IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

ATAD 2019/20

Algoritmos e Tipos Abstratos de Dados

Enunciado de Laboratório

AVALIADO

8

Calculadora Polaca, ADT Stack e ADT Queue

Objetivos:

- Aumentar competências em algoritmia.
- Implementação de uma calculadora Polaca com ADT Stack e ADT Queue.

Notas importantes:

- Deverá garantir que possui o **valgrind** instalado e incluir o *bash script* na pasta do projeto com o nome `mem_check.sh` (ver laboratórios anteriores).
- Deverá manter a possibilidade de compilar os diferentes níveis no ficheiro *Makefile* – *os que não solicitarem um programa são dispensáveis, como é óbvio.*
- No template de submissão do relatório deverá também colocar o **output da execução do *valgrind*** em cada nível (quando aplicável). **Garanta que toda a memória é libertada na execução de cada programa. O nível só será cotado na totalidade se isto acontecer.**
- Deverá gerar a documentação *doxygen* para todos os níveis (quando aplicável).

Nível 1

1. A Notação Polaca (também conhecida como notação ***prefixa***) é uma forma de notação, usada para lógica, aritmética e álgebra, introduzida em 1920 pelo matemático polaco Jan Lukasiewicz. Com esta notação torna-se desnecessário o uso de parêntesis nas expressões aritméticas garantindo, ao mesmo tempo, o cálculo correto das expressões através da escrita sistemática dos operadores **antes** dos operandos e não no meio deles, como de costume.

Exemplos:

Operação	Notação Convencional (infixa)	Notação Polaca (prefixa)
$a+b$	a + b	+ a b
$\frac{a+b}{c}$	(a + b) / c	/ + a b c
$(a-b)\times c$	(a - b) * c	* - a b c
$\frac{(a+b)\times c}{(d-e)}$	(a + b) * c / (d – e)	/ * + a b c – d e

2. Na pasta de trabalho relativa a este laboratório crie o ficheiro *prefix_notation.txt* e escreva em notação polaca as seguintes expressões aritméticas:

```
i)    5+7×9−4
```

ii) $(5+7)\times(9-4)$
iii) $5\times(6+3)/3$
iv) $(3+7)-6/2$

Nível 2

3. Em meados de 1950, o cientista da computação australiano Charles L. Hamblin propôs a Notação Polaca Inversa (também conhecida por notação ***posfixa*** ou ***RPN*** do inglês Reverse Polish Notation). Esta notação, ao invés da anterior, coloca os operadores **a seguir** aos operandos e é muito usada devido, entre outras coisas, à sua fácil implementação num sistema usando uma pilha. Esta notação ganhou inclusive uma grande notoriedade por ter sido adotada pelas calculadoras HP[1].

Pegando nos exemplos anteriores:

Operação	Notação Convencional (infixa)	Notação Polaca (prefixa)	Notação Polaca Inversa (posfixa)
$a+b$	a + b	+ a b	a b +
$\frac{a+b}{c}$	(a + b) / c	/ + a b c	a b + c /
$(a-b)\times c$	(a - b) * c	* - a b c	a b – c *
$\frac{(a+b)\times c}{(d-e)}$	(a + b) * c / (d – e)	/ * + a b c – d e	a b + c * d e - /

Nota: Como se pode constatar na tabela de cima, a notação posfixa **NÃO é** a notação prefixa de trás para a frente!

4. Crie o ficheiro posfix_notation.txt e escreva as mesmas expressões aritméticas solicitadas no Nível 1 em notação posfixa.

Nível 3

5. Crie um ficheiro main3.c para este nível.

6. Crie uma função, `evaluatePostfixExpression`, que calcula o resultado (devolvido por referência) de uma expressão posfixa (ou RPN) bem formada. Assuma quatro operadores (**+**, **-**, ***** e **/** – divisão inteira) e que os operandos têm apenas um dígito.

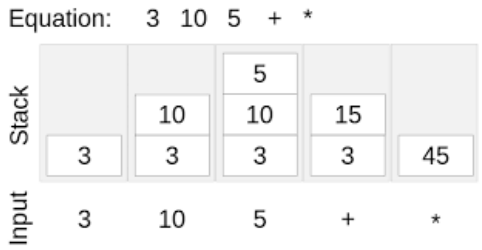
   - A função **recebe uma string** contendo a expressão;
   - A função **devolve por referência o resultado** calculado;
   - A função **devolve um booleano**: *true* se a expressão é válida e foi calculada com sucesso; *false* se a expressão é inválida.

Para tal utilizaremos o ADT Stack e uma implementação qualquer disponível e o seguinte algoritmo:

[1] https://www.hpmuseum.org/rpn.htm

```
1. Criar uma pilha.
2. Ler a expressão aritmética da esquerda para a direita lendo cada
   caractere:
      a. Se o caractere for um número inteiro, inseri-lo na pilha.
      b. Se for um operador, retirar dois valores da pilha, aplicar esse
         operador, e inserir o resultado na pilha.
3. No final, fica apenas um valor na pilha, que é o resultado da expressão.
4. Se a pilha não corresponder a nenhuma das situações anteriores (por
   exemplo ficar vazia), então a expressão é incorreta.
```

Exemplo do algoritmo ilustrado:



https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_algorithms/Reverse_polish

7. Teste a função no ficheiro main3.c, invocando-a na função main(). **Teste todas as expressões escritas no nível 2.**

Exemplo:

```
Insert a posfix (RPN) expression > 532*+
Stack contents (top to bottom):
5
--- bottom ---

Stack contents (top to bottom):
3
5
--- bottom ---

Stack contents (top to bottom):
2
3
5
--- bottom ---

Stack contents (top to bottom):
6
5
--- bottom ---

Stack contents (top to bottom):
11
--- bottom ---

The result is: 11
```

Nota: o conteúdo da stack está a ser imprimido numa perspetiva de visualização da execução do algoritmo – e também útil para a validação do trabalho, dentro da função `evaluatePostfixExpression`.

Nível 4

8. Crie agora uma função, `convertInfixToPostfix`, que converte uma expressão de entrada dada na notação tradicional, i.e., infixa para a notação posfixa e que devolva *true* caso a operação tenha decorrido com sucesso e *false* caso contrário.
   - A função **recebe uma string** contendo a expressão infixa;
   - A função **devolve por referência uma string** contendo a expressão posfixa;
   - A função **devolve um booleano**: *true* se a expressão é válida e foi calculada com sucesso; *false* se a expressão é inválida.

   Desta vez recorreremos à utilização do ADT Stack e ADT Queue para implementar esta função. Inspire-se no algoritmo seguinte:

```
Sendo I a expressão infixa de entrada e P a expressão posfixa:

1. Inserir um parêntesis esquerdo '(' na fila FI.
2. Inserir cada carater de I na fila FI.
3. Inserir um parêntesis direito ')' na fila FI.
4. Remover elemento da fila FI e repetir os passos 5 a 8 enquanto FI não
   estiver vazia.
5. Se o elemento encontrado for um operando/número, adicioná-lo a uma fila
   FP.
6. Se o elemento for um parêntesis esquerdo '(', inseri-lo na pilha.
7. Se se encontrar um operador então:
    a. Retirar os elementos da pilha e adicioná-los à fila FP repetidamente
       enquanto estes tiverem a mesma ou precedência superior ao elemento
       encontrado.
    b. Inserir o operador encontrado na pilha.
8. Se se encontrar um parêntesis direito ')' então:
    a. Retirar os elementos da pilha e adicioná-los à fila FP repetidamente
       até se encontrar um parêntesis esquerdo '('.
    b. Remover o parêntesis esquerdo da pilha. Não o adicione à fila FP.
9. Remova os elementos da fila FP para a expressão P.
```

Exemplo do algoritmo ilustrado:

Conversão Infix para Postfix **5 + (3 * 2)**		
Queue FI	**stack**	**Queue FP**
(5+(3*2))	empty	empty
5+(3*2))	(	empty
+(3*2))	(	5
(3*2))	(+	5
3*2))	(+(	5

*2))	(+(	53
2))	(+(*	53
))	(+(*	532
)	(+	532*
empty	empty	532*+

10. Teste a função no ficheiro main4.c, usando as expressões do Nível 2 e compare os resultados.

Nível 5

11. Crie um programa, num ficheiro main5.c, que represente a ***Calculadora Polaca***, que converte expressões dadas na notação infixa para a notação posfixa apresentando no final o resultado da expressão. Para tal terá de usar as funções acima implementadas.

Exemplo:

```
********** POLISH CALCULATOR **********

Insert an infix expression (single-digit only) > 5+(3*2)
The RPN expression is > 532*+

Estado da Stack:
---- topo ----
5
---- base ----

Estado da Stack:
---- topo ----
3
5
---- base ----

Estado da Stack:
---- topo ----
2
3
5
---- base ----

Estado da Stack:
---- topo ----
6
5
---- base ----

Estado da Stack:
---- topo ----
11
---- base ----

The result is: 11
More calculations (Y/N)?
```

12. Na documentação doxygen acrescente a informação sobre a complexidade algorítmica de cada função desenvolvida (exceptua-se o main).

13. **(Bónus)** Pense agora no que teria de alterar para se poderem introduzir expressões na notação infixa com operandos inteiros com mais de um dígito, converter para expressões na notação posfixa com os operandos e os operadores separados por espaços em branco, e devolvendo o resultado. Por exemplo:

```
Expressão infixa:      ( 55 + 33 ) × 22 / ( 66 − 44 )
Expressão posfixa:     55 33 + 22 × 66 44 − /
Resultado:             88
```

Sugestão: Pode usar a função *split* introduzida no LAB 7.

Consegue arranjar uma implementação de calculadora?!

(fim de enunciado)