



Software Fundamentals

Software is a general term used to describe any set of programs that controls the operation of a computer system. Hardware and software work very closely together, and all computer systems need both hardware and software to be useful.

1. Algorithms

An algorithm is a *precise set of instructions* that describes how to perform a specific task. It describes the **environment** and **sequence/logical flow** of how to go about accomplishing the task. A simple and common example of an algorithm is a recipe. A recipe details what ingredients, equipment (environment) and cooking steps (sequence and logical flow) one needs to follow in order to accomplish the task of preparing a meal.

When writing an algorithm that is aimed at controlling a computer system (rather than making lunch), one needs to first specify the environment: the possible input data, their nature and location or source (what input or storage devices will this data be accessed from), and possibly, the relevant output devices. Also, one needs to specify the sequence and logical flow of the steps required to perform the required processing of this data.

Algorithms are a fundamental first step towards getting computers to do what one wants. Computer hardware is extremely fast, but is not 'intelligent'. In other words, one needs to tell a computer precisely what to do; unlike a human being, it can't figure things out for itself.

However, not only do we need to tell a computer what to do, and how to do it, but we also need to describe it in a language that the computer understands. This is where programs, programming languages and compilers become relevant.

2. Programs and programming languages

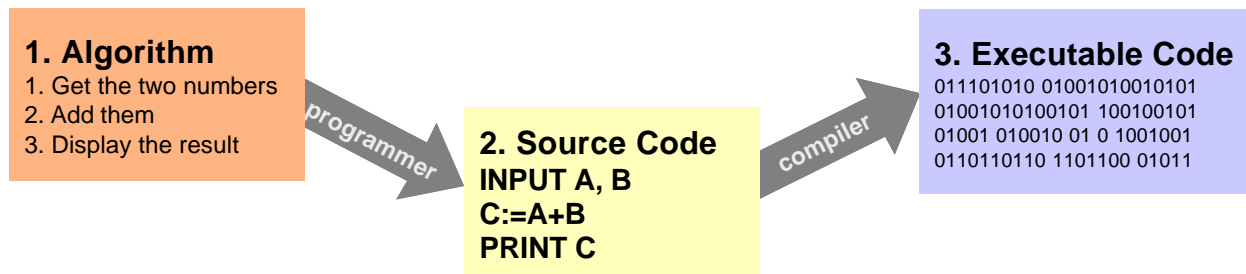
The CPU of a computer understands a fairly limited set of very simple instructions, involving comparisons, arithmetic and data transfer (recall that these are called machine-level instructions). Also, these instructions have to be represented in binary. It is tremendously tedious to have to specify everything in terms of these binary coded machine level instructions. In the early days of computers, this was how people controlled computers; however, they soon started creating higher-level (i.e. more human-language like) languages, specifying their instructions in these languages, and then translating these instructions automatically into binary-coded machine level instructions that the CPU could understand.

In general, a **programming language** is a language that is used to code an algorithm in a way that a computer can eventually understand. Most programming languages are somewhat similar to human language (why?), but are more precise, structured and unambiguous. Just as there are different human languages, there are also a variety of programming languages. Popular programming languages include **Basic, Pascal, C++, Perl** and **Java**.

A **program** is an algorithm or set of algorithms, written using a programming language. Programs are often referred to as **code**, the process of writing programs is commonly called **coding**, and the people who write programs are called **programmers** or **coders**.

In order to convert a program into a set of binary-coded machine level instructions, one needs to **compile** the program. A **compiler** is a special program that accomplishes this task – it converts programming language instructions into machine-level instructions.

Before a program is compiled, the set of instructions, which is written in the programming language, and therefore can potentially be understood and modified by anyone who knows the language, is called **source code**. After the program has been compiled, the resulting set of instructions (which are indecipherable to anyone except the computer) is called **executable code** (since it is in a form that can be directly executed by the CPU). When one loads (transfers from secondary storage) an executable program into primary memory, one is said to be **running** the program. We shall return to these concepts later in the semester, when we discuss the *open source movement*.



3. Operating systems and software packages

An **operating system (OS)** is a collection of programs that coordinates the basic activities of a computer system, including managing computer files, and communicating with the input/output and storage devices. Think of the operating system as the high-level resource manager of the computer.

There are a number of operating systems available to computer users. One of the first operating systems for personal computers was Microsoft **DOS**. Currently, the most popular family of operating systems is the Windows family: **Windows 95**, **Windows 98** and **Windows NT**. Other popular operating systems include the **MacOS** (used on all Apple computers), **Linux**, **Unix**, **Solaris** and **BeOS**.

