Chapter 1: Preliminaries
In this chapter, you will learn about:

• Unit analysis
• Exponential and scientific notations
• Software development
• Algorithms
• Software, hardware, and computer storage
• Common programming errors
Preliminary 1: Unit Analysis

- Make sure you perform a unit analysis

$$\text{days} \times \frac{24 \text{ hr}}{\text{day}} \times \frac{60 \text{ min}}{\text{hr}}$$
Many engineering and scientific applications deal with extremely large and extremely small numbers

- Written in *exponential notation* to make entering the numbers in a computer program easier
- Written in *scientific notation* to performing hand calculations for verification purposes
### Preliminary 2: Exponential and Scientific Notations (continued)

- Examples of exponential and scientific notation:

<table>
<thead>
<tr>
<th>Decimal Notation</th>
<th>Exponential Notation</th>
<th>Scientific Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1625.</td>
<td>1.625e3</td>
<td>$1.625 \times 10^3$</td>
</tr>
<tr>
<td>63421.</td>
<td>6.3421e4</td>
<td>$6.3421 \times 10^4$</td>
</tr>
<tr>
<td>.00731</td>
<td>7.31e-3</td>
<td>$7.31 \times 10^{-3}$</td>
</tr>
<tr>
<td>.000625</td>
<td>6.25e-4</td>
<td>$6.25 \times 10^{-4}$</td>
</tr>
</tbody>
</table>
Using Scientific Notation

• Rule 1:
  \[10^n \times 10^m = 10^{n+m}\] for any values, positive or negative, of \(n\) and \(m\)

• Rule 2:
  \[\frac{1}{10^{-n}} = 10^n\] for any positive or negative value of \(n\)
Using Scientific Notation (continued)

• If exponent is positive, it represents the actual number of zeros that follow the 1
• If exponent is negative, it represents one less than the number of zeros after the decimal point and before the 1
• Scientific notation can be used with any decimal number
  – Not just powers of 10
Using Scientific Notation (continued)

<table>
<thead>
<tr>
<th>Scientific Notation</th>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-12}$</td>
<td>p</td>
<td>pico</td>
</tr>
<tr>
<td>$10^{-9}$</td>
<td>n</td>
<td>nano</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>μ</td>
<td>micro</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>m</td>
<td>milli</td>
</tr>
<tr>
<td>$10^3$</td>
<td>k</td>
<td>kilo</td>
</tr>
<tr>
<td>$10^6$</td>
<td>M</td>
<td>mega</td>
</tr>
<tr>
<td>$10^9$</td>
<td>G</td>
<td>giga</td>
</tr>
<tr>
<td>$10^{12}$</td>
<td>T</td>
<td>tera</td>
</tr>
</tbody>
</table>

**Table 1.2 Scientific Notational Symbols**
Preliminary 3: Software Development

• **Computer program:**
  Self-contained set of instructions used to instruct a computer to produce a specific result
Preliminary 3: Software Development (continued)

• **Software development procedure:** Helps developers understand the problem to be solved and create an effective, appropriate software solution

• **Software engineering:**
  – Concerned with creating readable, efficient, reliable, and maintainable programs and systems
  – Uses software development procedure to achieve this goal
Figure 1.2 The three phases of program development
Phase I: Development and Design

• **Program requirement**: Request for a program or a statement of a problem

• After a program requirement is received, Phase I begins:

• Phase I consists of four steps:
  – Analysis
  – Design
  – Coding
  – Testing
Phase I: Development and Design (continued)

Figure 1.3 The development and design steps
Phase I: Development and Design (continued)

- **Step 1: Analyze the Problem**
  - Determine and understand the output items the program must produce
  - Determine the input items
Phase I: Development and Design (continued)

• Step 2: Develop a Solution
  – Select the exact set of steps, called an “algorithm,” to solve the problem
  – Refine the algorithm
    • Start with initial solution in the analysis step until you have an acceptable and complete solution
  – Check solution
Phase I: Development and Design (continued)

- Step 2: Develop a Solution (continued)

**Figure 1.4** A first-level structure diagram
Phase I: Development and Design (continued)

- Step 2: Develop a Solution (continued)

Figure 1.5 A second-level structure diagram
Phase I: Development and Design (continued)

• Step 3: Code the Solution
  – Consists of actually writing a C++ program that corresponds to the solution developed in Step 2
  – Program should contain well-defined patterns or structures of the following types:
    • Sequence
    • Selection
    • Iteration
    • Invocation
Step 3: Code the Solution (continued)

- **Sequence**: Defines the order in which instructions are executed
- **Selection**: Allows a choice between different operations, based on some condition
- **Iteration**: Allows the same operation to be repeated based on some condition
  - Also called looping or repetition
- **Invocation**: Involves invoking a set of statements when needed
Phase I: Development and Design (continued)

