

1

Introduction to Computers, the Internet and World Wide Web



The chief merit of language is clearness.

— Galen

Our life is frittered away by detail.... Simplify, simplify.

— Henry David Thoreau

He had a wonderful talent for packing thought close, and rendering it portable.

— Thomas B. Macaulay

Man is still the most extraordinary computer of all.

— John F. Kennedy



OBJECTIVES

- In this chapter you will learn:
- Basic hardware and software concepts.
- Basic object-technology concepts, such as classes, objects, attributes, behaviors, encapsulation and inheritance.
- The different types of programming languages.
- Which programming languages are most widely used.
- A typical C++ program development environment.
- The history of the industry-standard object-oriented system modeling language, the UML.
- The history of the Internet and the World Wide Web.
- To test-drive C++ applications in two popular C++ environments—GNU C++ running on Linux and Microsoft's Visual C++® .NET running on Windows® XP.



Outline

- 1.1 Introduction
- 1.2 What Is a Computer?
- 1.3 Computer Organization
- 1.4 Early Operating Systems
- 1.5 Personal, Distributed and Client/Server Computing
- 1.6 The Internet and the World Wide Web
- 1.7 Machine Languages, Assembly Languages and High-Level Languages
- 1.8 History of C and C++
- 1.9 C++ Standard Library
- 1.10 History of Java
- 1.11 FORTRAN, COBOL, Pascal and Ada
- 1.12 Basic, Visual Basic, Visual C++, C# and .NET
- 1.13 Key Software Trend: Object Technology
- 1.14 Typical C++ Development Environment
- 1.15 Notes About C++ and *C++ How to Program, 5/e*
- 1.16 Test-Driving a C++ Application
- 1.17 Software Engineering Case Study: Introduction to Object Technology and the UML (Required)
- 1.18 Wrap-Up
- 1.19 Web Resources



1.1 Introduction

- **Software**
 - **Instructions to command computer to perform actions and make decisions**
- **Hardware**
- **Standardized version of C++**
 - **United States**
 - **American National Standards Institute (ANSI)**
 - **Worldwide**
 - **International Organization for Standardization (ISO)**
- **Structured programming**
- **Object-oriented programming**



1.2 What Is a Computer?

- **Computer**
 - Device capable of performing computations and making logical decisions
- **Computer programs**
 - Sets of instructions that control computer's processing of data
 - Written by people called computer programmers
- **Hardware**
 - Various devices comprising computer
 - Keyboard, screen, mouse, disks, memory, CD-ROM, processing units, etc.



1.3 Computer Organization

- **Six logical units of computer**
 - **Input unit**
 - “Receiving” section
 - Obtains information from input devices
 - Keyboard, mouse, microphone, scanner, networks, etc.
 - **Output unit**
 - “Shipping” section
 - Places information processed by computer on output devices
 - Screen, printer, networks, etc.
 - Information can also be used to control other devices



1.3 Computer Organization (Cont.)

- **Six logical units of computer (Cont.)**
 - **Memory unit**
 - **Rapid access, relatively low capacity “warehouse” section**
 - **Retains information from input unit**
 - **Immediately available for processing**
 - **Retains processed information**
 - **Until placed on output devices**
 - **Often called memory or primary memory**
 - **Arithmetic and logic unit (ALU)**
 - **“Manufacturing” section**
 - **Performs arithmetic calculations and logic decisions**



1.3 Computer Organization (Cont.)

- **Six logical units of computer (Cont.)**
 - **Central processing unit (CPU)**
 - “Administrative” section
 - Coordinates and supervises other sections of computer
 - **Secondary storage unit**
 - Long-term, high-capacity “warehouse” section
 - Stores inactive programs or data
 - Secondary storage devices
 - Hard drives, CDs, DVDs
 - Slower to access than primary memory
 - Less expensive per unit than primary memory



1.4 Early Operating Systems

- **Early computers**
 - **Single-user batch processing**
 - **Only one job or task at a time**
 - **Process data in groups (batches)**
 - **Decks of punched cards**
 - **Users had to wait hours or days for results**
- **Operating systems**
 - **Software systems that manage transitions between jobs**
 - **Increased throughput**
 - **Amount of work computers can process**



1.4 Early Operating Systems (Cont.)

- **Multiprogramming**
 - Many jobs or tasks sharing computer's resources
 - “Simultaneous” operation of many jobs
- **Timesharing**
 - Special case of multiprogramming
 - Users access computer through terminals
 - Devices with keyboards and screens
 - Dozens, even hundreds of users
 - Computer use is shared among all users
 - Performs small portion of one user's job, then moves on to service next user
 - Advantage
 - User receives almost immediate responses to requests



