Fundamentals of Programming

Chapter One

Introduction to Programming

Unit Introduction

These days, computers are indispensable. Everyone utilizes them to communicate, have fun, and solve challenging work-related problems and are used in various fields, including business, entertainment, telecommunications, and banking. It is hardly an exaggeration to argue that computers form the neurological network of our modern civilization, and it is hard to imagine life without them (Iverson, 1962).

Despite their widespread use, few individuals are familiar with how computers operate. The software that runs on the computers—not the computers themselves—matters. Computers are valuable to users because the software enables a wide range of services that improve our quality of life (Wirth, 1971).

Professionals use programming as a fundamental tool to design and build complicated software systems, automate tedious operations, and address challenging issues. In our digital age, a necessary ability is becoming increasingly significant. Numerous industries, like healthcare, finance, & entertainment, have been transformed through programming. Therefore, to become a successful developer or programmer, it is imperative to possess a firm understanding of the foundations of programming (Rosen, 1972).

Did you know?

Ada Lovelace created the initial computer program in the middle of the nineteenth century.

Learning Objectives

Students are going to able to comprehend the following after reading this chapter:

1. The fundamental idea of programming.
2. Knowledge of information processing by computers.
3. Computer programming history.
4. Aspects of several programming languages.
5. Software development lifecycle.

Key Terms

1. Algorithms
2. BASIC
3. Computers
4. C++
5. COBOL
6. FORTRAN
7. Python
8. Programming
9. Java
10. Script
11. Software
12. Testing

1.1. Processing of Information

To comprehend what it means to program, one can generally relate a computer or its operating system to a vast factory, including all its workshops, warehouses, and transportation. Visualizing the degree of complexity present in a modern computer using this basic comparison is simpler. A computer runs several of that correspond to a factory's workshops and production lines. The operational memory (RAM), the hard drive, and its contention s serve warehouses, while the various protocols serve as transportation systems for information input and output (Aho et al., 1979).

Different workshops produce the many goods that are produced at a factory. They use the warehouses' raw materials and keep their finished products there. The suppliers bring the finished goods from their warehouses to the outlets, while the warehouses deliver the raw materials to the warehouses. Different modes of transportation are employed to achieve this. Raw materials entering the plant undergo many processes and eventually emerge as finished goods. Each factory transforms unprocessed supplies into a finished good that may be consumed (Hosoya & Pierce, 2003).

A computer is a tool for processing information. In contrast to a manufacturing facility in our instance, information serves as the computer's raw material and final product. Most of the time, the input data is taken from one of the storage areas (files or RAM), where it was previously moved. It then goes through one or more procedures and transforms into a new product. A good example is web-based applications. Information processing often entails extracting content from databases &

processing it for display in HTML. They employ HTTP for transferring raw materials and finished goods (Shaikhha et al., 2019).

1.2. Managing the Computer

Numerous levels of management are involved in the entire process of producing goods in a factory. Operators run the machinery and assembly lines, managers oversee the workshops, and general executives oversee the entire plant. They all exert control over various processes at various tiers. The lowest level of service providers are the machine operators, who use buttons and levers to operate the machines. The workshop managers will advance to the levels carried out, watching commands (Holsapple, 2005).

Computers and software have numerous levels of control and management, much like this. The simplest level is controlled by the processor & its registers (this is done by utilizing low-level machine programs); we can relate it to managing the equipment in the workshops. Are all under greater control of the operating system (Windows 7, for instance) – It can be compared to how many workshops and departments function in manufacturing. The application software is at the top of the hierarchy. It manages a large number of tasks that use a lot of processor activities. General executives operate the entire production at this level to optimize resource usage and deliver superior results (Hosoya & Pierce, 2003).

1.3. The Essence of Programming

The goal of a program is to manage every aspect of a computer's operation. The "orders" and "commands" provided by the programmer, referred to as programming instructions, are used to do this. To "program" is to arrange computer operations through a series of instructions. The computer (namely, the OS, the CPU, and peripheral devices) implicitly complies with these written orders (instructions). Writing a "program" refers to organizing a set of instructions to get a computer to do something. "computer programs" or "scripts" describes these sets of instructions (Wadler, 1992).

