Vocabulary for Week 1

This is a list of terms that you should know related to the week 1 material.

* Software: The programs and other operating information used by a computer. A set of instructions that directs a computer’s hardware to perform a task is called a program, or software program. Software can be difficult to describe because it is "virtual," or not physical like computer hardware. Instead, software consists of lines of code written by computer programmers that have been compiled into a computer program. Software programs are stored as binary data that is copied to a computer's hard drive, when it is installed.
* Hardware: Incudes the physical parts of a computer including the central processing unit (CPU), data storage, graphics card, motherboard, monitor, etc.
* IDE: (Integrated development environment) A software application that provides facilities for software development. An IDE normally consists of at least a source code editor, build automation tools, and a debugger. Some IDEs provide a runtime environment (RTE) for testing software programs. When a program is run within the RTE, the developer can track each event that takes place within the application being tested. This can be useful for finding and fixing bugs and locating the source of memory leaks. Because IDEs provide a centralized user interface for writing code and testing programs, a programmer can make a quick change, recompile the program, and run the program again. Programming is still hard work, but IDE software helps streamline the development process.

- Source code editor: A text editor that can assist in writing software code with features such as syntax highlighting with visual cues, providing language specific auto-completion, and checking for bugs as code is being written.

- Local build automation: Utilities that automate simple, repeatable tasks as part of creating a local build of the software for use by the developer. For example, compiling computer source code into binary code, packaging binary code, and running automated tests.

- Debugger: A program for testing other programs that can graphically display the location of a bug in the original code.

* CPU: (Central processing unit) The primary component of a computer that processes instructions. It runs the operating system and application, constantly receiving input from the user or active software programs. It processes the data and produces output, which may be stored by an application or displayed on the screen.

At its core, a CPU takes instructions from a program or application and performs a calculation. This process breaks down into three key stages: Fetch, decode, and execute. A CPU fetches the instruction from RAM, decodes what the instruction actually is, and then executes the instruction using relevant parts of the CPU.

The executed instruction, or calculation, can involve basic arithmetic, comparing numbers, performing a function, or moving numbers around in memory. Since everything in a computing device is represented by numbers, you can think of the CPU as a calculator that runs incredibly fast. The resulting workload might start up Windows, display a YouTube video, or calculate compound interest in a spreadsheet.

In modern systems, the CPU acts like the ringmaster at the circus by feeding data to specialized hardware as it is required.

* Main memory: Refers to the physical memory that is internal to the computer. The word “main” is used to distinguish it from external mass storage devices such as disk drive. Other terms used to mean main memory include “RAM” and primary storage. The computer can manipulate only data that is in main memory. Therefore, every program you execute and every file you access must be copied from a storage device into main memory. The amount of main memory on a computer is crucial because it determines how many programs can be executed at one time and how much data can be readily available to a program. Computers often have too little main memory to hold all the data they need. Computer engineers invented a technique called swapping, in which portions of data are copied into main memory as they are needed. Swapping occurs when there is no room in memory for needed data. When one portion of data is copied into memory, an equal-sized portion is copied (swapped) out to make room.
* Storage (and secondary storage): External hard drives, USB flash drives, etc.
* Input: Data that a computer receives.
* Output: Data that a computer sends.
* Application software: Commonly defined as any program or number of programs designed for end-users. It is usually written in high-level or general-purpose languages. Application software needs system software to function.
* System software: A type of computer program that is designed to run a computer’s hardware and application programs. It is written in system programming languages. If we think of the computer system as a layered model, the system software is the interface/liaison between the hardware and user applications. System software is used to manage the computer itself. It runs in the background, maintaining the computer’s basic functions so users can run higher-level application software to perform certain tasks. Essentially, system software provides a platform for application software to be run on top of. The computer’s OS is a well-known example of system software.
* Binary: Binary (or base-2) is a numeric system that only uses two digits- 0 and 1. Computers operate in binary, meaning they store data and perform calculations using only zeros and ones. A single binary digit can only represent True (1) or False (0) in Boolean logic. However, multiple binary digits can be used to represent large numbers and perform complex functions. In fact, any integer can be represented as binary. One bit contains a single binary value – either a 0 or a 1. A byte contains 8 bits, which means is can have 256 (2^8) different values. No matter how large a file or program is, at its most basic level, it is simply a collection of binary digits that can be read by a computer processor.
* Hexadecimal: Describes a base-16 number system which uses 16 symbols to represent a particular value: (0-9) & (A-F). The system most used is the base-10 or decimal system. Most error codes and other values used inside a computer are represented in the hexadecimal format. Programmers use hexadecimal numbers because their values are shorter than they would be displayed in decimal, and much shorter than in binary. By dividing a binary number up into groups of 4 bits, each group or set of 4 digits can now have a possible value of between “0000” (0) and “1111” (8+4+2+1) or ( (2^3) + (2^2)…).

