Lista Simplesmente Encadeada

Estrutura de Dados — QXD0010

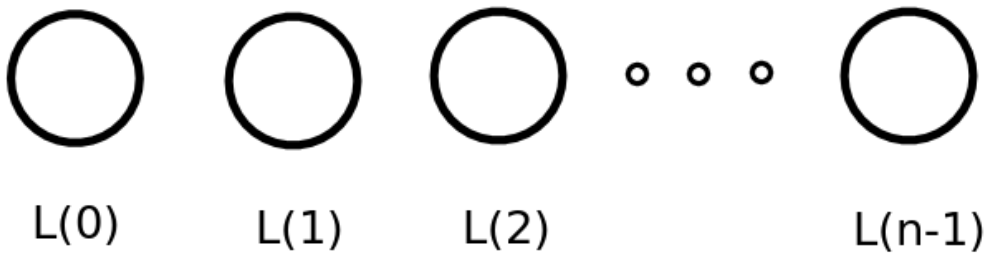
Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará
Campus Quixadá

2º semestre/2025

Estrutura de dados: Lista linear

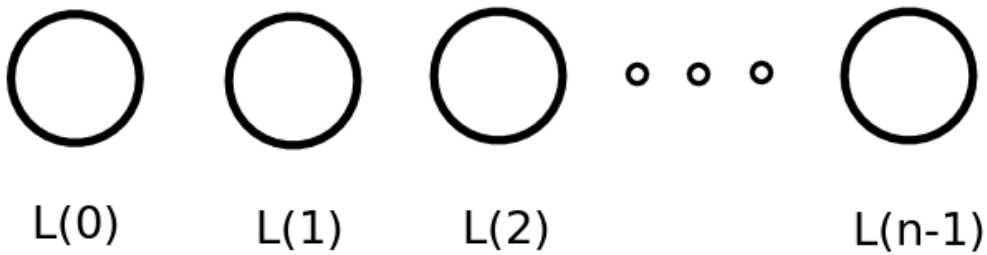
- Uma lista linear L é um conjunto de $n \geq 0$ elementos (nós) $L_0, L_1, \ldots, L_{n-1}$ tais que suas propriedades estruturais decorrem, unicamente, da posição relativa dos nós dentro da sequência linear:



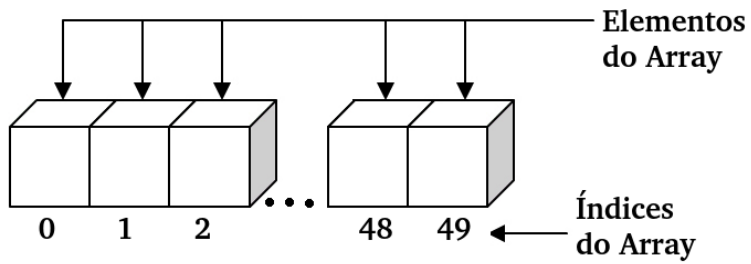
Estrutura de dados: Lista linear

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Uma lista linear L é um conjunto de $n \geq 0$ elementos (nós) $L_0, L_1, \ldots, L_{n-1}$ tais que suas propriedades estruturais decorrem, unicamente, da posição relativa dos nós dentro da sequência linear:



- Vimos que uma lista linear pode ser implementada por meio de um array usando alocação dinâmica de memória (alocação sequencial).



Alocação sequencial

Vantagens do uso de vetores:

- operações de acesso aos elementos são rápidas: $O(1)$
- poucos ponteiros: maior parte do espaço é utilizada para dados

Alocação sequencial

Vantagens do uso de vetores:

- operações de acesso aos elementos são rápidas: $O(1)$
- poucos ponteiros: maior parte do espaço é utilizada para dados

Desvantagens do uso de vetores:

- estão alocados contiguamente na memória
  - pode ser que tenhamos espaço na memória
  - mas não para alocar um vetor do tamanho desejado

Alocação sequencial

Vantagens do uso de vetores:

- operações de acesso aos elementos são rápidas: $O(1)$
- poucos ponteiros: maior parte do espaço é utilizada para dados

Desvantagens do uso de vetores:

- estão alocados contiguamente na memória
  - pode ser que tenhamos espaço na memória
  - mas não para alocar um vetor do tamanho desejado
- têm um tamanho fixo
  - ou alocamos um vetor pequeno e o espaço pode acabar
  - ou alocamos um vetor grande e desperdiçamos memória
  - Solução: criar lista sequencial redimensionável

Alocação sequencial

Vantagens do uso de vetores:

