UNIVERSIDAD POLITÉCNICA DE YUCATÁN

Professor: Luis Gerardo Camara Salinas.

Subject: Structured Programming.

Homework #1 RESEARCH.

Jorge Gabriel Chuc Baqueiro

10/05/2021

Stages of compilation

There are five stages of compiling a program:

1. lexical analysis
2. symbol table construction
3. syntax analysis
4. code generation
5. optimization

Lexical analysis

- Comments and unnecessary spaces are removed.
- Keywords, constants and identifiers are replaced by 'tokens', which are symbolic strings to identify what the elements are.

  As we can see this first time is just for verify if we left something that is unnecessary because it’s just taking space unneeded.

  That’s just a revision for see our own mistakes or texting that doesn't make sense for someone else or just it’s something that the only one who could understand we are.

Symbol table construction

- A table stores the names and addresses of all variables, constants and arrays.
- Variables are checked to make sure they have been declared and to determine the data types used.

  At this time, we check our code but this time it’s not for erasing unnecessary things, it's the opposite, because we will check if our spelling has correctly written and if we’ve done that with the correct specifications and characteristics about as we need.

Syntax analysis

- Tokens are checked to see if they match the syntax of the programming language.
- If syntax errors are found, error messages are produced.

  This is so similar to the before step, because we are checking our syntax error, maybe it could be a missed semicolon, just and space, quotes marks, or something else. If we don’t check this our program could fail.

Code generation

- Machine code is generated in this stage.

  This time, we just run our program to see if it’s done. We’d see if our program isn't working just testing it, if it’d be not correctly, we just try to repeat the before steps for check, fix and repeat.

Optimization

- Code optimization makes the program more efficient so it runs faster and uses fewer resources.

  At this time we shall have finished all the before steps, our program is correctly programmed, and there’s just one more step, and it’s to make more efficient our program. Maybe reducing codes lines or trying to make it easier than before. That’s like when we finish a homework, but it’s kind of dirt, we just copy and fix the bad things being clearer to understand.

