Unoeste
Campus I Rua José Bongiovani, 700
Cidade Universitária CEP 19050 900
Fone (18) 3229 2003

Faculdade de Informática de Presidente Prudente
Fone (18) 3229 1060
www.unoeste.br/fipp
e-mail: fipp@fipp.unoeste.br

FIPP FACULDADE DE INFORMÁTICA DE PRESIDENTE PRUDENTE CAMPUS I

Estruturas de Dados II

Prof. Francisco Assis da Silva

– Interpretador de programas escritos em JavaScript –

Seu trabalho é fazer um interpretador (uma simulação de execução) de um pequeno programa na linguagem JavaScript.

Exemplo de programa em JavaScript:

```
function calcula(x, y)
{
    return x + y;
}
let a = 5;
let b = 10;
let c = 0;
c = calcula(a, b);
console.log(a, b, c);
a = calcula(7, a+b+c);
console.log("7", a+b+c, a);
c = calcula(a*b, a/b);
console.log(a*b, a/b, c);
a = calcula(b, b+b);
let msg = 'resultado = ';
console.log(msg, a, b, c);
```

Na Figura 1 é mostrada a tela do interpretador (simulador), com a execução do programa apresentado no exemplo acima.

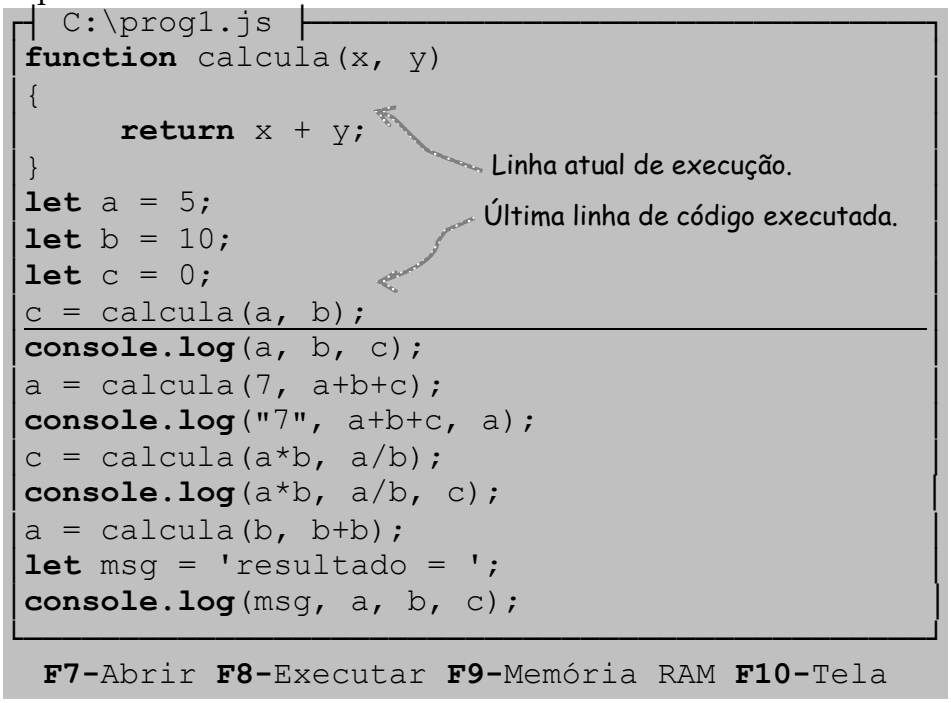


Figura 1. Simulação da execução de um programa na linguagem JavaScript contendo a tela do interpretador (simulador).

A partir da abertura do código-fonte do programa (arquivo ".js"), você deverá percorrer linha por linha do arquivo e inserir cada termo (*token*) em uma estrutura de Lista de Listas, conforme pode ser observado no exemplo na Figura 2.