- operações de acesso aos elementos são rápidas: $O(1)$
- poucos ponteiros: maior parte do espaço é utilizada para dados

Desvantagens do uso de vetores:

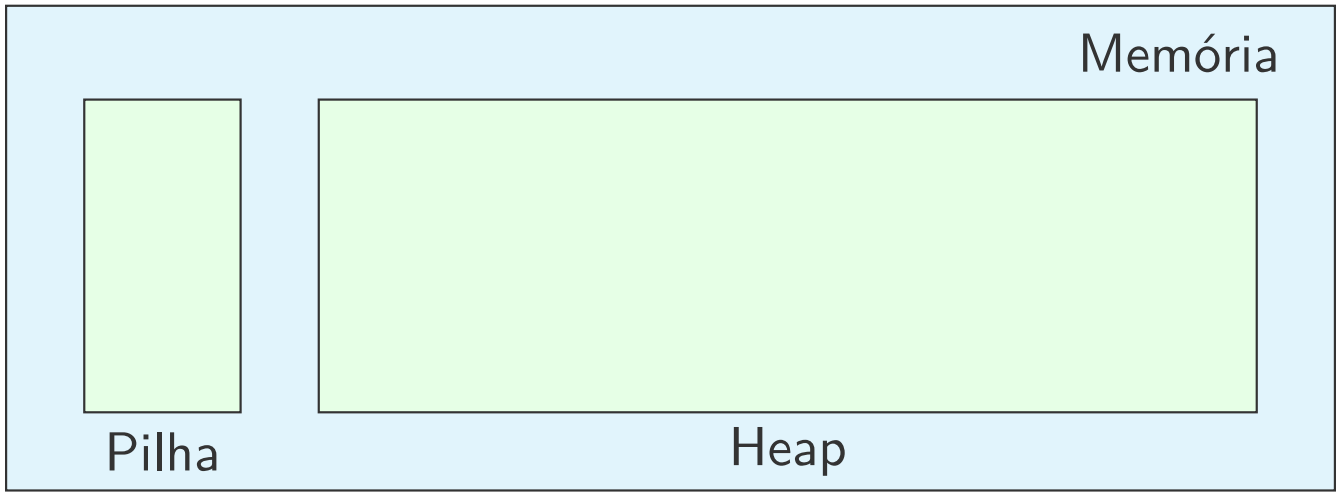
- estão alocados contiguamente na memória
  - pode ser que tenhamos espaço na memória
  - mas não para alocar um vetor do tamanho desejado
- têm um tamanho fixo
  - ou alocamos um vetor pequeno e o espaço pode acabar
  - ou alocamos um vetor grande e desperdiçamos memória
  - Solução: criar lista sequencial redimensionável
- inserção e remoção de elementos podem vir a ser custosas: $O(n)$

