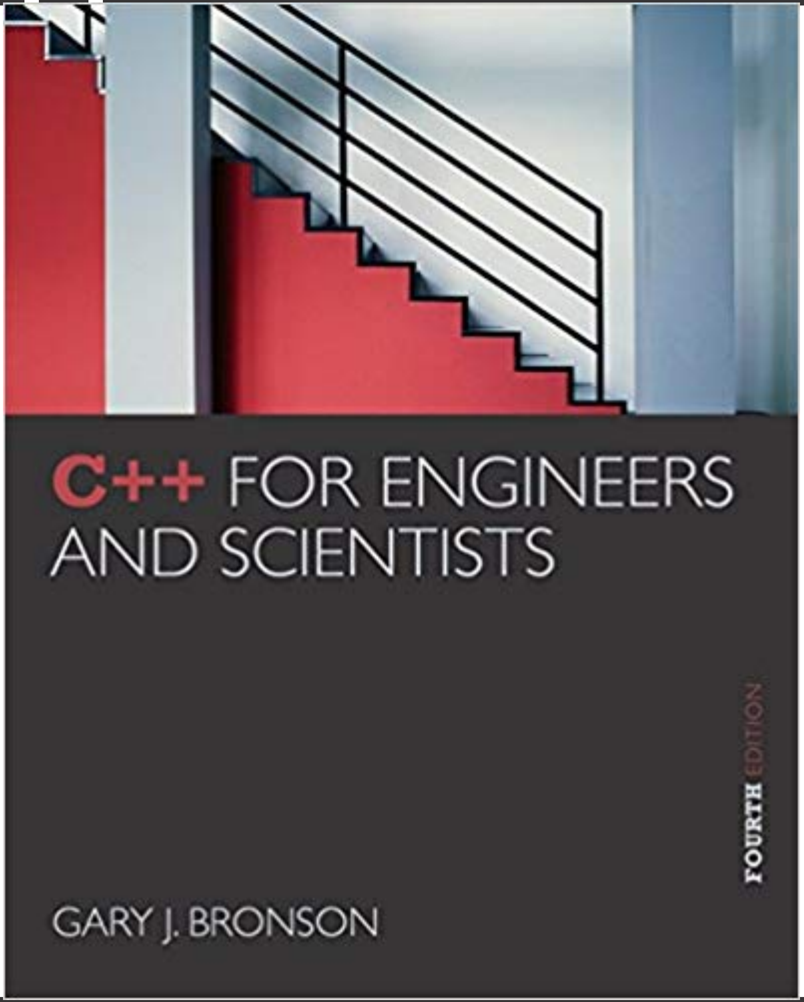
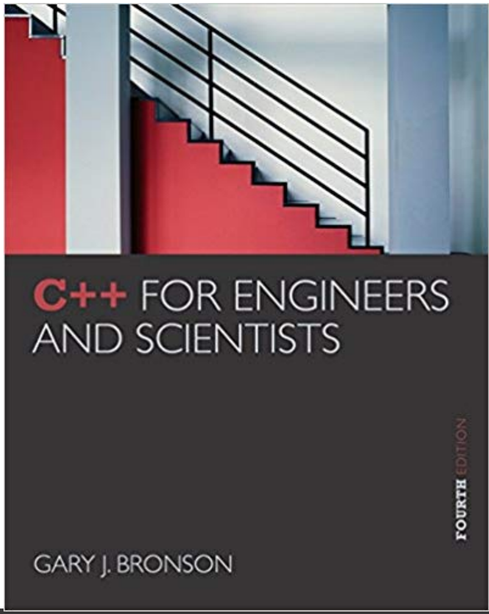
ELEG 1043
Computer Applications in Engineering



Source Materials

- Textbooks
  - **Required**
    - **C++ for Engineers and Scientists *4th Edition* , Gary J. Bronson, Thompson Learning, ISBN-13: 978-1133187844 , ISBN-10: 1133187846**
  - **Recommended**
    - **Programming and Problem Solving with C++ by Nell Dale 6th Edition**



Chapter 1: Preliminaries

Acknowledgement

- Some of the slides or images are from various sources. The copyright of those materials belongs to their original owners.