1.5 Personal, Distributed and Client/Server Computing

- **Personal computers**
 - Popularized in 1977 by Apple Computer
 - Economical enough for individual
 - Legitimized in 1981 by IBM Personal Computer
 - “Standalone” units
- **Computer networks**
 - Connected over telephone lines
 - Local area networks (LANs)
- **Distributed computing**
 - Organization’s computing distributed over networks



1.5 Personal, Distributed and Client/Server Computing (Cont.)

- **Workstations**
 - Provide enormous capabilities
- **Client/server computing**
 - File servers
 - Offer common store of programs and data
 - Client computers
 - Access file servers across network
- **UNIX, Linux, Mac OS X and Microsoft's Window-based systems support these capabilities**



1.6 The Internet and the World Wide Web

- **Internet**
 - **Global network of computers**
 - **Initiated almost four decades ago**
 - **Accessible by computers worldwide today**
- **World Wide Web**
 - **Allows computer users to locate and view multimedia-based documents**
 - **Internet has become one of the world's premier communication mechanisms**



1.7 Machine Languages, Assembly Languages and High-Level Languages

- **Three types of computer languages**
 - **Machine language**
 - Only language computer directly understands
 - “Natural language” of computer
 - Defined by hardware design
 - Generally consist of strings of numbers
 - Ultimately 0s and 1s
 - Instruct computers to perform elementary operations
 - Cumbersome for humans
 - Example
 - +1300042774
 - +1400593419
 - +1200274027



1.7 Machine Languages, Assembly Languages and High-Level Languages (Cont.)

- **Three types of computer languages (Cont.)**
 - **Assembly language**
 - English-like abbreviations representing elementary computer operations
 - Clearer to humans
 - Incomprehensible to computers
 - Convert to machine language by translator programs (assemblers)
 - Example
 - load basepay
 - add overpay
 - store grosspay



1.7 Machine Languages, Assembly Languages and High-Level Languages (Cont.)

- **Three types of computer languages (Cont.)**
 - **High-level languages**
 - **Similar to everyday English**
 - **Uses common mathematical notations**
 - **Single statements accomplish substantial tasks**
 - **Converted to machine language by translator programs (compilers)**
 - **Interpreter programs**
 - **Directly execute high-level language programs**
 - **Execute more slowly than the compiled program**
 - **Example**
 - `grossPay = basePay + overTimePay`



1.8 History of C and C++

- **History of C**
 - **Evolved from BCPL and B**
 - **Developed by Dennis Ritchie (Bell Laboratories)**
 - **Development language of UNIX**
 - **Hardware independent**
 - **Can write portable programs**
 - **ANSI and ISO standard for C published in 1990**
 - *ANSI/ISO 9899: 1990*



Portability Tip 1.1

Because C is a standardized, hardware-independent, widely available language, applications written in C often can be run with little or no modification on a wide range of computer systems.



1.8 History of C and C++ (Cont.)

- **History of C++**
 - **Extension of C**
 - **Developed by Bjarne Stroustrup (Bell Laboratories) in early 1980s**
 - **Provides new features to “spruce up” C**
 - **Provides capabilities for object-oriented programming**
 - **Objects: reusable software components**
 - **Model items in the real world**
 - **Object-oriented programs**
 - **Easier to understand, correct and modify**



1.9 C++ Standard Library

- **C++ programs**
 - Built from pieces called classes and functions
- **C++ Standard Library**
 - Rich collections of existing classes and functions
 - Reusable in new applications



Software Engineering Observation 1.1

Use a “building-block” approach to create programs. Avoid reinventing the wheel. Use existing pieces wherever possible. Called software reuse, this practice is central to object-oriented programming.



Software Engineering Observation 1.2

When programming in C++, you typically will use the following building blocks: Classes and functions from the C++ Standard Library, classes and functions you and your colleagues create and classes and functions from various popular third-party libraries.



Performance Tip 1.1

Using C++ Standard Library functions and classes instead of writing your own versions can improve program performance, because they are written carefully to perform efficiently. This technique also shortens program development time.



Portability Tip 1.2

Using C++ Standard Library functions and classes instead of writing your own improves program portability, because they are included in every C++ implementation.