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

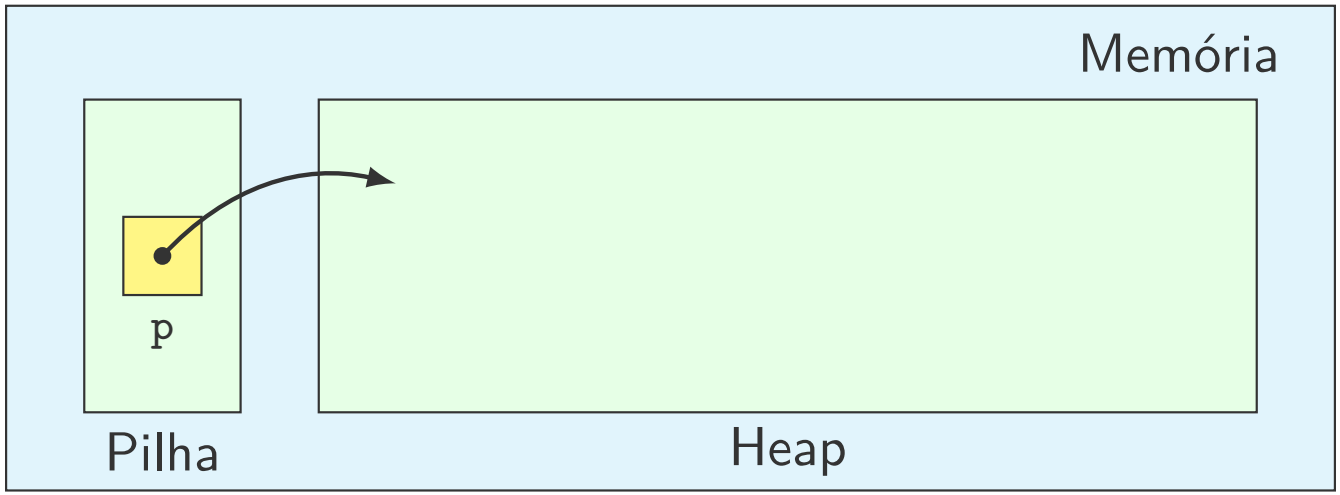
Listas Simplesmente Encadeadas

Lista Simplesmente Encadeada

Lista Simplesmente Encadeada

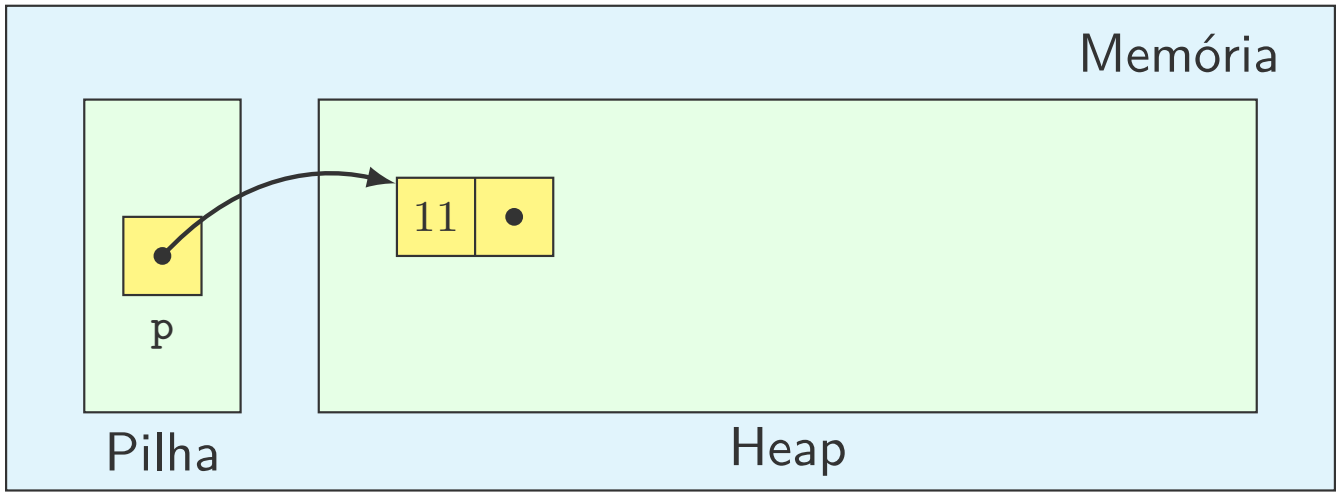


Lista Simplesmente Encadeada



- declaramos um ponteiro para a lista no nosso programa

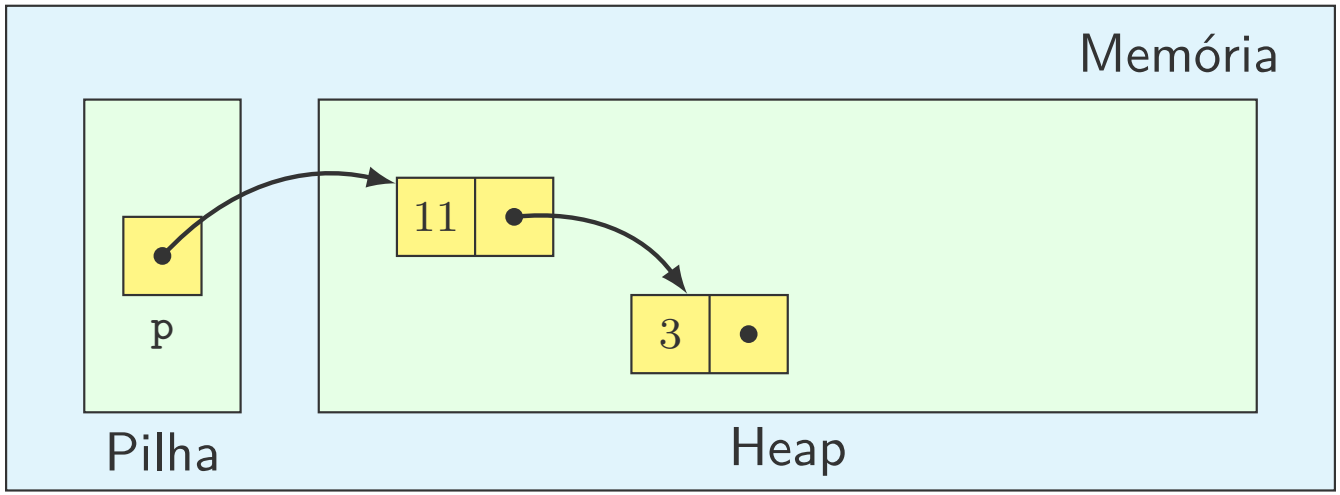
Lista Simplesmente Encadeada



- declaramos um ponteiro para a lista no nosso programa
- alocamos memória conforme o necessário

