Programação Procedimental

Listas Ligadas

Aula 09

Prof. Felipe A. Louza

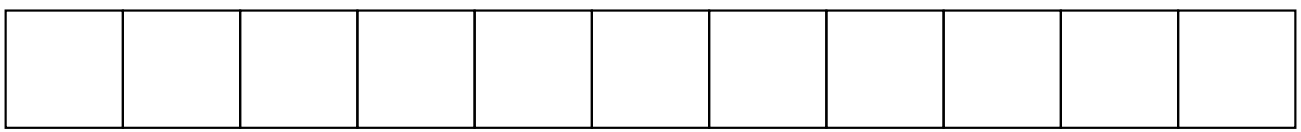
Roteiro

1. Introdução
2. Implementação
3. Inserção
4. Remoção
5. Desalocar espaço
6. Busca
7. Lista ordenada
8. Referências

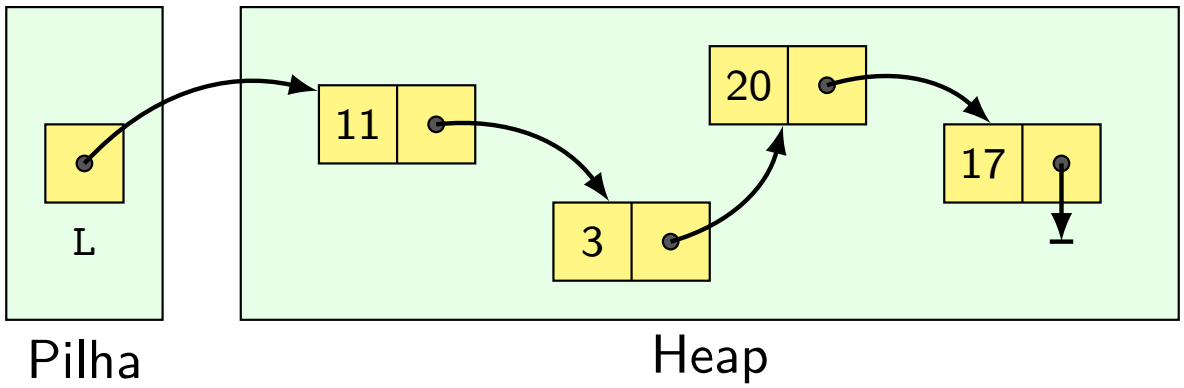
Vetores

Vetores:

- estão alocados contiguamente na memória
  - pode ser que tenhamos espaço na memória, mas não para alocar um vetor do tamanho desejado
- tem um tamanho fixo
  - ou alocamos um vetor grande e desperdiçamos memória
  - ou alocamos um vetor pequeno e o espaço pode acabar



Alternativa - Lista Ligada



- alocamos memória conforme o necessário
- guardamos um **ponteiro** para a estrutura em uma variável
- o primeiro **nó** aponta para o segundo
- o segundo nó aponta para o terceiro
- o último nó aponta para `NULL`

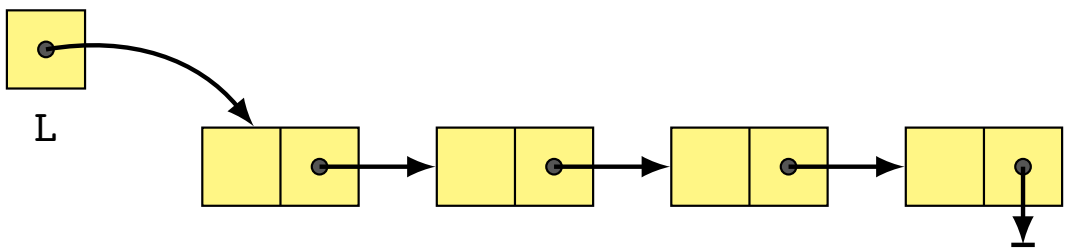
Listas ligadas

Nó: elemento alocado dinamicamente que contém

- um conjunto de dados
- um ponteiro para outro nó

Lista ligada:

- Conjunto de nós ligados entre si de maneira sequencial



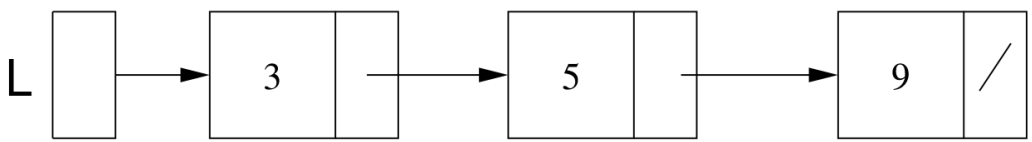
Observação:

- a lista ligada é acessada a partir de `L`

Introdução

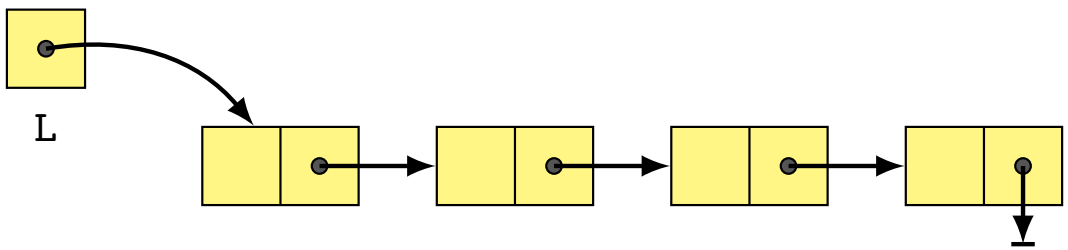
Quando utilizamos listas ligadas?

- Em geral, quando não sabemos antecipadamente que tamanho a coleção pode alcançar.
- Permitem inserções/remoções de nós em qualquer posição.
- Não permitem acesso direto a um nó.



Introdução

Listas ligadas (ou lista encadeada ou simplesmente lista).



- Listas podem ser usadas para implementar muitas estruturas de dados, como filas, pilhas, grafos etc.

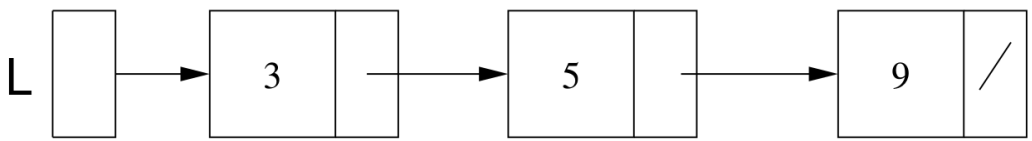
Roteiro

1. Introdução
2. Implementação
3. Inserção
4. Remoção
5. Desalocar espaço
6. Busca
7. Lista ordenada
8. Referências

Listas ligadas

Definição do no:

- um ou mais campos de informação (`valor`) e
- um apontador para o próximo nó da lista.

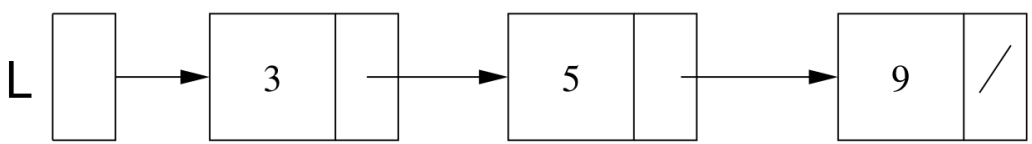



```
1  struct no {
2    int valor;
3    struct no* prox;
4  };
5
6  typedef struct no No;
```

Listas ligadas

Importante:

- A figura pode dar a falsa impressão de que os nós da lista ocupam posições consecutivas na memória.
- Na realidade, as células estão tipicamente espalhadas pela memória.

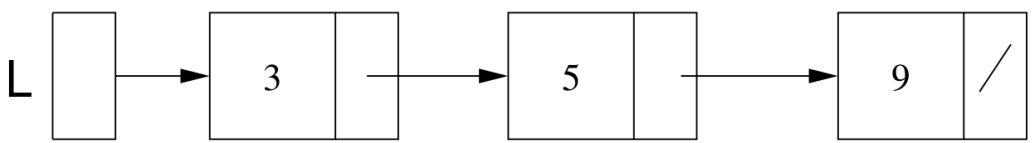


Listas ligadas

Endereço de uma lista ligada:

- O endereço de uma lista é o endereço de sua primeira célula (armazenado em `L`).
- A lista está vazia (ou seja, não tem célula alguma) se e somente se `L == NULL`.

