MAC2166 – Introdução à Computação para Engenharia

Escola Politécnica

Terceira Prova – 21 de junho de 2010

Nome: _____

Assinatura: _____

Nº USP: _____ Turma: _____

Professor: _____

Instruções:

1. Não destaque as folhas deste caderno.
2. A prova consta de 4 questões. Verifique antes de começar a prova se o seu caderno de questões está completo.
3. A prova pode ser feita a lápis. Cuidado com a legibilidade.
4. Qualquer questão pode ser resolvida em qualquer página. Se a questão não está na página correspondente ao enunciado basta indicar isto na página e escrever QUESTÃO X em letras ENORMES antes da solução.
5. Não é necessário apagar rascunhos no caderno de questões.
6. Não é permitido o uso de folhas avulsas para rascunho.
7. Não é permitido o uso de calculadoras.
8. Não é permitido a consulta a livros, apontamentos ou colegas.

DURAÇÃO DA PROVA: 2 horas

Questão	Valor	Nota
1	2.5	
2	2.5	
3	2.5	
4	2.5	
Total	10.0	

Questão 1

Simule a execução do programa abaixo, destacando a sua saída. A saída do programa consiste de tudo que resulta dos comandos `printf`. **Para efeito de correção só será considerada a saída do programa.**

```
#include <stdio.h>

int f0(int nusp, int V[]) {
 int n=0 ;
 while(nusp!=0) {
   V[n] = nusp%10 ;
   n++;
   nusp=nusp/10 ;
 }
 V[n/2] = V[n/2] + 10;
 V[n/2 + 1] = V[n/2 + 1] - 10;
 return n;
}

void f1(int V[], int i1, int *i2) {
  int j, temp;
  for (j = i1 + 1; j <= *i2; j++){
     if (V[j-1] > V[j]){
        temp = V[j-1];
        V[j-1] = V[j];
        V[j] = temp;
      }
  }
  *i2 = *i2 - 1;
}

void f2(int V[], int *i1, int i2) {
  int j, temp;
  for (j = i2 - 1; j >= *i1; j--){
     if (V[j+1] < V[j]){
        temp = V[j+1];
        V[j+1] = V[j];
        V[j] = temp;
      }
  }
  *i1 = *i1 + 1;
}

void f3(int V[], int *i1, int i2) {
  int j;
  int min, temp;
  min = *i1;
  for (j = *i1 + 1; j <= i2; j++){
    if (V[j] < V[min]) min = j;
  }
  temp = V[*i1];
  V[*i1] = V[min];
  V[min] = temp;
  *i1 = *i1 + 1;
}
```

```
int main() {

  int nusp, i, ind1, ind2, tam, V[7] ;


  printf("Digite o seu NUSP: ") ;
  scanf("%d", &nusp) ;

  tam = f0(nusp, V) ;
  printf("nusp= %d \n", nusp) ;


  printf("f0: ") ;
  for(i=0; i<tam; i++) printf("%d ", V[i]) ;
  ind1 = 0;
  ind2 = tam -1;
  printf(" imin= %d, imax=%d\n\n",ind1,ind2);


  f1(V,ind1,&ind2) ;
  printf("f1: ") ;
  for(i=0; i<tam; i++) printf("%d ", V[i]) ;
  printf(" imin= %d, imax=%d\n\n",ind1,ind2);


  f2(V,&ind1,ind2) ;
  printf("f2: ") ;
  for(i=0; i<tam; i++) printf("%d ", V[i]) ;
  printf(" imin= %d, imax=%d\n\n",ind1,ind2);


  f3(V,&ind1,ind2) ;
  printf("f3: ") ;
  for(i=0; i<tam; i++) printf("%d ", V[i]) ;
  printf(" imin= %d, imax=%d\n\n",ind1,ind2);

  return 0;


}
```

Dados para a simulação: **o seu número USP**

saída
`Digite o seu Nusp:`

Questão 2

Escreva as seguintes funções

(a) (0.5 ponto) `float media(int n, float x[]);`

Recebe um inteiro `n` e um vetor `x` com `n` números reais e retorna sua média.

(b) (0.5 ponto) `float stddev(int n, float x[]);`

Recebe um inteiro `n` e um vetor `x` com `n` números reais e retorna o desvio padrão. Para isso use a função do item (a) mesmo que não a tenha implementado. Lembre que o desvio padrão dos elementos de um vetor x de dimensão n é definido por

$$\text{stddev} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}},$$

em que $\bar{x}$ representa a média dos elementos de x.

Obs: você pode usar a função `sqrt` do `math.h`

(c) (1.5 ponto) `float mediana(int n, float x[]);`

Recebe um inteiro `n` e um vetor `x` com `n` números reais e retorna a mediana de `x`. Se `n` é ímpar, a mediana é um elemento do vetor `x` que é maior do que os $(\texttt{n}-1)/2$ menores elementos do vetor e menor do que os $(\texttt{n}-1)/2$ maiores. Se n é par, a mediana é um elemento do vetor `x` que é maior do que os $\texttt{n}/2$ menores elementos do vetor e menor do que os $\texttt{n}/2-1$ maiores. Por exemplo, a mediana de $1, 1, 5, 4, 7$ é 4. Já a de $8, 1, 4, 5, 5, 3$ é o 5. Ou seja, a mediana é o elemento de `x` que divide os seus valores "ao meio".

Questão 3

Considere o seguinte problema: *Dados os comprimentos e duas sequências de inteiros que representam conjuntos, calcule o conjunto união dos conjuntos.* Por exemplo para os conjuntos $\{1, 2, 3\}$ e $\{3, 4\}$ um exemplo da execução de um programa que resolve o problema é:

```
Digite o tamanho do conjunto: 3
Digite seus elementos: 1 2 3
Digite o tamanho do conjunto: 2
Digite seus elementos: 3 4
Conjunto uniao: 1 2 3 4
```

Para resolver o problema, cada conjunto é armazenado em um vetor e quatro funções auxiliares são criadas:

(a) `busca`: recebe o comprimento de um vetor `n`, o vetor `v` e um inteiro `x`, e retorna 1 se `x` está em `v` e 0 caso constrário;

(b) `le_conjunto`: lê um conjunto do teclado;

(c) `imprime_conjunto`: imprime um conjunto na tela; e

(d) `une`: recebe o comprimento `int na` e o conjunto A (`int A[MAX]`), o comprimento `int nb` e o conjunto B (`int B[MAX]`), e retorna o inteiro `nuniao` e o vetor `Uniao` representando o comprimento e conjunto união, respectivamente.

Desta forma, o programa principal pode ser construído apenas por chamadas a essas funções. As funções `busca`, `le_conjunto`, `imprime_conjunto` e o programa já foram implementos como segue:

```
#include <stdio.h>
#define MAX 100

int busca(int n, int v[], int x) {
    int i;
    for (i = 0; i < n && v[i] != x; i++);
    if (i < n) return 1;
    else return 0;
}

void le_conjunto(int *n, int A[]) {
    int i;
    printf("Digite o tamanho do conjunto: ");
    scanf("%d", n);
    printf("Digite seus elementos:");
    for (i = 0; i < *n; i++)
        scanf("%d", &A[i]);
}

void imprime_conjunto(int n, int A[]) {
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", A[i]);
    printf("\n");
}
void une(int na, int A[], int nb, int B[], int *nuniao, int Uniao[]);

int main () {

    int A[MAX], B[MAX], Uniao[MAX];
    int na, nb, nuniao;

    le_conjunto(&na, A);
    le_conjunto(&nb, B);
    une(na, A, nb, B, &nuniao, Uniao);
    printf("Conjunto uniao: ");
    imprime_conjunto(nuniao, Uniao);

    return 0;
}
```

Quatro possíveis implementações da função `une` estão apresentadas a seguir. Alguns testes foram executados para este problema. Para cada uma das saídas seguintes, assinale **UMA** implementação que poderia ter sido utilizada para produzir tal resultado. Note que mais de uma implementação poderá ter dado o mesmo resultado.

saída do programa	**assinale uma das possíveis implementacões**

`Digite o tamanho do conjunto: 3`
`Digite seus elementos: 3 2 -1`
`Digite o tamanho do conjunto: 2`
`Digite seus elementos: 3 4`
`Conjunto uniao: 2 -1 4`
	☐ imp1 ☐ imp2 ☐ imp3 ☐ imp4

`Digite o tamanho do conjunto: 3`
`Digite seus elementos: 3 2 -1`
`Digite o tamanho do conjunto: 2`
`Digite seus elementos: 3 4`
`Conjunto uniao: 3 2 -1 4`
	☐ imp1 ☐ imp2 ☐ imp3 ☐ imp4

`Digite o tamanho do conjunto: 3`
`Digite seus elementos: 3 2 -1`
`Digite o tamanho do conjunto: 2`
`Digite seus elementos: 3 4`
`Conjunto uniao: 3 3`
	☐ imp1 ☐ imp2 ☐ imp3 ☐ imp4

`Digite o tamanho do conjunto: 3`
`Digite seus elementos: 3 2 -1`
`Digite o tamanho do conjunto: 2`
`Digite seus elementos: 3 4`
`Conjunto uniao: 2621450 3 2 -1`
	☐ imp1 ☐ imp2 ☐ imp3 ☐ imp4