Objectives

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 One: Unit Analysis

- Using consistent and correct units when making computations is crucial
- Performing a **unit analysis:**
  - Include only the units and conversion factors in an equation
  - Cancel out corresponding units in the numerator and denominator

$$\cancel{days} \times \frac{24\ \cancel{hr}}{\cancel{day}} \times \frac{60\ min}{\cancel{hr}}$$

Example of Unit Analysis

Task: Converting Days to Seconds

1st conversion:
days to hours
(cross out the days)

2nd conversion:
hours to minutes
(cross out the hours)

3rd conversion:
minutes to seconds
(cross out the minutes)

$$\cancel{days} \times \frac{24\ \cancel{hr}}{\cancel{day}} \times \frac{60\ \cancel{min}}{\cancel{hr}} \times \frac{60\ sec}{\cancel{min}} = sec$$

Preliminary Two: Exponential and Scientific Notations

- 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 Two: Exponential and Scientific Notations (continued)

- Examples of exponential and scientific notation:

Decimal Notation	Exponential Notation	Scientific Notation
1625.	1.625e3	1.625×10^3
63421.	6.3421e4	6.3421×10^4
.00731	7.31e-3	7.31×10^{-3}
.000625	6.25e-4	6.25×10^{-4}

Using Scientific Notation

- Convenient for evaluating formulas that use very large or very small numbers
- Two basic rules
  - Rule 1: $10^n \times 10^m = 10^{n+m}$ for any values, positive or negative, of *n* and *m*
  - Rule 2: $1/10^{-n} = 10^n$ for any positive or negative value of *n*

$$\frac{10^2 \times 10^5}{10^4} = \frac{10^7}{10^4} = 10^7 \times 10^{-4} = 10^3$$

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)

- Common scientific notations have their own symbols

Value	Scientific Notation	Symbol	Name
0.000,000,000,001	10^{-12}	p	pico
0.000,000,001	10^{-9}	n	nano
0.000,001	10^{-6}	µ	micro
0.001	10^{-3}	m	milli
1000	10^{3}	k	kilo
1,000,000	10^{6}	M	mega
1,000,000,000	10^{9}	G	giga
1,000,000,000,000	10^{12}	T	tera

Table 1.2 Scientific Notational Symbols

Preliminary Three: Software Development

- **Computer program:** Self-contained set of instructions used to operate a computer to produce a specific result
  - Also called **software**
  - Solution developed to solve a particular problem, written in a form that can be executed on a computer

Preliminary Three: 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

Preliminary Three: Software Development (continued)