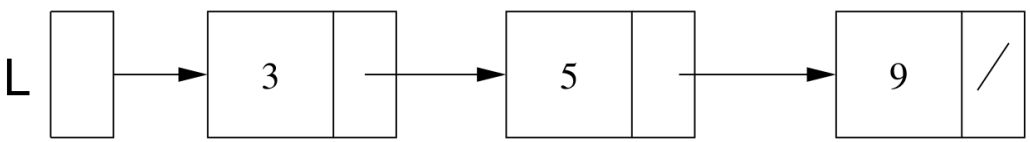
```
1   No *L = NULL;
```



Busca em uma lista encadeada

Como percorrer uma lista ligada?

```
1  void imprime_lista(No *q) {// q recebe L
2    while(q != NULL){
3      printf("%d\n", q->valor);
4      q = q->prox;
5    }
6  }
```



Listas ligadas

Próximas aulas:

1. Inserção
2. Remoção
3. Busca

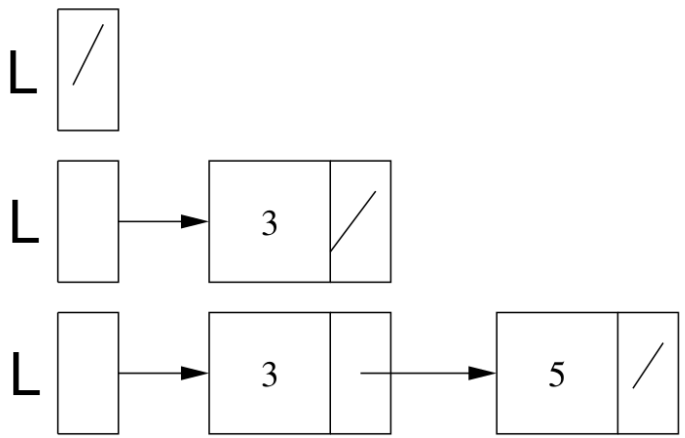
Roteiro

1. Introdução
2. Implementação
3. Inserção
4. Remoção
5. Desalocar espaço
6. Busca
7. Lista ordenada
8. Referências

Inserção no final da lista

Como inserir no final da lista:

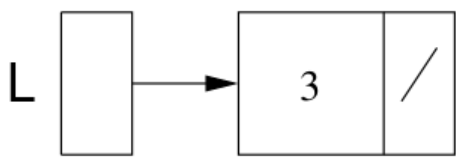
- Percorre a lista até o final e insere um novo nó.



Inserção no final da lista

Percorre a lista até o final e insere um novo nó.

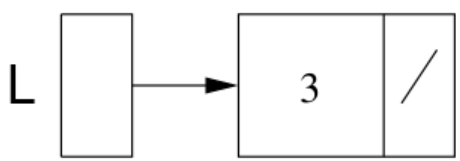
```
1   void insere_final(No** p, int v) {// p recebe &L
2     No *aux, *q;
3     q = (No*) malloc(sizeof(No));
4     q->valor = v;
5     q->prox = NULL;  /* Ultimo elemento da lista */
6     if (*p == NULL) *p = q;
7     else {
8       aux = *p;
9       while(aux->prox != NULL) aux = aux->prox;
10      aux->prox = q;
11    }
12  }
```



Inserção no final da lista

Função principal:

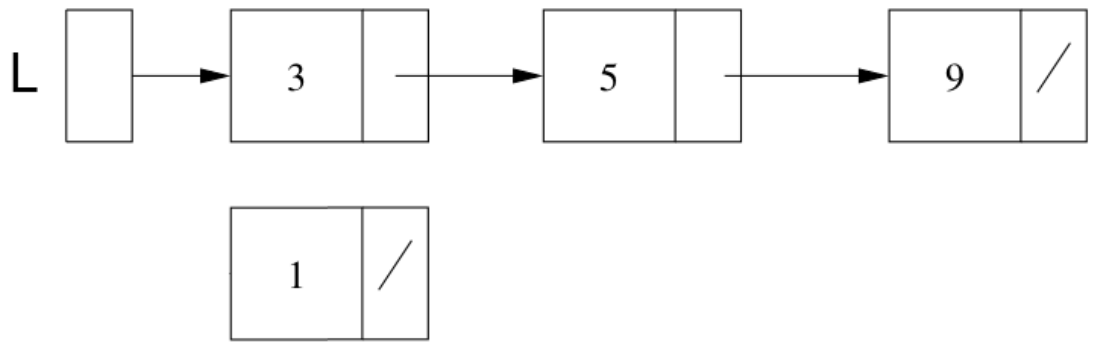
```c
#include <stdlib.h>
...
void insere_final(No** p, int v);
...
int main(){
  No* L = NULL;

  insere_final(&L, 3);
  ...

return 0;
}
```



Inserção no começo da lista

Como inserir no início da lista:

- Aloca novo espaço para q.
- Atualiza L.

