

##### Report ICL language

Computer Science

Interpretação e Compilação de Linguagens

# *Gonçalo Feliciano nº44292 Tomás Pessanha nº41774*

# *December of 2019*

# Introduction

Many programmers know how to write code in several different languages and nowadays more mature and complex languages keep showing up, in this project we were tasked with the development of a programming language with the option of running it on an interpreter or compile it to run in on the JVM, with this project we expect to be able to have a deeper understanding of the mechanisms we use on a daily basis and be able to work on similar projects in the future.

Note: This report will talk about the implementation choices and the work itself, there will be a readme included in the bitbucket repository on how to run the code.

# Language

This section will discuss the grammar and the choices made.

To easily view the grammar an html is generated in “/Language/src/main/Grammar/jjdoc” that contains a summary of the grammar.

The language is LL(1) meaning that the parser knows how to proceed simply by looking at the next token.

The project has level 2 requirements meaning it has both interpreter and compiler up to and including functions.

The language is divided into two types expressions and statements, the difference is that expression return a value, and statements do not and only return ***Void***, this is useful for instance in the ***println*** instruction where there is no point in printing a ***While*** for instance.

The only exception to this rule is an ***Assignment*** it was planned for the assignment to return the reference it was assigned to but it never got implemented correctly and it got scratched out of the project.

**Operator Precedence**

The following list shows the operator precedence in the language from least important to more important:

1. ‘:=’ (assignment)
2. ‘||’ (Disjunction), ‘|’ (Eager Disjunction)
3. ‘&&’ (Conjunction), ‘&’ (Eager Conjunction)
4. ‘==’ (Equality), ‘!=’ (Inequality)
5. ‘<=’ (Lesser or Equal), ‘>=’ (Greater or Equal), ‘<’ (Lesser), ‘>’ (Greater)
6. ‘+’ (Addition), ‘-’ (Subtraction)
7. ‘\*’ (Multiplication), ‘/’ (Division)
8. ‘as’ (as operator, converts int to float and float to int)
9. ‘-’ (Negate), ‘~’ (Not), ‘!’ (Dereference)

## Choices

### Syntax

The ***While*** was removed from ***Fact*** to the ***Statement*** this makes it impossible to have something like (5 + ***while***…), that makes no sense.

Same with ***PrintLn***.

The calling of functions (like f(5,0)), were moved to the Fact to be the same as a normal Identifier.

### Compiler

The compiler generates a scope 0, for the program even when no ***let*** is used, this is because in the beginning we believed that we would be able to add variables to the main scope so we wanted to have it ready, but it turns out that we never could.

## Types/Values

Apart from the expected like ***Int***, ***Boolean, Ref…*** types of the project the following were added, some are either partly implemented or not implemented at all:

* Any – Super to all types (Partial)
* Void – When no value is expected (Completed)
* Float – Added as an extra (Completed)
* String – Class was added but never used (Unused)
* Number – Super to all numeric types like ***Int*** and ***Float*** (Abstract)

# What is working

## Parser

Everything up to and including functions, some problems when using functions, example when calling a function that return another function ((f())()), this will not work, it could be easily fixed but it was noticed a bit too late.

## Interpreter

Everything up to and including functions

## Compiler

Everything up to and including functions but with some problems that sometimes show up and could not been fixed on time.

# Testing

An effort was made to create a good testing workflow, as such we use Junit for testing.

To add a test you create a file ‘name.icl’ in the “test/resources/compilerTests” directory.

The tests will find the file automatically.

To test the code in the begenning of the file you add the expected results in the form “//{value}:{type}(,{value}:{type})\*”.

# Conclusion

This was a very free and interesting work as it wasn’t just following the guidelines, we could use our imagination and have our own opinion on how we wanted the language to work.

It offered us a deeper understanding of language compilation that will be useful if we work in such a field and useful for understanding the code we write and what it is doing.