

Programação de Computadores II

Cap. 10 – Listas Encadeadas

Livro: Waldemar Celes, Renato Cerqueira, José Lucas Rangel. *Introdução a Estruturas de Dados*, Editora Campus (2004)
Slides adaptados dos originais dos profs.: Marco Antonio Casanova e Marcelo Gattass (PUC-Rio)

1

Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004)
Capítulo 10 – Listas encadeadas

2

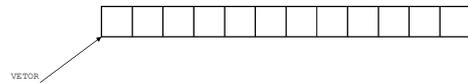
Tópicos

- Motivação
- Listas encadeadas

3

Motivação

- Vetor
 - ocupa um espaço contíguo de memória
 - permite acesso randômico aos elementos
 - deve ser dimensionado com um número máximo de elementos



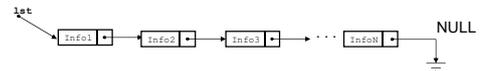
4

Motivação

- Estruturas de dados dinâmicas:
 - crescem (ou decrescem) à medida que elementos são inseridos (ou removidos)
 - Exemplo:
 - listas encadeadas:
 - amplamente usadas para implementar outras estruturas de dados

5

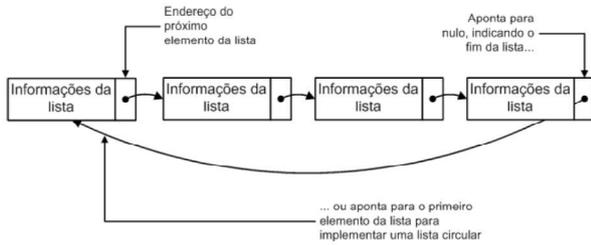
Listas Encadeadas



- Lista encadeada:
 - sequência encadeada de elementos, chamados de *nós da lista*
 - nó da lista é representado por dois campos:
 - a informação armazenada e
 - o ponteiro para o próximo elemento da lista
 - a lista é representada por um ponteiro para o primeiro nó
 - o ponteiro do último elemento é NULL

6

Listas Encadeadas



(Estrutura de Dados com Algoritmos e C, Marcos Laureano, 2012)

7

Listas Encadeadas

- Estruturas encadeadas dinâmicas possuem tamanho variável, pois diferentemente de vetores dinâmicos, sua **memória é reservada** por elemento e não para toda a estrutura
- Listas encadeadas podem ser **simplesmente ou duplamente encadeadas**
 - uma lista simplesmente encadeada contém um elo (endereço) com o próximo item da lista
 - uma lista duplamente encadeada contém elos (endereços) com o elemento anterior e com o elemento posterior a ele

8

Estrutura com ponteiro para ela mesma

```
struct lista {
    int info;
    struct lista* prox;
};
typedef struct lista Lista;
```



9

Exemplo: Lista de inteiros

```
struct lista {
    int info;
    struct lista* prox;
};
typedef struct lista Lista;
```

1. **Lista** é uma estrutura **auto-referenciada**, pois o campo **prox** é um ponteiro para uma próxima estrutura do mesmo tipo.
2. uma lista encadeada é representada pelo ponteiro para seu primeiro elemento, do tipo **Lista***

10

Exemplo: Lista de inteiros (outra forma)

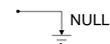
```
typedef struct lista Lista;

struct lista {
    int info;
    Lista *prox;
};
```

11

Listas Encadeadas de inteiros: Criação

```
/* função de criação: retorna uma lista vazia */
Lista* lst_cria (void)
{
    return NULL;
}
```

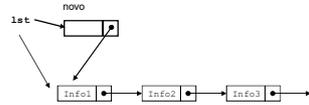


Cria uma lista vazia, representada pelo ponteiro NULL

12

Listas encadeadas de inteiros: Inserção

- aloca memória para armazenar o elemento
- encadeia o elemento na lista existente



```

/* inserção no início: retorna a lista atualizada */
Lista* lst_inserere (Lista* lst, int val)
{
    Lista* novo = (Lista*) malloc(sizeof(Lista));
    novo->info = val;
    novo->prox = lst;
    return novo;
}
    
```

