Listas

Estrutura de Dados

Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação

Conteúdo

Listas

Definição

- Uma lista é uma estrutura linear, composta de um conjunto de n elementos $x_1, x_2, \ldots, x_n$ chamados nós, organizados de forma a manter uma relação entre eles, isto é:
  - se $n = 0$, então a lista está vazia.
  - se $n > 0$ então,
    - x_1 é o primeiro nó;
    - x_n é o último nó;
    - para $k \in \mathbb{Z}$ e $1 < k < n$, o nó x_k é precedido pelo nó x_{k-1} e seguido pelo nó x_{k+1}.

Aplicações de Listas

- Lista de valores inteiros (ex: lista dos números primos)
- Lista de tarefas
- Lista de objetos geométricos (em uma aplicação gráfica, jogo etc.)
- Implementação de outras estruturas de dados
  - Pilhas
  - Filas

	2	3	5	7	11	13	17	19	23
29	31	37	41	43	47	53	59	61	67
71	73	79	83	89	97	101	103	107	109
113	127	131	137	139	149	151	157	163	167
173	179	181	191	193	197	199	211	223	227
229	233	239	241	251	257	263	269	271	277
281	283	293	307	311	313	317	331	337	347
349	353	359	367	373	379	383	389	397	401
409	419	421	431	433	439	443	449	457	461
463	467	479	487	491	499	503	509	521	523
541	547	557	563	569	571	577	587	593	599
601	607	613	617	619	631	641	643	647	653
659	661	673	677	683	691	701	709	719	727
733	739	743	751	757	761	769	773	787	797
809	811	821	823	827	829	839	853	857	859
863	877	881	883	887	907	911	919	929	937
941	947	953	967	971	977	983	991	997	

Listas

- Operações comuns:
  - consultar (*get*) o nó x_k;
  - atribuir (*set*) um novo valor ao nó x_k;
  - inserir um novo nó antes ou depois do nó x_k ou em um dos extremos da lista;
  - remover o nó x_k;
- Uma lista é uma estrutura de dados **dinâmica**, isto é, o seu comprimento (= número de nós) **pode ser alterado em tempo de execução**.
- Outras operações:
  - concatenar duas listas;
  - determinar o comprimento n de uma lista;
  - localizar o nó que contém um determinado dado;
  - partir uma lista em duas ou mais;
  - fazer uma cópia de uma lista;
  - ordenar os nós de uma lista;
  - etc.

Representação de Listas

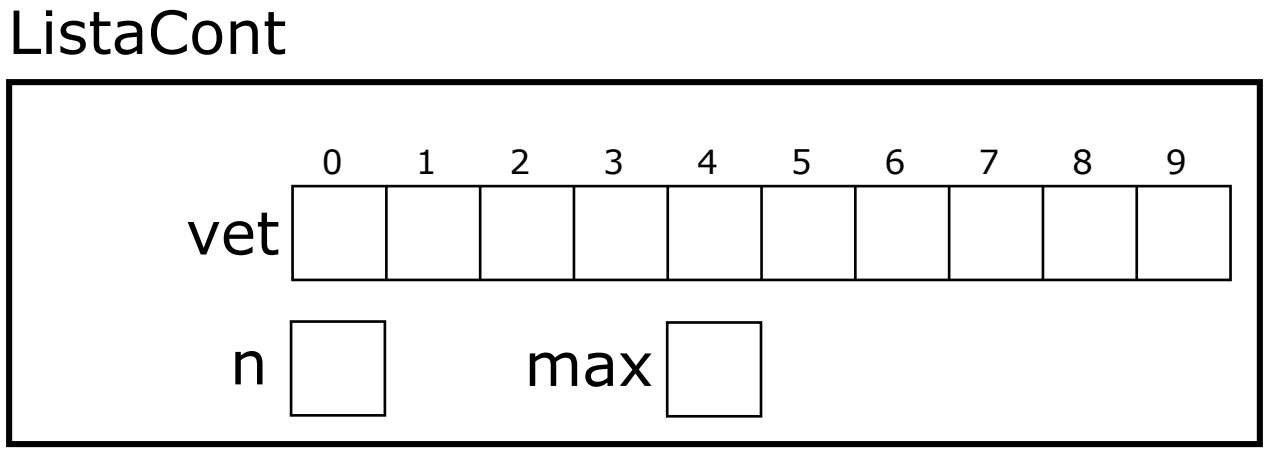
- Existem várias maneiras de representar uma lista, devendo ser escolhida a mais adequada para a aplicação em questão (melhor desempenho, melhor compatibilidade com o tipo de memória usada e etc).
- As representações mais comuns são por:
  - **contiguidade** dos nós;
  - **encadeamento** dos nós.

Listas Contíguas

Listas Contíguas

- Usando a alocação contígua (ou sequencial), os nós da lista são alocados fisicamente em posições consecutivas da memória.
- Assim, é comum usar um vetor como meio de alocação de posições consecutivas, onde podem ser armazenados os nós de uma lista.
- Esquematicamente, uma lista contígua com n nós poderia ser armazenada em um vetor `vet` com n elementos.
- Mas como lista é uma **estrutura de dados dinâmica**, pode ser necessário a utilização de mais de n nós. Desta forma, o mais usual é trabalhar com um vetor de max elementos, sendo $max \geq n$.