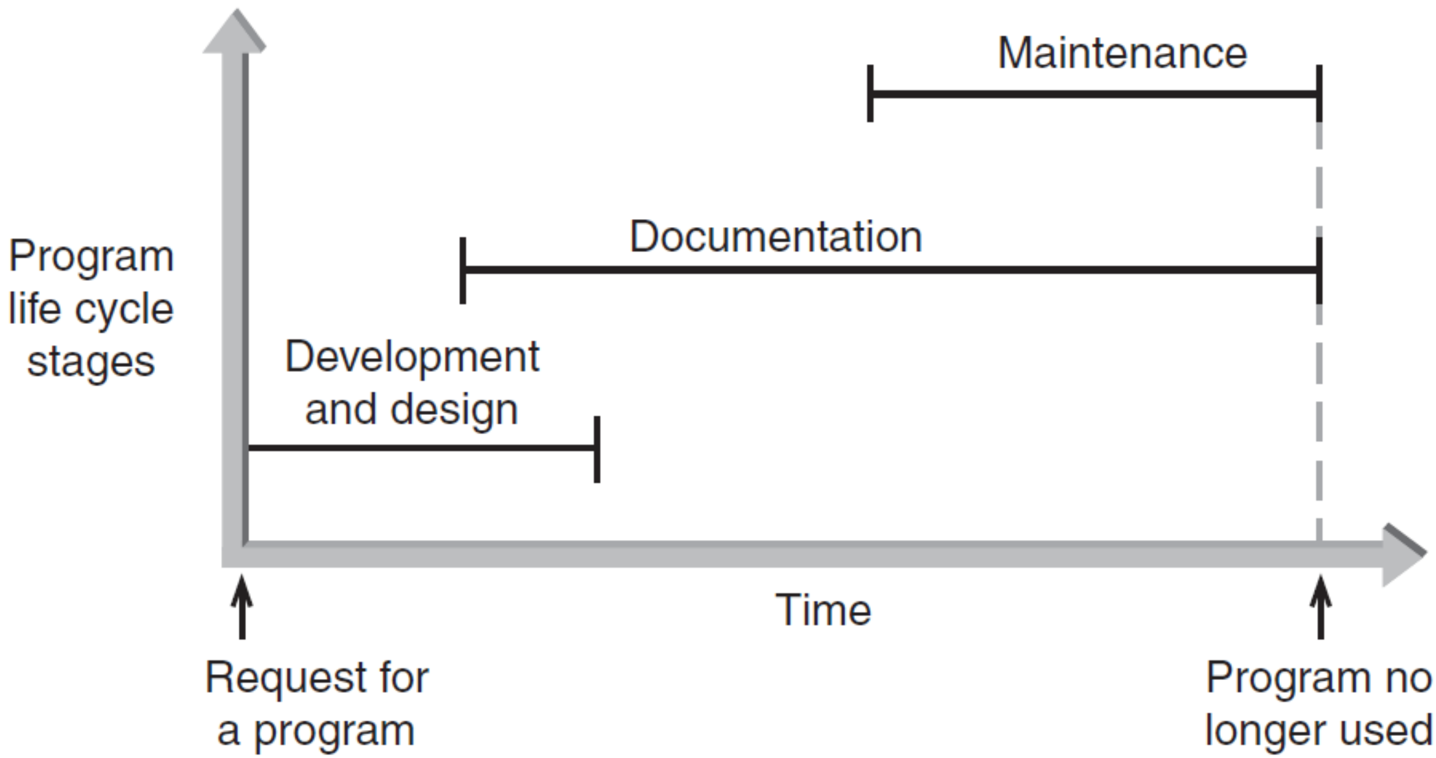


Figure 1.2 The three phases of program development

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)

