# Questão 1

Considere a seguinte declaração para listas generalizadas.

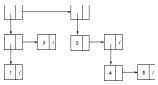
```
enum elem_t {tipo_int, tipo_sublista};

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

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

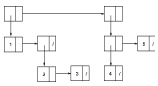
## Questão 1a (versão A)

Desenhe a lista generalizada correspondente a (((1), 2), (3, (4, 5))).



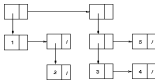
## Questão 1a (versão B)

Desenhe a lista generalizada correspondente a ((1, (2, 3)), ((4), 5)).



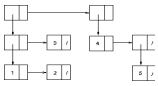
## Questão 1a (versão C)

Desenhe a lista generalizada correspondente a ((1, (2)), ((3, 4), 5)).



## Questão 1a (versão D)

Desenhe a lista generalizada correspondente a (((1, 2), 3), (4, (5))).



---

# Questão 1b

Dadas as funções abaixo (um pouco diferentes das apresentadas em sala de aula), você deve escrever uma função que retorna um apontador para uma lista equivalente a ...*(verifique versão da prova)*.

```
/* 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 exemplo: retorna um apontador para ((1)) */
No *exemplo() {
  return sl(al(1, NULL), NULL);
}
```

## Questão 2b (Versão A)

```
/* Retorna um apontador para  ((1, 2, (3)), 4) */
/* Dica: use variáveis auxiliares para compor a lista */
No *retorna_lista() {
  No* l0 = sl(al(3, NULL), NULL); /* ((3)) */
  No* l1 = al(2, l0);   /* (2, (3)) */
  No* l2 = al(1, l1);  /* (1, 2, (3)) */
  No* l3 = al(4, NULL); /* (4) */
  return sl(l2, l3);    /* ((1, 2, (3)), 4) */
}
```

## Questão 2b (Versão B)

```
/* Retorna um apontador para  ((1, (2), 3), 4) */
/* Dica: use variáveis auxiliares para compor a lista */
No *retorna_lista() {
  No* l0 = al(3, NULL); /* (3) */
  No* l1 = al(2, NULL); /* (2) */
  No* l2 = sl(l1, l0);  /* ((2), 3) */
  No* l3 = al(1, l2);   /* (1, (2), 3) */
  No* l4 = al(4, NULL); /* (4) */
  return sl(l3, l4);    /* ((1, (2), 3), 4) */
}
```

## Questão 2b (Versão C)

```
/* Retorna um apontador para ((1, 2, 3), (4)) */
/* Dica: use variáveis auxiliares para compor a lista */
No *retorna_lista() {
  No* l0 = sl(al(4, NULL), NULL); /* ((4)) */
  No* l1 = al(3, NULL); /* (3) */
  No* l2 = al(2, l1);   /* (2, 3) */
  No* l3 = al(1, l2);   /* (1, 2, 3) */
  return sl(l3, l0);    /* ((1, 2, 3), (4)) */
}
```

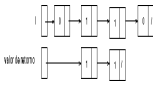
### Questão 2b (Versão D)

```
/* Retorna um apontador para (((1), 2, 3), 4) */
/* Dica: use variáveis auxiliares para compor a lista */
No *retorna_lista() {
  No* l0 = al(1, NULL); /* (1) */
  No* l1 = al(3, NULL); /* (3) */
  No* l2 = al(2, l1);   /* (2, 3) */
  No* l3 = sl(l0, l2);  /* ((1), 2, 3) */
  No* l4 = al(4, NULL); /* (4) */
  return sl(l3, l4);    /* (((1), 2, 3), 4) */
}
```

---

# Questão 2

Escreva uma função **recursiva** que percorre uma lista ligada de inteiros e remove (liberando adequadamente a memória) todos os nós que contêm elementos nulos (vide figura). A função retorna um apontador para a lista modificada. Utilize as declarações abaixo.



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

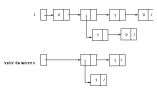
### Resposta

```
No* remove_zeros(No* l) {
  if (l == NULL)
   return NULL;
  if (l->v == 0) {
   No* r = l;
   l = l->prox;
   free(r);
   return remove_zeros(l);
  }
 else{
   l->prox = remove_zeros(l->prox);
  return l;
 }
}
```

---

# Questão 3

Escreva uma função **recursiva** que percorre uma **lista generalizada** como definida no exercício 1 e remove (liberando adequadamente a memória) todos os nós átomos que contêm elementos nulos (vide figura). A função retorna um apontador para a lista modificada.



```
No* remove_zeros(No* l) {
```

```
 if (l == NULL)
    return NULL;
  if (l->tipo == tipo_sublista)
    l->info.sublista = remove_zeros(l->info.sublista);
  else
    if (l->info.i == 0) {
      No* r = l;
      l = l->prox;
      free(r);
      return remove_zeros(l);
    }
  l->prox = remove_zeros(l->prox);
  return l;
}
```

---

# Questão 4

O objetivo do programa abaixo é (i) iniciar uma pilha (cuja implementação utiliza uma lista ligada) indicando que ela está vazia e (ii) empilhar um elemento. O código em C apresentado cumprirá este objetivo? Caso você considere que este programa tem um ou mais erros, indique as linhas que contêm problemas, explique-os e mostre como eles poderiam ser corrigidos de maneira que o código possa funcionar adequadamente, mantendo seus objetivos e divisão em funções.

```
 1: #include
 2: #include
 3:
 4: typedef struct no {
 5:   char c;
 6:   struct no* prox;
 7: } No;
 8:
 9: typedef No *Pilha;
10:
11: void cria_pilha(Pilha *p) {
12:   p = NULL;
13: }
14:
15: void empilha(Pilha *p, char *c) {
16:   No *n = (No *) malloc (sizeof (No));
17:   n->c = *c;
18:   n->prox = p;
19:   p = &n;
20: }
21:
22: int main() {
23:   Pilha *p;
24:   char *c;
25:
26:   cria_pilha(p);
27:   scanf("%c ", c);
28:   empilha (p, c);
29:   return 0;
30: }
```

### Resposta

- Linha 23 - É alocado espaço para um apontador para uma estrutura Pilha e não para a

estrutura propriamente dita. O correto seria:

```
Pilha p;
```

ou

```
Pilha *p = malloc (sizeof(Pilha));
```

- Linha 24 - Mesmo problema para o caracter c. Note que não é necessário alterar o cabeçalho de empilha.
- Linha 12 - O apontador para a Pilha é alterado, mas a pilha não é inicializada.
- Linha 18 - A atualização de n->prox não está correta. Note que n->prox é do tipo No* e p é do tipo No**. O correto seria n->prox = *p.
- Linha 19 - Problema semelhante ao anterior. O apontador para a pilha receberá um endereço que se tornará inválido após o final da execução.

Código corrigido.

```
 1: #include
 2: #include
 3:
 4: typedef struct no {
 5:   char c;
 6:   struct no* prox;
 7: } No;
 8:
 9: typedef No *Pilha;
10:
11: void cria_pilha(Pilha *p) {
12:   *p = NULL;
13: }
14:
15: void empilha(Pilha *p, char *c) {
16:   No *n = (No *) malloc (sizeof (No));
17:   n->c = *c;
18:   n->prox = *p;
19:   *p = n;
20: }
21:
22: int main() {
23:   Pilha p;
24:   char c;
25:
26:   cria_pilha(&p);
27:   scanf("%c ", &c);
28:   empilha (&p, &c);
29:   return 0;
30: }
```