unesp

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Bauru

Faculdade de Ciências

Revisão

Disciplina: Estruturas de Dados I – **ED1**
Professora: Dra Simone das Graças Domingues Prado

Nome:_____ **RA:** _____

(ALOCAÇÃO ESTÁTICA) **1)** Observe o programa classificador, em pseudocódigo, apresentado abaixo.

```
1.  inicio
2.      variavel texto nome[5], naux
3.      variavel real nota[5], aux
4.      variavel inteiro i, j
5.      para i de 1 ate 5
6.          escrever "Nome ", i, " = "      ler nome[i-1]
7.          escrever "Nota ", i, " = "      ler nota[i-1]
8.      proximo
9.      para i de 0 ate 4
10.         para j de i+1 ate 4
11.             se nota[i] <= nota[j] entao
12.                 aux <- nota[i]          nota[i] <- nota[j]
13.                 nota[j] <- aux          naux <- nome[i]
14.                 nome[i] <- nome[j]      nome[j] <- naux
15.             fimse
16.         proximo
17.     proximo
18.     para i de 1 ate 5
19.         escrever nome[i-1], ": ", nota[i-1], "\n"
20.     proximo
21. fim
```

(1,0pt) Esse programa faz a classificação de que forma?

(a) Ordenação decrescente por nota.
(b) Ordenação crescente por nota.
(c) Ordenação decrescente por nome.
(d) Ordenação crescente por nome.

(ALTERNATIVA CORRETA: A)

(ALOCAÇÃO ESTÁTICA) **2) (1,0pt)** Considere que um professor queira saber se existem alunos cursando, ao mesmo tempo, as disciplinas A e B, e para isso tenha implementado um programa que:

I. Inicializa um vetor A de 30 (na) posições que contém as matriculas dos alunos da disciplina A;
II. Inicializa um vetor B de 40 (nb) posições que contém as matriculas dos alunos da disciplina B;
III. Imprime a matrícula dos alunos que estão cursando as disciplinas A e B ao mesmo tempo.

Considere, ainda, que os vetores foram declarados e inicializados, não estão necessariamente ordenados, e seus índices variam entre 0 e n-1, sendo n o tamanho do vetor.

```
for (i = 0; i < na; i++) {
    for (j = 0; j < nb; j++){
        /* comandos.... */     }}
```

Com base nessas informações, conclui-se que o trecho a ser incluído no código acima, para que o programa funcione corretamente, será?

(a) if (a[i] == b[j]) printf (a[i]); (CORRETA)
(b) if (a[j] == b[i]) printf (a[j]);
(c) if (a[i] == b[j]) printf (a[j]);
(d) if (a[i] == b[i]) printf (a[i]);
(e) if (a[j] == b[j]) printf (a[j]);

(ALOCAÇÃO ESTÁTICA) **3) (1,0pt)** O algoritmo a seguir recebe um vetor V de números inteiros e rearranja esse vetor de tal forma que seus elementos, ao final, estejam ordenados de forma crescente.

```
void ordena(int *V, int n){
        int i, j, chave;
        for (i=1; i<n; i++){
                chave = v[i];
                j = i-1;
                while (j>=0 &&  v[j] > chave){
                        /* comando */
                        j= j-1; }
                v[j+1]=chave;  }}
```

(a) v[j+1] = v[j];
(b) v[j+1] = v[i];
(c) v[j-1] = v[j];
(d) v[i+1] = v[i];
(e) v[i-1] = v[i];

(ALTERNATIVA CORRETA: A)

(ALOCAÇÃO DINÂMICA) Um ponteiro é um elemento que proporciona maior controle sobre a memória do computador, principalmente por ser utilizado em conjunto com mecanismos de alocação dinâmica de memória. Dessa forma, o domínio sobre este tipo de dado é muito importante. O código, a seguir, foi escrito na linguagem C++ e trabalha com ponteiros e estruturas dinâmicas.

```
using namespace std;
struct No{  int Dado;  struct No* prox; };
int main(){
        struct No *L, *I; int n;
        scanf("%d",&n);
        if (n==0) L = NULL;
        else{     L = new No;       L->Dado = n--;
                  L->prox = NULL;
                  for ( ; n>0; ){
                            I = new No;    I -> Dado = n--;
                            I->prox = L;   L = I; }}
        while (L!=NULL){
                  printf("%d ",L->Dado);   L = L->prox;
                 }
        return 0; }
```

4) (1,0pt) Qual será a saída do programa para n=6?

(a) 1 2 3 4 5 6
(b) 6 5 4 3 2 1
(c) 0 1 2 3 4 5
(d) 5 4 3 2 1 0

(ALTERNATIVA CORRETA = A)

5) (1,0pt) Como é a construção da lista, ou seja, a inserção dos elementos.

(a) A inserção do elemento é sempre feita no final da lista.
(b) A inserção do elemento é sempre feita no início da lista.
(c) A inserção do elemento não tem regra

(ALTERNATIVA CORRETA = B)

(ALOCAÇÃO DINÂMICA) **6) (1,0pt)** Uma lista ligada possui a seguinte definição de nó;

```
struct lista{
        int info;
        struct lista* prox;};
typedef struct lista* def_lista;
```

Como a rotina abaixo deve ser completada para inverter uma lista ligada?

```
void inverte (def_lista* h){
def_lista p, q;
if (*h != NULL){
        p = (*h)->prox;   (*h)->prox = NULL;
        while (p!=NULL){
                /* colocar os comandos dentro deste while */
}}}
```

(a) p->prox = *h; q = p->prox; *h=p; p=q;
(b) q = p->prox; *h = p; p = q; p->prox = *h;
(c) p->prox = *h; *h = p; p = q; q = p->prox;
(d) q = p -> prox; p -> prox = *h; *h = p; p = q; (ALTERNATIVA CORRETA)
(e) p->prox = *h; *h = p; q = p->prox; p = q;

(RECURSÃO) Os números de Fibonacci constituem uma sequencia de números na qual os dois primeiros elementos são 0 e 1 e os demais, a soma dos dois elementos imediatamente anteriores na sequencia. Abaixo, apresenta-se uma implementação em linguagem C para essa relação de recorrência.

```
long int fibonacci (int x){
        if (x<=1) return x;
        else return (fibonacci (x-1) +
                     fibonacci (x-2));}
```

7) (1,0pt) Considerando que o programa ao lado, calcule quantas chamadas a função fib são necessárias para computar **fibonacci(5), sem considerar a primeira chamada no main().**

(a) 10 (b) 12 **(c) 14** (d) 16 (e) 20

(ALTERNATIVA CORRETA: C)

(RECURSÃO) Considere a rotina abaixo.

```
int A (int m, int n){
        if (m==0) return n+1;
        else      if (n ==0) return A(m-1, 1);
                  else return A (m-1,A(m,n-1)):
}
```

8) (1,0pt) Indique quantas chamadas recursivas aconteceu (excluindo a primeira chamada da função) quando executado para A(1,2),

(a) 1 (d) 4
(b) 2 **(e) 5 (ALTERNATIVA CORRETA)**
(c) 3

(RECURSÃO) **9)** No modo recursivo de representação, a descrição de um conceito faz referência ao próprio conceito. Julgue os itens abaixo, com relação à recursividade como paradigma de programação. **Atribua F(falso) ou V (verdadeiro) para as afirmativas** a seguir.

(_V_) (0,25pt) São elementos fundamentais de uma definição recursiva: o caso-base (base da recursão) e a reaplicação da definição.

(_F_) (0,25pt) O uso da recursão não é possível em linguagens com estruturas para orientação a objetos

(_V_) (0,25pt) No que diz respeito ao poder computacional, as estruturas iterativas e recursivas são equivalentes.

(_F_) (0,25pt) Estruturas iterativas e recursivas não podem ser misturadas em um mesmo programa.

Considere o programa seguir, desenvolvido em linguagem C.

```
int F1 (int X, int Y){
        if (X<Y) return X;
        else return F1(X-Y, Y);  }
int F2 (int X, int Y){
        if (X<Y) return 0;
        else return 1 + F2(X-Y, Y); }
int F3 (int X, int Y){
        if (X<Y) printf("%d",X);
        else{    F3(F2(X,Y),Y);
                 printf("%d", F1(X, Y)); }
```

```
int main(){
        int A, B;
        scanf("%d %d", &A, &B);
        if ((A>0) && (A<1000) && (B>1) && (B<10)){
                F3(A,B);
                printf("\n"); }
        return 0:
}
```

10) No programa apresentado, a técnica de recursividade foi aplicada às três funções F1, F2 e F3. Essa técnica envolve a definição de uma função ou rotina que pode invocar a si própria. Com relação ao programa apresentado e à técnica de recursão**, atribua F(falso) ou V (verdadeiro) para as afirmativas** a seguir.

(_V_) (0,2pt) A chamada da função F1, através da expressão F1(X,Y), pode ser substituída, sem alterar o resultado do programa, pela expressão X%Y.
JUST: o que F1() faz é divisão usando subtrações sucessivas.

(_F_) (0,2pt) o objetivo da função F2 é retornar o valor da variável X elevado à Y-ésima potencia.
JUST: a rotina F2() conta quantas subtrações são feitas até que X seja menor que Y

(_F_) (0,2pt) a chamada à função F3 entrará em uma recursão sem fim se o valor da variável X for maior que o valor da variável Y.
JUST: não entrará em recursão sem fim porque existe manipulação do valor X até que ele seja menor que Y e para.

(_F_) (0,2pt) a função main não é recursiva, pois na linguagem C não é possível implementar esta técnica na função principal do programa.
JUST: ***a rotina main() é uma rotina especial que pela qual será iniciada a execução do programa. Normalmente não é recursiva. Porém ela por ser uma rotina, ela pode ser construída usando o conceito de recursão. Não dará erro nem na compilação e nem na execução.***

(_V_) (0,2pt) a expressão ((A>0) && (A<1000) && (B>1) && (B<10)), da função main, pode ser substituída pela expressão (!((A<=0) || (A>=1000) || (B<=1) || (B>=10))), sem alterar o resultado do comando condicional if nesta expressão.
JUST: já que a segunda expressão é a negação da inversa da primeira.

Revisão

1	2	3	4	5	6	7	8	9				10				
A	A	A	A	B	D	C	E	V	F	V	F	V	F	F	F	V