1.10 History of Java

- **Java**
 - **Originally for intelligent consumer-electronic devices**
 - **Designed by Sun Microsystems**
 - **Then used for creating Web pages with dynamic content**
 - **Now also used for:**
 - **Develop large-scale enterprise applications**
 - **Enhance World Wide Web server functionality**
 - **Provide applications for consumer devices (cell phones, etc.)**



1.11 FORTRAN, COBOL, Pascal and Ada

- **FORTRAN**

- **FORmula TRANslator**
- **Used in engineering applications**

- **COBOL**

- **COmmon Business Oriented Language**
- **Used in business software**

- **Pascal**

- **Designed to teach structured programming**

- **Ada**

- **Capable of multitasking**



1.12 Basic, Visual Basic, Visual C++, C# and .NET

- **BASIC**
 - Beginner's All-Purpose Symbolic Instruction Code
 - Familiarize novices with programming techniques
- **.NET platform**
 - Provides developers with capabilities
- **Visual Basic .NET**
 - Based on BASIC
- **Visual C++**
 - Based on C++
- **C#**
 - Based on C++ and Java



1.13 Key Software Trend: Object Technology

- **Objects**
 - **Reusable software components that model real world items**
 - **Meaningful software units**
 - **Time objects, paycheck objects, record objects, etc.**
 - **Any noun can be represented as an object**
 - **More understandable, better organized and easier to maintain than procedural programming**
 - **Libraries of reusable software**
 - **MFC (Microsoft Foundation Classes)**
 - **.NET Framework Class Library**
 - **Rogue Wave**



Software Engineering Observation 1.3

Extensive class libraries of reusable software components are available over the Internet and the World Wide Web. Many of these libraries are available at no charge.



1.14 Typical C++ Development Environment

- **C++ programs normally undergo six phases**
 - **Edit**
 - Programmer writes program (and stores source code on disk)
 - **Preprocess**
 - Perform certain manipulations before compilation
 - **Compile**
 - Compiler translates C++ programs into machine languages
 - **Link**
 - Link object code with missing functions and data
 - **Load**
 - Transfer executable image to memory
 - **Execute**
 - Execute the program one instruction at a time



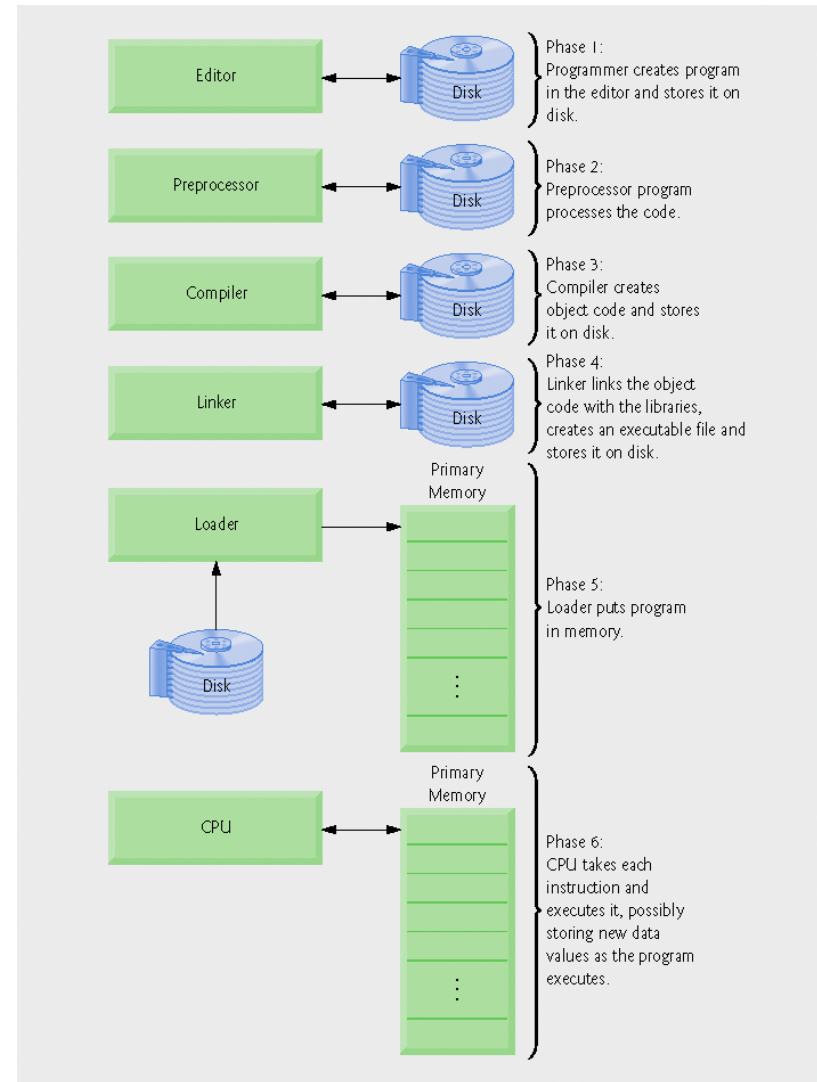


Fig. 1.1 | Typical C++ environment.



