MAC 122 - PDA
2. Semestre de 2013 - prof. Marcilio – IME USP BMAC
Primeira Prova – 03 de Outubro de 2013

Questão	1	2	3	4	5	Total
Valor	2	2	2	2	2	10,0
Nota						

Nome:_____ NUSP:_____

Questão 1 (2 pontos):

Supondo a seguinte prioridade de operadores nas expressões aritméticas (4 é a maior e 1 a menor):

-	+	/	*
1	2	3	4

Traduza as seguintes expressões para a notação pós-fixa:

a) A - B + C / D * E

A B C D E * / + -

b) A / (B + C) * (D + C / E * F)

A B C + D C E F * / + * /

c) A * (B - C + D) / (E - F + G)

A B C D + - * E F G + - /

d) (A - B + C) / (A + B - C) * (A + B + C)

A B C + - A B + C - A B + C + * /

Traduza as mesmas expressões supondo todas as operações com a mesma prioridade:

a) A - B + C / D * E

A B - C + D / E *

b) A / (B + C) * (D + C / E * F)

A B C + / D C + E / F * *

c) A * (B - C + D) / (E - F + G)

A B C - D + * E F - G + /

d) (A - B + C) / (A + B - C) * (A + B + C)
A B - C + A B + C - / A B + C + *

Questão 2 (2 pontos):

Escreva uma função **`int CalcValorExprPos(int exp[])`** que calcula e devolve o valor da expressão aritmética em **`exp[]`** . A expressão já está em notação pós-fixa. O vetor **`exp[]`** contém os valores dos operandos que são todos maiores que zero. Os 4 operadores (**`+, -, *, /`**) são codificados como números negativos com a seguinte convenção: **`-1 = +; -2 = -; -3 = *; -4 = /`**. O número zero termina a expressão.

Exemplo - Considere a expressão: **`(1+2)/(3-4)*(5-6)`**
Em notação pós-fixa fica: **`1 2 + 3 4 - / 5 6 - *`**
No vetor **`exp[]`** ficará armazenada na seguinte forma:

i	0	1	2	3	4	5	6	7	8	9	10	11
exp	1	2	-1	3	4	-2	-4	5	6	-2	-3	0

```
int CalcPos(int exp[]) {
    int pilha[100], topo = -1, i = 0;
    while (exp[i] != 0) {
          if (exp[i] > 0) pilha[++topo] = exp[i];
          else { /* verifique qual o operador e faça a operação */
                 switch (exp[i]) {
                    case -1: pilha[topo - 1] = pilha[topo - 1] + pilha[topo]; break;
                    case -2: pilha[topo - 1] = pilha[topo - 1] - pilha[topo]; break;
                    case -3: pilha[topo - 1] = pilha[topo - 1] * pilha[topo]; break;
                    case -4: pilha[topo - 1] = pilha[topo - 1] / pilha[topo]; break;
                 }
                 topo--;
          }
          i++;
    }
    return pilha[0];
}
```

Questão 3: (2 pontos)

Um polinômio pode ser representado em um vetor **double**, de **MX** elementos, onde o i-ésimo elemento do vetor é o coeficiente de x^i. Exemplo: O polinômio $2 - 0.12x^2 + x^4$:

0	1	2	3	4	5	6	7	8	...	...	MX-1
2.0	0.0	-0.12	0.0	1.0	0.0	0.0	0.0	0.0	...	...	0.0

O tipo **Polinomio** é um ponteiro para um vetor de **double**. Supor também o **#define** abaixo para o valor de **MX**.

```
#define MX 1000
typedef double * Polinomio;
```

a) Escreva uma função **Polinomio IniciaPol()** que aloca memória para um polinômio e o devolve com todos os coeficientes nulos.

```
Polinomio IniciaPol() {
   int i;
   Polinomio t = malloc (MX * sizeof(double));
   /* Zera tudo */
   for(i = 0; i < MX; i++) t[i]=0;
   return t;
}
```

