / 10^(N-1));
r = number % 10^(N-1);

digit2 = (int)(r / 10^(N-2));
r = r % 10^(N-2);
.
.
.
digit(N-2) = (int)(r / 100);
r = r % 100;

digit(N-1) = (int)(r / 10);
digitN = r % 10;
```

1st pair of statements

2nd pair of statements

(N-2)th pair of statements

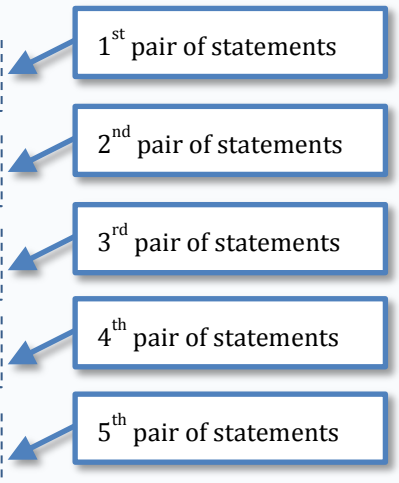
(N-1)th pair of statements

For example, if you want to split a six-digit integer, you need to write five pairs of statements.

project_13_1_2b

```
#include <iostream>
using namespace std;
int main() {
  int digit1, digit2, digit3, digit4, digit5, digit6, number, r;
```

```
  cout << "Enter an six-digit integer: ";
  cin >> number;

  digit1 = (int)(number / 100000);
  r = number % 100000;

  digit2 = (int)(r / 10000);
  r = r % 10000;

  digit3 = (int)(r / 1000);
  r = r % 1000;

  digit4 = (int)(r / 100);
  r = r % 100;

  digit5 = (int)(r / 10);
  digit6 = r % 10;

  cout << digit1 << ", " << digit2 << ", " << digit3 << ", " << endl;
  cout << digit4 << ", " << digit5 << ", " << digit6 << endl;
  return 0;
}
```



Second Approach

Once more, let's try to understand the second approach using an arithmetic example. Take the same number, 6753, for example.

Fourth digit = 3	The fourth digit can be isolated if you divide the given number by 10 to get the integer remainder `digit1 = 6753 % 10`
Remaining digits = 675	The remaining digits can be isolated if you divide the given number by 10 to get the integer quotient `r = (int)(6753 / 10)`
Third digit = 5	The third digit can be isolated if you divide the remaining digits by 10 to get the integer remainder `digit3 = 675 % 10`
Remaining digits = 67	The remaining digits are now `r = (int)(675 / 10)`
Second digit = 7	The second digit can be isolated if you divide the remaining digits by 10 to get the integer remainder `digit3 = 67 % 10`

First digit = 6

The last remaining digit, which happens to be the first digit, is

```
digit1 = (int)(67 / 10)
```

The C++ program for this algorithm is shown here.

project_13_1_2c

```
#include <iostream>
using namespace std;
int main() {
  int digit1, digit2, digit3, digit4, number, r, sum;

  cout << "Enter a four-digit integer: ";
  cin >> number;

  digit4 = number % 10;
  r = (int)(number / 10);

  digit3 = r % 10;
  r = (int)(r / 10);

  digit2 = r % 10;
  digit1 = (int)(r / 10);

  sum = digit1 + digit2 + digit3 + digit4;
  cout << sum;
  return 0;
}
```

To further help you, there is also a general purpose C++ program that can be used to split any given integer. This program uses the second approach. Once again, since the length of your program depends on the number of the digits, N, all you have to do is write N-1 pairs of statements.

```
cout << "Enter a N-digit integer: ";
cin >> number;

digit(N) = number % 10;
r = (int)(number / 10);

digit(N-1) = r % 10;
r = (int)(r / 10);
.
.
.
digit3 = r % 10;
r = (int)(r / 10);
```

1st pair of statements

2nd pair of statements

(N-2)th pair of statements

```
digit2 = r % 10;
digit1 = (int)(r / 10);
```

(N-1)th pair of statements

For example, if you want to split a five-digit integer, you must use four pairs of statements.

project_13_1_2d

```
#include <iostream>
using namespace std;
int main() {
  int digit1, digit2, digit3, digit4, digit5, number, r;

  cout << "Enter a five-digit integer: ";
  cin >> number;
  digit5 = number % 10;
  r = (int)(number / 10);
  digit4 = r % 10;
  r = (int)(r / 10);
  digit3 = r % 10;
  r = (int)(r / 10);
  digit2 = r % 10;
  digit1 = (int)(r / 10);

