EP2 - MAC0316 Conceitos de Linguagens de Programação

Washington Luiz Meireles de Lima

nUSP: 10737157

Ygor Tavela Alves

nUSP: 10687642

Instruções

- Primeiramente deve-se compilar este programa utilizando o comando `make`.
- Após o passo da compilação, caso seja necessário apenas traduzir a gramática para uma sintaxe reconhecida pelo parser do racket, execute:

```
./mcalc < arquivoteste
```

- Caso, queira também executar o código traduzido utilizando o parser do racket, execute:

```
./mcalc < arquivoteste | ./direto
```

Testes

- Todos os arquivos de testes, estão localizados na pasta tests e estão nomeados com o seguinte padrão `test-<type>-<number>`, onde `type := func | var` e `number:= 1 | 2 | 3`, ou seja, um possível teste para uma função está nomeado como `test-func-2`.
- Há um shell script básico na pasta raiz, `tests.sh`, que realiza a execução de todos os testes, imprimindo as suas saídas no terminal. Para executá-lo basta executar:

```
./tests.sh
```

- Lembrando que o tests.sh deve ter permissão para execução, para isto, basta utilizar o comando `chmod u+x tests.sh`

Documentação

Nos códigos estão presentes alguns comentários sobre o código em si, além disso, as decisões tomadas referente a escolha da nossa gramática é descrita abaixo para cada caso implementado:

Funções: Em nossa linguagem, em relação as funções, podemos utilizar funções de forma auto-invoca utilizando lambda da seguinte forma:

```
return 10 + (lambda x : x + x)(2);
```

A expressão esperada para o bloco acima é `(+ 10 (call (func x (+ x x)) 2))`*, e o valor 14.*

Ademais, podemos também utilizar funções nomeadas da seguinte forma:

```
fn teste(y) {
    y = y + 1;
    return y;
};
return teste(3);
```

A expressão esperada para o bloco acima é `(def teste (func y (seq (:= y (+ y 1)) y)) (call teste 3))`*, e o valor 4.*

Variáveis: Em nossa linguagem podemos definir a símbolos valores de expressões ou, então, lambdas. Para isto devemos inicialmente associar o símbolo e o seu valor a um local da memória, para isto basta utilizar

```
let x = 10;
let y = (lambda z : 2 * z);
return x + y(3);
```

A expressão esperada para o bloco acima é `(def x 10 (def y (func z (* 2 z)) (+ x (call y 3))))`*, e o valor igual a 16.*

Deve-se destacar que em todo bloco de código é esperado um return de uma expressão. Além disto, podemos também realizar mutações nos valores inicialmente atribuídos aos símbolos, como descrito abaixo:

```
let x = 10;
x = x + 10;
return x;
```

A expressão esperada para o bloco acima é `(def x 10 (def x (+ x 10) x))`*, e o valor igual a 20.*

Observação

As funções que foram implementadas no EP1 se encontram nos respectivos arquivos de teste:

1. **factorial**: *test-func-1*
2. **cube**: *test-var-2*
3. **half**: *test-var-3*