

**UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO**

Estrutura de Dados I / Projeto de Algoritmos I

**Prof. Denis Rosário
Email: denis@ufpa.br**



Agenda

Unidade 3: Listas Lineares

1. Listas lineares
2. Pilhas
3. Filas
4. Implementações

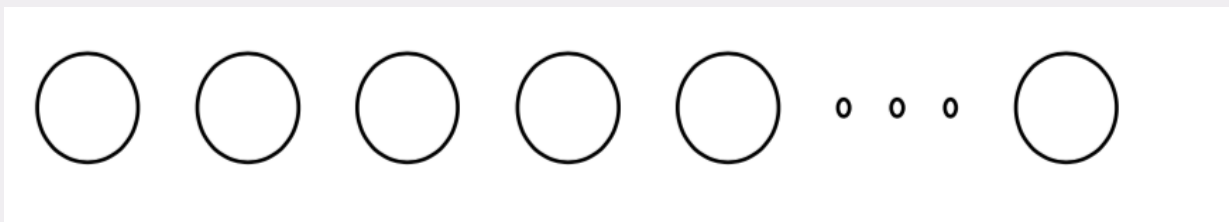


Lista Lineares

- Uma das formas mais simples de interligar os elementos de um conjunto
- Estrutura em que as operações inserir, retirar e localização são definidas
- São estruturas muito flexíveis: podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda
- Itens podem ser acessados, inseridos ou retirados de uma lista.

Lista Lineares

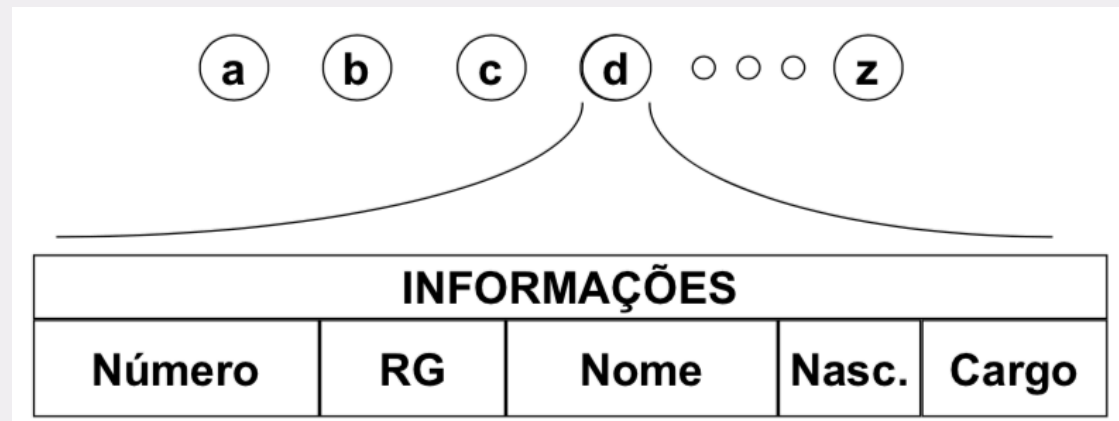
- Uma Lista Linear (LL) é uma sequência de nodos



- São as estruturas de mais simples manipulação

Lista Lineares

■ Estrutura dos nodos





Lista Lineares

- Duas listas podem ser concatenadas para formar uma lista única, ou uma pode ser partida em duas ou mais listas.
- Adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.
- São úteis em aplicações tais como manipulação simbólica, gerência de memória, simulação e compiladores.

Lista Lineares

- Sequência de zero ou mais itens x_1, x_2, \dots, x_n , na qual x_i é de um determinado tipo e n representa o tamanho da lista linear
- Sua principal propriedade estrutural envolve as posições relativas dos itens em uma dimensão

Lista Lineares

Assumindo $n \geq 1$, x_1 é o primeiro item da lista e x_n é o último item da lista.

x_i precede x_{i+1} para $i = 1, 2, \dots, n - 1$

x_i sucede x_{i-1} para $i = 2, 3, \dots, n$

o elemento x_i é dito estar na i -ésima posição da lista.



