Métodos Computacionais

Listas Encadeadas

Vetores

- Declaração de vetor implica na especificação de seu tamanho
  - Não se pode aumentar ou diminuir tamanho
- Outra alternativa no uso de vetores é alocar dinamicamente o espaço necessário e guardar endereço inicial do vetor
- Porém, depois de determinado o tamanho do vetor, não podemos liberar posição de memória arbitrária
  - Não se pode diminuir tamanho do vetor

Possível desperdício ou falta de memória!

Listas Encadeadas

- Objetivo
  - Permite o armazenamento de um conjunto de dados, de forma similar a um vetor, porém com maior flexibilidade:
    - Alocação dinâmica de memória (permite o crescimento ou diminuição do conjunto de forma dinâmica, evitando o desperdício ou a falta de memória).

Listas Encadeadas

- Principais Características
  - Para cada novo elemento inserido na lista, aloca-se um espaço de memória para armazená-lo
  - Junto a cada elemento deve-se armazenar o ponteiro (endereço) para o próximo elemento (elemento + ponteiro = nó da lista)
    - Caso não exista próximo elemento, o ponteiro para o próximo elemento é NULL
  - O acesso aos elementos é sempre seqüencial
    - Não se pode garantir que os elementos ocuparão um espaço de memória contíguo (não se pode localizar elementos com base em um deslocamento constante).

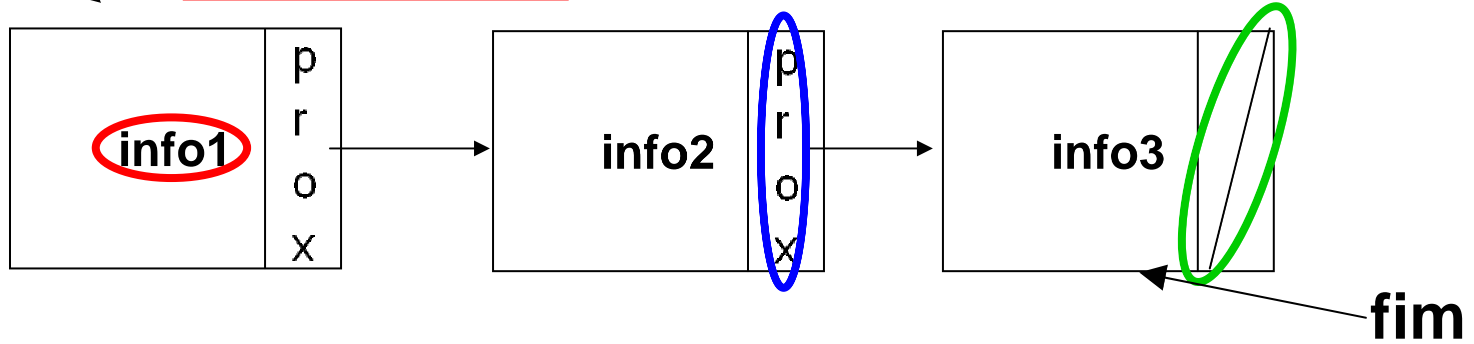
Representação de Listas Encadeadas

Informação armazenada no nó da lista

Armazena o endereço do próximo nó

Armazena endereço nulo

primeiro




```
struct   lista  {
  int info ; /* info pode ser de qualquer tipo */
  struct lista *prox; /* campo que armazena
                  endereço do próximo elemento */
} ;
typedef   struct lista   Lista ;
```

