

Listas encadeadas

Prof. Henrique Y. Shishido

Universidade Tecnológica Federal do Paraná

shishido@utfpr.edu.br

Crédito

Aluna: Renata Carina Soares (Estudante de eng. comp.)

Orientação: Prof. Dr. Danilo Sanches

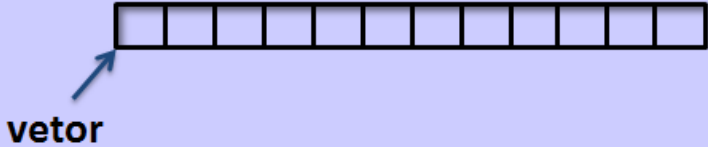
Tópicos

Motivação

1. Lista Simplesmente Encadeada
2. Lista Duplamente Encadeada
3. Listas Circulares
4. Implementações Recursivas
5. Listas de Tipos Estruturados



Motivação

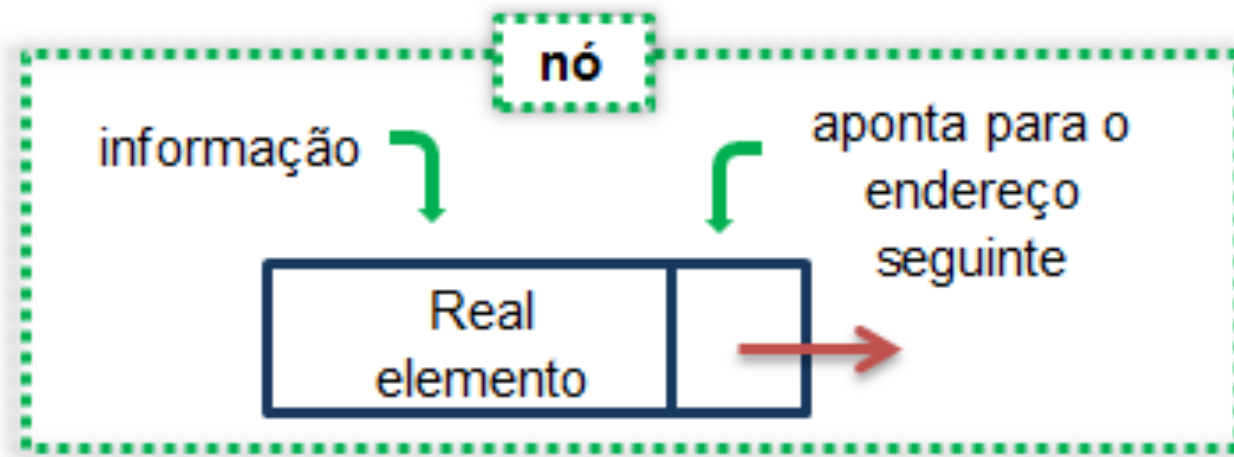
Vetor	Estruturas de dados dinâmicas
Ocupa um espaço contíguo de memória	Crescem ou decrescem à medida que elementos são inseridos ou removidos
Permite acesso randômico aos elementos	
Deve ser dimensionado com um número máximo de elementos	Exemplo: Listas encadeadas
	Amplamente usadas para implementar outras estruturas de dados

