MC102 - Algoritmos e Programação de Computadores

Segundo semestre de 2010 - Turmas C e D

1^a Prova - versão B

Nome:					
RA:		Data:	14/10/2010	Turma:	

Instruções:

- O tempo de duração da prova é de duas horas.
- A prova deve ser feita INDIVIDUALMENTE. A comunicação entre dois ou mais alunos durante a prova resultará em zero para todos os envolvidos.
- Nenhum material pode ser consultado.
- As questões podem ser respondidas a lápis.
- Os aparelhos eletrônicos de comunicação (e.g. celular) devem ser desligados antes de começar a prova.
- Quando for necessário escrever código-fonte, indente-o.
- O comando `goto` não é permitido em nenhum dos exercícios.
- **Boa prova!**

Questões

1.

a) (1,0 ponto) Crie um registro com as seguintes características:

- o primeiro campo é do tipo vetor de `char` de tamanho 70 e se chama `nome`;
- o segundo campo é do tipo `int` e se chama `ra`;

Redefina esse registro para criar um tipo chamado `Aluno`.

```
typedef struct _aluno{
    char nome[70];
    int ra;
} Aluno;
```

b) (2,0 pontos) O vetor `classe[]` é do tipo `Aluno`, como definido na questão anterior. Suponha que este vetor está ordenado pelo campo `ra`, do menor para o maior. Complete a função abaixo para que ela realize uma busca binária por um aluno procurado, que é um dos parâmetros da função. Se o aluno for achado, a função deve retornar a sua posição no vetor, caso contrário deve retornar -1 (menos um). Considere que não existam dois nomes ou dois RAs iguais.

```
int buscaBinaria(Aluno classe[], int tam_Classe, Aluno procurado){
  int achou = -1; /*o valor -1 significa que nao achou*/
  int esq = 0;
  int dir = tam_Classe -1;
  int meio = (esq + dir) / 2;
```

```
  while( ( achou == -1 ) && ( esq <= dir ) ){
    if(classe[meio].ra == procurado.ra)
      achou = meio;
    else{
      if(classe[meio].ra < procurado.ra)
        esq = meio + 1;
      else
        dir = meio -1;

      meio = (esq + dir) / 2;

    }
  }
}
```

c) (1,0 ponto) Suponha que o vetor `classe[]` do item anterior não esteja ordenado. Complete a função `buscaSequencial` de forma que ela implemente a busca sequencial por um aluno procurado. Se o aluno for achado, a função deve retornar a sua posição no vetor, caso contrário deve retornar -1 (menos um). Considere que não existam dois nomes ou dois RAs iguais.

```
int buscaSequencial(Aluno classe[], int tam_Classe, Aluno procurado){

  int achou = -1;
  int i = 0;

  while ( ( achou == -1) && ( i < tam_Classe )){
    if(classe[i].ra == procurado.ra)
      achou = i;
    else
      i++;
  }
  return achou;
}
```

2.(2,0 pontos) Suponha que os algoritmos de ordenação por bolha (*bubblesort*) e por inserção (*insertionsort*) sejam executados para ordenar os vetores de inteiros abaixo. Quais seriam os vetores intermediários, para cada um dos algoritmos, até que o vetores estivessem totalmente ordenados? Se houver vetores intermediários repetidos (i.e. inalterados), escreva-os quantas vezes forem necessárias.

Bubblesort					
5	**4**	**3**	**6**	**2**	**1**
4	3	5	2	1	6
3	4	2	1	5	6
3	2	1	4	5	6
2	1	3	4	5	6
1	2	3	4	5	6

Insertionsort					
5	**4**	**3**	**6**	**2**	**1**
4	5	3	6	2	1
3	4	5	6	2	1
3	4	5	6	2	1
2	3	4	5	6	1
1	2	3	4	5	6

3.(2,0 pontos) Um palíndrome pode ser uma sequência de inteiros que é idêntica quando lida da esquerda para a direita ou da direita para a esquerda. Veja a seguir dois exemplos de palíndromes: [1,2,3,3,2,1] e [1,2,3,4,3,2,1]. Complete a função `verificaPalindrome` que tem como parâmetros um vetor chamado `palindrome[]` do tipo `int` e um parâmetro `tam` que representa o tamanho do vetor `palindrome[]`. A função deve retornar 0 (zero) se esse vetor for um palíndrome e 1 (um) caso contrário.

```
int verificaPalindrome(int palindrome[], int tam){

  int i = 0;
  int isPalindrome = 0;

  while( (isPalindrome == 0) && ( i < (tam / 2) ) ){
    if( palindrome[i] != palindrome[ tam - 1 - i ]){
      isPalindrome = -1;
    }
    i++;
  }
  return isPalindrome;
}
```

4.(2,0 pontos) A matriz identidade é uma matriz quadrada que possui o elemento 1 repetido na diagonal principal[1] e os demais elementos são zeros. Veja abaixo um exemplo de uma matriz identidade 3 x 3:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Complete a função `verificaMatrizIdentidade` que tem como parâmetros a matriz `mat[][]` com `NCOLS` colunas (suponha que NCOLS é uma constante que foi definida no começo do programa) e o outro parâmetro é `nlins` que representa o número de linhas da matriz `mat[][]`. Se a matriz `mat[][]` for a matriz identidade, a função deve retornar 0 (zero), caso contrário retorna 1 (um).

```
int verificaMatrizIdentidade(int mat[][NCOLS],int nlins){
  int i , j;
  int identidade = 0;

  i = 0;
  while( ( identidade == 0 ) && ( i < nlins ) ){
    j = 0;
    while( ( j < NCOLS ) && ( identidade == 0 ) ){
      if( ( ( j == i ) && ( matriz[i][j] != 1 )) ||( ( i != j ) && ( matriz[i][j] != 0 ) )){
        identidade = -1;
      }
      j++;
    }
    i++;
  }
  return identidade;


}
```

[1] a diagonal principal é a diagonal da matriz quando o número da linha é igual ao número da coluna