• Step 4: Test and Correct the Program
  – **Testing:** Method to verify correctness and that requirements are met
  – **Bug:** A program error
  – **Debugging:** The process of locating an error, and correcting and verifying the correction
  – Testing may reveal errors, but does not guarantee the absence of errors
Phase I: Development and Design (continued)

- Step 4: Test and Correct the Program (continued)

<table>
<thead>
<tr>
<th>Step</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyze the problem</td>
<td>10%</td>
</tr>
<tr>
<td>Develop a solution</td>
<td>20%</td>
</tr>
<tr>
<td>Code the solution (write the program)</td>
<td>20%</td>
</tr>
<tr>
<td>Test the program</td>
<td>50%</td>
</tr>
</tbody>
</table>

Table 1.3 Effort Expended in Phase I
Phase II: Documentation

• Five main documents are needed:
  – Program description
  – Algorithm development and changes
  – Well-commented program listing
  – Sample test runs
  – Users' manual
Phase III: Maintenance

• **Maintenance** includes:
  – Ongoing correction of newly discovered bugs
  – Revisions to meet changing user needs
  – Addition of new features
• Usually the longest phase
• May be the primary source of revenue
• Good documentation vital for effective maintenance
Backup

- Process of making copies of program code and documentation on a regular basis

- Backup copies = insurance against loss or damage
  - Consider using off-site storage for additional protection
Preliminary 4: Algorithms

• **Algorithm:** Step-by-step sequence of instructions
  – Must terminate
  – Describes how the data is to be processed to produce the desired output

• **Pseudocode:** English-like phrases used to describe steps in an algorithm

• **Formula:** Mathematical equations

• **Flowchart:** Diagrams with symbols
Problem: Calculate the sum of all whole numbers from 1 through 100

Method 1 - Columns: Arrange the numbers from 1 to 100 in a column and add them.

$\begin{align*}
1 \\
2 \\
3 \\
4 \\
\vdots \\
98 \\
99 \\
+100 \\
\hline \\
5050
\end{align*}$

Figure 1.6 Summing the numbers 1 to 100
Method 2 - Groups: Arrange the numbers in groups that sum to 101 and multiply the number of groups by 101.

\[
\begin{align*}
1 + 100 &= 101 \\
2 + 99 &= 101 \\
3 + 98 &= 101 \\
4 + 97 &= 101 \\
\vdots \\
49 + 52 &= 101 \\
50 + 51 &= 101
\end{align*}
\]

50 groups

\[(50 \times 101 = 5050)\]

Figure 1.6 Summing the numbers 1 to 100 (continued)
Method 3 - Formula: Use the formula.

\[
\text{sum} = \frac{n(a + b)}{2}
\]

where

- \( n \) = number of terms to be added (100)
- \( a \) = first number to be added (1)
- \( b \) = last number to be added (100)

\[
\text{sum} = \frac{100(1 + 100)}{2} = 5050
\]

Figure 1.6 Summing the numbers 1 to 100 (continued)
**Figure 1.7 Flowchart symbols**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminal</td>
<td>Input/output</td>
<td>Indicates the beginning or end of a program</td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>Indicates input or output operation</td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>Indicates computation or data manipulation</td>
</tr>
<tr>
<td></td>
<td>Flow lines</td>
<td>Indicates computation or data manipulation</td>
</tr>
<tr>
<td></td>
<td>Decision</td>
<td>Indicates a program branch point</td>
</tr>
</tbody>
</table>
Figure 1.7 Flowchart symbols (continued)

- **Loop**: Indicates the initial, limit, and increment values of a loop.
- **Predefined process**: Indicates a predefined process, as in calling a function.
- **Connector**: Indicates an entry to, or exit from, another part of the flowchart or a connection point.
- **Report**: Indicates a written output report.
Figure 1.8 Flowchart for calculating the average of three numbers
Software, Hardware, and Computer Storage

- **Programming**: Process of writing a program, or software
- **Programming language**:  
  - Set of instructions used to construct a program  
  - Comes in a variety of forms and types
Machine Language

- **Machine language programs**: only programs that can actually be used to operate a computer
  - Also referred to as executable programs (executables)
  - Consists of a sequence of instructions composed of binary numbers
  - Contains two parts: an instruction and an address
Assembly Language

• **Assembly language programs**: Substitute word-like symbols, such as ADD, SUB, and MUL, for binary opcodes
  – Use decimal numbers and labels for memory addresses
    • Example: `ADD 1, 2`

• **Assemblers**: Translate programs into machine language

![Figure 1.10](image) Assembly-language programs must be translated
Low- and High-Level Languages

- **Low-level languages**: Languages that use instructions tied directly to one type of computer
  - Examples: machine language, assembly language

- **High-level languages**: Instructions resemble written languages, such as English
  - Can be run on a variety of computer types
  - Examples: Visual Basic, C, C++, Java
Low- and High-Level Languages (continued)

• **Source code:** The programs written in a high- or low-level language
  - **Interpreted:** Each statement is translated individually and executed immediately after translation

• **Compiled:** All statements are translated and stored as an executable program, or object program; execution occurs later
  - C++ is predominantly a compiled language
Low- and High-Level Languages (continued)

• **Linker:** Combines all of the compiled code required for the program
Procedural and Object Orientations

- **Procedural**: Available instructions are used to create self-contained units called procedures.

