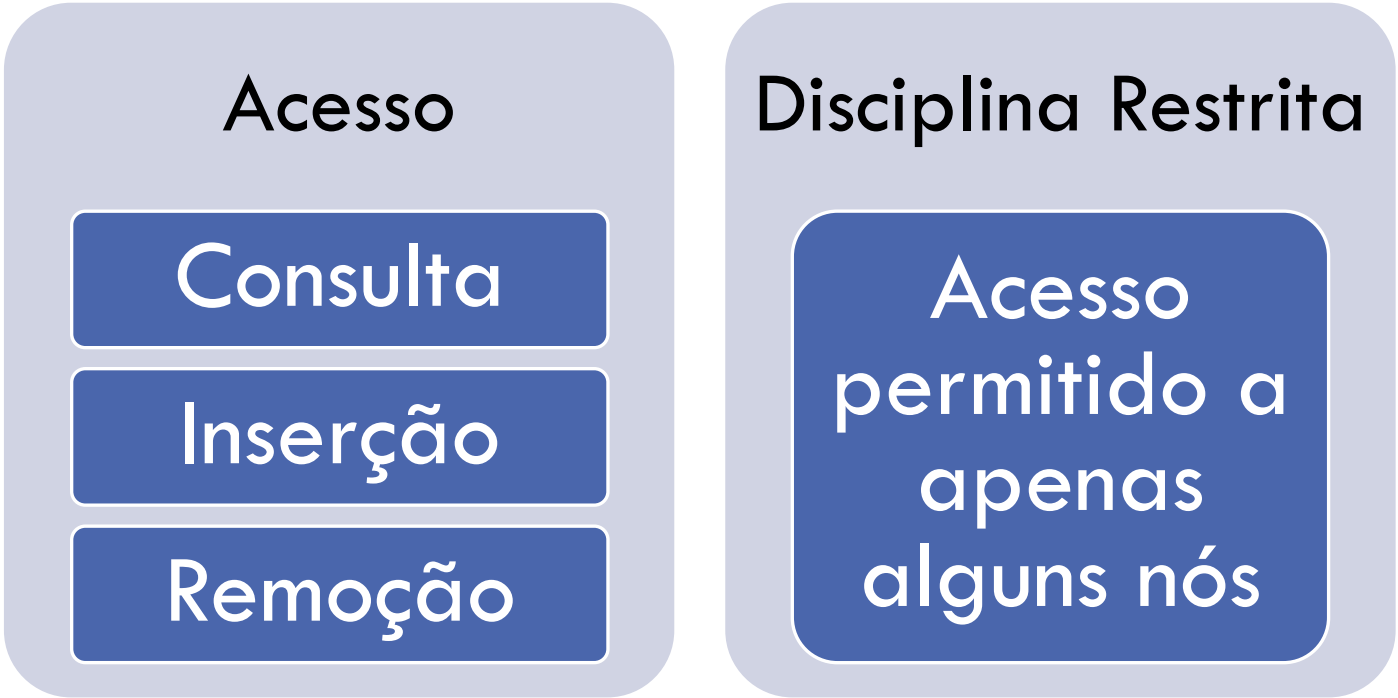
REVISÃO DE PILHAS E FILAS

Vanessa Braganholo

Estruturas de Dados e Seus Algoritmos

PILHAS E FILAS

São tipos especiais de **listas** com **disciplina restrita de acesso**

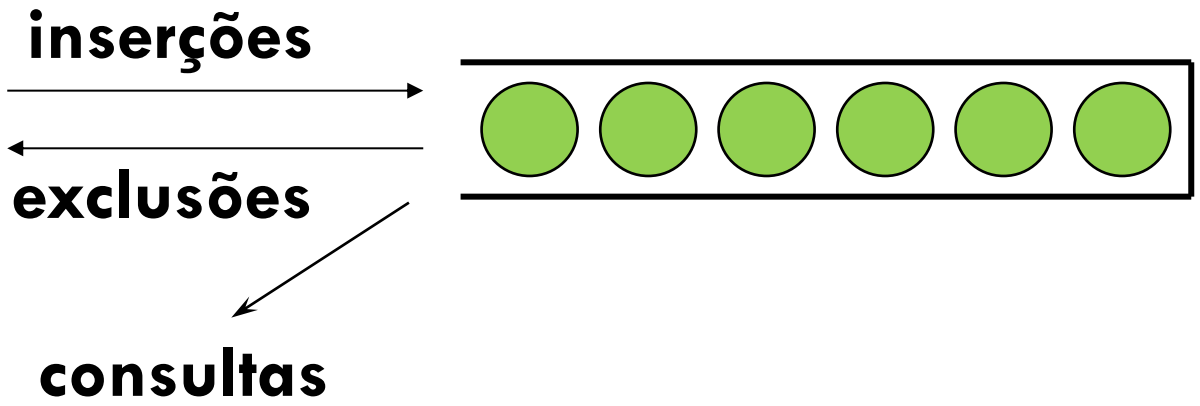


PILHAS

PILHAS

Todas as operações são executadas na **mesma extremidade** da pilha: o último componente inserido é o primeiro a ser retirado

