UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Estrutura de Dados

Pilhas

Aula – 04

Thiago Naves
Universidade Tecnológica Federal do Paraná
Bacharelado em Ciência da Computação

Tipos de dados e Estruturas de dados

- Tipos básicos (primitivos)
  - inteiro, real, e caractere
- Tipos de estruturados (construídos)
  - arranjos (vetores e matrizes)
  - estruturas
  - sequências (conjuntos)
  - referências (ponteiros)
- Tipos definidos pelo usuário
  - Estruturas

Tipos de dados e Estruturas de dados

- Tipos de dados básicos
  - São fornecidos pela Linguagem de Programação
- Estruturas de Dados
  - É uma estruturação conceitual dos dados
  - Reflete um relacionamento lógico entre dados, de acordo com o problema considerado
- As Estruturas de dados nos levam a conhecer os Tipos Abstratos de Dados (TADs)

Tipo Abstrato de Dado

Um **TAD** é uma forma de definir um novo tipo de dado juntamente com as operações que manipulam esse novo tipo de dado

Tipo Abstrato de Dado

- Separação entre conceito (definição do tipo) e implementação das operações
- Visibilidade da estrutura interna do tipo fica limitada às operações
- Aplicações que usam o TAD são denominadas *clientes* do tipo de dado
- Cliente tem acesso somente à forma abstrata do TAD

Introdução

- Exemplo de uma estrutura para jogo de cartas
- Para representar um baralho precisamos:
  - Representar cartas: naipe e valor

    struct carta { char naipe; char valor; };
  - Operações de manipulação:
    - Comprar a carta no topo do baralho
    - Colocar uma carta no fundo do baralho
    - Embaralhar
    - Retirar uma carta aleatória do meio do baralho

Tipo Abstrato de Dado

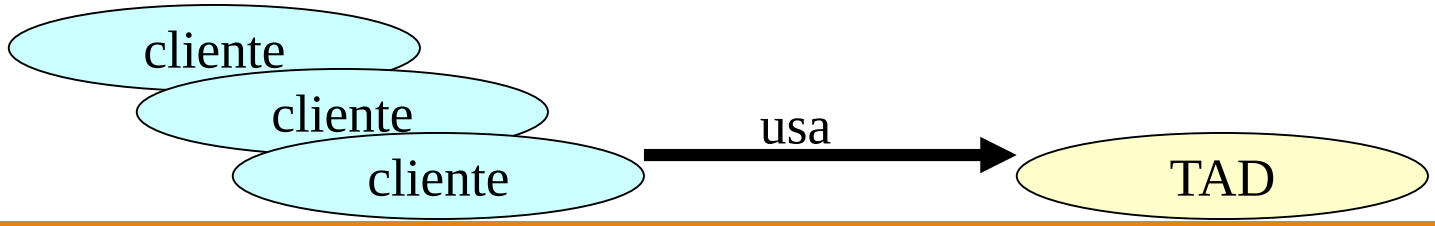
Exemplo (contaBancaria.h)

```
// definição do tipo
typedef struct {
 int numero;
 float saldo;
} ContaBancaria;

// cabeçalho das funções
void Inicializa (ContaBancaria*, int, float);
void Deposito (ContaBancaria*, float);
void Saque (ContaBancaria*, float);
void Imprime (ContaBancaria);
```

Tipo Abstrato de Dado

- Possibilidade de utilização do mesmo TAD em diversas aplicações diferentes
- Possibilidade de alterar o TAD sem alterar as aplicações que o utilizam
- Código do cliente do TAD não depende da implementação
- Segurança:
  - clientes não podem alterar a representação
  - clientes não podem tornar os dados inconsistentes