13

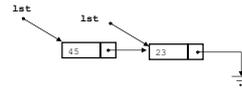
Listas Encadeadas: exemplo

- cria uma lista inicialmente vazia e insere novos elementos

```

int main (void)
{
    Lista* lst;
    lst = lst_cria();
    lst = lst_inserere (lst, 23);
    lst = lst_inserere (lst, 45);
    return 0;
}
    
```

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



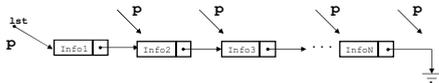
14

Listas Encadeadas: impressão

- Imprime os valores dos elementos armazenados

```

/* função imprime: imprime valores dos elementos */
void lst_imprime (Lista* lst)
{
    Lista* p;
    for (p = lst; p != NULL; p = p->prox)
        printf("info = %d\n", p->info);
}
    
```



15

Listas Encadeadas: Teste de vazia

- Retorna 1 se a lista estiver vazia ou 0 se não estiver vazia

```

/* função vazia: retorna 1 se vazia ou 0 se não vazia */
int lst_vazia (Lista* lst)
{
    return (lst == NULL);
}
    
```

16

Listas Encadeadas: busca

- recebe a informação referente ao elemento a pesquisar
- retorna o ponteiro do nó da lista que representa o elemento, ou NULL, caso o elemento não seja encontrado na lista

```

/* função busca: busca um elemento na lista */
Lista* busca (Lista* lst, int v)
{
    Lista* p;
    for (p=lst; p!=NULL; p = p->prox) {
        if (p->info == v)
            return p;
    }
    return NULL;
}
    
```

17

Listas Encadeadas: remover um elemento

- recebe como entrada a lista e o valor do elemento a retirar
- atualiza o valor da lista, se o elemento removido for o primeiro



- caso contrário, apenas remove o elemento da lista



18

```

/* função retira: retira elemento da lista */
Lista* lst_retira (Lista* lst, int val)
{
    Lista* ant = NULL; /* ponteiro para elemento anterior */
    Lista* p = lst; /* ponteiro para percorrer a lista */
    /* procura elemento na lista, guardando anterior */
    while (p != NULL && p->info != val) {
        ant = p;
        p = p->prox;
    }
    /* verifica se achou elemento */
    if (p == NULL)
        return lst; /* não achou: retorna lista original */
    /* retira elemento */
    if (ant == NULL)
        { /* retira elemento do inicio */
            lst = p->prox; }
    else { /* retira elemento do meio da lista */
        ant->prox = p->prox; }
    free(p);
    return lst;
}

```

Listas Encadeadas: Libera a lista

- destrói a lista, liberando todos os elementos alocados

```

void lst_libera (Lista* lst)
{
    Lista* p = lst; Lista* t;
    while (p != NULL) {
        t = p->prox; /* guarda referência p/ próx. elemento */
        free(p); /* libera a memória apontada por p */
        p = t; /* faz p apontar para o próximo */
    }
}

```

TAD Listas Encadeadas de Inteiros

```

/* TAD: lista de inteiros */

typedef struct lista Lista;

Lista* lst_cria (void);
void lst_libera (Lista* lst);

Lista* lst_insere (Lista* lst, int val);
Lista* lst_retira (Lista* lst, int val);

int lst_vazia (Lista* lst);
Lista* lst_busca (Lista* lst, int val);
void lst_imprime (Lista* lst);

```

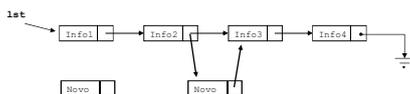
```

#include <stdio.h>
#include "lista.h"
int main (void)
{
    Lista* lst; /* declara uma lista não iniciada */
    lst=lst_cria(); /* inicia lista vazia */
    lst=lst_insere(lst,23); /* insere na lista o elemento 23 */
    lst=lst_insere(lst,45); /* insere na lista o elemento 45 */
    lst=lst_insere(lst,56); /* insere na lista o elemento 56 */
    lst=lst_insere(lst,78); /* insere na lista o elemento 78 */
    lst_imprime(lst); /* imprimirá: 78 56 45 23 */
    lst=lst_retira(lst,78);
    lst_imprime(lst); /* imprimirá: 56 45 23 */
    lst=lst_retira(lst,45);
    lst_imprime(lst); /* imprimirá: 56 23 */
    lst_libera(lst);
    return 0;
}

```

Listas Encadeadas

- Manutenção da lista ordenada
 - função de inserção percorre os elementos da lista até encontrar a posição correta para a inserção do novo



```

/* função insere_ordenado: insere elemento em ordem */
Lista* lst_insere_ordenado (Lista* lst, int val)
{
    Lista* novo;
    Lista* ant = NULL; /* ponteiro para elemento anterior */
    Lista* p = lst; /* ponteiro para percorrer a lista */
    /* procura posição de inserção */
    while (p != NULL && p->info < val)
        { ant = p; p = p->prox; }
    /* cria novo elemento */
    novo = (Lista*) malloc(sizeof(Lista));
    novo->info = val;
    /* encadeia elemento */
    if (ant == NULL)
        { /* insere elemento no inicio */
            novo->prox = lst; lst = novo; }
        else { /* insere elemento no meio/final da lista */
            novo->prox = ant->prox;
            ant->prox = novo; }
    return lst;
}

```

Igualdade de listas

```
int lst_igual (Lista* lst1, Lista* lst2);
```

- implementação não recursiva
 - percorre as duas listas, usando dois ponteiros auxiliares:
 - se duas informações forem diferentes, as listas são diferentes
 - ao terminar uma das listas (ou as duas):
 - se os dois ponteiros auxiliares são NULL, as duas listas têm o mesmo número de elementos e são iguais

25

Listas iguais: não recursiva

```
int lst_igual (Lista* lst1, Lista* lst2)
{
    Lista* p1; /* ponteiro para percorrer l1 */
    Lista* p2; /* ponteiro para percorrer l2 */
    for (p1=lst1, p2=lst2;
         p1 != NULL && p2 != NULL;
         p1 = p1->prox, p2 = p2->prox)
    {
        if (p1->info != p2->info) return 0;
    }
    return p1==p2;
}
```

26

Exercício

Considere listas encadeadas que armazenam estruturas do tipo Compromisso. Cada elemento da lista é do tipo Elemento, descrito abaixo:

```
typedef struct elemento {
    Compromisso compr; /* Compromisso armazenado */
    struct elemento *prox; /* Ponteiro para o próximo elemento */
} Elemento;
```

Escreva uma função que busque em uma lista encadeada a quantidade de elementos (nodos) que são de um determinado mês. A função recebe como parâmetro o ponteiro para a lista e o mês de interesse e retorna o número de compromissos no mês. O protótipo da função é:

```
int busca(Elemento* lst, int mes);
```

09/06/11

```
typedef struct data {
    int dd, mm, aa; /* Dia, mes e ano */
} Data;
```

```
typedef struct compromisso {
    char descricao[81]; /* Descricao do compromisso */
    Data dta; /* Data do compromisso */
} Compromisso;
```

```
typedef struct elemento {
    Compromisso compr; /* Compromisso armazenado */
    struct elemento *prox; /* Ponteiro para o próximo elemento */
} Elemento;
```

09/06/11

```
int busca(Elemento* lst, int mes);
Elemento* lista_insere (Elemento* lst, char *descricao, int dia, int mes, int ano);
```

```
int busca(Elemento* lst, int mes){
    Elemento *ptr;
    int cont=0;
    for (ptr = lst; ptr != NULL; ptr = ptr->prox){
        if (ptr->compr.dta.mm == mes)
            cont++;
    }
    return cont;
}
Elemento* listaInsere (Elemento* lst, char *descricao, int dia, int mes, int ano) {
    Elemento* novo = (Elemento*) malloc(sizeof(Elemento));
    strcpy(novo->compr.descricao, descricao);
    novo->compr.dta.dd = dia;
    novo->compr.dta.mm = mes;
    novo->compr.dta.aa = ano;
    novo->prox = lst;
    return novo;
}
```

09/06/11

```
main(void) {
    Elemento *lista=NULL;
    Elemento *primeiro;
    lista = listaInsere(lista, "compromisso um",25,11,10);
    lista = listaInsere(lista, "compromisso dois",30,11,10);
    lista = listaInsere(lista, "compromisso tres",01,12,10);
    printf("\nNumero de compromissos no mes 11=%d\n",busca(lista,11));
}
```

09/06/11

Outras implementações

Listas encadeadas



Listas circulares



Listas duplamente encadeadas