  cout << digit1 << ", " << digit2 << ", " << digit3 << ", " << endl;
  cout << digit4 << ", " << digit5 << endl;
  return 0;
}
```

1st pair of statements

2nd pair of statements

3rd pair of statements

4th pair of statements

Exercise 13.1-3 *Displaying an Elapsed Time*

Write a C++ program that prompts the user to enter an integer that represents an elapsed time in seconds and then displays it in the format "DD days HH hours MM minutes and SS seconds". For example if the user enters the number 700005, the message "8 days 2 hours 26 minutes and 45 seconds" must be displayed.

Solution

As you may already know, there are 60 seconds in a minute, 3600 seconds in an hour, and 86400 seconds in a day.

Let's try to analyze number 700005 using the first approach.

Days = 8	The number of days can be isolated if you divide the given integer by 86400 to get the integer quotient `days = (int)(700005 / 86400);`
Remaining seconds = 8805	The remaining seconds can be isolated if you divide the given integer by 86400 to get the integer remainder `r = 700005 % 86400;`
Hours = 2	The number of hours can be isolated if you divide the remaining seconds by 3600 to get the integer quotient `hours = (int)(8805 / 3600);`
Remaining seconds = 1605	The remaining seconds are now `r = 8805 % 3600;`
Minutes = 26	The number of minutes can be isolated if you divide the remaining seconds by 60 to get the integer quotient `minutes = (int)(1605 / 60);`
Seconds = 45	The last remainder, which happens to be the number of seconds left, is `seconds = 1605 % 60;`

The C++ program for this algorithm is as follows.

project_13_1_3

```
#include <iostream>
using namespace std;
int main() {
  int days, hours, minutes, number, r, seconds;

  cout << "Enter a period of time in seconds: ";
  cin >> number;

  days = (int)(number / 86400);   // 60 * 60 * 24 = 86400
  r = number % 86400;

  hours = (int)(r / 3600);        // 60 * 60 = 3600
  r = r % 3600;

  minutes = (int)(r / 60);
  seconds = r % 60;
```

```
  cout << days << " days " << hours << " hours " << endl;
  cout << minutes << " minutes and " << seconds << " seconds";
  return 0;
}
```

Exercise 13.1-4 *Reversing a Number*

Write a C++ program that prompts the user to enter a three-digit integer and then reverses it. For example, if the user enters the number 375, the number 573 should be displayed.

Solution

To isolate the three digits of the given number, you can use either first or second approach. Afterward, the only difficulty in this exercise is to build the reversed number. Take the number 375, for example. The three digits, after isolation, are

```
digit1 = 3;
digit2 = 7;
digit3 = 5;
```

You can build the reversed number by simply calculating the sum of the products:

$$\text{digit3} \times 100 + \text{digit2} \times 10 + \text{digit1} \times 1$$

For a change, you can split the given number using the second approach. The C++ program will look like this.

project_13_1_4

```
#include <iostream>
using namespace std;
int main() {
  int digit1, digit2, digit3, number, r, reversed;

  cout << "Enter a three-digit integer: " << endl;
  cin >> number;

  digit3 = number % 10;           //This is the rightmost digit
  r = (int)(number / 10);

  digit2 = r % 10;                //This is the digit in the middle
  digit1 = (int)(r / 10);         //This is the leftmost digit

  reversed = digit3 * 100 + digit2 * 10 + digit1;
  cout << reversed;
  return 0;
}
```

13.2 Review Exercises

Complete the following exercises.

1. Write a C++ program that prompts the user to enter any integer and then multiplies its last digit by 8 and displays the result.

   Hint: You can isolate the last digit of any integer using a modulus 10 operation.

2. Write a C++ program that prompts the user to enter a five-digit integer and then reverses it. For example, if the user enters the number 32675, the number 57623 must be displayed.

3. Write a C++ program that prompts the user to enter an integer and then it displays 1 when the number is odd; otherwise, it displays 0. Try not to use any decision control structures since you haven't learned anything about them yet!

4. Write a C++ program that prompts the user to enter an integer and then it displays 1 when the number is even; otherwise, it displays 0. Try not to use any decision control structures since you haven't learned anything about them yet!

5. Write a C++ program that prompts the user to enter an integer representing an elapsed time in seconds and then displays it in the format "WW weeks DD days HH hours MM minutes and SS seconds." For example, if the user enters the number 2000000, the message "3 weeks 23 days 3 hours 33 minutes and 20 seconds" should be displayed.

6. Inside an ATM bank machine there are notes of $20, $10, $5, and $1. Write a C++ program that prompts the user to enter the amount of money he or she wants to withdraw (using an integer value) and then displays the least number of notes the ATM should give. For example, if the user enters an amount of $76, the program should display the message "3 notes of $20, 1 note of $10, 1 note of $5, and 1 note of $1".

7. A robot arrives on the moon in order to perform some experiments. Each of the robot's steps is 25 inches long. Write a C++ program that prompts the user to enter the number of steps the robot made and then calculates and displays the distance traveled in miles, feet, yards, and inches. For example, if the distance traveled is 100000 inches, the program should display the message "1 mile, 1017 yards, 2 feet, and 4 inches".

   It is given that

   - 1 mile = 63360 inches
   - 1 yard = 36 inches
   - 1 foot = 12 inches

Chapter 14
Manipulating Strings

14.1 Introduction

Generally speaking, a string is anything that you can type using the keyboard, including letters, symbols (such as &, *, and @), and digits. Sometimes a program deals with data that comes in the form of strings (text). In C++, a string is always enclosed in double quotes like this: “ ”.

Each statement in the next example outputs a string.

```
#include <iostream>
using namespace std;
int main() {
  cout << "Everything enclosed in double quotes is a ";
  cout << "string, even the numbers below:" << endl;
  cout << "3, 4, 7" << endl;
  cout << "You can even mix letters, symbols and ";
  cout << "numbers like this:" << endl;
  cout << "The result of 3 + 4 equals to 4";
  return 0;
}
```

Strings are everywhere—from word processors, to web browsers, to text messaging programs. Many exercises in this book actually make extensive use of strings. Even though C++ supports many useful functions for manipulating strings, this chapter covers only those functions that are absolutely necessary for this book’s purpose. However, if you need even more information you can visit

http://www.cplusplus.com/reference/string/string/

***Notice**: C++ string functions can be used when there is a need, for example, to isolate a number of characters from a string, to remove spaces that might exist at the beginning of it, or to convert all of its characters to uppercase.*

***Remember**! Functions are nothing more than small subprograms that solve small problems. A subprogram can be defined as a block of statements packaged as a unit that has a name and performs a specific task.*

Moreover, even though there are many useful string functions that C++ supports, some of the ones this book uses are not part of the standard C++; rather, they are defined in the Boost C++ Libraries. In this case, each C++ project that uses any of those functions must include these libraries. This can be done if, from the main menu, you select “File→Project Properties” and in category “C++ Compiler” you include the directory (the folder) of your Boost C++ Libraries as shown in **Figure 14–1**.

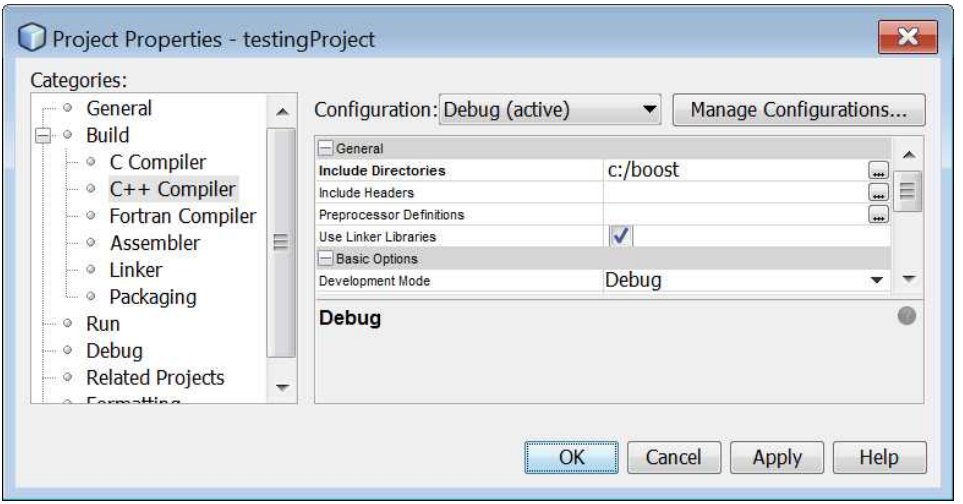


Figure 14–1 Including the Boost C++ Libraries in a C++ project

14.2 The Position of a Character in a String

Let's use the text «Hello World» for the following example. The string consists of 11 characters (including the space character between the two words). The position of each character is shown here.

0	1	2	3	4	5	6	7	8	9	10
H	**e**	**l**	**l**	**o**		**W**	**o**	**r**	**l**	**d**

C++ numerates characters assuming that the first one is at position 0, the second one is at position 1, and so on. Please note that the space between the two words is considered a character as well.

Remember! *A space is a character just like any other character. Just because nobody can see it, it doesn't mean it doesn't exist!*

14.3 Retrieving an Individual Character From a String

C++ allows you to retrieve the individual characters of a string using substring notation. You can use index 0 to access the first character, index 1 to access the second character, and so on. The index of the last character is 1 less than the length of the string. The following C++ program shows an example.

project_14_3

```
#include <iostream>
using namespace std;
int main() {
  string a;