Lista Simplesmente Encadeada

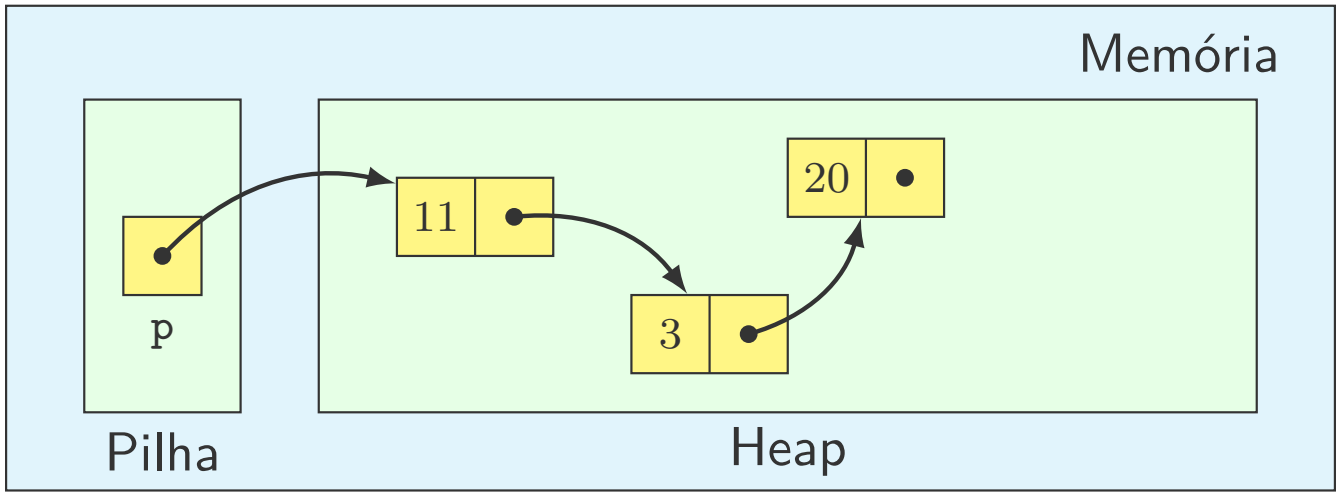
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- declaramos um ponteiro para a lista no nosso programa
- alocamos memória conforme o necessário
- o primeiro nó aponta para o segundo

Lista Simplesmente Encadeada

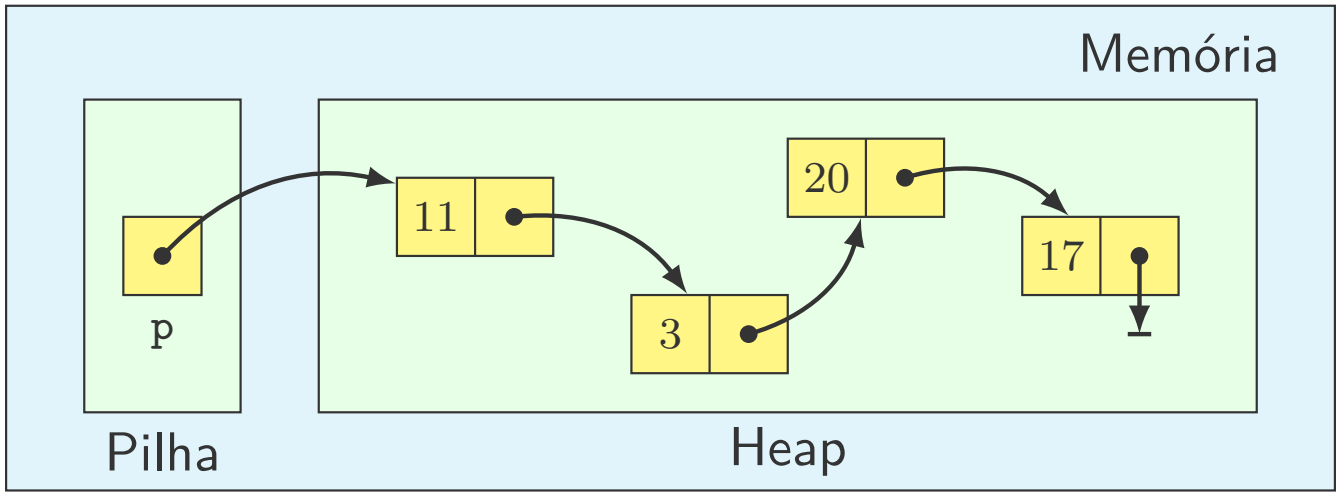
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- declaramos um ponteiro para a lista no nosso programa
- alocamos memória conforme o necessário
- o primeiro nó aponta para o segundo
- o segundo nó aponta para o terceiro