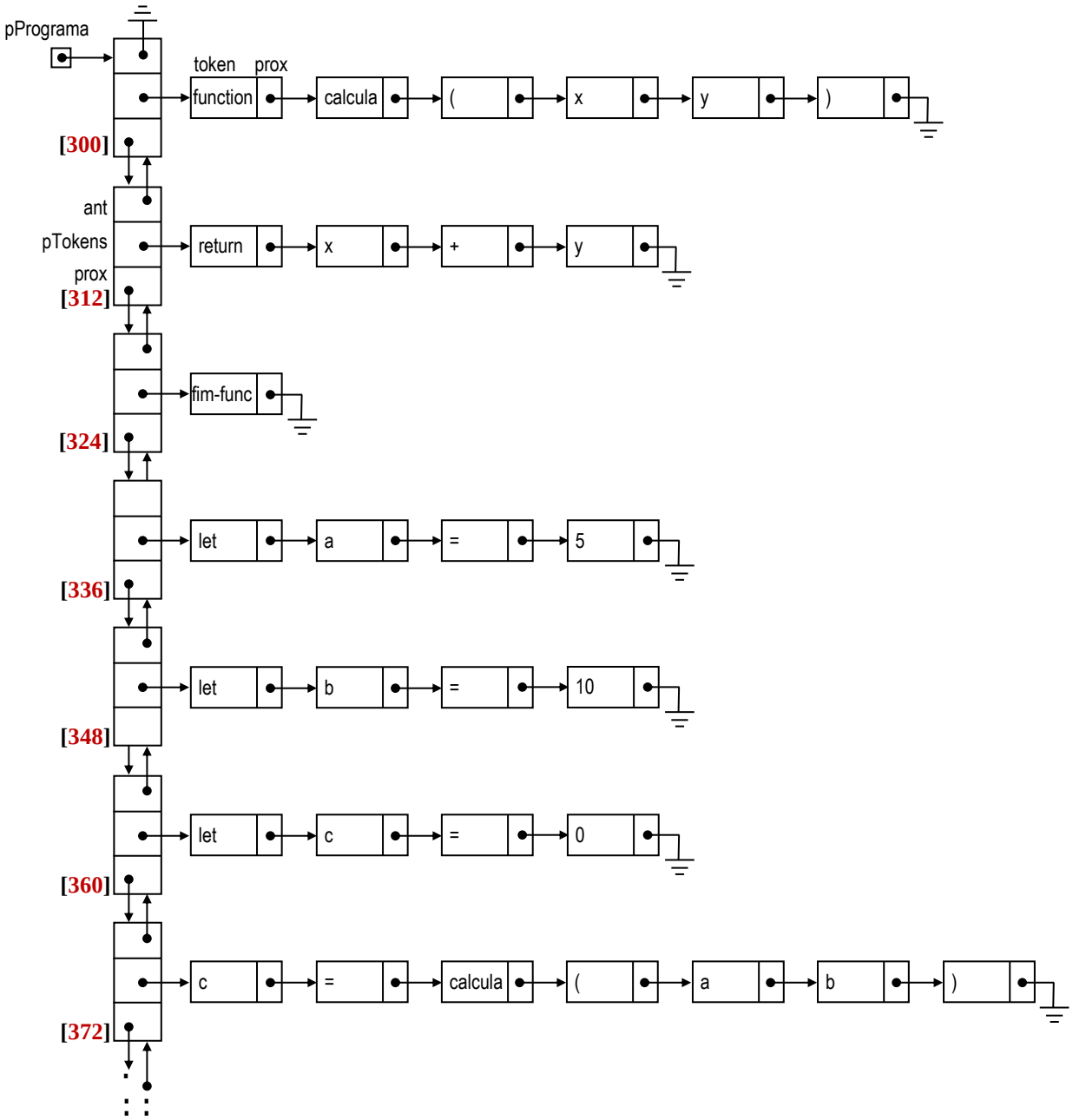


Figura 2. Parte inicial do programa de exemplo inserido na estrutura de Lista de Listas, contendo cada termo do programa em JavaScript separado, linha por linha.

Durante a execução passo a passo do programa, você precisará gerenciar a memória RAM do programa JavaScript utilizando uma pilha, contendo as variáveis do programa e também os endereços das chamadas de funções usadas para controlar escopo. Essa pilha deve ser implementada de forma dinâmica (caixinhas com ponteiros). A Figura 3 é apenas uma representação dessa pilha de variáveis em formato de tabela, contendo o estado atual da memória no momento da execução do programa de exemplo (Figura 1). Observe que no endereço &112 da pilha de variáveis contém um ponteiro (endereço representativo de memória [372]) para o *heap* em que a sétima caixa de linha (na lista vertical) foi alocada. Isso indica a linha de execução que fez a chamada da função "c = calcula(a, b);", ou seja, quando a função "calcula" terminar a sua execução, o argumento de retorno é atribuído à variável "c" e o interpretador (simulador) passará a executar as demais linhas, abaixo dessa.

	&	identificador	valor	ponteiro
Topo →	120	y	10	NULL
	116	x	5	NULL
	112	[**endereço**]	-	**[372]**
	108	c	0	NULL
	104	b	10	NULL
	100	a	5	NULL

Figura 3. Estado da memória RAM do programa de exemplo em execução (Figura 1).

