Basics of Programming

Programming Basics (DG Junior, 2023)

A **computer program** is a sequential set of instructions, known as **codes**, written or "*programmed*" in a computer language to let the computer perform a specific computation task. Examples include system software, web browsers, utility software, multimedia software, and spreadsheet software.

It is important to note that the set of instructions in a computer program must be performed sequentially unless directed otherwise. The instructions in the set will express a unit of work that a computer language can support. A **computer language** is a set of grammatical rules that commands a computer or a device to behave in a specific way. It also refers to **programming language**.

Programming

It is the art and science of instructing computers to perform tasks using a specific programming language. It creates codes that tell a computer how to solve problems and accomplish various operations. By writing code in programming languages, **programmers** can develop computer programs or software applications, design websites, analyze data, and automate processes, among other operations**.**

As programming becomes a sought-after skill in various industries, proficiency in programming opens up myriad career opportunities, from software development and web development to data science and artificial intelligence. Programming also enhances critical thinking, logical reasoning, and problem-solving abilities that help programmers break down complex tasks into manageable steps and develop efficient algorithms.

Evolution and History of Programming

Programming involved writing machine code initially. This consisted of binary instructions (0 and 1) directly communicating with the computer's hardware. As technology progressed, high-level programming languages developed to simplify the coding process. These languages allowed programmers to write code using more human-readable syntax.

Currently, a wide ecosystem of programming languages and tools caters to different programming paradigms and application domains.

Programming Paradigms

The term "*paradigm*" is synonymous with "*pattern*." These refer to the different approaches to structuring and organizing code. These provide a conceptual framework for solving problems and designing software. Here are some commonly used programming paradigms:

- **Procedural Programming** – or **imperative programming**, focuses on organizing code into reusable procedures or functions, emphasizing the sequence of steps to execute a program. Examples include BASIC, C and C++, Pascal, and Java.
- **Object-oriented programming (OOP)** – revolves around the concepts of objects that encapsulate data and behavior that promotes modularity, code reusability, and scalability. Examples include Python, VB.NET, and C#.
- **Functional Programming -** treats computation as evaluating mathematical functions, emphasizing consistency, and avoiding side effects.

The nature of the project, requirements, and the desired programming style are some of the basis for choosing the appropriate programming paradigm.

Programming Languages

Numerous programming languages are available, each with its unique syntax, features, areas of specialization, and community support, making it suitable for specific projects. This can be classified into two (2) Low-level and high-level languages.

- **Low-level Language** – a programming language closer to machine code and hardware in terms of syntax. It provides direct control over the computer's hardware and resources, allowing programmers to write code at a more detailed level. It is commonly used for tasks requiring precise control and efficient execution. Examples of this language include the following:
    - **Assembly Language** – uses specific instructions to control a computer's hardware.
    - **Machine Language** – all instructions are written as binary numbers (1 and 0).

- **High-level Language** – a programming language designed to be easy for humans to read, write, and understand, allowing programmers to write computer programs and interact with a computer system without needing specific knowledge of the processor or hardware that the program will run on. Examples include C++, Pascal, PHP, Python, and Java.

Choosing a programming language depends on different factors such as project requirements, performance needs, community support, and available libraries or frameworks.

Other Programming Terminologies

Here are some terminologies that can be useful to understand programming further.

Syntax

It is a set of rules defining the various combinations and arrangements of symbols or characters to create a valid statement in a language.

Command

It is the unique instruction given to a computer application to perform a task of a function such as "*print*" to display text on the screen.

Integrated Development Environment (IDE)

It is a software application for formatting the code, checking the syntax, and running and testing the code. IDEs can work with multiple programming languages, while some are specific to only one language.

Library

It is a collection of useful resources, such as objects and functions, that can be used individually and must be configured to work together. It is pre-built or installed in an IDE or Integrated Development Environment.

Interpreter

It is a program that directly executes instructions in a high-level language without converting them into a machine language.

Assembler

It is a program that converts instructions written in low-level assembly code into relocatable machine language.

Compiler

It is a program that converts high-level languages into machine-readable code that a computer can execute.

Algorithm, Pseudocode, and Flowchart (Chaudhuri, 2020)

In programming, an **algorithm** is a set of steps that generates a finite sequence of simple computational operations leading to the solution of a given problem. It must be expressed in a natural language that anyone can follow, such as directions that can be written in English.

Example:
Design an algorithm that finds and displays the volume of a rectangle. It is required to know the rectangle's length, width, and height, and the formula for finding its volume. The formula is volume = length × width × height.