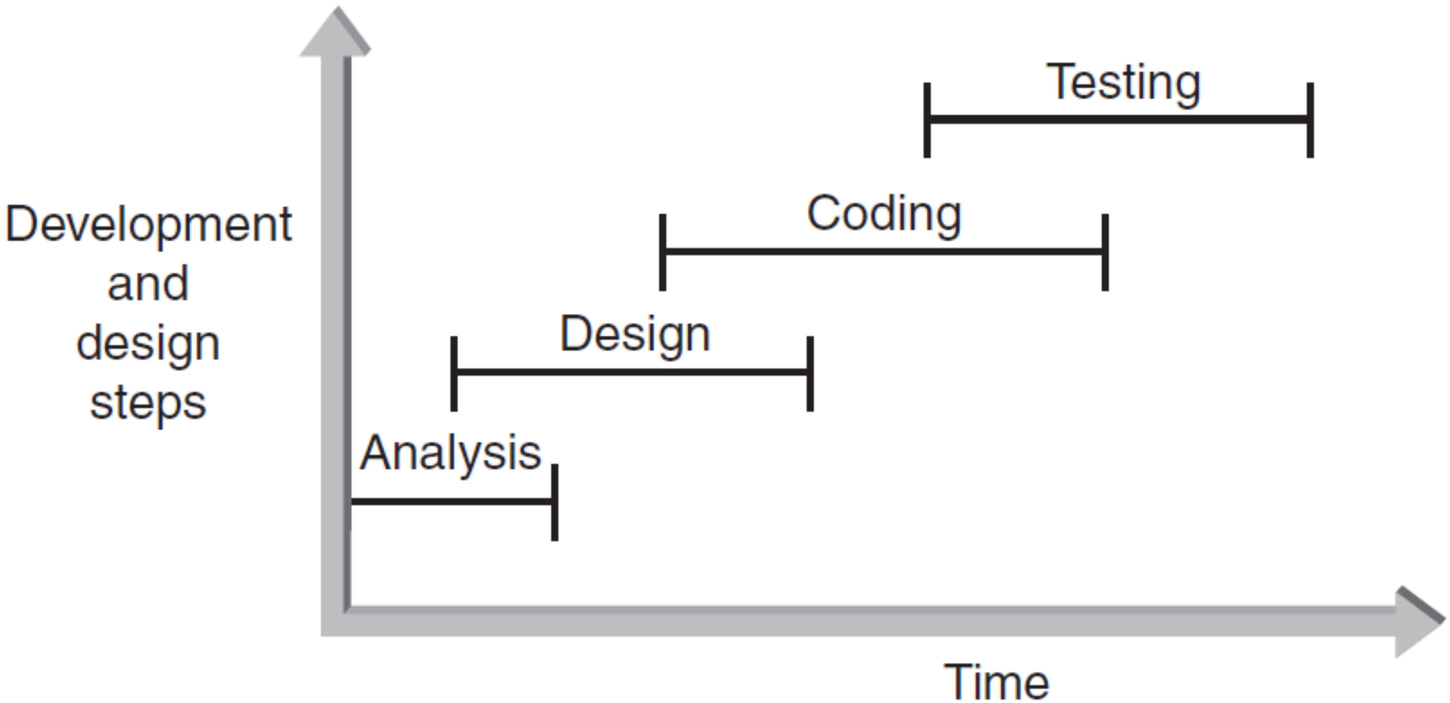


Figure 1.3 The development and design steps

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
  - Both items referred to as the problem's input/output (I/O)

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)
  - Example: a first-level structure diagram for an inventory tracking system

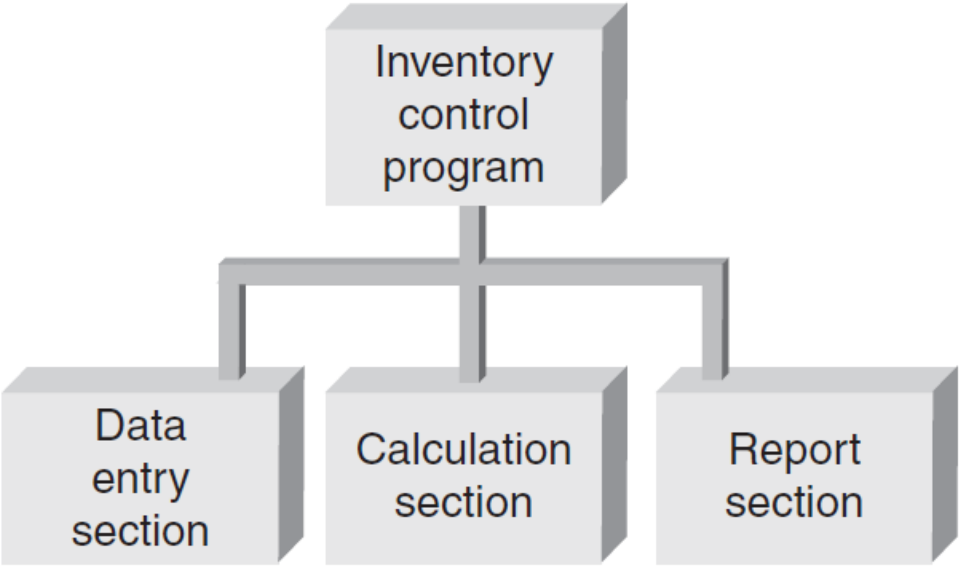


Figure 1.4 A first-level structure diagram

Phase I: Development and Design (continued)

- Step 2: Develop a Solution (continued)
  - Example: a second-level structure diagram for an inventory tracking system with further refinements