`Digite o tamanho do conjunto: 2`
`Digite seus elementos: 1 2`
`Digite o tamanho do conjunto: 2`
`Digite seus elementos: 3 4`
`Conjunto uniao: 1 2 3 4`
	☐ imp1 ☐ imp2 ☐ imp3 ☐ imp4

```
/* imp1 */
void une(int na, int A[], int nb, int B[], int *nuniao, int Uniao[]) {
    int i;

    *nuniao = 0;
    for (i = 0; i < na; i++) {
        *nuniao = *nuniao + 1;
        Uniao[*nuniao] = A[i];
    }
    for (i = 0; i < nb; i++) {
        if (busca(*nuniao, Uniao, B[i]) == 0) {
            *nuniao = *nuniao + 1;
            Uniao[*nuniao] = B[i];
        }
    }
}
```

```
/* imp2 */
void une(int na, int A[], int nb, int B[], int *nuniao, int Uniao[]) {
    int i;

    *nuniao = 0;
    for (i = 0; i < na; i++) {
        if (busca(nb, B, A[i]) == 0) {
            Uniao[*nuniao] = A[i];
            *nuniao = *nuniao + 1;
        }
    }
    for (i = 0; i < nb; i++) {
        if (busca(na, A, B[i]) == 0) {
            Uniao[*nuniao] = B[i];
            *nuniao = *nuniao + 1;
        }
    }
}
```

```
/* imp3 */
void une(int na, int A[], int nb, int B[], int *nuniao, int Uniao[]) {
    int i;

    *nuniao = 0;
    for (i = 0; i < na; i++) {
        if (busca(na, A, B[i]) == 1) {
            Uniao[*nuniao] = A[i];
            *nuniao = *nuniao + 1;
        }
    }
    for (i = 0; i < nb; i++) {
        if (busca(nb, B, A[i]) == 1) {
            Uniao[*nuniao] = B[i];
            *nuniao = *nuniao + 1;
        }
    }
}
```

```
/* imp4 */
void une(int na, int A[], int nb, int B[], int *nuniao, int Uniao[]) {
    int i;

    *nuniao = 0;
    for (i = 0; i < na; i++) {
        Uniao[*nuniao] = A[i];
        *nuniao = *nuniao + 1;
    }
    for (i = 0; i < nb; i++) {
        if (busca(*nuniao, Uniao, B[i]) == 0) {
            Uniao[*nuniao] = B[i];
            *nuniao = *nuniao + 1;
        }
    }
}
```

Questão 4

Dados um inteiro n ($n > 1$) e uma matriz A ($n \times n$) de números inteiros, imprimir quantas submatrizes quadradas (de dimensões $1{\times}1$, $2{\times}2$, ...$n{\times}n$, respectivamente) formadas apenas por zeros (0) são encontradas dentro da matriz A. Por exemplo, para a matriz A ($n = 4$)

0	0	20	40
0	0	0	0
35	0	0	0
10	0	0	0

Submatrizes de Dimensao = 1 X 1, Quantidade = 12
Submatrizes de Dimensao = 2 X 2, Quantidade = 5
Submatrizes de Dimensao = 3 X 3, Quantidade = 1
Submatrizes de Dimensao = 4 X 4, Quantidade = 0

Note que encontramos as 5 submatrizes de dimensão 2×2 da seguinte forma:

0	0	20	40
0	0	0	0
35	0	0	0
10	0	0	0

0	0	20	40
0	0	0	0
35	0	0	0
10	0	0	0

0	0	20	40
0	0	0	0
35	0	0	0
10	0	0	0

0	0	20	40
0	0	0	0
35	0	0	0
10	0	0	0

0	0	20	40
0	0	0	0
35	0	0	0
10	0	0	0

Para resolver o problema acima, faça cada um dos itens a seguir. Considere:

```
#define MAX 100
```

(a) (1.0 ponto) Faça uma função que dada a matriz M, i e j que correspondem à posição (i, j) da matriz e uma dimensão d, retorne 1 se a submatriz de dimensão $d \times d$ a partir da posição (i, j) (canto superior esquerdo da submatriz) é formada exclusivamente por zeros, e 0 caso contrário.

```
int zeros (int M[MAX][MAX], int i, int j, int d){
```

(b) (1.5 pontos) Faça um programa que lê um inteiro n ($n > 1$) e uma matriz A ($n \times n$) de números inteiros, e imprime: a quantidade de submatrizes de dimensões $1 \times 1,\ 2 \times 2, ...n \times n$ constituídas apenas por zeros (como mostrado no exemplo da questão).

Utilize a função definida no item anterior, mesmo que você não a tenha feito.