Lista Simplesmente Encadeada

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- declaramos um ponteiro para a lista no nosso programa
- alocamos memória conforme o necessário
- o primeiro nó aponta para o segundo
- o segundo nó aponta para o terceiro
- o último nó aponta para `NULL`

Lista com nó sentinela

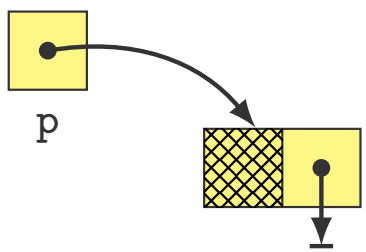
- Uma lista encadeada pode ser vista de duas maneiras diferentes, dependendo do papel que seu primeiro nó desempenha.

Lista com nó sentinela

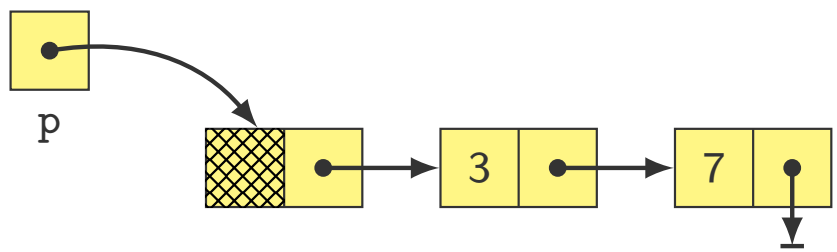
- Uma lista encadeada pode ser vista de duas maneiras diferentes, dependendo do papel que seu primeiro nó desempenha.
- Na **lista com nó sentinela**, o primeiro nó serve apenas para marcar o início da lista e portanto o seu conteúdo é irrelevante.

Lista com nó sentinela

- Uma lista encadeada pode ser vista de duas maneiras diferentes, dependendo do papel que seu primeiro nó desempenha.
- Na **lista com nó sentinela**, o primeiro nó serve apenas para marcar o início da lista e portanto o seu conteúdo é irrelevante.
- O ponteiro `p` **sempre** aponta para o nó sentinela.
- Quando a lista está vazia, o nó sentinela é o único nó na lista e seu campo `next` aponta para `NULL`.



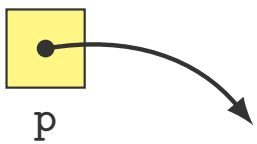
Lista vazia



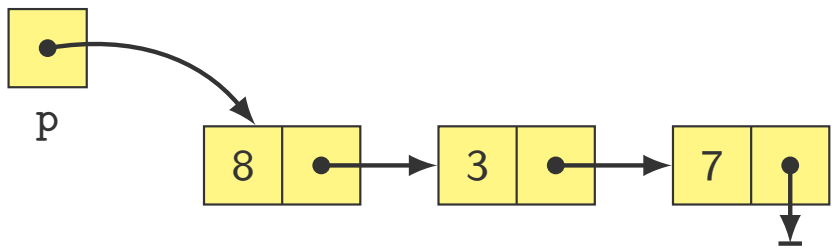
Lista com 2 elementos

Lista sem nó sentinela

- Na **lista sem nó sentinela**, o conteúdo do primeiro é tão relevante quanto o dos demais.
- Quando a lista está vazia, o ponteiro `p` aponta para `NULL`.