An algorithm is a series of processes to accomplish a goal, finish a task, or produce a specific outcome. This is why programming and algorithms relate to one another. Programming entails putting the desired action on the computer into a step-by-step or algorithmic description.

Did you know?

IBM created FORTRAN, the initial high-level programming language, in the 1950s.

Computers are controlled by these instructions, which programmers make. These directives are referred to as programs. Many programs were written in a variety of programming languages. Each

language targets different levels of computer control. There are languages designed for the lowest (machine) level of computation – For instance, an assembler. Others, like C, are most beneficial when used at the system level (when interfacing with the operating system). High-level languages are also employed in the development of application programs. Some examples of these languages are Java, Perl, C#, Visual Basic, Python, Ruby, PHP, C++, & JavaScript (Jones, 2005).

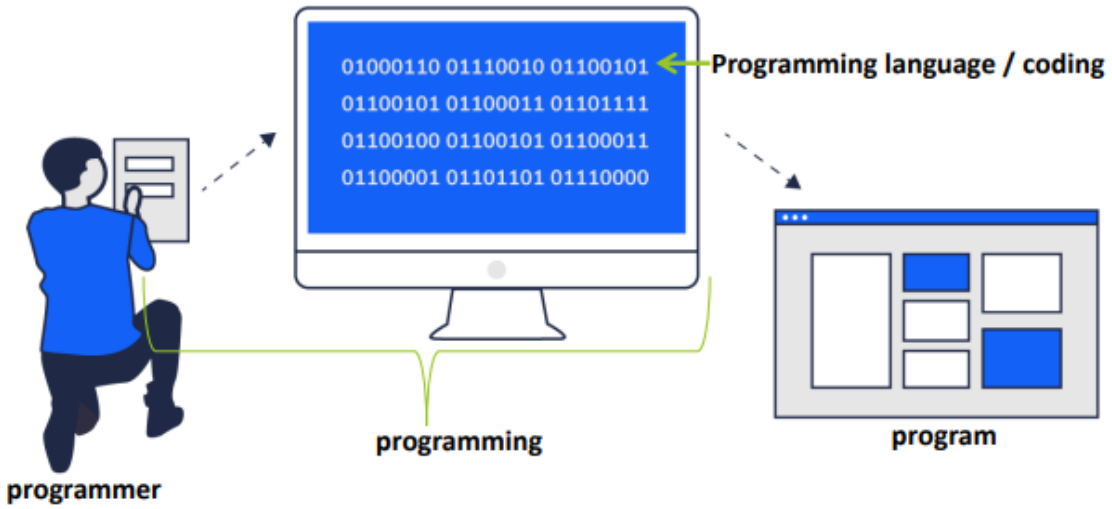

Figure 1.1. Illustration of relation between programmer, programming, and program (Source: Wan Fazlini, Creative Commons License).

1.4. History of Programming

The earliest mechanical computers were developed in the early 1800s when programming began. These machines weren't programmable in the traditional sense; but were capable of rudimentary arithmetic calculations.

Ada Lovelace, a mathematician, wrote the very first computer program in the middle of the nineteenth century. It was intended to operate on Charles Babbage's Analytical Engine. Although Lovelace's program was never used, her work set the stage for subsequent programming projects (Bergin, 2007). Many significant computing tools, such as the Hollerith tabulating device, were created in the late 1800s and early 1900s and utilized for the 1890 US census. The Hollerith machine employed punch cards to represent data so that a machine could read it and automatically tally it. The idea of a universal machine that could be programmed to carry out any computation which could be defined algorithmically was created by mathematician Alan Turing in the 1930s. Turing's work was essential to advancing contemporary computing and established the groundwork for the 1940s that produced the first electronic computers (Sammet, 1972).

The circuitry in the initial electronic computers was made of vacuum tubes, which were big, expensive machines largely employed for scientific and military purposes, such as deciphering codes throughout World War II and calculating the yield of an atomic weapon.

