MC 102Z – Exame Final - 11/12/12

Considere o programa abaixo

```
#include <stdio.h>
#define BASE 3

int fx(int m, int n){
   if((m == 0) || (n == 0)) return 0;
   return m*(n % BASE) + fx(m * BASE, n / BASE);
}

int main(){
    printf("%d\n",fx(14,15));
    printf("%d\n",fx(15,14));
    printf("%d\n",fx(0,15));
    printf("%d\n",fx(23,27));
    printf("%d\n",fx(27,23));
}
```

1. Qual será a saída desse programa ao ser executado ? (valor 1.0)

```
fx(14,15) = 14*0+fx(42,5) = 0+42*2+fx(126,1) = 84+126*1+f(378,0)  => "210"
fx(15,14) = 15*2+fx(45,4) = 30+45*1+fx(135,1) = 75+135*1+f(405,0) => "210"
fx(0,15) = 0 => "0"
fx(23,27) = 23*0+fx(69,9) = 69*0 + fx(207,3) = 207*0 + fx(621,1) = 621*1 +
fx(1863,0) => "621"
fx(27,23) = 27*2 + fx(81,7) = 54 + 81*1 + fx(243,2) = 135 + 243*2 + fx(929,0)
=> "621"
```

2. Escreva uma função não recursiva equivalente a fx (valor 2.5)

```
int fx(int m, int n){
    if(m == 0) return 0;
    int s = 0;
    while( n != 0){
           s += m * (n % BASE);
           m *= BASE;
           n /= BASE;
    }
    return s;
}
```

Curiosidades a respeito dessa função: (1) ela calcula o produto de m por n. (2) resultado é o mesmo para qualquer base maior que 1. (3) Usamos exatamente esse algoritmo para fazer multiplicações em decimal (BASE = 10). (4) os computadores usam o mesmo algoritmo para multiplicar inteiros binários (BASE = 2).

Considere o trecho de programa abaixo

```
#include <stdio.h>
/*** Descreve um 'produto' ***/
struct Produto{
       int cod;      // código do material (deve ser único)
       float valor;  // valor unitário
       char* descr;  // descrição do material
};

/*** Descreve um 'item de estoque'  ***/
struct Item{
            int quantidade;            // quantidade em estoque
            struct Produto *produto;   // referência ao 'produto'
       };

/***  Libera a estrutura referenciada por 'item'  ***/
void liberaItem( struct Item *item){
     if(item != NULL){
         struct Produto *prod = (*item).produto;
         if(prod != NULL){
                 char *descr = (*prod).descr;
                 if(descr != NULL) free(descr);
                 free(prod);
         }
         free(item);
     }
}

/** Aloca memória para  um novo item de estoque e retorna a referência  **/
struct Item *novoItem(int quantidade, int cod, float valor, char* descr);
```

3. Escreva a função novoItem(), de acordo com o protótipo definido. Essa função deve ser compatível com a liberação de memória feita pela função liberaItem(). Notar que (a) em novoItem(), os nomes dos parâmetros são iguais aos respectivos campos da estrutura e (b) liberaItem() faz três liberações de memória (relativas a descrição do produto, produto e item). (valor 2.5). Se necessário use as funções de <strings.h>:
    - int strlen(char* s): retorna o tamanho do string passado como parâmetro.
    - strcpy(char *destino, char* origem): copia o string origem para o string destino.

```
struct Item *novoItem(int quantidade, int cod, float valor, char* descr){
       struct Produto *prod = malloc(sizeof(struct Produto));
       char *dd = malloc(strlen(descr)+1);
       strcpy(dd, descr);
       (*prod).cod = cod;
       (*prod).valor = valor;
       (*prod).descr = dd;
       struct Item *item = malloc(sizeof(struct Item));
       (*item).quantidade = quantidade;
       (*item).produto = prod;
       return item;
}
```

Considere o trecho de programa a seguir:

```
#include <stdio.h>
#define NDIAS 365

/**  número de dias em cada mes (não considera anos bissextos  **/
int diasMes[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

struct Data{
       int dia; // de 1 a 31
       int mes; // 0 a 11
       int ano; // 2012, 2013, 2014, ...
       };

struct Data hoje(){
       struct data h = {11, 11, 2012};
       return h;
}
```

4. Escreva a função que compara duas datas passadas como parâmetro, obedecendo ao seguinte protótipo:

```
int compara(struct Data d1, struct Data d2);
```

Essa função deve retornar -1 se d1 for anterior a d2, 0 se as datas forem iguais e 1 se d1 for posterior a d2. (valor 2.0).

```
int comparaDatas(struct Data d1, struct Data d2){
     int c = d2.ano - d1.ano;
     if(c == 0) c = d2.mes - d1.mes;
     if(c == 0) c = d2.dia - d1.dia;
     if(c < 0) c = -1; else if(c > 0) c = 1;
     return c;
}
```

5. Escreva uma função que retorna a data correspondente a 'daqui a n dias' (data de hoje mais n dias). Sua função deve obedecer ao seguinte protótipo:

```
struct Data  hojeMaisNDias(int n);
```

Sua função deve considerar o número de dias de cada mês e não precisa considerar os anos bissextos (valor 3.0). Se necessário escreva funções auxiliares.

Observação: A soma das questões é 11.0. Caso alguém obtenha nota maior que 10.0 esta será truncada para 10.0.

```
/* Número de dias até o final do ano, a partir de  uma certa data d */
int diasAF(struct Data d){
    int r = diasMes[d.mes] - d.dia;
    int i = d.mes + 1;
    while(i < 12) r += diasMes[i++];
    return r;
}

struct Data hojeMaisNDias(int dias){
       struct Data res, d = hoje();
       int anos = dias / NDIAS;  // anos 'inteiros' de acréscimo
       int dm = dias % NDIAS;    // dias a mais, além dos anos 'inteiros'
       int af = diasAF(d);       // dias até o final do ano a partir de d
       int m = d.mes;            // 'mes base'
       int di = d.dia;           // 'dia base'
       // verifica se 'dias a mais' pode ir além do final do ano
       if(dm > af) {
            anos++;    // muda o ano
            dm -= af;  // desconta os dias
            m = 0;     // e a 'base' passa a ser o início de janeiro ...
            di = 0;
       }
       dm += di;
       // avança nos dias, mes a mes
       for(; dm > diasMes[m]; m++) dm -= diasMes[m];
       res.ano = d.ano + anos;
       res.mes = m;
       res.dia = dm;
       return res;
}
```