Estruturas de Dados: MC202

Exercícios

1. A tabela abaixo apresenta três trechos de programas e suas respectivas análises de eficiência, simples e detalhada. Estas análises consideram uma unidade de tempo para qualquer tipo de operação. Verifique estes resultados e justifique os valores correspondentes.

	...
	x = a + b;
	...
		...
		for(i=1; $i \leq n$; i++)
		x = a + b;
		...
			...
			for(i=1; $i \leq n$; i++)
			for(j=1; $j \leq n$; j++)
			x = a + b;
			...

análise simples	1	n	n^2
análise detalhada	2	$5n + 2$	$5n^2 + 5n + 2$

2. Determine quantas vezes é executado o comando de atribuição $x = x + 1$;, no trecho do programa abaixo, em função de n:

```
for (i=1; i<=n; i++)
 for (j=1; j<=i; j++)
   for(k=1; k<=j; k++)
     x = x + 1 ;
```

3. Uma matriz quadrada a é dita diagonal se para todos os valores distintos de i e j tem-se $a[i, j] = 0$. Sugira uma maneira mais eficiente de implementá-la do que a usual.

4. Implemente um tipo abstrato de dados *Polinômio* baseado em listas ligadas que efetue as operações de adição, subtração, multiplicação e divisão de polinômios de grau $n \geq 0$ denotados por:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x^1 + a_0 x^0,$$

em que $a_n \neq 0$, exceto possivelmente se $n = 0$. Neste caso, são lidos os coeficientes e expoentes de dois polinômios a serem processados (veja o exemplo de soma de polinômios apresentado em sala de aula).

5. Quais as vantagens e desvantagens de se representar um grupo de elementos com um vetor versus uma lista ligada linear?

6. Escreva um programa para executar cada uma das seguintes operações:

   (a) Incluir um elemento no final de uma lista.

   (b) Concatenar duas listas.

   (c) Liberar todos os nós de uma lista.

   (d) Inverter uma lista de modo que o primeiro se torne o último, e assim por diante.

   (e) Eliminar o último elemento de uma lista.

   (f) Eliminar o n-ésimo elemento de uma lista.

   (g) Combinar duas listas ordenadas numa única lista.

   (h) Formar uma lista contendo a união dos elementos de duas listas.

   (i) Formar uma lista contendo a interseção dos elementos de duas listas.

(j) Inserir um elemento após o n-ésimo elemento de uma lista.

(k) Eliminar cada segundo elemento de uma lista.

(l) Colocar os elementos de uma lista em ordem crescente.

(m) Retornar a soma dos inteiros numa lista.

(n) Retornar o número de elementos numa lista.

(o) Deslocar um nó dado p n posições numa lista.

(p) Criar uma segunda cópia de uma lista.

7. Escreva programas para executar cada uma das operações acima sobre elementos armazenados num vetor.

8. Escreva programas para executar cada uma das operações do item 6 supondo que cada lista contenha um nó cabeça com o número de elementos da lista.

9. Escreva programas para executar cada uma das operações do item 6 considerando listas circulares. Quais são mais eficientes com listas circulares do que com listas lineares? Quais são menos eficientes?

10. Escreva uma função que receba uma lista encadeada e retorne o endereço da célula que esteja o mais próximo possível do ponto médio da lista. Isto deve ser feito sem calcular o tamanho n da lista.

11. Aplique às expressões infixas abaixo o algoritmo de conversão para notação posfixa:

```
A*B+C*D^E/F=-G*H
(A+B)*D+E/(F+A*D)+C
(A*(B+(C*(D+(E*(F+G))))))
```

12. A função a seguir traduz uma expressão da sua notação infixa para posfixa. A função considera que a expressão de entrada (string $infix$) está correta e contém apenas letras (uma por operando), parênteses e os símbolos $+$, $-$, $*$ e $/$. Além disto, a função toda começa com um "abre parêntese"e termina por um "fecha parêntese" seguido de $'\backslash 0'$.

    Exemplo de expressões de entrada são: (A) ; $(A + B * C)$; $(A * (B + C)/D - E)$ etc

```
/* recebe uma expressao infixa e devolve a correspondente posfixa */

char *InfixaParaPosfixa (char infix[]) {
    char *posfix, x ;
    char *p; int t ;
    int n, i, j ;
    n = strlen(infix) ;
    posfix = malloc(n*sizeof(char)) ;
    p = malloc(n*sizeof(char)) ;
    t = 0; p[t++] = infix[0] ;          /* empilha '(' */

    for (j=0 , i=1 ; infix[i] != '\0' ; i++) {
        /* p[0..t-1] eh uma pilha de caracteres */
        switch (infix[i]) {
            case '(' : p[t++] = infix[i] ;       /* empilha */
                       break ;
            case ')' : while (1) {                /* desempilha */
                           x = p[--t] ;
                           if ( x == '( ')  break ;
                           posfix[j++] = x ;  }
                       break ;
            case '+' :
            case '-' : while (1) {
                           x = p[t-1] ;
                           if (x == '(') break ;
                           --t ;      /* desempilha */
                           posfix[j++] = x ;  }
                       p[t++] = infix[i] ;     /* empilha */
                       break ;
            case '*' :
            case '/' : while (1) {
                           x = p[t-1] ;
                           if (x == '('  || x == '+' || x == '-')
                               break ;
                           --t ;
                           posfix[j++] = x ;  }
                       p[t++] = infix[i] ;
                       break ;
           default : posfix[j++] = infix[i] ;  }
    }
    free (p) ;
    posfix[j] = '\0' ;
    return posfix ;
}
```

(a) Aplique à expressão infixa $(A + B) * D + E/(F + A * D) + C$ o algoritmo de conversão para notação posfixa.

(b) Na função $InfixaParaPosFixa$, suponha que a string $infix$ tem n caracteres (sem contar o caractere nulo final). Que altura a pilha pode atingir, no pior caso (valor máximo da variável t)?

(c) Reescreva a função $InfixaParaPosFixa$ sem supor que a expressão infixa está incluída entre um par de parênteses.

(d) Reescreva a função $InfixaParaPosFixa$ supondo que a expressão infixa pode estar incorreta.

(e) Reescreva a função $InfixaParaPosFixa$ supondo que a expressão pode ter parênteses e chaves.

13. Implemente uma pilha com lista encadeada sem nó-cabeça. A pilha será especificada pelo endereço do primeiro nó da lista.

14. Defina uma pilha em termos de um TAD fila.

15. Defina uma fila em termos de um TAD pilha.

16. Implemente uma fila com uma lista ligada circular com nó-cabeça. O primeiro elemento da fila ficará no segundo nó e o último elemento ficará no nó anterior ao nó-cabeça. Para manipular a fila basta conhecer o endereço p do nó-cabeça.

17. Implemente uma fila com uma lista duplamente ligada sem nó-cabeça. Mantenha um ponteiro para o primeiro nó e um ponteiro para o último.

18. Árvores binárias são uma generalização de listas ligadas. Justifique.

19. Dada a árvore binária da figura abaixo, proponha uma forma de representação da mesma usando uma estrutura de dados matricial.