  a = "Hello World";
```

```
   cout << a.at(0) << endl;         //it displays the first letter
   cout << a.at(6) << endl;         //it displays the letter W
   cout << a.at(10) << endl;        //it displays the last letter
   return 0;
}
```

***Notice**: Please note that the space between the words "Hello" and "World" is considered a character as well. So, the letter W exists in position 6 and not in position 5.*

If you attempt to use an invalid index such as a negative one or an index greater than the length of the string, C++ displays an error message as shown in **Figure 14–2**.

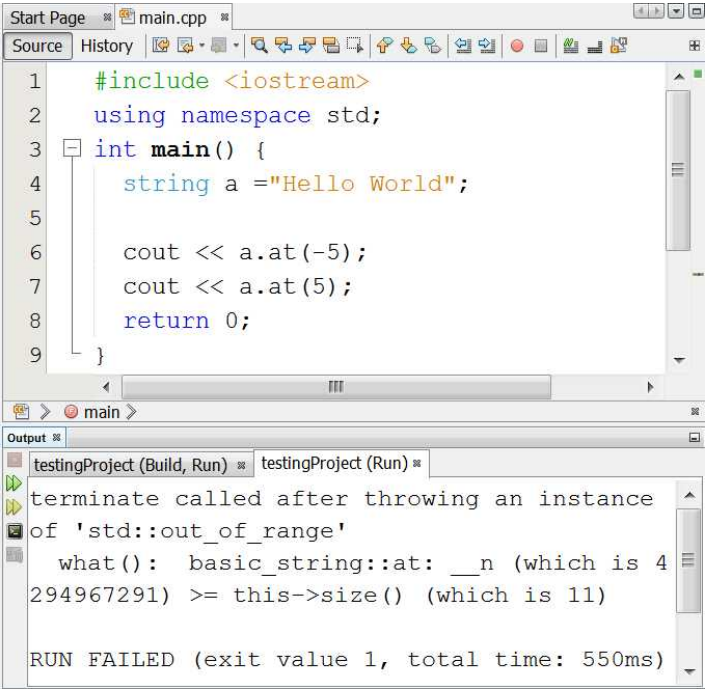


Figure 14–2 An error message indicating an invalid index

***Remember**! String indexes must be in a range from 0 to one less than the length of the string.*

Exercise 14.3-1 *Displaying a String Backwards*

Write a C++ program that prompts the user to enter any string with four letters and then displays its contents backwards. For example, if the string entered is "Zeus", the program should display "sueZ".

Solution

Let's say that user's input is assigned to variable `str`. You can access the fourth letter using `str.at(3)`, the third letter using `str.at(2)`, and so on.

The C++ program is shown here.

project_14_3_1

```
#include <iostream>
using namespace std;
int main() {
  string str;

  cout << "Enter a word with four letters: ";
  cin >> str;

  cout << str.at(3) << str.at(2) << str.at(1) << str.at(0);
  return 0;
}
```

14.4 Useful String Functions

Trimming

Trimming is the process of removing whitespace characters from the beginning or the end of a string.

Some of the whitespace characters that are removed with the trimming process are:

- " " an ordinary space
- "\t" a tab
- "\n" a new line (line feed)
- "\r" a carriage return

For example, you can trim any spaces that the user mistakenly entered at the end of a string.

The function that you can use to trim a string is

```
trim_copy( subject )
```

This removes any whitespace characters from both the beginning and the end of *subject*. It is defined in the `boost` library.

Example

project_14_4a

```
#include <iostream>
#include <boost/algorithm/string.hpp>
using namespace boost::algorithm;
using namespace std;
int main() {
  string a, b;

  a = "       Hello          ";
  b = trim_copy(a);
```

```
  cout << b << " Poseidon!" << endl;              //Outputs: Hello
Poseidon!
  return 0;
}
```

String replacement

```
replace_all_copy( subject, search, replace )
```

This searches in *subject* and replaces all occurrences of the *search* string with the *replace* string. It is defined in the `boost` library.

Example

project_14_4b

```
#include <iostream>
#include <boost/algorithm/string.hpp>
using namespace boost::algorithm;
using namespace std;
int main() {
  string a, b;

  a = "I am newbie in Java. Java rocks!";
  b = replace_all_copy(a, "Java", "C++");

  cout << b << endl;       //Outputs: I am newbie in C++. C++ rocks
  return 0;
}
```

Counting the number of characters

```
subject.length()
```

This returns the length of *subject* or, in other words, the number of characters *subject* consists of (including space characters, symbols, numbers, and so on).

Example

project_14_4c

```
#include <iostream>
using namespace std;
int main() {
  string a, b;

  a = "Hello Olympians!";
  cout << a.length() << endl;     //Outputs: 16

  b = "I am newbie in C++";
  cout << b.length() << endl;     //Outputs: 18
  return 0;
}
```

Finding string position

```
subject.find( search )
```

This finds the numeric position of the first occurrence of *search* in *subject*.

Example

project_14_4d

```
#include <iostream>
using namespace std;
int main() {
  int i;
  string a;

  a = "I am newbie in C++. C++ rocks!";
  i = a.find("newbie");

  cout << i << endl;       //Outputs: 5
  return 0;
}
```

***Remember**! The first character is at position 0.*

Converting to lowercase or uppercase

There are two functions that let you convert all letters in a string to lowercase or uppercase.

To lowercase

```
to_lower_copy( subject )
```

This returns the *subject*, converted to lowercase. It is defined in the `boost` library.

Example

project_14_4e

```
#include <iostream>
#include <boost/algorithm/string.hpp>
using namespace boost::algorithm;
using namespace std;
int main() {
  string a, b;

  a = "My NaMe is JohN";
  b = to_lower_copy(a);

  cout << b << endl;       //Outputs: my name is john
  return 0;
}
```

To uppercase