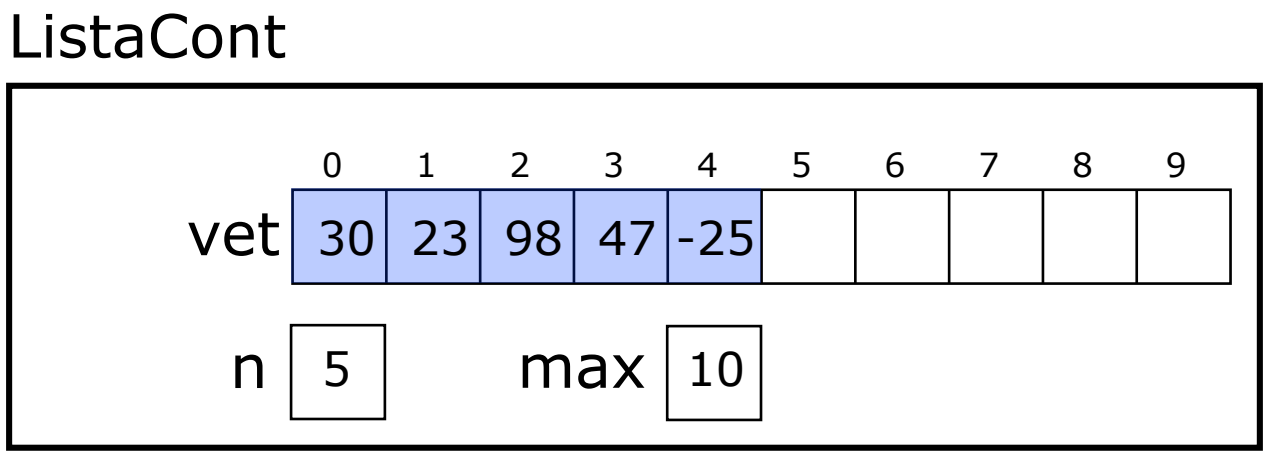
Listas Contíguas



Listas Contíguas

Exemplo de uma lista de valores inteiros usando a representação por contiguidade dos nós:

- Capacidade máxima: 10.
- Itens armazenados: 5.
- Vamos trabalhar apenas com listas de valores inteiros por simplicidade, mas pode-se criar listas de qualquer tipo de dados (lista de alunos, lista de pontos etc.).



Listas Contíguas

- TAD `ListaCont` para listas de inteiros.
  - Construtor
  - Destrutor
  - *Get* (consulta) / *Set* (atribui)
  - Insere no final da lista
  - Remove o último nó (final da lista)
  - Insere um novo nó na posição k
  - Remove o nó da posição k

Listas Contíguas

```
class ListaCont
{
 public:
   ListaCont(int tam);
   ~ListaCont();

   int get(int k);
   void set(int k, int val);
   void insereFinal(int val);     // insere no final
   void removeFinal();            // remove ultimo
   void insereK(int k, int val); // antes de xk
   void removeK(int k);           // remove xk

 private:
   int max;     // capacidade maxima de elementos
   int n;       // quantidade de nos da lista
   int *vet;    // vetor que armazena a lista
};
```

Listas Contíguas

- Construtor
  - Inicializa os dados.
  - Aloca memória de forma dinâmica para o vetor que representa a lista.
- Destrutor: desaloca a memória alocada de forma dinâmica.

```
ListaCont::ListaCont(int tam)
{
  max = tam;
  n = 0;
  vet = new int[max];
}

ListaCont::~ListaCont()
{
  delete [] vet;
}
```

Listas Contíguas

```
int ListaCont::get(int k)
{
  if(k >= 0 && k < n)
    return vet[k];
  else
  {
    cout << "Indice invalido!" << endl;
    exit(1);
  }
}

void ListaCont::set(int k, int val)
{
  if(k >= 0 && k < n)
    vet[k] = val;
  else
  {
    cout << "Indice invalido!" << endl;
    exit(2); }
}
```

Listas Contíguas

- Inserir um novo nó na posição final da lista é simples.

```
void ListaCont::insereFinal(int val)
{
  if(n == max)
  {
    cout << "Vetor Cheio!" << endl;
    exit(3);
  }

  vet[n] = val;
  n = n + 1;
}
```

Listas Contíguas

- Remover o último nó da lista é simples.
- Basta modificar o valor de `n`.

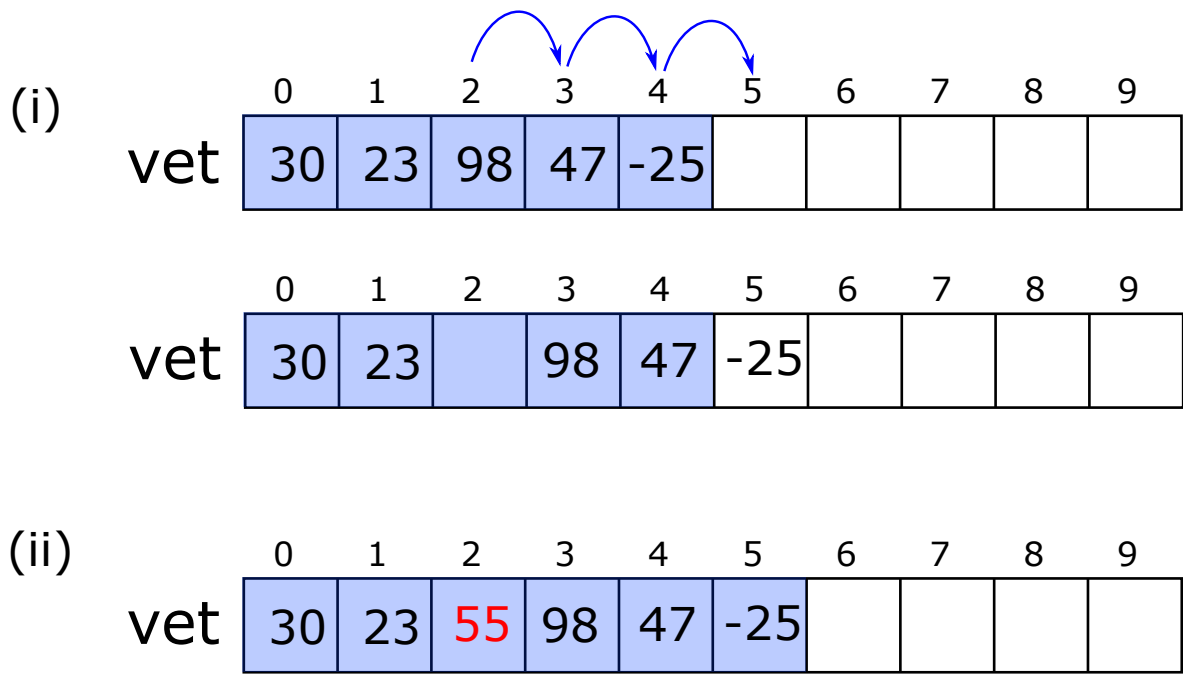
```
void ListaCont::removeFinal()
{
  if(n == 0)
  {
    cout << "Lista Vazia!" << endl;
    exit(6);
  }

  n = n - 1;
}
```

