Listas Duplamente Encadeadas
Algoritmos e Estrutura de Dados
Alocação Dinâmica

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia

Roteiro da Aula

1 Introdução

2 Exercício 1

- Assim como fizemos com a Lista Simplesmente Encadeada, precisamos definir a estrutura de dados **nodo**.
- O nodo é uma entidade elementar das estruturas de dados **Pilha**, **Fila** e **Lista**.
- Diferentemente do nodo presente na Lista Simplesmente Encadeada, na lista Duplamente Encadeada cada nodo possui:
  - Uma ligação para o próximo nodo.
  - Uma ligação para o nodo anterior.

Primeiramente, vamos implementar o tipo de dados **nodo**. Vamos analisar o pseudo-código apresentado nas aulas anteriores.

Introdução

Estrutura Nodo

Algoritmo 1: Nodo

```
1 início
2     registro {
3         Inteiro: item;
4         Ponteiro Nodo: prox, ant;
5     } Nodo;
```

Introdução

Estrutura Nodo

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 typedef struct nodo_t {
2     int item;
3     struct nodo_t *prox, *ant;
4 } nodo;
```

Vamos chamar o arquivo que contém esse código de `nodo.h`.

Roteiro da Aula

1 Introdução

2 Exercício 1

Exercício 1

Lista Dupla

Neste exercício, vamos implementar a estrutura de dados **Lista Duplamente Encadeada**:

- Nessa estrutura existem dois ponteiros:
  - um aponta para o **primeiro** nodo da Lista.
  - o outro que aponta para o **último** nodo da Lista.
- Diferentemente da Pilha e da Fila, não existe o nodo **cabeça**.
- Quando a Lista está vazia, os ponteiros primeiro e último apontam para NULL.

Exercício 1

Lista Dupla

Algoritmo 2: Lista

```
1 início
2     registro {
3         Ponteiro Nodo: primeiro, último;
4     } Lista;
```

Exercício 1

Lista Dupla

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 #include "nodo.h"
2 typedef struct {
3   nodo *primeiro;
4   nodo *ultimo;
5 } lista;
```

Vamos chamar o arquivo que contém esse código de `lista.h`.

Exercício 1

Lista Dupla

A seguir, vamos implementar a função CriarListaVazia(L).

Exercício 2

Lista Dupla

Vamos adicionar o protótipo dessa função no arquivo `lista.h`.

```
1 void criarListaVazia(lista *l);
```

Agora, vamos implementar essa função.

Exercício 1

Lista Dupla

- Primeiramente, vamos criar um arquivo chamado `lista.c` que deve conter a implementação dessas funções.
- Para evitar que o compilador não encontre determinada função, vamos incluir o protótipo das funções, inserindo a linha de código a seguir no início do arquivo `lista.c`:

```
1 #include "lista.h"
```

- A seguir, vamos implementar a função CriarListaVazia(L), que recebe uma lista L e aponta devidamente os ponteiros `primeiro` e `ultimo`.

Exercício 1

Lista Dupla

Algoritmo 3: CriarListaVazia

Entrada: Lista L.

```
1 início
2     L.primeiro ← NULL
3     L.último ← NULL
```

Exercício 1

Lista Dupla

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void criarListaVazia(lista *l) {
2     l->primeiro = NULL ;
3     l->ultimo = NULL ;
4 }
```

Exercício 1

Lista Dupla

A seguir, vamos implementar a função ListaVazia(L), que verifica se uma lista está vazia.

Exercício 1

Lista Dupla

Algoritmo 4: ListaVazia

Entrada: Lista L.
Saída: Booleano (V ou F) indicando se L está vazia.

```
1 início
2     se (L.primeiro = NULL) então
3         retorna Verdadeiro
4     senão
5         retorna Falso
```

Exercício 1

Lista Dupla

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 int listaVazia(lista * l ) {
2     return l->primeiro == NULL ;
3 }
```

Exercício 1

Lista Dupla

A seguir, vamos implementar a função InserirInicio(L, x), que insere o elemento x no início de uma lista.

Exercício 1

Lista Dupla

Algoritmo 5: InserirInício

Entrada: Lista L, item x.

```
1  início
2      novo ← ALOCA_NODO()
3      novo.item ← x
4      novo.prox ← L.primeiro
5      // Verifica se a lista está vazia.
6      se (ListaVazia(L)) então
7          L.último ← novo
8      senão
9          L.primeiro.ant ← novo
10     L.primeiro ← novo
11     L.primeiro.ant ← NULL
```

Exercício 1

Lista Dupla

Na linguagem C, a implementação fica conforme o código a seguir:

```
1  void inserirInicio(lista *l, int x) {
2    nodo *novo = malloc(sizeof(nodo));
3    novo->item = x;
4    novo->prox = l->primeiro;
5    if (l->primeiro == NULL)
6      l->ultimo = novo;
7    else
8      l->primeiro->ant = novo;
9    l->primeiro = novo;
10   l->primeiro->ant = NULL;
11 }
```

Exercício 1

Lista Dupla

A seguir, vamos implementar a função InserirFinal(L, x), que insere o elemento x no final de uma lista.

Exercício 1

Lista Dupla

Algoritmo 6: InserirFinal

Entrada: Lista L, item x.

```
1  início
2      novo ← ALOCA_NODO()
3      novo.item ← x
4      novo.prox ← NULL
5      // Verifica se a lista está vazia.
6      se (ListaVazia(L)) então
7          L.primeiro ← novo
8      senão
9          L.último.prox ← novo
10     novo.ant ← L.último
11     L.último ← novo
```

Exercício 1

Lista Dupla

Na linguagem C, a implementação fica conforme o código a seguir:

```
1  void inserirFinal(lista *l, int x) {
2    nodo *novo = malloc(sizeof(nodo));
3    novo->item = x;
4    novo->prox = NULL;
5    if (l->primeiro == NULL)
6      l->primeiro = novo;
7    else
8      l->ultimo->prox = novo;
9    novo->ant = l->ultimo;
10   l->ultimo = novo;
11 }
```

Exercício 1

Lista Dupla

A seguir, vamos implementar a função BuscarPosicao(L, p), que retorna um ponteiro para o nodo na posição p de uma lista.

Exercício 1

Lista Dupla

Algoritmo 7: BuscarPosicao

Entrada: Lista L, item p.
Saída: Nodo na posição p ou NULL caso não encontrado.

```
1 início
2     aux ← L.primeiro
3     c ← 1
4     enquanto (aux ≠ NULL) AND (c < p) faça
5         aux ← aux.prox
6         c ← c + 1
7     retorna aux
```

Exercício 1

Lista Dupla

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 nodo *buscarPosicao(lista *l, int p ) {
2     nodo *aux = l->primeiro;
3     int c = 1;
4     while ( aux != NULL && c < p ) {
5         aux = aux->prox;
6         c ++;
7     }
8     return aux;
9 }
```

Exercício 1

Lista Dupla

A seguir, vamos implementar a função inserirPosicao(L, x, p), que insere um elemento x na posição p de uma lista.

Exercício 1

Lista Dupla

Algoritmo 8: ListaInserirPosição

Entrada: Lista L, item x, posição p.

```
1  início
2      // Verifica se o nodo deve ser inserido no início.
3      se (p = 1) então
4          ListaInserirInício(L, x)
5      senão
6          novo ← ALOCA_NODO()
7          novo.item ← x
8          // Busca o nodo na posição anterior a p.
9          anterior ← ListaBuscarPosição(L,p − 1)
10         // Insere o novo nodo entre os nodos
11         // anterior e posterior.
12         posterior ← anterior.prox
13         anterior.prox ← novo
14         novo.ant ← anterior
15         novo.prox ← posterior
16         // Verifica se o nodo na posição p − 1 é o último,
17         // ou seja, seu posterior é NULL.
18         se (posterior = NULL) então
19             L.último ← novo
20         senão
21             posterior.ant ← novo
```

Exercício 1

Lista Dupla

Na linguagem C, a implementação fica conforme o código a seguir:

```
10 void inserirPosicao(lista *l, int x, int p) {
11   if (p==1)
12     inserirInicio(l,x);
13   else {
14     nodo *anterior = l->primeiro, *posterior;
15     nodo *novo = malloc(sizeof(nodo));
16     novo->item = x;
17     anterior = buscarPosicao(l,p-1);
18     posterior = anterior->prox;
19     anterior->prox = novo;
20     posterior->ant = novo;
21     novo->ant = anterior;
22     novo->prox = posterior;
23     if (posterior == NULL)
24       l->ultimo = novo;
25   }
26 }
```

Exercício 1

Lista Dupla

A seguir, vamos implementar a função RemoverInicio(L), que remove um elemento que se encontra no início de uma lista.

Exercício 1

Lista Simples

Algoritmo 9: ListaRemoverInício

Entrada: Lista L, item x.

```
1  início
2      // Verifica se a lista está vazia.
3      se (ListaVazia(L)) então
4          Imprima "Erro underflow: lista vazia"
5      senão
6          aux ← L.primeiro
7          x ← aux.item
8          L.primeiro ← L.primeiro.prox
9          // Verifica se a lista possui apenas um nodo.
10         se (L.primeiro = NULL) então
11             // Primeiro e Último apontam para NULL.
12             L.último ← NULL
13         senão
14             L.primeiro.ant ← NULL
15         aux.prox ← NULL
16         aux.ant ← NULL
17         DESALOCA_NODO(aux)
18         retorna x
```

Exercício 1

Lista Dupla

```
int removerInicio(lista *l) {
    nodo *aux;
    int x = -1;
    if (listaVazia(l))
        printf("Lista Vazia\n");
    else {
        aux = l->primeiro;
        x = aux->item;
        l->primeiro = l->primeiro->prox;
        if (l->primeiro == NULL)
            l->ultimo = NULL;
        else
            l->primeiro->ant = NULL;
        aux->prox = NULL;
        aux->ant = NULL;
        free(aux);
    }
    return x;
}
```

Exercício 1

Lista Dupla

A seguir, vamos implementar a função RemoverFinal(L), que remove um elemento que se encontra no final de uma lista.

Exercício 1

Lista Dupla

Algoritmo 10: ListaRemoverFinal

Entrada: Lista L
Saída: Item removido

```
1  início
2      // Verifica se a lista está vazia.
3      se (ListaVazia(L)) então
4          Imprima "Erro underflow: lista vazia"
5      senão
6          // Verifica se a lista possui um único elemento.
7          se (L.primeiro = L.último) AND (NOT(ListaVazia(L))) então
8              aux ← L.primeiro
9              L.primeiro ← NULL
10             L.último ← NULL
11         // Caso contrário, a lista possui
12         // pelo menos dois elementos.
13         senão
14             anterior ← L.último.ant
15             aux ← L.último
16             anterior.prox ← NULL
17             L.último ← anterior
18         x ← aux.item
19         aux.prox ← NULL
20         aux.ant ← NULL
21         DESALOCA_NODO(aux)
22         retorna x
```

Exercício 1

Lista Dupla

```
27 int removerFinal(lista *l) {
28     int x = -1;
29     nodo *aux, *anterior;
30     if (listaVazia(l))
31         printf("Lista Vazia\n");
32     else if (l->primeiro == l->ultimo) {
33         aux = l->primeiro;
34         l->primeiro = NULL;
35         l->ultimo = NULL;
36     }
37     else {
38         anterior = l->ultimo->ant;
39         aux = anterior->prox;
40         anterior->prox = NULL;
41         l->ultimo = anterior;
42     }
43     x = aux->item;
44     aux->prox = NULL;
45     aux->ant = NULL;
46     free(aux);
47     return x;
48 }
```

Exercício 1

Lista Dupla

A seguir, vamos implementar a função removerPosicao(L, p), que remove um elemento que se encontra na posição p de uma lista.

Exercício 1

Lista Dupla

Algoritmo 11: ListaRemoverPosição

Entrada: Lista L, posição p
Saída: Item removido

```
1  início
2      // Verifica se a lista está vazia.
3      se (ListaVazia(L)) então
4          Imprima "Erro underflow: lista vazia"
5      senão
6          // Verifica se o nodo a ser removido é o primeiro.
7          se (p = 1) então
8              retorna ListaRemoverInicio(L)
9          senão
10             // Busca o nodo na posição anterior a p.
11             anterior ← ListaBuscarPosição(L,p − 1)
12             // Remove o nodo na posição p, apontado por aux.
13             aux ← anterior.prox
14             posterior ← aux.prox
15             anterior.prox ← posterior
16             // Verifica se o nodo a ser removido é o último.
17             se (aux = L.último) então
18                 L.último ← anterior
19             senão
20                 posterior.ant ← anterior
21             x ← aux.item
22             aux.prox ← NULL
23             aux.ant ← NULL
24             DESALOCA_NODO(aux)
25             retorna x
```

Exercício 1

Lista Simples

```
49 int removerPosicao(lista *l, int p) {
50     int x = -1;
51     nodo *anterior, *posterior, *aux;
52     if (p==1)
53         x = removerInicio(l);
54     else {
55         anterior = buscarPosicao(l,p-1);
56         aux = anterior->prox;
57         posterior = aux->prox;
58         anterior->prox = posterior;
59         if (aux == l->ultimo)
60             l->ultimo = anterior;
61         else
62             posterior->ant = anterior;
63         x = aux->item;
64         aux->ant = NULL;
65         aux->prox = NULL;
66         free(aux);
67     }
68     return x;
69 }
```

Exercício 1

Lista Simples

- Ainda falta uma função: aquela que apaga todos os nodos de uma lista.
- Vamos implementá-la a seguir.

Exercício 1

Lista Simples

Algoritmo 12: ApagarLista

Entrada: Lista L.

```
1 início
2     aux ← L.primeiro
3     enquanto (L.primeiro ≠ NULL) faça
4         L.primeiro ← L.primeiro.prox
5         L.primeiro.ant = NULL
6         DESALOCA_NODO(aux)
7         aux = L.primeiro
```

Exercício 1

Lista Simples

Na linguagem C, a implementação fica conforme o código a seguir:

```
1 void apagarLista(lista *l) {
2   nodo *aux = l->primeiro;
3   while (aux != NULL) {
4     l->primeiro = l->primeiro->prox;
5     l->primeiro->ant = NULL;
6     free(aux);
7     aux = l->primeiro;
8   }
9 }
```

Exercício 1

Lista Simples

Vamos implementar a função que imprime os valores de uma lista.

Exercício 1

Lista Simples

Algoritmo 13: ApagarLista

Entrada: Lista L.

```
1 início
2     aux ← L.primeiro
3     enquanto (aux ≠ NULL) faça
4         Imprima(aux.item)
5         aux = aux.prox
```

Exercício 1

Lista Simples

Na linguagem C, a implementação fica conforme o código a seguir:

```c
void imprimirLista(lista *l) {
  nodo *aux = l->primeiro;
  while (aux != NULL) {
    printf("%d ", aux->item);
    aux = aux->prox;
  }
  printf("\n");
}
```

Exercício 1

Lista Simples

Após implementar essas funções, faça o teste, inserindo e removendo alguns valores.

Exercício 1

Lista Simples

Em um arquivo chamado executa_lista.c, insira o código a seguir:

```c
#include "lista.h"
int main() {
  int i;
  lista *l = malloc(sizeof(lista));
  criarListaVazia(l);
  for (i=0; i<10; i++) {
    inserirInicio(l,i);
    printf("Inseriu Inicio %d\n",i);
  }
  for (i=-1;i>-3;i--) {
    inserirFinal(l,i);
    printf("Inseriu final %d\n",i);
  }
  printf("Inseriu %d na posicao 4\n",10);
  inserirPosicao(l,10,4);
  imprimirLista(l);
  /** Continua no proximo slide **/
```

Exercício 1

Lista Simples

```c
#include "lista.h"
int main() {
  /** Codigo anterior **/
  i = removerInicio(l);
  printf("Removeu Inicio %d\n",i);
  imprimirLista(l);
  i = removerFinal(l);
  printf("Removeu Final %d\n",i);
  imprimirLista(l);
  i = removerInicio(l);
  printf("Removeu Inicio %d\n",i);
  imprimirLista(l);
  i = removerPosicao(l,7);
  printf("Removeu %d na posicao %d\n",i,7);
  apagarLista(l);
  printf("Apagou a lista\n");
  free(l);
}
```

Listas Duplamente Encadeadas
Algoritmos e Estrutura de Dados
Alocação Dinâmica

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia