#### Instituição de Ensino FEI

# Compiladores

## CC6252

### SAD Programming language

#### Integrantes:

Guilherme Gomes Chagas - 22.119.013-5

Luca Milla - 22.119.002-8

Matheus Luiz Gonçalves Pelicer - 22.119.024-2

Thiago Soares Cardoso da Silva - 22.119.044-0

Ciência da Computação FEI

6º Ciclo – 2021

Período Noturno

**Tipos primitivos**:

A SAD suporta os seguintes tipos primitivos de valores: integrity (integer), breakdown (float), string e camus (boolean). Pode-se ver exemplos do uso de cada um desses tipos na seção de declarações de variáveis.

- o tipo Integrity é o mesmo à int em outras linguagens, sendo aceitos apenas números inteiros.

- o tipo Breakdown é o mesmo à float em outras linguagens, sendo aceitos apenas números com ponto flutuante.

- o tipo string é o mesmo ao mesmo tipo outras linguagens, porém são apenas aceitos caracteres minúsculos, espaços e vírgulas.

- o tipo Camus é o mesmo ao bool em outras linguagens, seus valores aceitos são “suicide” sendo o equivalente ao “true” e “coffee” o equivalente ao “false”.

**Declaração de variáveis**:

Os nomes de varáveis em SAD podem ser compostos por qualquer letra maiúscula ou minúscula apenas, desta forma não sendo permitidos caracteres especiais. As variáveis devem ser inicializadas com algum valor, seja ele qualquer um dos valores primitivos da linguagem, como nos exemplos abaixo:

integrityVar = 5

breakdownVar = 6.5

camusVar = suicide

stringVar = “thats a string”

**Condições**:

A linguagem SAD é composta pelo bloco what -ever, sendo comparado diretamente ao if -else do Python, serve como um dos controles de fluxo da SAD, as comparações utilizadas não devem conter parênteses, já que a linguagem não suporta esse tipo de priorização de cálculo.

what damageVariable + goodVariable >= legendaryVariable{

printingVariable = "ha ha, you are too weak"

}

ever{

printingVariable = "i declare the legendary variable even legendary"

legendaryVariable = legendaryVariable + 1

}

**Cálculos**:

Cálculos matemáticos simples, sendo eles, soma, subtração, multiplicação e divisão, podem ser feitos como em qualquer outra linguagem de programação, como pode ser visto nos exemplos abaixo.

sumResult = 5 + 10

minusResult = sumResult – 4

multiResult = sumResult \* minusResult

divResult = 5 / 10

**Geração de código**

A linguagem SAD utiliza para interpretação e geração de seu código um Parser Recursivo Descendente para a geração de seus tokens e a partir desses tokens o interpretador faz o uso da estrutura de Árvore Sintática Abstrata (AST) para a organização dos tokens gerador pelo Parser.

**Parser Recursivo Descendente:**

O Parser Recursivo Descendente funciona baseado na GLC desenvolvida para a SAD, disponível no repositório da própria linguagem. Este funciona criando tokens para cada uma das cadeias terminais e não-terminais da linguagem usando uma estrutura de pilha iniciando a partir dos não-terminais.

**Árvore Sintática Abstrata:**

O compilador da SAD utiliza da geração de uma Árvore Sintática Abstrata como uma forma de organização para que possa ser feita posteriormente a análise semântica do código. Cada operação deve ter seu próprio tipo de nó que serão ligados uns aos outros com a geração da árvore, por exemplo, para as operações básicas, devem existir nós específicos para soma, subtração, multiplicação e divisão.