Listas Contíguas

- Para inserir um novo nó na posição k é preciso "abrir espaço" e deslocar os itens à frente do nó x_k para direita.

lista.insereK(2,-55);



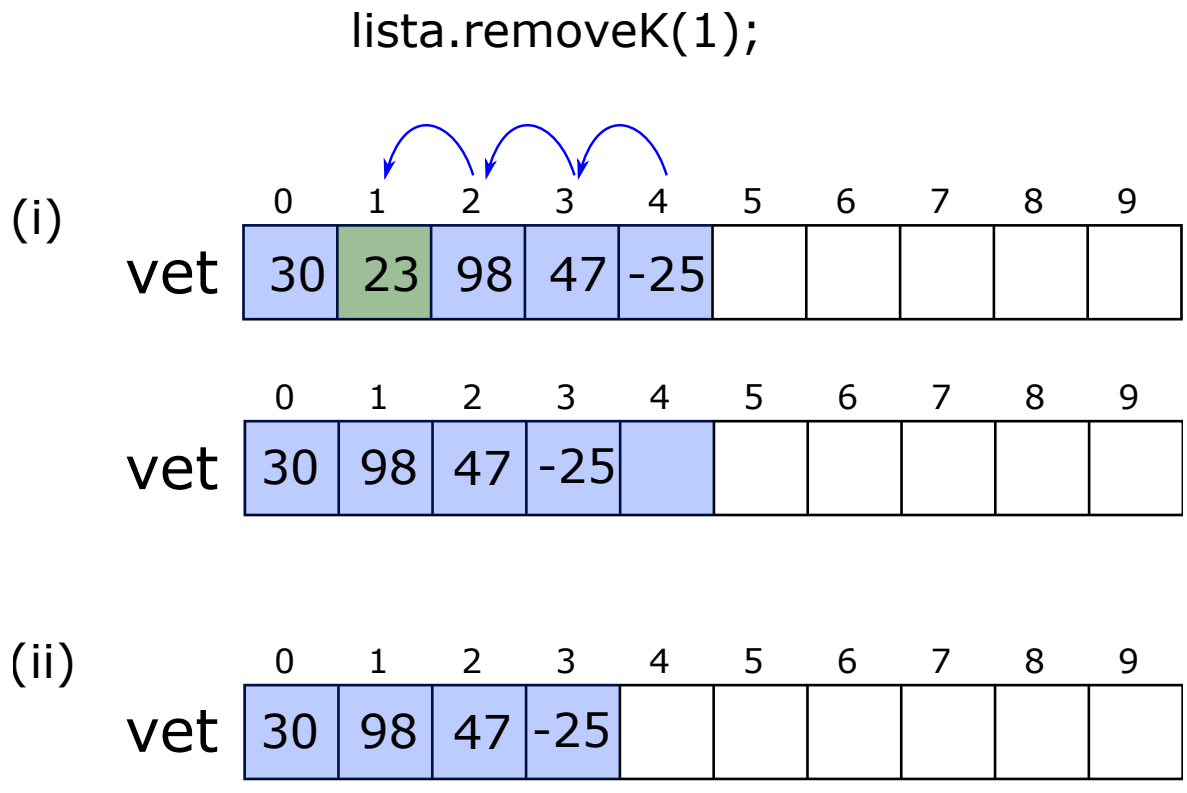
Listas Contíguas

```
void ListaCont::insereK(int k, int val)
{
  if(n == max) {
    cout << "Vetor Cheio!" << endl;
    exit(3);
  }
  if(k >= 0 && k < n)
  {
    for(int i = n-1; i >= k; i--)
      vet[i+1] = vet[i];

    vet[k] = val;
    n = n + 1;
  }
  else
  {
    cout << "Indice invalido!" << endl;
    exit(4);
  }
}
```

Listas Contíguas

- Para remover um nó da posição k é preciso "fechar o seu espaço" e deslocar os itens à frente do nó x_k para esquerda.



Listas Contíguas

```
void ListaCont::removeK(int k)
{
  if(k >= 0 && k < n)
  {
    // copia da dir. para esq.
    for(int i = k; i < n-1; i++)
      vet[i] = vet[i+1];

    n = n - 1;
  }
  else
  {
    cout << "Indice invalido!" << endl;
    exit(5);
  }
}
```