1. Lista Simplesmente Encadeada



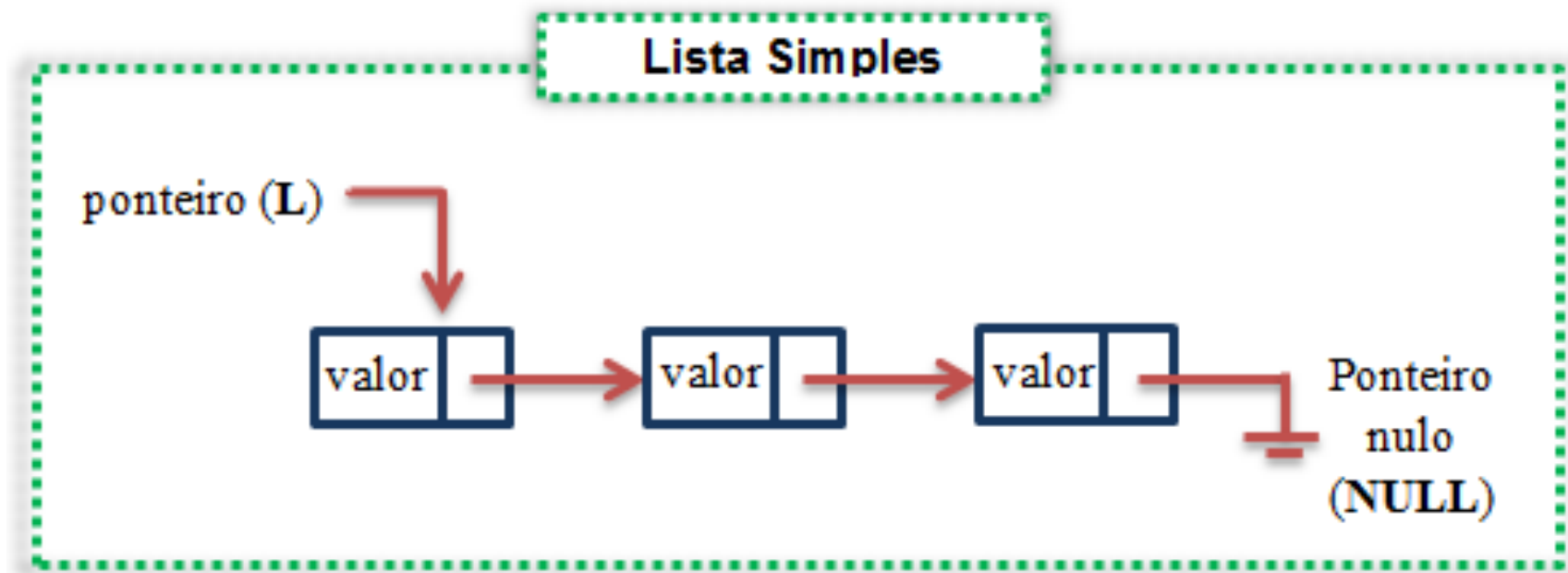
1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

- Sequência encadeada de elementos, denominados de **nós** da lista.
- Um nó é composto por dois campos:
 - ***Campo de informação (info)***: armazena o valor do nó.
 - ***Campo do endereço seguinte (prox)***: ponteiro que aponta para o próximo elemento da lista.



1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

- A lista é representada por um ponteiro para o **primeiro nó**.
- O ponteiro do último elemento é **NULL**.



Criação do tipo abstrato de dado (TAD)

```
typedef struct lista{  
    int info;           //refere-se ao valor contido no elemento  
    struct lista* prox; //ponteiro para o próximo do tipo criado  
}ListaSimp;
```

- Uma lista é uma estrutura auto-referenciada, pois o campo `prox` é um ponteiro para uma próxima estrutura do mesmo tipo.
- É representada pelo ponteiro para seu primeiro nó, do tipo *ListaSimp*.

1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

Função para criar a lista

- Criar uma *lista vazia*, representada pelo ponteiro NULL.

```
ListaSimp* criar_lista(void){  
    return NULL;    //atribuição NULL para a lista  
}
```


1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

Função para inserir elementos na lista

- Aloca memória para armazenar o elemento.
- Encadeia o nó na lista existente.

```
ListaSimp* inserir_elementos(ListaSimp* L, int valor){  
    ListaSimp* novo = (ListaSimp*)malloc(sizeof(ListaSimp));  
  
    novo -> info = valor;  
    novo -> prox = L;  
  
    return novo;  
}
```

1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

Função para imprimir os valores contidos na lista

```
void imprimir (ListaSimp* L){  
    ListaSimp* p;  
    if(lista_vazia(L))  
        printf("\nLista vazia!\n");  
    else{  
        for (p = L; p != NULL; p = p -> prox) {  
            printf("%d ", p -> info);  
        }  
        printf("\n\n");  
    }  
}
```

- Variável auxiliar p:
 - Ponteiro usado para armazenar o endereço de cada nó.
 - Dentro do loop, aponta para cada um dos elementos da lista.

