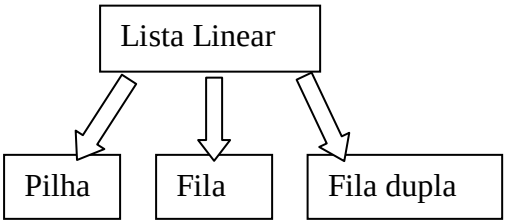
FeMASS
- Revisão/teoria sobre Lista Lineares, Pilhas e Filas
- Breve resumo sobre alocação estática x dinâmica

**1) Listas Lineares**

- Casos especiais de listas lineares (visão geral) que serão estudadas:



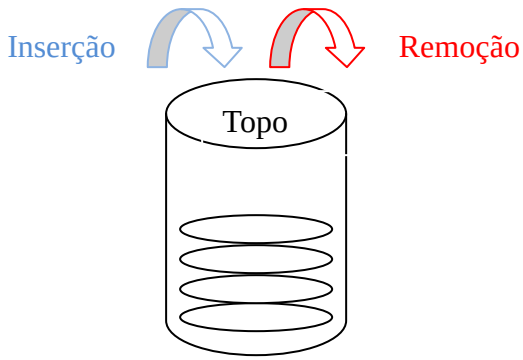

**Uma lista linear (L)** é uma coleção de elementos cuja propriedade estrutural baseia-se na posição dos elementos que são dispostos linearmente:

$L = [a_1, a_2, a_3, \ldots, a_n]$, $n \geq 0$.

Logo, $a_1$ é o primeiro elemento de L;
$a_n$ – É o último elemento de L;
$a_k$, $1 < k < n$, é procedido por $a_{k-1}$ e seguido por $a_{k+1}$ em L.
Se n=0, então dizemos que a lista L é vazia.

**Pilha:** Trata-se de um tipo de lista linear onde todas as inserções, remoções e acessos são realizados em um único extremo, também conhecidas como LIFO (Last-in/First-out – "O último a entrara é o primeiro que sai"). É possível inserir elementos/objetos em uma pilha a qualquer momento, mas somente o objeto inserido mais recentemente (no topo da Pilha) pode ser removido a qualquer momento.

**Pilha = "Stack":**



- Métodos fundamentais para “Stack” (Pilha):

- push(o): Insere o objeto (o) no topo da pilha.
  Entrada: objeto. Saída: nenhuma.
- pop( ): Retira o objeto no topo da pilha e o retorna. Ocorre um erro no caso da pilha estiver vazia.
  Entrada: nenhuma. Saída: objeto.

- Métodos auxiliares:

- size( ): Retorna o número de objetos na pilha.
  Entrada: nenhuma. Saída: número inteiro: 0 – n.
- isEmpty( ): Retorna um booleano indicando se a pilha está vazia. (“True” – caso positivo).
  Entrada: nenhuma. Saída: booleano.
- top( ): Retorna o objeto no topo da pilha, sem removê-lo. Ocorre erro no caso da pilha estiver vazia.
  Entrada: nenhuma. Saída: objeto.

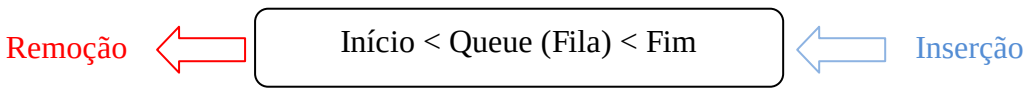
**Observação: Outros métodos auxiliares podem ser criados, para fins de inicialização, verificação de limites, etc.**

**Exemplo:** Analise a pilha conforme a sequência de operações:

| Operação | Saída | S:[ ] |
|---|---|---|
| push(5) | - | [ 5 ] |
| push(3) | - | [ 5, 3 ] |
| pop( ) | 3 | [ 5 ] |
| push(7) | - | [ 5, 7 ] |
| pop( ) | 7 | [ 5 ] |
| top() | 5 | [ 5 ] |
| pop( ) | 5 | [ ] |

| | | |
|---|---|---|
| pop( ) | “ERROR” | [ ] |
| isEmpty() | “TRUE” | [ ] |
| push(9) | - | [ 9 ] |
| push(7) | - | [ 9, 7 ] |
| push(3) | - | [9, 7, 3] |
| push(5) | - | [9, 7, 3, 5] |
| size() | 4 | [9, 7, 3, 5] |
| pop() | 5 | [ 9, 7, 3] |
| push(8) | - | [ 9, 7, 3, 8] |
| pop() | 8 | [ 9, 7, 3] |
| pop() | 3 | [ 9, 7 ] |

**Fila:** É um tipo especial de Lista Linear em que a inserções são feitas em um extremo e as remoções no outro extremo. O princípio da Fila é de que “o primeiro que entra é o primeiro que sai” (FIFO – First-in/First-out).




**Observação: “Queue” = Fila. Trata-se de uma Estrutura de Dado fundamental, muito usada quando se deseja garantir ordem.**

- Métodos fundamentais:

- enqueue(o): insere o objeto (o) no fim da fila.
  Entrada: objeto. Saída: nenhuma.
- dequeue( ): retira e retorna o objeto no início da fila. Ocorre erro caso esteja vazia.
  Entrada: nenhuma. Saída: objeto.

- Métodos auxiliares:

- size( ): Retorna o número de objetos na fila.
  Entrada: nenhuma. Saída: número inteiro: 0 – n.
- isEmpty( ): Retorna um booleano indicando se a fila está vazia. (“True” – caso positivo).
  Entrada: nenhuma. Saída: booleano.
- front( ): Retorna o objeto no início da fila, sem removê-lo. Ocorre erro no caso da fila estiver vazia.
  Entrada: nenhuma. Saída: objeto.

**Observação: Métodos adicionais poderão ser previstos, do mesmo modo como tratado sobre Pilha.**

**Exemplo:** analise as operações:

| Operações | Saída | Início < Q < Fim |
|---|---|---|
| Enqueue(5) | - | 5 |
| Enqueue(3) | - | 5, 3 |
| Dequeue( ) | 5 | 3 |
| Enqueue(7) | - | 3, 7 |
| Dequeue( ) | 3 | 7 |
| Front( ) | 7 | 7 |
| Dequeue( ) | 7 | - |
| Dequeue( ) | “Error” | - |
| isEmpty( ) | “True” | - |
| Enqueue(9) | - | 9 |
| Enqueue(7) | - | 9, 7 |
| Size( ) | 2 | 9, 7 |
| Enqueue(3) | - | 9, 7, 3 |
| Enqueue(5) | - | 9, 7, 3, 5 |
| Dequeue( ) | 9 | 7, 3, 5 |

**Atenção:**

**Os conceitos de PILHA e FILA poderão ser estendidos por meio de técnicas de implementação e recursos de Listas Lineares. Por exemplo, estudaremos a aplicação de Listas de extremidade dupla, circular, duplamente encadeada, entre outros recursos.**

## 2) Alocação de memória

Ao desenvolver uma implementação para listas lineares, o primeiro problema que surge é: como podemos armazenar os elementos da lista, dentro do computador? É sabido que antes de executar um programa, o computador precisa carregar seu código executável para a memória principal (RAM). Neste momento, uma parte da memória total disponível no sistema é reservada para uso do programa e o restante fica livre, pois raramente um programa irá ocupar toda a memória instalada.

Da área de memória reservada ao programa, uma parte é usada para as instruções a serem executadas e outra ao armazenamento de dados durante a execução. O compilador é quem determina quanto de memória será alocado para instruções. No entanto, o programador é o responsável por alocar área de armazenamento de dados, neste caso, sob o ponto de vista de categorias no quadro abaixo.

| | **Sequencial** | **Encadeada** |
|---|---|---|
| **Estática** | **Estática Sequencial** | **Estática Encadeada** |
| **Dinâmica** | **Dinâmica Sequencial** | **Dinâmica Encadeada** |

### 2.1) Alocação Estática versus Dinâmica

Uma variável em código de programa nada mais é que uma área de memória, dizemos que sua alocação é estática se sua existência já era prevista no código do programa (pré-definida e imutável); de outra forma, caso o programa seja capaz de criar novas variáveis enquanto executa (áreas de memória passarão a existir) dizemos que sua alocação é dinâmica.

**Exemplo:** Um programa que utiliza da declaração estática de uma variável ptr como ponteiro para guardar endereço de memória livre alocada durante a execução (conhecida como variável anônima, neste caso) via comando malloc, e atribuição de valor inteiro [12345].

### 2.2) Alocação Sequencial e Encadeada

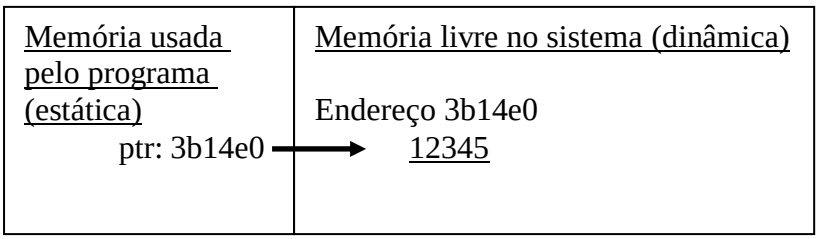
```c
#include <iostream>
#include <stdlib.h>
int main(){
   //declaração de variável ponteiro para inteiro
   int *ptr = (int*) malloc(sizeof(int));
   //atribuindo o endereço da variável valor ao ponteiro
   *ptr = 12345;
   printf("Utilizando ponteiros\n\n");
   printf ("Conteudo da variavel ponteiro ptr (endereco): %x \n", ptr);
   printf ("Acessando a variavel apontada pelo ponteiro ptr (dado): %d \n", *ptr);
   system("pause");
}
```

**Código C/C++**

```
Utilizando ponteiros

Conteudo da variavel ponteiro ptr (endereco): 3b14e0
Acessando a variavel apontada pelo ponteiro ptr (dado): 12345
Pressione qualquer tecla para continuar. . .
```

**Saída da execução em terminal**




**Organização de memória para alocação estática e dinâmica exemplificado**