Lista Lineares

- Para criar uma TAD Lista, é necessário definir um conjunto de operações
- Um conjunto de operações necessário a uma maioria de aplicações é:
 1. Criar uma lista linear vazia.
 2. Inserir um novo item imediatamente após o i -ésimo item.
 3. Retirar o i -ésimo item.
 4. Localizar o i -ésimo item para examinar e/ou alterar o conteúdo de seus componentes.
 5. Combinar duas ou mais listas lineares em uma lista única.




Lista Lineares

- Um conjunto de operações necessário a uma maioria de aplicações é: (cont)
 6. Partir uma lista linear em duas ou mais listas.
 7. Fazer uma cópia da lista linear.
 8. Ordenar os itens da lista em ordem ascendente ou descendente, de acordo com alguns de seus componentes.
 9. Pesquisar a ocorrência de um item com um valor particular em algum componente.



Lista Lineares

- Várias estruturas de dados podem ser usadas para representar listas lineares, cada uma com vantagens e desvantagens particulares.
- Vamos estudar duas maneiras distintas
 - Usando alocação sequencial e estática (com vetores).
 - Usando alocação não sequencial e dinâmica (com ponteiros): Estruturas Encadeadas.



Implementação de Listas por meio de Arranjos

- Armazena itens em posições contíguas de memória.
- A lista pode ser percorrida em qualquer direção.
- A inserção de um novo item pode ser realizada após o último item com custo constante.
- A inserção de um novo item no meio da lista requer um deslocamento de todos os itens localizados após o ponto de inserção.
- Retirar um item do início da lista requer um deslocamento de itens para preencher o espaço deixado vazio.

Implementação de Listas por meio de Arranjos

Itens	
Primeiro = 1	x_1
2	x_2
	\vdots
Último - 1	x_n
	\vdots
MaxTam	

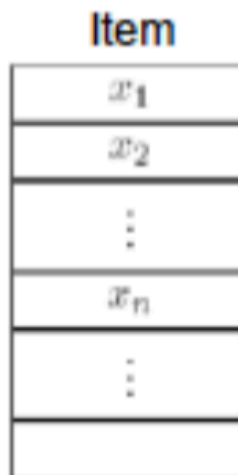


Implementação de Listas por meio de Arranjos

- Os itens são armazenados em um vetor de tamanho suficiente para armazenar a lista.
- O campo Último contém a posição após o último elemento da lista.
- O i -ésimo item da lista está armazenado na i -ésima posição do vetor, $0 \leq i \leq \text{Último}$.
- A constante MaxTam define o tamanho máximo permitido para a lista.

Estrutura da Lista Usando Arranjo

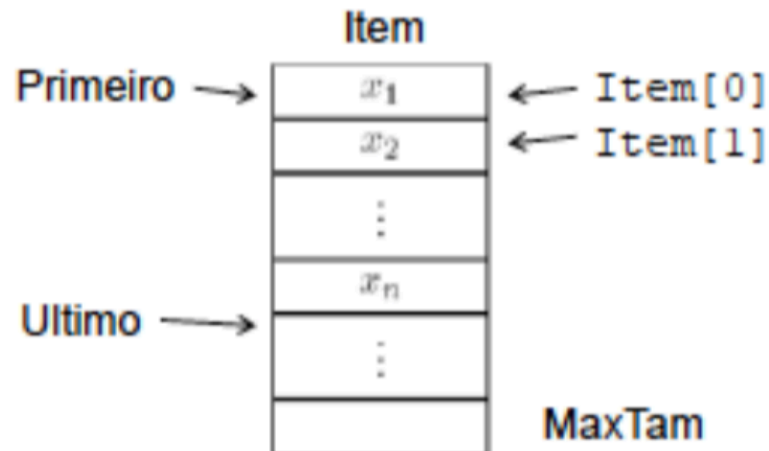
- Os itens armazenados em um array.
- MaxTam define o tamanho máximo permitido para a lista.
- O campo Último aponta para a posição seguinte a do último elemento da lista. (primeira posição vazia)
- O i -ésimo item da lista está armazenado na i -ésima-1 posição do array, $0 \leq i < \text{Último}$. (Item[i])



```
1  typedef int Posicao;  
2  #define InicioVetor 0  
3  #define MaxTam 1000  
4  
5  typedef struct tipoitem {  
6      int valor;  
7      /* outros componentes */  
8  }TipoItem;  
9  
10 typedef struct tipolista{  
11     TipoItem Item[MaxTam];  
12     Posicao Primeiro, Ultimo;  
13 }TipoLista;
```

