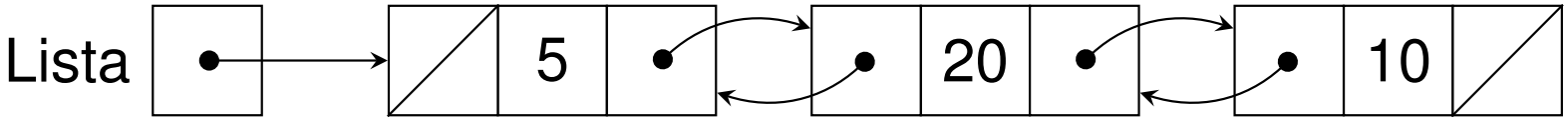
Estrutura de Dados I

Luciana Lee

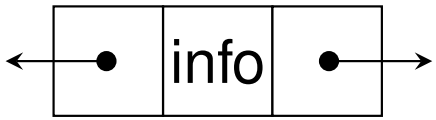
Tópicos da Aula

Lista Duplamente Encadeada



- Um nó da lista possui três campos:

- a informação armazenada;
- o ponteiro para o elemento sucessor na lista;
- o ponteiro para o elemento antecessor na lista.

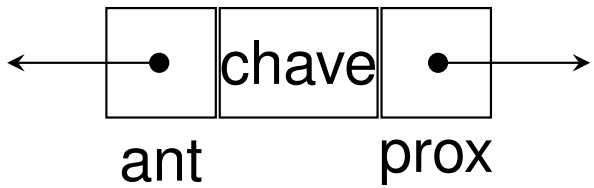


- A lista é representada por um ponteiro para o primeiro nó da lista;
- O primeiro elemento possui antecessor igual a `NULL`;
- O último elemento da lista possui sucessor igual a `NULL`.

Estrutura

- A estrutura de uma lista duplamente encadeada pode ser implementada da seguinte forma:

```
typedef struct no {
   int chave;
   struct no *prox, *ant;
}No;
```



Busca na Lista

- Veremos duas versões de algoritmo de busca:
  - Para listas não ordenadas;
  - Para listas ordenadas.
- Quando trabalhamos com listas simplesmente encadeadas, utilizamos um ponteiro para guardar o predecessor do ponteiro aux na busca.
- Quando a lista é duplamente encadeada, não precisamos mais do ponteiro para o predecessor de aux.

Busca em Lista não Ordenada

```
No *buscaLista (No *L, int ch) {
  No *aux = L;
  while (aux != NULL && ch != aux->chave) {
    if (aux->prox == NULL) break;
    aux = aux->prox;
  }
  return aux;
}
```

- Casos:
  - Lista vazia
  - O elemento existe na lista
  - O elemento não existe na lista

Busca em Lista Ordenada

```
No *buscaOrd (No *L, int ch) {
  No *aux = L;
  while (aux != NULL && ch > aux->chave) {
    if (aux->prox == NULL) break;
    aux = aux->prox;
  }
  return aux;
}
```

- Casos:
  - Lista vazia
  - O elemento existe na lista
  - O elemento não existe na lista

Inserção no Início da Lista

```
No *insereInicio (No *L, int ch) {
  No *novo = criaNo(ch);
  novo->prox = L;
  if (L != NULL) L->ant = novo;
  return novo;
}
```

- Casos:
  - Lista vazia
  - Lista não vazia

Inserção no Final da Lista

```
No *insereFinal (No *L, int ch) {
  No *novo = criaNo(ch);
  No *aux = L;
  if (L == NULL) L = novo;
  else {
    while (aux->prox != NULL)
      aux = aux->prox;
    aux->prox = novo;
    novo->ant = aux;
  }
  return L;
}
```

- Casos:
  - Lista vazia
  - Lista não vazia

Inserção em uma Lista Ordenada

- Casos:
  - Lista vazia
  - A nova chave é maior que todos da lista
  - A nova chave é menor que todos da lista
  - A chave é inserida no “meio” da lista

Inserção em uma Lista Ordenada

```
No *insereOrd (No *L, int ch){
  No *novo = criaNo(ch);
  No *aux = buscaOrd(L, ch);
  if (aux == NULL) L = novo;
```

- Casos:
  - Lista vazia
  - A nova chave é maior que todos da lista
  - A nova chave é menor que todos da lista
  - A chave é inserida no “meio” da lista

Inserção em uma Lista Ordenada

```
No *insereOrd (No *L, int ch){
  No *novo = criaNo(ch);
  No *aux = buscaOrd(L, ch);
  if (aux == NULL) L = novo;
  else {
    if (aux->chave < ch) {
      aux->prox = novo;
      novo->ant = aux;
```

- Casos:
  - Lista vazia
  - A nova chave é maior que todos da lista
  - A nova chave é menor que todos da lista
  - A chave é inserida no “meio” da lista

Inserção em uma Lista Ordenada

```
No *insereOrd (No *L, int ch){
  No *novo = criaNo(ch);
  No *aux = buscaOrd(L, ch);
  if (aux == NULL) L = novo;
  else {
    if (aux->chave < ch) {
      aux->prox = novo;
      novo->ant = aux;
    } else {
      novo->prox = aux;
      if (aux->ant == NULL) L = novo;
      else {
```

- Casos:
  - Lista vazia
  - A nova chave é maior que todos da lista
  - A nova chave é menor que todos da lista
  - A chave é inserida no “meio” da lista

Inserção em uma Lista Ordenada

```
No *insereOrd (No *L, int ch){
  No *novo = criaNo(ch);
  No *aux = buscaOrd(L, ch);
  if (aux == NULL) L = novo;
  else {
    if (aux->chave < ch) {
      aux->prox = novo;
      novo->ant = aux;
    } else {
      novo->prox = aux;
      if (aux->ant == NULL) L = novo;
      else {
        aux->ant->prox = novo;
        novo->ant = aux->ant;
      }
      aux->ant = novo;
    }
  }
  return L;
}
```

- Casos:
  - Lista vazia
  - A nova chave é maior que todos da lista
  - A nova chave é menor que todos da lista
  - A chave é inserida no “meio” da lista

Exclusão do Primeiro da Lista

```
No *excluiInicio (No *L) {
  No *aux = L;
  if (L == NULL) return NULL;
  L = aux->prox;
  if (L != NULL) L->ant = NULL;
  free(aux);
  return L;
}
```

- Casos:
  - Lista vazia
  - Lista não vazia com um único elemento
  - Lista não vazia com mais de um elemento

Exclusão do Último da Lista

```
No *excluiFinal (No *L) {
  No *aux = L;
  if (L == NULL) return NULL;
  while (aux->prox != NULL)
    aux = aux->prox;
  if (aux->ant == NULL)
    L = NULL;
  else
    aux->ant->prox = NULL;
  free(aux);
  return L;
}
```

- Casos:
  - Lista vazia
  - Lista não vazia com um único elemento
  - Lista não vazia com mais de um elemento

Exclusão de uma Chave da Lista

- Casos:
  - Lista vazia ou chave inexistente
  - A chave está no primeiro elemento da lista
  - A chave está no “meio” da lista
  - A chave está no último elemento da lista

Exclusão de uma Chave da Lista

```
No *excluiChave (No *L, int ch) {
  No *aux = buscaLista(L, ch);
  if (aux == NULL || aux->chave != ch)
    printf("Chave inexistente.\n");
```

- Casos:
  - Lista vazia ou chave inexistente
  - A chave está no primeiro elemento da lista
  - A chave está no “meio” da lista
  - A chave está no último elemento da lista

Exclusão de uma Chave da Lista

```
No *excluiChave (No *L, int ch) {
  No *aux = buscaLista(L, ch);
  if (aux == NULL || aux->chave != ch)
    printf("Chave inexistente.\n");
  else {
    if (aux != NULL) {
      if (aux->ant == NULL) L = aux->prox;
      else
        aux->ant->prox = aux->prox;
```

- Casos:
  - Lista vazia ou chave inexistente
  - A chave está no primeiro elemento da lista
  - A chave está no “meio” da lista
  - A chave está no último elemento da lista

Exclusão de uma Chave da Lista

```
No *excluiChave (No *L, int ch) {
  No *aux = buscaLista(L, ch);
  if (aux == NULL || aux->chave != ch)
    printf("Chave inexistente.\n");
  else {
    if (aux != NULL) {
      if (aux->ant == NULL) L = aux->prox;
      else
        aux->ant->prox = aux->prox;
      if (aux->prox != NULL)
        aux->prox->ant = aux->ant;
      free(aux);
    }
  }
  return L;
}
```

- Casos:
  - Lista vazia ou chave inexistente
  - A chave está no primeiro elemento da lista
  - A chave está no “meio” da lista
  - A chave está no último elemento da lista

Exclusão de uma Chave da Lista

```
No *excluiChave (No *L, int ch) {
  No *aux = buscaLista(L, ch);
  if (aux == NULL || aux->chave != ch)
    printf("Chave inexistente.\n");
  else {
    if (aux != NULL) {
      if (aux->ant == NULL) L = aux->prox;
      else
        aux->ant->prox = aux->prox;
      if (aux->prox != NULL)
        aux->prox->ant = aux->ant;
      free(aux);
    }
  }
  return L;
}
```

- Casos:
  - Lista vazia ou chave inexistente
  - A chave está no primeiro elemento da lista
  - A chave está no “meio” da lista
  - A chave está no último elemento da lista

Exercício

1. Considere a estrutura de uma lista duplamente encadeada apresentada na aula.
   a. Implemente uma função que recebe uma lista duplamente encadeada e retira e retorna o *i*-ésimo nó da lista. Tenha certeza de que tal nó existe. A lista e o número inteiro *i* deverão ser passados para a função.
   b. Implemente uma funçaõ que faça a fusão de duas listas duplamente encadeadas ordenadas de inteiros em uma única lista ordenada.
   c. Escreva uma função que recebe as listas L_1 e L_2, e remove da lista L_1 os nós cujas posições devem ser encontradas na lista ordenada L_2. Por exemplo, se L_1 = (A B C D E) e L_2 = (2 4 8), então o segundo e o quarto nós devem ser removidos da lista L_1 (o oitavo nó não existe) e, depois da remoção, L_1 = (A C E).
   d. Escreva uma função que verifica se duas listas têm o mesmo conteúdo.
   e. Escreva a função *main* com um menu para o usuário testar as funções implementadas acima.