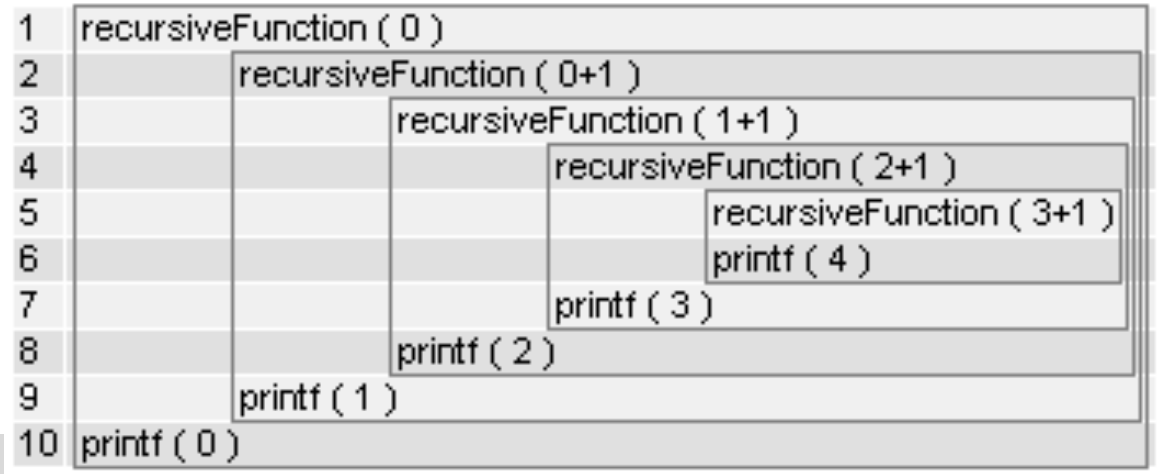
LIFO: Last In, First Out



EXEMPLO DE APLICAÇÃO: RECURSÃO

```
void recursiveFunction(int num)
{
    if (num < 5)
    {
        recursiveFunction(num + 1);
        printf("%d\n", num);
    }
}

int main() {
    recursiveFunction(0);
}
```

RECURSÃO

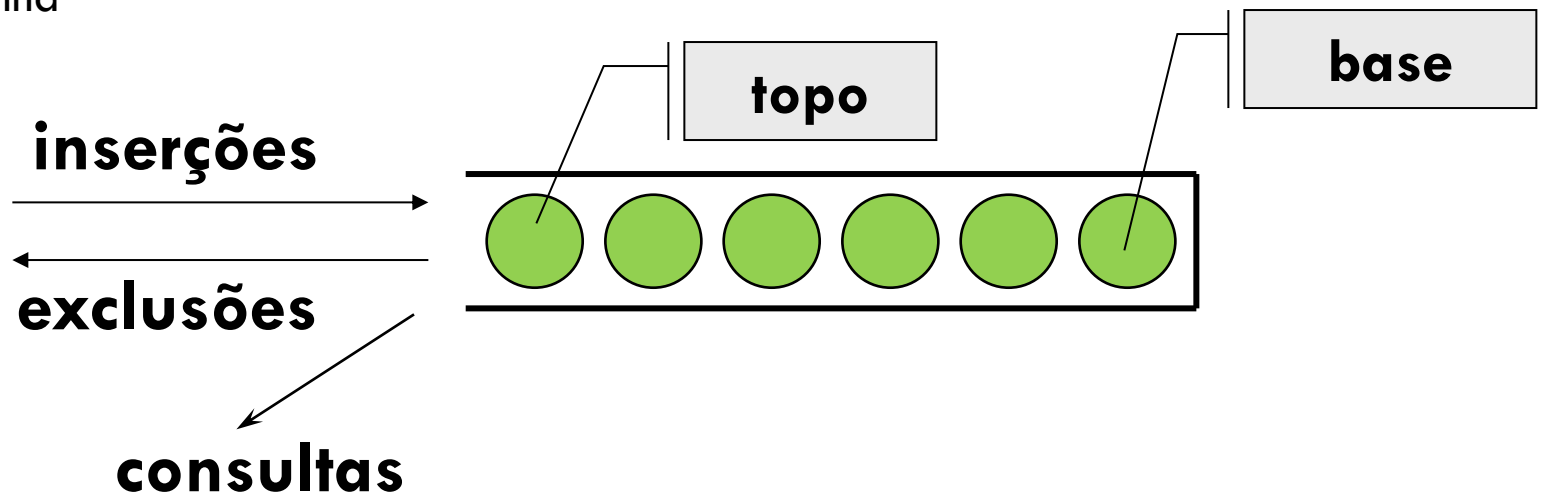



```
void recursiveFunction(int num)
{
    if (num < 5)
    {
        recursiveFunction(num + 1);
        printf("%d\n", num);
    }
}

int main() {
    recursiveFunction(0);
}
```

OPERAÇÕES SOBRE PILHAS

Operações válidas:

- Criar uma pilha vazia
- Inserir um nó no topo da pilha
- Excluir o nó do topo da pilha
- Consultar/Modificar o nó do topo da pilha
- Destruir a pilha



