Exercício : 1 _____ / _____ / _____

3 funções recursivas

Máximo Divisor Comum

Este algoritmo recursivo é devido a Euclides

1: inteiro MDC(inteiro a, b) {obrigatoriamente: $a \geq b$}
2: **se** $b = 0$ **então**
3: devolva a
4: **senão**
5: devolva MDC$(b, (a \ mod \ b))$
6: **fim se**
7: fim {função}

Suponha que far-se-á a chamada ao algoritmo acima, como segue:

```
MDC(1200, 1119)

O que aconteceria ?
Chamando com 1200 1119
Chamando com 1119 81
Chamando com 81 66
Chamando com 66 15
Chamando com 15 6
Chamando com 6 3
Chamando com 3 0
Resposta é 3
```

Potência recursiva

Seja o seguinte algoritmo que calcula x^y

1: inteiro função POTE (inteiro x, y)
2: inteiro t
3: **se** y $= 0$ **então**
4: devolva 1
5: **senão**
6: t $\leftarrow$ POTE $(x, \lfloor (y \div 2))$
7: **se** $0 = (y \ mod \ 2)$ **então**
8: devolva $t \times t$
9: **senão**
10: devolva $x \times t \times t$
11: **fim se**
12: **fim se**
13: end {função}

Suponha que far-se-á a chamada ao algoritmo acima, como segue:

```
POTE(2, 10)

O que aconteceria ?
Entrando com 2 10
Entrando com 2 5
Entrando com 2 2
Entrando com 2 1
Entrando com 2 0
A resposta é 1024
```

A pergunta que cabe aqui, é qual a vantagem do algoritmo recursivo acima sobre o algoritmo convencional de potência:

1: inteiro função POTECONV (inteiro x, y)
2: inteiro RESP, I
3: RESP $\leftarrow$ 1
4: **para** I de 1 até Y **faça**
5: RESP $\leftarrow$ RESP $\times$ x
6: **fim para**
7: devolva RESP
8: fim {função}

Perceba que a função POTÊNCIA RECURSIVA vai dividindo sucessivamente o expoente por 2. Esse comportamento caracteriza a curva da função logarítmica, pelo qual pode-se dizer que a complexidade dessa função é O( ).

Já a função POTÊNCIA CONVENCIONAL executa um laço de programação que tem exatamente o tamanho do expoente. Por isso, pode-se afirmar que a complexidade da função é O(_____).

Um pouco de LISP

Não se pode deixar de lado o LISP quando se fala em recursividade. LISP respira recursividade. Para interpretar a função abaixo, são necessárias algumas noções da linguagem.

defun define uma função do usuário. Contém os seguintes parâmetros: defun, o nome da função, a lista de parâmetros e várias listas de comandos.

cond contem pares de (condição)(ação). Se a condição é *t* executa-se a ação e desvia-se para após o cond. Se ela é *nil* (falsa), desvia-se para a próxima condição.

null pergunta-se se a lista é nula (vazia)

eql retorna *t* se há igualdade e *nil* senão

car(lista) devolve o primeiro elemento da lista

cdr(lista) devolve a lista que sobra quando é excluído o *car* da lista

+ como todas as funções lisp, aplica o *car* da lista sobre o *cdr* da lista. Então `(+ 1 2 3) é 6`.

format comando de impressão. Exige um *t*, seguido de um string de controle e depois os parâmetros a listar. No string, tem-se `S`: um objeto LISP e `%`: CR+LF

A função que deve ser estudada é

```
(defun quizumba (elemento lista)
  (format t "~S ~%" lista)
    (cond
      ((null lista) 9999)
      ((eql elemento (car lista)) 1)
      (t (+ 1 (quizumba elemento (cdr lista))))
    )
)
```

Note que cada vez que a função é chamada recursivamente, ela imprime a lista. O que este exercício pede, é que você informe o que é impresso na TERCEIRA chamada da função quizumba.

A pergunta é: "o que faz a função quizumba" ? A resposta é uma função que recebe um elemento e uma lista e devolve a posição do elemento na lista. Note que se o elemento citado não pertence à lista, a resposta é $\geq$ 9999.

Se quiser mais detalhes de LISP, consulte o livro LIVROIA, de Pedro Kantek, disponível na Intranet da UP. Se quiser baixar o interpretador XLISP para brincar (ou resolver este exercício), vá em http://almy.us/xlisp.html. Trata-se de um programa simples, que não exige instalação, nem altera o Registro, basta copiá-lo.

Para interpretar funções LISP, abra um editor ASCII qualquer, de preferência que faça "pareamento"de parênteses – uma sugestão é o scintilla (www.scintilla.org) e digite a função acima. Salve-a com um nome qualquer, com a terminação LSP, no mesmo diretório onde guardou o programa XLWIN32.EXE.

Feito isto, já com XLISP carregado, execute `(load 'nome)`, onde `nome` é o nome do arquivo LSP que você deu quando o salvou acima.

Finalmente, execute a função definida ou outra qualquer escrevendo `(quizumba 'a '(1 2 3 a b c d))`. Note o correto pareamento de parênteses. Não esqueça que LISP pode ser entendido como *Lot of Idiot and Stupid Parenthesis*.

☞ Para você fazer

Algoritmo máximo divisor comum Chame o algoritmo mdc com os valores

```
1992 1452
```

Responda no quadro próprio, qual o valor da dupla a,b na terceira chamada à função MDC Obs: No exemplo dado acima, seriam os valores 81,66

Algoritmo potência recursiva Chame o algoritmo com a dupla de valores

```
6   6
```

Responda no quadro próprio, qual o valor da dupla x,y na terceira chamada à função POTE Obs: No exemplo dado acima, seriam os valores 2,2

Função quizumba

```
(quizumba '2 '(5 6 7 8 9))
```

Responda no quadro próprio, qual o valor impresso na TERCEIRA chamada à função.