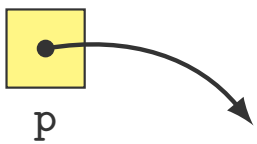
Lista vazia



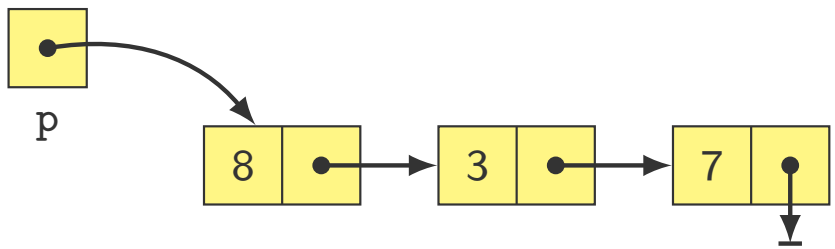
Lista com 3 elementos

Lista sem nó sentinela

- Na **lista sem nó sentinela**, o conteúdo do primeiro é tão relevante quanto o dos demais.
- Quando a lista está vazia, o ponteiro `p` aponta para `NULL`.



Lista vazia



Lista com 3 elementos

Nas aulas, trabalharei apenas com listas sem nó sentinela.

A implementação das funções em listas com nó sentinela ficam como exercício para casa.

Os nós da lista

Uma lista encadeada é formada por um conjunto de registros chamados **nós**.

Definição

Nó é um elemento alocado dinamicamente que contém:

- o dado armazenado
- um ponteiro para o nó seguinte na lista

Os nós da lista

Uma lista encadeada é formada por um conjunto de registros chamados **nós**.

Definição

Nó é um elemento alocado dinamicamente que contém:

- o dado armazenado
- um ponteiro para o nó seguinte na lista

- Um nó pode ser implementado como um `struct` na linguagem C.

Definição do tipo Node na linguagem C

Universidade Federal do Ceará
Campus Quixadá

```c
struct node {
    int data;
    struct node *next;
};

typedef struct node Node;
```

Definição do tipo `Node` na linguagem C

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```c
struct node {
    int data;
    struct node *next;
};

typedef struct node Node;
```

- Note que `Node` é um tipo recursivo.

Criando uma variável do tipo Node

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  int main() {
2      Node nó; // uma variável do tipo Node
3      Node *p; // um ponteiro para Node
4
5      nó.data = 7;
6      nó.next = NULL;
7
8      p = &nó;
9
10     printf("valor do nó = %d\n", p->data);
11
12     return 0;
13 }
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Manipulando Listas Encadeadas

Criando lista vazia

Criando lista vazia


```
1 Node* list_create(void) {
2     return NULL;
3 }
```

Destruindo uma lista

Função Recursiva

Destruindo uma lista

Função Recursiva


```
1 Node* list_free_rec(Node *p) {
2     if(p == NULL) {
3         return NULL;
4     } else {
5         p->next = list_free_rec(p->next);
6         free(p);
7         return NULL;
8     }
9 }
```

Destruindo uma lista

Função Iterativa

Destruindo uma lista

Função Iterativa


```
1 Node* list_free(Node *p) {
2     while(p != NULL) {
3         Node *temp = p;
4         p = p->next;
5         free(temp);
6     }
7     return NULL;
8 }
```

Inserindo no início da lista

Inserindo no início da lista


```
1  Node* list_push_front(Node *p, int value) {
2      Node *new_node = (Node*) malloc(sizeof(Node));
3      new_node->data = value;
4      new_node->next = NULL;
5      if(p == NULL) {
6          return new_node;
7      } else {
8          new_node->next = p;
9          p = new_node;
10         return p;
11     }
12 }
```

Inserindo no final da lista

Função Recursiva

Inserindo no final da lista

Função Recursiva


```
1  Node* list_push_back_rec(Node *p, int value) {
2      if(p == NULL) {
3          Node *new_node = (Node*) malloc(sizeof(Node));
4          new_node->data = value;
5          new_node->next = NULL;
6          return new_node;
7      } else {
8          p->next = list_push_back_rec(p->next, value);
9          return p;
10     }
11 }
```

Inserindo no final da lista

Função Iterativa

Inserindo no final da lista

Função Iterativa


```
1  Node* list_push_back(Node *p, int value) {
2      Node *new_node = (Node*) malloc(sizeof(Node));
3      new_node->data = value;
4      new_node->next = NULL;
5
6      if(p == NULL) return new_node;
7
8      Node *temp = p;
9      while(temp->next != NULL) {
10         temp = temp->next;
11     }
12     temp->next = new_node;
13     return p;
14 }
```

Imprimir lista

Função Recursiva

Imprimir lista

Função Recursiva


```
1 void list_print_rec(Node *p) {
2     if(p != NULL) {
3         printf("%d ", p->data);
4         list_print_rec(p->next);
5     }
6 }
```

Imprimir lista

Função Iterativa

Imprimir lista

Função Iterativa