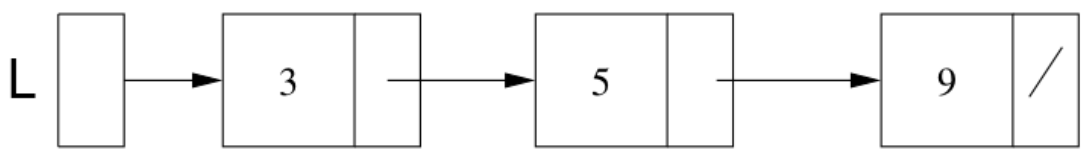


Inserção no começo da lista

Inserir no começo da lista:

- Aloca novo espaço para q, e aponta q->prox para p
- Atualiza o ponteiro L para q.

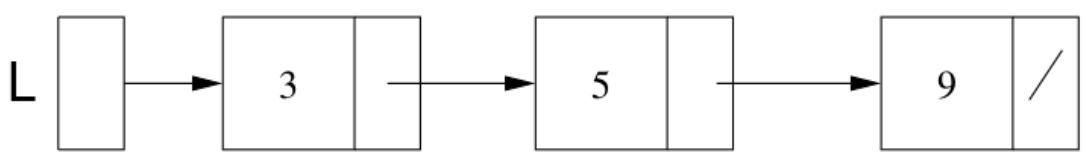
```
1  void insere_comeco(No** p, int valor) {// p recebe &L
2    No* q;
3    q = (No*) malloc(sizeof(No));
4    q->valor = valor;
5    q->prox = *p;
6    *p = q;
7  }
```



Inserção no começo da lista

Função principal:

```c
#include <stdlib.h>
...
void insere_comeco(No** p, int v);
...
int main(){
  No* p = NULL;

  insere_comeco(&L, 9);
  insere_comeco(&L, 5);
  insere_comeco(&L, 3);

return 0;
}
```



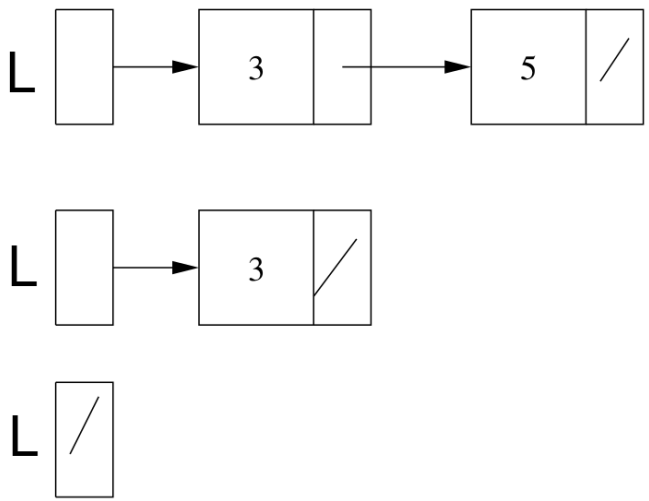
Roteiro

1. Introdução
2. Implementação
3. Inserção
4. Remoção
5. Desalocar espaço
6. Busca
7. Lista ordenada
8. Referências

Remoção no final da lista

Como remover no final da lista:

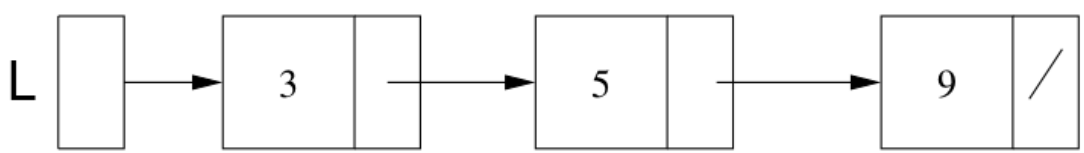
- Percorre a lista até o final e remove o último nó.



Remoção no final da lista

Percorre a lista até o final e remove o último nó.

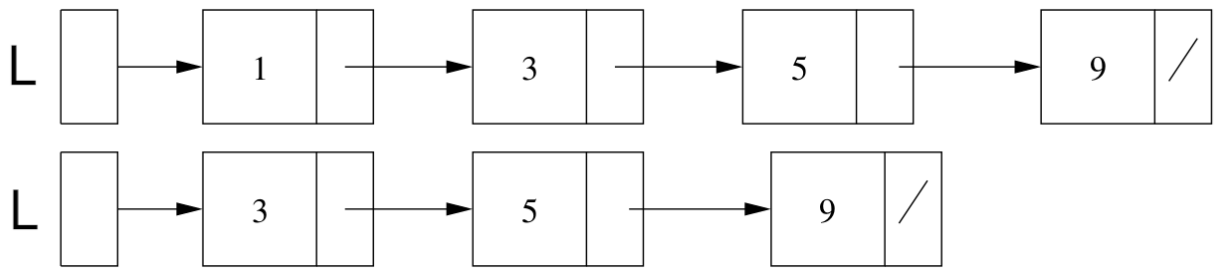
```
1  void remove_final(No** p) {// p recebe &L
2    No* q = *p;
3    if(q==NULL) return; //lista vazia
4    if(q->prox==NULL){ //apenas 1 elemento
5      *p = NULL;
6      free(q);
7      return;
8    }
9    while (q->prox->prox != NULL)  q = q->prox;
10   free(q->prox);
11   q->prox = NULL;
12 }
```