The algorithm to find and display the volume of the rectangle is as follows:

1. Get the length of the rectangle
2. Get the width of the rectangle
3. Get the height of the rectangle
4. Find the volume using the formula: volume = length × width × heights
5. Display the computed volume

The algorithm is also part of the problem-solving process in the programming environment. The complete steps include:

1. **Problem Analysis** – evaluating and outlining the problem and its solution requirements
2. **Algorithm Design** – designing an algorithm to solve problems.
3. **Coding** – implementing the algorithm in a programming language.
4. **Execution** – verifying whether the algorithm works or not.

The two (2) commonly used tools in representing algorithms are through a pseudocode or a flowchart.

Pseudocode

It is a technique used to describe distinct steps of an algorithm that is much easier to understand for anyone with basic programming knowledge.

Here are some rules that are frequently followed when writing pseudocode:

- Symbols that are used for common operations:
  - Arithmetic operations (+, -, *, /)
  - Assignment (=)
  - Comparison (=, ≠, <, >, ≤, ≥)
  - Logical (and, or)
- Keywords can be used as a command, such as PRINT, WRITE, READ, SET, and GO TO.
- Indentation is used to indicate branches and loops of instructions.

Example:
Using the same example above:

The pseudocode to find and display the volume of the rectangle is:

```
READ length
READ width
READ height
SET volume to 0
COMPUTE volume as length * width * height
PRINT volume
```

Flowchart

It is a diagrammatic representation of the steps of an algorithm. It is a pictorial representation that can be used as a substitute for an algorithm, as a textual description of an algorithm may not be understood easily.

Flowcharts are classified into two (2) categories: Program flowcharts and System flowcharts.

- **Program flowcharts** – illustrate the logical steps in a software program or programming task to understand a process, workflow, or algorithm. They contain the steps of solving a problem unit for a specific result.
- **System flowcharts** – show how parts of a system work together by displaying the data flow and how decisions can affect the events surrounding it. ss

Boxes of different shapes are used to represent various types of operations. Lines then connect these boxes with arrows representing the direction or flow to which one should proceed to know the next step.

Here are the standard symbols used in program flowcharts.

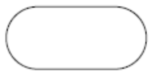
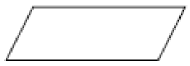
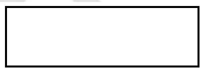
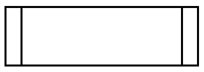
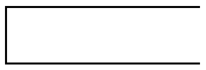
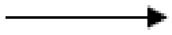
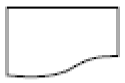
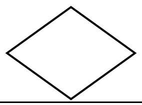
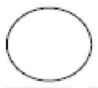
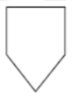
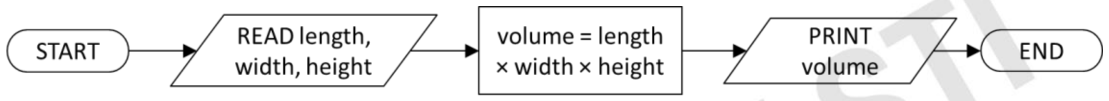
Symbol	Name	Description
	Terminal	Shows the start and end of a set of computer-related processes
	Input/Output	Shows any input/output operation
	Computer Processing	Shows any processing performed by a computer system
	Predefined Processing	Indicates any process not specially defined in the flowchart
	Comment	Used for writing any explanatory statement required to clarify something
	Flow line	Used for connecting the symbols
	Document Input/Output	Used when input comes from a document, and output goes to a document
	Decision	Shows any point in the process wherein a decision must be made to determine further action
	On-page Connector	Connects parts of a flowchart continued on the same page
	Off-page Connector	Connects parts of a flowchart continued to separate pages

Table 1. Flowchart symbols. Retrieved from Chaudhuri, A. (2020). Flowchart and algorithm basics: The art of programming. Mercury Learning and Information.

Example:
Using the example in Algorithm:

The flowchart to find and display the volume of the rectangle is:



Five (5) Rules for Creating Program Flowcharts
The following rules must be observed while creating program flowcharts.

- Only the standard symbols should be used in program flowcharts.
- The program logic should only show the flow from top to bottom and/or left to right.
- Each symbol should contain only one entry point and one exit point, except the decision symbol. It is known as the ***single rule.***
- The operations shown within a symbol should be expressed independently of any programming language.
- All decision branches should be well-labeled.

References:
Chaudhuri, A. (2020). *Flowchart and algorithm basics: The art of programming.* Mercury Learning and Information
DG Junior (2023). *Basics of programming: A comprehensive guide for beginners.* DG Junior.