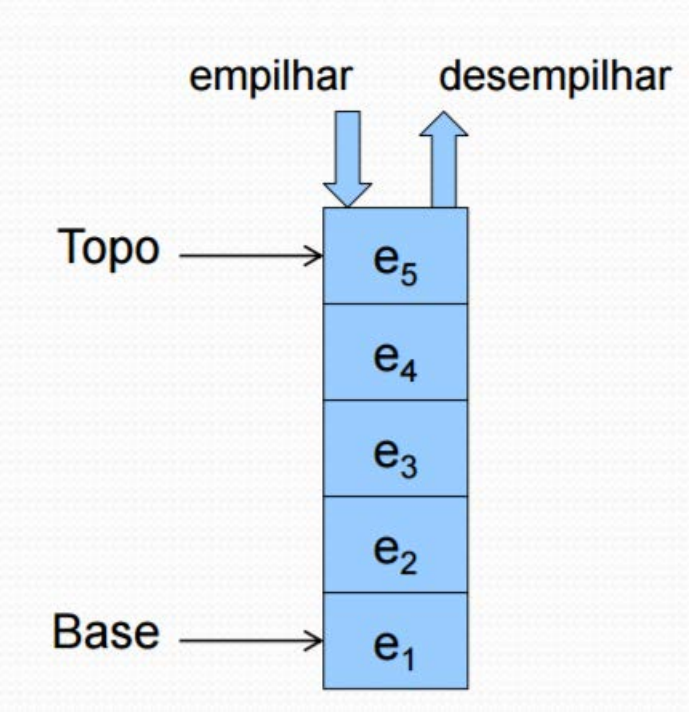


Pilha

- São estruturas de dados do tipo LIFO (last-in first-out)
  - O último elemento a ser inserido, será o primeiro a ser retirado.
- A manipulação dos elementos é dada apenas por uma das extremidades da lista - topo
- Para processar o penúltimo item inserido, deve-se remover o último.
- Exemplos de pilhas são:
  - pilha de pratos,
  - pilha de livros,
  - pilha de cartas de um baralho,
  - etc.

Pilha

- Na implementação de pilha, em apenas uma das extremidades, chamada de topo, é realizada a manipulação dos elementos, em oposição a outra extremidade, chamada de base.
- Todas as operações em uma pilha podem ser imaginadas como as que ocorre numa pilha de pratos em um restaurante ou como num jogo com as cartas de um baralho



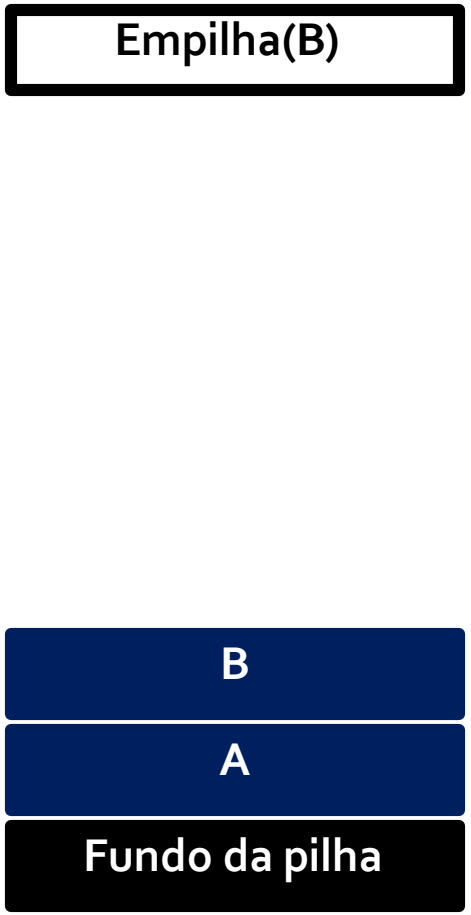
Operações com Pilha

- Operações:
  - Criar uma pilha
  - Empilhar um elemento
  - Desempilhar um elemento
  - Recuperar o tamanho da pilha
  - Destruir uma pilha

- Último elemento a entrar é o primeiro elemento a sair

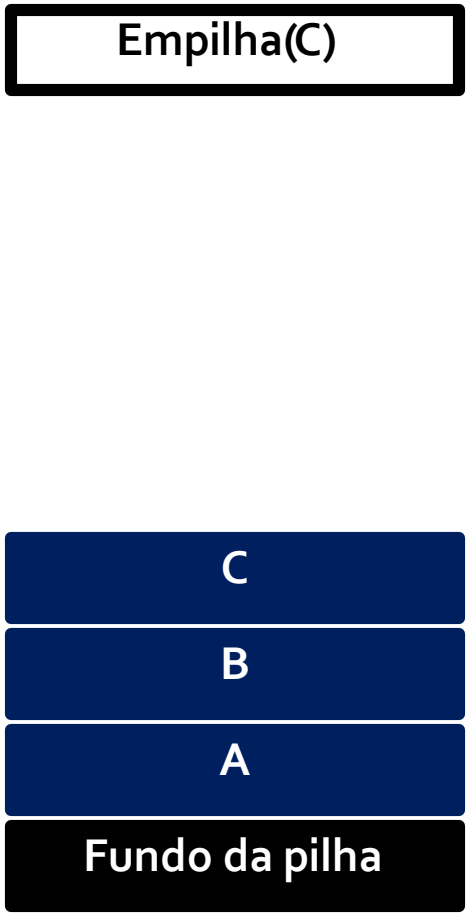
Operações com Pilha

- Operações:
  - Criar uma pilha
  - Empilhar um elemento
  - Desempilhar um elemento
  - Recuperar o tamanho da pilha
  - Destruir uma pilha
- Último elemento a entrar é o primeiro elemento a sair