1.14 Typical C++ Development Environment (Cont.)

- **Input/output**
 - **cin**
 - **Standard input stream**
 - **Normally inputs from keyboard**
 - **cout**
 - **Standard output stream**
 - **Normally outputs to computer screen**
 - **cerr**
 - **Standard error stream**
 - **Displays error messages**



Common Programming Error 1.1

Errors like division by zero occur as a program runs, so they are called runtime errors or execution-time errors. Fatal runtime errors cause programs to terminate immediately without having successfully performed their jobs.

Nonfatal runtime errors allow programs to run to completion, often producing incorrect results.

[*Note:* On some systems, divide-by-zero is not a fatal error. Please see your system documentation.]



1.15 Notes About C++ and C++ How to Program, 5/e

- **This book is geared toward novice programmers**
 - Stresses programming clarity
- **Portability**
 - C and C++ programs can run on many different computers



Good Programming Practice 1.1

Write your C++ programs in a simple and straightforward manner. This is sometimes referred to as KIS (“keep it simple”). Do not “stretch” the language by trying bizarre usages.



Portability Tip 1.3

Although it is possible to write portable programs, there are many problems among different C and C++ compilers and different computers that can make portability difficult to achieve. Writing programs in C and C++ does not guarantee portability. The programmer often will need to deal directly with compiler and computer variations. As a group, these are sometimes called platform variations.



Good Programming Practice 1.2

Read the manuals for the version of C++ you are using. Refer to these manuals frequently to be sure you are aware of the rich collection of C++ features and that you are using them correctly.



Good Programming Practice 1.3

Your computer and compiler are good teachers. If after reading your C++ language manual, you still are not sure how a feature of C++ works, experiment using a small “test program” and see what happens. Set your compiler options for “maximum warnings.” Study each message that the compiler generates and correct the programs to eliminate the messages.



1.16 Test-Driving a C++ Application

- **Running and interacting with a C++ application**
 - Windows XP Command Prompt
 - Linux shell



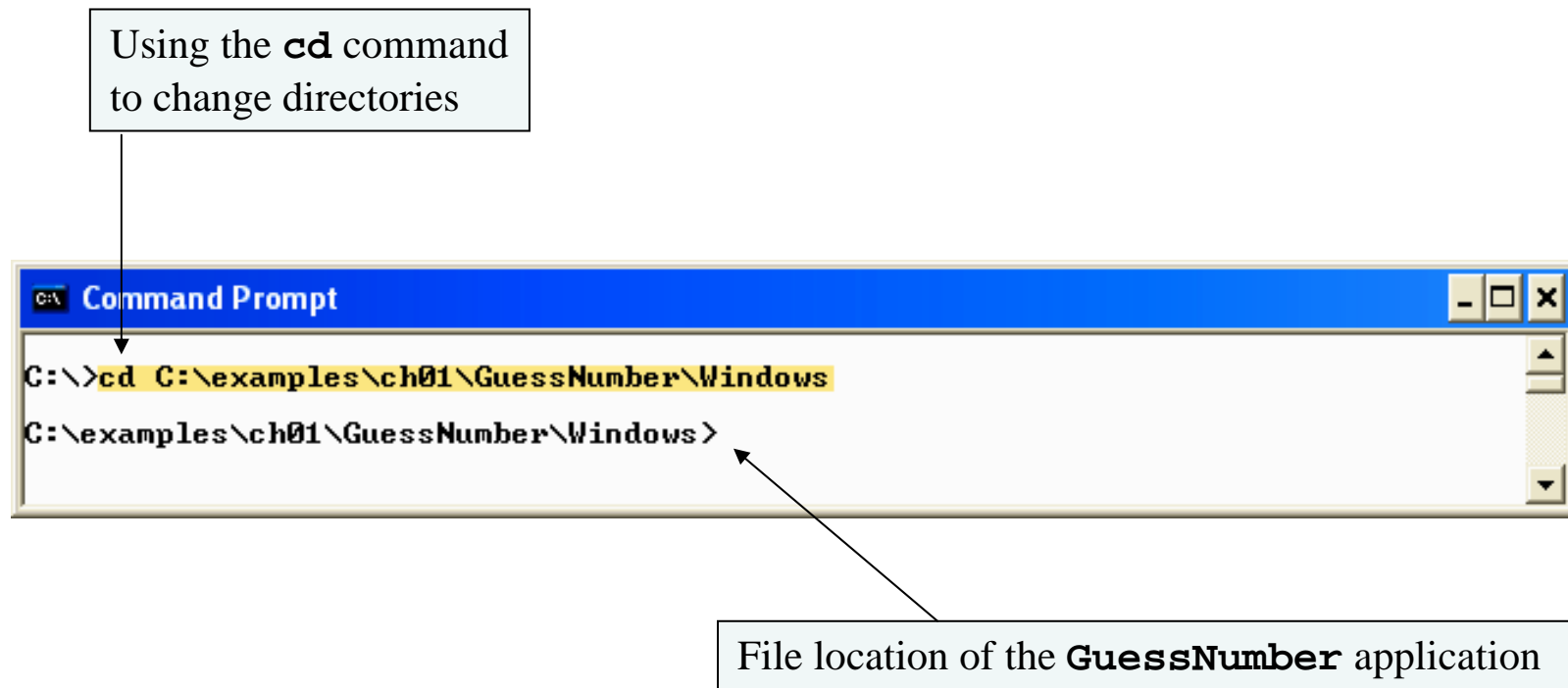


Fig. 1.2 | Opening a Command Prompt window and changing the directory.



