Linguagem C: Listas

Luis Martí
Instituto de Computação
Universidade Federal Fluminense
lmarti@ic.uff.br - http://lmarti.com

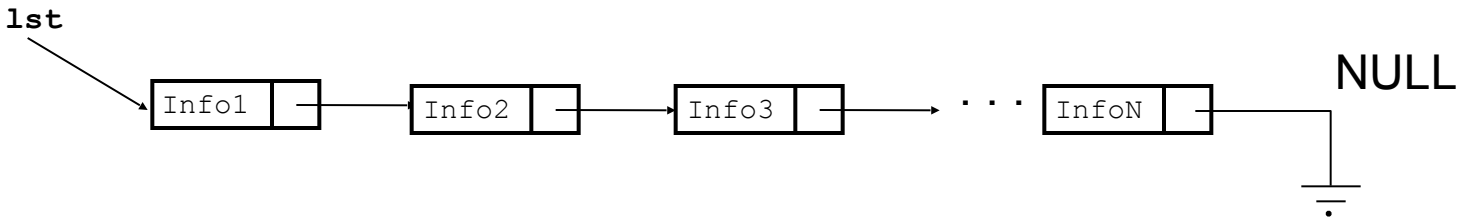
Tópicos Principais

- Listas Encadeadas
  - Definição
  - Funções básicas
- Listas Ordenadas
- Implementações recursivas

Listas Encadeadas

- 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

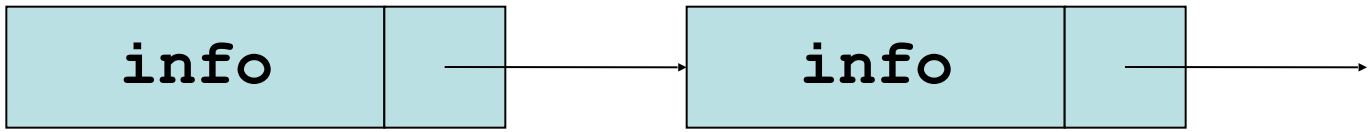
Listas Encadeadas



- Lista encadeada:
  - seqüência 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

Listas Encadeadas

```
struct elemento {
    int info;
    struct elemento* prox;
};
typedef struct elemento Elemento;
```



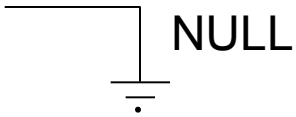
Exemplo: Lista de inteiros

```
struct elemento {
    int info;
    struct elemento* prox;
};
typedef struct elemento Elemento;
```

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

Listas Encadeadas de inteiros: Criação

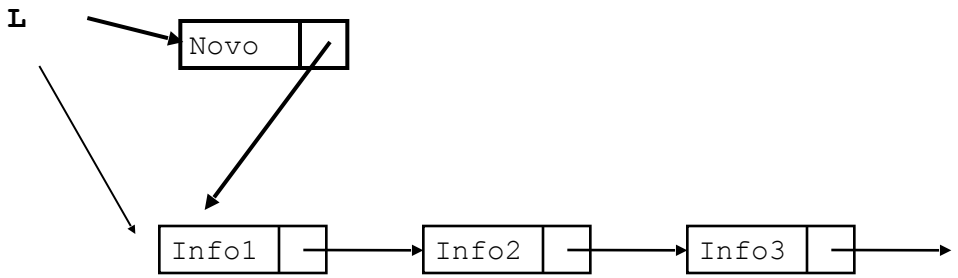
```
/* função de criação: retorna uma lista vazia */
Elemento* lst_cria (void)
{
   return NULL;
}
```



Cria uma lista vazia, representada pelo ponteiro NULL

Listas encadeadas de inteiros: Inserção

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




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

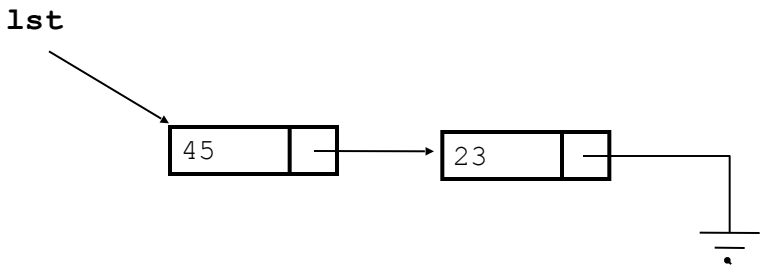
Listas Encadeadas: exemplo

- cria uma lista inicialmente vazia e insere novos elementos

```
int main (void)
{
   Elemento* lst;              /* declara uma lista não inicializada */
   lst = lst_cria();        /* cria e inicializa lista como vazia */

   lst = lst_insere(lst, 23); /* insere na lista o elemento 23 */
   lst = lst_insere(lst, 45); /* insere na lista o elemento 45 */
   ...
   return 0;
}
```