Listas Contíguas

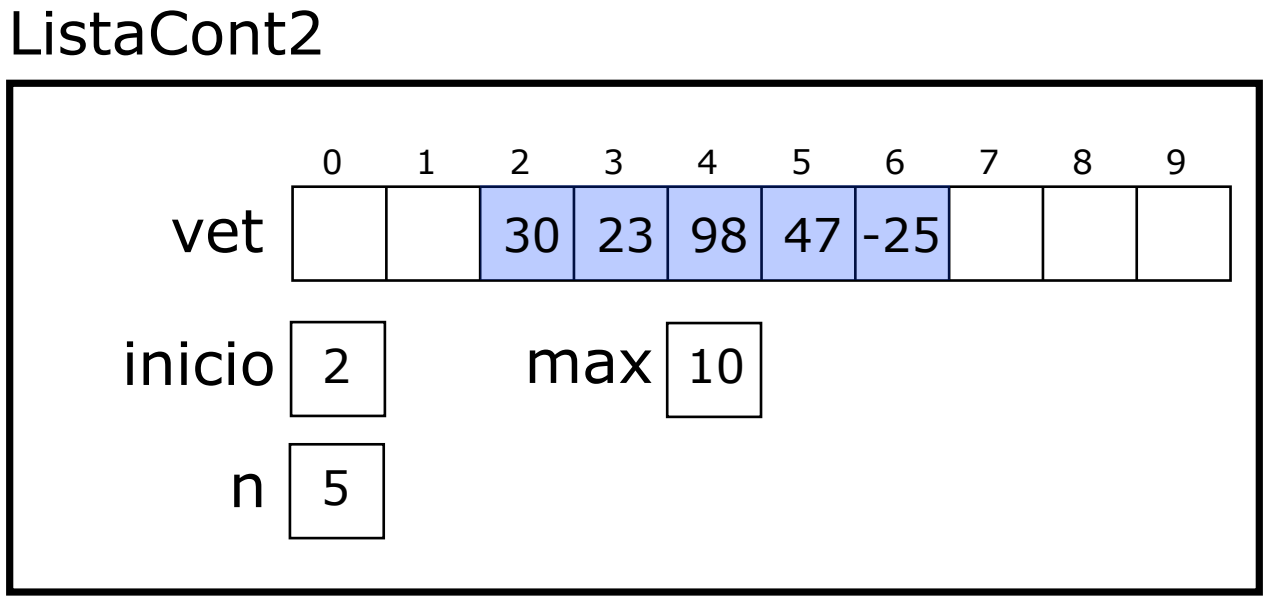
Complexidade

- Considere as operações:
  (i) inserir na última posição livre; ou
  (ii) inserir em uma posição qualquer da lista.
- Qual a complexidade das operações (i) e (ii)?
- E qual a complexidade de:
  (i) remover o último nó; ou
  (ii) remover um nó de uma posição qualquer da lista.

Listas Contíguas

Outra representação

- Pode-se trabalhar também com a seguinte forma alternativa de representar uma lista de forma contígua.
- Agora, além da quantidade, é necessário conhecer também o índice do primeiro nó da lista (`inicio`).



Listas Contíguas

- **Exercício 1**: defina a classe do TAD `ListaCont2` e implemente as suas operações utilizando essa nova representação.
- Desenvolver também novas operações:
  - remover o nó do início;
  - inserir um nó antes do início.

Listas Contíguas

- **Exercício 2**: desenvolva um programa que cria uma lista de valores inteiros (`ListaCont` ou `ListaCont2`) com capacidade máxima 100 e, em seguida, leia do teclado valores inteiros até que um valor negativo seja lido. Para cada valor lido, inserir o mesmo na lista somente se este for um número par.
- Ao final, calcule e imprima a soma dos elementos da lista.

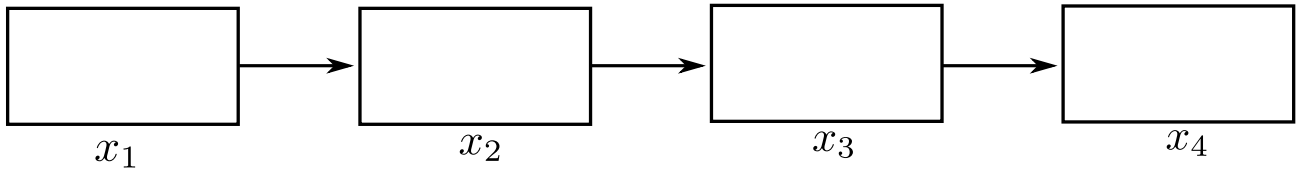
Listas Encadeadas

Listas Encadeadas

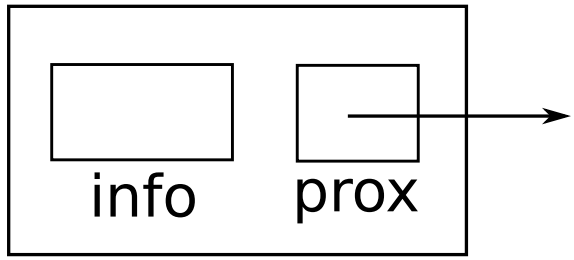
- Existe outra forma de representar uma lista.
- Antes de seguir, pense na representação contígua:
  - Desvantagens?
  - Vantagens?
- **Lista encadeada.**
- Serão apresentados os seguintes casos de listas encadeadas:
  - Lista Simplesmente Encadeada
  - Lista com Descritor
  - Lista Duplamente Encadeada
  - Lista Circular

Listas Encadeadas

- Nessa estrutura de dados, um nó deve conter além de seu valor, um ponteiro para o nó seguinte, representando uma sequência dos nós da lista.
- Esquematicamente:



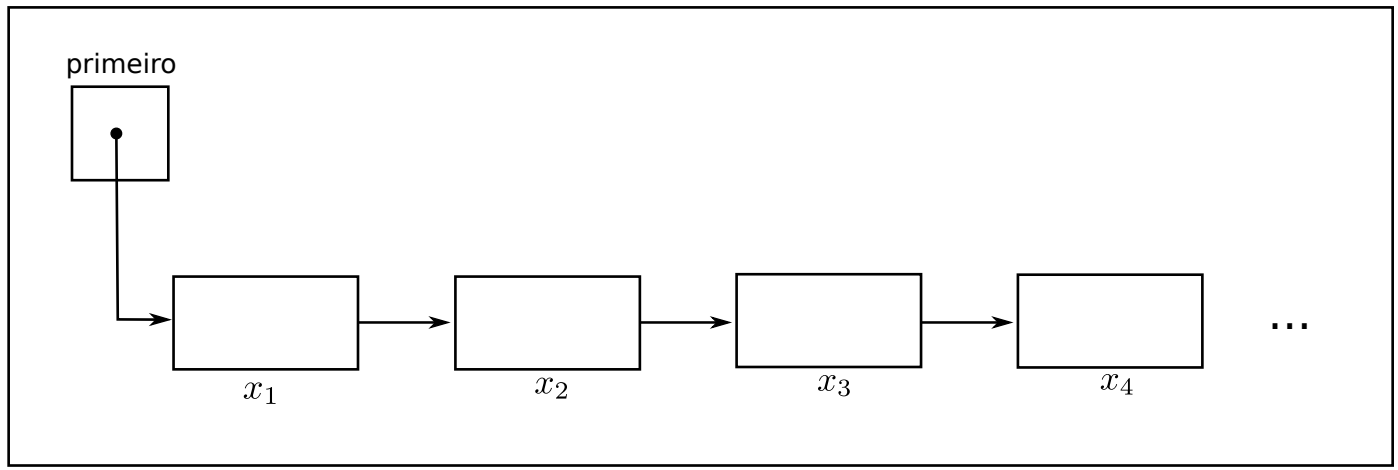
- Assim, um nó passa a ter a seguinte representação:
  - `info`: informação a ser armazenada na lista (ex: `int`, `float`, `Ponto2D` etc.)
  - `prox`: ponteiro para o próximo nó da lista



Lista Simplesmente Encadeada

- Além disso, em uma lista simplesmente encadeada é necessário dispor de um ponteiro para o primeiro nó da lista.