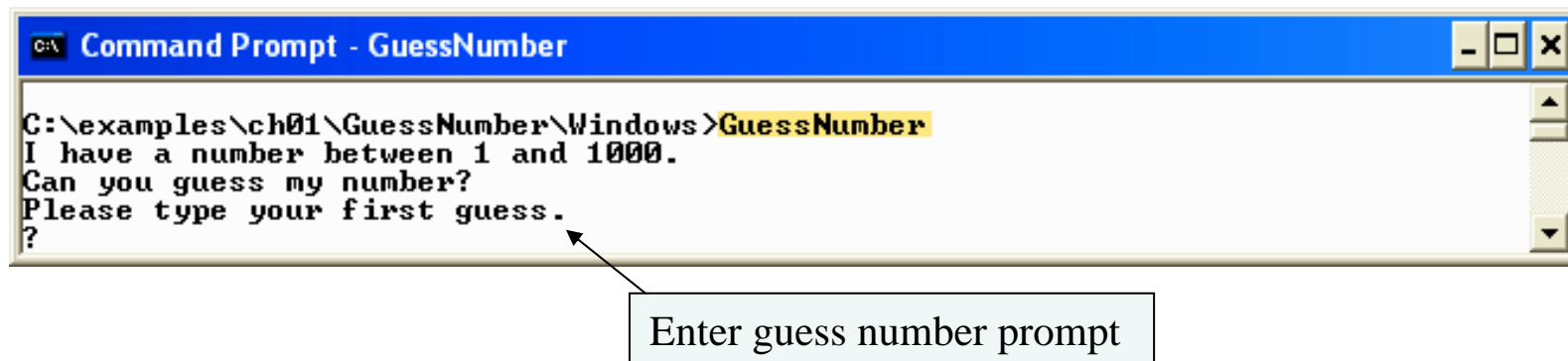
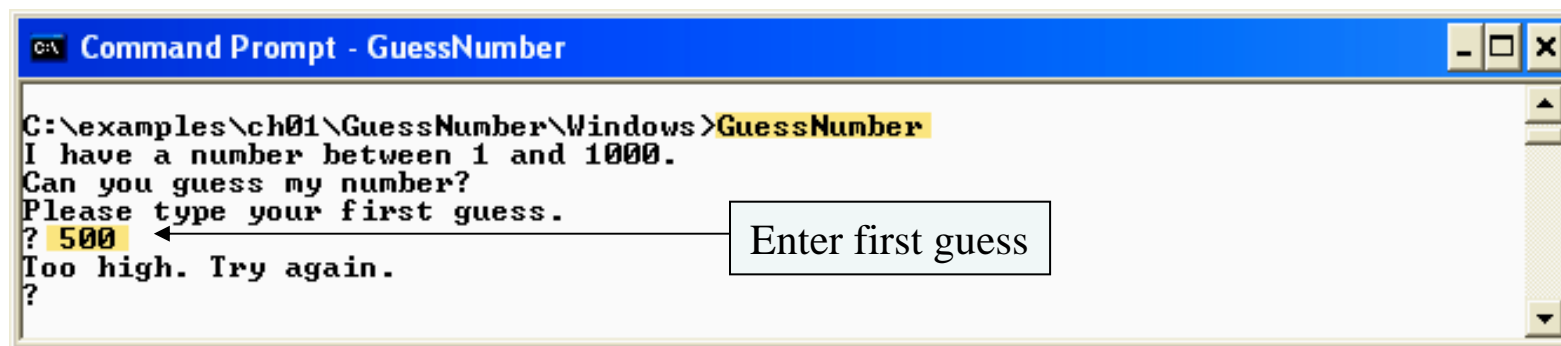


Fig. 1.3 | Running the GuessNumber application.