ALTERNATIVAS DE IMPLEMENTAÇÃO

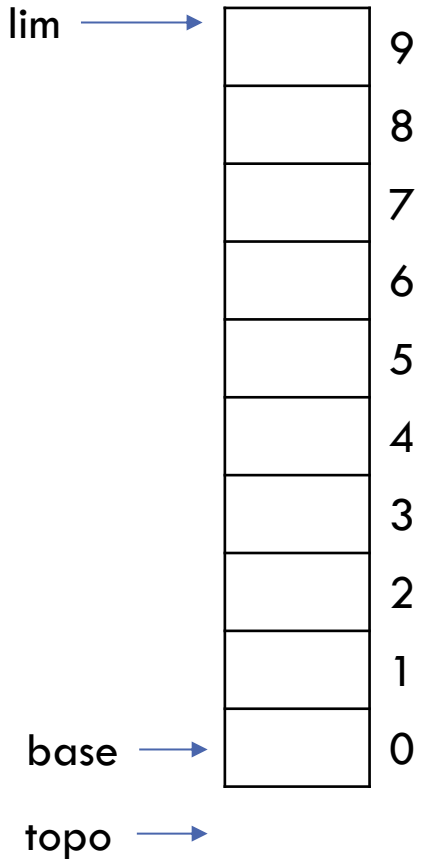
Vetores

Listas Encadeadas

IMPLEMENTAÇÃO DE PILHAS COM VETORES

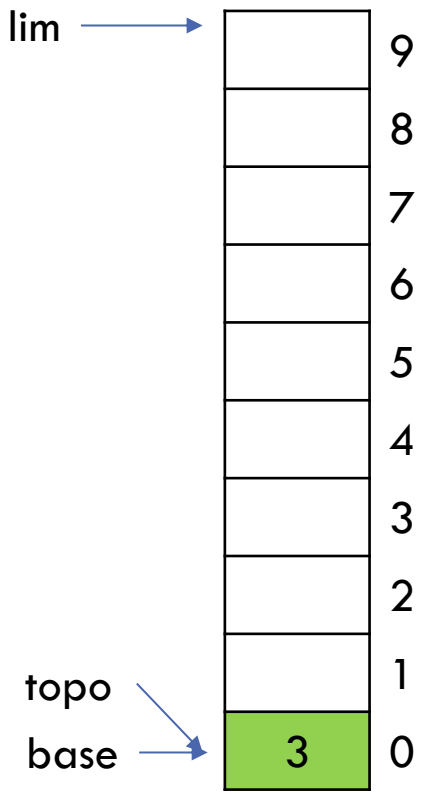
EXEMPLO DE MANIPULAÇÃO DE PILHA COM VETOR

1. Inicializar pilha de inteiros com máximo de 10 nós



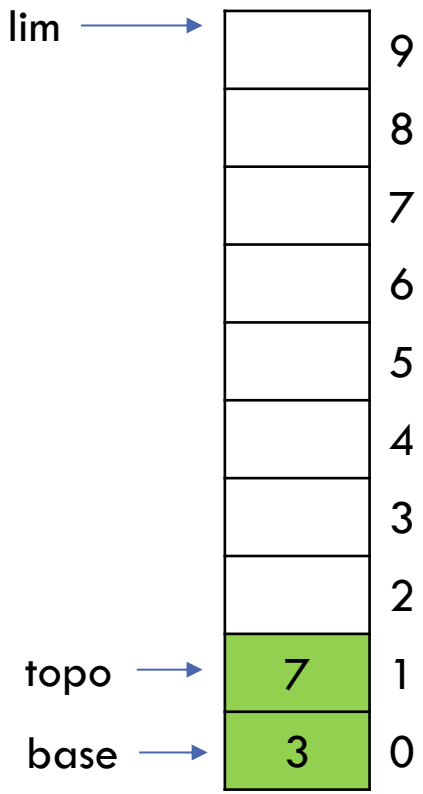
EXEMPLO DE MANIPULAÇÃO DE PILHA COM VETOR

1. Inicializar pilha de inteiros com máximo de 10 nós
2. Inserir nó com valor 3



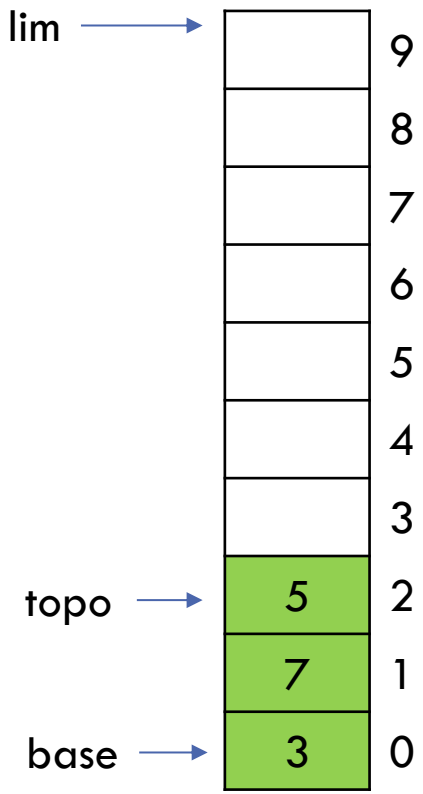
EXEMPLO DE MANIPULAÇÃO DE PILHA COM VETOR

1. Inicializar pilha de inteiros com máximo de 10 nós
2. Inserir nó com valor 3
3. Inserir nó com valor 7



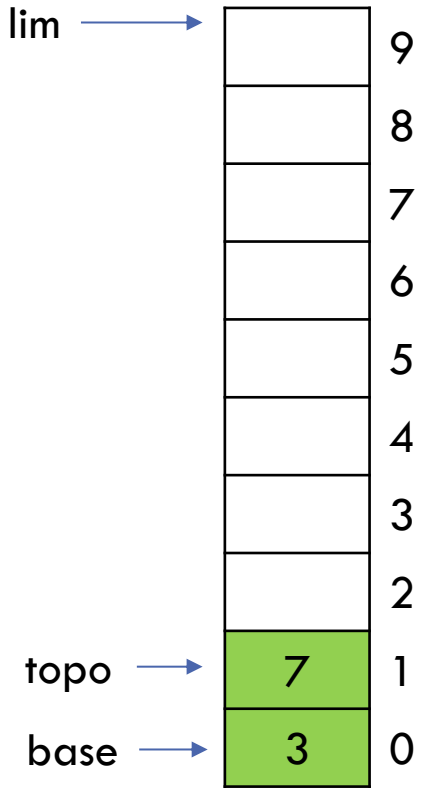
EXEMPLO DE MANIPULAÇÃO DE PILHA COM VETOR

1. Inicializar pilha de inteiros com máximo de 10 nós