Remoção no começo da lista

Como remover no inicio da lista:

- O endereço armazenado em L deve ser alterado

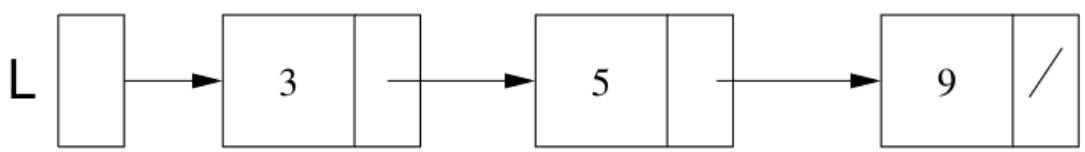


Remoção no começo da lista

Remoção no começo da lista:

- Atualiza o endereço da lista (ponteiro `L`).
- Libera o espaço do primeiro nó.

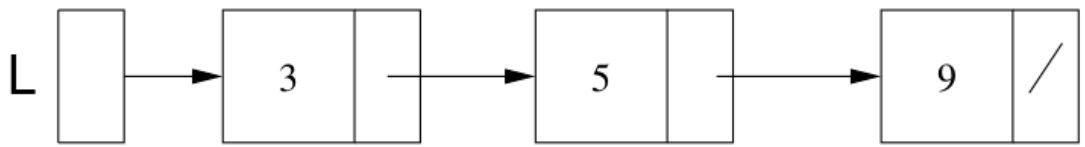
```
1  void remove_comeco(No** p) {// p recebe &L
2    No* q = *p;
3    if(q==NULL) return; //lista vazia
4    *p = q->prox;
5    free(q);
6  }
```



Remoção no meio da lista

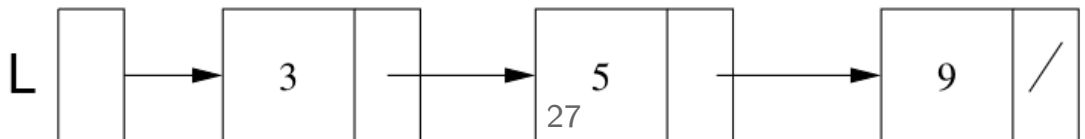
Como remover um valor `x` no meio da lista:

- Podemos modificar a função `remove_final`:



Remoção no meio da lista

```
1  void remove_valor(No** p, int v) {// p recebe &L
2    No* q = *p;
3    if(q==NULL) return; //lista vazia
4    if(q->valor==v){ //encontrou no 1o elemento
5      *p = q->prox;
6      free(q);
7      return;
8    }
9    while (q->prox != NULL){
10     if(q->prox->valor==v) break;
11     q = q->prox;
12   }
13   if(q->prox==NULL) return; //não encontrou o valor
14   No* tmp = q->prox;
15   q->prox = tmp->prox;
16   free(tmp);
17 }
```



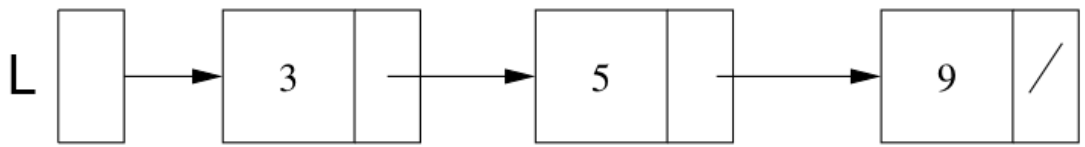
Roteiro

1. Introdução
2. Implementação
3. Inserção
4. Remoção
5. Desalocar espaço
6. Busca
7. Lista ordenada
8. Referências

Como liberar todo o espaço de uma lista

Como liberar todo o espaço de uma lista:

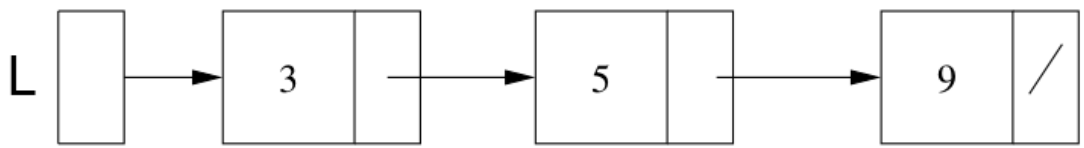
- **Atenção:** o comando `free(L)` libera apenas o primeiro nó.
- É nessário percorrer a lista liberando todos os nós.



Como liberar todo o espaço de uma lista

Percorrer a lista liberando todos os nós.

```
1  void libera_lista(No** p) {// p recebe &L
2    No* q;
3    while (*p != NULL) {
4      q = *p;
5      *p = (*p)->prox;
6      free(q);
7    }
8  }
```



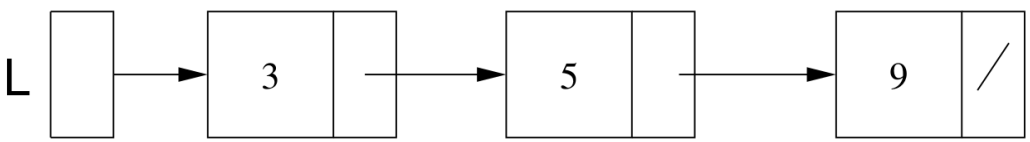
Roteiro

1. Introdução
2. Implementação
3. Inserção
4. Remoção
5. Desalocar espaço
6. Busca
7. Lista ordenada
8. Referências

Busca em uma lista ligada

Já vimos como percorrer uma lista ligada:

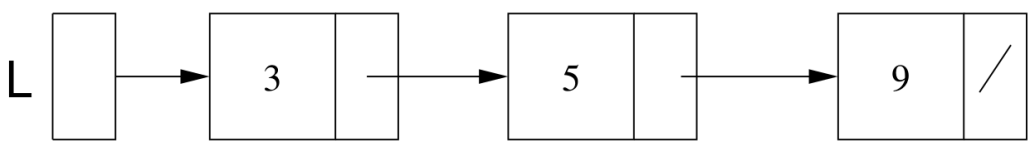
```
1  void imprime_lista(No *q) {// q recebe L
2    while(q != NULL){
3      printf("%d\n", q->valor);
4      q = q->prox;
5    }
6  }
```



Busca em uma lista ligada

Como verificar se um elemento x está na lista?

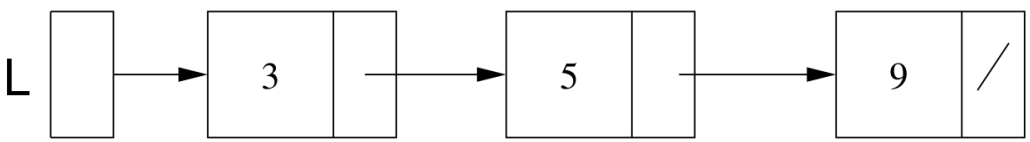
```
1  //retorna (1 == true) se x existe na lista ou (0 == false), caso contrário
2  int busca_lista(No *q, int x) {// q recebe L
3    while(q != NULL){
4      if(q->valor == x) return 1; //true!
5      q = q->prox;
6    }
7  return 0;//false == não encontrou
8  }
```



Busca em uma lista ligada

Como contar quantos elementos iguais à x estão lista?

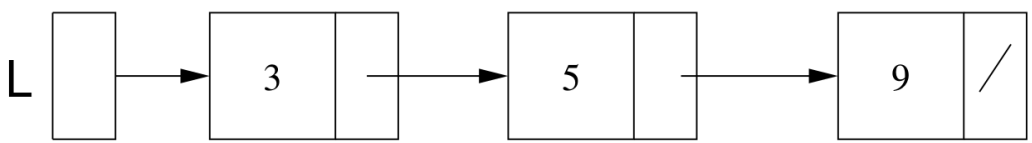
```
1  //retorna a quantidade de elementos iguais a x na lista
2  int contar_lista(No *q, int x) {// q recebe L
3    int sum=0;
4    while(q != NULL){
5      if(q->valor == x) sum++;
6      q = q->prox;
7    }
8  return sum;
9  }
```



Busca em uma lista ligada

Como encontrar a posição do primeiro elemento `x` na lista?

```
1   //retorna a posição do primeiro elemento igual a x na lista, ou -1.
2   int posicao_lista(No *q, int x) {// q recebe L
3     int pos=0;
4     while(q != NULL){
5       if(q->valor == x) return pos;
6       q = q->prox;
7       pos++;
8     }
9   return -1;//não encontrou
10  }
```