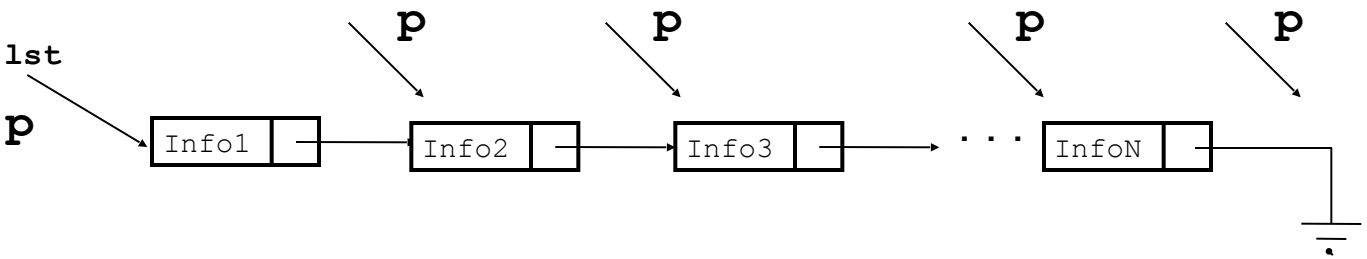
deve-se atualizar a variável que representa a lista a cada inserção de um novo elemento.



Listas Encadeadas: impressão

- Imprime os valores dos elementos armazenados

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



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 (Elemento* lst)
{
   return (lst == NULL);
}
```

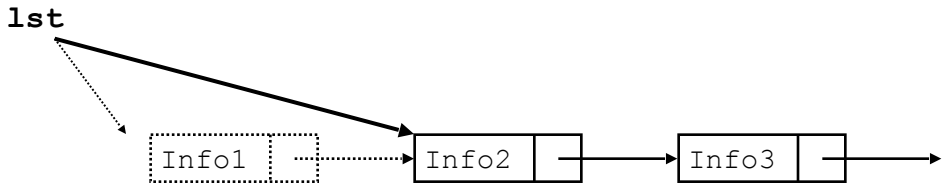
Listas Encadeadas: busca

- 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

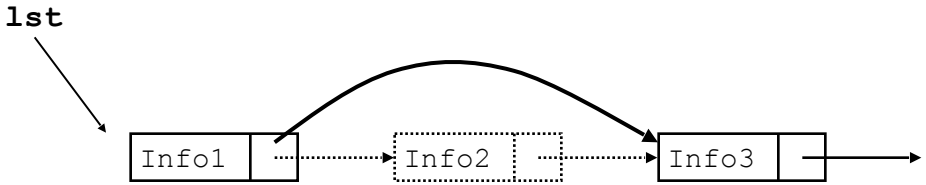
```
/* função busca: busca um elemento na lista */
Elemento* busca (Elemento* lst, int v)
{ Elemento* p;
   for (p=lst; p!=NULL; p = p->prox) {
      if (p->info == v)
         return p;
   }
   return NULL;        /* não achou o elemento */
}
```

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



```c
/* função retira: retira elemento da lista */
Elemento* lst_retira (Elemento* lst, int val)
{
   Elemento* ant = NULL; /* ponteiro para elemento anterior */
   Elemento* 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;
}
```

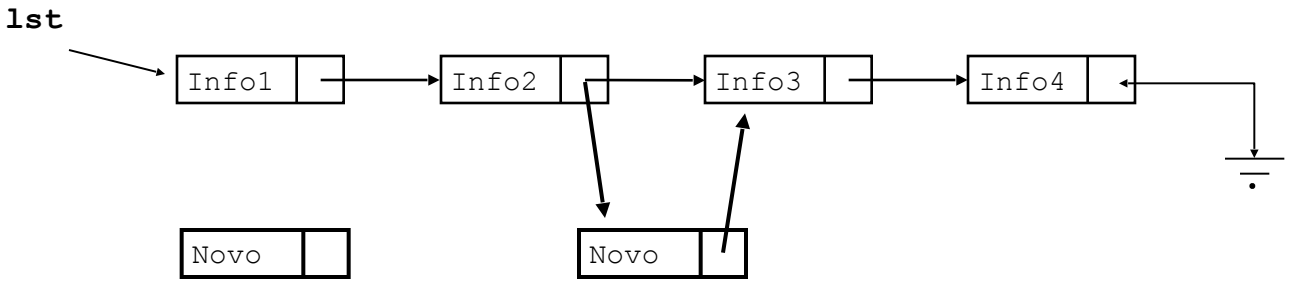
Listas Encadeadas: Libera a lista

- destrói a lista, liberando todos os elementos alocados