Programming languages were created in the 1950s and 1960s to facilitate software creation for these systems. To enable scientists and businesspeople to construct programs for scientific & commercial applications, languages like FORTRAN, COBOL, & BASIC were developed (Liskov, 1996).

Several programming languages were created in the 1970s and 1980s that emphasized various facets of programming. Systems programming was intended for C and Pascal, whereas advanced applications were intended for languages including Smalltalk and Lisp.

New programming languages, as well as frameworks, were developed in the 1990s and 2000s as a result of the growth of the internet and personal computing. For online development, languages like Java, Python, and Ruby gained popularity, while the proliferation of mobile devices sparked the creation of emerging mobile programming languages such as Swift and Kotlin (Liskov, 1996).

Today, software powers anything from smartphones to automobiles to medical gadgets, making programming an essential part of contemporary civilization. New languages, frameworks, and tools are being developed to suit the increasing needs of modern computing as the area continues to advance quickly.

1.5. Types of Programming Languages

Programming languages may be generally categorized based on their traits and features. The following is a description of a few of the most popular types of programming languages (Ancona et al., 2016):

1.5.1. High-Level Programming Languages

These languages have been created to be simple enough for anybody to read and understand. In contrast to low-level programming languages, they frequently used English-like syntax & had a greater level of abstraction (Barnett & Greenes, 1970).

1.5.2. Low-Level Programming Languages

These languages provide a higher level of management of system resources and are intended to be more directly related to the technology of a system of computers. Low-level languages include assembly languages (Baillie, 2005).

1.5.3. Procedural Programming Languages

These languages are built around procedures and subroutines, standalone coding units that carry out particular functions. The most common procedural programming languages are C and Pascal.

1.5.4. Object-Oriented Programming Languages

The foundation of these languages is the idea of objects of desire, data, and behavior. Examples include Java, Python, and C++ (Jordan et al., 2015).

1.5.5. Functional Programming Languages

Because functions are considered first-class citizens in these languages, they can be supplied like arguments, retrieved as values, and kept in variables. Functional programming languages include Haskell, Lisp, and Clojure (Hudak, 1989).

1.5.6. Scripting Languages

These languages are meant for quick and simple task automation and prototyping. They frequently have a dynamic type and trash collection and tend to be interpreted instead of compiled. Examples include JavaScript, Python, and Ruby (Xie & Aiken, 2006).

1.5.7. Markup Languages

Instead of specifying the functionality of software applications, these languages are employed to define the structure and display of texts. Markup languages include things like HTML, XML, and LaTeX (Vilhjálmsson et al., 2007).

1.5.8. Query Languages

These coding languages are employed to retrieve and modify data from databases. The most popular query language is SQL (Angles et al., 2017).

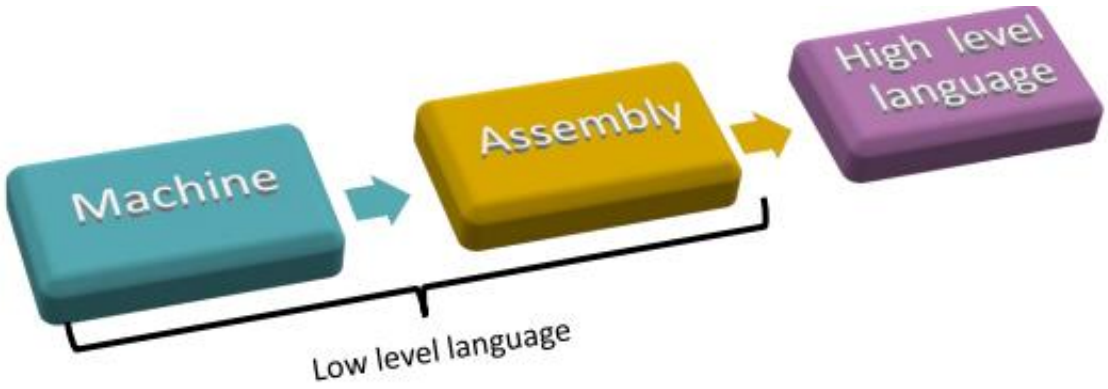

Figure 1.2. Image of different programming languages (Source: Wan Fazlini, Creative Commons License).

