William T. Doan

A First Trimesterly Review of CS 1436

Written for Dr. Brian Wescott Ricks and his 001 and 009 sections

September 2024

Dedication

This work is dedicated to Dr. Brian Ricks for having seen potential in one young student—one short year ago.

Contents

1

Introduction to Computers and Programming

1.1 Why Program?

- A computer is a programmable electronic device that stores, retrieves, and processes a large quantity of data both quickly and accurately.

1.2 Computer Systems: Hardware and Software

- A computer is a system of hardware and software devices/components.
- The physical components of the computer are called hardware.

1.2.1 Major Hardware Component Categories

1. Central Processing Unit (CPU)
2. Main memory
3. Secondary storage devices
4. Input devices
5. Output devices

1.3 Central Processing Unit (CPU)

- The central processing unit, or CPU, is the part of the computer that runs programs.
- The CPU is comprised of two components: the control unit and the arithmetic and logic unit (ALU).

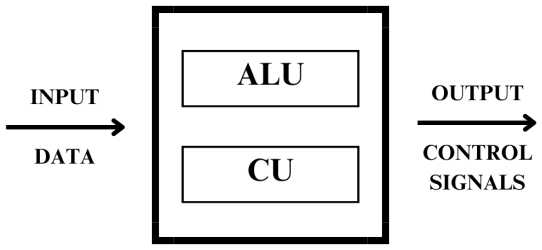

Fig. 1.1. Organization of the CPU

1.4 Main Memory

- Main Memory is rapid-access, relatively low-capacity storage for the instructions and data of executing/running programs.
- Main Memory is volatile. Data and instructions stored here are erased when the program terminates or the computer is turned off.
- Also called Random Access Memory (RAM).
- Memory is divided into locations called bytes.
  - One byte is enough memory to store a letter of the alphabet or a small number.
- **Addresses** – Each byte in memory is identified by a unique number known as an address.

1.5 Secondary Storage

- Secondary storage is long-term, high-capacity storage for programs and data not currently in use.
- Secondary storage is persistent (non-volatile); data is retained when the program stops running or the computer is turned off.

1.6 Input & Output Devices

- **Input** is data the computer accepts from outside for processing.
- **Output** is data the computer sends to the outside.

1.7 Software – Programs that Run on a Computer

Categories of software:

- **System software**: programs that manage the computer hardware and the programs that run on them. Examples: operating systems, utility programs, and software development tools.

- **Application software**: programs that provide services to the user. Examples: word processors, spreadsheets, image editing software, games, and other programs to solve specific problems.

1.8 Programs and Programming Languages

- A **program** is a set of instructions that the computer follows to perform a task.

1.8.1 Example Programming Problem

Write a program that calculates and displays an hourly worker's gross pay given the hours worked and pay rate entered by the user, when the program runs.

Here is a list of things the program should do:

1. Display a message on the screen asking "How many hours did you work?"
2. Wait for the user to enter the number of hours worked. Once the user enters a number, store it in memory.
3. Display a message on the screen asking, "How much do you get paid per hour?"
4. Wait for the user to enter an hourly pay rate. Once the user enters a number, store it in memory.
5. Multiply the number of hours by the hourly pay rate and store the result in memory.
6. Display a message on the screen that tells the amount of money earned. The message must include the result of the calculation performed in Step 5.

1.9 Algorithms and Machine Language

- The list of steps forms an **algorithm**. An algorithm is a set of well-defined, ordered steps for performing a task or solving a problem.
- Although we can easily understand this algorithm, it is not understandable by a computer.
- Computers only understand algorithms expressed in **machine language**.

1.9.1 Machine Language

- Computers execute programs composed of numeric instructions. Machine language is the numeric language understood by a computer processor.
- A machine language instruction is a binary pattern (containing only 1's and 0's).

- It takes multiple instructions to perform a simple mathematical calculation.

  Example of machine language instructions:

```
0001 0011 1001 0100
0011 0001 0010 0000
0011 0010 0011 0000
0110 0001 0001 0010
1001 0011 0001 0000
```

1.10 High-Level Languages

- Programs today are typically written in programming languages that use words and symbols instead of binary patterns.
- Special software is used to convert programs from these languages to the machine language understood by a particular computer processor.
- In C++, the program that does the conversion is called a **compiler**.

1.11 Integrated Development Environments (IDEs)

- An **Integrated Development Environment** (IDE) combines the tools needed to write, compile, and debug a program into a single software application.

1.12 What is a Program Made of?

Common elements in high-level programming languages:

- **Key Words**
- **Programmer-Defined Identifiers**
- **Operators**
- **Punctuation**
- **Syntax**

1.12.1 C++ Key Words

1.12.2 Syntax

- The rules of grammar that must be followed when writing a program.
- Controls the use of key words, operators, programmer-defined symbols, and punctuation.

1.12.3 Variables

- A **variable** is a named storage location in the computer's memory for holding a piece of data.
- A variable can hold one data item at a time, but the data held can be changed while the program is running.
- A variable holds a specific type of data.

1.13 Input, Processing, and Output

Three steps that programs typically perform:

1. Gather input data:
   - From keyboard
   - From files on disk drives
   - From other hardware devices
2. Process the input data.
3. Display the results as output:
   - Send it to the screen
   - Write it to a file
   - Send it to another hardware device

1.14 The Programming Process

- The programming process includes:
  - Analysis
  - Design
  - Coding
  - Testing
  - Debugging

1.14.1 Design

- **Pseudocode** is a cross between human language and a programming language that is often used by programmers to create an algorithm.
- An **algorithm** is an ordered sequence of well-defined steps for solving a problem or accomplishing a task.

  Sample pseudocode:

```
Get the number of hours worked
Get the hourly pay rate
Multiply the hours by the pay rate
and store the calculated pay
Display the pay
```

1.15 Procedural and Object-Oriented Programming

- Two popular methodologies for developing computer programs are **procedural programming** and **object-oriented programming**.
- C++ can be used to write programs using either of these methodologies.
- In this course, our programs will be procedural in nature.

1.15.1 Procedural Programming

- Focus is on the tasks to be performed.
- Procedures/functions are written to perform the various tasks.
- Procedures contain their own variables and commonly share variables with other procedures.

1.15.2 Object-Oriented Programming

- Focus is on data, designing entities (objects/classes) which contain the data and the means to manipulate the data (methods).
- Objects encapsulate their data and methods.
- Objects do not know how other objects are implemented and cannot directly access another object's data.

2

Introduction to C++

2.1 The Parts of a C++ Program

Let's examine a simple C++ program from `Pr2-1.cpp`:

```
// A simple C++ program that
// displays a message on the screen
#include <iostream>
using namespace std;
int main()
{
    cout << "Programming is fun";
    return 0;
}
```

Each part of the program serves a specific purpose:

- `// A simple C++ program that displays a message on the screen` — This is a **comment**.
- `#include <iostream>` — This is a **preprocessor directive**.
- `using namespace std;` — Specifies which **namespace** to use.
- `int main()` — Marks the **beginning of the function named** `main`.
- { and } — The **beginning and end of the block** for `main`.
- `cout << "Programming is fun";` — An **output statement**.
- `return 0;` — Sends a value of 0 to the operating system.

2.2 Functions

- A **function** is a named subroutine, a collection of programming statements for performing a specific task that can be executed by name.
- Every C++ program must have a function called `main`. The function `main` is the starting point for the execution of a C++ program.

- Every function has a **header** that provides important information about the function:
  - **Name**
  - **Return type**
  - **Parameters**
- Every function has a **body**; which is the group of statements that perform the task of the function.
- The body of every function begins with a left (opening) brace, {, and ends with a right (closing) brace, }.

2.2.1 Special Characters

Char	Name	Description
`//`	Double slash	Marks the beginning of a comment
`#`	Pound sign	Marks the beginning of a preprocessor directive
`< >`	Angle brackets	Encloses a filename in a `#include` directive
`( )`	Parentheses	Used when naming a function
{ }	Braces	Encloses a group of statements
`" "`	Quotation marks	Encloses a string of characters
`;`	Semicolon	Marks the end of a complete programming statement

Table 2.1. Special Characters in C++

2.3 The `cout` Object

- The `cout` object is the standard output object.
- It displays output on the computer screen.
- To use `cout`, you must include the `iostream` header file.
- `cout` is classified as a stream object, which means it works with streams of data.
- The stream insertion operator, `<<`, is used to send data items to `cout`.

Example:

```
cout << "Programming is fun!";
```

2.3.1 Multiple Insertions

- It is possible to send more than one data item to `cout`.

Example:

```
cout << "Hello " << "there!";
```

2.3.2 The `endl` Stream Manipulator

- If the `endl` manipulator is inserted into a stream, it will start a new line of output.

  Example:

```
cout << "Hello" << endl << "class!";
```

  This produces the following output:

```
Hello
class!
```

2.3.3 The Newline Escape Sequence `\n`

- You can also use the newline escape sequence `\n` to start a new line of output.

  Example:

```
cout << "Programming is\nfun!";
```

  This produces:

```
Programming is
fun!
```

2.4 The `#include` Directive

- Instructs the preprocessor to insert the contents of another file into the program.
- Example: `#include <iostream>`
- These are not part of the "core" of the C++ language. They are part of the input-output stream library.
- Do not place a semicolon at the end of a preprocessor directive.

2.5 Variables, Literals, and Assignment Statements

2.5.1 Variables

- A variable is a named location in the computer's memory that can store a value that may be changed during the execution of the program.
- In C++, variables must be defined before they are used.
- A variable definition consists of the data type followed by the name of the variable.

  Example:

```
int number;
```

2.5.2 Variables and Literals

- After defining the variable, you can assign a value to it.

  Example:

```
number = 5;
cout << "The value in number is " << number << endl;
```

2.5.3 Literals

- A literal is a piece of data that is written directly into the source code of a program.
- Examples:
  - `"hello, there"` (string literal)
  - `12` (integer literal)
  - `12.5` (floating point literal)

3

Expressions and Interactivity

3.1 The `cin` Object

- The `cin` object is the standard input object.
- It is used for reading input from the console (or keyboard).
- You must `#include <iostream>` to use the `cin` object.
- Data is read from `cin` using the stream extraction operator, `>>`.
- The data read is stored in variable(s).

3.1.1 Using `cin` for Input

- Gathering console input is normally a two-step process:
  1. Use the `cout` object to display a prompt on the computer screen.
  2. Use the `cin` object to read a data item from the keyboard.
- A prompt is a message that instructs the user to enter data.

  Example:

```
int month;
cout << "Enter the whole number corresponding to the month: ";
cin >> month;
```

3.1.2 How `cin` Works

- The `cin` object causes a program to wait until data is typed at the keyboard and the `[Enter]` key is pressed.
- The keystrokes typed by the user are echoed on the standard output device so the user can see what they typed.
- Pressing the `[Enter]` key results in a newline being printed on the screen.
- The stream extraction operator, `>>`, automatically converts the data read to the data type of the variable that is its right operand.

- It is possible to cascade a sequence of stream extraction operators in a `cin` statement to read multiple data items.

  Example:

```
int age;
double weight;
cin >> age >> weight;
```

3.1.3 Potential Buffer Issues

- When mixing inputs of different types or when the user input does not match the expected format, data can remain in the input buffer, leading to unexpected behavior.
- For example, if the user types `10.7` when an integer is expected, only `10` will be read into the integer variable, and the `.7` will remain in the buffer.

3.1.4 Defensive Programming with `cin`

- Write clear prompts to guide the user.
- Avoid reading multiple inputs in a single `cin` statement unless they are components of one entity.
- Prompt for and read each data item separately to reduce the chance of input errors.

3.2 Mathematical Expressions

- In C++, an expression is an arrangement of literals, identifiers, and operators that evaluates to a single value.
- Mathematical expressions can be built with several operators.
- Operators are applied according to the rules of operator precedence and associativity.

3.2.1 Order of Operations

1. Unary minus has higher precedence than multiplication, division, and modulus operators.
2. Multiplication, division, and modulus operators have higher precedence than addition and subtraction.
3. Operators of the same precedence are evaluated from left to right.

Operators	Description	Associativity
`( )`	Parentheses	
`- (unary)`	Unary minus	Right to left
`*, /, %`	Multiplication, Division, Modulus	Left to right
`+, -`	Addition, Subtraction	Left to right

Table 3.1. Operator Precedence and Associativity

3.2.2 Operator Precedence and Associativity

3.2.3 Grouping with Parentheses

- Parentheses may be used to force the evaluation of subexpressions.
- Example:

$$\texttt{slope} = \frac{y_2 - y_1}{x_2 - x_1};$$

- In code:

```
slope = (y2 - y1) / (x2 - x1);
```

3.2.4 No Exponentiation Operator in C++

- There is no exponentiation operator in C++.
- To raise a number to a power, use the `pow` function from the `cmath` library.
- Example:

```
#include <cmath>
area = pow(side, 2);
```

3.3 Overflow and Underflow

- **Overflow** occurs when a variable is assigned a value that is too large for its data type.
- **Underflow** occurs when a variable is assigned a value that is too small for its data type.
- For integer overflow, values may wrap around.
- For floating-point types, the behavior may vary by system.

3.4 Type Conversion

- When an operator's operands are of different data types, C++ automatically converts the operands to the same data type.
- This automatic type conversion is known as coercion.

3.4.1 Hierarchy of Data Types

- Data types are ranked by their size and precision.
- When performing operations, operands are converted to the higher-ranking type.

3.4.2 Type Casting

- A type cast expression is used to manually convert a value to a different data type.
- Syntax:

```
static_cast<data-type>(expression)
```

- Example:

```
int x = 13;
double result;
result = static_cast<double>(x) / y;
```

3.5 Combined Assignment Operators

- C++ provides shorthand operators for common arithmetic operations that modify a variable.
- Examples:
  - `balance += deposit;` is equivalent to `balance = balance + deposit;`
  - `count -= 1;` is equivalent to `count = count - 1;`

3.6 Formatting Output

- The `cout` object provides ways to format data as it is being displayed.
- Stream manipulators are used for formatting.
- To use certain manipulators, you must `#include <iomanip>`.

3.6.1 `setprecision`

- Controls the number of significant digits displayed.
- Example:

```
cout << setprecision(4) << number;
```

3.6.2 `fixed` and `showpoint`

- The `fixed` manipulator displays floating-point numbers in fixed-point notation.
- The `showpoint` manipulator forces the display of trailing zeros.

3.6.3 `setw`, `left`, and `right`

- The `setw` manipulator sets the minimum field width for the next output.
- The `left` and `right` manipulators control justification.

3.7 Working with Characters and `string` Objects

3.7.1 Reading Strings with Spaces

- The `getline` function reads an entire line, including spaces, into a `string` object.
- Syntax:

```
getline(cin, name);
```

3.7.2 Reading Characters

- To read a single character, including whitespace, use `cin.get()`.
- Example:

```
char ch;
ch = cin.get();
```

3.7.3 Using `cin.ignore`

- The `cin.ignore` function skips characters in the input buffer.
- Useful when switching from `cin >>` to `getline` or `cin.get()`.

3.8 Mathematical Library Functions

- The `cmath` library provides various mathematical functions.
- Common functions include `pow`, `sqrt`, `sin`, `cos`, `tan`, `log`, `exp`.

3.8.1 Random Numbers

- The `rand` function generates pseudo-random numbers.
- To seed the random number generator, use `srand` with a seed value.
- Example:

```
#include <cstdlib>
#include <ctime>
srand(time(0));
int randomNumber = rand();
```

3.9 Hand Tracing a Program

- Hand tracing involves simulating the execution of a program by tracking the values of variables.
- Useful for debugging and understanding program flow.

4

Number Systems

4.1 Number Systems

- Binary is a positional numbering system (as is decimal) meaning that the position of the digits with respect to a radix point implies the value associated with the sequence of digits.
- **Decimal**
  - Base 10
  - Ten distinct digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
  - A decimal number represents a power series of 10
- **Binary**
  - Base 2
  - Two distinct digits (0, 1)
  - A binary number represents a power series of 2

4.2 Decimal Numbers

Given the decimal integer: 1368

$$1368 = (1 \times 10^3) + (3 \times 10^2) + (6 \times 10^1) + (8 \times 10^0)$$

4.3 Binary Numbers

Given the unsigned binary integer: `0101`

$$0101_2 = (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 5_{10}$$

4.3.1 Conversion from Binary to Decimal

1. Form the power series represented by the number.
2. Evaluate the series using decimal arithmetic.

Example:
What is the decimal equivalent of the unsigned binary value `1101011`?

$$\begin{aligned}1101011_2 &= (1 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &= 64 + 32 + 0 + 8 + 0 + 2 + 1 \\ &= 107_{10}\end{aligned}$$

4.3.2 Powers of 2

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- $2^8 = 256$

4.3.3 Conversion from Decimal to Binary

1. Repeatedly subtract the largest power of 2 that produces a nonnegative result.
2. Construct the binary number by placing ones in the positions corresponding to the powers of two subtracted in step 1 and zeros in all the other positions.

Example:
Convert the decimal value 50 to binary.

$$\begin{aligned}50 - 2^5(32) &= 18 \\ 18 - 2^4(16) &= 2 \\ 2 - 2^1(2) &= 0\end{aligned}$$

So, $50_{10} = 110010_2$.

4.4 Hexadecimal Numbers

- Hexadecimal is another positional numbering system that is frequently used as a shorthand for binary.
- **Hexadecimal**
  - Base 16
  - Sixteen distinct digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)
  - A hexadecimal number represents a power series of 16

4.4.1 Conversion from Binary to Hexadecimal

1. Moving from the radix point, group the binary digits into groups of four. (Insert leading zeros as needed at the left of the binary value).
2. Replace each group of four digits with the equivalent hexadecimal digit.

Example:
What is the hexadecimal equivalent of the binary value `100011111001`?

$$\begin{aligned} 1000\ 1111\ 1001 &= 8\ F\ 9 \\ &= (8F9)_{16} \end{aligned}$$