Guião 4

Laboratório de Algoritmia I Laboratórios de Informática II
Ano letivo 2020/21

Resumo

- Grande subida de dificuldade em relação aos guiões anteriores
- Testes muito exigentes
- Difícil ter 100% no MOOshak!
- Dificuldades:
  - Polimorfismo
  - Arrays
  - Gestão de memória

Arrays e strings

```
""      Criar uma string
[]      Criar um array
~       Colocar na stack todos os elementos do array
+       Concatenar strings ou arrays
        (ou array/string com elemento)
*       Concatenar várias vezes strings ou arrays
,       Tamanho ou range
=       Ir buscar um valor por índice
< >     Ir buscar X elems/carat do início ou fim
( )     Remover 1º ou últ. elt. e colocar na stack
        após o array/string
#       Procurar substring na string e devolver o índice
        Ou -1 se não encontrar
t       Ler todo o input => String
/       Separar string por substring => Array
S/      Separar uma string por whitespace => Array
N/      Separar uma string por newlines => Array
```

Exemplos

Input	Resultado
5 ,	01234
5 , ~ \	01243
[1 2 3] 2 * [4 5] \ +	45123123
[3 1 9]) 7 * + 3 *	316331633163
[7 1 4] (	147
[7 1 4] (+ _ , S \	147 3
[2 3 * 1 5.0 / 98 c 5 2 \ #]	60.2b32
t N/ ~ #	3
planetas	
neta	
“planetas” 3 >	tas
[[2 3 4] [5 6] \ + _ ,]	562345

Exemplos

Input	Resultado
[7 2 3] ,	3
“abc” 3 * _ S \ ,	abcabcabc 9
1 [2 3] + 3 *	123123123
[3 5 7 1 2] 2 =	7
[1 2 3] [4 5] \ +	45123
[7 2 9] (	297
5 , 3 >	234
[1 2 3] (+ [7 5] +	23175
[1 2 3] ~ * +	7
“olaqqabcqqxyz” “qq” / ,	3
t S/ ,	5
tres tristes tigres	
barao vermelho	

Sugestões

- Proibido usar variáveis globais
- Sempre que possível, usar funções em vez de macros
- Escrever funções pequenas e genéricas
- Cada função deve tentar resolver um único problema
- Evitar funções grandes que tentam atacar muitas frentes ao mesmo tempo
- Evitar muitos ciclos e/ou condições aninhados
- Documentar todo o código!

Sugestões: Parser

- Remover o strtok
- Criar uma função que separa o token do resto do input
- Devolve o token e o resto da linha

```
char *get_token(char *line, char **rest);
```

line A linha que recebe

rest Serve para devolver o resto da linha

Sugestões: parser de arrays e strings

- Recebe uma linha que começa com [ou "
- Devolve a parte da linha que contém o array ou string que está no início da linha
- É preciso cuidado nos arrays porque podem estar aninhados!

```
char *get_delimited(char *line, char *seps, char **rest);
```

line	A linha que recebe
seps	Os separadores: a string `"\""` ou `"[]"`
rest	Serve para devolver o resto da linha

Sugestões: avaliador

- Recebe a linha e um apontador para a estrutura (e.g., stack)
- Se a estrutura for NULL, cria uma nova
- Senão usa a que recebeu
- Devolve um apontador para a stack

```
STACK *eval(char *line, STACK *init_stack);
```

line	a linha para avaliar
init_stack	stack inicial, pode ser NULL

Sugestões: função eval

- Receber a linha e a stack inicial
- Se não existe stack inicial $=>$ cria
- Enquanto houver linha
  - Pegar no token
  - Tratar números
  - Tratar arrays
  - Tratar strings
  - Tratar operações
  - Linha passa a ser o resto
- Devolve stack atual

Sugestões: arrays

Se o token atual começar com um [

1. Usar a função `get_delimited` para receber a parte da linha que contém o array e o resto da linha
2. Usar a função `eval` para tratar do array (passar a stack vazia)
3. Pegar na stack que é devolvida e colocar na stack atual como um ARRAY

Sugestões: operadores

- Há operadores que funcionam de forma diferente conforme o que está na stack
- Optar por funções mais genéricas conforme o que está no topo da stack
  - `handle_arithmetic`
  - `handle_logic`
  - `handle_array`
  - `handle_string`