Criando Listas Encadeadas Vazias

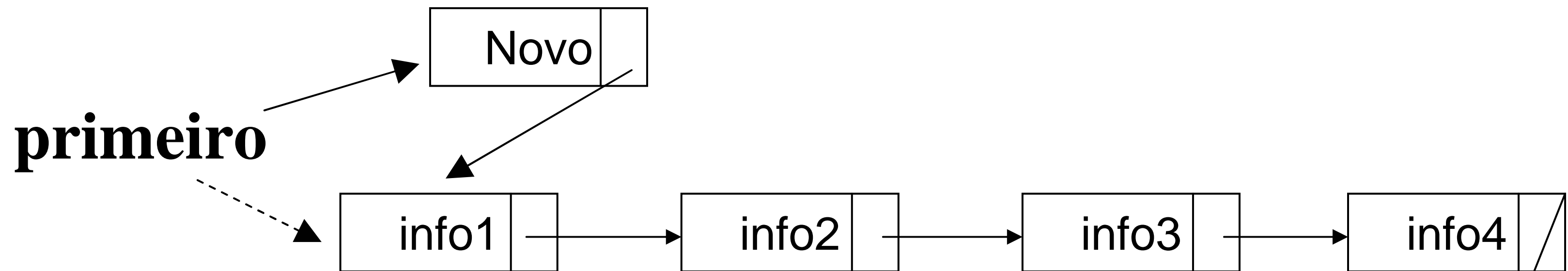
- Pode-se pensar em duas formas de escrever uma função para criar uma lista vazia:
  - Criação de um nó sem nenhuma informação
  - Retornar um endereço nulo
    - Abordagem que será trabalhada
    - Mais simples

```
/* função de criação: retorna uma lista
vazia */

Lista* lst_cria(){
    return   NULL ;
}
```

Inserção de Elementos em Listas Encadeadas

- Diferentes formas de inserir um elemento na lista:
  - No começo
    - Mais simples
  - No fim
    - Neste caso deve-se percorrer a lista toda

Inserindo Elementos em Listas Encadeadas




```
/* inserção início da lista: retorna a lista
atualizada */
Lista* lst_insere(Lista* lis,int i){
  Lista* novo = (Lista*) malloc(sizeof(Lista));
  novo->info  =  i ;
  novo->prox  = lis ;
  return   novo  ;
}
```

Endereço do começo da lista

Usando Funções de Criação e Inserção de Listas Encadeadas

```
int   main() {
  Lista* lis;/* declara uma lista não inicializada
*/
  lis = lst_cria();/*cria e inicializa lista como
vazia */
  lis = lst_insere(lis,23); /* insere o elemento 23
*/
  lis = lst_insere(lis,45); /* insere o elemento 45
*/
   ...
  return   0  ;
}
```

Não esquecer de atualizar a variável que representa a lista a cada inserção!

Imprimindo Listas Encadeadas

```
void  lst_imprime(Lista* lis){
  Lista* p; /*variável auxiliar para percorrer a
            lista */
  for(p = lis; p != NULL; p = p->prox){

      printf ( "info = %d\n ",  p->info ) ;
  }
}
```

Laço para percorrer todos os nós da lista

Verificando se a Lista Está Vazia

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

```
/* versão compacta da função lst_vazia */
int   lst_vazia  ( Lista* lis ){
  return(lis == NULL) ;
}
```

Buscando um Elemento em Listas Encadeadas

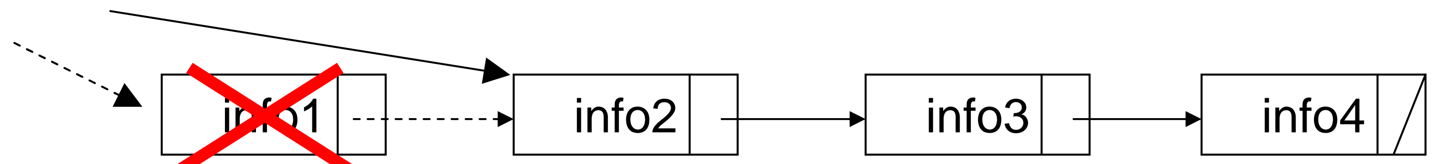
```
/* função busca: busca um elemento na lista */
Lista*  lst_busca(Lista* lis,int v){
  Lista   *p ;
  for (p = lis; p != NULL; p = p->prox)  {
      if (p->info == v)
         return p ;
  }
  return NULL;      /* não achou o elemento */
}
```

Se não achou elemento, retorna o endereço nulo!

