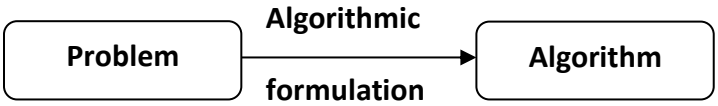
CHAPTER II: PROGRAM DEVELOPMENT

I/ Introduction:

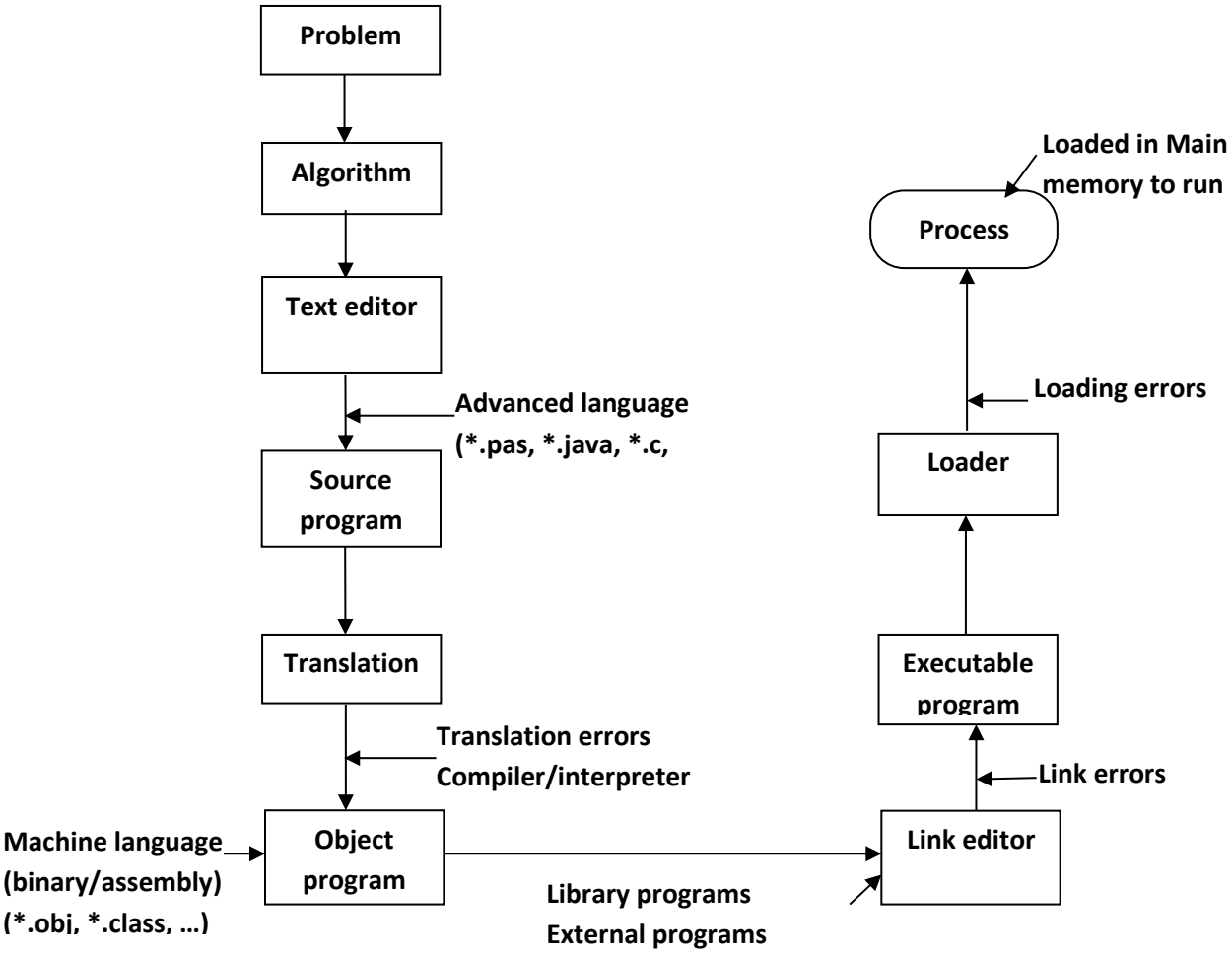
For a problem to be interpreted, executed and produce the results expected by the machine's processor, it must be formulated in a language that the machine can understand, and the operating system must control its execution.