b) Escreva uma função **Polinomio SomaPol(Polinomio A,Polinomio B)** que devolve a soma dos polinômios **A** e **B**.

```
Polinomio SomaPol(Polinomio A, Polinomio B) {
   int i;
   /* Cria polinomio */
   Polinomio s = IniciaPol();
   /* soma os coeficientes de A e B*/
   for(i = 0; i < MX; i++) s[i] = A[i] + B[i];
   return s;
}
```

c) Escreva uma função **Polinomio MultPol(Polinomio A, Polinomio B)** que devolve a multiplicação dos polinômios **A** e **B**. Pode supor que o grau do polinômio produto é menor que **MX**.

```
Polinomio MultPol(Polinomio A, Polinomio B) {
   int i, j, gA, gB;
   /* Cria polinomio */
   Polinomio s = IniciaPol();
   /* descobre graus de A e B para não multiplicar todos */
   for (i = MX - 1; i >= 0; i--)
       if (A[i] != 0) {gA = i; break;}
   for (i = MX - 1; i >= 0; i--)
       if (B[i] != 0) {gB = i; break;}
   /* multiplica cada elemento de A por todos de B */
```

```
   for(i = 0; i <= gA; i++)
      for (j = 0; j <= gB; j++)
         s[i+j] = s[i+j] + A[i] * B[j];
   return s;
}
```

Questão 4 (2 pontos):

Considere uma lista ligada, na qual os elementos têm a seguinte definição:

```
typedef struct elem * link;
struct elem {
     int info;
     link prox;
};
```

a) Escreva uma função **`link ClassificadaLL (link p)`** verifica se os elementos da lista ligada, apontada por **`p`**, estão em ordem crescente. Ou seja, se o campo **`info`** de cada elemento é menor ou igual ao campo **`info`** do elemento seguinte. Retorna **`NULL`** se está classificada. Se não estiver, retorna ponteiro para o primeiro elemento encontrado fora de ordem. Use um algoritmo não recursivo.

```
link ClassificadaLL(link p) {
     link t = p, s;
     /* verifica os casos particulares
        lista vazia ou com um só elemento */
     if (t == NULL) return NULL;
     if (t -> prox == NULL) return NULL;
     /* A lista tem pelo menos 2 elementos */
     s = t -> prox; /* s indica o segundo elemento */
     /* percorre a lista até o final */
     t = elemento atual e s = próximo elemento */
     while (s != NULL)
           if (t->info <= s -> info) {
               /* continua a verificação */
               t = t -> prox; s = s -> prox;
           }
           else return s;  /* encontrou um fora de ordem */

     /* chegou ao fim da lista */
     return NULL;
}
```

b) Idem usando um algoritmo recursivo.

```
link ClassificadaLL(link p) {
     link t = p, s;
     /* verifica os casos particulares
        lista vazia ou com um só elemento */
     if (t == NULL) return NULL;
     if (t -> prox == NULL) return NULL;
     /* A lista tem pelo menos 2 elementos */
     s = t -> prox; /* s indica o segundo elemento */
     /* compara os 2 elementos */
     if (t->info <= s -> info)
        /* recursão no resto da lista */
        return ClassificadaLL (t -> prox);
```

```
    return s;  /* encontrou um fora de ordem */
}
```

Questão 5 (2 pontos):

Escreva uma função recursiva **`int Conta(int A[], int X, int N)`** que calcula e devolve quantos elementos iguais a **`X`** existem no vetor **`A`** de **`N`** elementos **`(A[0] ... A[N-1])`**.

```
int Conta(int A[], int X, int N) {
   int k;
   /* verifica se chegou ao fim da tabela */
   if (N < 0) return 0;
   /* conta mais 1 se é igual a X */
   if (A[N - 1] == X) k = 1 else k = 0;
   return k + Conta(A, X, N - 1);
}
```