# Chapter 2 – A simple Compiler

## Luis Ricardo Espinoza Rodriguez A01229141

**Defining the language**

On this chapter the authors create a simple compiler. They start by laying out the definitions of the types, keywords and variables.

In this language there are only two types. Integer and float.

There are three keywords. **f** for declaring floats, **i** for declaring integers and **p** for printing a variable.

Variables. There are 23 possible variables. These are the lowercase roman alphabet excluding the three keywords.

**Going in depth to ac**

The language has to translate the language that was defined previously to *dc.* This is a stack-based calculator that uses RPN. But first we need to define the syntax for the ac language an also we need to define de semantic rules. Most programming languages use context free grammars to specify the language’s syntax and regular expressions to specify the basic symbols of the language.

Then we explain a context free grammar. There are terminal and non-terminal symbols.

The parser’s job is to generate a stream of tokens based on the input characters. It also creates an abstract syntax tree.

Then we do the semantic analysis. In this stage the Abstract Syntax Tree is edited≥.

***Symbol-table construction is a semantic- processing activity***

**MACHINE CODE GENERATION**

The final task undertaken by a compiler is the formulation of target-machine instructions that faithfully represent the semantics (i.e., meaning) of the source program.

Code generation proceeds by traversing the AST, starting at its root and working toward its leaves.