Did you know?

Python, a computer language, was named after Monty Python's Flying Circus, a prominent 1970s British sketch comedy program.

1.6. Stages in Software Development

Software development can be extremely difficult and time consuming, requiring an extensive group of software engineers and additional experts. As a result, various techniques and procedures that simplify the work of programmers have been developed. The only thing that has similarities is that each software product passes through several stages during its development (Cysneiros & Kushniruk, 2003):

i. assembling the task's requirements for the product

ii. planning and putting together the architecture and design

iii. implementation (which involves developing program code)

iv. Product evaluations (testing)

v. Deployment and use; and

vi. Support

Most implementation, testing, deployment, and support tasks utilize programming (Smith & Woodside, 1999).

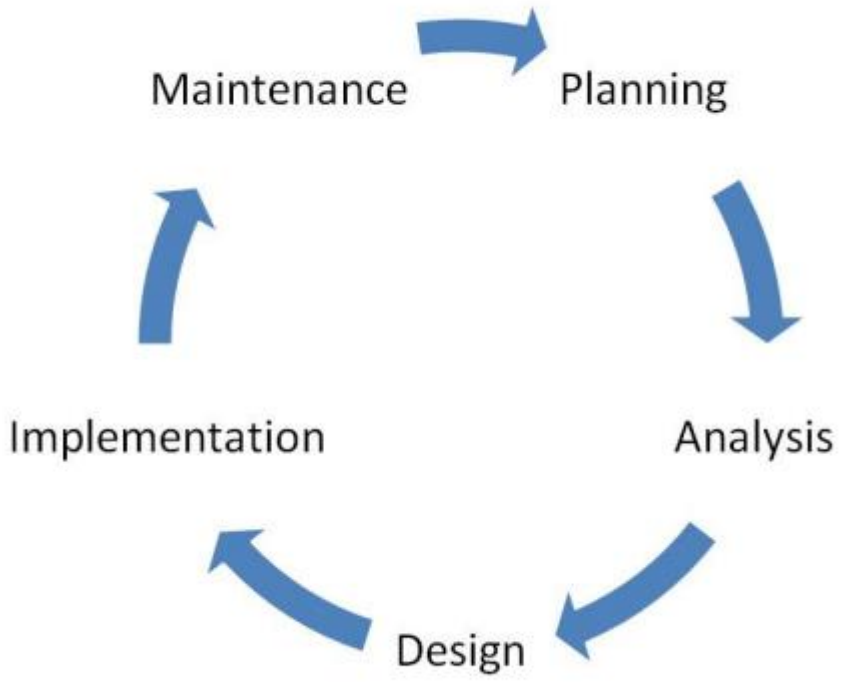

Figure 1.3. Representation of software development lifecycle (Source: Kenneth Leroy, Creative Commons License).

1.6.1. Gathering the Requirements

Only the concept of a particular product is present at first. It provides a set of specifications that outline what both the user & the machine must do. In most cases, these actions simplify previously simple tasks; examples include calculating ballistic trajectories, generating salaries, and using Google Maps to find the shortest route. In many instances, the software adds functionality that wasn't there before, such as automating a certain task (Ruparelia, 2010).

The specifications for the good are typically given in writing, whether in English or another language. At this time, no programming has been done. Experts familiar with the issues in a particular field define the requirements. Additionally, may write them up so that coders can easily grasp them. These experts are typically called business analysts and not programming specialists (Saeed et al., 2019).