O simulador deve conseguir lidar com os seguintes comandos e estruturas:

a) Fazer chamadas de funções com passagem de parâmetro e retornar valores por `return`;
b) Fazer chamadas de funções dentro de outra função;
c) Resolver equações aritméticas e algumas funções matemáticas: raiz quadrada e módulo (valor absoluto);
d) Executar comparação usando `if` e `else` e operadores lógicos `&&`, `||` e `!`;
e) Manipular Arrays em JavaScript, que serão simulados com Lista Generalizada;
f) Executar a estrutura de repetição `while`, `do-while` e `for`;
g) Executar o comando de exibição `console.log()`.

#Função console.log(...)

Exemplos:

```
const calc = 2 + 2;
let svalor = '237';
console.log('Resultado: ', calc);
console.log('Resultado: ', calc.toString());
console.log('Resultado: ', parseInt(svalor));
```

#Operadores

Aritméticos	Comparação	Lógicos
+	== (igual)	&&
-	!= (diferente)	||
*	> (maior)	!
/	< (menor)	
%	>= (maior ou igual)	
**	<= (menor ou igual)	

Exemplos:

```
console.log("3 * 2 = ", 3*2);          //6
console.log("7 / 2 = ", 7/2);          //3.5
console.log("4 - 2 = ", 4-2)           //2
console.log("3 ** 2 = ", 3**2)         //9
console.log("3 % 2 = ", 3%2)           //1
```

#Comando de Seleção (if)

```
if (condição)
    comando(s);
else
    if (condição)
        comando(s);
    else
        comando(s);
```

Exemplo:

```
let var1 = 1;
let var2 = 2;
if (var1 == 1)
    console.log("var1 igual a 1");
else
    if (var1 !=1 && var2 == 2)
        console.log("var1 diferente de 1 e var2 igual a 2");
    else
        if (var1 >= 1000 || var2 <= -1000)
            console.log ("var1 maior que 1000 ou var2 menor que -1000");
        else
            console.log ("nenhuma das alternativas anteriores");
```

Comando de repetição (while)

```
while (condição)
    comando(s);
```

Exemplo:

```
let i = 0;
while (i<10)
{
    console.log("i =", i); // 0 1 2 3 4 5 6 7 8 9
    i++;
}
```

Comando de repetição (do-while)

```
do
    comando(s);
while (condição);
```

Exemplo:

```
let i = 0;
do
{
    console.log("i =", i); // 0 1 2 3 4 5 6 7 8 9
    i++;
} while (i<10);
```

Comando de repetição (for)

```
for (variável = vi; variável <= vf; vi++)
    comando(s);
```

Exemplo:

```
for(let i=0; i<10; i++)
{
    console.log("i = ", i); // 0 1 2 3 4 5 6 7 8 9
}
```

Arrays

Exemplos:

```
let array = ['oi', 2.0, 5, [10, 20]];
console.log(array[0]);    // oi
console.log(array[1]);    // 2.0
console.log(array[2]);    // 5
console.log(array[3]);    // [10, 20]
console.log(array[3][1]); // 20
```

Podemos ordenar um array. Exemplo:

```
let frutas = ['laranja', 'banana', 'abacaxi'];
frutas.sort();
console.log(frutas);         // ['abacaxi', 'banana', 'laranja']
console.log(frutas.length); // 3
```

Podemos inserir no início de um array. Exemplo:

```
let frutas = ['laranja', 'banana', 'abacaxi'];
frutas.unshift('ameixa');
console.log(frutas);         // ['ameixa', 'laranja', 'banana', 'abacaxi']
console.log(frutas.length); // 4
```

Podemos inserir ao final de um array. Exemplo:

```
let frutas = ['laranja', 'banana', 'abacaxi'];
frutas.push('ameixa');
console.log(frutas);         // ['laranja', 'banana', 'abacaxi', 'ameixa']
console.log(frutas.length); // 4
```

Podemos remover o primeiro elemento do array. Exemplo:

```
let frutas = ['laranja', 'banana', 'abacaxi'];
let aux = frutas.shift();
console.log(frutas);   // ['banana', 'abacaxi']
console.log(aux);   // laranja
```

Podemos remover o último elemento do array. Exemplo:

```
let frutas = ['laranja', 'banana', 'abacaxi'];
let aux = frutas.pop();
console.log(frutas);   // ['laranja', 'banana']
console.log(aux);   // abacaxi
```