- **Object-oriented**: Reusable objects, containing code and data, are manipulated.
  - Object-oriented languages support reusing existing code more easily.

- **C++** contains features of both.
Application and System Software

- **Application software:** Programs written to perform particular tasks for users

- **System software:** Collection of programs to operate the computer system
Application and System Software (continued)

• **Operating system**: The set of system programs used to operate and control a computer

• **Tasks performed by the OS include**:  
  – Memory management  
  – Allocation of CPU time  
  – Control of input and output  
  – Management of secondary storage devices
• **Multi-user system**: A system that allows more than one user to run programs on the computer simultaneously

• **Multitasking system**: A system that allows each user to run multiple programs simultaneously
  – Also called multiprogrammed system
The Development of C++

• The purpose of most application programs is to process data to produce specific results

Figure 1.12 Basic procedural operations
The Development of C++ (continued)

• Early procedural languages included:
  – FORTRAN: Formula Translation
  – ALGOL: Algorithmic Language
  – COBOL: Common Business Oriented Language
  – BASIC: Beginners All-purpose Symbolic Instruction Code
  – Pascal
  – C

• Early object-oriented language:
  – C++
Computer Hardware

- **Computer hardware**: Components that support the capabilities of the computer

*Figure 1.15* Basic hardware units of a computer
Computer Hardware (continued)

• Components include:
  – **Arithmetic and logic unit (ALU):** Performs arithmetic and logic functions
  – **Control unit:** Directs and monitors overall operations
  – **Memory unit:** Stores instructions and data
  – **Input and output (I/O) unit:** Interfaces to peripheral devices
  – **Secondary storage:** Nonvolatile permanent storage such as hard disks
  – **Central processing unit (CPU):** Also called microprocessor; combines the ALU and control unit on a single chip
Computer Storage

- **Bit:** Smallest unit of data; value of 0 or 1
- **Byte:** Grouping of 8 bits representing a single character
- **Character codes:** Collection of patterns of 0s and 1s representing characters
  - Examples: ASCII, EBCDIC
**Computer Storage (continued)**

- **Number codes:** Patterns used to store numbers
- **Two’s complement** number code: Represents a decimal number as a binary number of 0s and 1s
  - Determine with a value box

![Binary to Decimal Conversion](image)

**Figure 1.18** Converting 10001101 to a base 10 number

---

C++ for Engineers and Scientists, Fourth Edition
Computer Storage (continued)

- **Word**: Grouping of one or more bytes
  - Facilitates faster and more extensive data access
- Number of bytes in a word determines the maximum and minimum values that can be stored:

<table>
<thead>
<tr>
<th>Word Size</th>
<th>Maximum Integer Value</th>
<th>Minimum Integer Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>127</td>
<td>-128</td>
</tr>
<tr>
<td>2 bytes</td>
<td>32,767</td>
<td>-32,768</td>
</tr>
<tr>
<td>4 bytes</td>
<td>2,147,483,647</td>
<td>-2,147,483,648</td>
</tr>
</tbody>
</table>

**Table 1.4**  Word size and Integer Values
Common Programming Errors

• Common errors include:
  – Failing to use consistent units
  – Using an incorrect form of a conversion factor
  – Rushing to write and run a program before fully understanding the requirements
  – Not backing up a program
  – Not appreciating that computers respond only to explicitly defined algorithms
Summary

• To determine correct forms of a conversion factor, perform a unit analysis
• Software: Programs used to operate a computer
• Programming language types:
  – Low-level languages
    • Machine language (executable) programs
    • Assembly languages
  – High-level languages
    • Compiler and interpreter languages
Summary (continued)

• Software engineering: discipline concerned with creating readable, efficient, reliable, and maintainable programs

• Three phases in software development:
  – Program development and design
  – Documentation
  – Maintenance
Summary (continued)

- Four steps in program development and design:
  - Analyze the problem
  - Develop a solution
  - Code the solution
  - Test and correct the solution
- Algorithm: Step-by-step procedure that describes how a task is performed
- Computer program: Self-contained unit of instructions and data used to operate a computer to produce a desired result