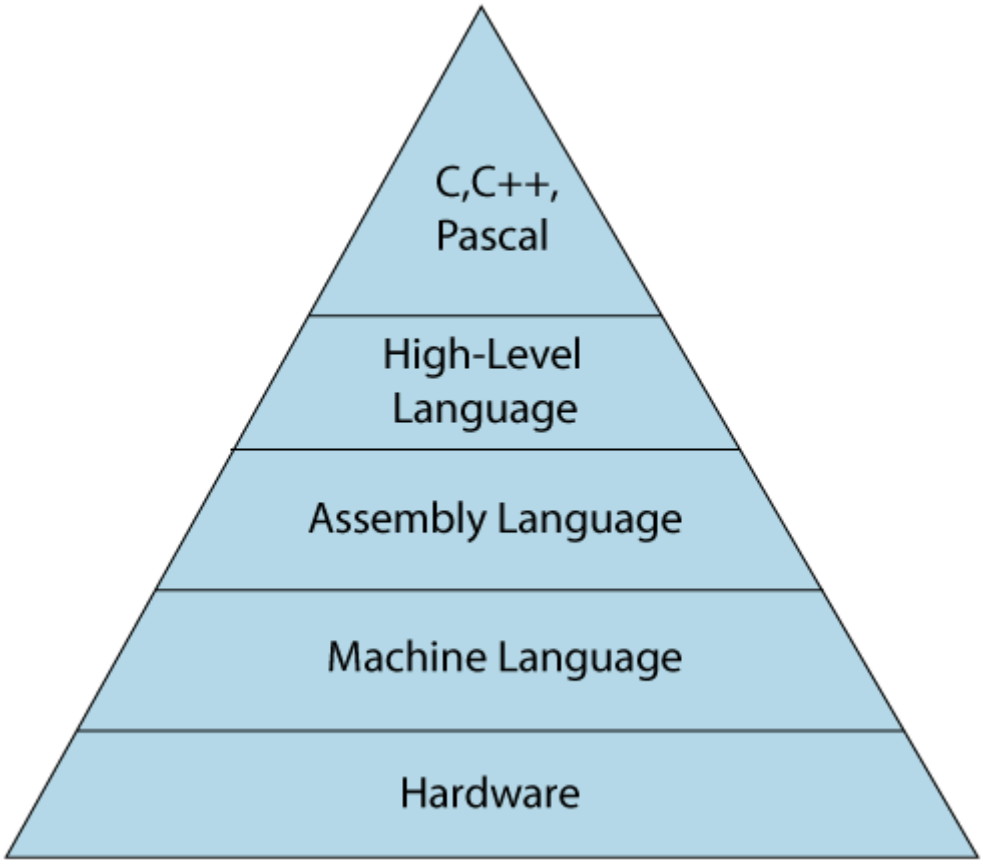
Levels of programming

There are four types of programming levels that are important:

1. Low-Level language
   2. Machine language
   3. Assembly language
   4. High-Level language

   Technical aspects of languages will consider linguistic structure, expressive features, possibility of efficient implementation, direct support for certain programming models, and similar concerns. There are 4 but there could be more than 4 languages because this concept is kind of abstract Sone examples:

   Machine, assembly, high-level, system, scripting, domain-specific, visual, esoteric languages.



Low-Level language

- In this language, programming provides no abstraction from the hardware, and its form ins just 0 or 1 forms. Which are the machine instructions.

For example A function in hexadecimal representation of 32-bit x86 machine code to calculate the *n*th Fibonacci number:

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
B9010000 008D0419 83FA0376 078BD989
C14AEBF1 5BC3
```

Machine-Level Language

- At this level, machine one, is a language that consist of a set of instructions that are in the binary for 0 or 1, as we know that computers can understand only machine instructions, the instructions can be only in binary codes. This is a very difficult language for programming because it's not easy to understand, also the maintenance is very high.

For example: Jumping to the address 1024:

```
[  op  |         target address         ]
    2                 1024                decimal
 000010 00000 00000 00000 10000 000000    binary
```

Assembly Language

- Assembly language contains some human-readable commands such as mov, add, sub, etc. The problems which we were facing in machine level language are reduce to some extent by using an extended for of machine-level language know as assembly language.
- As we know that computers can only understand the machine-level instructions, so we require a translator that converts the assembly code into machine code.

  For example: Returning to the original example, while the x86 opcode 10110000 (B0) copies an 8-bit value into the AL register, 10110001 (B1) moves it into CL and 10110010 (B2) does so into DL. Assembly language examples for these follow.

```
MOV AL, 1h        ; Load AL with immediate value 1
MOV CL, 2h        ; Load CL with immediate value 2
MOV DL, 3h        ; Load DL with immediate value 3
```

The syntax of MOV can also be more complex as the following examples show.