```
1 void list_print(Node *p) {
2     while(p != NULL) {
3         printf("%d ", p->data);
4         p = p->next;
5     }
6 }
```

Exercícios

Exercícios

Escreva em C funções que resolvam os seguintes problemas em listas encadeadas:

(1) determinar o tamanho de uma lista (número de nós)

(2) somar os elementos com valor par

(3) procurar um elemento com valor k na lista e retornar `true` se ele estiver na lista

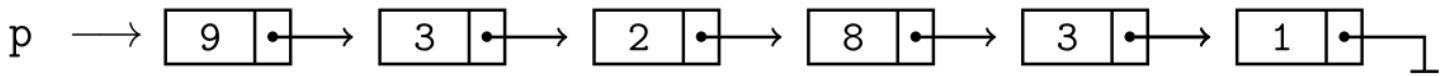
(4) encontrar o maior elemento da lista

(5) imprimir os elementos das posições ímpares da lista

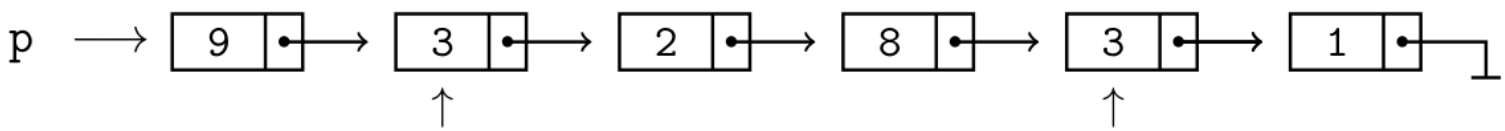
Exercício 6

Procurando um elemento repetido

- Imagine que o ponteiro **p** aponta para uma lista encadeada



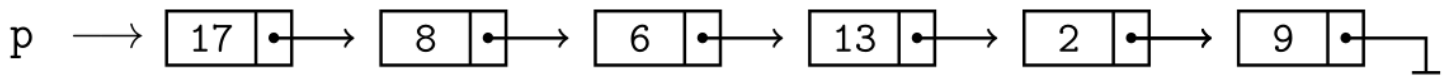
- A tarefa é **Verificar se a lista contém algum elemento repetido**
- No exemplo acima, isso nos daria a resposta **true**.



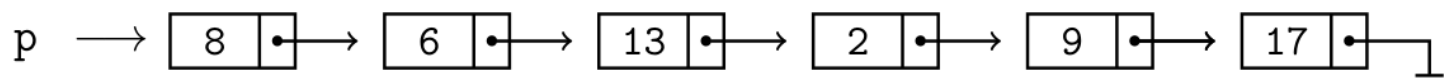
Exercício 7

Movendo o primeiro para o fim

- Imagine que o ponteiro **p** aponta para uma lista encadeada



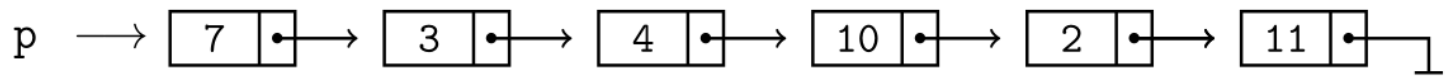
- A tarefa é **Mover o primeiro elemento para o final da lista**
- No exemplo acima, isso nos daria a lista abaixo:



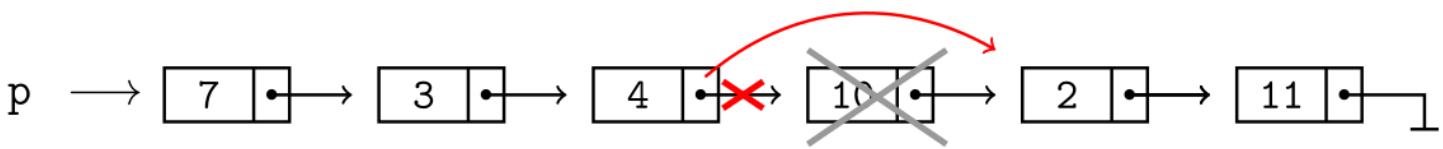
Exercício 8

Removendo o elemento k

- Imagine que o ponteiro **p** aponta para uma lista encadeada



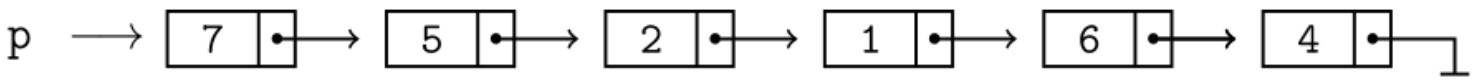