Mais operações com arrays: Exemplo:

```
let frutas = ['laranja', 'banana', 'abacaxi']
frutas.push([3, 2, 1])
frutas.push([[9.9, ['a', 'b']]])
console.log(frutas) // ['laranja', 'banana', 'abacaxi', [3, 2, 1], [[9.9, ['a', 'b']]]]

frutas[3].sort();
console.log(frutas) // ['laranja', 'banana', 'abacaxi', [1, 2, 3], [[9.9, ['a', 'b']]]]
```

Funções

Exemplos:

```
function calcula_quadrado(x)
{
    let quadrado = x ** 2;
    return quadrado;
}

console.log(calcula_quadrado(5));  // 25

#----------------------------
function imprime_nome(nome)
{
    console.log("Nome: " + nome);
}

imprime_nome("Joao");  // Nome: Joao
imprime_nome("Jose");  // Nome: Jose
imprime_nome("Maria"); // Nome: Maria
```

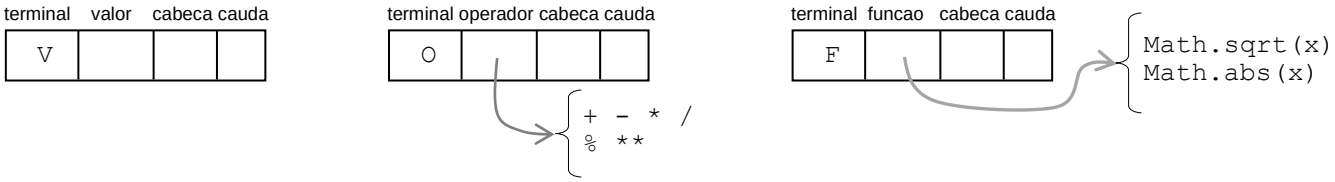
Seu programa deve permitir ao usuário abrir um arquivo fonte na linguagem JavaScript (F7), executar o programa passo a passo (F8), mostrar o conteúdo da memória RAM no momento atual da execução (F9) e mostrar a tela (resultados dos comandos `console.log()`) (F10). Para a execução passo a passo, deve-se apertar ENTER para executar a linha atual (linha com fundo demarcado, observe a Figura 1) e passar para a próxima linha.

Você deve utilizar apenas estruturas de listas para construir seu programa interpretador da linguagem JavaScript (simulador):

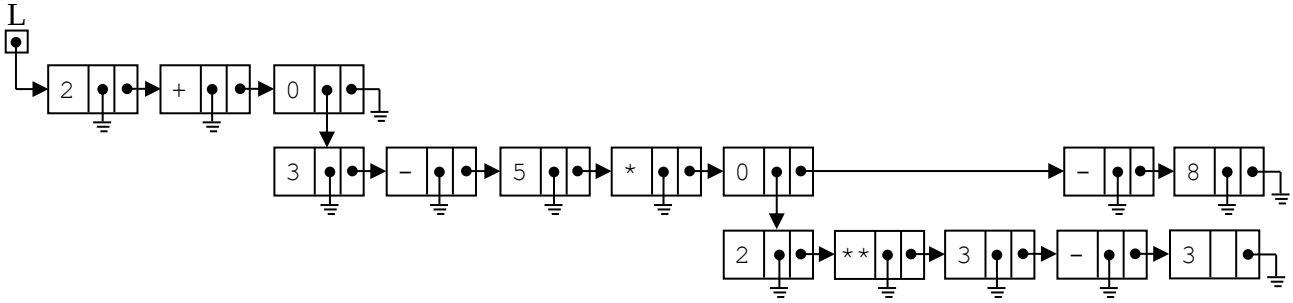
a) Deverá ter uma lista para representar a pilha de variáveis;
b) Deverá ter uma lista de listas para guardar as linhas de comandos do programa em JavaScript que abriu. Cada nodo da lista principal (considere uma lista vertical) terá uma lista vinculada a ela para guardar em cada nodo os termos (*tokens*) da linha de comandos (considere uma lista horizontal vinculada ao nodo vertical).
Exemplo: Se a linha de comandos fosse "**`function`** `calcula(x, y)`", você teria, na lista horizontal, um nodo para "`function`", outro para "`calcula`", outro para "`(`", outro para "`x`", outro para "`y`" e outro para "`)`";

Para resolver as expressões matemáticas, você deverá utilizar uma lista generalizada, diferente do modelo conceitual de aula. A partir de uma **string** passada por parâmetro para uma de suas funções, crie um algoritmo para construir uma lista generalizada segundo as prioridades e resolver a expressão "podando" os nodos da lista generalizada. A medida que cada linha de operação for resolvida, os nodos da lista generalizada devem ser removidos e sobrar