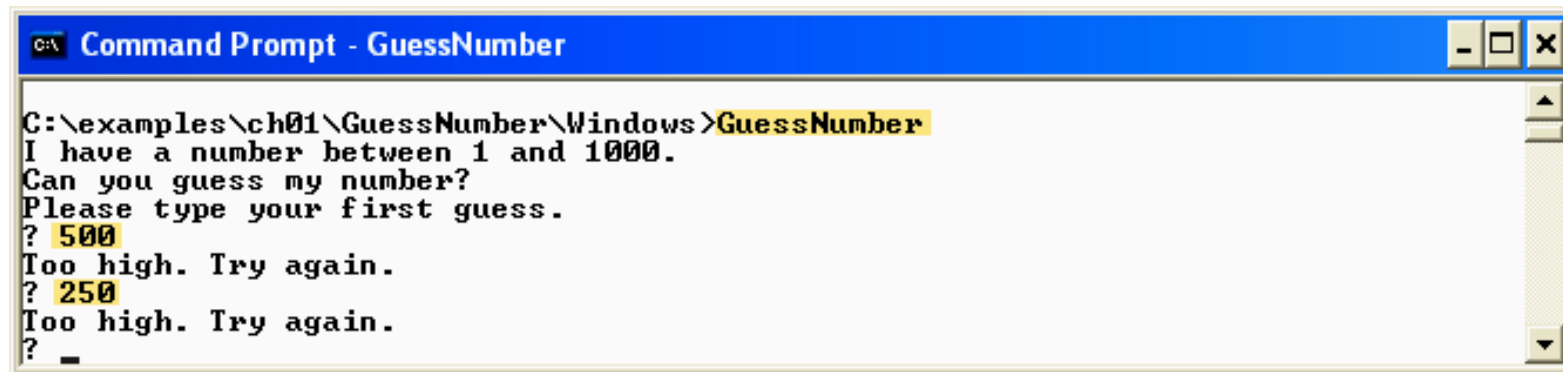


```
C:\examples\ch01\GuessNumber\Windows>GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
?
```

Enter first guess

Fig. 1.4 | Entering your first guess.





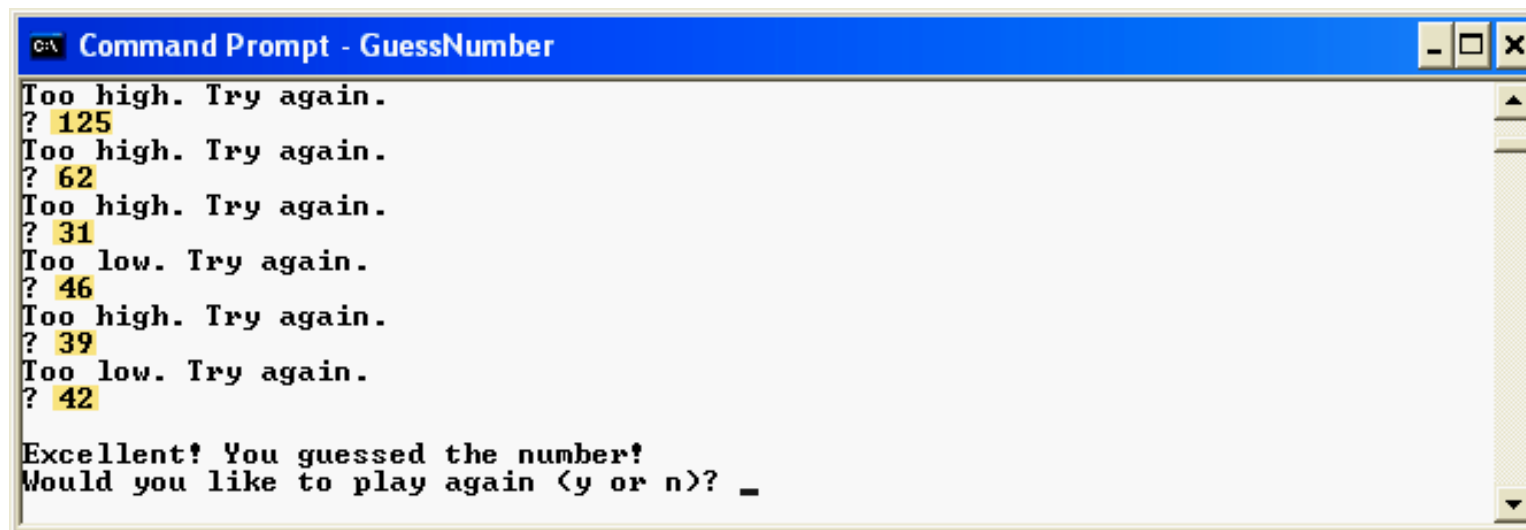
```

C:\examples\ch01\GuessNumber\Windows>GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too high. Try again.
? _

```

Fig. 1.5 | Entering a second guess and receiving feedback.





```

C:\ Command Prompt - GuessNumber
Too high. Try again.
? 125
Too high. Try again.
? 62
Too high. Try again.
? 31
Too low. Try again.
? 46
Too high. Try again.
? 39
Too low. Try again.
? 42

Excellent! You guessed the number!
Would you like to play again (y or n)? _
  
```

Fig. 1.6 | Entering additional guesses and guessing the correct number.



