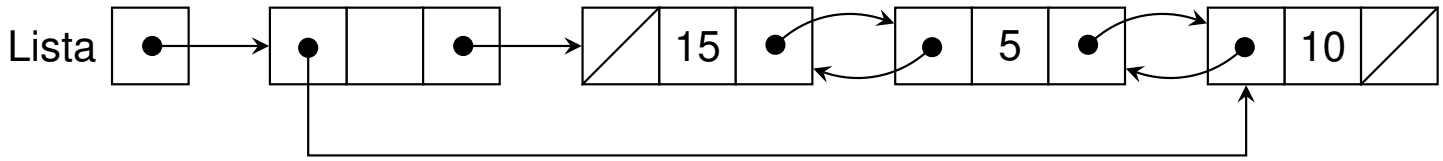
Estrutura de Dados I

Luciana Lee

Tópicos da Aula

Lista Duplamente Encadeada com Nó Cabeça



- O nó cabeça da lista duplamente encadeada será um nó da lista, tal que:
  - O campo de informação não será utilizado;
  - O ponteiro para o próximo elemento do nó apontará para o primeiro elemento da lista que guarda algum dado;
  - O ponteiro para o elemento antecessor na lista apontará para o último elemento da lista.
- Note que:
  - O primeiro elemento possui antecessor igual a `NULL`;
  - O último elemento da lista possui sucessor igual a `NULL`;
  - Podemos utilizar o campo de informação do nó cabeça para guardar o número de nós da lista;

Criação de uma Lista

- Podemos implementar uma função que aloca uma lista vazia, ou seja, uma lista apenas com o nó cabeça.

Criação de uma Lista

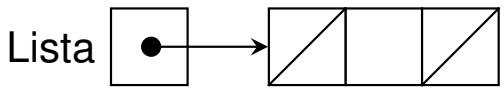
- Podemos implementar uma função que aloca uma lista vazia, ou seja, uma lista apenas com o nó cabeça.

```
No *novaLista (){
  No *novo = (No *) calloc (1,sizeof(No));
  if (novo == NULL){
    printf("ERRO: nao foi possivel alocar o no.\n");
    exit(1);
  }
  return novo;
}
```

Criação de uma Lista

- Podemos implementar uma função que aloca uma lista vazia, ou seja, uma lista apenas com o nó cabeça.

```
No *novaLista (){
  No *novo = (No *) calloc (1,sizeof(No));
  if (novo == NULL){
    printf("ERRO: nao foi possivel alocar o no.\n");
    exit(1);
  }
  return novo;
}
```



Busca na Lista

- Veremos duas versões de algoritmo de busca:
  - Para listas não ordenadas;
  - Para listas ordenadas.

Busca em Lista não Ordenada

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

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

Busca em Lista Ordenada

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

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

Inserção no Início da Lista

- Casos:
  - Lista vazia
  - Lista não vazia

Inserção no Início da Lista

- Casos:
  - Lista vazia
  - Lista não vazia

```
void insereInicio (No *L, int ch) {
  No *novo = criaNo(ch);
  novo->prox = L->prox;
  if (novo->prox != NULL)
    novo->prox->ant = novo;
  else // L estava vazio
    L->ant = novo;
  L->prox = novo;
}
```

Inserção no Final da Lista

- Casos:
  - Lista vazia
  - Lista não vazia

Inserção no Final da Lista

- Casos:
  - Lista vazia
  - Lista não vazia

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

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

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

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

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

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

Exclusão do Primeiro da Lista

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

Exclusão do Primeiro da Lista

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

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

Exclusão do Último da Lista

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

Exclusão do Último da Lista

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

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

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

```
void excluiChave (No *L, int ch) {
  No *aux = buscaLista(L, ch);
  if (aux == NULL)
    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

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

    if (aux->prox != NULL)
      aux->prox->ant = aux->ant;
    else
      L->ant = aux->ant;
  }
}
```

- 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

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

    if (aux->prox != NULL)
      aux->prox->ant = aux->ant;
    else
      L->ant = aux->ant;
  }
}
```

- 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

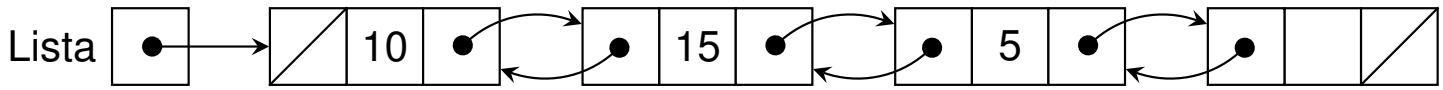
Exclusão de uma Chave da Lista

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

    if (aux->prox != NULL)
      aux->prox->ant = aux->ant;
    else
      L->ant = aux->ant;
  }
}
```

- 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

Lista Duplamente Encadeada com Nó Sentinela



- O nó sentinela da lista duplamente encadeada será um nó da lista, tal que:
  - Será sempre o último nó da lista, indicando o fim dela;
  - O campo de informação não será utilizado.

Criação de uma Lista

- Podemos implementar uma função que aloca uma lista vazia, ou seja, uma lista apenas com o nó cabeça.

Criação de uma Lista

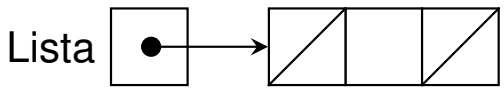
- Podemos implementar uma função que aloca uma lista vazia, ou seja, uma lista apenas com o nó cabeça.

```
No *novaLista(){
  No *sentinela = (No*) calloc (1,sizeof(No));
  if (sentinela == NULL){
    printf("ERRO: nao foi possivel alocar o no.\n");
    exit(1);
  }
  return sentinela;
}
```

Criação de uma Lista

- Podemos implementar uma função que aloca uma lista vazia, ou seja, uma lista apenas com o nó cabeça.

```
No *novaLista(){
  No *sentinela = (No*) calloc (1,sizeof(No));
  if (sentinela == NULL){
    printf("ERRO: nao foi possivel alocar o no.\n");
    exit(1);
  }
  return sentinela;
}
```



Busca na Lista

- Veremos duas versões de algoritmo de busca:
  - Para listas não ordenadas;
  - Para listas ordenadas.

Busca em Lista não Ordenada

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

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

Inserção no Início da Lista

- Casos:
  - Lista vazia
  - Lista não vazia

Inserção no Início da Lista

- Casos:
  - Lista vazia
  - Lista não vazia

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

Inserção no Final da Lista

- Casos:
  - Lista vazia
  - Lista não vazia

Inserção no Final da Lista

- Casos:
  - Lista vazia
  - Lista não vazia

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

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);

  novo->prox = aux;
  novo->ant = aux->ant;
  aux->ant = novo;
  if (novo->ant != NULL)
    novo->ant->prox = novo;
  else
    L = 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

Inserção em uma Lista Ordenada

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

  novo->prox = aux;
  novo->ant = aux->ant;
  aux->ant = novo;
  if (novo->ant != NULL)
    novo->ant->prox = novo;
  else
    L = 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

Inserção em uma Lista Ordenada

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

  novo->prox = aux;
  novo->ant = aux->ant;
  aux->ant = novo;
  if (novo->ant != NULL)
    novo->ant->prox = novo;
  else
    L = 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

Inserção em uma Lista Ordenada

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

  novo->prox = aux;
  novo->ant = aux->ant;
  aux->ant = novo;
  if (novo->ant != NULL)
    novo->ant->prox = novo;
  else
    L = 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

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

Exclusão do Primeiro da Lista

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

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

Exclusão do Último da Lista

- 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;
  while (aux->prox != NULL)
    aux = aux->prox;
  aux = aux->ant;
  if (aux != NULL) {
    aux->prox->ant = aux->ant;
    if (aux->ant != NULL)
      aux->ant->prox = aux->prox;
    else
      L = aux->prox;
    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->prox == NULL)
    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->prox == NULL)
    printf("Chave inexistente.\n");
  else {
    aux->prox->ant = aux->ant;
    if (aux->ant != NULL)
      aux->ant->prox = aux->prox;
    else
      L = aux->prox;
    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->prox == NULL)
    printf("Chave inexistente.\n");
  else {
    aux->prox->ant = aux->ant;
    if (aux->ant != NULL)
      aux->ant->prox = aux->prox;
    else
      L = aux->prox;
    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->prox == NULL)
    printf("Chave inexistente.\n");
  else {
    aux->prox->ant = aux->ant;
    if (aux->ant != NULL)
      aux->ant->prox = aux->prox;
    else
      L = aux->prox;
    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

Dúvidas?