Tony Gaddis 5th Ed
Starting Out with C++

COMPUTER SCIENCE

CHAPTER 1
INTRODUCTION TO COMPUTERS AND PROGRAMMING

Mohammad Al-Husseini

Why Program?

- Computers are excellent at performing repetitive and analytical actions quickly
  - Who would benefit from this? Everyone
    - Accountants balance books and profits/losses
    - Factory workers control manufacturing machines
    - Mechanics test computer systems in vehicles
    - Scientists run millions of calculations on data
    - End users benefit from drivers and software
  - But these individuals likely do not possess the skills to write the software they use and need

Hardware and Software

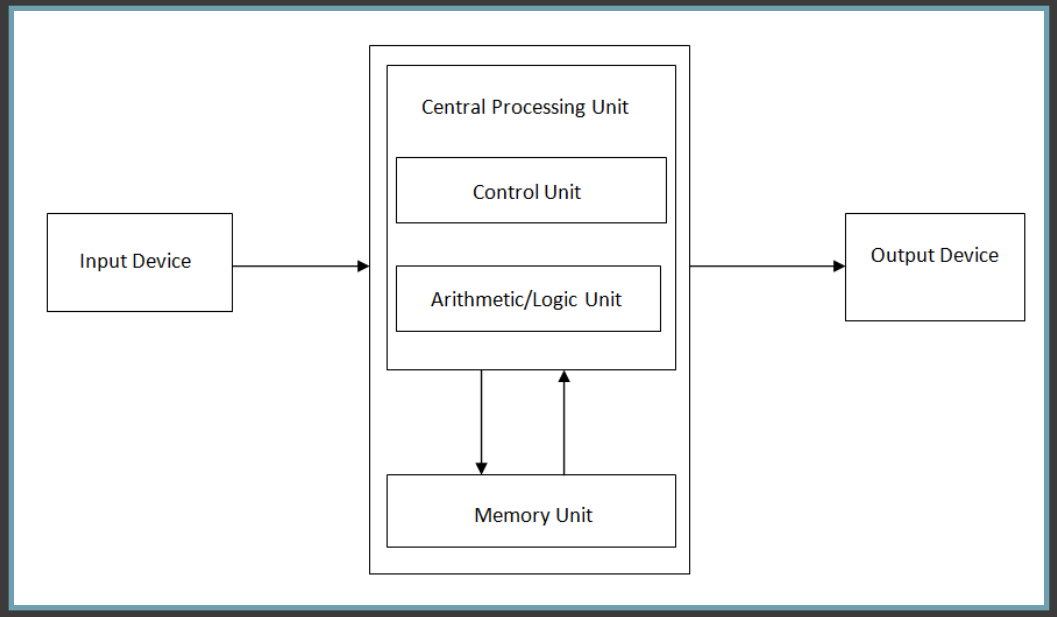
- Hardware
  - Central Processing Unit (CPU)
    - Fetches, Decodes, and Executes instructions
    - Consists of two components
      - Arithmetic and Logic Unit (ALU) – performs mathematical operations
      - Control Unit – sends electrical signals to the other components of the computer to perform operations
  - Main Memory (RAM)
    - Divided into sections consisting of 8 bits or a byte
    - Each byte is assigned a unique address
    - When power is lost, the memory is erased

Hardware and Software

- Hardware
  - Secondary Storage Devices (HDD/SSD/USB)
    - Long term memory that does not lose data during power loss
    - Stores programs and data until it is needed and loaded into RAM, which is faster
  - Input Devices
    - Any interface which puts information into the computer
      - Mouse, keyboard, microphone, CD/DVD, USB, internet, etc.

Hardware and Software

- Hardware
  - Output Devices
    - Any device which sends information out of the computer
      - Monitor, speakers, printer, CD/DVD-R, USB, internet, etc.



Hardware and Software

- Software
  - Two categories:
    - Operating Systems – manages the computer's hardware and controls their processes
      - Single Tasking – Capable of running only a single program at a time, such as MS-DOS
      - Multitasking – Capable of running multiple programs at a time
        - Note: Still only runs one instruction at a time, but very quickly (3 GHz processor – 3 billion cycles per second)
      - Also differentiates as single-user or multi-user
    - Applications / Programs – makes the computer useful to the user by solving problems or running operations

What Is a Program?

- A set of instructions that tells the computer how to solve a problem or perform a task
  - Gross Pay
    - Display: “How many hours did you work?”
    - Wait: For user input
    - Store: The user input in memory
    - Repeat: For Hourly Pay
    - Multiply: Hours Worked by Hourly Pay
    - Store: The result in memory
    - Display: “You earned “ result “ money.”
  - This is an algorithm (a set of instructions)

What Is a Program?