1.6.2. Planning and Preparing the Architecture and Design

The planning stage is the next step after gathering all the information needed. A technical implementation strategy for the project is now developed, outlining the program's platforms, technologies, and first architectural (design). Software engineers with a lot of expertise do this step,

which involves a lot of creative labor. Software architects are another name for them. The specifications select the following components (Zachman, 1987):

The kind of application, such as console applications, desktop GUI applications, client-server applications, web applications, rich internet applications (RIA), mobile applications, peer-to-peer applications, or other applications;

The software's architecture, such as its SOA architecture, multi-layer, single-layer, double-layer, and, or triple-layer architecture (Maranzano et al., 2005);

The programming language, such as C#, Ruby, Python, PHP, Java, JavaScript, or C++, is most appropriate for the implementation;

The technologies which will be used include the following: the platform (Microsoft.NET, Java EE, LAMP, or another), the database servers (Oracle, SQL Server, MySQL database, NoSQL database, or another), the user experience (Silverlight, Flash, Windows, JavaServer Faces, and the Eclipse program RCP, the ASP.NET Forms, WPF, or another), the data access (for example, Hibernate, JPA, or ADO.NET Entity Framework), the reporting (Zachman, 1999).

The development frameworks, such as ASP.NET MVC (for.NET), Knockout.js (such as JavaScript), Rails (such as Ruby), Django (such as Python), and many others, will make development easier.

The size and expertise of a development team (sizable, seasoned teams of developers complete huge, important projects) (Brancheau et al., 1989);

The development plan divides functionality into phases and specifies resources and completion dates for each phase.

Other factors (team size, location, communication techniques, etc.) (Sowa & Zachman, 1992).

Although many principles make accurate analysis and planning easier, this stage still calls for a good bit of intuition and insight. This step already predetermines the next phase in the development process. At this point, just preparation is being done instead of programming.

1.6.3. Implementation

The phase that has the strongest ties to programming is the implementation phase. At this stage, a program (application) is carried out (written) by the specified task, design, and architecture. By creating the source code for the program, programmers take part. When building a small project, the other stages can be cut short or eliminated, but the implementation must always be included; otherwise, the procedure is not software development. This book describes the implementation skills needed to construct software programs, including developing a programmer's mindset and

understanding how to use all the tools offered through the C# language & the.NET platform (Wulf, 1980).

1.6.4. Product Testing

Testing products is a crucial phase in the development of software. Its goal is to guarantee that all regulations are carefully followed. Although manual implementation of this process is an option, automated testing is preferred. Small programs are used in these studies to automate them as much as feasible. To check the accuracy of the code, product testing contains both automated and human procedures because some functionality is very difficult to automate (Stefik & Hanenberg, 2014).

QAs (Quality control engineers) implement the testing (trials) procedure. They collaborate closely with programmers to identify and fix software faults. Finding bugs in the code is a top priority, and hardly any new code is developed.

The testing phase typically uncovers many flaws and faults, returning the program to the implantation phase. Before a software product meets both requirements, thus is prepared for deployment. Therefore, the usage phases of the two preceding stages are closely related, and it is typical for any software product to flip between them repeatedly (Hailpern & Santhanam, 2002).

1.6.5. Deployment and Operation

The act of putting a certain software product into use is known as deployment. This phase may be the most time-consuming and expensive if the good is sophisticated and serves a large population. This is a quick and simple technique for smaller programs. The most typical scenario involves the creation of an installer—a specialized application. It guarantees the product's rapid and simple installation (Kontogiannis, 2008). Additional software for support is created specifically for the deployment if the product will be used in a big business with thousands of copies. The product is prepared for use when the installation is successfully finished. Training staff members to use it in the following stage (Fischer et al., 2012).

Deployment of the latest edition of Microsoft Windows within the state government would serve as an illustration. This includes setting up and configuring the program and instructing staff members on utilizing it.

Typically, the software development team or certified deployment experts perform the deployment. Systems administrators, database administrators (DBAs), system engineers, specialized consultants, and other professionals can be among them. Nearly no new code is written, but the current code is tweaked and configured until it satisfies all the necessary conditions for a successful deployment (Dearle, 2007).

Did you know?

Grace Hopper, an information technology scientist, develop the term "bug" to characterize a computer error after discovering a moth lodged in a single of her computer circuits.

