Pilhas

Listas lineares especiais

Disciplina restrita de acesso

Acesso

- consulta
- inserção
- remoção

Disciplina restrita

- acesso permitido somente em alguns nós

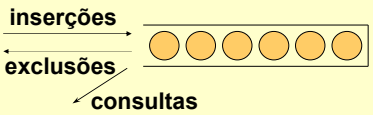
Listas lineares especiais mais usuais

LIFO *Last In First Out*
o último componente inserido é o primeiro a ser retirado

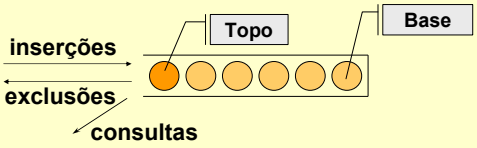
Pilha

Pilhas e Filas

Pilhas:



Operações sobre Pilhas



Operações válidas:

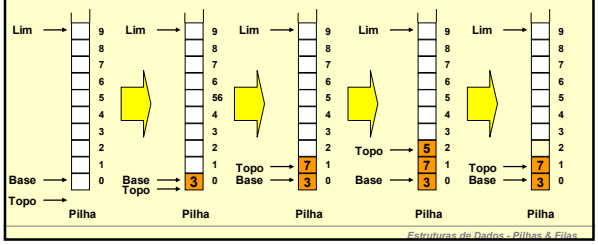
- criar uma pilha vazia
- inserir um nó no topo da pilha
- excluir o nó do topo de pilha
- consultar / modificar nó do topo da pilha
- destruir a pilha

Pilhas
Contigüidade física

Exemplo de manipulação de uma pilha

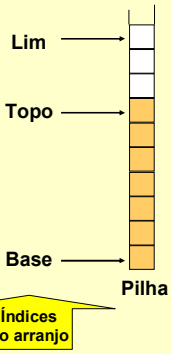
1. Inicializar pilha de valores inteiros,a partir do índice 1, máximo 10 nós
2. Inserir nó com valor 3
3. Inserir nó com valor 7
4. Inserir nó com valor 5
5. Remover nó
6. Consultar pilha

Retorna "7"



Pilha - contigüidade física

```
typedef struct T_Produto {
                    int cod;
                    char nome[40];
                    float preco;
                }TProduto;

                int base, limite, topo;
```



Operações válidas:

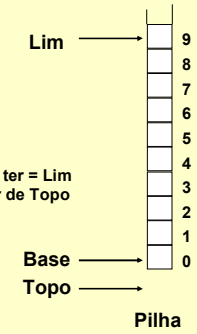
- criar uma pilha vazia
- inserir um nó no topo da pilha
- excluir o nó do topo da pilha
- consultar / modificar nó do topo da pilha
- destruir a pilha

Índices do arranjo

Pilha - contigüidade física

Operações :

- **criar uma pilha vazia**
- inserir um nó no topo da pilha
- excluir o nó do topo da pilha
- consultar / modificar nó do topo da pilha
- destruir a pilha



1. Definir valor do índice de Base da pilha
2. Definir valor máximo de nós que a pilha pode ter = Lim
3. Indicar que a pilha está vazia através do valor de Topo

Exemplo:

base = ????
limite = ????
topo = ????

Criar pilha vazia - contigüidade física

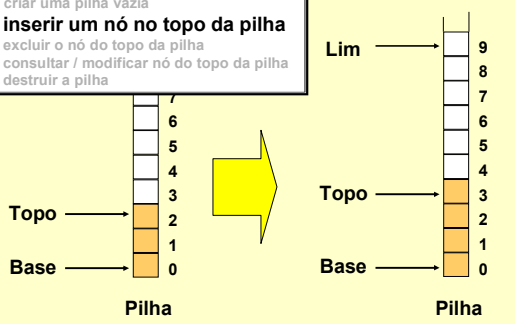
```
void inicializa ( TProduto t[], int *base, int *limite, int *topo) {
    int i;

    for (i=0; i<MAX; i++) {
        strcpy(t[i].nome,"");
        t[i].cod=0;
        t[i].preco=0;
    }

    *base = 0;
    *limite = 10;
    *topo = base -1;
}
```


Pilha - contigüidade física

Operações :