To do this, the programmer begins by writing down his problem in a structured way, in the form of an algorithm.



This algorithm is written in human language, and to make it executable by the computer, the programmer takes it through several stages, using programming language and system tools.

II/ Program flow stages:



These steps ensure that the source code is properly translated into machine code, while allowing the programmer to correct any errors that may have arisen in each step.

1/ Program editing:

This is the stage at which we capture the algorithm established to solve the problem, and translate it into a program written in an advanced language (e.g. Java, Pascal, Delphi, C++, ...) or in Assembly language. The text editor is the tool that enables us to carry out this step. It's an interactive program, either associated with the system or with the programming language. It enables us to apply all the necessary operations to a file called "Source Code".

This source code has a specific extension depending on the language used (e.g. *.pas, *.java, *.cpp, ...).

Example: Notepad, Wordpad, MS-DOS Edit, C compiler editor, JEDPlus for java, ...

2/ Machine language translation:

The translator is a system tool that translates instructions written in a programming language into instructions written in machine language (Binary). There are two types of translators:

- **The compiler:**

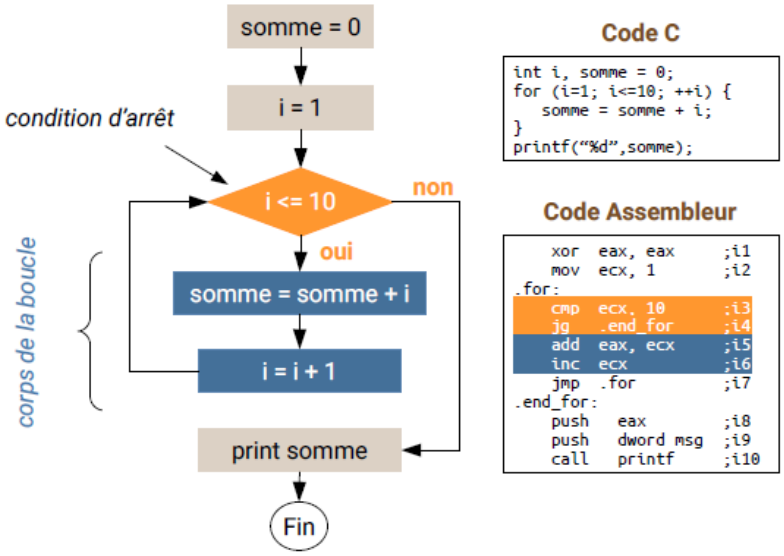
Is a software program of increasing complexity, which translates a source program into machine language via :

- ✓ **Lexical analysis**: the role of this phase is to recognize lexical units (tokens), by inspecting the source code character by character. These units can be numbers, identifiers, keywords, operators, etc. Each unit is described by a type (keyword, identifier, constant, operator, etc.) and a value (the name that appears in the source code). Then the result of this step:
  - Symbol table creation.
  - Elimination of blanks and comments.
  - Flagging of lexical errors.
- ✓ **Syntax analysis**: checks that the source code to be compiled complies with the language's syntax rules. The result is the creation of a syntax tree based on the lexical units and all the language's syntax rules. It updates the symbol table, adding any missing information such as identifier types, and flags any errors due to non-compliance with the language's syntax.
- ✓ **Semantic analysis**: checks the semantics of the program. This analyzer uses the syntax tree to identify operands and operations, and the symbol table to identify operand types. It signals any semantic errors that may be made in the code.
- ✓ **Code optimization**: Code optimization is a program modification strategy that endeavors to enhance the intermediate code, so a program utilizes the least potential memory, minimizes its CPU time and offers high speed.

- It is the fourth stage of a compiler, and it allows you to choose whether or not to optimize your code, making it really optional.
- It aids in reducing the storage space and increases compilation speed.
- It takes source code as input and attempts to produce optimal code.
- Functioning the optimization is tedious; it is preferable to employ a code optimizer to accomplish the assignment.

✓ **Transformation into object code**: this is the part that actually translates source code into object code.

During these stages, the compiler signals errors to be corrected, which may appear in each stage. The result of this phase is a file called object code, with a specific extension depending on the programming language (e.g. *.obj Pascal, *.class Java, ...).



- **<u>The interpreter</u>:**

The interpreter is a program that translates the program into machine language and executes it at the same time.

This type of translator does not result in a file containing the object code, which requires interpretation each time it is executed (e.g. HTML interpreter).