Estrutura da Lista Usando Arranjo

- Os itens armazenados em um array.
- MaxTam define o tamanho máximo permitido para a lista.
- O campo Último aponta para a posição seguinte a do último elemento da lista. (primeira posição vazia)
- O *i*-ésimo item da lista está armazenado na *i*-ésima-1 posição do array, $0 \leq i < \text{Último}$. (Item[i])




```
1  typedef int Posicao;  
2  #define InicioVetor 0  
3  #define MaxTam 1000  
4  
5  typedef struct tipoitem {  
6      int valor;  
7      /* outros componentes */  
8  }TipoItem;  
9  
10 typedef struct tipolista{  
11     TipoItem Item[MaxTam];  
12     Posicao Primeiro, Ultimo;  
13 }TipoLista;
```


Lista Lineares

■ Exemplo de conjunto de operações:

1. FLVazia(Lista). Faz a lista ficar vazia.
2. Insere(x, Lista). Insere x após o último item da lista.
3. Retira(p, Lista, x). Retorna o item x que está na posição p da lista, retirando-o da lista e deslocando os itens a partir da posição p+1 para as posições anteriores.
4. Vazia(Lista). Esta função retorna true se lista vazia; senão retorna false.
5. Imprime(Lista). Imprime os itens da lista na ordem de ocorrência.



Operações sobre Lista Usando Arranjo

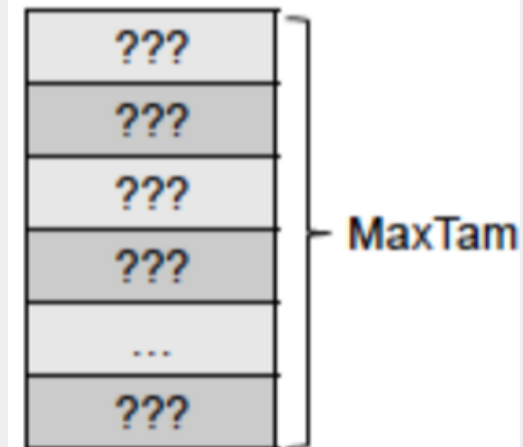
`/* Faz a lista ficar vazia */`

FLVazia(Lista)

- Input: L (Lista)
- Output: L'
- Pré-condição: L é definida
- Pós-condição: L' é definida e vazia

Operações sobre Lista Usando Arranjo

```
/* Faz a lista ficar vazia */  
void FLVazia (TipoLista* Lista) {  
    Lista->Primeiro = InicioVetor;  
    Lista->Ultimo = Lista->Primeiro;  
}
```



Operações sobre Lista Usando Arranjo

```
/* Faz a lista ficar vazia */  
void FLVazia (TipoLista* Lista) {  
    Lista->Primeiro = InicioVetor;  
    Lista->Ultimo = Lista->Primeiro;  
}
```





Operações sobre Lista Usando Arranjo

`/*Verifica se a lista está vazia*/`

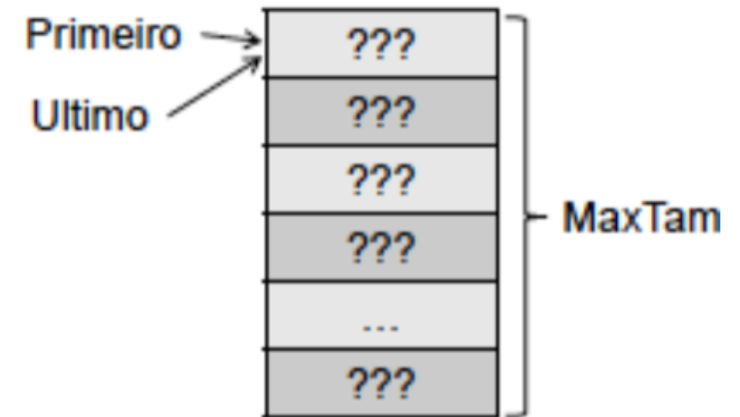
Vazia(Lista)

- Input: L (Lista)
- Output: B (true se lista vazia; senão retorna false)
- Pré-condição: L é definida
- Pós-condição: L não é modificada

Operações sobre Lista Usando Arranjo

```
/* Faz a lista ficar vazia */  
void FLVazia (TipoLista* Lista) {  
    Lista->Primeiro = InicioVetor;  
    Lista->Ultimo = Lista->Primeiro;  
}
```

```
/*Verifica se a lista está vazia*/  
int Vazia (TipoLista* Lista){  
    return (Lista->Primeiro == Lista->Ultimo);  
}
```





Operações sobre Lista Usando Arranjo

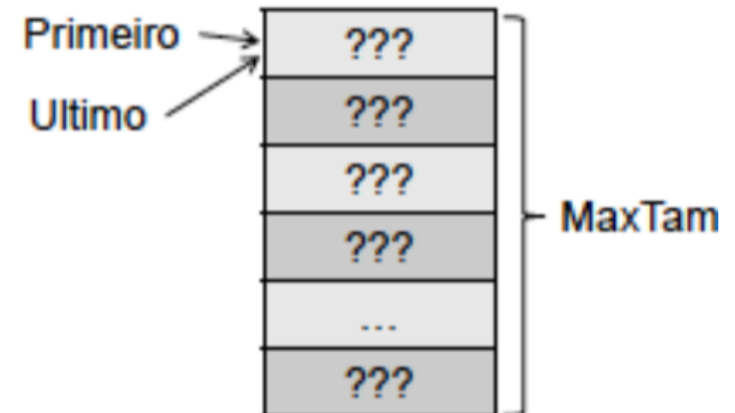
```
/* Insere x após o último elemento da lista
*/
```

Insere(x, Lista). Insere x após o último

- Input: x (Item da Lista) e L (Lista)
- Output: L'
- Pré-condição: L é definida e x é um Item válido da lista
- Pós-condição: L' é definida e vazia e o elemento item de L' é igual a x

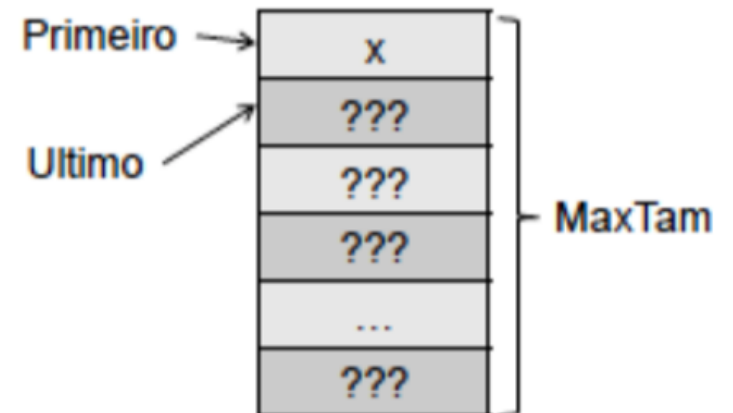
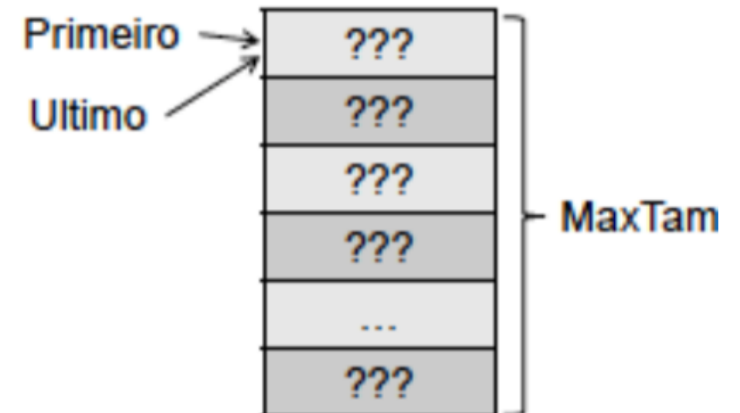
Operações sobre Lista Usando Arranjo


```
/* Insere x após o último elemento da lista */  
void Insere (TipoItem* x, TipoLista *Lista) {  
    if (Lista ->Ultimo >= MaxTam)  
        cout << "Lista está cheia" << endl;  
    else{  
        Lista ->Item[Lista->Ultimo] = *x;  
        Lista->Ultimo++;  
    }  
}
```



Operações sobre Lista Usando Arranjo

```
/* Insere x após o último elemento da lista */  
void Insere (TipoItem* x, TipoLista *Lista) {  
    if (Lista ->Ultimo >= MaxTam)  
        cout << "Lista está cheia" << endl;  
    else{  
        Lista ->Item[Lista->Ultimo] = *x;  
        Lista->Ultimo++;  
    }  
}
```





Operações sobre Lista Usando Arranjo

`/*Retorna o item x que está na posição p da lista, retirando-o da lista e deslocando os itens a partir da posição p+1 para as posições anteriores */`

`Retira(p, Lista, x)`

- Input: p (posição válida da lista) e L (Lista)
- Output: x (item da lista da posição p)
- Pré-condição: L é definida e p é uma posição válida da lista
- Pós-condição: L' é a lista L sem o item x, com todos os itens deslocados de uma posição

Operações sobre Lista Usando Arranjo

/*Retorna o item x que está na posição p da lista, retirando-o da lista e deslocando os itens a partir da posição p+1 para as posições anteriores */

```
TipoItem* Retira (Posicao p, TipoLista* Lista) {  
    int Aux;  
    TipoItem* item;  
    item = (TipoItem*) malloc(sizeof(TipoItem));  
    if (Vazia(Lista) || p >= Lista->Ultimo){  
        cout << "A posição não existe!" << endl;  
        return NULL;  
    }  
    *item = Lista->Item[p];  
    Lista->Ultimo--;  
    for (Aux = p; Aux < Lista->Ultimo; Aux++)  
        Lista->Item[Aux] = Lista->Item[Aux+1];  
    return item;  
}
```

Operações sobre Lista Usando Arranjo

```
/*Retorna o item x que está na posição p da lista,
retirando-o da lista e deslocando os itens a partir
da posição p+1 para as posições anteriores */
```

```
TipoItem* Retira (Posicao p, TipoLista* Lista) {
```

```
    int Aux;
```

```
    TipoItem* item;
```

```
    item = (TipoItem*) malloc(sizeof(TipoItem));
```

```
    if (Vazia(Lista) || p >= Lista->Ultimo){
```

```
        cout << "A posição não existe!" << endl;
```

```
        return NULL;
```

```
    }
```

```
    *item = Lista->Item[p];
```

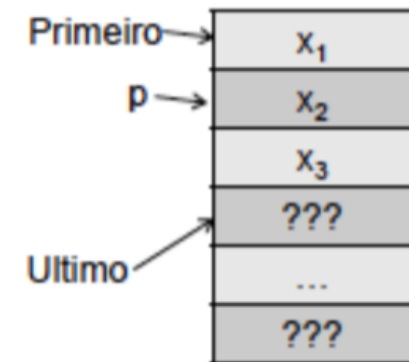
```
    Lista->Ultimo--;
```

```
    for (Aux = p; Aux < Lista->Ultimo; Aux++)
```

```
        Lista->Item[Aux] = Lista->Item[Aux+1];
```

```
    return item;
```

```
}
```



Operações sobre Lista Usando Arranjo

/*Retorna o item x que está na posição p da lista, retirando-o da lista e deslocando os itens a partir da posição p+1 para as posições anteriores */

```
TipoItem* Retira (Posicao p, TipoLista* Lista) {
```

```
    int Aux;
```

```
    TipoItem* item;
```

```
    item = (TipoItem*) malloc(sizeof(TipoItem));
```

```
    if (Vazia(Lista) || p >= Lista->Ultimo){
```

```
        cout << "A posição não existe!" << endl;
```

```
        return NULL;
```

```
    }
```

```
    *item = Lista->Item[p];
```

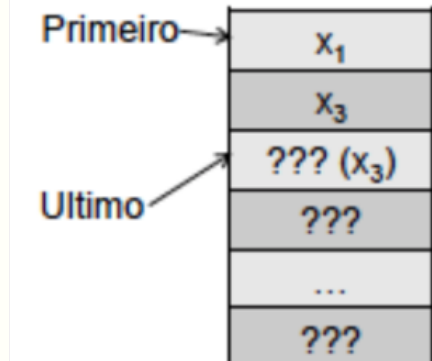
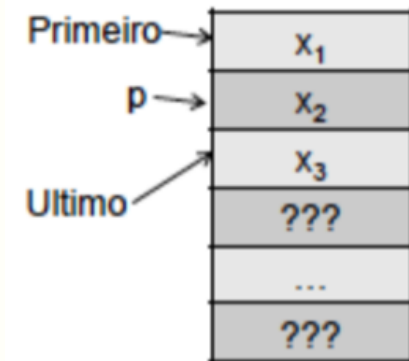
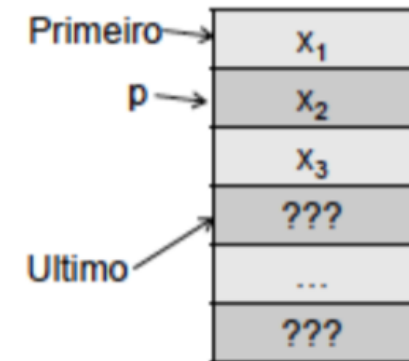
```
    Lista->Ultimo--;
```


```
    for (Aux = p; Aux < Lista->Ultimo; Aux++)
```

```
        Lista->Item[Aux] = Lista->Item[Aux+1];
```

```
    return item;
```

```
}
```





Operações sobre Lista Usando Arranjo

`/*Imprime os itens da lista na ordem de ocorrência */`

Imprime(Lista)

- Input: L (Lista)
- Output: nenhum
- Pré-condição: L é definida e não está vazia
- Pós-condição: L não é modificada e seus elementos são impressos

Operações sobre Lista Usando Arranjo

```
/*Imprime os itens da lista na ordem de ocorrência
*/
void Imprime (TipoLista* Lista){
    cout << "p - key" << endl;
    for (int Aux = Lista->Primeiro; Aux < Lista->Ultimo;
        Aux++){
        cout << Aux << " - " << Lista->Item[Aux].valor <<
        endl;
    }
}
```

Arquivo lista.h

```
1  typedef int Posicao;
2  #define InicioVetor 0
3  #define MaxTam 1000
4
5  typedef struct tipoitem {
6      int valor;
7      /* outros componentes */
8  }TipoItem;
9
10 typedef struct tipolista{
11     TipoItem Item[MaxTam];
12     Posicao Primeiro, Ultimo;
13 }TipoLista;
14
15 TipoLista* InicializaLista();
16 void FLVazia (TipoLista* Lista);
17 int Vazia (TipoLista* Lista);
18 int Tamanho_lista(TipoLista* Lista);
19 void Insere (TipoItem* x, TipoLista* Lista);
20 TipoItem* Busca(int chave, TipoLista* lista);
21 TipoItem* Retira (Posicao p, TipoLista* Lista);
22 void Imprime (TipoLista* Lista);
23 TipoItem* InicializaTipoItem();
24 void ModificaValorItem (TipoItem* x, int valor);
25 void ImprimeTipoItem(TipoItem* x);
```




Arquivo lista.cpp

```
#include <iostream.h>
#include "lista.h"

TipoLista* InicializaLista()
{...}

void FLVazia (TipoLista* Lista)
{...}

int Vazia (TipoLista* Lista)
{...}

void Insere (TipoItem* x, TipoLista *Lista)
{...}

TipoItem* Busca(int chave, TipoLista* lista)
{...}

int Tamanho_lista(TipoLista* Lista)
{...}

TipoItem* Retira (Posicao p, TipoLista* Lista)
{...}

void Imprime (TipoLista* Lista)
{...}
```

Arquivo main.cpp

```
1  #include <iostream>
2  #include "lista.h"
3
4  using namespace std;
5
6  int main(void){
7      TipoLista* list;
8      list = InicializaLista();
9      cout << "lista vazia?: " << Vazia(list) << endl;
10     TipoItem* item;
11     item->valor = 10;
12     cout << "valor a ser inserido: " << item->valor << endl;
13     Insere(item, list);
14     item->valor = 20;
15     cout << "valor a ser inserido: " << item->valor << endl;
16     Insere(item, list);
17     cout << "lista vazia?: " << Vazia(list) << endl;
18     cout << "Tamanho da lista: " << Tamanho_lista(list) << endl;
19
20     cout << "consulta pelo valor: 10" << endl;
21     Busca(10, list);
22     cout << "Imprimir a Lista" << endl;
23     Imprime(list);
24     cout << "Remover na posicao: 0" << endl;
25     Retira(0, list);
26     cout << "tamanho da lista: " << Tamanho_lista(list) << endl;
27     FLVazia(list);
28     cout << "Faz a lista ficar vazia" << endl;
29     cout << "tamanho da lista: " << Tamanho_lista(list) << endl;
30     cout << "lista vazia?: " << Vazia(list) << endl;
31 }
```

```
lista vazia?: 1
valor a ser inserido: 10
valor a ser inserido: 20
lista vazia?: 0
Tamanho da lista: 2
consulta pelo valor: 10
Item existe
Imprimir a Lista
p - key
0 - 10
1 - 20
Remover na posicao: 0
tamanho da lista: 1
Faz a lista ficar vazia
tamanho da lista: 0
lista vazia?: 1
```



Lista usando Arranjo – Vantagens e Desvantagens

■ Vantagem:

- economia de memória (os apontadores são implícitos nesta estrutura).
- Acesso a um item qualquer é constante ($O(1)$)

■ Desvantagens:

- custo para inserir ou retirar itens da lista, que pode causar um deslocamento de todos os itens, no pior caso;
- em aplicações em que não existe previsão sobre o crescimento da lista, a utilização de arranjos em linguagens como o Pascal pode ser problemática porque nesse caso o tamanho máximo da lista tem de ser definido em tempo de compilação.

Exercício

- Assumindo que os elementos de uma lista L são inteiros positivos, escreva um programa que informe os elementos que ocorrem mais e menos em L (forneça os elementos e o número de ocorrências correspondente)
- Escreva um programa que gera uma lista L2, a partir de uma lista L1 dada, em que cada registro de L2 contém dois campos de informação
 - Ou seja, a variável *elem* contém um elemento de L1, e a variável *count* contém o número de ocorrências deste elemento em L1