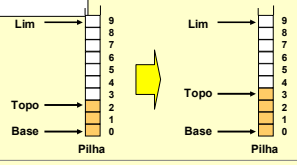
- criar uma pilha vazia
- **inserir um nó no topo da pilha**
- excluir o nó do topo da pilha
- consultar / modificar nó do topo da pilha
- destruir a pilha

Operação PUSH



Inserir nó em pilha - contigüidade física

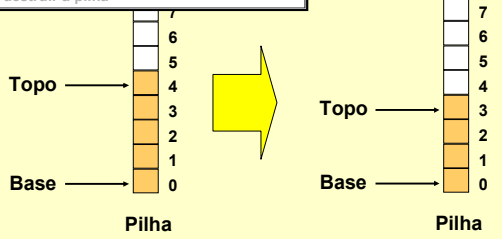
```
void PushPilha ( TProduto t[], int limite, int *topo) {
  int i;
  if ( *topo < limite )
  {
    *topo = *topo +1;

    /* Lendo os dados*/
    printf("Codigo: "); scanf("%d", &t[*topo].cod);
    printf("Nome: "); scanf ("%s", t[*topo].nome);
    printf("Preco: "); scanf ("%f", &t[*topo].preco);
  }
}
```



Pilha - contigüidade física

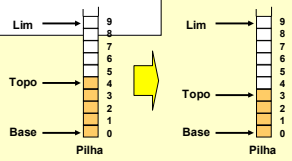
Operações :
- criar uma pilha vazia
- inserir um nó no topo da pilha
- **excluir o nó do topo da pilha**
- consultar / modificar nó do topo da pilha
- destruir a pilha

Operação POP



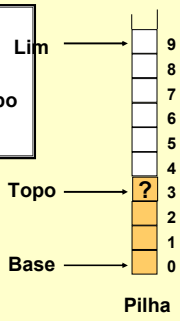
Excluir nó de pilha - contigüidade física

```
int PopPilha( TProduto t[], int base, int limite, int *topo) {
  rem=0;

  if ( *topo >= base)
  {
    rem =  t[*topo].cod;
    *topo = *topo -1;
    return rem;
   }
}
```



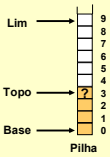
Pilha - contigüidade física

Operações :
- criar uma pilha vazia
- inserir um nó no topo da pilha
- excluir o nó do topo da pilha
- **consultar / modificar nó do topo da pilha**
- destruir a pilha



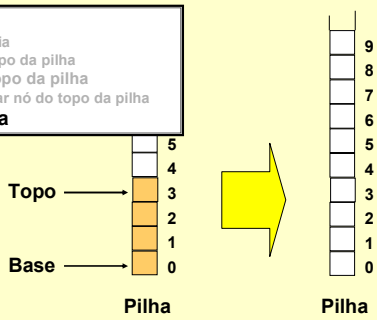
Consultar pilha - contigüidade física

```
int consulta ( TProduto t[], int base, int topo) {

  if ( topo >= base)
      return t[topo].cod;
  else
      return -1;

}
```



Pilha - contigüidade física

Operações :
- criar uma pilha vazia
- inserir um nó no topo da pilha
- excluir o nó do topo da pilha
- consultar / modificar nó do topo da pilha
- **destruir a pilha**

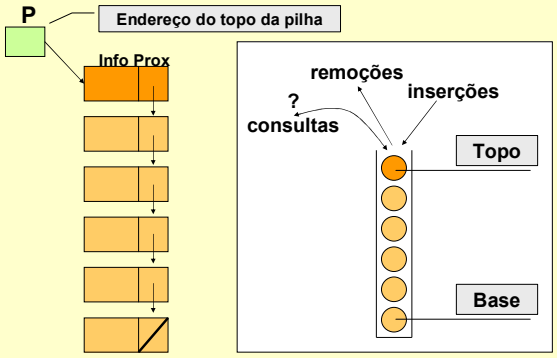


Destruir pilha - contigüidade física

```
void destroi ( TProduto t[], int *base, int *limite, int *topo) {
    int i;

    *base = -1;
    *limite = -1;
    *topo = -1;
}
```