- But there's one problem
  - Computers don't speak English
  - They speak Machine Code – a string of binary numbers (10100011)
- Okay, there's another problem
  - Human's don't speak Machine Code
  - And that Machine Code can very between systems
- Thus a middle ground: Programming Languages (C++)
  - Which get converted into Machine Code

What Is a Program?

- Gross Pay Example

```
// This progam calculates the user's pay.
#include <iostream>
using namespace std;

int main()
{
   double hours, rate, pay;

   // Get the number of hours worked.
   cout << "How many hours did you work? ";
   cin >> hours;

   // Get the hourly pay rate.
   cout << "How much do you get paid per hour? ";
   cin >> rate;

   // Calculate the pay.
   pay = hours * rate;

   // Display the pay.
   cout << "You have earned $" << pay << endl;
   return 0;
}
```

  - How many hours did you work? 10 [Enter]
  - How much do you get paid per hour? 15 [Enter]
  - You have earned $150

Programming Languages

- Low-Level Programming Languages
  - Closer to Machine Code
    - Ex. Assembly
- High-Level Programming Languages
  - Closer to English
    - Ex. BASIC, FORTRAN, COBOL, Pascal, C/C++/C#, Java, Visual Basic, etc.
- Portability – a program's ability to be run on various types of systems
- C++ excels for it's high-level ease with low-level functionality and reasonable portability

Code Transformation

- Source Code
  - Written by the programmer as seen on slide 9
  - Still primarily English due to being high-level
- Preprocessor
  - Takes the source code and modifies it based on any preprocessor directives (#)
  - Creates a modified source code file
- Compiler
  - Converts the modified source code into machine language, checking for syntax errors
  - Creates an object code file

Code Transformation

- Linker
  - Takes the object code and mixes it with libraries and hardware specific code
  - Creates an executable code file (exe)
- This .exe file can then be run on the computer

What Makes a Program?

- Key Words
  - Reserved words that have special meaning and can only be used for their intended purpose
  - All keywords are lowercase
    - #include, using namespace, main, double, return
- Programmer-Defined Identifiers
  - Words created with by the programmer to represent variables or routines
    - hours, rate, pay

What Makes a Program?

- Operators
  - Perform operations on one or more operands, usually numbers or values
    - * represents multiplication between hours and rate
    - = sets pay equal to the calculation on the right
- Punctuation
  - Characters that mark the start or end of a statement or separate items in a list
    - , separating variables
    - ; marking the end of a statement

What Makes a Program?

- Syntax
  - The grammar dictating how keywords and operators may be used and when to use punctuation
- Note that whitespace does not matter in the program (aside from strings)
  - But good use of spacing should be used to improve code readability

What Makes a Program?

- Lines and Statements
  - Lines represent the horizontal rows of code
  - Statements are any complete instruction of code, causing the computer to perform an action, generally terminated by a semicolon
    - cout << “How many hours did you work? “;
    - Note that this is a line because it does not appear on two lines and a statement because it is terminated by a semicolon

What Makes a Program?

- Variables
  - Represents a location in memory with a distinct address that holds a piece of information
  - Think of a mailbox
  - The name we assign to a variable is symbolic, usually representing the kind of data it is going to hold
    - pay holds the result of hours * rate
    - We could refer to pay by its address, but it is more complex, abstract, and difficult to remember

What Makes a Program?

- Variable
  - Two general types of data: numbers and characters
  - Numbers can be separated further
    - Integers: 5, 7, -129, 32154
    - Floating-Point: 3.14159, -6.7, 0.14
  - You must know what type of variable to create based on the type of data it will hold
    - hours, rate, and pay can hold floating-point or decimal numbers, so they are declared double
    - More on that in Chapter 2

Input, Process, Output

- Input
  - Get information from the user
    - cin >> hours;
    - cin >> rate;
- Process
  - Do something on that information
    - pay = hours * rate;
- Output
  - Give the result to the user so that it is useful or request information
    - cout << “You have earned $” << pay << endl;
    - cout << “How many hours did you work? “;

Designing and Creating

1. Define what the program will do.
2. Visualize the program running on the computer.
3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to model the program.
4. Check the model for logic errors.
5. Type the code, save, and compile.
6. Correct any errors found. Repeat steps 5 and 6 until no more errors are found.

Designing and Creating

7. Run the program with test data.
8. Correct any run-time errors or incorrect results. Repeat steps 5 through 8 until completely correct.

Procedural and OOP

- You will learn both methods over the duration of this course
  - Procedural Programming is focused on passing data between multiple procedures
  - Object-Oriented Programming (OOP) is focused on storing data and procedures that work on that data within an object and passing the object