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

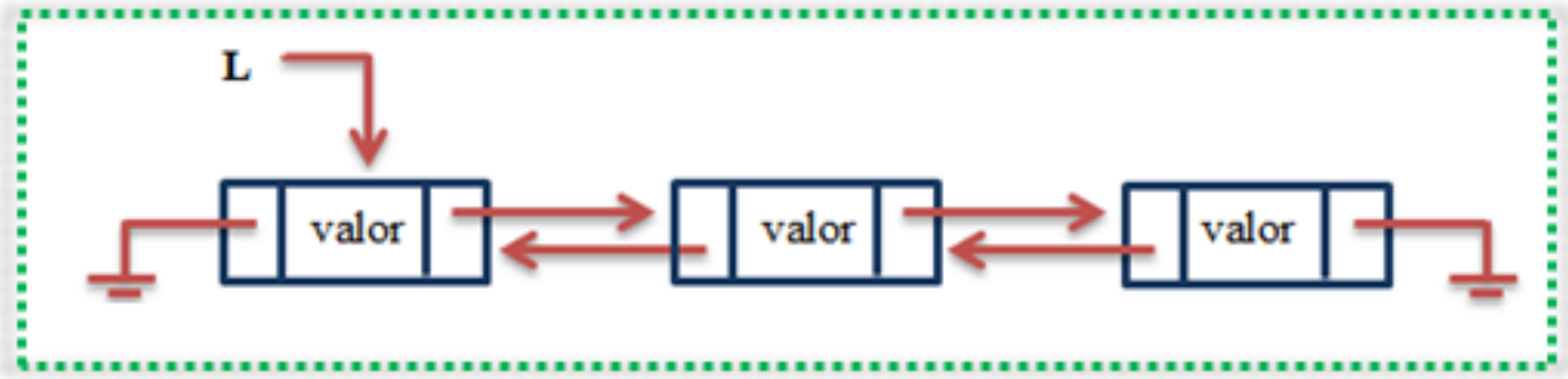
Motivação

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

2. Lista Duplamente Encadeada

2. Lista Duplamente Encadeada (source: lst_dupl_encadeada.c)

- Um **nó** é composto por **três** campos:
  - ***Campo de informação***: armazena o real elemento da lista.
  - ***Campo do endereço seguinte***: usado para acessar determinado nó. Este campo do endereço seguinte dentro de um nó pode ser visto como uma ligação ou um ponteiro para o próximo nó.
  - ***Campo do endereço anterior***: este campo dentro de um nó pode ser visto como uma ligação ou um ponteiro para o nó anterior.



2. Lista Duplamente Encadeada (source: lst_dupl_encadeada.c)

Lista Simples x Lista Dupla

Lista Simples		Lista Dupla
Dado um elemento é **possível** acessar somente o **próximo**.		Dado um elemento, é **possível** acessar o **próximo** e o **anterior**.
Somente com o ponteiro prox **não** é possível percorrer a lista em **ordem inversa**.		Dado um ponteiro para o último elemento da lista, é **possível** percorrer a lista em **ordem inversa**.

2. Lista Duplamente Encadeada (source: lst_dupl_encadeada.c)

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
        struct lista* ant;       // ponteiro para o anterior do tipo criado
}ListaDupla;
```

- Uma lista é uma estrutura auto-referenciada, pois o campo prox e o ant são ponteiros para uma próxima estrutura do mesmo tipo.
- Uma lista encadeada é representada pelo ponteiro para seu primeiro nó, do tipo *ListaDupla.*

Função para inserir elementos na lista

```
/*Inserção no início: retorna a lista atualizada*/
ListaDupla* inserir_elementos(ListaDupla* L, int valor){

        ListaDupla* novo = (ListaDupla*)malloc(sizeof(ListaDupla));

        novo -> info = valor;
        novo -> prox = L;
        novo -> ant = NULL;

        if(L != NULL) /*verifica se a lista está vazia*/
              L -> ant = novo;

        return novo;
}
```

Função para buscar um elemento na lista

- Recebe a informação referente ao nó a pesquisar

```
ListaDupla* buscar_elemento (ListaDupla * L, int valor ){

        ListaDupla * p;

        for(p = L; p != NULL; p = p -> prox) {
                if(p -> info == valor)
                /*retorna o ponteiro do nó da lista que representa o elemento*/
                        return p;
        }
        return NULL;                    /*não achou o elemento e retorna NULL*/
}
```

Obs: Implementação idêntica à lista encadeada simples.

Função para retirar um elemento da lista

- O ponteiro p aponta para o elemento a retirar.
- Se o ponteiro **p** aponta para um elemento no **meio** da lista:
  - O anterior passa a apontar para o próximo:

p -> ant -> prox = p -> prox;

  - O próximo passa a apontar para o anterior:

p -> prox -> ant = p ->ant;

Função para retirar um elemento da lista

- Se o ponteiro **p** aponta para o **último** elemento:
  - Não é possível escrever:

    p -> prox -> ant, pois **p -> prox** é NULL

- Se o ponteiro **p** para o **primeiro** elemento:
  - Não é possível escrever:

    p -> ant -> prox , pois **p ->ant** é **NULL**

  - É necessário atualizar o valor da lista, pois o primeiro elmento será removido.

2. Lista Duplamente Encadeada (source: lst_dupl_encadeada.c)

```
ListaDupla* retirar_elemento(ListaDupla* L, int valor){

        ListaDupla* p = buscar_elemento(L, <valor a ser procurado>);

        if (p == NULL)
                return L;  /*não achou o elemento: retorna a lista inalterada*/

        if (L == p)                              /*testa se é o primeiro elemento*/
                L = p -> prox;
        else
                p -> ant -> prox = p -> prox;

        if (p -> prox != NULL)                   / *testa se é o último elemento*/
                p -> prox -> ant = p -> ant;

        free(p);
        return L;
}
```