1.6.6. Technical Support

Inevitably, issues may arise during the exploitation process. They may be brought on by various things, such as software bugs, improper usage, or poor configuration, but most issues arise when users alter their requirements. These issues prevent the program from accomplishing the business task for which it was designed. The professionals in support and development must be more involved in this. No matter how amazing a software product is, the support process typically lasts the entire duration of its existence (Chakraborty et al., 2021).

The development team and specially trained support specialists provide the support. Numerous individuals may participate in the process based on the modifications made - administrators, executives, architects, programmers, quality assurance engineers, etc.

When the tax legislation that affects the serviced accounting procedure changes, for instance, a software program that calculates salaries will need to be updated; if the end user's hardware is updated, for instance, the software will need to be set up and configured once more, necessitating the assistance of the support staff (Kert, 2011).

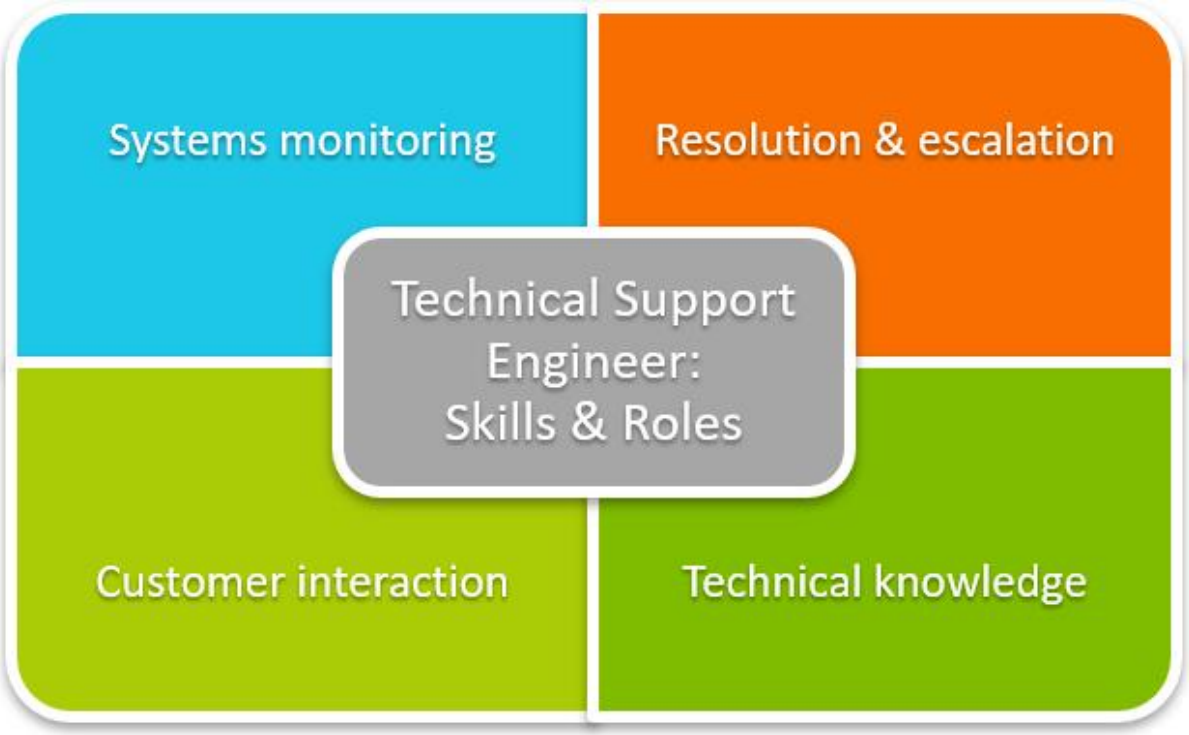

Figure 1.4. Illustration of skills and roles of technical support engineer (Source: Chrissy Kid, Creative Commons License).

1.6.7. Documentation

The documentation stage runs concurrently with all the previous stages; it is not a distinct step. The goal of documentation, a crucial component of software development, is to transfer information among the various parties involved in creating and maintaining a software product. Both inside and outside of a single stage, information is transmitted. The people who develop programs (architects, programmers, QA engineers, and others) typically write the development documentation, a collection of documents (Yngve & Sammet, 1963).