```
to_upper_copy( subject )
```

This returns the *subject*, converted to uppercase. It is defined in the `boost` library.

Example

project_14_4f

```
#include <iostream>
#include <boost/algorithm/string.hpp>
using namespace boost::algorithm;
using namespace std;
int main() {
  string a, b;

  a = "My NaMe is JohN";
  b = to_upper_copy(a);

  cout << b << endl;       //Outputs: MY NAME IS JOHN
  return 0;
}
```

Getting part of a string

```
subject.substr( beginIndex [, length] )
```

This returns a portion of *subject*. Specifically, it starts from position *beginIndex* and returns a substring of *length* characters or up to the end of the *subject*, whichever comes first. The argument *length* is optional. If it is omitted, the substring starting from position *beginIndex* until the end of *subject* is returned.

Example

project_14_4g

```
#include <iostream>
using namespace std;
int main() {
  string a;

  a = "Hello Athena";

  cout << a.substr(6, 3) << endl;          //Outputs: Ath
  cout << a.substr(6, 300) << endl;                //Outputs: Athena
  cout << a.substr(7) << endl;             //Outputs: thena
  return 0;
}
```

Checking if string is numeric

```
cin.fail()
```

This returns `true` if the last `cin` statement fails, and `false` otherwise. Moreover, if you try to read something with the `cin` statement and something goes wrong, you need to clear the error state of `cin`. This is why the `cin.fail()` function is always used in combination with the functions `cin.clear()` and `cin.ignore()`.

Example

project_14_4h

```
#include <iostream>
#include <boost/algorithm/string.hpp>
using namespace boost::algorithm;
using namespace std;

int main() {
  int a;                          //Variable a is an integer

  cin >> a;                       //Assume that user enters the word "Hello".
  cout << cin.fail() << endl;  //Since cin fails, this displays 1 (true)

  //Clear the error state of cin
  cin.clear();
  cin.ignore(100, '\n');

  cin >> a;                          //Assume that user enters an integer.
  cout << cin.fail() << endl;     //If cin succeeds, display 0 (false)

  //If the last cin statement succeeds,
  //there is no need to clear its error state

  return 0;
}
```

Retrieving an individual character

```
subject.at( index )
```

This returns the character located at *subject*'s specified *index*. As already mentioned, the string indexes start from zero. The index of the last character is 1 less than the length of the string.

Example

project_14_4i

```
#include <iostream>
using namespace std;
int main() {
  string a = "Hello Hermes";

  cout << a.at(0) << endl;        //outputs: H
  cout << a.at(11) << endl;       //outputs: s
  cout << a.at(5) << endl;        //outputs a space character
  return 0;
}
```

Exercise 14.4-1 *Switching the Order of Names*

Write a C++ program that prompts the user to enter in one single string both first and last name. In the end, the program should change the order of the two names.

Solution

This exercise is not the same as the one that you learned in Chapter 8, which swapped the numeric values of two variables. In this exercise, you must split the string and assign each name to a different variable. If you manage to do so, then you can just rejoin them in a different order.

Let's try to understand this exercise using an example. The string that you must split and the position of its individual character is shown here.

0	1	2	3	4	5	6	7	8
T	**o**	**m**		**S**	**m**	**i**	**t**	**h**

The character that visually separates the first name from the last name is the space character between them. The problem is that this character is not always at position 3. Someone can have a short first name like "Tom" and someone else can have a longer one like "Robert". Thus, you need something that actually finds the position of the space character regardless of the content of the string.

Function `find()` is what you are looking for! If you use it to find the position of the space character in the string "Tom Smith", it returns the value 3. But if you use it to find the space character in another string, such as "Angelina Brown", it returns the value 8 instead.

***Notice**: The value 3 is not just the position where the space character exists. It also represents the number of characters that the word "Tom" contains! The same applies to the value 8 that is returned for the string "Angelina Brown". It represents both the position where the space character exists and the number of characters that the word "Angelina" contains!*

The C++ program for this algorithm is shown here.

project_14_4_1

```
#include <iostream>
using namespace std;
int main() {
  string full_name, name1, name2;
  int space_pos;

  cout << "Enter your full name: ";
  getline(cin, full_name);

  //find the position of space character. This is also the number
  //of characters first name contains
```

```
  space_pos = full_name.find(" ");

  //get space_pos number of characters starting from position 0
  name1 = full_name.substr(0, space_pos);

  //get the rest of the characters starting from position space_pos + 1
  name2 = full_name.substr(space_pos + 1);

  full_name = name2 + " " + name1;

  cout << full_name;
  return 0;
}
```

Remember! *The statement* `cin` *considers space characters as terminating characters when reading a string from the keyboard. Thus, reading a string with* `cin` *means to always read a single word, not a phrase or an entire sentence. To read an entire line you can use the* `getline()` *function.*

Notice*: Please note that this program cannot be applied to a Spanish name such as "Maria Teresa García Ramírez de Arroyo." The reason is obvious!*

Exercise 14.4-2 *Creating a Login ID*

Write a C++ program that prompts the user to enter his or her last name and then creates a login ID from the first four letters of the name (in lowercase) and a three-digit random integer.

Solution

To create a random integer you can use the `rand()` function. Since you need a random integer of three digits, the range must be between 100 and 999.

The C++ program for this algorithm is shown here.

project_14_4_2