2. Inserir nó com valor 3
3. Inserir nó com valor 7
4. Inserir nó com valor 5



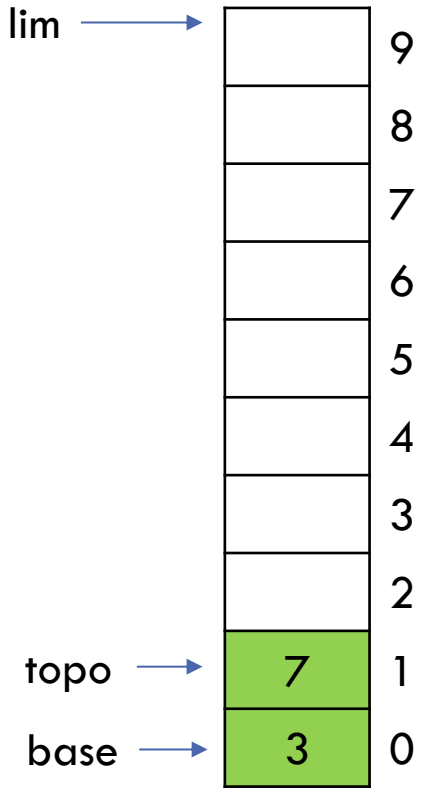
EXEMPLO DE MANIPULAÇÃO DE PILHA COM VETOR

1. Inicializar pilha de inteiros com máximo de 10 nós
2. Inserir nó com valor 3
3. Inserir nó com valor 7
4. Inserir nó com valor 5
5. Remover nó (nó removido é sempre o nó do topo)



EXEMPLO DE MANIPULAÇÃO DE PILHA COM VETOR

1. Inicializar pilha de inteiros com máximo de 10 nós
2. Inserir nó com valor 3
3. Inserir nó com valor 7
4. Inserir nó com valor 5
5. Remover nó (nó removido é sempre o nó do topo)
6. Consultar pilha (Retorna 7)

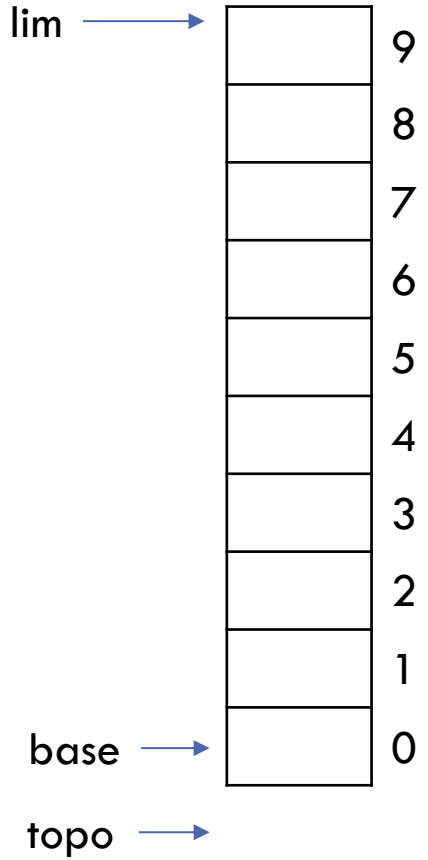


DECLARAÇÃO EM C

```
typedef struct pilha {
    int info;
} TPilha;

int base, lim, topo;
```