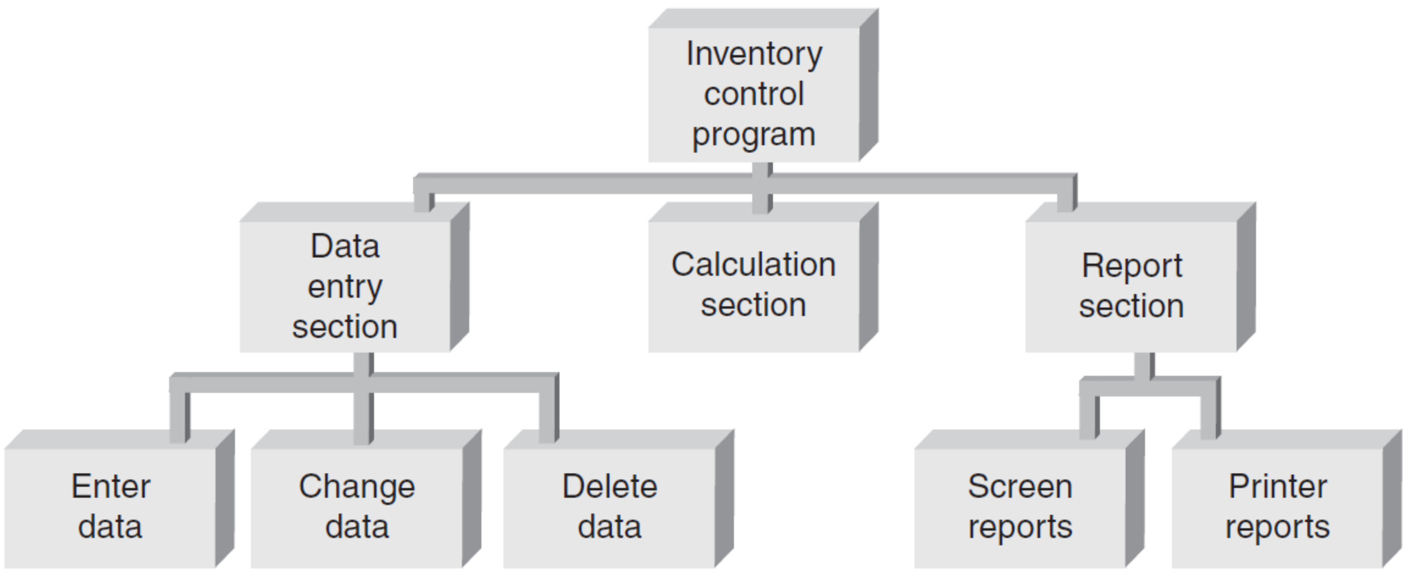


Figure 1.5 A second-level structure diagram

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
    - …

Phase I: Development and Design (continued)

- 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

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 1.3 lists the comparative amount of effort typically expended on each development and design step in large commercial programming projects

Step	Effort
Analyze the problem	10%
Develop a solution	20%
Code the solution (write the program)	20%
Test the program	50%

Table 1.3 Effort Expended in Phase I

Phase II: Documentation

- Five main documents for every problem solution:
  - 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
- Good documentation vital for effective maintenance

Preliminary Four: Algorithms

- **Algorithm:** Step-by-step sequence of instructions
  - Must terminate
  - Describes how the data is to be processed to produce the desired output
- **Formula:** Mathematical equations
- **Flowchart:** Diagrams with symbols

Preliminary Four: Algorithms (continued)

- 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.

$$
\begin{array}{r}
1 \\
2 \\
3 \\
4 \\
. \\
. \\
. \\
98 \\
99 \\
+100 \\
\hline
5050
\end{array}
$$

Figure 1.6 Summing the numbers 1 to 100

Preliminary Four: Algorithms (continued)

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

$$\left.\begin{array}{r} 1 + 100 = 101 \\ 2 + 99 = 101 \\ 3 + 98 = 101 \\ 4 + 97 = 101 \\ . \quad\quad . \\ . \quad\quad . \\ 49 + 52 = 101 \\ 50 + 51 = 101 \end{array}\right\} \text{50 groups} \rightarrow (50 \times 101 = 5050)$$

Figure 1.6 Summing the numbers 1 to 100 (continued)

Preliminary Four: Algorithms (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)

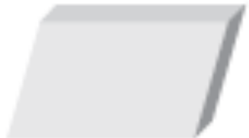
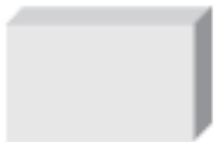
Symbol	Name	Description
	Terminal	Indicates the beginning or end of a program
	Input/output	Indicates an input or output operation
	Process	Indicates computation or data manipulation
	Flow lines	Used to connect the other flowchart symbols and indicate the logic flow
	Decision	Indicates a program branch point