0001 0000 (16) ?

* Unicode: Unicode is a universal character encoding standard. It defined the way individual characters are represented in text files, web pages, and other types of documents. Unicode was designed to support characters (around 1,000,000) from all languages around the world. Unicode supports up to 4 bytes for each character.
* Compiler error: A compile error happens when the compiler reports something wrong with your program and does not produce a machine-language translation. They are relatively easy to fix most of the time (don’t let them bother you). Usually, compiler warnings are an indication that something might go wrong at runtime. A common example is using the assignment operator ('=') instead of the equality operator ('==') inside an if statement. Your compiler may also warn you about using variables that haven't been initialized and other similar mistakes. Generally, you can set the warning level of your compiler--I like to keep it at its highest level so that my compiler warnings don't turn in to bugs in the running program ('runtime bugs'). Compiler errors are restricted to single source code files and are the result of 'syntax errors'. What this really means is that you've done something that the compiler cannot understand.
* Compile-time error: Errors that occur when the rules of writing syntax are known as Compile-time errors. This compile error indicates something that must be fixed before the code can be compiled. These errors are detected at the time of code development. The semantic errors exist when the statements are not meaningful to the compiler.
* Syntax error: This is an error in the coding or programming language entered by a programmer. Syntax errors are caught by a software called a compiler, and the programmer must fix them before the program is compiled and then run. Common syntax errors include: a typing error made by the programmer, or the programmer forgets the format/sequence of some word or command. Syntax errors are different from errors that affect programs during run time. Many logical errors in computer programming do not get caught by the compiler, because although they may cause grievous errors as the program runs, they do conform to the program’s syntax. In other words, the computer cannot tell whether a logical error is going to create problems, but it can tell when code does not conform to the syntax, because the understanding of that syntax is built into the computer’s native intelligence. Computers cannot use input that is not perfectly designed.
* Run-time error: A program error that occurs while the program is running. The term is often used in contrast to other types of program errors, such as syntax and compile-time errors. There are many different types of runtime errors. One example is a logic error, which produces the wrong output. For example, a miscalculation in the source code or a spreadsheet program may produce the wrong result when a user enters a formula into a cell. Another type of runtime error is a memory leak. This type of error causes a program to continually use up more RAM while the program is running. A memory leak may be due to an infinite loop, not deallocating unused memory, or other reasons. A program crash is the most noticeable type of runtime error since the program unexpectedly quits while running. Crashes can be caused by memory leaks or other programming errors. Common examples include dividing by zero, referencing missing files, calling invalid functions, or not handling certain input correctly.
* Logic error: A mistake in a program’s source code that results in incorrect or unexpected behavior. It is a type of runtime error that may simply produce the wrong output or may cause a program to crash while running. Many different types of programming can cause logic errors. For example, assigning a value to the wrong variable may cause a series of unexpected program errors. Multiplying two numbers instead of adding them together may also produce unwanted results. Even small typos that do not produce syntax errors may cause logic errors.
* Debugging: Computer programmers are humans, therefore imperfect. This means the programs they write sometimes have small errors, or “bugs” in them. These bugs can be minor, such as not recognizing user input, or more serious, such as a memory leak that crashes the program. Before releasing their software to the public, programmers “debug” their programs to eliminate as many errors as possible. This debugging process often takes a long time, as fixing some errors may introduce others.