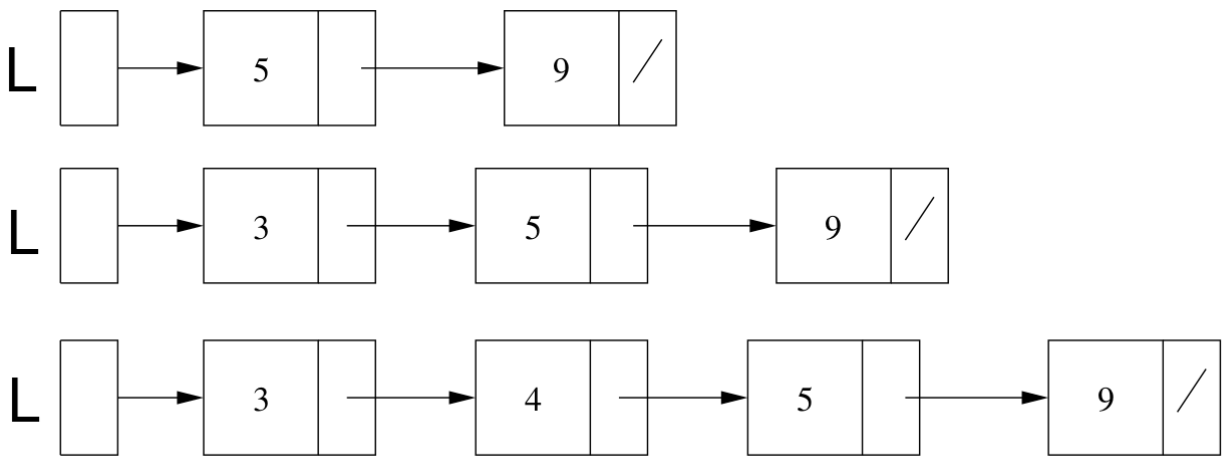
Roteiro

1. Introdução
2. Implementação
3. Inserção
4. Remoção
5. Desalocar espaço
6. Busca
7. Lista ordenada
8. Referências

Lista ordenada

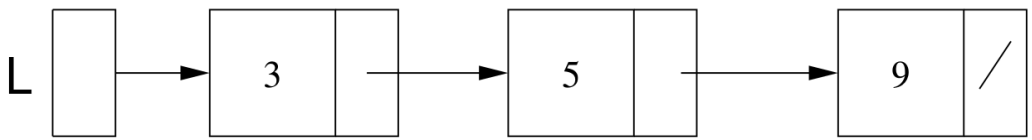
Lista ordenada:

- Os elementos são mantidos de forma ordenada na lista.



Lista ordenada

Busca em uma lista ordenada:

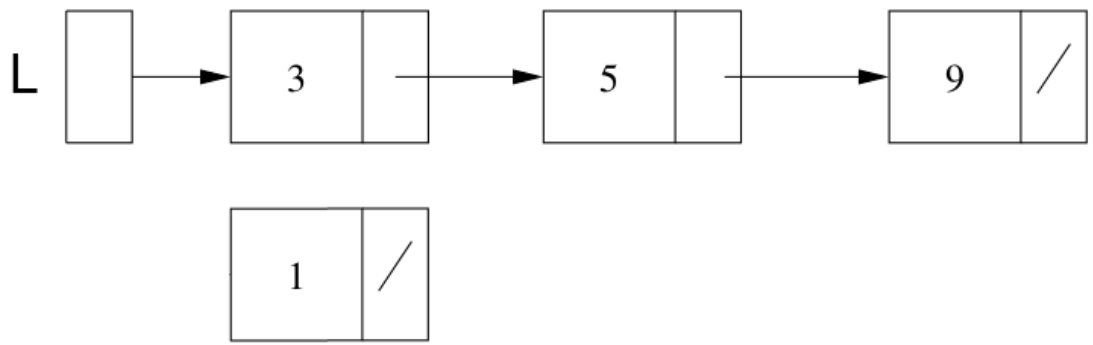


- A busca pode ser interrompida quando:
  - O elemento foi encontrado ou
    - encontramos **um elemento maior** que o desejado

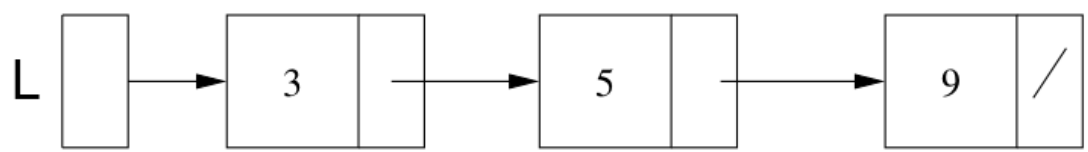
Inserção ordenada

Como inserir e manter uma lista ordenada:

- Percorre a lista até encontrar $q- > valor > x$, ou o final da lista.
- Insere o novo campo na lista, e atualiza os ponteiros.



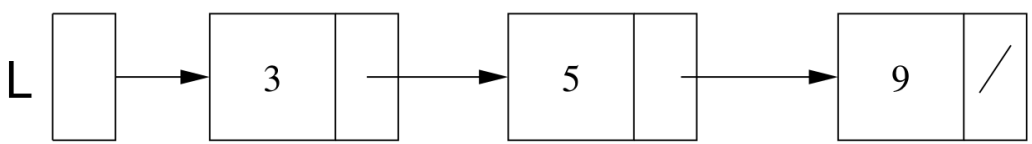
```c
void insere_ordenado(No** p, int x) {// p recebe &L
  No *q = (No*) malloc(sizeof(No)); //novo nó
  q->valor = x;
  q->prox = NULL;
  if (*p == NULL) *p = q; //lista vazia
  else{
    No *aux = *p;
    if(aux->valor > x){ //verifica o primeiro elemento
      *p = q;
      q->prox = aux;
      return;
    }
    while(aux->prox!=NULL){ //percorre a lista
      if(aux->prox->valor > x) break;
      aux = aux->prox;
    }
    q->prox = aux->prox;
    aux->prox = q;
  }
}
```



Busca em uma lista ordenada

Como verificar se um elemento x está na lista ordenada?

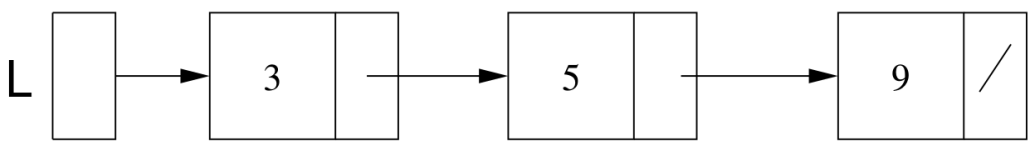
```
1  //retorna (1 == true) se x existe na lista ou (0 == false), caso contrário
2  int busca_lista_ordenada(No *q, int x) {// q recebe L
3    while(q != NULL){
4      if(q->valor == x) return 1; //true!
5      else if(q->valor > x) return 0; //false
6      q = q->prox;
7    }
8  return 0;//false == não encontrou
9  }
```



Busca em uma lista ordenada

Como contar quantos elementos iguais à x estão lista ordenada?

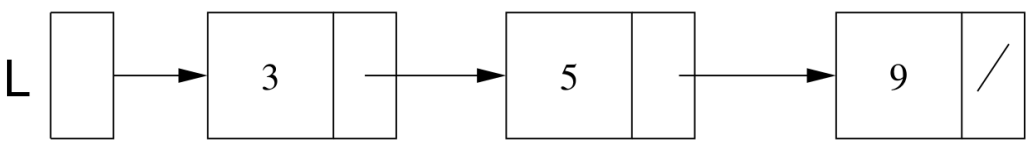
```
1   //retorna a quantidade de elementos iguais a x na lista
2   int contar_lista_ordenada(No *q, int x) {
3     int sum=0;
4     while(q != NULL){
5       if(q->valor == x) sum++;
6       else if(q->valor > x) return sum;
7       q = q->prox;
8     }
9   return sum;
10  }
```



Busca em uma lista ordenada

Como encontrar a posição do primeiro elemento `x` na lista ordenada?

```
1  //retorna a posição do primeiro elemento igual a x na lista, ou -1.
2  int posicao_lista_ordenada(No *q, int x) {
3    int pos=0;
4    while(q != NULL && q->valor <= x){
5      if(q->valor == x) return pos;
6      else if(q->valor > x) return -1; //não encontrou
7      q = q->prox;
8      pos++;
9    }
10 return -1;//não encontrou
11 }
```



Fim

Dúvidas?

Roteiro

1. Introdução
2. Implementação
3. Inserção
4. Remoção
5. Desalocar espaço
6. Busca
7. Lista ordenada
8. Referências

Referências

1. Feofiloff, Paulo. Algoritmos em linguagem C. Elsevier Brasil, 2009.
2. Materiais adaptados dos slides dos Profs. Rafael Schouery e Lehilton L. C. Pedrosa, da UNICAMP.