Figure 1.7 Flowchart symbols

	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 1.7 Flowchart symbols (continued)

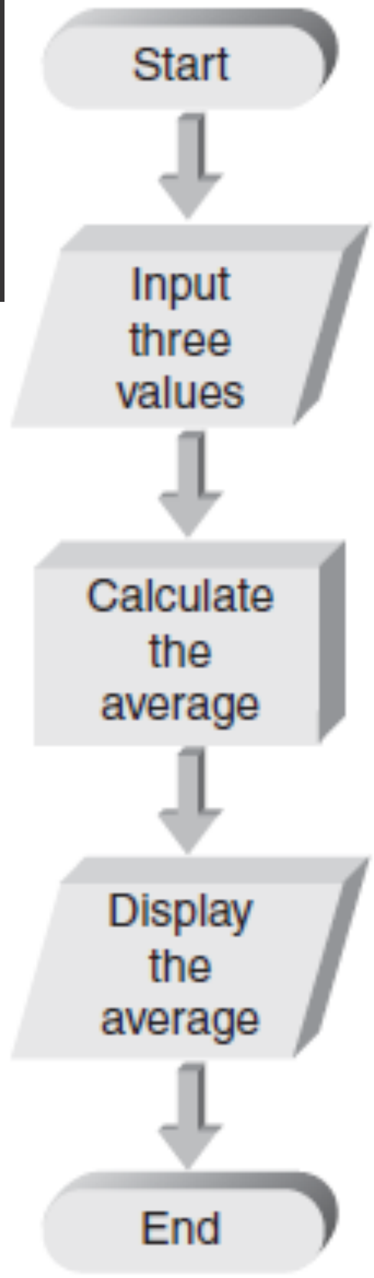


Figure 1.8 Flowchart for calculating the average of three numbers

Preliminary Four: Algorithms (continued)

- **Select an algorithm and understand the required steps**
- **Coding the algorithm**

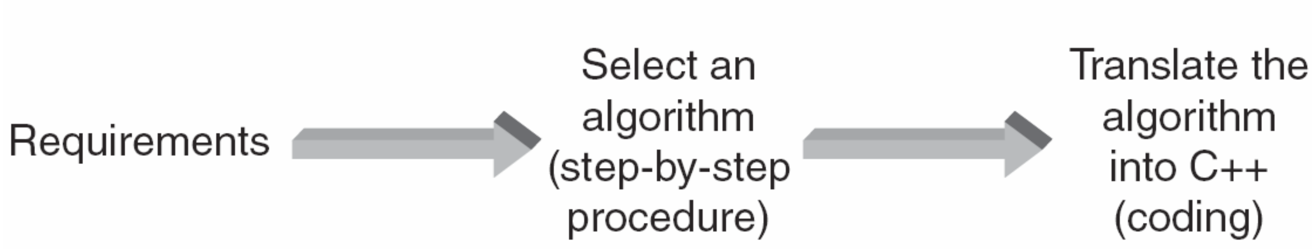


Figure 1.9 Coding an algorithm

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

```
11000000  000000000001  000000000010
11110000  000000000010  000000000011
```

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

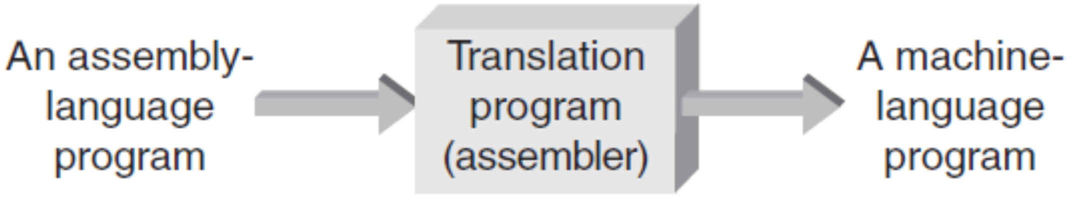


Figure 1.10 Assembly-language programs must be translated

Example

- **Machine Language to Assembly Language**

```
11000000 000000000001 000000000010
11110000 000000000010 000000000011
```

Machine language

```
ADD 1, 2
MUL 2, 3
```

Assembly Language

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, Python

Low- and High-Level Languages (continued)