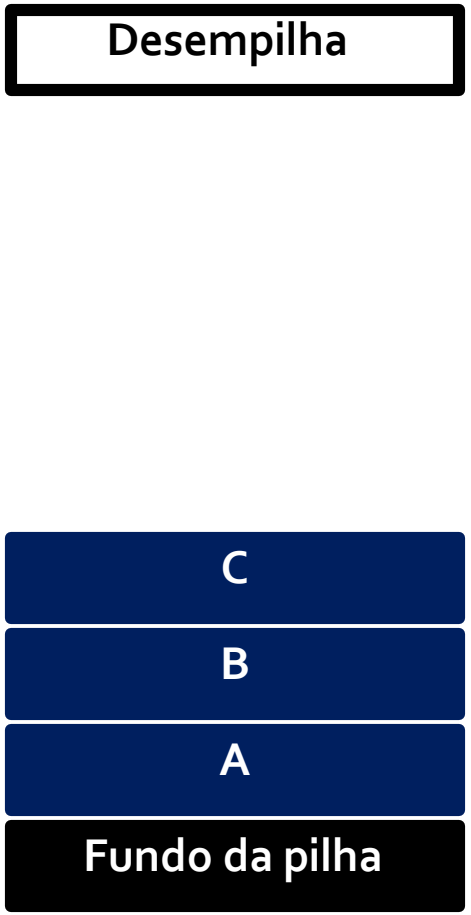
Operações com Pilha

- Operações:
  - Criar uma pilha
  - Empilhar um elemento
  - Desempilhar um elemento
  - Recuperar o tamanho da pilha
  - Destruir uma pilha
- Último elemento a entrar é o primeiro elemento a sair



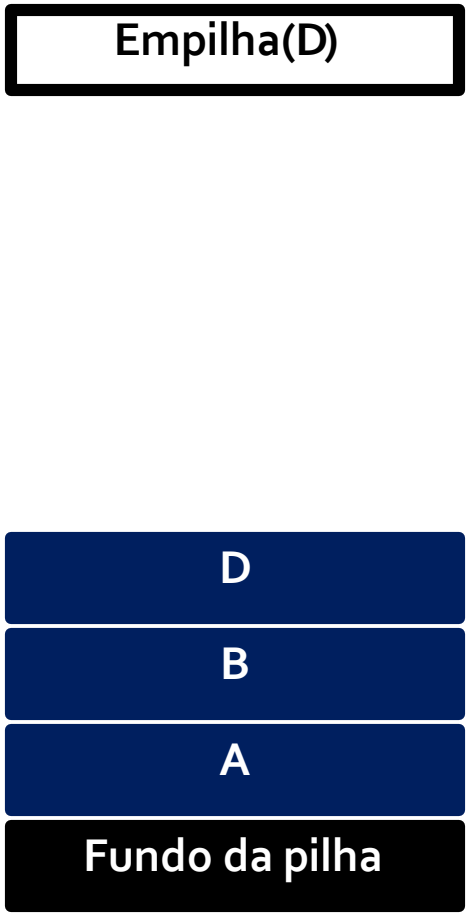
Operações com Pilha

- Operações:
  - Criar uma pilha
  - Empilhar um elemento
  - Desempilhar um elemento
  - Recuperar o tamanho da pilha
  - Destruir uma pilha
- Último elemento a entrar é o primeiro elemento a sair