```
#include <iostream>
#include <boost/algorithm/string.hpp>
#include <ctime>
#include <cstdlib>
using namespace boost::algorithm;
using namespace std;
int main() {
  int random_int;
  string last_name;

  srand(time(NULL));
```

```
  cout << "Enter last name: ";
  cin >> last_name;

  //Get Random integer between 100 and 999
  random_int = 100 + rand() % 900;

  cout << to_lower_copy(last_name.substr(0, 4)) << random_int;
  return 0;
}
```

***Notice**: Please note how the function* `substr()` *is nested within the function* `to_lower_copy()`*. The result of the inner (nested) function is used as an argument for the outer function. This is a writing style that most programmers prefer to follow because it helps to save a lot of code lines. Of course, if you nest too many functions, no one will be able to understand your code. A nesting of up to four levels is quite acceptable.*

Exercise 14.4-3 *Creating a Random Word*

Write a C++ program that displays a random word consisting of five letters.

Solution

To create a random word you need a string that contains all 26 letters of the English alphabet. Then you can use the `rand()` function to choose a random letter between position 0 and 25.

The C++ program for this algorithm is shown here.

project_14_4_3

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;
int main() {
  string alphabet;

  srand(time(NULL));

  alphabet = "abcdefghijklmnopqrstuvwxyz";

  cout << alphabet.at(rand() % 26) <<
          alphabet.at(rand() % 26) <<
          alphabet.at(rand() % 26) <<
          alphabet.at(rand() % 26) <<
          alphabet.at(rand() % 26);

  return 0;
```

```
}
```

14.5 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A string is anything that you can type using the keyboard.
2. Strings must be enclosed in parentheses.
3. The phrase "Hi there!" contains 8 characters.
4. In the phrase "Hi there!" the letter "t" is at position 3.
5. The statement `y = a.at(1)` assigns the second letter of the string contained in variable `a` to variable `y`.
6. The statement

   ```
   y = a.at(-1);
   ```

   is a valid statement.
7. Trimming is the process of removing whitespace characters from the beginning or the end of a string.
8. The statement `y = trim_copy("Hello Aphrodite")` assigns the value "`HelloAphrodite`" to variable `y`.
9. The statement `cout << replace_all_copy("Hi there!", "Hi", "Hello")` displays the message "`Hello there!`".
10. The following code fragment

    ```
    a = "Hi there";
    index = a.find("the");
    ```

    assigns the value 4 to the variable `index`.
11. The statement `cout << to_upper_copy("hello there!")` displays the message "`Hello There`".
12. The following code fragment

    ```
    a = "Hello there!";
    cout << a.substr(0);
    ```

    displays the message "`Hello there!`".
13. The statement `cout << a.substr(0, a.length())` displays some letters of the variable `a`.
14. The statement `cout << a.substr(a.length() - 1, 1)` is equivalent to the statement `cout << a.at(a.length() - 1))`.
15. The following code fragment

    ```
    y = "hello there!";
    cout << to_upper_copy(y).substr(0, 5);
    ```

    displays the word "`HELLO`".

16. The following code fragment

```
y = "HELLO THERE!";
cout << to_lower_copy(y);
```

displays the message "`Hello there!`".

17. The statement `a.substr(0, 1)` is equivalent to the statement `a.at(0)`.

14.6 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. Which of the following is **not** a string?
   a. "Hello there!"
   b. "13"
   c. "13.5"
   d. All of the above are strings.
2. In which position does the space character in the string "Hello Zeus!", exist?
   a. 6
   b. 5
   c. Space is not a character.
   d. none of the above
3. The statement `cout << a.substr(a.length() - 1, 1)` displays
   a. the last character of variable `a`.
   b. the second to last character of variable `a`.
   c. The statement is not valid.
4. The statement

```
trim_copy(replace_all_copy(replace_all_copy(a,"a","b"),"w","y"))
```

is equivalent to the statement
   a. `replace_all_copy(replace_all_copy(trim_copy(a),"a","b"),"w","y")`
   b. `replace_all_copy(trim_copy(replace_all_copy(a,"w","y")),"a","b")`
   c. `replace_all_copy(replace_all_copy(trim_copy(a),"w","y"),"a","b")`
   d. all of the above
5. The statement `replace_all_copy(a, " ", "")`
   a. adds a space between each letter in the variable `a`.
   b. removes all space characters from the variable `a`.
   c. empties the variable `a`.
6. The statement `replace_all_copy("  Hello  ", " ", "")` is equivalent to the statement
   a. `replace_all_copy("  Hello  ", "", " ")`.
   b. `trim_copy("  Hello  ")`.