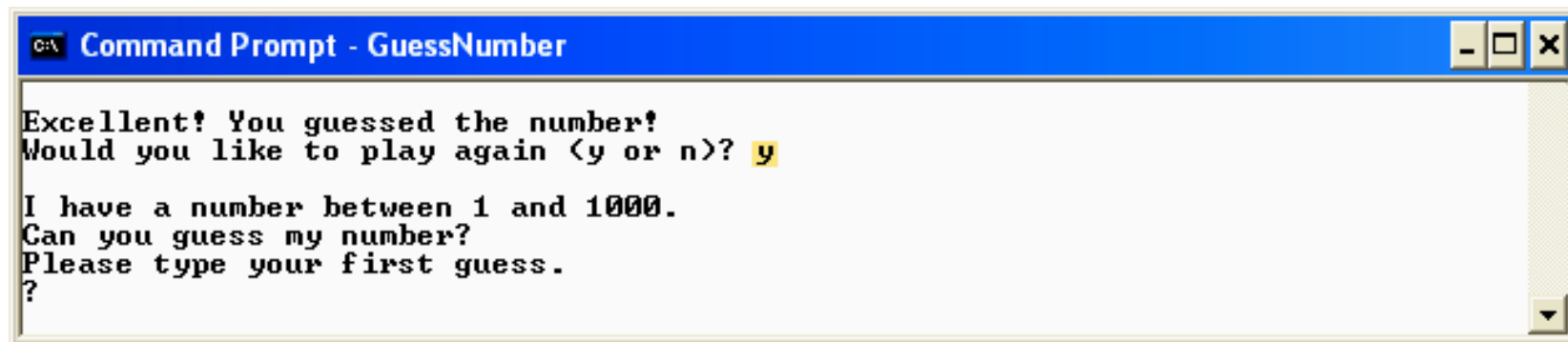


Fig. 1.7 | Playing the game again.



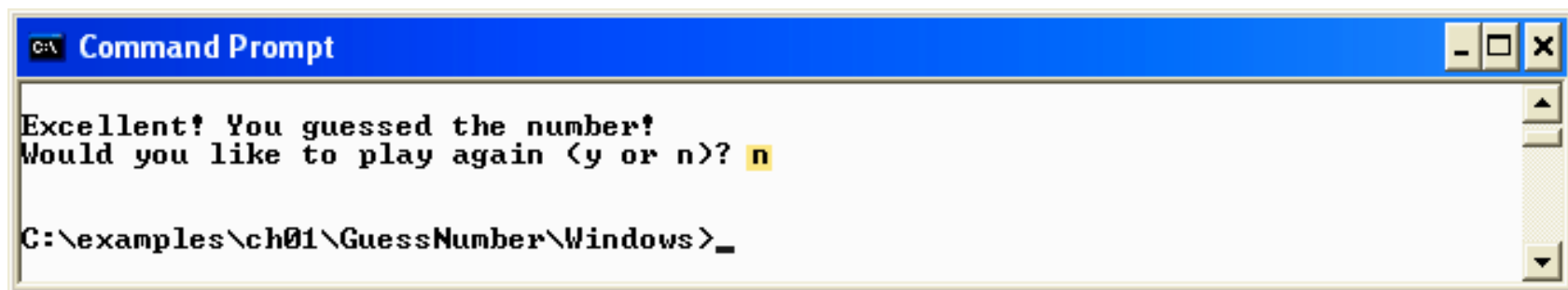


Fig. 1.8 | Exiting the game.



```
~$ cd examples/ch01/GuessNumber/GNU_Linux  
~/examples/ch01/GuessNumber/GNU_Linux$
```

Fig. 1.9 | Changing to the GuessNumber application's directory after logging in to your Linux account.