```
void lst_libera (Elemento* lst)
{
   Elemento *p = lst, *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       */
   }
}
```

Listas Encadeadas Ordenadas

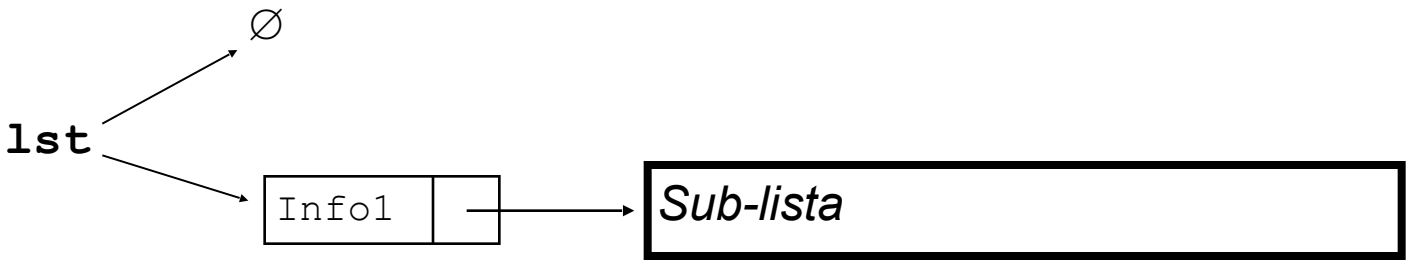
- Manutenção da lista ordenada
  - função de inserção percorre os elementos da lista até encontrar a posição correta para a inserção do novo



```
/* função insere_ordenado: insere elemento em ordem */
Elemento* lst_insere_ordenado (Elemento* lst, int val)
{
   Elemento* novo;
   Elemento* ant = NULL;  /* ponteiro para elemento anterior */
   Elemento* 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 = (Elemento*) malloc(sizeof(Elemento));
   novo->info = val;
   /* encadeia elemento */
   if (ant == NULL)
              { /* insere elemento no início  */
                novo->prox = lst; lst = novo; }
       else { /* insere elemento no meio da lista */
                novo->prox = ant->prox;
                ant->prox = novo; }
   return lst;
}
```

Definição recursiva de lista

- uma lista é
  - uma lista vazia; ou
  - um elemento seguido de uma (sub-)lista



Exemplo - Função recursiva para imprimir uma lista

- se a lista for vazia, não imprima nada
- caso contrário,
  - imprima a informação associada ao primeiro nó, dada por **`lst->info`**
  - imprima a sub-lista, dada por **`lst->prox`**, chamando recursivamente a função

Função imprime recursiva

```
/* Função imprime recursiva */
void lst_imprime_rec (Elemento* lst)
{
   if ( ! lst_vazia(lst)) {
      /* imprime primeiro elemento */
      printf("info: %d\n",lst->info);
      /* imprime sub-lista    */
      lst_imprime_rec(lst->prox);
   }
}
```

Função imprime recursiva invertida

```
/* Função imprime recursiva invertida */
void lst_imprime_rec (Elemento* lst)
{
   if ( !lst_vazia(lst) ) {
      /* imprime sub-lista    */
      lst_imprime_rec(lst->prox);
      /* imprime ultimo elemento   */
      printf("info: %d\n",lst->info);

   }
}
```

Exemplo - função para retirar um elemento da lista

- retire o elemento, se ele for o primeiro da lista (ou da sub-lista)
- caso contrário, chame a função recursivamente para retirar o elemento da sub-lista

Lista: Retira recursiva

```
/* Função retira recursiva */
Elemento* lst_retira_rec (Elemento* lst, int val)
{
   Elemento* t;
   if (!lst_vazia(lst)) {
      /* verifica se elemento a ser retirado é o primeiro */
      if (lst->info == val) {
         t = lst;  /* temporário para poder liberar */
         lst = lst->prox;
         free(t);
      }
      else {
         /* retira de sub-lista */
         lst->prox = lst_retira_rec(lst->prox,val);
      }
   }
   return lst;
}
```

é necessário reatribuir o valor de **`l->prox`** na chamada recursiva, já que a função pode alterar a sub-lista

Igualdade de listas

```
int lst_igual (Elemento* lst1, Elemento* 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

Listas iguais: não recursiva

```
int lst_igual (Elemento* lst1, Elemento* lst2)
{
   Elemento* p1;  /* ponteiro para percorrer l1 */
   Elemento* 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;
}
```

Igualdade de listas (recursiva)

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

- implementação recursiva
  - se as duas listas dadas são vazias, são iguais
  - se não forem ambas vazias, mas uma delas é vazia, são diferentes
  - se ambas não forem vazias, teste
    - se informações associadas aos primeiros nós são iguais e
    - se as sub-listas são iguais

Listas iguais: recursiva

```
int lst_igual (Elemento* lst1, Elemento* lst2)
{
   if (lst1 == NULL && lst2 == NULL)
      return 1;
   else if (lst1 == NULL || lst2 == NULL)
      return 0;
   else
      return (lst1->info == lst2->info) &&
             lst_igual(lst1->prox, lst2->prox);
}
```

Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004)

- Capítulo 10 – Listas Encadeadas

Material adaptado por Luis Martí a partir dos slides de José Viterbo Filho que forem elaborados por Marco Antonio Casanova e Marcelo Gattas para o curso de Estrutura de Dados para Engenharia da PUC-Rio, com base no livro *Introdução a Estrutura de Dados*, de Waldemar Celes, Renato Cerqueira e José Lucas Rangel, Editora Campus (2004).