Pilhas
Encadeamento

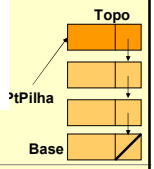
Pilha encadeada



Pilha encadeada

Operações válidas:

```
TipoPilha* InicializaPilha (TipoPilha *Topo);
// cria pilha vazia

TipoPilha* PushPilha (TipoPilha *Topo, TipoInfo Dado);
// insere nodo no topo da pilha

int PopPilha (TipoPilha **Topo, TipoInfo *Dado);
// exclui nodo do topo da pilha

TipoInfo ConsultaPilha (TipoPilha *Topo);
// retorna 0 se a pilha estiver vazia

TipoPilha* DestroiPilha (TipoPilha *Topo);
// libera posições ocupadas pela pilha
```



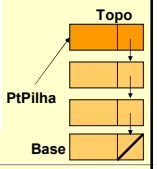
Pilha encadeada

```
typedef struct TPtPilha TipoPilha;
typedef int TipoInfo;
```

```
struct TPtPilha{
      TipoInfo dado;
      struct TPtPilha *elo;
};
```

Operações válidas:

```
TipoPilha* InicializaPilha (TipoPilha *Topo);
// cria pilha vazia
int Vazia (TipoPilha *Topo)
// retorna 1 se a pilha estiver vazia
TipoPilha* PushPilha (TipoPilha *Topo, TipoInfo Dado);
// insere nodo no topo da pilha
int PopPilha (TipoPilha **Topo, TipoInfo *Dado);
// exclui nodo do topo da pilha
//retorna 1 se exclui e zero se não exclui
TipoInfo ConsultaPilha (TipoPilha *Topo);
// retorna 0 se a pilha estiver vazia
TipoPilha* DestroiPilha (TipoPilha *Topo);
// libera posições ocupadas pela pilha
```

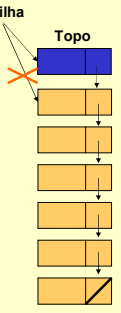


Criar Pilha Vazia

```
TipoPilha* inicializa(void)
{
      return NULL;
}
```

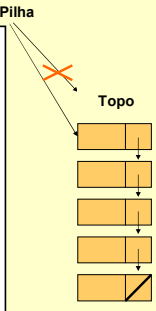

Inserção em pilha encadeada

```
TipoPilha* PushPilha (TipoPilha *Topo, TipoInfo Dado)
{
      TipoPilha *novo; //novo elemento
      TipoPilha *ptaux = Topo;

      /*aloca um novo nodo */
      novo = (TipoPilha*) malloc(sizeof(TipoPilha));
      novo->elo = NULL;

      /*insere a informação no novo nodo*/
      novo->dado = Dado;

      /*encaeia o elemento*/
      novo->elo = Topo;
      Topo = novo;
      return Topo;
}
```



Remoção de pilha encadeada

```
int PopPilha (TipoPilha **Topo, TipoInfo *Dado)
//retorna 1 se exclui e zero se não exclui
{
  TipoPilha* ptaux;

  if (Topo == NULL)
    return 0;   //não tem nada na pilha
  else
  {
      *Dado = (*Topo)->dado;     // devolve o valor do topo
       ptaux = *Topo;          //guarda o endereço do topo
      *Topo = (*Topo)->elo;     //o próximo passa a ser o topo
      free(ptaux);           //libera o que estava no topo
      ptaux=NULL;
      return 1;
  }
}
```



Consulta à pilha encadeada

```
TipoInfo ConsultaPilha (TipoPilha *Topo)
{
 if (Topo==NULL)
   return 0;
 else
   return Topo->dado;

}
```


Destruição de uma pilha encadeada

```
TipoPilha* DestroiPilha (TipoPilha *Topo)
{
       TipoPilha *ptaux;
       while (Topo != NULL)
       {
         ptaux = Topo;         //guarda o endereço do topo
         Topo = Topo->elo;     //o próximo passa a ser o topo
         free(ptaux);          //libera o que estava no topo
       }
       return NULL;
}
```


Exercícios sugeridos - pilhas

Considere 2 pilhas (P1 e P2) cujos campos de informação são formados por 1 campo inteiro. Estas pilhas estão ordenadas pelos campos de valor inteiro (ordem crescente a partir da base da pilha).

Formar uma terceira pilha que contenha todos os elementos das 2 pilhas originais. A terceira pilha deverá estar, também, ordenada, em ordem crescente.

Resolva este problema considerando alocação encadeada.

Única estrutura de dados permitida para solução do exercício PILHA