

Estruturas de Dados

Prof. Mateus Mendelson

mateus.silva@projecao.br

Pilhas

1. Pilhas

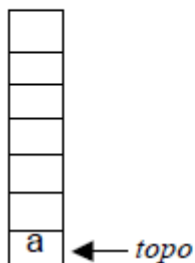
- A ideia fundamental da pilha é que todo o acesso a seus elementos é feito através do topo.
- Pensem em uma pilha de pratos.
- Se quisermos adicionar um prato na pilha, o colocamos no topo. Para pegar um prato da pilha, retiramos o do topo. Assim, temos de retirar o prato do topo para ter acesso ao próximo prato.
- Em uma pilha, seus elementos só podem ser retirados na ordem inversa à ordem em que foram introduzidos: o primeiro que sai é o último que entrou.
- A sigla LIFO – *last in, first out* – é usada para descrever essa estratégia.

1. Pilhas

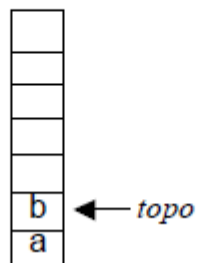
- Há duas operações básicas:

- ✓ *push* (empilhar)
- ✓ *pop* (desempilhar)

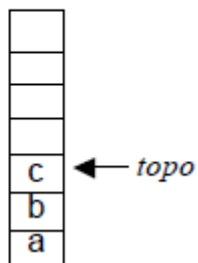
push (a)



push (b)

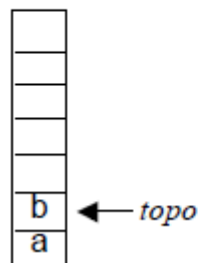


push (c)

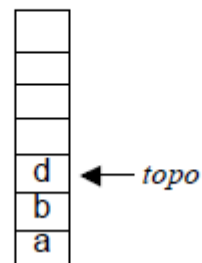


pop ()

retorna-se C

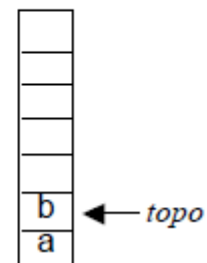


push (d)



pop ()

retorna-se d



1. Pilhas

- O exemplo de utilização de pilha mais próximo é a própria pilha de execução da linguagem C. As variáveis locais das funções só tem acesso às variáveis que estão no topo.

1. Pilhas

```
#include <stdio.h>
int fat (int n);
int main(){
    int n = 5, r;
    r = fat(n);

    printf("Fatorial de %d: %d\n", n, r);

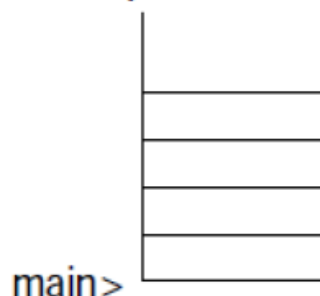
    return 0;
}
int fat (int n){
    int f = 1;

    while(n != 0){
        f *= n;
        n--
    }

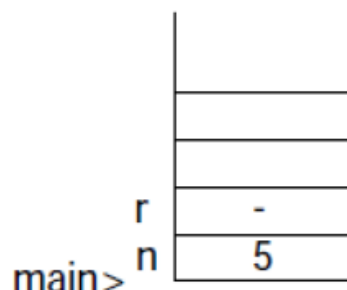
    return f;
}
```

1. Pilhas

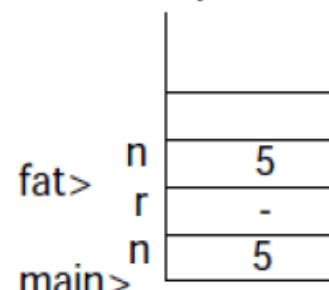
1 – Início do programa:
pilha vazia



2 – Declaração de
variáveis: n,r

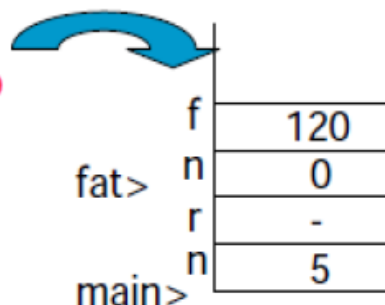


3 – Chamada da função:
empilha variáveis

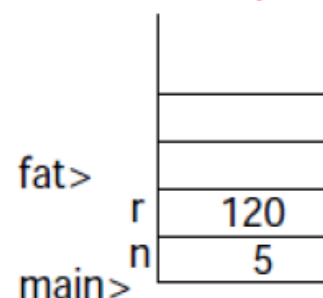


4 – Final do laço

Acesso às variáveis que
estão na função do topo



5 – Retorno da função:
desempilha



2. Pilhas como Vetores

- Muitas vezes, o tamanho máximo de elementos possíveis é conhecido, ou seja, a estrutura da pilha tem um limite conhecido.
- Nesses casos, a implementação da pilha pode ser feita por um vetor, o que é muito simples. Devemos ter um vetor para armazenar os elementos da pilha.
- Os elementos inseridos ocupam as primeiras posições do vetor. Dessa forma, se temos n elementos armazenados na pilha, o elemento $n-1$ representa o topo.
- Vantagem: implementação simples.
- Desvantagens: deve-se saber o tamanho máximo de antemão e há desperdício de memória.

