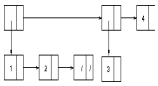
# Questão 1

Considere a seguinte declaração para listas generalizadas.

```
enum elem_t {tipo_int, tipo_sublista};

union info_lista {
  int i;
  struct No* sublista;
};

typedef struct No {
  enum elem_t tipo;
  union info_lista info;
  struct No* prox;
} No;
```

## Questão 1a

Desenhe a lista generalizada correspondente a ((1,2,()),(3),4). Note que um dos elementos é uma sublista vazia.



---

# Questão 1b

Dadas as funções abaixo, você deve escrever (utilizando a notação com parênteses) o resultado da chamada da função {\tt f1()}.

```
/* Cria um átomo com valor i e o concatena à lista l */
/* Retorna um apontador para a nova lista */
No* al(int i, No *l) {
  No* n = (No*) malloc (sizeof(No));
  n->tipo = tipo_int;
  n->info.i = i;
  n->prox = l;
  return n;
}

/* Cria um nó com apontador para a sublista s e o concatena à lista l */
/* Retorna um apontador para a nova lista */
No* sl(No* s, No *l) {
  No* n = (No*) malloc (sizeof(No));
  n->tipo = tipo_sublista;
  n->info.sublista = s;
  n->prox = l;
  return n;
}

/* Função f1: escreva ao lado a lista retornada. */
No *f1() {
  return sl(al(1,al(2,sl(al(3,NULL),NULL))),sl(NULL,al(4,NULL)));
}
```

### Resposta:

```
((1,2,(3)),(),4)
```

### Resposta:

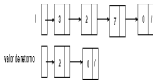
### Questão 1c

```
/* Deve retornar um apontador para  (1,(2,()),(3),4) */
No *f2() {
  No* l0 = al(2,sl(NULL,NULL));  /* (2,()) */
  No* l1 = al(3,NULL);           /* (3) */
  return al(1,sl(l0,sl(l1,al(4,NULL))));
}
```

---

# Questão 2

Escreva uma função **recursiva** que percorre uma lista ligada de inteiros e retorna uma outra lista (com novos nós) que contém apenas os elementos da primeira lista cujo conteúdo é um número par (vide figura). A função retorna um apontador para a lista modificada. Utilize as declarações abaixo.



```
typedef struct no {
  int v;
  struct no *prox;
} No;
```

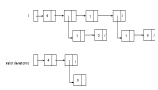
### Resposta

```
No* apenas_pares(No* l) {
  if (l == NULL)
    return NULL;
  if (l->v % 2 == 0) {
    No* n = malloc(sizeof(No));
    n->v = l->v;
    n->prox = apenas_pares(l->prox);
    return n;
  }
  return apenas_pares(l->prox);
}
```

---

# Questão 3

Escreva uma função **recursiva** que percorre uma **lista generalizada** como definida no exercício 1 e retorna uma outra lista (com novos nós) que contém apenas os elementos da primeira lista cujo conteúdo é um número par. Caso uma sublista contenha apenas elementos com conteúdo ímpar, esta sublista não deve ser incluída no retorno (vide figura).



```
No* apenas_pares(No* l) {
  if (l == NULL)
    return NULL;
  if (l->tipo == tipo_sublista) {
    No* s = apenas_pares(l->info.sublista);
    if (s != NULL) {
      No* n = malloc(sizeof(No));
      n->tipo = tipo_sublista;
      n->info.sublista = s;
      n->prox = apenas_pares(l->prox);
      return n;
    }
    else
      return apenas_pares(l->prox);
  }
  if (l->info.i % 2 == 0) {
    No* n = malloc(sizeof(No));
    n->tipo = tipo_int;
    n->info.i = l->info.i;
    n->prox = apenas_pares(l->prox);
    return n;
  }
  else
    return apenas_pares(l->prox);
}
```

---

# Questão 4

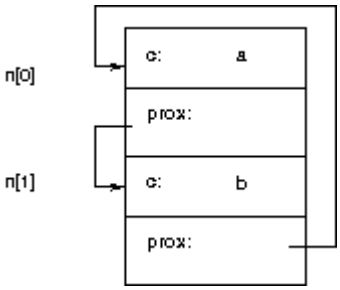
Indique se o código abaixo poderá executar corretamente ou será interrompido por um erro de acesso à memória. Se você achar que o programa irá apresentar um erro, explique a razão e exiba as linhas que serão impressas antes do erro ocorrer. Caso contrário, exiba a saída completa do programa.

```
#include <stdlib.h>
#include <stdio.h>

typedef struct no {
  char c;
  struct no* prox;
} No;

 void inicia_valores(No* ap_n, char c, No* prox) {
   ap_n->c = c;
   ap_n->prox = prox;
 }

 int main() {
   No n[2];
   inicia_valores(&n[0], 'a', &n[1]);
   inicia_valores(&n[1], 'b', &n[0]);
   printf("1: %c\n", n[0].c);
   printf("2: %c\n", n[0].prox->c);
   printf("3: %c\n", n[1].prox->c);
   printf("4: %c\n", n[1].prox->prox->c);
```

```
   return 0;
}
```

## Resposta

O programa **não** contém erros de acesso à memória. Forma-se uma estrutura semelhate ao esquema abaixo:




A saída é:

```
1: a
2: b
3: a
4: b
```