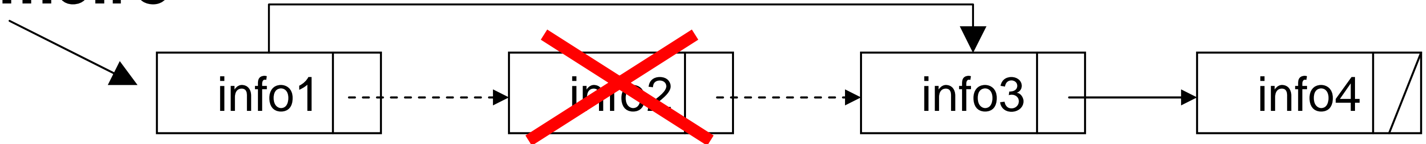
Retorna o endereço do primeiro nó que contém o elemento buscado

Removendo Elementos de Listas Encadeadas

- Dois casos devem ser tratados para a remoção de um elemento da lista



Remoção do primeiro elemento da lista



Remoção de um elemento no meio da lista

Removendo Elementos de Listas Encadeadas

```
Lista* lst_retira(Lista* lis,int v){
    Lista* ant = NULL; /* guarda elementoanterior */
    Lista* p = lis;/* para percorrer a lista */
  while(p != NULL && p->info != v){
      ant  =  p ;
      p  =  p->prox ;
  }
/* verifica se achou o elemento */
  if (p == NULL)
     return lis;
  if (ant == NULL)
     lis = p->prox;
  else
     ant->prox = p->prox;
  free(p) ;
  return lis ;
}
```

Procura elemento na lista,guardando anterior

Não achou, retorna lista original

Retira elemento do início

Retira elemento do meio

Liberando Listas Encadeadas

```
void    lst_libera(Lista* lis){
  Lista* p  =  lis ;
  while(p != NULL )  {
     Lista* t = p->prox;
     free(p); /* libera a memória apontada por "p"*/
     p = t;   /* faz  "p" apontar para o próximo */
  }
}
```

Guarda o endereço do próximo nó para poder libera-lo na próxima iteração

TAD Lista de Inteiros

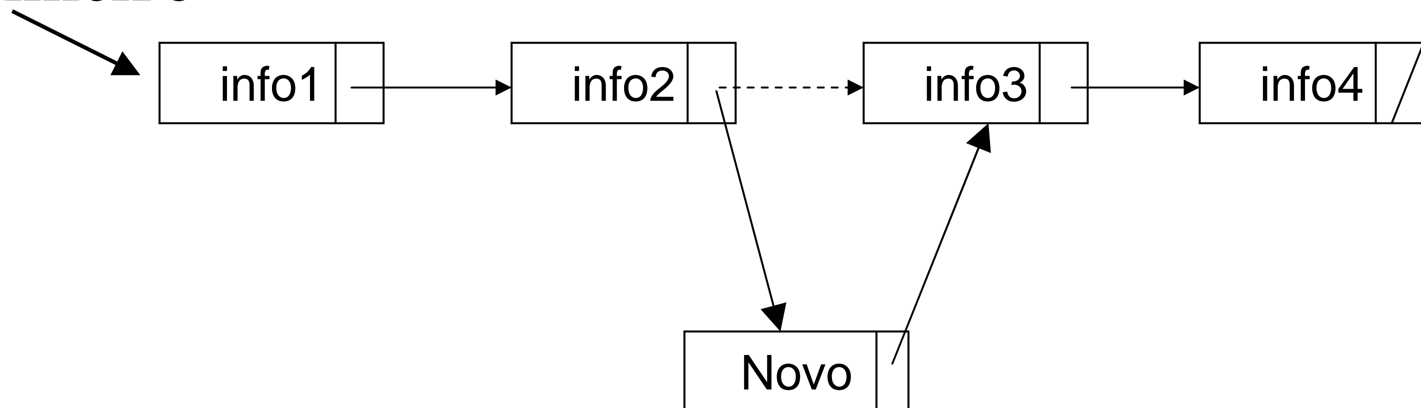
```
/* interface dada  por  lista.h  */
/* TAD: lista de inteiros */

typedef   struct  lista   Lista ;

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

Lista* lst_insere(Lista* lis,int i) ;
Lista* lst_retira(Lista* lis,int v) ;

int lst_vazia(Lista* lis);
Lista* lst_busca(Lista* lis,int v);
void   lst_imprime(Lista* lis);
```

TAD Lista de Inteiros

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

Inserindo Elementos de Forma Ordenada em Listas Encadeadas

- Já sabemos inserir no começo da lista
- Problema: Como inserir no meio da lista?