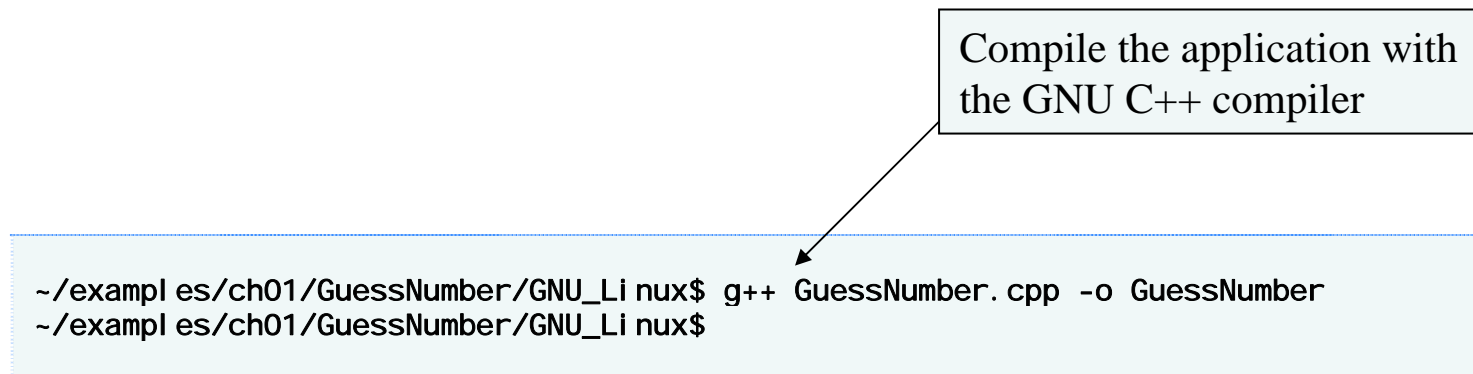


Fig. 1.10 | Compiling the GuessNumber application using the g++ command.



```
~/examples/ch01/GuessNumber/GNU_Linux$ ./GuessNumber  
I have a number between 1 and 1000.  
Can you guess my number?  
Please type your first guess.  
?
```

Fig. 1.11 | Running the GuessNumber application.



```
~/examples/ch01/GuessNumber/GNU_Linux$ ./GuessNumber  
I have a number between 1 and 1000.  
Can you guess my number?  
Please type your first guess.  
? 500  
Too high. Try again.  
?
```

Fig. 1.12 | Entering an initial guess.



```
~/examples/ch01/GuessNumber/GNU_Linux$ ./GuessNumber
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too low. Try again.
?
```

Fig. 1.13 | Entering a second guess and receiving feedback.



```

Too low. Try again.
? 375
Too low. Try again.
? 437
Too high. Try again.
? 406
Too high. Try again.
? 391
Too high. Try again.
? 383
Too low. Try again.
? 387
Too high. Try again.
? 385
Too high. Try again.
? 384

Excellent! You guessed the number.
Would you like to play again (y or n)?

```

Fig. 1.14 | Entering additional guesses and guessing the correct number.



Excellent! You guessed the number.
Would you like to play again (y or n)? y

I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
?

Fig. 1.15 | Playing the game again.



```
Excellent! You guessed the number.  
Would you like to play again (y or n)? n  
  
~/examples/ch01/GuessNumber/GNU_Linux$
```

Fig. 1.16 | Exiting the game.



1.17 Software Engineering Case Study: Introduction to Object Technology and the UML (Required)

- **Object orientation**
 - A natural way of thinking about the world and computer programs
- **Unified Modeling Language (UML)**
 - Graphical language that uses common notation
 - Allows developers to represent object-oriented designs



1.17 Software Engineering Case Study (Cont.)

- **Objects**

- **Reusable software components that model real-world items**
- **Examples are all around you**
 - **People, animals, cars, telephones, microwave ovens, etc.**
- **Have attributes**
 - **Size, shape, color, weight, etc.**
- **Exhibit behaviors**
 - **Babies cry, crawl, sleep, etc.; cars accelerate, brake, turn, etc.**



1.17 Software Engineering Case Study (Cont.)

- **Object-oriented design (OOD)**
 - Models real-world objects in software
 - Models communication among objects
 - Encapsulates attributes and operations (behaviors)
 - Information hiding
 - Communication through well-defined interfaces
- **Object-oriented language**
 - Programming in object oriented languages is called object-oriented programming (OOP)
 - C++ is an object-oriented language
 - Programmers can create user-defined types called classes
 - Contain data members (attributes) and member functions (behaviors)



Software Engineering Observation 1.4

Reuse of existing classes when building new classes and programs saves time, money and effort. Reuse also helps programmers build more reliable and effective systems, because existing classes and components often have gone through extensive testing, debugging and performance tuning.



1.17 Software Engineering Case Study (Cont.)

- **Object-Oriented Analysis and Design (OOAD)**
 - Analyze program requirements, then develop solution
 - Essential for large programs
 - Plan in pseudocode or UML



1.17 Software Engineering Case Study (Cont.)

- **History of the UML**
 - **Used to approach OOAD**
 - **Object Management Group (OMG) supervised**
 - **Brainchild of Booch, Rumbaugh and Jacobson**
 - **Version 1.5 is current version**
 - **Version 2 under development**
- **UML**
 - **Graphical representation scheme**
 - **Enables developers to model object-oriented systems**
 - **Flexible and extendible**

