**W1 Chapter1 Introduction**

|  |
| --- |
| Objectives: Upon completion of today’s class, you should be able to   * draw the architecture of a computer and explain its major components and their roles * investigate the information of your system * explain the difference between machine languages and higher-level programming languages * explain what a compiler does and the difference between compiler and interpreter * apply the g++ program with different options and explain what type of output each option generates * apply essential Linux commands in our programming environment * explain the difference between compile-time and run-time errors * explain the concept of an algorithm and pseudocode. |

Note: the following notes were prepared using the textbook and AI platforms such as ChatGPT and Gemini

1. **Computer Architecture**
   1. **Concept**

A computer architecture is the blueprint that defines how a computer is designed and built. It's about understanding the fundamental components and how they work together to create a functional system. -- gemini

A diagram of a computer hardware system

Description automatically generated

**1.2 Key Components of Computer Architecture**

* **Central Processing Unit (CPU, or Processor)**
  + the brain of the computer
  + executes instructions and performs calculations
  + Like the foreman on a construction site, overseeing all the work
* **Random Access Memory(RAM, or simply memory)**
  + Temporary and volatile storage that the CPU uses to store and quickly access data including a program
  + Loses the in the RAM when the power is turned off
* **Input/Output Devices:**
* the ways the computer communicates with the outside world.
* Keyboards, mice, monitors, printers, etc.
* **Storage Devices:**
* store data permanently even when the power is turned off
* e.g., Hard drives (HDD), solid-state drives (SSDs), CD drive, etc.

**1.3 Instruction Set Architecture (ISA)**

* A list of commands that a computer's CPU (understands and can execute
* Different ISAs for Different CPUs
* X86 and x86-64: developed by Intel and used by both Intel and AMD processors. x86 is for 32-bit systems, while x86-64 (also known as AMD64) extends it to 64-bit systems; popular in personal computers and servers
* ARM: widely used in mobile devices (smartphones and tablets)

**1.4 How They Work Together**

* When you open a program or a file, the operating system *loads* it *from permanent storage (like a hard drive or SSD) into RAM*. This allows the CPU to access and manipulate the data much faster than if it were reading from HDD or SDD
* **How CPU execute a program once it is loaded into RAM**

**CPU simply repeats the following steps over and over; this is called CPU instruction cycle**

* Instruction Fetch: fetches an instruction from memory.
* Instruction Decode: decodes the instruction to understand what it needs to do.
* Instruction Execute: executes the instruction, which might involve performing calculations, loading data from memory, or storing data.
* Result Writeback: writes the result of the operation back to memory.
* **Bus as the Pathway between Components**
* a bus is a set of pathways used for transferring data and instructions between different components, like the processor (CPU), memory (RAM), and other hardware

1. **Investigate the Architecture of Your Computer**

* Windows users: In the search bar, type ‘System’ and select ‘System Information’
* MAC users: Click the Apple Logo -> Select “About This Mac” -> View the Overview tab
* Write down the information of the components below
* Operating System:
* System Name:
* System Type:
* CPU Type:
* RAM size:
* Input/Output Devices:

1. **High-level language vs Machine Code**

* High-Level Language:
* a type of computer language that is designed to be easy for humans to read and write
* provide a level of abstraction from the computer’s hardware
* e.g., C, C++, Java, Python, C#, Javascript, etc.
* Assembly Language
* a low-level language that is closer to machine code than high-level languages
* e.g.,
* Machine Code
* the language the CPU directly understands

1. **Compiler vs Interpreter**

* Compiler
* analyzes the entire source code at once and generates an executable file (object code) that can be run directly by the computer
* Translation happens before the program is executed.
* Typically faster execution since the translation is done upfront e.g., C, C++, Java
* **Interpreter:**
* Executes the source code line by line, translating each line into machine code as it's encountered.
* Translation happens during program execution.
* Generally slower than compiled languages due to the on-the-fly translation
* e.g., Python, JavaScript, Ruby

1. **g++ and its Options**

* **g++** is part of the GNU Compiler Collection (GCC) which is a suite of compilers for various programming languages, including C, C++, and Fortran under the GNU project.
* It is a command-line tool used to compile C++ source code into executable programs.
* g++ options
* g++ -c main.cpp: create an object code, main.o
* g++ -S main.cpp: create an assembly source code, main.s
* g++ main.o -o main.exe: create an executable code, main.exe

1. **Essential Linux commands**

* ls: list files and subdirectories(, or folders)
* cp file1 file2: copy file1 to create file2
* rm file1: remove file1
* mkdir sub: make a subdirectory named sub
* rmdir sub: remove the subdirectory named sub

1. **Compile-time vs Run-time Errors**

* Key Differences
* Timing: Compile-time errors are detected before the program runs, while run-time errors occur during execution.
* Causes: Compile-time errors are usually caused by syntax or type errors, while run-time errors can be caused by a variety of factors.
* Compile-time errors
* occur due to the violations of the syntax or semantic rules of the programming language
* are detected before the program is executed
* Run-time errors
* occur during execution, which prevents a running program from continuing
* can be caused by factors such as:
* Input/output errors
* Memory errors: Running out of memory or accessing memory that is not allocated.
* Arithmetic errors: Attempting to perform an invalid mathematical operation (e.g., dividing by zero).
* Logical errors: Incorrect algorithms or logic that lead to unexpected results.

1. **Algorithm and Pseudocode**

* Algorithm
  + a step-by-step procedure to follow in order to perform a task or solve a problem;
  + each step must be clear and unambiguous
* Pseudocode
  + a simplified version of a programming language.
  + It's a way to describe an algorithm in a human-readable format without worrying about the specific syntax of a particular programming language