Lista Encadeada



- Como cada nó contém um ponteiro para o próximo, pode-se acessar qualquer nó a partir do ponteiro `primeiro`.
- Como definir o final da lista?

Lista Simplesmente Encadeada

- Em C++, atribui-se a constante **NULL** (nulo ou vazio) a um ponteiro para indicar que ele não está apontando para nenhuma posição da memória.

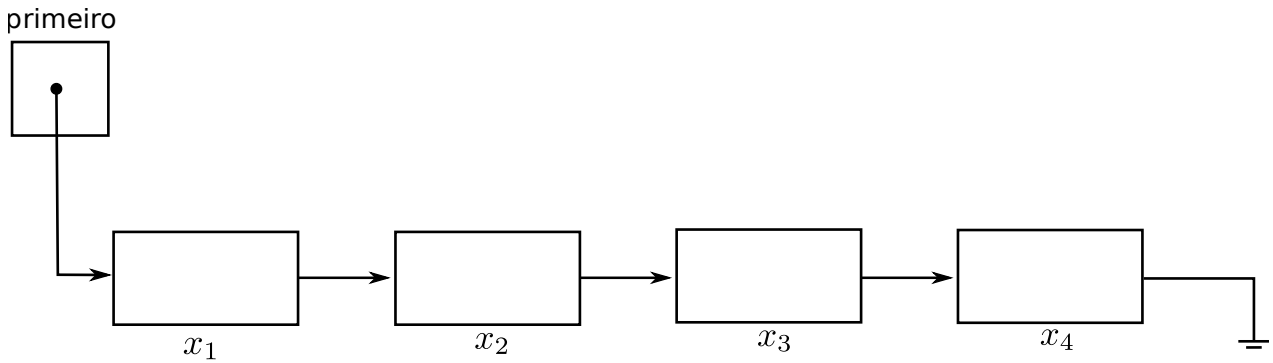
```
int x = 10;
int * px = &x;

px = NULL;   // px nao aponta mais para x
             // px nao aponta para mais nada
```

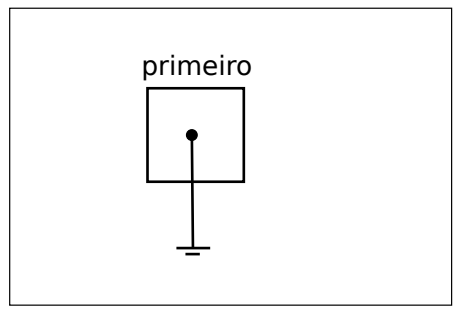
- Portanto, o ponteiro do último nó de uma lista simplesmente encadeada sempre termina com o valor `NULL`.

Lista Simplesmente Encadeada

- Seja uma lista com n nós.
- Se $n > 0$, então o ponteiro próximo do último nó é igual a `NULL`.



- Se $n = 0$, então a lista está vazia e assim `primeiro` é `NULL`.



- Ou seja, se `primeiro == NULL` a lista está vazia.

Lista Simplesmente Encadeada

- Uma lista simplesmente encadeada consiste de dois objetos distintos:
  - o nó;
  - e a própria lista.
- Assim, é necessário definir dois TADs: `No` e `ListaEncad`.
- A seguir são apresentados os TADs de uma lista simplesmente encadeada de valores inteiros.

Nó da Lista Simplesmente Encadeada

```
class No
{
  public:
    No();
    ~No();
    int getInfo();
    No* getProx();
    void setInfo(int val);
    void setProx(No *p);

  private:
    int info;   // informacao
    No *prox;   // ponteiro para o proximo
};
```

Nó da Lista Simplesmente Encadeada

```
No::No()  { }

No::~No() { }

int No::getInfo() {
  return info;
}

No* No::getProx() {
  return prox;
}

void No::setInfo(int val) {
  info = val;
}

void No::setProx(No *p) {
  prox = p;
}
```

Lista Simplesmente Encadeada

```
class ListaEncad
{
public:
   ListaEncad();
   ~ListaEncad();

   void insereInicio(int val);
   bool busca(int val);

 private:
   No* primeiro; // ponteiro para o primeiro
};
```

Lista Simplesmente Encadeada

```
ListaEncad::ListaEncad()
{
  primeiro = NULL;
}

void ListaEncad::insereInicio(int val)
{
  No *p = new No();  // cria No apontado por p

  p->setInfo(val);        // preenche informacao
  p->setProx(primeiro);   // preenche proximo

  primeiro = p;       // no apontado por p passa
                      // a ser o primeiro da lista
}
```

Lista Simplesmente Encadeada

- **Atenção**: o comando `No *p = new No()` aloca um novo nó de forma dinâmica, isto é, este novo nó só é criado quando a operação `insereInicio(val)` é executada.
- Portanto, novos nós são adicionados à lista em tempo de execução de acordo com a demanda da aplicação.
- Logo, em um instante de tempo, a memória ocupada pela lista é proporcional ao número de nós armazenados na mesma.
- Como é o uso de memória da lista contígua?

Lista Simplesmente Encadeada

```
bool ListaEncad::busca(int val)
{
  No* p = primeiro;

  while(p != NULL)
  {
    if(p->getInfo() == val)
      return true;

    p = p->getProx();
  }

  return false;
}
```

Lista Simplesmente Encadeada

- Também pode-se usar uma estrutura `for`...

```
bool ListaEncad::busca(int val)
{
  No* p;

  for(p = primeiro; p != NULL; p = p->getProx())
  {
    if(p->getInfo() == val)
      return true;
  }

  return false;
}
```

Lista Simplesmente Encadeada

- Destrutor

```
ListaEncad::~ListaEncad()
{
  No *p = primeiro;
  while(p != NULL)
  {
    No *t = p->getProx();
    delete p;

    p = t;
  }
}
```

Programa com uma Lista Simplesmente Encadeada

```
int main()
{
  ListaEncad l;

  int n, val;
  cin >> n;
  for(int i = 0; i < n; i++)
  {
    cin >> val;
    l.insereInicio(val);
  }

  l.imprime();

  return 0;
};
```

Lista Simplesmente Encadeada

Exercícios

1. Implemente a operação `imprime()` que imprime na tela todos os valores armazenados na lista.
2. Qual a saída do programa se os seguintes valores $5, 10, 20, 30, 40, 50$ forem lidos do teclado nessa ordem?
3. Desenvolver uma nova operação do TAD `ListaEncad` para inserir um novo nó na lista contendo o valor real `val` como o último nó da lista.

