Aula 6 - Listas Dinâmicas

Listas Lineares

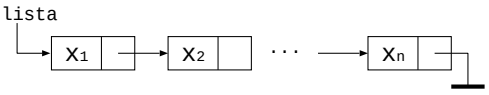
Listas sequenciais:

- algumas operações exigem um grande esforço computacional
- baixo desempenho
- estimativa do tamanho máximo
- se aplicação requer ultrapassar o tamanho máximo?

Na aula anterior vimos que algumas operações sobre listas lineares sequenciais exigem um grande esforço computacional, o que pode ser um fator determinante do baixo desempenho do programa, caso estas operações sejam frequentemente executadas. Além disso, não é possível alterar dinamicamente o tamanho máximo da lista.

Lista Encadeada

- os elementos da lista são registros com um dos componentes destinado a guardar o endereço do registro sucessor
- posições não contíguas de memória



Cada item da lista é encadeada com o seguinte mediante uma variável do tipo apontador. Este tipo de implementação permite utilizar posições não contíguas de memória, sendo possível inserir e eliminar elementos sem haver a necessidade de deslocar os itens seguintes da lista.

Características

- inserção e eliminação sem deslocamento
- cada nó (registro ou célula) contém um item da lista e um apontador para o próximo nó
- um apontador armazena o endereço do primeiro nó da lista

Estrutura

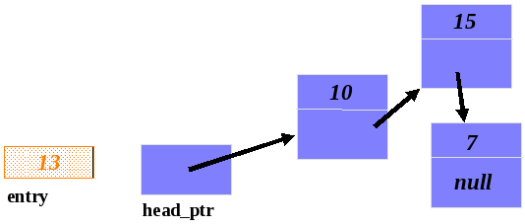
```
typedef int elem_t;
typedef struct no_lista {
  elem_t v;
  struct no_lista *prox;
} No_lista;

typedef No_lista* Lista;
```

Inicialmente

- `head_ptr` é um ponteiro que possui o endereço o primeiro elemento da lista
- `head_ptr` inicialmente deve conter `null`, que implica em lista vazia

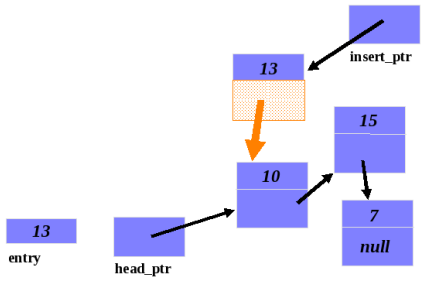
Inserção no início

- vamos inserir 13 no início da lista



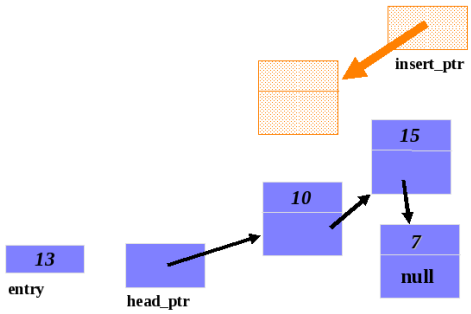
Inserção no início

- novo nó aponta para o primeiro elemento da lista



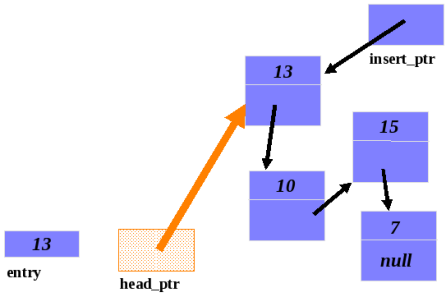
Inserção no início

- cria um novo nó



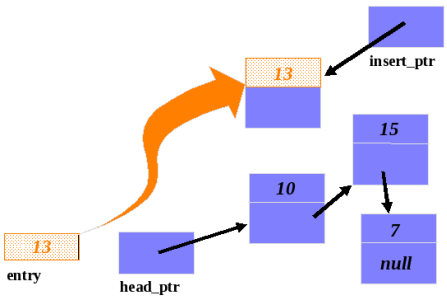
Inserção no início

- `head_ptr` aponta para novo nó



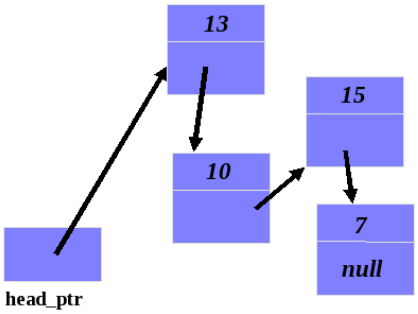
Inserção no início

- insere o valor 13 no nó criado



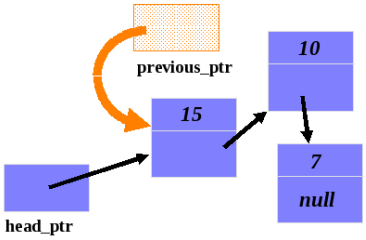
Inserção no início

- após a inserção:



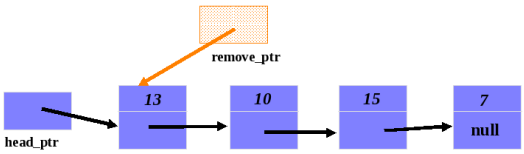
Inserção

- inserir 13 após o 15
- `previou_ptr` ponteiro auxiliar



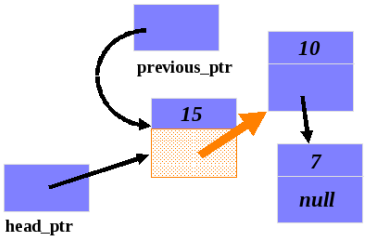
Eliminação do primeiro elemento

- `remove_ptr` aponta para o início da lista



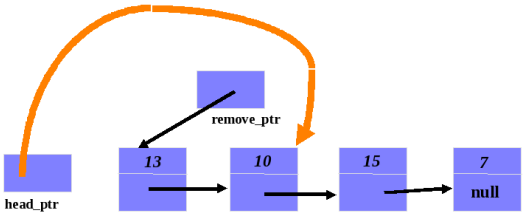
Inserção

- verifica conteúdo do auxiliar: `*previou_ptr`
- `previou_ptr`: aponta p/ a sublista com 10 e 7



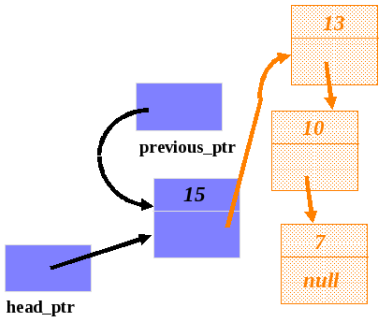
Eliminação do primeiro elemento

- `head_ptr` deve apontar para o próximo do `remove_ptr`



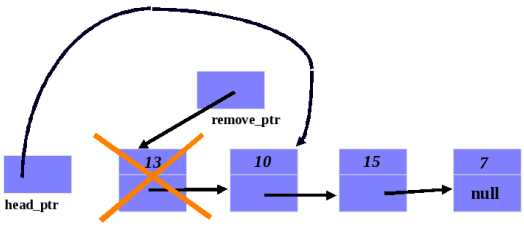
Inserção

- insere o elemento no início da sublista



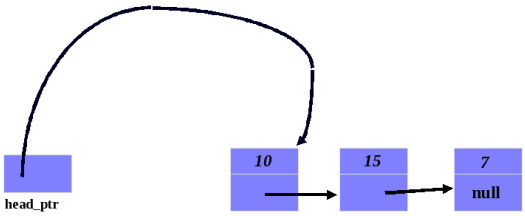
Eliminação do primeiro elemento

- libera espaço de memória



Eliminação do primeiro elemento

- após a eliminação:



Vantagens

- não é necessário pré-definir um tamanho máximo para a lista
  - limite é o tamanho da memória
  - não há alocação desnecessária de espaço
- economia de memória

Desvantagens

- acesso não é indexado – necessário percorrer i nós para encontrar o i-ésimo elemento
- consumo de tempo para alocação e liberação de memória em operações de inserção e remoção

Implementação

- `void cria(Lista *p_l);`
- `int vazia(Lista *p_l);`
- `void insere_inicio(Lista *p_l, elem_t e);`
- `void insere_fim(Lista *p_l, elem_t e);`
- `int insere_ordenado(Lista *p_l, elem_t e);`
- `int ordenada(Lista *p_l);`
- `void ordena(Lista *p_l);`
- `int remove_inicio(Lista *p_l, elem_t *p_e);`
- `int remove_fim(Lista *p_l, elem_t *p_e);`

- `int remove_valor(Lista *p_l, elem_t e);`
- `void inverte(Lista *p_l);`
- `void libera(Lista *p_l);`
- `void exibe(Lista *p_l);`

Bibliografia

- Michael Main and Walter Savitch, *Data Structures and Other Objects Using C++*, 2. edição, Addison Wesley, 2004.
- Roberto Ferrari, *Curso de estruturas de dados*, São Carlos, 2006. Apostila disponível em: `http://www2.dc.ufscar.br/~ferrari/ed/ed.html`