- **Source code:** The programs written in a high- or low-level language
  - Source code must be translated to machine instructions in one of two ways:
    - **Interpreter:** Each statement is translated individually and executed immediately after translation (Python)
    - **Compiler:** All statements are translated and stored as an executable program, or object program; execution occurs later
      - C++ is predominantly a compiled language

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
  - Also called OS
- Tasks performed by the OS include:
  - Memory management
  - Allocation of CPU time
  - Control of input and output
  - Management of secondary storage devices

Application and System Software (continued)

- **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

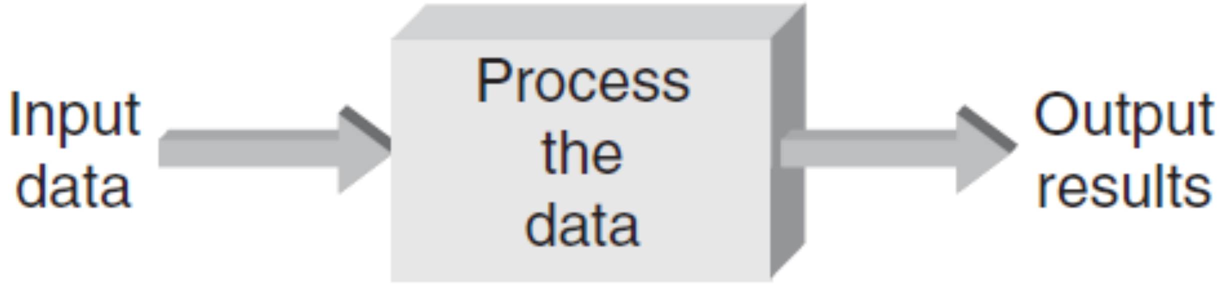


Figure 1.12 Basic procedural operations

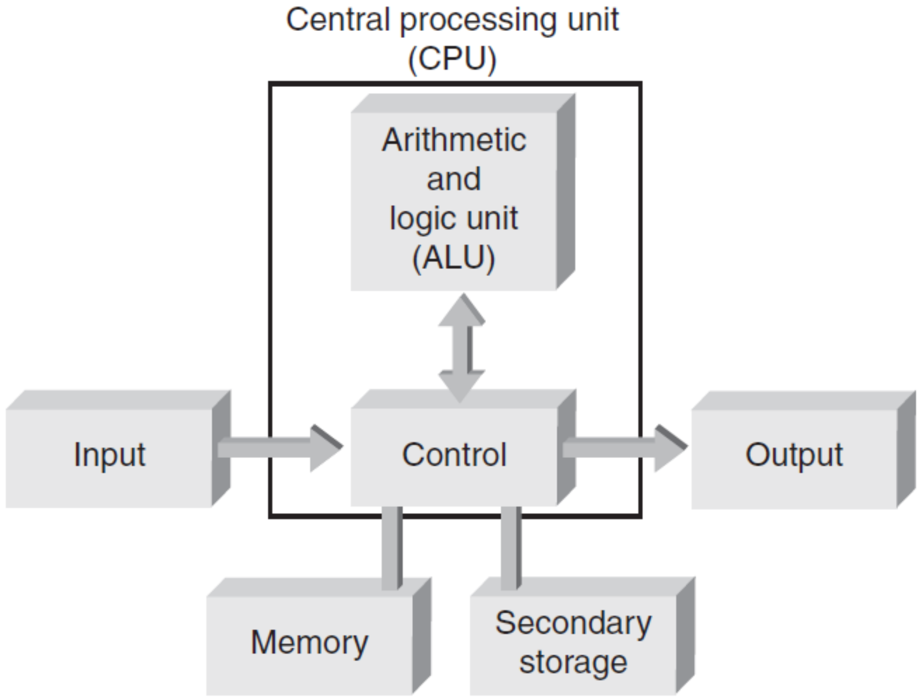
The Development of C++ (continued)

- 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)

- **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:

Word Size	Maximum Integer Value	Minimum Integer Value
1 byte	127	-128
2 bytes	32,767	-32,768
4 bytes	2,147,483,647	-2,147,483,648

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

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