<u>Note</u>: the compiler defines two types of code in the object code:

- **Absolute code**, which has a fixed address in memory.

- **Translatable code**, which can be placed in any memory location.

The translation operation consists in adding to each location containing an address, the value of the actual address of the final memory location.

3/ Links editing:

When translating a program into object code, the compiler associates a type with each instruction:

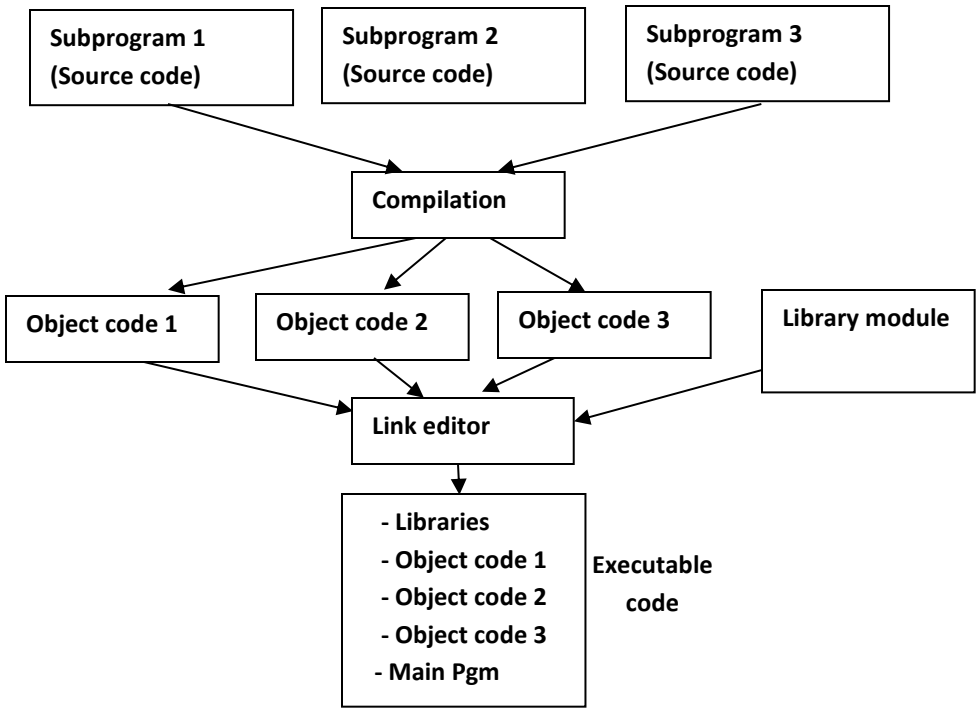
- Variable data
- Constant data
- Instructions
- Calls to procedures, modules or external variables.

In the last type, the programmer can reference structures or modules external to the module in question (these are external references from the language library or personal references). The compiler then marks these references without bringing them back and adding them to the object code.

Each reference is therefore associated with either:

- *A link which is internal to the program, but which can be accessed by other modules → usable link (external definition).*
- *An external link belonging to another module called in this program → link to be satisfied (external reference).*

The link editor is a system tool whose purpose is to accomplish the compiler's task by adding the object codes of all links to be satisfied to the main object code. The link editor then searches for the origin of each external reference and establishes a global table of modules containing the following information: module name, size and implementation address.



The work of the link editor is based on **the function table**, which contains all the information concerning the library. We can distinguish between two types of link editors:

Static linker: this is a linker that is set up once and for all, and generates a result file linking all external references with the main program, called the executable file. This type of link editor does not need to be redone each time it is run.

Example: the C language linker → *.exe files.

Dynamic linker: this is a linker that is created each time the program is run, and does not generate an executable file.

Example: the Java language link editor

4/ The loader:

Once the links have been edited, the program is ready to be executed, by loading it into main memory. This phase is performed by a system tool called the loader. This step consists in reading the instructions in secondary memory, and transferring them to main memory, while reading the loading address provided by the linker at the beginning. In this case, the program becomes a process.

There are two types of loading:

Absolute loading: the loaded code must not be put into another data block different from the one indicated in the object code (in other words, the code must be copied).

Translatable loading: loading can take place at any address in main memory. The way addresses are translated is based on information provided by the linker.

4/ The debugger:

This is a software that lets you run the program step by step, display variable values at any time and set breakpoints on conditions or program lines. It allows the programmer to control execution and detect the origin of uncorrected errors.