# MC-202 — Aula 3
# Operações em listas e variações

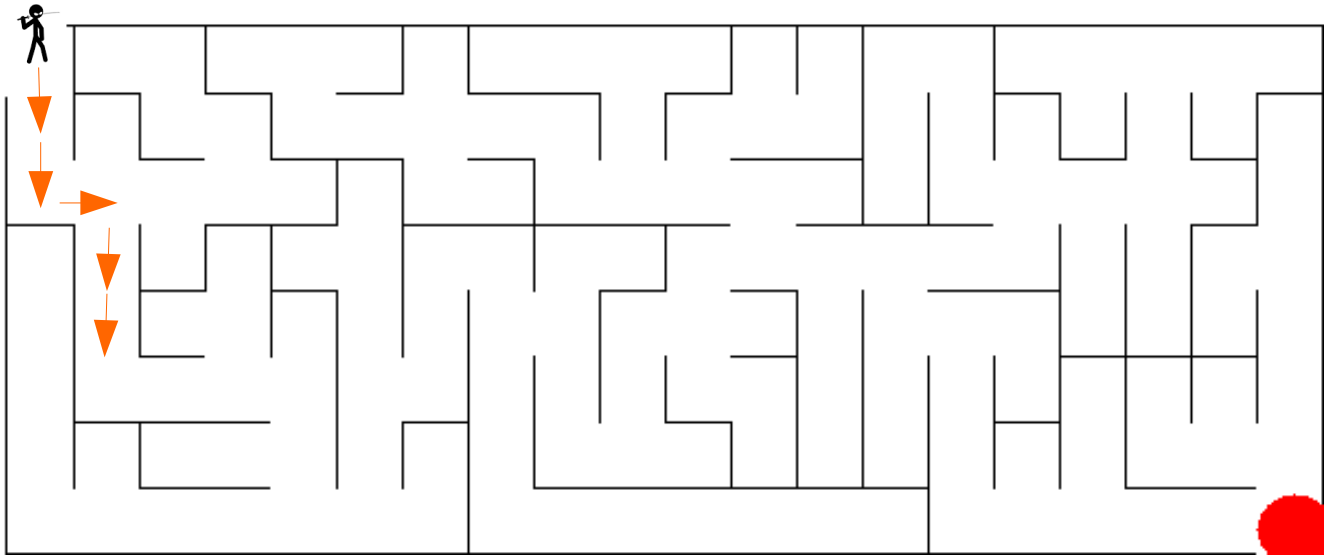
Lehilton Pedrosa

Instituto de Computação – Unicamp

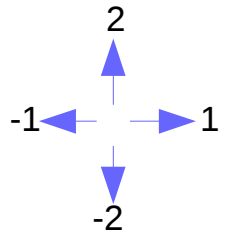
Segundo Semestre de 2015

# Roteiro

# Labirinto



sul, sul, leste, sul, sul,...




## Problema

Faça um programa que leia as orientações (de acordo com o número) para chegar ao destino e imprima as orientações de ida-e-volta.

# Pseudocódigo

## Indo e voltando

```
1. lista_ida = ler instruções de ida
2. lista_volta = copiar lista_ida
3. lista_volta = inverter lista_volta
4. para cada nó em lista_volta:
      a) trocar 1 por -1 e 2 por -2
5. lista_ida_volta = juntar lista_ida e lista_volta
6. imprimir lista_ida_volta
```

# Operações em lista encadeada

Vamos usar um TAD para escrever um código “parecido”.

# Operações em lista encadeada

Vamos usar um TAD para escrever um código "parecido".

## lista.h

```
typedef struct No { int dado; struct No *prox; } No;
```

# Operações em lista encadeada

Vamos usar um TAD para escrever um código "parecido".

## lista.h

```
typedef struct No { int dado; struct No *prox; } No;

void iniciar_lista(No **lista);
void inserir_elemento_final(No **lista, int x);
void destruir_lista(No **lista);
No *copiar_lista(No *lista);
void inverter_lista(No **lista);
void concatenar_lista(No **lista, No **outra);
```

# Operações em lista encadeada

Vamos usar um TAD para escrever um código "parecido".

## lista.h

```
typedef struct No { int dado; struct No *prox; } No;

void iniciar_lista(No **lista);
void inserir_elemento_final(No **lista, int x);
void destruir_lista(No **lista);
No *copiar_lista(No *lista);
void inverter_lista(No **lista);
void concatenar_lista(No **lista, No **outra);

// outras operações
No *procurar_elemento(No *lista, int x);
No *acessar_elemento_pos(No *lista, int k);
void remover_elemento(No **lista, No *no);
void inserir_em_orden(No **lista, int x);
void separar_lista(No **lista, No **nova, No *meio);
...
```

# Copiando

```
No *copiar_lista(No *lista) {
    No *copia, *novo, *anterior, *p;
    p = lista;
    copia = NULL;

    while (p) { // mesmo que p != NULL
        novo = malloc(sizeof(No));
        // checa memória...

        novo->dado = p->dado;
        novo->prox = NULL;
        if (anterior != NULL) {
            anterior->prox = novo;
        } else {
            copia = novo;
        }
        anterior = novo;
        p = p->prox;
    }
    return copia;
}
```

# Invertendo

```
void inverter_lista(No **lista) {
    No *p, *q, *invertida;
    p = *lista;
    invertida = NULL;

    while (p) {
        q = p;
        p = q->prox;
        q->prox = invertida;
        invertida = q;
    }
    *lista = invertida;
}
```

# Invertendo

```
void inverter_lista(No **lista) {
    No *p, *q, *invertida;
    p = *lista;
    invertida = NULL;

    while (p) {
        q = p;
        p = q->prox;
        q->prox = invertida;
        invertida = q;
    }
    *lista = invertida;
}
```

O que acontece se colocarmos `p = q->prox;` no final do loop?

# Concatenando

```
void concatenar_lista(No **lista, No **outra) {
    No *p;
    if (*lista == NULL) {
        *lista = *outra;
    } else {
        p = *lista;
        while (p->prox)
            p = p->prox;
        p->prox = *outra;
    }
    *outra = NULL; // invalida lista!
}
```

# Concatenando

```
void concatenar_lista(No **lista, No **outra) {
    No *p;
    if (*lista == NULL) {
        *lista = *outra;
    } else {
        p = *lista;
        while (p->prox)
            p = p->prox;
        p->prox = *outra;
    }
    *outra = NULL; // invalida lista!
}
```

A variável original de `outra` não pode mais ser acessada!

# Código em C para o labirinto

## Indo e voltando

```
#include <stdio.h>
#include "lista.h"
int main() {
    int lido;
    No *lista, *volta, *ida_volta, *p;

    // lê lista
    iniciar_lista(&lista);
    do {
        scanf("%d", &lido);
        if (lido != 0)
            inserir_elemento_final(&lista, lido);
    } while(lido != 0);

    // obtém inversa
    volta = copiar_lista(lista);
    inverter_lista(volta);
```

# Código em C para o labirinto (cont.)

```
    // inverte direções
    p = volta;
    while (p) {
        p->dado = - p->dado;
        p = p->prox;
    }

    // junta listas
    concatenar_lista(&lista, &volta);

    // imprime lista
    for(p = lista; p; p = p->prox)
        printf("%d\n", p->dado);

    return 0;
}
```

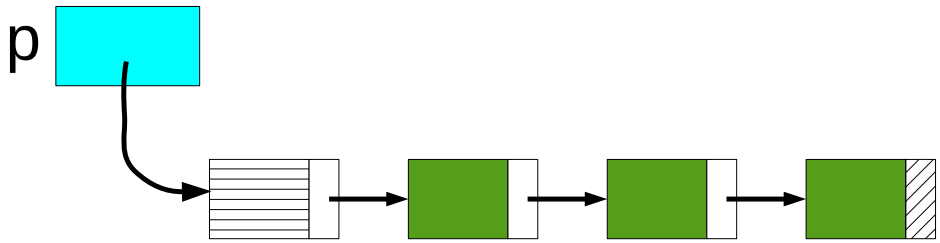
# Código em C para o labirinto (cont.)

```
    // inverte direções
    p = volta;
    while (p) {
        p->dado = - p->dado;
        p = p->prox;
    }

    // junta listas
    concatenar_lista(&lista, &volta);

    // imprime lista
    for(p = lista; p; p = p->prox)
        printf("%d\n", p->dado);

    return 0;
}
```

- duas formas de percorrer a lista
- a variável volta não pode mais ser usada!

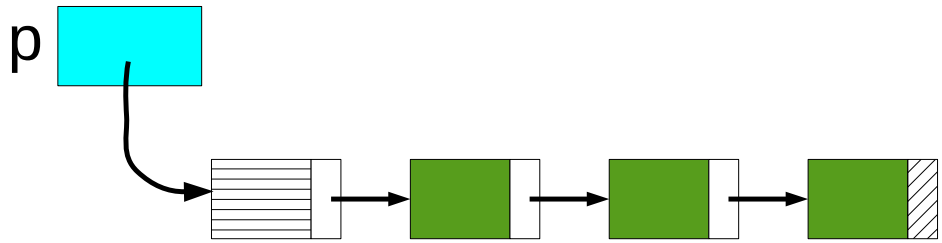
# I) Listas com cabeça

Lista com cabeça:

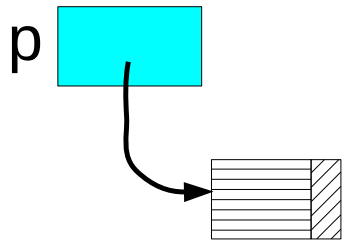


# I) Listas com cabeça

Lista com cabeça:




Lista com cabeça vazia:

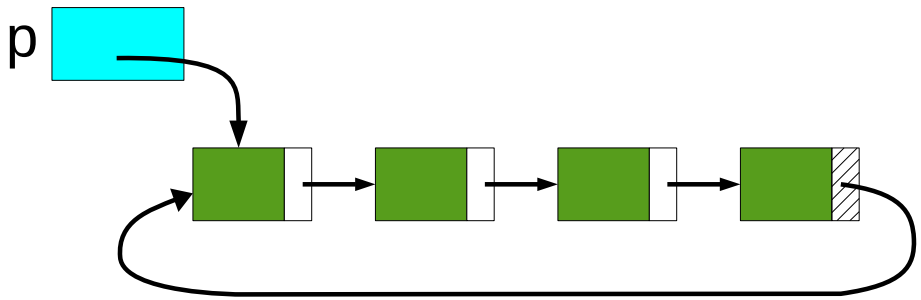


# II) Listas circulares

## Usos

- Cardápio rotativo do restaurante
- buffers em sistemas operacionais

Lista circular:

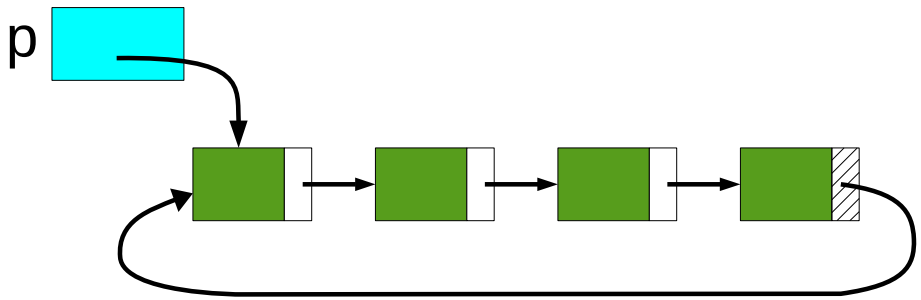


# II) Listas circulares

## Usos

- Cardápio rotativo do restaurante
- buffers em sistemas operacionais

Lista circular:



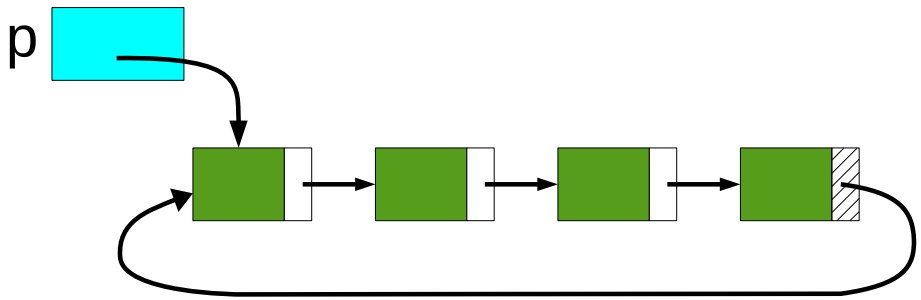

Lista circular vazia:

# II) Listas circulares

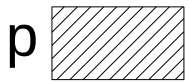
## Usos

- Cardápio rotativo do restaurante
- buffers em sistemas operacionais

Lista circular:




Lista circular vazia:



# Inserindo em lista circular

```
void inserir_elemento_circular(No **lista, int x) {
    No *novo;
    novo = malloc(sizeof(No));
    ... // verificar memória

    novo->dado = x;

    if (*lista == NULL) {
        *lista = novo;
        novo->prox = novo;
    } else {
        novo->prox = (*lista)->prox;
          (*lista)->prox = novo;
    }
}
```

# Inserindo em lista circular

```
void inserir_elemento_circular(No **lista, int x) {
    No *novo;
    novo = malloc(sizeof(No));
    ... // verificar memória

    novo->dado = x;

    if (*lista == NULL) {
        *lista = novo;
        novo->prox = novo;
    } else {
        novo->prox = (*lista)->prox;
          (*lista)->prox = novo;
    }
}
```

E se tivéssemos um nó cabeça?

# Removendo de lista circular

```
void remover_elemento_circular(No **lista, No *no) {
    No *anterior;
    // se contém só um elemento
    if (no->prox == no) {
        *lista = NULL;
    } else {
        // se nó removido é o início da lista, avança a lista
        if (no == *lista)
            *lista = (*lista)->prox;
        // encontra anterior
        anterior = no->prox;
        while (anterior->prox != no)
            anterior = anterior->prox;
        // remove nó da lista
        anterior->prox = no->prox;
    }
    free(no);
}
```

# Removendo de lista circular

```
void remover_elemento_circular(No **lista, No *no) {
    No *anterior;
    // se contém só um elemento
    if (no->prox == no) {
        *lista = NULL;
    } else {
        // se nó removido é o início da lista, avança a lista
        if (no == *lista)
            *lista = (*lista)->prox;
        // encontra anterior
        anterior = no->prox;
        while (anterior->prox != no)
            anterior = anterior->prox;
        // remove nó da lista
        anterior->prox = no->prox;
    }
    free(no);
}
```

Teste com listas de 1, 2 ou 3 elementos!

# Removendo de lista circular

```
void remover_elemento_circular(No **lista, No *no) {
    No *anterior;
    // se contém só um elemento
    if (no->prox == no) {
        *lista = NULL;
    } else {
        // se nó removido é o início da lista, avança a lista
        if (no == *lista)
            *lista = (*lista)->prox;
        // encontra anterior
        anterior = no->prox;
        while (anterior->prox != no)
            anterior = anterior->prox;
        // remove nó da lista
        anterior->prox = no->prox;
    }
    free(no);
}
```

Teste com listas de 1, 2 ou 3 elementos! Quando dá pra melhorar?

# Percorrendo uma lista circular

```
void imprimir_lista_circular(No *lista) {
    No *p;
    p = lista;
    do {
        printf("%d\n", p->dado);
        p = lista->prox;
    } while (p != lista);
}
```

# Percorrendo uma lista circular

```
void imprimir_lista_circular(No *lista) {
    No *p;
    p = lista;
    do {
        printf("%d\n", p->dado);
        p = lista->prox;
    } while (p != lista);
}
```

E se tivéssemos usado `while` ao invés de `do ... while`?

# Jogo de turnos

## Jogo

Em uma roda de jogadores, no turno de cada um:

- sorteia-se uma letra
- o jogador escreve uma palavra que começa com essa letra
- se ele errar, sai do jogo
- se ele acertar, passa para o próximo

Quem nunca errar, vence!

# Implementando jogo

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

typedef struct Jogador { char *nome; struct Jogador *prox; } Jogador;
```

# Implementando jogo

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

typedef struct Jogador { char *nome; struct Jogador *prox; } Jogador;

int main() {
    Jogador *lista, *aux;
    char nome[100];
    char letra;

    // lê nomes dos jogadores e inicializa lista
    inicializa_lista_circular(&lista);
    do {
        scanf("%s", nome);
        if (nome[0] != 0)
            adicioar_elemento(&lista, nome);
    } while (nome[0] != 0);
```

# Implementando jogo

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

typedef struct Jogador { char *nome; struct Jogador *prox; } Jogador;

int main() {
    Jogador *lista, *aux;
    char nome[100];
    char letra;

    // lê nomes dos jogadores e inicializa lista
    inicializa_lista_circular(&lista);
    do {
        scanf("%s", nome);
        if (nome[0] != 0)
            adicioar_elemento(&lista, nome);
    } while (nome[0] != 0);
```

De acordo com essa especificação de nó, como será salvo o nome do jogador? Implemente as funções que faltam!

# Implementando jogo (cont.)

```
// enquanto existe mais de um jogador
while(lista->prox != lista) {
```

# Implementando jogo (cont.)

```
// enquanto existe mais de um jogador
while(lista->prox != lista) {
    // sorteia uma letra aleatória
    letra = rand() % 26 + 'a'; // por que isso funciona?
    printf("%s, escreva nome com %c: ", lista->prox->nome, letra);
    scanf("%s", nome);
```

# Implementando jogo (cont.)

```
// enquanto existe mais de um jogador
while(lista->prox != lista) {
    // sorteia uma letra aleatória
    letra = rand() % 26 + 'a'; // por que isso funciona?
    printf("%s, escreva nome com %c: ", lista->prox->nome, letra);
    scanf("%s", nome);

    if (nome[0] != letra) {   // se errou, remove
        aux = lista->prox;
        lista->prox = lista->prox->prox;
        free(aux);
    }
```

# Implementando jogo (cont.)

```
// enquanto existe mais de um jogador
while(lista->prox != lista) {
    // sorteia uma letra aleatória
    letra = rand() % 26 + 'a'; // por que isso funciona?
    printf("%s, escreva nome com %c: ", lista->prox->nome, letra);
    scanf("%s", nome);

    if (nome[0] != letra) {   // se errou, remove
        aux = lista->prox;
        lista->prox = lista->prox->prox;
        free(aux);
    } else {                  // se acertou, avança
        lista = lista->prox;
    }
}
```

# Implementando jogo (cont.)

```
    // enquanto existe mais de um jogador
    while(lista->prox != lista) {
        // sorteia uma letra aleatória
        letra = rand() % 26 + 'a'; // por que isso funciona?
        printf("%s, escreva nome com %c: ", lista->prox->nome, letra);
        scanf("%s", nome);

        if (nome[0] != letra) {   // se errou, remove
            aux = lista->prox;
            lista->prox = lista->prox->prox;
            free(aux);
        } else {                  // se acertou, avança
            lista = lista->prox;
        }
    }
    printf("Vencedor é %s!", lista->nome);
}
```

# Implementando jogo (cont.)

```
    // enquanto existe mais de um jogador
    while(lista->prox != lista) {
        // sorteia uma letra aleatória
        letra = rand() % 26 + 'a'; // por que isso funciona?
        printf("%s, escreva nome com %c: ", lista->prox->nome, letra);
        scanf("%s", nome);

        if (nome[0] != letra) {   // se errou, remove
            aux = lista->prox;
            lista->prox = lista->prox->prox;
            free(aux);
        } else {                  // se acertou, avança
            lista = lista->prox;
        }
    }
    printf("Vencedor é %s!", lista->nome);
}
```

- Como verificar se a palavra é repetida?

# Implementando jogo (cont.)

```
    // enquanto existe mais de um jogador
    while(lista->prox != lista) {
        // sorteia uma letra aleatória
        letra = rand() % 26 + 'a'; // por que isso funciona?
        printf("%s, escreva nome com %c: ", lista->prox->nome, letra);
        scanf("%s", nome);

        if (nome[0] != letra) {   // se errou, remove
            aux = lista->prox;
            lista->prox = lista->prox->prox;
            free(aux);
        } else {                  // se acertou, avança
            lista = lista->prox;
        }
    }
    printf("Vencedor é %s!", lista->nome);
}
```

- Como verificar se a palavra é repetida?
- Como você faria para verificar se a palavra é real? Sua solução seria rápida?

# III) Duplamente encadeada

## Percorrimento em duas direções:

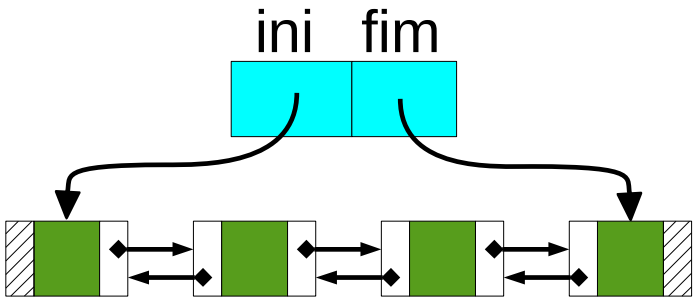
- histórico do navegador
- menu desfazer/refazer

# III) Duplamente encadeada

## Percorrimento em duas direções:

- histórico do navegador
- menu desfazer/refazer

Lista dupla:

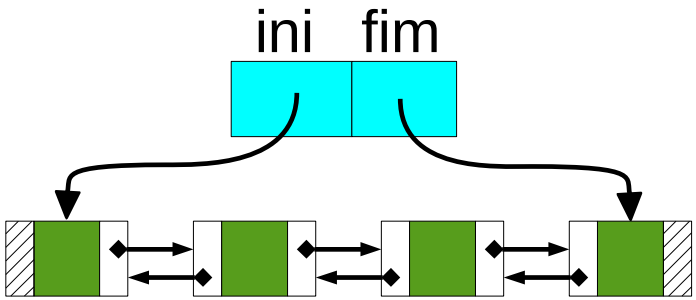


# III) Duplamente encadeada

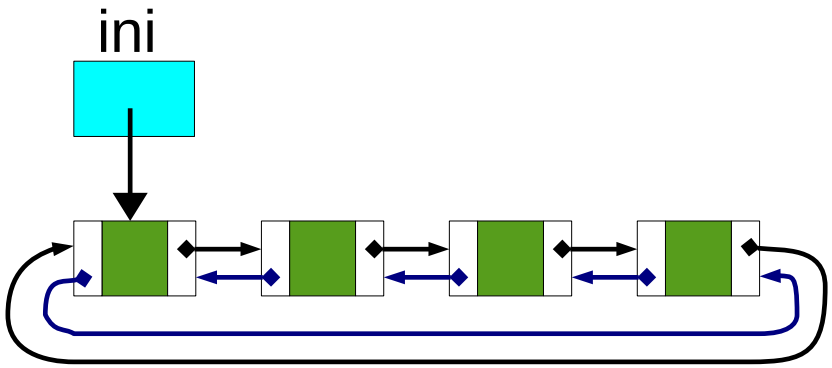
## Percorrimento em duas direções:

- histórico do navegador
- menu desfazer/refazer

Lista dupla:



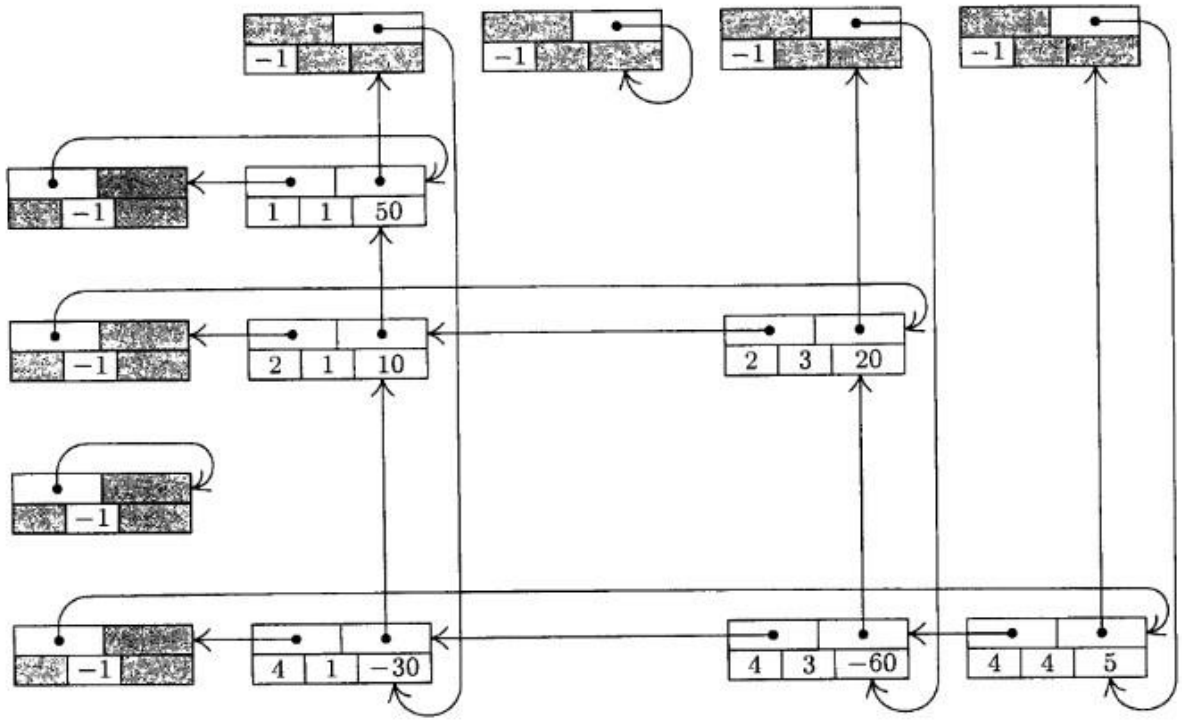

Lista dupla circular:



# Exercício 1

1. Implemente as operações faltantes do TAD lista descrito na aula
2. Implemente as operações *inserir elemento* e *remover elemento* de uma lista duplamente encadeada.
3. Para cada operação do TAD, escreva uma documentação: verifique como são passados os parâmetros (referência ou valor), se memória é alocada pela função, se algum ponteiro se torna inválido, etc.
4. Em uma lista encadeada simples com acesso ao nó inicial, quanto tempo se gasta para inserir no final da lista? Como seria possível inserir no final em tempo constante?

# Exercício 2 - Matriz esparsa de novo[1]




## Exercício

Utilizando o nó (`struct`) definido na aula passada, implemente a operação alterar valor da matriz equivante a

```
matrix[i][j] = x;
```

[1] Imagem do livro The Art of Computer Programming - I, Knuth.