```
MOV EAX, [EBX]   ; Move the 4 bytes in memory at the address contained
in EBX into EAX
MOV [ESI+EAX], CL ; Move the contents of CL into the byte at address
ESI+EAX
MOV DS, DX        ; Move the contents of DX into segment register DS
```

Here are some differences between assembly and programming machine-level

Machine-level language	Assembly language
The machine-level language comes at the lowest level in the hierarchy, so it has zero abstraction level from the hardware.	The assembly language comes above the machine language means that it has less abstraction level from the hardware.
It cannot be easily understood by humans.	It is easy to read, write, and maintain.
The machine-level language is written in binary digits, i.e., 0 and 1.	The assembly language is written in simple English language, so it is easily understandable by the users.
It does not require any translator as the machine code is directly executed by the computer.	In assembly language, the assembler is used to convert the assembly code into machine code.
It is a first-generation programming language.	It is a second-generation programming language.

High-level programming language

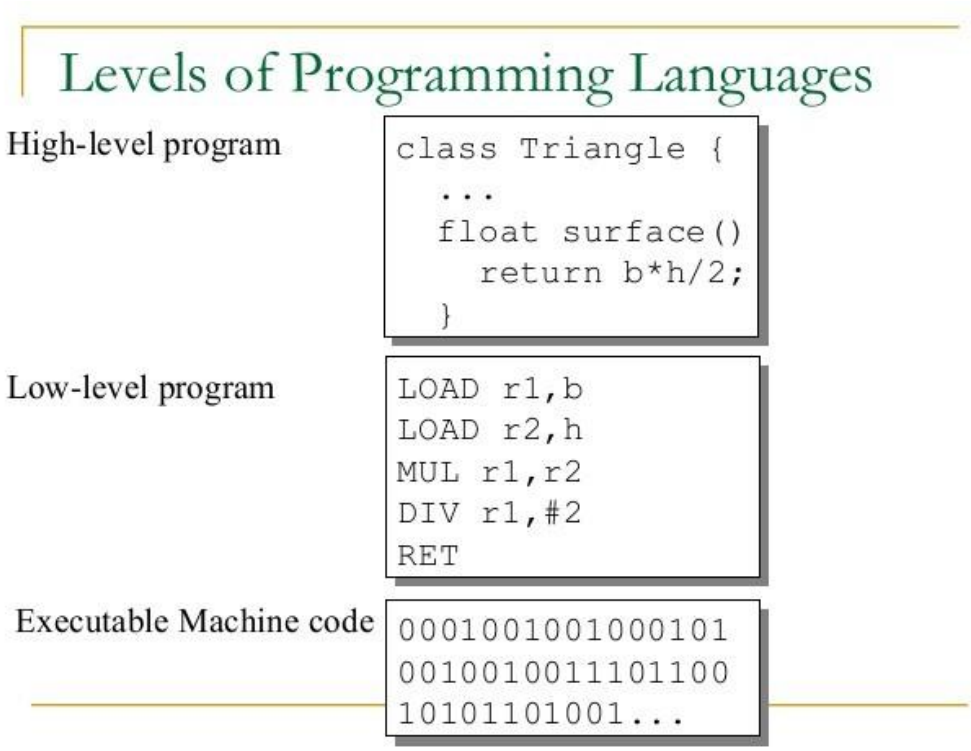
- The high-level language is a programming language that allows a programmer to write the programs which are independent of a particular type of computer. The high-level languages are considered as high-level because they are closer to human languages than machine-level languages.

- When writing a program in a high-level language, then the whole attention needs to be paid to the logic of the problem.