1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

Comandos necessários na função principal

- Deve-se atualizar a variável que representa a lista a cada inserção de um novo elemento.

```
main(){  
    ListaSimp* L;           //declara uma lista não inicializada  
    L = criar_lista();       //cria e inicializa a lista como vazia  
    L = inserir_elementos(L, 4); //insere na lista o elemento de valor 4  
    L = inserir_elementos(L, 30); //insere na lista o elemento de valor 30  
    ...  
    imprimir(L);  
    return 0;  
}
```

Função para verificar se uma lista está vazia

- Retorna 1 (verdadeiro) se a lista estiver vazia, caso contrário, retorna 0 (falso).

```
int lista_vazia(ListaSimp* L){  
    return(L == NULL);  
}
```

1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

Função para buscar um elemento na lista

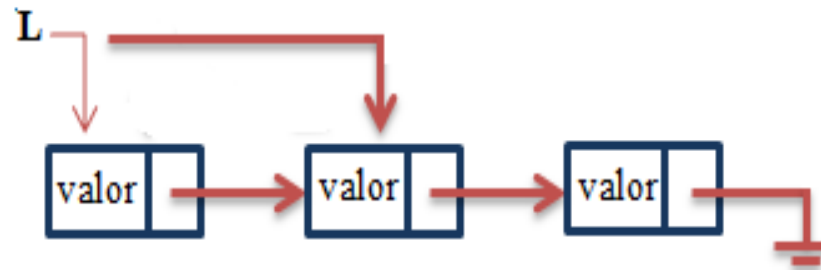
- Informa o número para a realização da pesquisa.
- Retorna o nó que contenha o valor ou NULL, caso não seja encontrado.

```
ListaSimp* buscar_elemento (ListaSimp* L, int valor) {  
    ListaSimp* p;  
    for(p = L; p != NULL; p = p -> prox) {  
        if(p -> info == valor)  
            return p;  
    }  
    return NULL;  
}
```

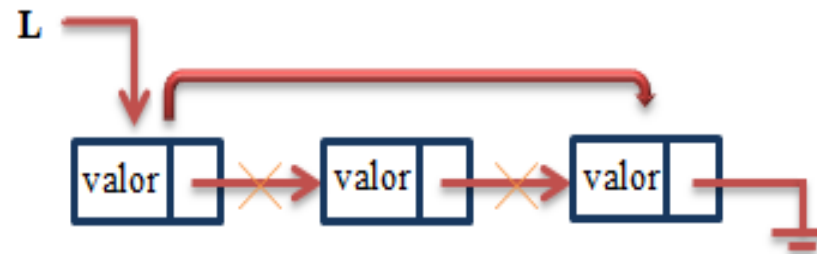
1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

Função para retirar um elemento da lista

- Informa o valor do nó a ser retirado.
- Atualiza o ponteiro L, se o elemento for o primeiro:



- Caso contrário, apenas retira o nó da lista:



1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

```
ListaSimp* retirar_elemento(ListaSimp* L, int valor){  
  
    ListaSimp* p = L;          /* ponteiro para o nó anterior*/  
    ListaSimp* ant = NULL;      /* ponteiro para percorrer a lista*/  
  
    /* procura elemento na lista, armazenando o anterior*/  
    while (p != NULL && p->info != valor){  
        ant = p;  
        p = p->prox;  
    }  
  
    /* verifica se achou o elemento*/  
    if (p == NULL)  
        return L;              /* não achou: retorna a lista original*/  
  
    /* retira elemento*/  
    if (ant == NULL)            /* retira elemento do início*/  
        L = p->prox;  
    else                        /* retira elemento do meio da lista*/  
        ant->prox = p->prox;  
  
    free(p);  
    return L;  
}
```

1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

Função para liberar uma lista

- Desaloca a lista, liberando todos os elementos alocados

```
void liberar(ListaSimp* L){  
    ListaSimp* p = L;  
    while (p != NULL) {  
        ListaSimp* t = p -> prox; /* guarda referência para próximo nó */  
        free(p);                  /* libera a memória apontada por p */  
        p = t;                    /* faz p apontar para o próximo nó */  
    }  
}
```


1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

TAD Lista de inteiros

```
/* TAD: lista de inteiros* /  
  
typedef struct lista ListaSimp;  
  
ListaSimp* criar_lista(void);  
void liberar(ListaSimp* L);  
  
ListaSimp* inserir_elementos(ListaSimp* L, int valor);  
ListaSimp* retirar_elemento(ListaSimp* L, int valor);  
  
int lista_vazia(ListaSimp* L);  
ListaSimp* buscar_elemento(ListaSimp* L, int valor);  
void imprimir(ListaSimp* L);
```

1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

```
#include<stdio.h>
```

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

```
int main(void)
```

```
{
```

```
    ListaSimp* L;
```

```
    L= criar_lista();
```

```
    L= inserir_elementos(L, 23);
```

```
    L= inserir_elementos (L, 45);
```

```
    L= inserir_elementos (L, 56);
```

```
    L= inserir_elementos (L, 78);
```

```
    imprimir(L);
```

```
    L= retirar_elemento(L, 78);
```

```
    imprimir(L);
```

```
    L= retirar_elemento(L, 45);
```

```
    imprimir(L);
```

```
    liberar(L);
```

```
    return 0;
```

```
}
```

Programa que atualiza as funções de lista exportadas

```
/* declara uma lista não iniciada* /
```

```
/* inicia lista vazia* /
```

```
/* insere na lista o elemento 23* /
```

```
/* insere na lista o elemento 45* /
```

```
/* insere na lista o elemento 56* /
```

```
/* insere na lista o elemento 78* /
```

```
/* imprimirá: 78 56 45 23* /
```

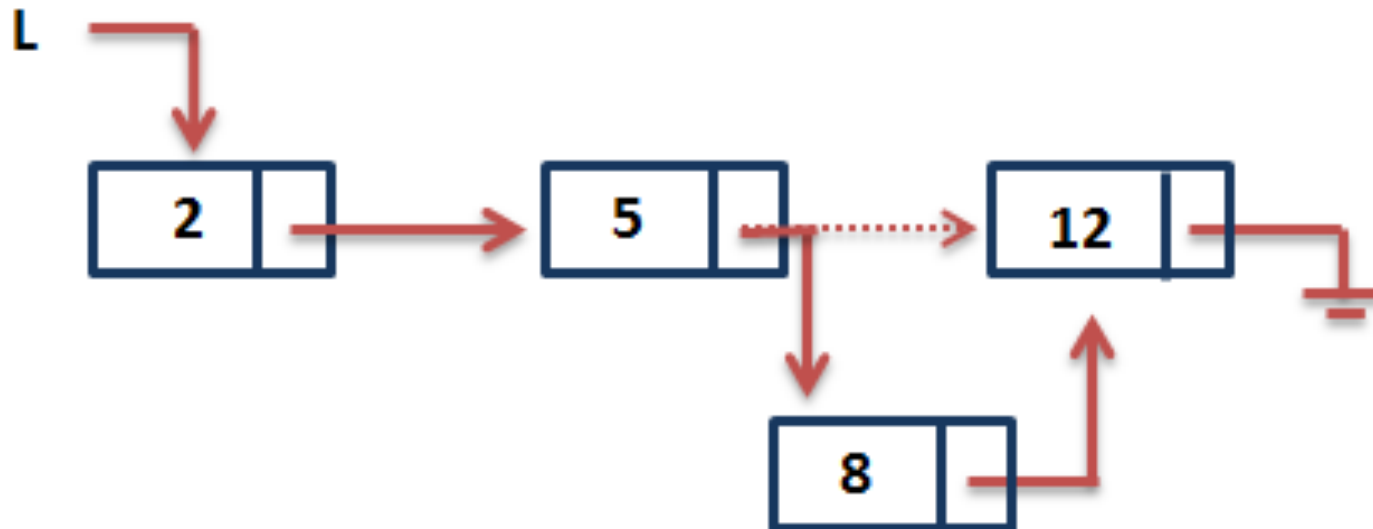
```
/* imprimirá: 56 45 23* /
```

```
/* imprimirá: 56 23* /
```

1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

Manutenção de uma lista ordenada

- A função de inserir ordenado percorre a lista até encontrar a posição correta para inserir o novo elemento.



1. Lista Simplesmente Encadeada (source: lst_simpl_encadeada.c)

```
ListaSimp* inserir_ordenado(ListaSimp* L, int valor){
    ListaSimp* ant = NULL;          /*ponteiro para o elemento anterior*/
    ListaSimp* p = L;              /*ponteiro para percorrer a lista*/

    ListaSimp* novo = (ListaSimp*)malloc(sizeof(ListaSimp)); /*cria novo nó*/
    novo -> info = valor;

    /*procura posição para inserção*/
    while(p!=NULL && p->info < valor){
        ant = p;
        p = p -> prox;
    }

    /*encadeia o elemento*/
    if(ant == NULL){                /*insere o elemento no início*/
        novo -> prox = L;
        L = novo;
    }
    else{                          /*insere o elemento no meio da lista*/
        novo -> prox = ant -> prox;
        ant -> prox = novo;
    }

    return L;
}
```