Lista Simplesmente Encadeada

- Operação para remover o primeiro nó da lista.

```
void ListaEncad::removeInicio()
{
  No* p;

  if(primeiro != NULL)
  {
    // p aponta para o No a ser excluido
    p = primeiro;

    // primeiro passa a apontar p/ atual segundo
    primeiro = p->getProx();

    delete p;
  }
}
```

Lista Simplesmente Encadeada

- Observação: Caso a lista esteja vazia, a operação de remoção de um nó pode prever a execução de uma rotina de tratamento de exceções, cujo parâmetro pode ser um valor inteiro que identifique uma exceção específica para as devidas providências.

```
void ListaEncad::removeInicio()
{
  No* p;
  if(primeiro != NULL)
  {
    p = primeiro;
    primeiro = p->getProx();
    delete p;
  } else {
    rotina_erro(1);
  }
}
```

Lista Simplesmente Encadeada

Exercícios

4. Desenvolver uma nova operação para remover o último nó da lista.
5. Desenvolver uma nova operação para contar e retornar o total de nós da lista.
6. Desenvolver uma nova operação para inserir um novo nó em uma determinada posição da lista, a qual será especificada por um parâmetro `k`.

Outras Listas Encadeadas

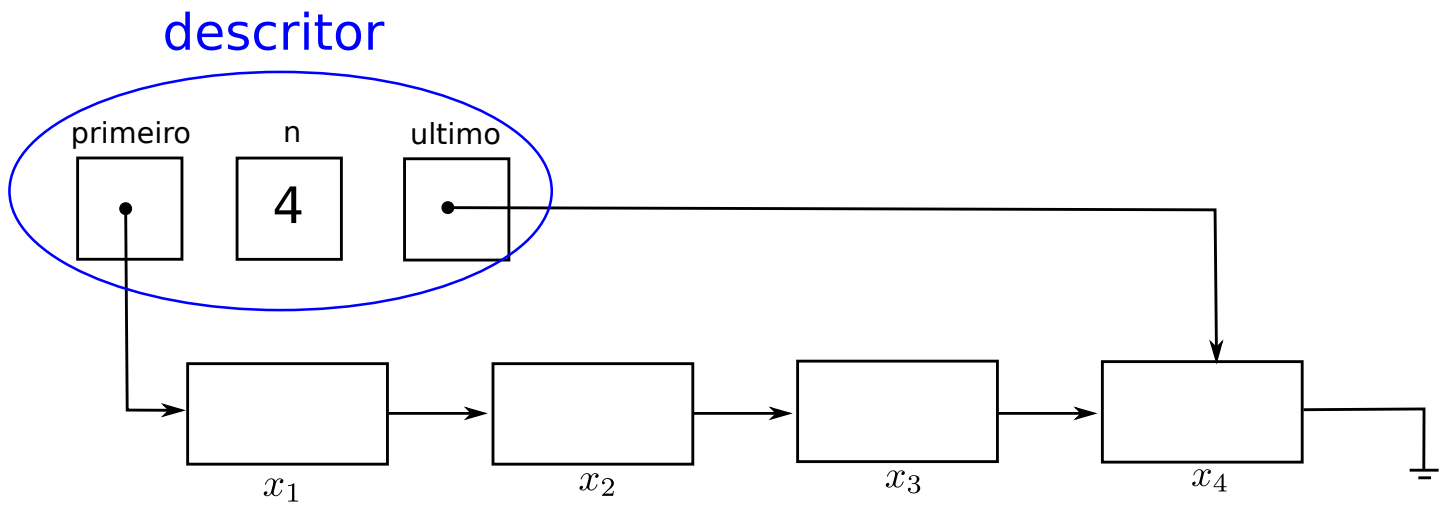
Outras Listas Encadeadas

Exercícios

- Existem várias outras formas de representar listas com encadeamento dos nós.
- Exemplos:
  - Lista com descritor
  - Lista duplamente encadeada
  - Lista circular

Lista com descritor

- Em algumas aplicações, torna-se interessante dispor de outras informações sobre a lista, como, por exemplo: o número de nós n, o endereço do último nó etc.
- O conjunto de dados que descrevem as propriedades da lista é chamado de **descritor** da lista.
- Nestes casos, a estrutura de dados lista poderia ser representada da seguinte forma:



Lista com descritor

- Tomando o TAD `ListaEncad` anterior como base, pode-se modificá-lo a fim de incluir outras informações (descritor).

```
class ListaEncad
{
public:
   ListaEncad();
   ~ListaEncad();
   void insereInicio(int val);
   void insereFinal(int val);
   void removeInicio();
   void removeFinal();
   bool busca(int val);
   void imprime();

 private:
   No* primeiro; // ponteiro para o primeiro
   No* ultimo;   // ponteiro para o ultimo
   int n;        // total de nos
};
```

Lista com descritor

- Nesse caso, algumas operações não sofrem alterações. Exemplos:
  - `imprime()`
  - `busca(int val)`
  - destrutor
  - etc ...
- Por outro lado, é preciso modificar outras operações para considerar os novos dados do descritor.

Lista com descritor

- Construtor

```
ListaEncad::ListaEncad()
{
  n = 0;
  primeiro = NULL;
  ultimo   = NULL;
}
```

Lista com descritor

- Outras operações precisam verificar a necessidade ou não de alterar também as informações do descritor da lista.
- Exemplo: inserir no início.

```
void ListaEncad::insereInicio(int val)
{
  No *p = new No();

  p->setInfo(val);
  p->setProx(primeiro);

  primeiro = p;

  n = n + 1;
  if(n == 1) ultimo = p;
}
```

Lista com descritor

- Exemplo: remover no início de uma lista encadeada sem descritor.

```
void ListaEncad::removeInicio()
{
  No *p;
  if(primeiro != NULL)
  {
    p = primeiro;
    primeiro = p->getProx();
    delete p;
  }
  else
    rotina_erro(2);
}
```

Lista com descritor

- Exemplo: remove no início.

```
void ListaEncad::removeInicio()
{
  No *p;
  if(primeiro != NULL)
  {
    p = primeiro;
    primeiro = p->getProx();
    delete p;

    // atualiza o descritor
    n = n - 1;
    if(n == 0) ultimo = NULL;
  }
  else
    rotina_erro(2);
}
```

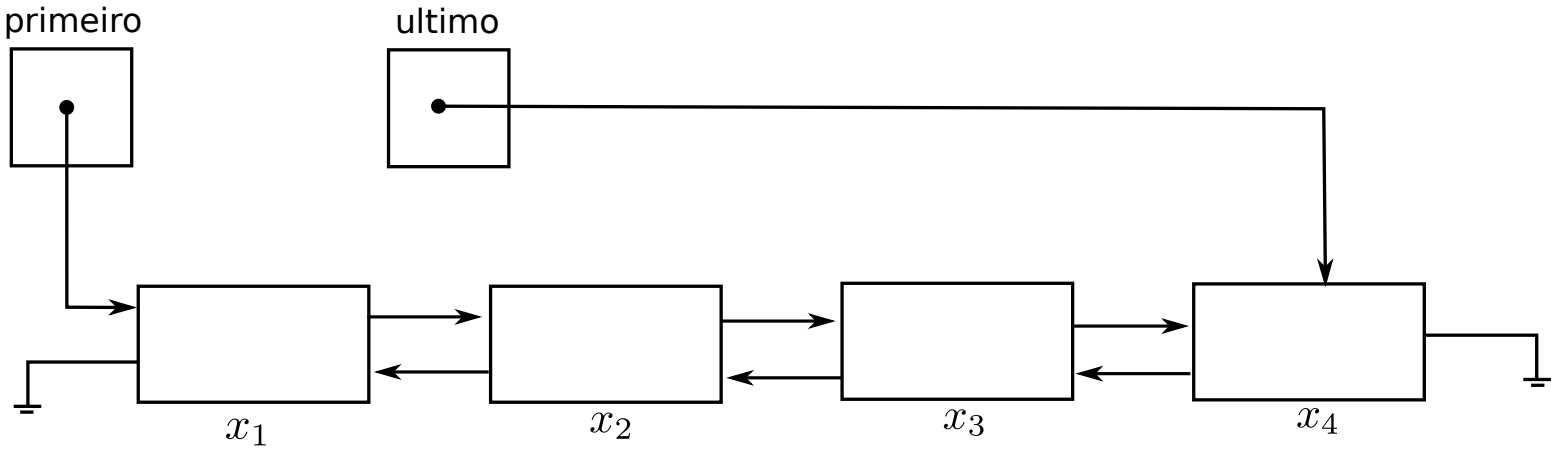
Lista com descritor

Exercícios

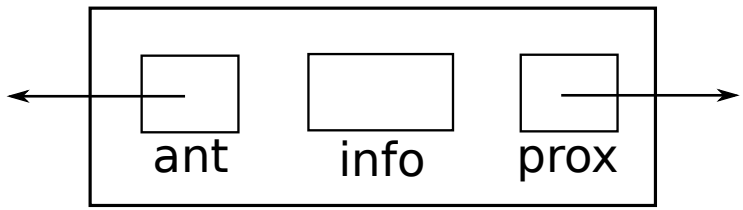
1. Implemente as seguintes operações no TAD `ListaEncad` com descritor:
   - `void insereFinal(int val);`
   - `void removeFinal();`
2. Qual é a complexidade computacional da operação `insereFinal(int val)` se implementada na (i) lista sem descritor e (ii) na lista com descritor e ponteiro para o último nó?
3. Implemente uma operação do TAD `ListaEncad` com descritor para inserir um valor depois de um nó cujo valor é dado por `x`. Se o valor `x` não for encontrado, a operação não deve inserir o valor e deve imprimir uma mensagem de erro.

Lista Duplamente Encadeada

- É uma lista onde cada nó possui dois ponteiros, um para o nó anterior e outro para o nó seguinte.



- Assim, é preciso definir outro TAD para representar o nó da lista.



Lista Duplamente Encadeada

```
class NoDuplo
{
 public:
   NoDuplo();
   ~NoDuplo();

   void setAnt(NoDuplo *p);
   void setProx(NoDuplo *p);
   void setInfo(int val);

   NoDuplo* getAnt();
   NoDuplo* getProx();
   int getInfo();

 private:
   NoDuplo *ant;  // ponteiro para anterior
   int info;      // informacao
   NoDuplo *prox; // ponteiro para proximo
};
```

Lista Duplamente Encadeada

```
NoDuplo::NoDuplo()   { }

NoDuplo::~NoDuplo() { }

void NoDuplo::setAnt(NoDuplo *p)   { ant = p;     }

void NoDuplo::setProx(NoDuplo *p) { prox = p;    }

void NoDuplo::setInfo(int val)     { info = val; }

NoDuplo* NoDuplo::getAnt()   { return ant;   }

NoDuplo* NoDuplo::getProx() { return prox; }

int NoDuplo::getInfo()       { return info; }
```

Lista Duplamente Encadeada

```
class ListaDupla
{
 public:
   ListaDupla();
   ~ListaDupla();
   bool busca(int val);
   void insereInicio(int val);
   void removeInicio();
   void insereFinal(int val);
   void removeFinal();

 private:
   NoDuplo *primeiro;
   int n;
   NoDuplo *ultimo;
};
```

Lista Duplamente Encadeada

- Algumas operações possuem implementação idênticas à do TAD `ListaEncad`.
- O que muda é a implementação dos métodos de inserção e remoção na lista, já que agora tem-se também um ponteiro para o nó anterior.

```
void ListaDupla::insereInicio(int val)
{
  NoDuplo *p = new NoDuplo();
  p->setInfo(val);
  p->setProx(primeiro);
  p->setAnt(NULL);

  if(n == 0)  ultimo = p;
  else        primeiro->setAnt(p);

  primeiro = p;
  n = n + 1;
}
```

Lista Duplamente Encadeada

```
void ListaDupla::removeInicio()
{
  NoDuplo *p;
  if(primeiro != NULL)
  {
    p = primeiro;
    primeiro = p->getProx();
    delete p;
    n = n - 1;

    if(n == 0)  ultimo = NULL;
    else        primeiro->setAnt(NULL);
  }
  else
    rotina_erro(3);
}
```

Lista Duplamente Encadeada

Exercícios

- Desenvolver uma função para inserir e remover nós no final da lista duplamente encadeada.
- Desenvolver uma função que imprime uma lista duplamente encadeada de trás para frente.

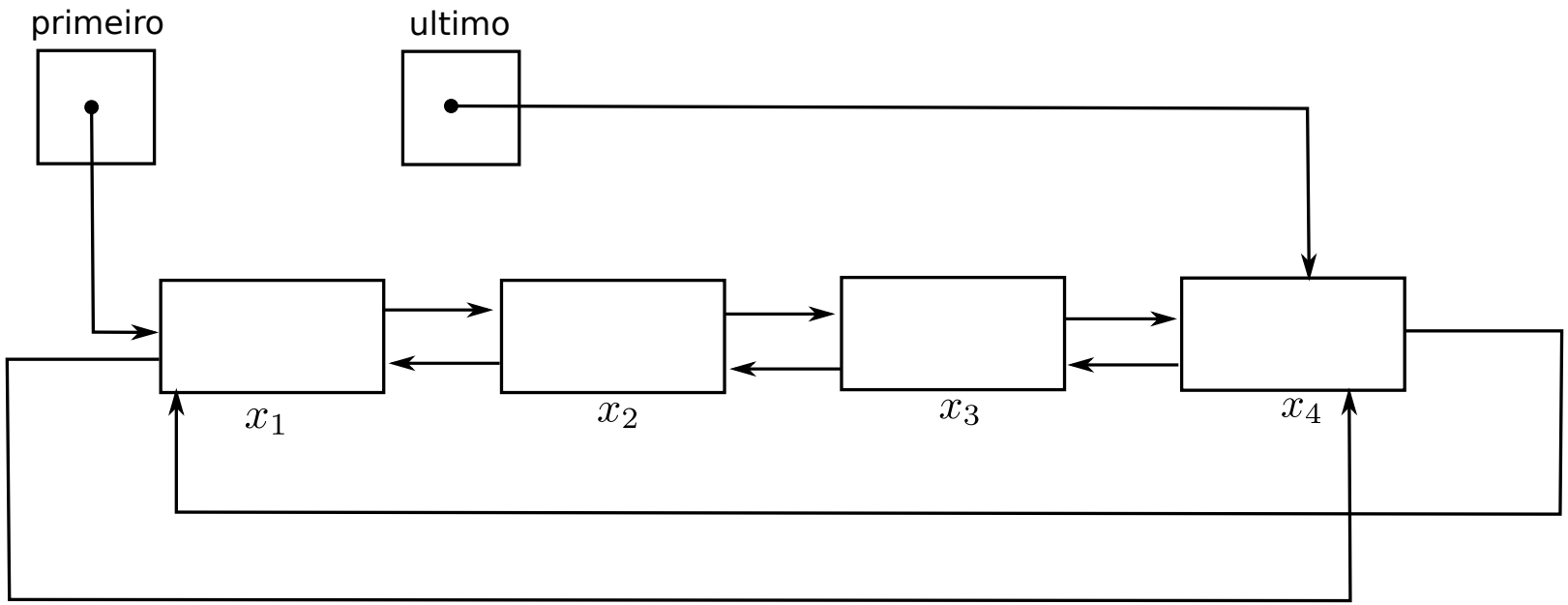
```
void ListaDupla::insereFinal(int val);

void ListaDupla::removeFinal();

void ListaDupla::imprimeReverso();
```

Lista Circular

- É uma lista duplamente encadeada com os dois nós extremos unidos.
- Que modificações devem ser feitas para transformar o TAD `ListaDupla` em um TAD `ListaCirc` que represente uma lista circular? Resposta: nenhuma.
- Basta fazer o ponteiro `ultimo` apontar para `primeiro`.



Lista Circular

```
void ListaCirc::insereInicio(int val)
{
  NoDuplo *p = new NoDuplo();
  p->setInfo(val);
  if(n == 0) {
    p->setProx(p);
    p->setAnt(p);
    ultimo = p;
  }
  else
  {
    p->setProx(primeiro);
    p->setAnt(ultimo);
    primeiro->setAnt(p);
    ultimo->setProx(p);
  }
  primeiro = p;
  n = n + 1;
}
```

Aplicações

- Lista Duplamente Encadeada
  - Cache do navegador, que permite usar o botão de "*Voltar*" (lista encadeada de URLs).
  - Funcionalidade de desfazer (*undo* – Ctrl+Z) no Photoshop, Word, Excel ou outros softwares.
  - Jogos: cartas na mão (baralho).
- Lista Circular
  - Jogos com vários jogadores onde um tem uma chance/jogada após o outro jogador e assim por diante.

Exercícios

1. Considerando o TAD de uma lista circular, desenvolver operações do seu MI para:
   - criar uma lista vazia (construtor);
   - inserir o primeiro nó;
   - remover o primeiro nó.
2. Desenvolver uma operação do TAD `ListCont` que leia diversos valores inteiros e positivos e insira os valores no final da lista. O último valor a ser lido é um FLAG igual a -1.
3. Idem ao anterior, montando uma lista simplesmente encadeada (sem descritor).
4. Considerando as listas das questões 2 e 3, montar uma operação para cada TAD para, dado um valor inteiro `val`, remover da lista todos os nós cujo valor seja igual a `val`.

Exercícios

5. Apresente a definição de um TAD `No` modificado que guarda como informação um vetor de 5 posições inteiras.
6. Considerando uma lista duplamente encadeada com descritor em que cada nó possui o formato do TAD `No` da questão 5, desenvolver uma operação da lista para remover o nó central caso o número de nós da lista seja ímpar.
7. Desenvolva uma operação para inserir um novo nó em uma lista simplesmente encadeada de inteiros de forma **ordenada**.