A high-level language gets away from all the constraints of a particular machine. HLLs may have features such as:

- Names for almost everything: variables, types, subroutines, constants, modules
- Complex expressions (e.g. 2 * (y^5) >= 88 && sqrt(4.8) / 2 % 3 == 9)
- Control structures (conditionals, switches, loops)
- Composite types (arrays, structs)
- Type declarations
- Type checking
- Easy, often implicit, ways to manage global, local and heap storage
- Subroutines with their own private scope
- Abstract data types, modules, packages, classes
- Exceptions

An example :



References

Computer Science | Understanding Computer Science. (2021). Program construction. 2021, de BBC Sitio web: https://www.bbc.co.uk/bitesize/guides/zmthsrd/revision/3

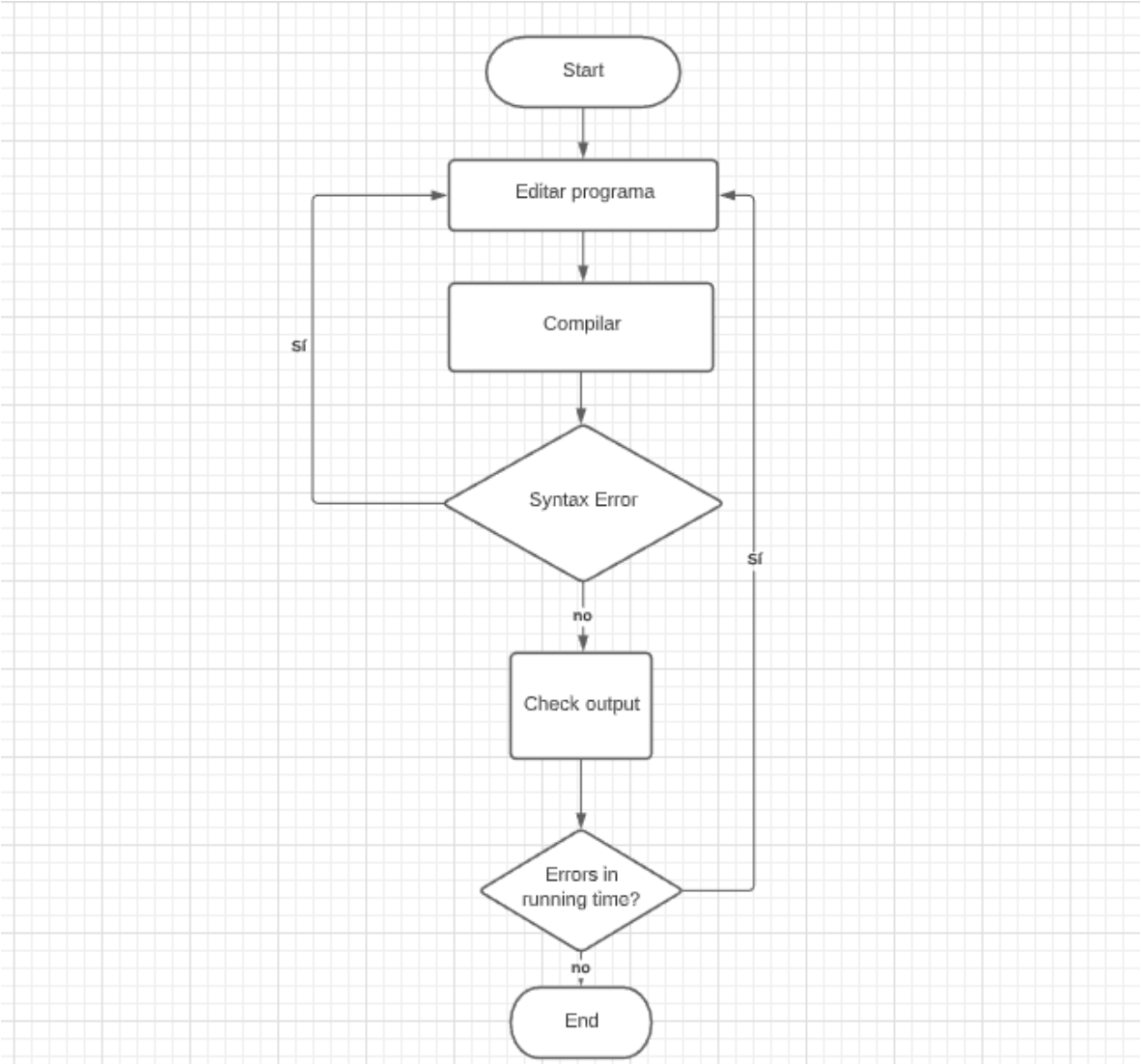
Priya Pedamkar. (Unknown). High level languages vs Low level languages. 2021, de EDUCBA Sitio web: https://www.educba.com/high-level-languages-vs-low-level-languages/

Unknown . (Unknown). Classifying Programming Languages. 2021, de LMU.EDU Sitio web: https://cs.lmu.edu/~ray/notes/pltypes/

JavaTPoint. (Unkwon). Classification of programming languages. 2021, de JavaTPoint.com Sitio web: https://www.javatpoint.com/classification-of-programming-languages

Uknown. (Uknown). programming language. 2021, de Wikipedia Sitio web: https://en.wikipedia.org/wiki/Programming_language

Isaac Computer Science. (Unknown). Stages of compilation. 2021, de isaaccomputerscience.org Sitio web: https://isaaccomputerscience.org/concepts/sys_trans_stageslevels



Classroom activity 1: Execution of a computer program