Inserindo Elementos de Forma Ordenada em Listas Encadeadas

```
Lista* lst_insere_ordenado(Lista* lis,int v ) {
  Lista* novo ; Lista* ant = NULL; Lista* p = lis  ;

  while(p != NULL && p->info < v) {
       ant = p ;
       p = p->prox;
  }
  novo = (Lista*)malloc(sizeof(Lista));
  novo->info  = v ;
  if (ant == NULL) {
       novo->prox  =  lis ;
       lis = novo ;
  } else {
       novo->prox = ant->prox  ;
       ant->prox  = novo ;
  }
  return  lis ;
}
```

Procura posição do elemento

Criando um novo nó

Inserindo no começo

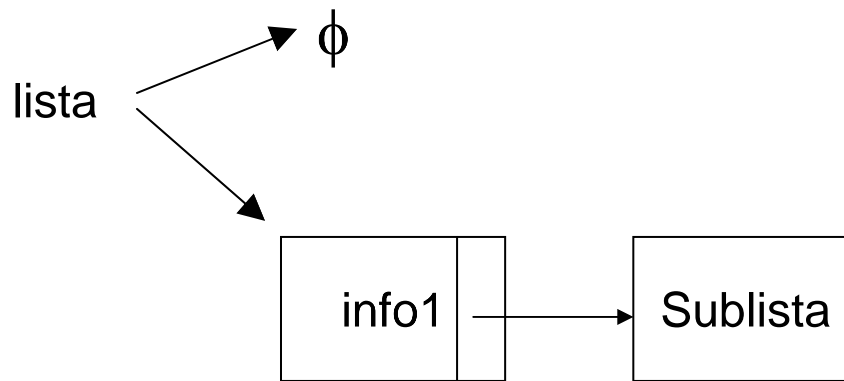
Inserindo no meio

Comparando Listas Encadeadas

```
int   lst_igual(Lista* lst1,Lista* lst2){
  Lista* p1 ; /* ponteiro para percorrer lst1 */
  Lista* p2 ;   /* ponteiro para percorrer lst2 */
  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) ;
}
```

Implementação Recursiva de Listas

- Uma lista pode ser definida recursivamente como:
  - Lista vazia; ou
  - Elemento seguido de uma (sub)lista.



Implementação Recursiva de Listas

- Impressão Recursiva

```
void lst_imprime_recursiva(Lista* lis){
  if (!lst_vazia(lis)) {
     /* imprime o primeiro elemento */
     printf("info: %d\n", lis->info);
     /* imprime sub_lista */
     lst_imprime_recursiva (lis->prox);
  }
}
```

Implementação Recursiva de Listas

- Função de remoção recursiva

```
Lista* lst_retira_recursiva (Lista* lis, int v) {
  if (!lst_vazia(lis)) {
     if (lis->info == v) {
        Lista*  t  =  lis;  /*  temporário  para  poder
                                  liberar */
        lis = lis->prox;
        free(t);
     }else {

        lis->prox=lst_retira_recursiva(lis->prox,v);
     }
  }
  return l;
}
```

Verifica se o elemento a ser retirado é o primeiro

Senão retira da sub-lista

Implementação Recursiva de Listas

- Liberando a lista recursivamente

```
void lst_libera_recursiva (Lista* lis) {
  if (!lst_vazia(lis)) {
      lst_libera_recursiva(lis->prox);
      free(lis);
  }
}
```

Implementação Recursiva de Listas

- Comparando listas recursivamente

```
int lst_igual_recursiva (Lista* lis1, Lista* lis2) {
  if (lis1 == NULL && lis2 == NULL)
     return 1;
  else if (lis1 == NULL || lis2 == NULL)
     return 0;
  else
     return lis1->info == lis2->info &&
      lst_igual_recursiva(lis1->prox, lis2->prox);
}
```