Departamento de Ciência da Computação – Universidade Federal de Roraima (UFRR)  
Boa Vista – RR

katherineghalabi@gmail.com, emailalunoB@ufrr.br

**Estudo sobre a Linguagem de Programação Julia**

**Katherine Halabi e**

**Resumo**

Este relatório apresenta uma aplicação prática que utiliza as ferramentas da linguagem de programação Julia para calcular o consumo de energia elétrica individual de uma residência. Foi utilizado para os cálculos a situação hipotética de uma residência com determinados equipamentos de uso comum entre três moradores. Um arquivo csv que contém os dados de potência dos equipamentos, tempo de consumo diário e consumo equivalente é lido. A saída do sistema se dá por um gráfico com as três curvas do consumo individual registrado no período de um mês.

Palavras chave: Julia, Atom, consumo, GR.

1. **História do Surgimento da Linguagem**

Julia é uma linguagem de programação compilada (JIT – Just in time) open source de alto nível projetado com foco na computação científica e numérica de alto desempenho. É relativamente jovem, posto que teve início no MIT em agosto de 2009 e, em fevereiro de 2012, tornou-se open source. É fruto do trabalho de três pesquisadores: Stefan Karpinski, Jeff Bezanson, e Viral Shah. Foi pensada como uma linguagem para computação científica suficientemente rápida como C ou Fortran, mas igualmente fácil de aprender como o MATLAB e o Mathematica, com o objetivo de facilitar a modelagem computacional. É escrito em C, C++ e Scheme, e a biblioteca padrão é escrita utilizando a própria linguagem Julia. Possui forte inspiração em MATLAB, Lisp, C, Fortran, Mathematica, Python, Perl, R, Ruby, Lua, além de compartilhar muitas características de Dylan e Fortress.

1. **Domínios de aplicação**

Julia é projetado para resolver problemas matemáticos numericamente, que consiste na manipulação numérica dos dados. E não deve ser confundido com programas como o Mathematica ou Maple, que trabalham com soluções simbólicas, fazendo a manipulação algébrica. A maioria dos problemas matemáticos reais (particularmente em engenharia) não têm soluções simbólicas puras.

1. **Paradigmas suportados pela linguagem**

Multiparadigma, contendo características de Funcional, Imperativo e Orientação a Objetos. Porém não é necessariamente uma orientação a objetos já que em Julia, todos os valores são objetos, mas as funções não são atreladas aos objetos em que operam.

1. **Variáveis e tipos de dados**

* Tipos inteiros de 8, 16, 32, 64 e 128 bits.
* Bool valendo ou true (verdadeiro) ou false (falso), que correspondem numericamente a 1 ou 0, respectivamente.
* Char, string
* Ponto flutuante:

Float32 — Números de ponto flutuante 32-bit seguindo o padrão IEEE 754.

Float64 — Números de ponto flutuante 64-bit seguindo o padrão IEEE 754.

* Números complexos: Para definir, usa-se a parte real em conjunto com a parte imaginária **im. Ex: *In*:** (1 + 8im) + (2 - 3im)

***Out*:** (3 + 5im)

* Constantes
* Tuplas
* Conjuntos
* Sequências
* Listas, array, vetores e matrizes

1. **Comandos de controle**

* Ponto e virgula ao final do comando oculta o resultado;
* Não há a necessidade de vetorizar uma variável do tipo vetor ou matriz;
* Comentários são na forma de #;
* O ponto antes de um operador matemático indica operação elemento-elemento em um vetor (ex: .\*, ./, .^);

# Uma informação do tipo: variable::Bool significa que a variável é do tipo booleano;

# A notação reversa polonesa (RPN) pode ser aplicada aos operadores matemáticos;

# Todos os pacotes são escritos usando a primeira letra maiúscula entre aspas duplas;

# Julia é "Case-Sensitive" F e f são diferentes;

# Funções e comandos de controle de fluxo (if, for, while...) necessita de "end" para terminar o comando;

# Macros são iniciados por @;

* O tipo é muito importante. 1.0 (float) e 1 (inteiro) são diferetes;

1. **Escopo (regras de visibilidade)**

Escopo global: arquivos .jl, módulos ou variáveis do prompt interativo

Escopo local Soft: for, while, try-catch.

Escopo local Hard: funções.

1. **Exemplo prático**

O exemplo pratico da linguagem junto com exemplo encontra-se no github, no link https://github.com/mitchelson/LP\_2017\_Projeto\_Final\_JULIA.git

1. **Conclusões**

A linguagem Julia mostrou-se ter melhor desempenho em várias aplicações científicas devido a sua forma de compilação Just in Time, por ser dinâmica e compilar em tempo de execução provendo um ambiente produtivo para experimentação e eficiente para produção de aplicativos que precisam de alto desempenho.

1. **Referências**

Sites oficiais da linguagem.

<https://julialang.org/>

<https://juliabox.com/>

O manual de julia.

<http://julia-ptbr.readthedocs.io/pt_BR/latest/manual/introduction.html>

Introduction to Computational Physics <http://www.infis.ufu.br/images/users/gerson/PhysComp/FisComp.pdf>