UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
I.C.M.C.

SCC-601 Algoritmos e Estruturas de Dados I (EC)
Profa. Graça Nunes
2º. Semestre de 2010

2ª. Prova (Gabarito)
21/10/2010

Nome: _____ Nro USP: _____

1) Considerando uma lista **ordenada** encadeada dinâmica com valores inteiros, L:

(a) (0.5)Faça a declaração de tipo, em C, para essa lista.

```
Typedef struct no {
        int info;
         struct no *lig;
        } Lista;
```

(b) (1.5) escreva uma função **recursiva** que busca uma chave x em L.

```
Boolean busca( int x, Lista *L) {
        if (L == NULL) return FALSE;
        if (L->info == x) return TRUE;
        if (L->info > x) return FALSE;
        return busca (x, L->lig);
 }
```

ou

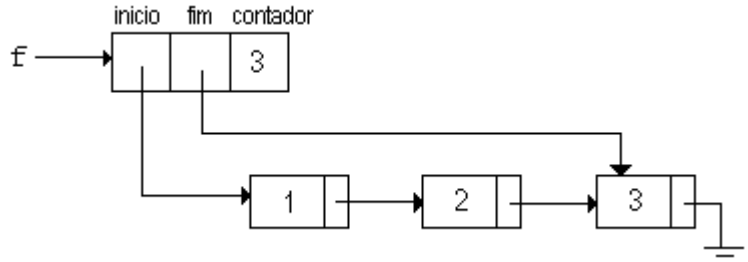
```
Lista * busca(int x, Lista *L) {
        if (L == NULL) return NULL;
        if (L->info == x) return L;
        if (L->info > x) return NULL;
        return busca (x, L->lig);
 }
```

(c)(0.5) A versão recursiva de (b) é mais vantajosa do que uma versão não-recursiva? Por que?
Não, por que ela gasta mais memória do que a não recursiva, além de gastar o mesmo tempo. A memória extra é para as chamadas recursivas, e isso é da O(n), onde n é o tamanho da lista.

2) (1.0) Considere a seguinte definição de fila encadeada, em que o nó cabeça guarda também um contador de número de elementos da fila:

```
typedef struct elem{
    int info;
    struct elem *lig;
}tipo_elem;
```

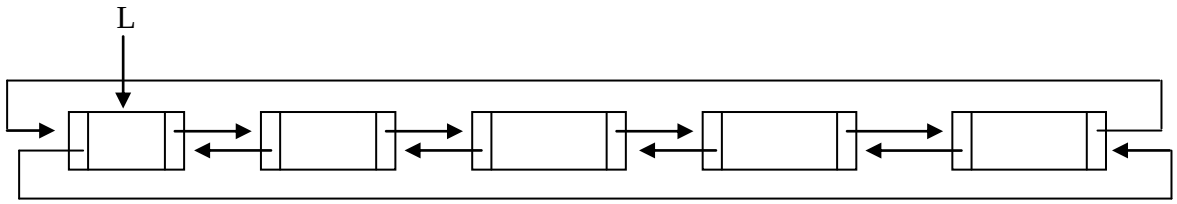
```
typedef struct{
    tipo_elem *inicio;
    tipo_elem *fim;
    int contador;
}fila;
fila f;
```



Escreva (não precisa ser na forma de função; basta a sequência de comandos) a operação de inserção (de um valor x) na fila (Atenção: a fila não requer qualquer ordenação).

```
var p: pont-no;
begin
    new(p);
    p^.info := x;
    p^.lig := nil;
    if f^.contador = 0 then begin
                            f^.inicio:= p; f^.fim:=p; f^.contador :=1
                            end
   else begin
            f^.fim^.lig := p; f^.fim:=p; f^.contador:= f^.contador+1
        end
end;
```

3) Considere a existência de uma lista duplamente encadeada e circular, como esquematizada na figura abaixo:



A diferença em relação à lista simplesmente encadeada, vista em classe, é que, além da indicação do sucessor, cada elemento guarda também a indicação do predecessor. Assim, ao invés de um único campo **lig**, neste caso, temos uma ligação à esquerda, para o predecessor e outra, à direita, para o sucessor: **ligesq e ligdir**. **L** aponta para o primeiro elemento da lista. A circularidade garante o percurso em toda a lista a partir de qualquer registro.
Suponha que essa lista é **dinâmica** e **não ordenada,** e cada elemento é um número inteiro.