- A tarefa é **emover o elemento k da lista**
- Abaixo está um exemplo de remoção do $k = 10$:



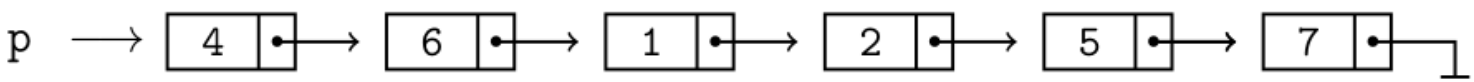
Exercício 9

Invertendo a lista encadeada

- Imagine que o ponteiro **p** aponta para uma lista encadeada



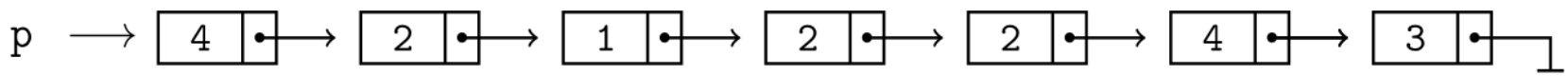
- A tarefa é **inverter a ordem dos elementos da lista**
- No exemplo acima, isso nos daria a lista abaixo:



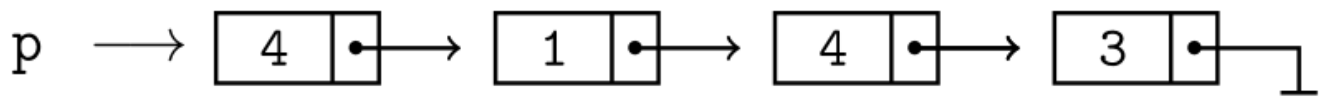
Exercício 10

Removendo elementos repetidos

- Imagine que o ponteiro p aponta para uma lista encadeada que pode conter elementos repetidos



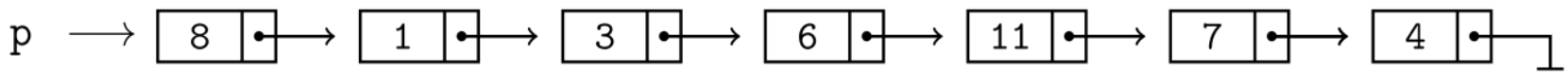
- A tarefa é **remover todas as cópias do número k**
- No exemplo acima, para $k = 2$ isso nos daria a lista abaixo:



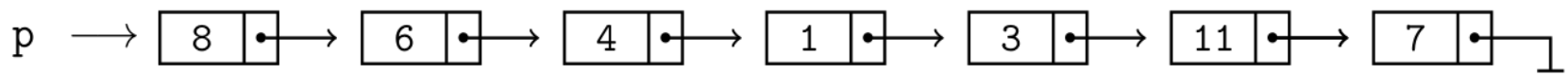
Exercício 11

Separando pares e ímpares

- Imagine que o ponteiro **p** aponta para uma lista encadeada



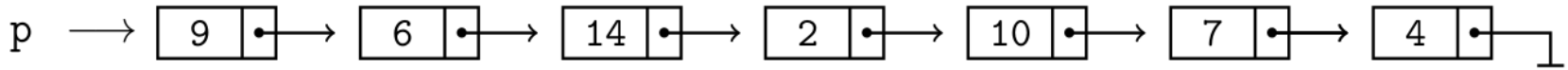
- A tarefa é **Colocar os elementos pares no início, e os elementos ímpares no fim da lista**
- No exemplo acima, ficaríamos com a lista abaixo:



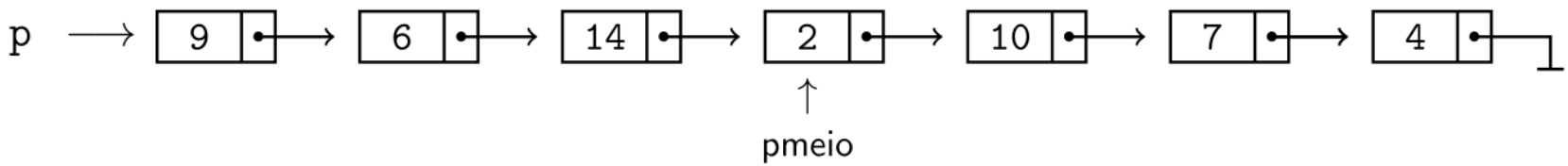
Exercício 12

Encontrando o elemento do meio

- Imagine que o ponteiro **p** aponta para uma lista encadeada



- A tarefa é **Encontrar o elemento central da lista e retornar um ponteiro para ele**
- No exemplo acima, isso nos daria:



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM