Program language generations and programming paradigms

Edited by Péter Vass

Some basic concepts and definitions

(Computer) program

- is a collection of suitably ordered instructions,
- the instructions are written (or coded) in some programming language,
- is executable by a computer to perform a specific task.

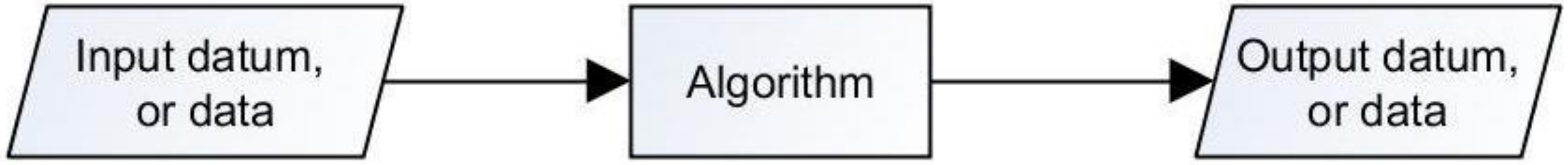
The logical content of a program is based on one or more algorithms.

Algorithm

- is a logical part of a computer program which specifies the solution of a well-defined task,
- is a self-contained step-by-step set of operations.

Some basic concepts and definitions

Each algorithm has its own *input datum* or *data*.
It processes the input by means of its operations and provides the *output datum* or *data* as a result.



So that a computer can execute an algorithm, we must compose it into a program by using a programming language.
The coded form of one or more related algorithms is a program.
The steps of a program are called *instructions*.

Some basic concepts and definitions

A **Programming language**
is a *formal language* which specifies a set of instructions and the rules of their application. By the correct use and combination of the instructions we can create our programs.

A lot of programming languages have been developed since the beginning of computer science and the development has not finished at all.
Actually, a continuous development is going on in the world of programming languages.
Some of them can be considered as general (*general-purpose languages*) while others are closely connected to some field of science or technology (*domain-specific languages*).

Some basic concepts and definitions

<u>Examples for general-purpose languages</u>:
C, Java, Python,

<u>Examples for domain-specific languages:</u>
MATLAB and GNU Octave for matrix programming,
Mathematica, Maple and Maxima for symbolic mathematics,
SQL for relational database queries.

Since the development is fast both in hardware and software some programming languages lost in importance years after (e.g. Fortran, Algol, Basic).
Programming languages may be classified into programming language generations.
This classification tries to mirror the development in programming styles and languages.

Programming language generation

First generation languages (1GL):

Numerical machine code languages are regarded as the lowest-level programming languages (lowest abstraction level).

The instructions are actually binary numbers which can be expressed in octal or hexadecimal format to reduce the length of instructions in a program.

These programming languages are hardware-dependent which means that the instruction set is specific to a class of processors using the same hardware architecture.

For example, the machine code 00000101 causes the CPU to decrement the numeric value actually held by B processor register in the case of a Zilog Z80 processor. (The registers of a processor are denoted by capital letters. A register is a data holding place inside a processor).

Programming language generation

First generation languages (1GL):

Because of the hardware-dependency, a programmer has to know the details of the hardware components to write a program in a machine code.

After loading the program into the memory, it is directly executed by a computer's central processing unit (CPU).

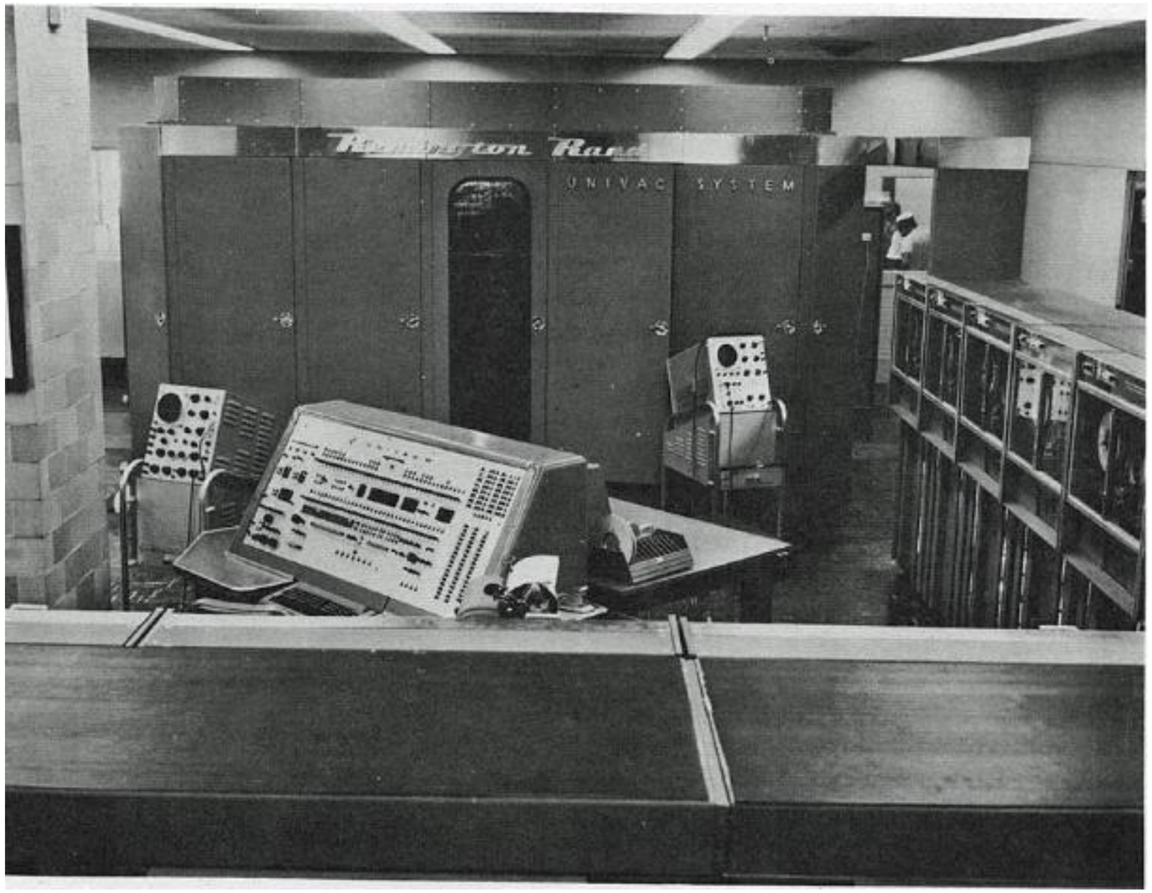
There is no need to apply any utility program which translates or interprets the code.

1GL languages were fundamentally used in the 1940-s an 1950-s. Some of them were applied later in programming industrial microcontrollers.

Programming language generation

UNIVAC (UNIVersal Automatic Computer)

the first commercial computer (1951 USA)



https://en.wikipedia.org/wiki/UNIVAC

Programming language generation

Second generation languages (2GL):

The so-called *assembly languages* use very simple symbolic names (mnemonics) to refer to machine code instructions instead of numeric values.

For example, DEC B instruction in an assembly language corresponds to the machine code 00000101on the Zilog Z80 processor.

They are also hardware-dependent (specific to a particular computer architecture).

They are low-level programming languages, but the abstraction level is higher than that of machine code languages.

It means that an assembly program is more readable, and programming in an assembly language is easier and faster than in a machine code language.

Programming language generation

Second generation languages (2GL):

An assembly language program is shorter than its correspondent machine code program.

Unlike a machine code program, an assembly language program cannot be executed directly by the CPU.

A special utility program called *assembler* is used for converting the source code into a machine code program.

They were mostly used in the 1950-s an 1960-s.

Until recently, assembly languages were used for writing low level, simple but efficient programs for specific hardware such as industrial microcontrollers or mobile phones.

Because of the luck of higher abstraction level statements (loops and conditional branches), writing a larger and more complex program is slow and hard.

This is the main reason why their use is declining.

Programming language generation

PDP-11 minicomputers (1970) manufactured by Digital Equipment Corporation (DEC)

https://www.theregister.co.uk/2013/06/19/nuke_plants_to_keep_pdp11_until_2050/

Programming language generation

Third generation languages (3GL):

They are also called *high level languages*.

The abstraction level of these languages was raised by introducing several new and useful elements in programming.

Their most important features are the application of

- different data types and structures (e.g. integer, float, double, arrays),
- control structures (conditional branches, loops),
- several operators (arithmetic, relational and logical operators).

Due to these innovations the program development became significantly easier and faster.

Keywords, numbers and other symbols are used for building the so-called source code of a program which is easily understandable (compared to an assembly code).

Programming language generation

Third generation languages (3GL):

Contrary to machine code and assembly languages, high level languages are not hardware-dependent.

The source code, however, cannot be executed directly by a computer.

Two different ways of processing source codes are used in practice.

The one is converting the whole source code into a machine code by means of a utility program called *compiler*. This process is called *compilation*.

The compiled code is stored in an executable file. This file can already be executed directly at any time by the computer.

Programming language generation

Third generation languages (3GL):

The other way is converting and executing a source code line by line by means of a utility program called *interpreter.*

The interpreter provides a runtime environment for source codes. But the interpretation process does not create an executable file for the program.

Therefore, the interpretation process will be repeated whenever a program must be run.

Examples for compiled languages: Fortran, Basic, Turbo Pascal, C, C++

Example for interpreted languages: Java, Perl, Ruby

High level languages have been widely used in programming since the second half of the 1960-s.

Programming language generation

Apple II home computer (1977)

https://en.wikipedia.org/wiki/Apple_II_series

Programming language generation

<u>Fourth-generation languages (4GL)</u>

The main purpose of designing newer languages with higher abstraction level was facilitating and accelerating the process of software development.

4GL languages are also known as *very high level languages* because a single 4GL instruction may replace several 3GL instructions, so the source codes are shorter and more similar to a text written in a natural language (generally in English) than those of 3GL languages.

Most of them are aimed at solving specific problems (domain-specific languages).

Programming language generation

<u>Fourth-generation languages (4GL)</u>

For example:

SQL is a language to query databases,

Oracle Reports is a language to make reports,

XUL is a language to construct graphical user interfaces

MATLAB is a language to solve scientific and engineering problems.

4GL languages allow also the users who are not computer professionals to develop programs and software.

The first 4GL languages were developed in the 1970s and this generation is flourishing in our days.

Programming language generation

<u>Fourth-generation languages (4GL)</u>

http://dewytech.in/matlab-training/

Programming language generation

Fifth Generation languages (5GL)

The development of these languages was closely connected to the artificial intelligence research of the 1980s (e.g. PROLOG, Mercury, LISP).

The instructions used in 5GL languages are highly sophisticated.
It means that dozens or hundreds of 4GL instructions may be replaced by a single 5GL instruction.

The most modern 5GL languages use visual or graphical development interfaces which enable the programmer to create programs without typing any source code.

Programming language generation

Fifth Generation languages (5GL)

The components of a program, the connections of the components, the operations and the control flow of the operations can be represented by graphical objects.

And the suitable arrangement of the required elements forms the program.

The development environment converts the graphically edited program into the source code of a 3GL or 4GL language.

These languages are mostly domain specific.

Programming paradigms

<u>Programming paradigm</u>

is an approach of programming which is supported by a set of theories, models and methods used in programming.

Several programming paradigms have been elaborated to facilitate programming since the first 3GL languages were introduced.

<u>For example:</u>

imperative programming, declarative programming, structured programming, procedural programming, functional programming, object-oriented programming, event-driven programming etc.

Some latter paradigms originate from former ones.

Some programming languages support some paradigms but not others.

Programming paradigms

Imperative programming
is a programming paradigm which based on the concept of the statement.
A statement is an instruction which orders the computer to perform a specified action.
Different statements causes different change in the state of a program.
A program is made up of a sequence of statements.
A statement may have simple or complex (with internal components).

Programming paradigms

Structured programming

is a programming paradigm which suggests that the whole problem should be decomposed into elementary problems.

The decomposition is suitable when the elementary problems

- do not overlap one another,
- the connections among them based on determined logic
- and each of them can be solved by using the tools of a programming language.

The permitted connections among the elementary problems are the following:

- sequence (ordered statements executed in sequence),
- conditional branch (or selection),
- conditional loop (iteration or finite repetition).

These connection types determine the control flow of a program.

Several programming languages support this paradigm and have structures for implementing the connection types above.

Programming paradigms

Procedural programming
is a programming paradigm which rests on the concept of the procedure and the procedure call.
A procedure, also known as routine, subroutine, or function (callable unit) contains a sequence of instructions which performs a specific task and its definition forms a closed unit within the source code of the whole program.
It cannot be executed independently of a program but its computational service may be used at any point of the program by means of the procedure call (or function call).
A procedure call (or function call) is a request in a program which performs the code of a procedure (or function).
A procedure may also contain one or more calls of other procedures in its code.
So, a larger program generally includes the definition and call of several procedures.

Programming paradigms

These three paradigms are collectively supported by several high-level programming languages such as FORTRAN, BASIC, Pascal, C, MATLAB, Python etc.

References

https://en.wikipedia.org/wiki/Programming_language_generations
https://en.wikipedia.org/wiki/Machine_code
https://www.slideshare.net/jocleph/generations-of-programming-languages-7747645
https://en.wikipedia.org/wiki/Assembly_language
https://en.wikipedia.org/wiki/Subroutine
https://en.wikipedia.org/wiki/Procedural_programming