2. Pilhas como Vetores

```
#define N 50
```

```
typedef struct pilha {  
    int topo;  
    float vet[N];  
} Pilha;
```

3. Pilhas como Listas

- Quando o número máximo de elementos que serão armazenados na pilha não é conhecido, devemos implementar a pilha com uma estrutura de dados dinâmica: uma lista encadeada.
- Os elementos são armazenados na lista e a pilha pode ser representada simplesmente por um ponteiro para o primeiro nó da lista.

```
typedef struct lista{  
    float info;  
    struct lista* prox;  
} Lista;
```

```
typedef struct pilha{  
    Lista* topo;  
} Pilha;
```

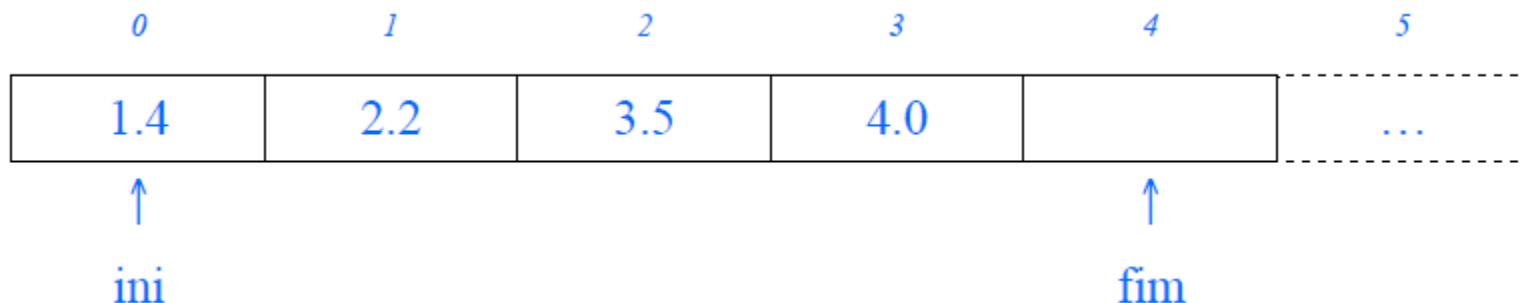
Filas

4. Filas

- Os acessos a elementos também seguem uma regra.
- O que a diferencia da pilha é a ordem de saída dos elementos: enquanto na pilha "o último que entra é o primeiro que sai", na fila "o primeiro que entra é o primeiro que sai".
- *FIFO – first in, first out.*
- Quem primeiro entra em uma fila é o primeiro a ser atendido.
- Sua ideia fundamental é que só podemos inserir um novo elemento no final da fila e só podemos retirar o elemento do início.

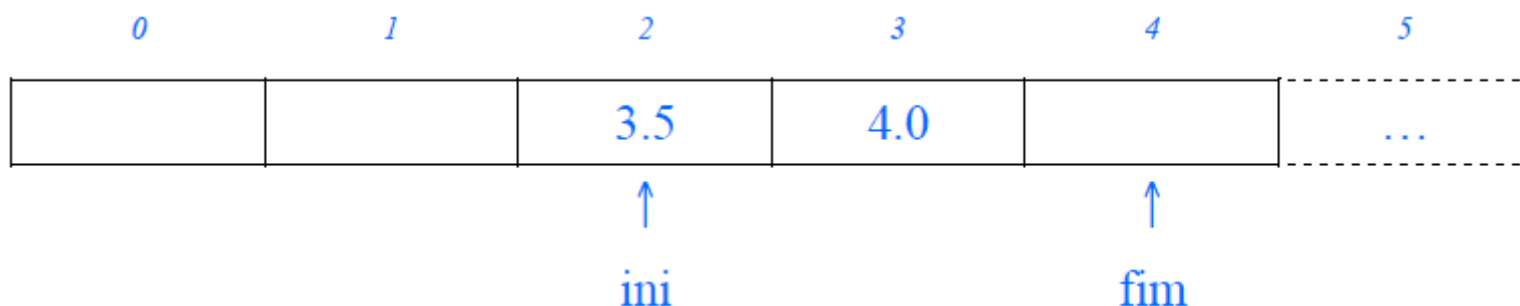
5. Filas como Vetores

- Se o número máximo de elementos N na fila for conhecido, podemos utilizar vetores para implementar esse TAD.
- Suponha a inserção dos valores 1.4, 2.2, 3.5 e 4.0.



5. Filas como Vetores

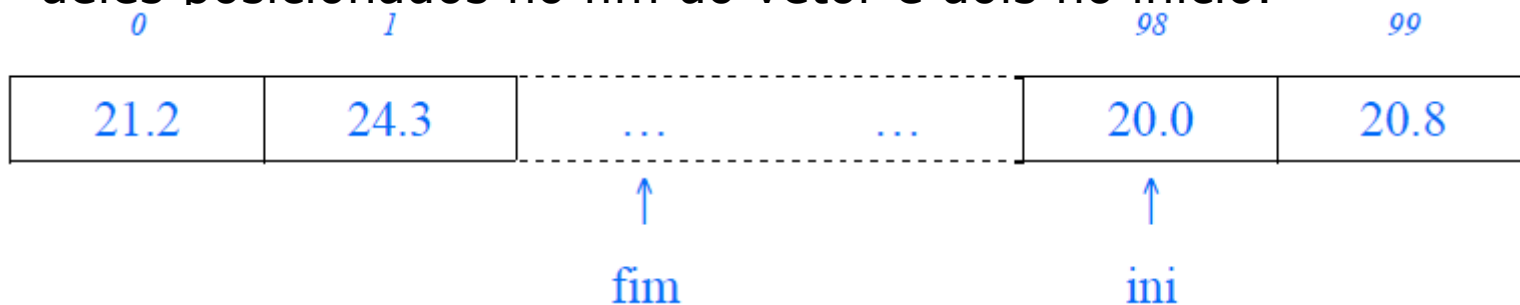
- Agora vamos retirar dois elementos da fila.



- Com essa estratégia, é fácil observar que, em um dado instante, a parte ocupada do vetor pode chegar à última posição, esgotando o vetor.

5. Filas como Vetores

- Para reaproveitar as primeiras posições livres do vetor sem implementar uma re-arrumação trabalhosa dos elementos, podemos incrementar as posições do vetor de forma circular.
- Se o último elemento da fila ocupa a última posição do vetor, inserimos os novos elementos a partir do início do vetor.
- Dessa forma, em um dado momento, poderíamos ter quatro elementos 20.0, 20.8, 21.2 e 24.3, sendo dois deles posicionados no fim do vetor e dois no início.



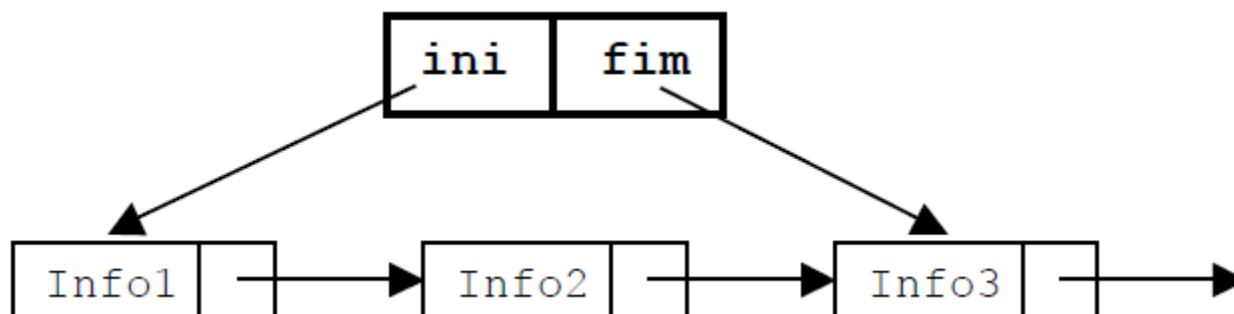
5. Filas como Vetores

- O que a função abaixo faz?

```
#define N 100  
.  
.  
.  
int incr(int i) {  
    return (i+1)%N  
}
```


6. Filas como Listas

- Há situações em que o tamanho máximo da fila é desconhecido. Nesses casos, é necessário utilizar listas encadeadas para implementar a fila.
- São armazenados dois ponteiros: um apontando para o início da fila e outro apontando para o final da fila.



6. Filas como Listas

```
typedef struct lista {  
    float info;  
    struct lista* prox;  
} Lista;
```

```
typedef struct fila {  
    Lista* ini;  
    Lista* fim;  
} Fila;
```

6. Filas como Listas

- Assim como existem diferentes tipos de listas encadeadas, existem diferentes tipos de filas.
- Vale a pena mencionar as filas duplas, nas quais é possível inserir novos elementos nas duas extremidades: no início e no fim. Consequentemente, pode-se retirar elementos das duas extremidades.