HULK

Rafael A. Sanchez Martinez
University of Havana
Group: C-113

Figure 1: Havana University Language for Kompilers

1 Introduccion

Welcome to the user manual of HULK, the programming language of the University of Havana. This manual is divided into 5 sections, each one dedicated to a part of the interpreter. The first section explains how to install the interpreter, the second section explains how to use the interpreter, the third section explains how to use the lexical analyzer, the fourth section explains how to use the syntax analyzer and the fifth section Explains how to use the expression evaluator.

1.1 Start

- To start using the interpreter, you must download the source code of the interpreter from the project's github repository. Once the source code has been downloaded, the project must be run using the 'dotnet run' command from a terminal open in the project folder.
- Another option is to run the script which will take care of everything
- See previous report in `https://github.com/ARKye03/mini_kompiler.git`

1.2 Summary

HULK (Havana University Language for Compilers) is an educational, type-safe, object-oriented and incremental programming language, designed for the Introduction to Compilers course of the Computer Science degree at the University of Havana.

A simple 'Hola Mundo' in HULK looks like this:
print('Hola mundo')

From a bird's eye view, HULK is an object-oriented programming language, with simple inheritance, polymorphism, and class-level encapsulation. Additionally, in HULK it is possible to define global functions outside the scope of all classes. It is also possible to define a single global expression that constitutes the entry point to the program.

Most syntactic constructs in HULK are expressions, including conditional statements and loops. HULK is a statically typed language with optional type inference, meaning that some (or all) parts of a program can be annotated with types and the compiler will check all operations for consistency.

But, in this case, since it is a project for 1st year students, it is a bit simplified. In this case, let's say, we have a one-line interpreter, which allow us, to evaluate expressions like:

- Like I said above, print("Hello World")
  - Which is equivalent to $print(\text{“}Hello\text{”} + \text{“}World\text{”})$
  - And print("Hello" @ " World")
- All basics 'instructions' can handle simple math expressions like:
  - $\text{“}2 \pm 5;\text{”}$ —— $\text{“}147.6 \pm -63;\text{”}$
  - $\text{“}8 \times 2;\text{”}$ —— $\text{“}200 \div 3;\text{”}$
  - $\text{“}2^{64};\text{”}$ —— $\text{“}sqrt(3969)\text{”} \rightarrow \text{“}\sqrt{3969}\text{”}$ —— $\text{“}\log(10);\text{”}$
  - Simple functions to handle trigonometrics expressions.
    - $\text{“}\sin(\pi)\text{”}$
    - $\text{“}\cos(\pi)\text{”}$
    - $\text{“}\tan(\pi)\text{”}$
    - $\text{“}\cot(\pi)\text{”}$
    - Others can be defined on the way too.
  - And thats pretty much it $\wedge \times \wedge$

2 Analizador Lexico

```
private object power()
{
    var left = primary();

    while (true)
    {
        var token = lexer.get_next_token();

        if (token.type != TokenType.Operator || token.value != "^")
        {
            lexer.unget_token(token);
            return left;
        }
        var right = primary();
        left = BinaryOperation(left, token, right);
    }
}
```

3 Introduction

This is the report of my compiler project. The compiler is written in C++ and

4 Introduction

This is the report of my compiler project. The compiler is written in C++ and

5 Introduction

This is the report of my compiler project. The compiler is written in C++ and