Application software (sometimes simply called an **application**) is a set of programs written to enable a user or business to use their computer to accomplish a specific class of tasks, or a set of tasks in a specific application area. This is an ambiguous definition, but so is the common usage of the term. Companies sell large sets of application software together as **application software packages**. Examples of software packages include the **Microsoft Excel** spreadsheet and the **Netscape Navigator** web browser.

Typically, consumer software packages fall into one of the following categories: **spreadsheets** (used for business calculations), **web browsers** (what are these used for?), **word processors** (used to create

documents), **presentation software** (used to create business presentations), **databases** (used to organize and retrieve data), and **graphics software** (used to create and modify graphics and artwork).

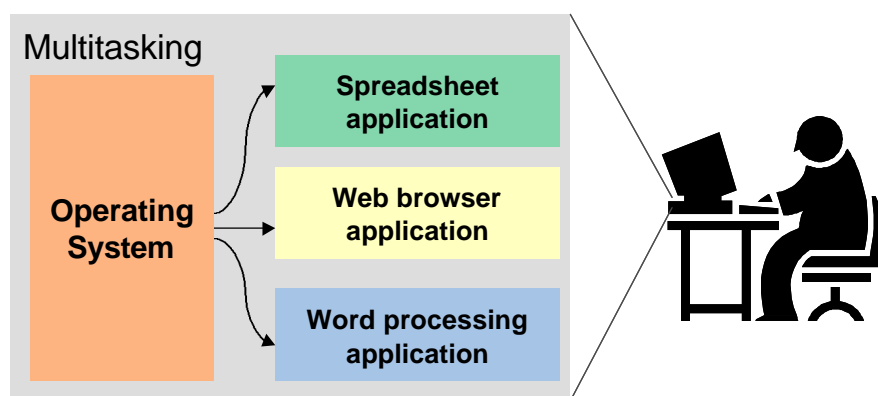
4. Computer Platforms

Every computer system needs a microprocessor and an operating system – a **platform** is a specific combination of a microprocessor family and an operating system. There are two popular platforms that consumers use – the **Wintel** platform (which comprises a Windows operating system and an Intel or Intel compatible microprocessor) and the **Macintosh** platform (which comprises a PowerPC microprocessor and the MacOS). The Wintel platform is the world's dominant platform, and powers over 80% of all personal computers.

Platforms are important because application software is **platform-specific**. In other words, when one writes the set of programs, one typically has to tailor it for a particular platform. Loosely, this is because different operating systems manage resources differently, and different microprocessors have different machine-level instructions. An application for a particular platform is said to be **compatible** with that platform. Often, it is very time-consuming to make an application for one platform compatible with a different platform. Hence, a lot of software writers tend to target the most popular platform, which is why there is so much software available for Wintel computers.

5. Other concepts of interest

Multitasking: An operating system is said to **multitask** when it lets a computer run multiple applications simultaneously, even when the computer has only one CPU. The operating system cycles rapidly between the different applications, allocating different segments of the CPU's time to each application. This happens in a way so that we don't visually notice the actual cycling – however, this is why one finds that one's computer is a lot slower when there are lots of applications running.



Graphical user interfaces (GUIs): A **GUI** is a set of programs, sometimes part of an operating system, that lets a user issue commands to their computer by using icons and 'pointing-and-clicking', rather than by typing in commands. Older operating systems, like DOS and Unix, do not have built-in GUI's, (such operating systems are called **command-driven**), and therefore tend to require a lot more learning before

one can use them. To overcome this problem, one can get a GUI for these operating systems, and run it **on top** of the OS. The initial versions of Windows were simply GUI's for DOS; other OS's like Linux and Unix are still command-driven, and have a host of GUIs available for them.

Interpreters: Some programming languages are designed so that there is never any executable code created. Instead, every time the program is run (i.e. at **run-time**), the instructions in the programming language are converted sequentially into machine-level instructions one by one, and then executed by the CPU. The software that performs the line-by-line translation at run-time is called an **interpreter**. Examples of languages that use interpreters are **Basic** and **Perl**.

Java and platform independence: In 1995, Sun Microsystems unveiled a new programming language called **Java**, which was **platform independent**. In other words, programs written in Java could be run on any platform. Java is popularly known today because Web browsers can run small Java programs, called **applets** (small applications), which most of us have probably seen. It is gaining popularity as a conventional programming language for traditional applications as well. However, some people are disappointed with Java's platform independence, since the speed and performance of Java code varies a lot across platforms.