CRIAÇÃO DO VETOR NA FUNÇÃO MAIN E INICIALIZAÇÃO

```
void inicializa(int *base, int *lim, int *topo)
{
    *base = 0;
    *lim = 9;
    *topo = -1;
}

int main() {
   TPilha pilha[10];
   inicializa(&base, &lim, &topo);
   ...
 }
```



CRIAÇÃO DO VETOR NA FUNÇÃO MAIN E INICIALIZAÇÃO

```
void inicializa(int *base, int *lim, int *topo)
{
    *base = 0;
    *lim = 9;
    *topo = -1;
}

int main() {
   TPilha pilha[10];
   inicializa(&base, &lim, &topo);
   ...
 }
```

Note o uso de ponteiros para permitir alteração das variáveis **base, lim** e **topo.**

INSERÇÃO DE ELEMENTO: PUSH

```
int push(TPilha *pilha, int lim, int *topo, int info) {
    if (pilha_cheia(lim, *topo)) {
        return -1; //pilha está cheia, inserção inválida
    }
    else {
        //faz a inserção
        *topo = *topo + 1;
        pilha[*topo].info = info;
        return info; //retorna elemento inserido
    }
}
```

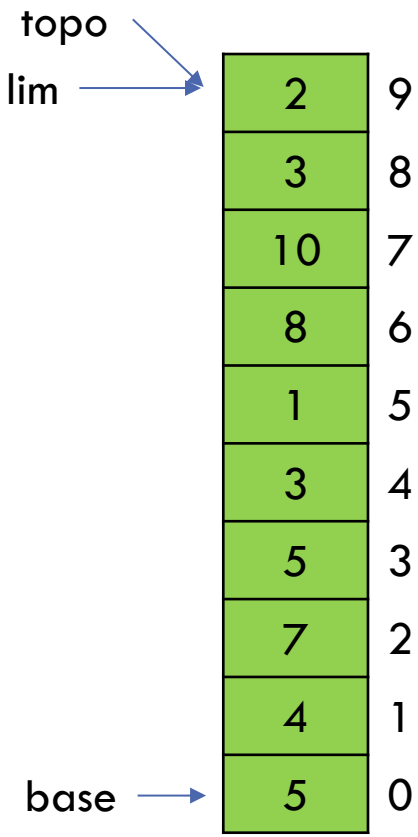
INSERÇÃO DE ELEMENTO: PUSH

```
int push(TPilha *pilha, int lim, int *topo, int info) {
    if (pilha_cheia(lim, *topo)) {
        return -1; //pilha está cheia, inserção inválida
    }
    else {
        //faz a inserção
        *topo = *topo + 1;
        pilha[*topo].info = info;
        return info; //retorna elemento
    }
}
```

Variáveis **lim** e **info** não serão alteradas (então não são ponteiros). Variável topo será alterada, então é ponteiro. Variável p é vetor, então é ponteiro.

CHECAR SE PILHA ESTÁ CHEIA

```
int pilha_cheia(int lim, int topo) {
    if (topo == lim)
        return 1;
    else
        return 0;
}
```

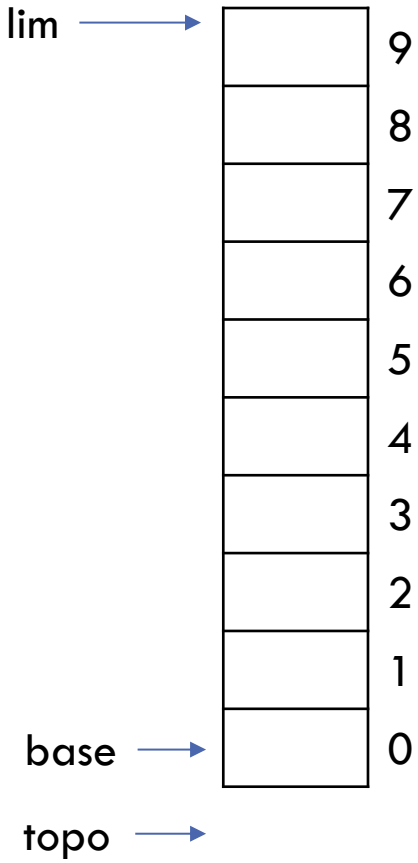


REMOÇÃO DE ELEMENTO: POP

```
int pop(TPilha *pilha, int base, int *topo) {
    if (pilha_vazia(base, *topo)) {
        return -1; //pilha vazia, remoção inválida
    }
    else {
        //faz a remoção
        int info = pilha[*topo].info;
        *topo = *topo - 1;
        return info; //retorna elemento removido
    }
}
```

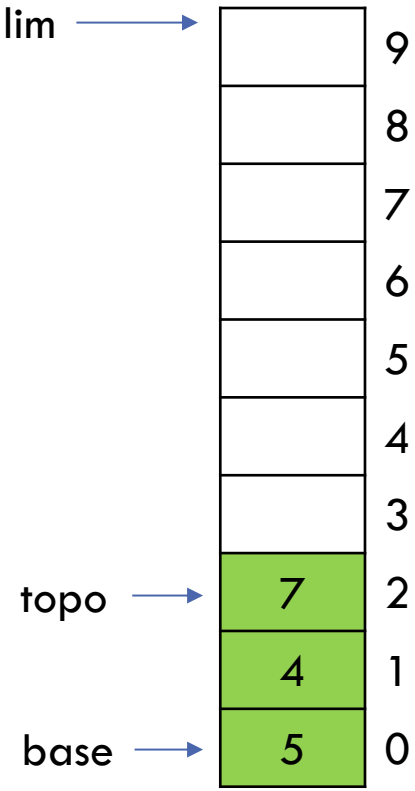
CHECAR SE PILHA ESTÁ VAZIA

```
int pilha_vazia(int base, int topo) {
    if (topo < base)
        return 1; //pilha vazia
    else
        return 0; //pilha não vazia

}
```



CONSULTAR TOPO DA PILHA: PEEK

```
int peek(TPilha *pilha, int base, int *topo) {
    if (pilha_vazia(base, *topo))
        return -1; //pilha vazia
    else {
        //faz consulta
        return pilha[*topo].info;
    }
}
```



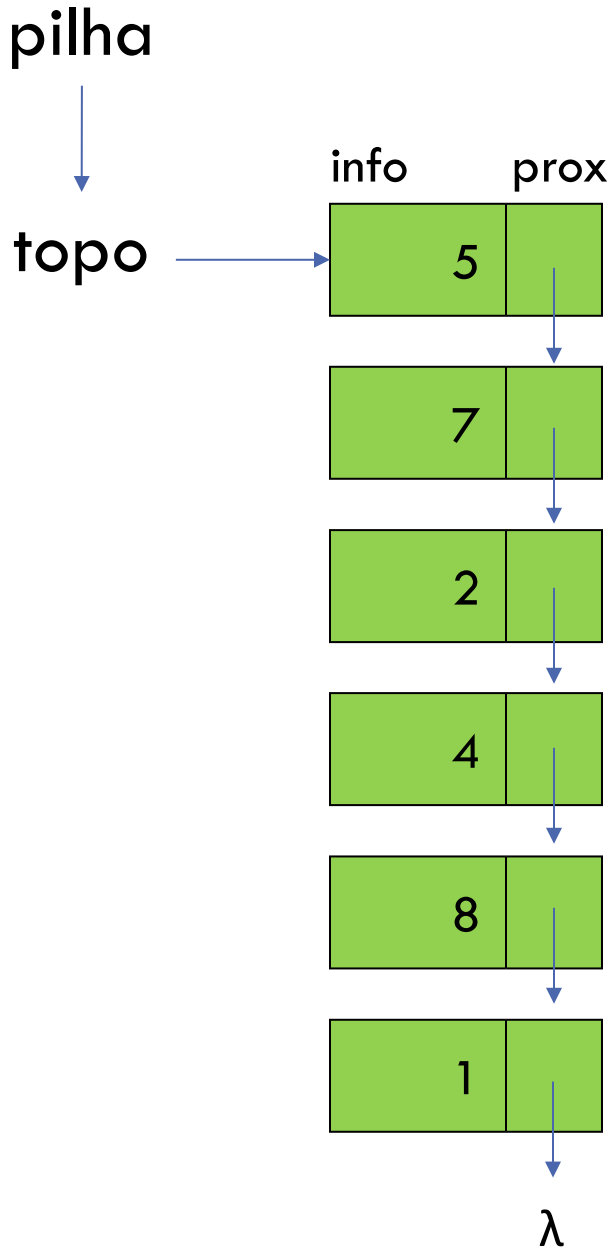
IMPLEMENTAÇÃO COMPLETA

A implementação completa está no site da disciplina

IMPLEMENTAÇÃO DE PILHAS COM LISTAS ENCADEADAS

PILHA COM LISTA ENCADEADA

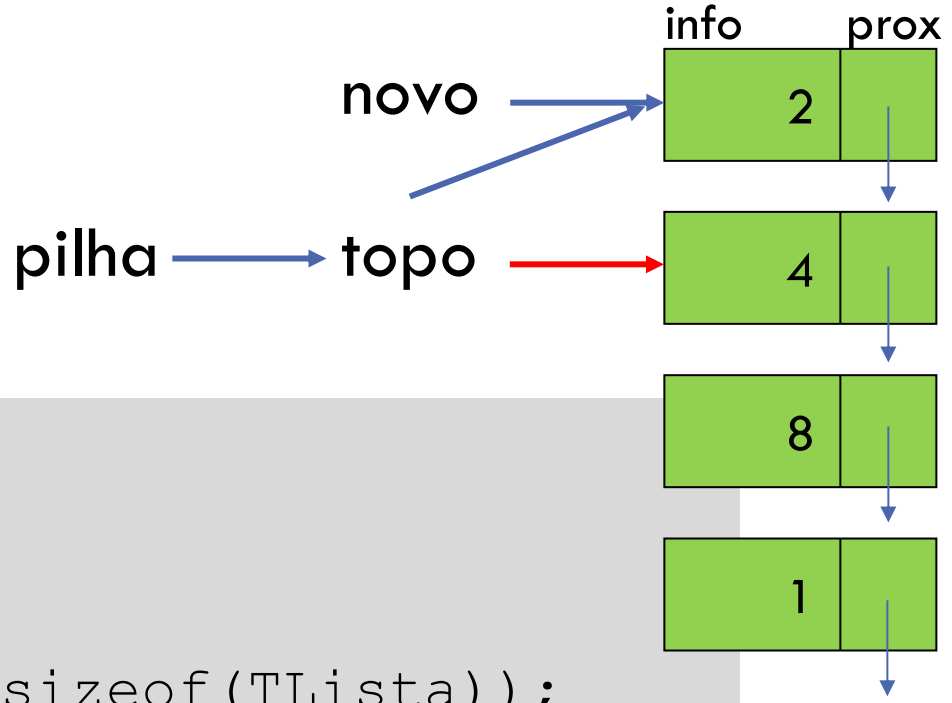
```
#include "lista-encadeada.h"

typedef struct pilha{
    TLista *topo;
} TPilha;
```



INICIALIZA


```
 TPilha *inicializa() {
     TPilha *pilha = (TPilha *)malloc(sizeof(TPilha));
     pilha->topo = NULL;
     return pilha;
 }

int main() {
     TPilha *pilha = inicializa();
     ...
}
```

PUSH

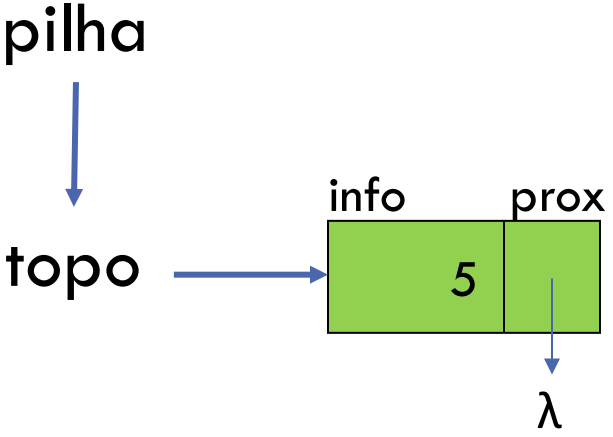



```
 /* *
  * Insere elem no topo da pilha
  * */
void push(TPilha *pilha, int elem) {
    TLista *novo = (TLista*) malloc(sizeof(TLista));
    novo->info = elem;
    novo->prox = pilha->topo;
    pilha->topo = novo;
}
```

λ

CHAMADA DO PUSH NA FUNÇÃO MAIN

```
 int main() {
     TPilha *pilha = inicializa();
     push(pilha, 5);
     ...
}
```



EXERCÍCIO: IMPLEMENTAR O POP

```
 /* *
  * Exclui o elemento do topo da pilha
  * retorna o info do elemento excluído
  */
int pop(TPilha *pilha) {
    //TODO

 }
```

EXERCÍCIO: IMPLEMENTAR O PEEK

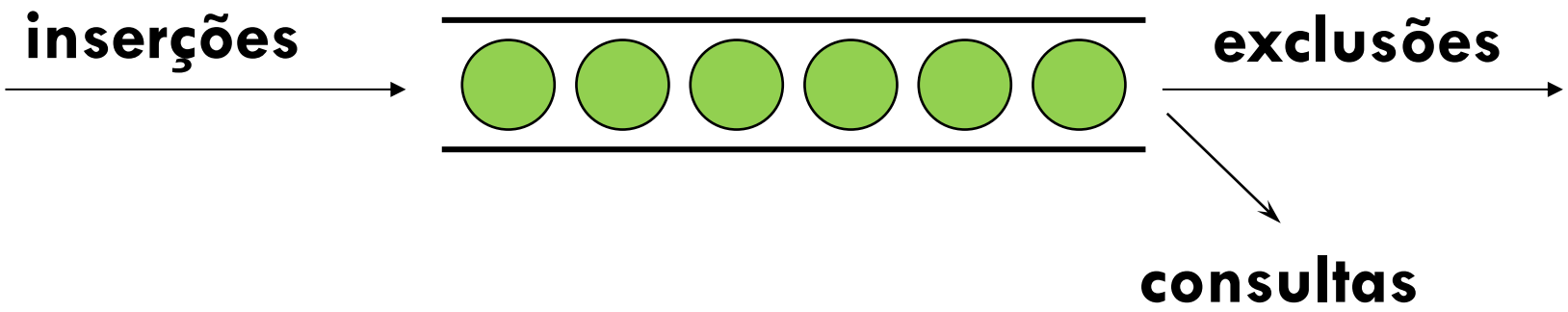
```
 /* *
  * Consulta o elemento do topo da pilha
  * retorna info do elemento do topo
  */
int peek(TPilha *pilha) {
    //TODO

 }
```

FILAS

FILAS

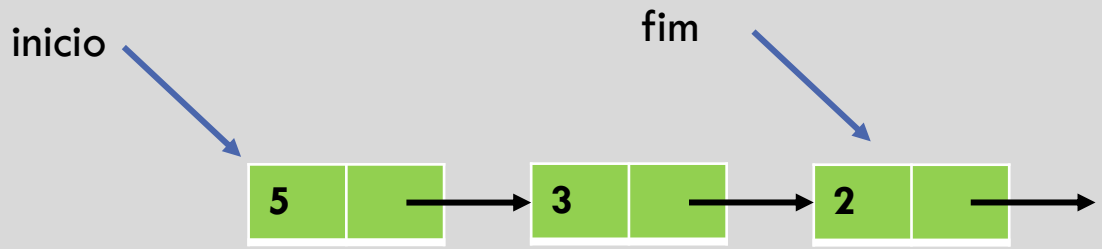
Inserções são executadas em uma extremidade, e exclusões na outra: o primeiro componente inserido é o primeiro a ser retirado

FIFO: First In, First Out

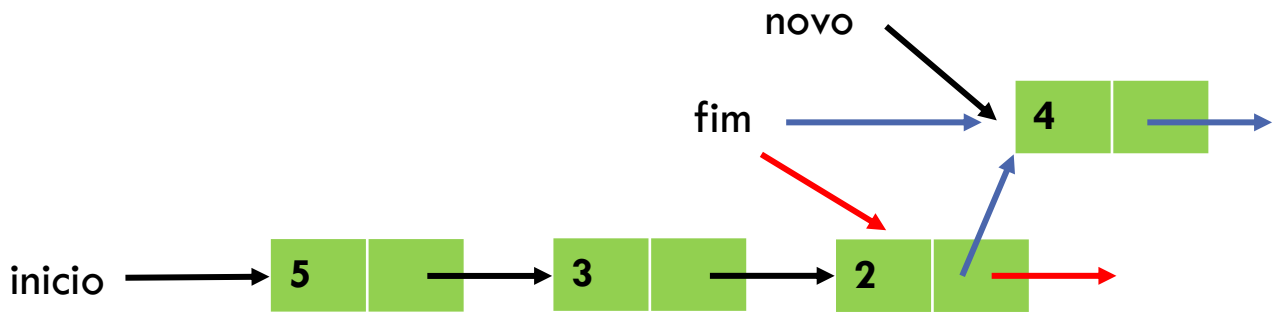


DECLARAÇÃO

```
#include "lista-encadeada.h"

typedef struct fila {
    TLista *inicio;
    TLista *fim;
} TFila;
```



INSERÇÃO (SEMPRE NO FIM)

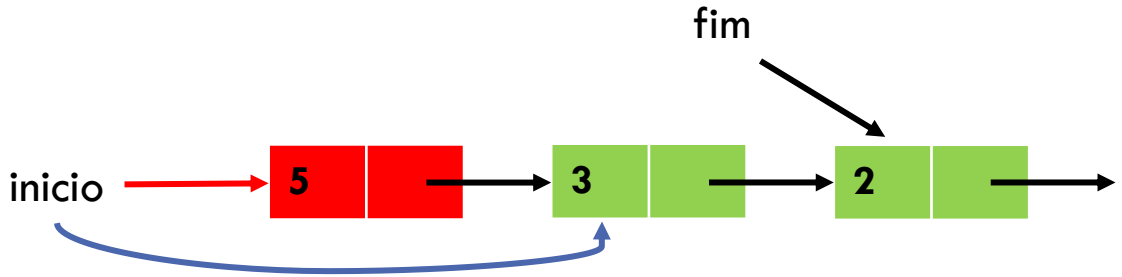



```
void insere(TFila *f, int elem){
    TLista *novo = (TLista *)malloc(sizeof(TLista));
    novo->info = elem;
    novo->prox = NULL; //inserção no fim da fila
    if (!fila_vazia(f)){
        f->fim->prox = novo;
    }
    else{
        f->inicio = novo;
    }
    f->fim = novo; //elt. novo é o novo fim da fila
}
```

FILA VAZIA

```
int fila_vazia(TFila *f){
    if (f->inicio == NULL) {
        return 1;
    }
    else return 0;
}
```

RETIRAR ELEMENTO DA FILA (SEMPRE DO INÍCIO)




```
int retira(TFila *f){
    if (fila_vazia(f)){
        exit(1);
    }
    int info = f->inicio->info;

    TLista *aux = f->inicio;
    f->inicio=f->inicio->prox;
    //se elemento removido era o único da fila
    //faz fim apontar para NULL também
    if (f->inicio == NULL) {
        f->fim = NULL;
    }
    free(aux);
    return info;
}
```

EXERCÍCIOS

1. Faça uma função que imprime o conteúdo da fila, usando as funções de inserção e remoção (ver especificação no Google Classroom)
2. Faça uma função que altera o elemento do início da fila (ver especificação no Google Classroom)
3. Faça uma função que altera o elemento do topo da pilha (ver especificação no Google Classroom)

AGRADECIMENTOS

Material baseado nos slides de Renata Galante

Instituto de Informática, UFRGS

Agradecimento a Isabel Rosseti pela implementação