c. all of the above
d. none of the above

7. The following code fragment

```
a = "";
cout << a.length();
```

displays

a. nothing.
b. 1.
c. 0.
d. The statement is invalid.
e. none of the above

8. Which value assigns the following code fragment

```
to_be_or_not_to_be = "2b || !2b";
Shakespeare = to_be_or_not_to_be.find("b");
```

to the variable `Shakespeare`?

a. 1
b. 2
c. 6
d. none of the above

9. What does the following code fragment?

```
a = "Hi there";
b = a.substr(a.find(" ") + 1);
```

a. It assigns the word "`Hi`" to the variable `b`.
b. It assigns a space character to the variable `b`.
c. It assigns the word "`there`" to the variable `b`.
d. none of the above

10. What does the following code fragment?

```
a = "Hi there";
b = replace_all_copy(a, "Hi", "Hello");
```

a. It assigns the string "`Hello there`" to the variable `a`.
b. It displays the message "`Hello there`".
c. all of the above
d. none of the above

14.7 Review Exercises

Complete the following exercises.

1. Write a C++ program that prompts the user to enter his or her first name, middle name, last name, and his or her preferred title (Mr., Mrs., Ms., Dr., and so on) and displays them formatted in all the following ways.

   Title FirstName MiddleName LastName

   FirstName MiddleName LastName

   LastName, *FirstName*

   LastName, *FirstName MiddleName*

   LastName, *FirstName MiddleName*, *Title*

   FirstName LastName

   For example, assume that the user enters the following:

   First name: Aphrodite

   Middle name: Maria

   Last name: Boura

   Title: Ms.

   The program should display the user's name formatted in all the following ways:

   Ms. Aphrodite Maria Boura

   Aphrodite Maria Boura

   Boura, Aphrodite

   Boura, Aphrodite Maria

   Boura, Aphrodite Maria, Ms.

   Aphrodite Boura

2. Write a C++ program that creates and displays a random word consisting of five letters. The first letter must be a capital letter.

3. Write a C++ program that prompts the user to enter his or her name and then creates a secret password consisting of three letters (in lowercase) randomly picked up from his or her name, and a random four-digit number. For example, if the user enters "Vassilis Bouras" a secret password can probably be one of "sar1359" or "vbs7281" or "bor1459".

Review Questions in “Sequence Control Structures”

Answer the following questions.

1. What is a sequence control structure?
2. What operations can a sequence control structure perform?
3. What does the term "type casting" mean in computer science?
4. Give some examples of how you can use the quotient and the remainder of an integer division.
5. What is a function?
6. What does the term “nest a function” mean?

Index

A

abs(), 123
add a breakpoint, 106
algorithm, 37
alphanumeric, 56
application software, 15, 16
Aristides, 9
arithmetic operator, 73
arrow, 39
assign, 52
assigned, 71
asterisk–slash, 47
at(), 159

B

BMI, 121
Body Mass Index, 121
bool, 58
Boolean, 56
breakpoint, 106
bug, 100

C

C++, 15
C++ project, 94
Celsius, 120
Central Processing Unit, 15
char, 58
character, 56
code, 17
command, 15, 17, 37, 43
comment, 46
compiled, 21
compiler, 17
complex mathematical expression, 136
compound assignment operator, 76
computer language, 16
computer program, 38
concatenation, 79
concatenation assignment, 79
Const, 54
constant, 53
control structure, 43
Corrado Böhm, 43
cos(), 123
cosine, 123
CPU, 15

D

data input, 38, 40, 67
data processing, 38
debugger, 104
debugging, 100
debugging step by step, 102
decision control structure, 43
Decision symbol, 40
declaration, 57
declare constant, 59
decrementing operator, 78
definiteness, 38
DIV, 143
double, 58, 124, 125
double slashes, 47, 79

E

effectiveness, 38
End symbol, 39
execute, 16, 99

F

Fahrenheit, 117
fail(), 158
find(), 157, 160

Finish Debugger Session, 105
finite, 37
finite sequence, 37
finiteness, 38
float, 58
flow of execution, 39, 103
flowchart, 39
function, 122, 152

G

Giuseppe Jacopini, 43
Grace Hopper, 101
graphical method, 39
green arrow, 104

H

hardware, 15
Heron, 122
Heron's formula, 141
high-level, 21
high-level language, 17
hosting application, 21

I

IDE, 17, 28
incoming off-page connector symbol, 40
incrementing operator, 78
index, 153
input, 37
input device, 16
input value, 37
instruction, 15, 17, 37
int, 58, 124
integer, 56
integer division, 74
integer value, 124
Integrated Development Environment, 17, 28
interpreted, 21
interpreter, 17

J

JDK, 29

K

Kelvin, 117
keyword, 42

L

Law of Least Astonishment, 48
left arrow, 52, 71
left gray margin, 106
length(), 156
line break, 65
logic error, 46, 101
loop control structure, 43
low-level language, 16

M

M_PI, 125
machine language, 16, 22
main memory, 15, 16
mathematical expression, 136
mathematical function, 123
method, 122
MOD, 74
modularization, 43
module, 43
modulus operator, 74

N

NetBeans, 28
number of characters, 156
number of spaces per indent, 34

O

off-page connector symbol, 40
operating system, 16
outgoing off-page connector symbol, 40
output, 37
output device, 16
output value, 37

P

palindrome, 142
parenthesis, 74
part of a string, 158
pink rectangle, 107
POLA, 48
post-decrementing operator, 78
post-incrementing operator, 78
pow(), 126, 130
power, 126
precedence rule, 74, 136
pre-decrementing operator, 78
pre-defined process, 40
pre-incrementing operator, 78
Principle of Least Astonishment, 48
problem, 37
Process symbol, 40
program, 15, 16, 21
program counter, 104
programmer, 15, 38
property, 37

Q

quotient, 142

R

RAM, 15
rand(), 126, 161
random, 126
Random Access Memory, 15
random word, 162
real, 56
real value, 124
red exclamation mark, 46, 101
remainder, 142
remove a breakpoint, 108
replace_all_copy(), 156
reserved word, 42
results output, 38, 40
reversed number, 150
round, 128
round(), 128
run, 16

S

script, 21
secondary storage device, 15, 16
semicolon, 46
sequence control structure, 43, 113
short int, 58
sin(), 129
sine, 129
slash–asterisk, 47
software, 16
software developer, 15
source code, 17
sqrt(), 129, 130, 132
square root, 126, 129
stand-alone, 21
Start symbol, 39
statement, 15, 37, 43
step by step, 102
step over, 104, 106, 107
string, 56, 152
string index, 154
string position, 157
string replacement, 156
structured code, 43
structured design, 43
structured programming, 43
subprogram, 40

substr(), 158, 162
substring notation, 153
sum, 143
swap, 87, 88
syntax error, 46, 101
system software, 16

T

tab character, 66
tab stop, 66
tan(), 130
tangent, 130
three main stages, 38
three parties, 38
to lowercase, 157
to uppercase, 157
to_lower_copy(), 157, 162
to_upper_copy(), 157
trace table, 85
trim_copy(), 155
trimming, 155

U

unsigned int, 58
unsigned short int, 58
user, 38
user-friendly, 47

V

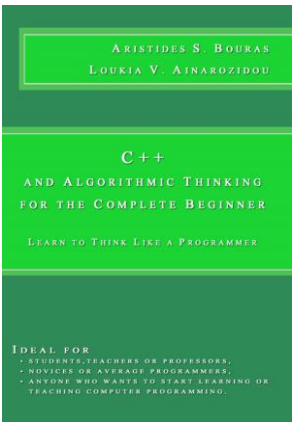
value assignment operator, 52, 71
variable, 51
visual representation, 39

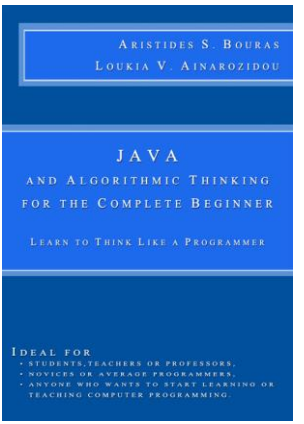
W

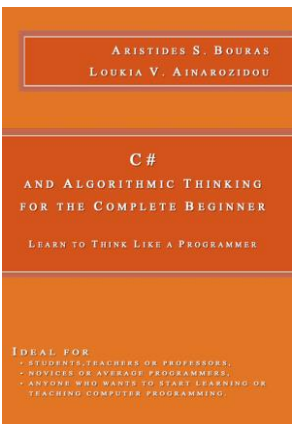
wavy red line, 101

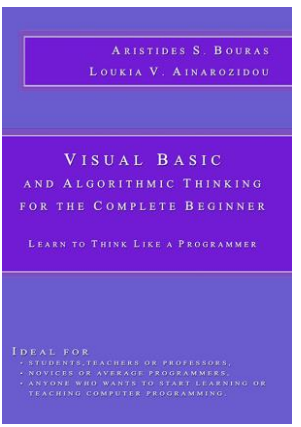
My Books

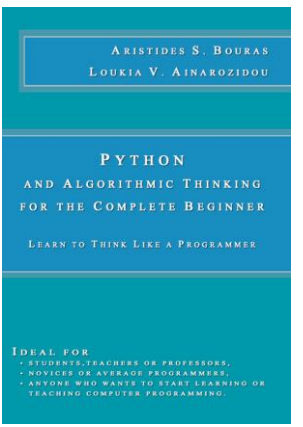
Available at amazon

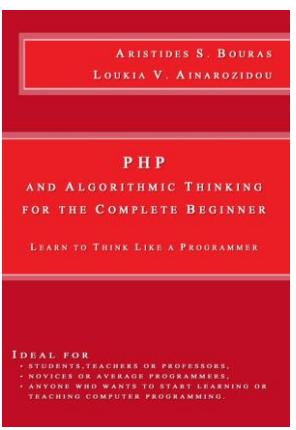












RCode: 160216