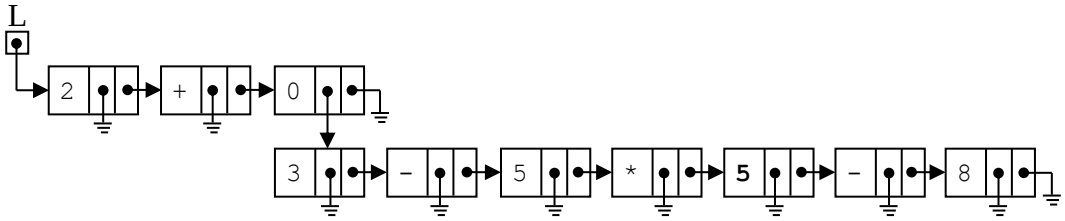
apenas um nodo com o resultado, cujo valor deve ser retornado. O nodo da lista deverá ser construído com o uso de **union**, uma vez que o nodo poderá ser um valor (**float**), um operador (**char[3]**) ou uma função (**sqrt** ou **abs**).
Tipos de nodos:



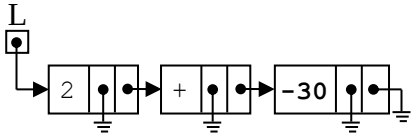
Exemplo de expressão aritmética a ser resolvido: "`2+(3-5*(2**3-3)-8)`"
A lista construída será:



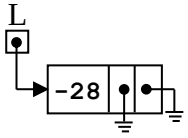
Após resolver os parênteses mais internos:



Após resolver o segundo nível de parênteses:



Após resolver toda a expressão:



Observações importantes:
Não utilize funções prontas da linguagem C, como por exemplo, strtok para separar string, comandos de string diferentes dos que aprendeu em aulas, também não pode usar filas, pilhas prontas etc.
Para resolver cada lista de expressões (cada nível da lista generalizada), você pode utilizar duas pilhas (criadas por você), uma de valor e outra de operador.

Alguns exemplos de códigos em JavaScript contendo alguns comandos e estruturas que seu programa deverá ser capaz interpretar (simular a execução):

Exemplo 1:

```
function calcula1(I, J, K)
{
    I=I+3;
    J=I+K;
    K=K+I+J;
    return K;
}

function calcula2(I, J, K)
{
    I=I+2;
    J=J+K;
    K=K+1;
    return K;
}

function executa(p1, p2, p3)
{
    p1 = calcula1(p3, p1, p2);
    p2 = calcula2(p2, p2, p2);
    return p1 + p2 + p3;
}

let A = 3;
let B = 5;
let C = 7;
let res = executa(A, B, C);
console.log("Resultado = ", res);
```

Exemplo 2:

```
function faz_uma_repeticao_while(X, n)
{
    let i=0;
    let result=1;
    while(i<n)
    {
        result=result * X;
        i++;
    }
    return result;
}

let s = faz_uma_repeticao_while(2,4);
console.log("Resultado = " + s);
```

Exemplo 3:

```
function faz_uma_repeticao_for(X, n)
{
    let i;
    let result=1;
    for (i=1; i<=n; i++)
        result=result * X;
    return result;
}
```

```
s = faz_uma_repeticao_for(2,4);
console.log("Resultado = ", s);
```

Exemplo 4:

```
function juros_composto(P, i, n)
{
    let M = P*(1 + i/100)**n;
    return M;
}

res = juros_composto(2000, 1.5, 10);
console.log("Resultado = " + res);
```

Exemplo 5:

```
function funcao(chave)
{
    let i=0;
    nomes = ["Joao", "Jose", "Maria"];
    numeros = [17, 123, 33];
    console.table(nomes);
    console.table(numeros);
    lista_mista = ["ola", 2.0, 5*2, [10, 20]];

    while(i<lista_mista.length && chave!=lista_mista[i])
        i++;

    if (i<lista_mista.length)
        return "S";
    return "N";
}

let ret = funcao("ola");
if (ret == "S")
    console.log("Econtrou!!!");
```

Importante!!!

O trabalho é em dupla e deverá ser apresentado pelos dois alunos! A data de apresentação já está no cronograma da disciplina!

O que deve enviar no Aprender antes da apresentação:

a) Todos os códigos-fonte;
b) Os programas em JavaScript usados para testar o simulador. Vocês precisam acrescentar outros programas de teste!
c) Um .docx com o desenho das estruturas (caixinhas das listas) usadas.