MAC 122 - PDA
2. Semestre de 2014 - prof. Marcilio – IME USP BMAC
Primeira Prova – 02 de Outubro de 2014

Questão	1	2	3	4	Total
Valor	2.5	2.5	2.5	2.5	10,0
Nota					

Nome:_____ NUSP:_____

Questão 1 (2.5 pontos):

Considere os seguintes operadores aritméticos e lógicos do C e respectivas prioridades (7 é a maior prioridade e 1 é a menor):

||	&&	==	<	>	+	-	*	/	!
1	2	3	4	4	5	5	6	6	7

O operador **!** (**não**)é unário, os demais são binários. Os parêntesis como sempre alteram a prioridade.

Traduza as seguintes expressões para a notação pós-fixa:

1) ! (A == B || C == D)

A B == C D == || !

2) A == B || C < D + E

A B == C D E + < ||

3) (A + B) > C && D == E + (F || G)

A B + C > D E F G || + == &&

4) A * B && ! (C > D || E + F < G)

A B * C D > E F + G < || ! &&

5) ! A || B && C / (D + E) * F

A ! B C D E + / F * && || **(corrigido)**

Questão 2 (2.5 pontos):

a) Escreva uma função **`int NomeDeDigito(char s[])`** que recebe uma cadeia de caracteres contendo o nome de um dígito (**`"zero", "um", "dois", ..., "nove"`**) e devolve o seu valor numérico (**`0, 1,2,..., 9`**). Devolve **`-1`** se **`s[]`** não é nome de dígito.

```
char nomes[10][20] = {"zero", "um", "dois", "tres", "quatro", "cinco",
"seis", "sete", "oito", "nove"};

int NomeDeDigito(char s[]) {
    int i;
    for (i = 0; i < 10; i++)
        if(!strcmp(s, nomes[i])) return i;
    return -1; /* caso não seja nome de dígito */
}
```

b) Escreva uma função **`int NomeDeNumero(char ss[][10])`** que recebe uma matriz **`char`** em que cada linha contém uma cadeia de caracteres com o nome de um dígito. A função devolve o valor do número constituído por estes dígitos.
No exemplo abaixo deve ser devolvido o número **`2046`**. Como sempre, a cadeia de caracteres termina pelo zero binário. Use a função do item a acima.

d	o	i	s	0					
z	e	r	o	0					
q	u	a	t	r	o	0			
s	e	i	s	0					
0									
0									

```
int NomeDeNumero(char ss[][10]) {
    int k = 0, i = 0;
    /* varre as linhas de ss até encontrar uma com string vazia */
    while (1) {
       if(strlen(ss[i]) == 0) return k;
       k = k * 10 + NomeDeDigito(ss[i]);
       i++;
    }
}
```

Questão 3 (2,5 pontos):

Considere uma lista duplamente ligada circular (LDLC) com sentinela. Os elementos têm a seguinte definição:

```
typedef struct elem * link;
struct elem {
     int info;  /* informação */
     link ant,  /* ponteiro para o anterior */
          prox; /* ponteiro para o seguinte */
};
```

a) Escreva uma função link **ProcuraLDLC(link p, int x)** que procura elemento com **info** igual a **x** nesta LDLC. Devolve apontador para o elemento encontrado ou **NULL** se não encontrar. O ponteiro **p** indica o início da LDLC e aponta para o elemento sentinela.

```
/* procura elemento com info igual a x na lista duplamente ligada
   p aponta para o elemento sentinela da lista
   retorna ponteiro para o elemento encontrado ou NULL */
link ProcuraLDLC(link p, int x) {
   link q = p -> prox; /* primeiro elemento após a sentinela */
   p -> info = x; /* coloca x na sentinela */
   while (q -> info != x)
       q = q -> prox; /* tenta o próximo */
   /* achou x - verifica se é o sentinela */
   if (q == p) return NULL; /* não achou */
   return q; /* achou */
}
```

b) Escreva uma função **link RemoveLDLC(link p, int x)** quer remove da LDLC elemento com **info** igual a **x**. Devolve ponteiro para o elemento removido ou **NULL** se não encontrar. O ponteiro **p** indica o início da LDLC e aponta para o elemento sentinela. Use obrigatoriamente a função anterior **ProcuraLDLC**.

```
/* remove elemento com info igual a x da lista duplamente ligada
   p aponta para o elemento sentinela da lista
   devolve ponteiro para o elemento a removido ou NULL */
link RemoveLDLC(link p, int x) {
   link q, q1, q2;
   q = ProcuraLDLC(p, x); /* procura elemento com info igual x */
   if (q == NULL) return NULL;
   q1 = q -> ant; /* anterior */
   q2 = q -> prox; /* proximo */
   q1 -> prox = q2; /* anterior aponta para o seguinte */
   q2 -> ant = q1;  /* seguinte aponta para o anterior */
   return q;
}
```

Questão 4 (2.5 pontos):

O tipo **Polinomio** é definido da seguinte forma:

```
typedef struct pol * Polinomio;
struct pol {
    int Grau; /* grau do polinômio */
    double * Coef; /* ponteiro para um vetor double
                      que contém os coeficientes */
};
```

O ponteiro **Coef** aponta para um vetor tal que **Coef[k]** é o coeficiente de x^k

a) Escreva uma função **Polinomio CriaPol(int g)** que cria e devolve um polinômio de grau **g**, isto é, o vetor **coef** aponta para um vetor de **(g+1)** elementos com todos os coeficientes zerados.

```
/* cria polinômio de grau g - zera os coeficientes */
Polinomio CriaPol(int g) {
   Polinomio p; int i;
   /* cria polinômio p */
   p = malloc(sizeof(struct pol));
   /* define o grau */
   p -> Grau = g;
   /* aloca espaço em memória para o polinômio = g + 1 coeficientes */
   p -> Coef  = malloc((g + 1) * sizeof(double));
   /* zera os g + 1 coeficientes */
   for (i = 0; i <= g; i++) p -> Coef[i] = 0.0;
   return p;
}
```

b) Escreva uma função **Polinomio MultiplicaPol(Polinomio p, Polinomio q)** que devolve o polinômio produto de **p** por **q**.

```
Polinomio ProdutoPol(Polinomio p, Polinomio q) {
     Polinomio r; int i, j;
     /* grau dos polinomios p e q */
     int gp = p -> Grau;
     int gq = q -> Grau;
     /* cria polinomio para conter o produto */
     r = CriaPol(gp + gq);
     /* faz r = p * q */
     for (i = 0; i <= gp; i++)
      for (j = 0; j <= gq; j++)
       r -> Coef[i + j] = r -> Coef[i + j] + p -> Coef[i] * q -> Coef[j];
     return r;
}
```