* Variable: A variable is used in programming to store specific values within a program. They are assigned both a data type as well as a value. For example, a variable of the string data type[ may contain a value “sample text” while a variable of the integer data type may contain a value of “11”. Some programming languages require variables to be declared before they can be used, while others allow variables to be created on the fly. The data type, if not defined explicitly, is determined based on the initial value given to the variable.
* Declaration: In programming, a declaration is a statement describing an identifier, such as the name of a variable or a function. Declarations are important because they inform the compiler or interpreter what the identifying word means, and how the identified thing should be used.
* Data type: A data storage format that can contain a specific type or range of values. When computer programs store data in variables, each variable must be assigned a specific data type. Some common data types include integers, floating point numbers, characters, strings, and arrays. They may also be more specific types, such as dates, timestamps, Boolean values, and varchar (variable character) formats. Some programming languages require the programmer to define the data type of a variable before assigning it a value. Other languages can automatically assign a variable’s data type when the initial data is entered into the variable. For example, if the variable “var1” is created with the value “1.25,” the variable would be created as a floating-point data type. If the variable is set to “Hello World!,” the variable would be assigned a string data type. Most programming languages allow each variable to store a single data type. Therefore, if the variable’s data type has already been set to an integer, assigning string data to the variable may cause the data to be converted to an integer format. Data types are also used by database applications. The fields within a data base often require a specific type of data to be input. For example, a company's record for an employee may use a string data type for the employee's first and last name. The employee's date of hire would be stored in a date format, while his or her salary may be stored as an integer. By keeping the data types uniform across multiple records, database applications can easily search, sort, and compare fields in different records.
* Floating point: As the name implies, floating point numbers are numbers that contain floating decimal points. For example, the numbers 5.5, 0.001, and -2,345.6789 are floating point numbers. Numbers that do not have decimal places are called integers. Computers recognize real numbers that contain fractions as floating point numbers. When a calculation includes a floating point number, it is called a "floating point calculation." Older computers used to have a separate floating point unit (FPU) that handled these calculations, but now the FPU is typically built into the computer's CPU.
* Literal: Literal in Java is a synthetic representation of boolean, numeric, character, or string data. It is a means of expressing particular values in the program, such as an integer variable named ‘’/count is assigned an integer value. A constant value assigned to the variable can be referred to as literal.

- When it comes to computers, the difference between 32-bit and a 64-bit is all about processing power. Computers with 32-bit processors are older, slower, and less secure, while a 64-bit processor is newer, faster, and more secure.

-But what do the numbers 32 and 64 even mean?

-Your computer’s central processing unit (CPU) functions like the brain of your computer. It controls all the communication and the flow of data to and from the other parts of your computer. Some computers use two or more processors. However, there are only two main categories of processors now: 32-bit processors and 64-bit processors. The type of processor that your computer uses affects its overall performance and what kind of software it can utilize.

Most computers made in the 1990s to early 2000s have a 32-bit system that can access 2^32 (or 4,294,967,296) bytes (units of digital information) of RAM (random access memory). Meanwhile, a 64-bit processor can handle 2^64 (or 18,446,744,073,709,551,616) bytes of RAM. In other words, a 64-bit processor can process more data than 4 billion 32-bit processors combined.

* In general: add 1 bit, double the number of patterns
* 1 bit - 2 patterns
* 2 bits - 4
* 3 bits - 8
* 4 bits - 16
* 5 bits - 32
* 6 bits - 64
* 7 bits - 128
* 8 bits - 256 - one byte
* Mathematically: n bits yields 2n patterns (2 to the nth power)