(a) (1.0) Declare em C a estrutura de dados correspondente a essa lista (tipo pont_lista).

```
struct lista {
 int info;
 struct lista *esq, *dir;
```

```
};
typedef struct lista *pont_lista;
```

(b) (2.0) Suponha que exista uma função de busca que retorna o endereço do inteiro procurado, ou *null*, se não encontrar: ***pont_lista* busca(int n, pont_lista* L).***

Escreva em C uma função para eliminar um número da lista (somente se ele estiver presente nela). Use a função **busca** para isso. Assuma que a lista não contém números repetidos. Se o registro eliminado for o apontado por L, o novo primeiro elemento deverá ser seu sucessor à direita.

```
boolean elimina (int n, pont_lista *L) {
/*retorna true, se eliminou; false, caso contrário (quando não encontra)*/
        pont_lista *p;
        p= busca(n,L);
        if (p == null) return FALSE;    /* n não está na lista*/
        else {   /* elimina registro apontado por p*/
                p->ligesq->ligdir = p->ligdir;  /*o reg. da esq. aponta para o da direita*/
                p->ligdir->ligesq = p->ligesq;  /*o reg. da dir. aponta para o da esquerda*/
               if (p == L)  L= L->ligdir; /*nova cabeça é o registro da direita*/
                free(p);
                return TRUE;
             }
}
```

4) (1.5) Considere a estrutura de lista generalizada dada abaixo, em C:

```
struct no {
            int tipo; /* 0 se átomo; 1 se sublista */
            union {
                  int atomo;
                  struct no *lista;
            } info;
            struct no *prox;
};
typedef struct no Rec;
Rec *Lista;
```

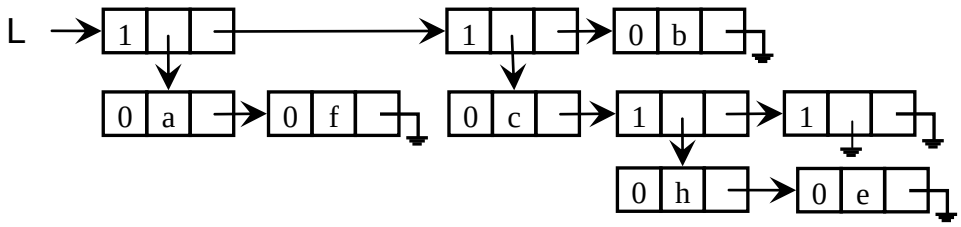
Considere a operação recursiva de busca de uma sublista, SL, numa lista generalizada, L, cujo resultado é o valor FALSE, se resultar insucesso, ou TRUE e o endereço de SL, caso contrário. A declaração da função em C que implementa essa operação aparece abaixo, porém incompleta. Veja que ela assume a existência de uma outra função, Igual (L1, L2), que verifica a igualdade de 2 listas generalizadas.

Pede-se: Complete a declaração da função nos pontos **(1) e (2)**

```
Boolean  BuscaSubLista (Rec *L, *SL, *Endereco){
/* Busca ocorrência da lista SL na lista L; se encontrar, retorna seu
Endereço no 3°. parâmetro  */

   if Igual(SL,L){ Endereco = L; return TRUE; }
   elseif (L !== null)
        { if (L->tipo == 0) {(1)return BuscaSubLista (L->prox, SL,
Endereco);} }
```

```
            else // é lista  {
                if ((2)(!(BuscaSubLista (L->info.lista,SL, Endereco)     )

                  return BuscaSubLista (L->prox,SL, Endereco);
                else return TRUE;
            }
    return FALSE;

}
```

5) (2.0) Uma lista generalizada L é uma sequência finita de n≥0 nós, sendo que cada nó armazena um átomo ou uma sub-lista, o que é indicado pelo campo "tipo" do nó: se tipo=0, então o nó armazena um átomo; se tipo=1, então o nó armazena uma sub-lista. A figura abaixo é um exemplo de representação da lista generalizada [[a,f],[c,[h,e],[]],b].



Escreva em C uma função **recursiva** que retorne o maior elemento (do tipo char) de uma lista generalizada. Por exemplo, para a lista anterior, o maior elemento retornado seria "h".
A declaração básica do nó já é dada abaixo.

```
struct no {
      int tipo;
      union {
          char info1;          //para átomo, se tipo=0
          struct no *info2;  //para lista, se tipo=1
        }elem;
      struct no *next;
}

struct no *L;
```

```
char maior_elem(struct no *L) {
        char aux1, aux2;
        if (L != null) {
          if (L-> tipo == 0){  // é átomo
          aux1 = maior_elem(L->next);
          if (aux1 > L->elem.info1) return aux1;
          else return L->elem.info1;
         }
         else { // é sublista
          aux1 = maior_elem(L->next);
          aux2 = maior_elem(L->elem.info2);
          if (aux1 > aux2) return aux1;
          else return aux2;
         }
        else return "menor character possível"; //se L == null
        }
```