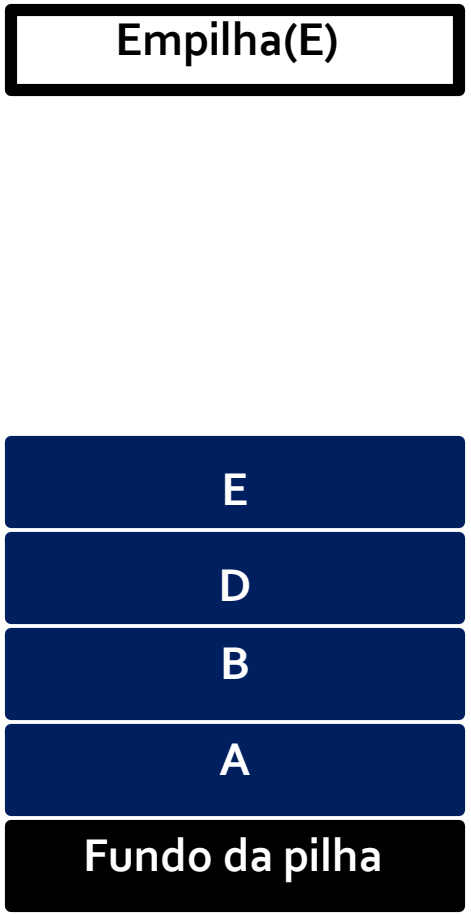
Operações com Pilha

- Operações:
  - Criar uma pilha
  - Empilhar um elemento
  - Desempilhar um elemento
  - Recuperar o tamanho da pilha
  - Destruir uma pilha
- Último elemento a entrar é o primeiro elemento a sair



Operações com Pilha

- Operações:
  - Criar uma pilha
  - Empilhar um elemento
  - Desempilhar um elemento
  - Recuperar o tamanho da pilha
  - Destruir uma pilha
- Último elemento a entrar é o primeiro elemento a sair

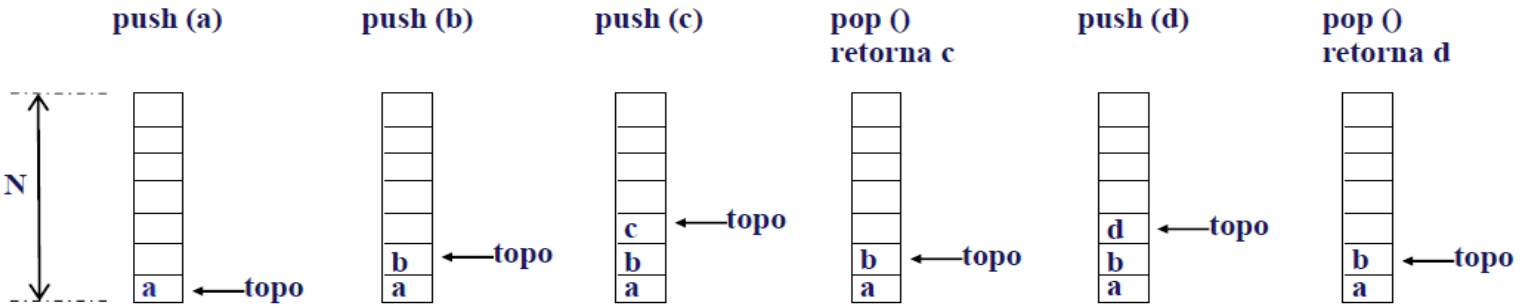


Uso da Pilha

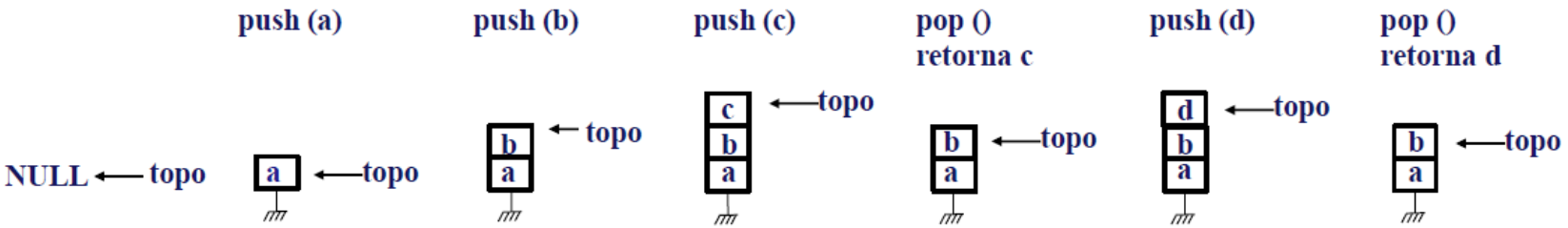
- Usa-se pilha em aplicações em que os dados são obtidos na ordem inversa àquela em que foram fornecidos.
- Exemplos:
  - Calculadora para expressões matemáticas;
  - Conversão de número decimal para binário;
  - Retirada de mercadorias de um caminhão de entregas;
  - Mecanismo de fazer/desfazer do Word;
  - Mecanismo de navegação de páginas na Internet (avançar e retornar).

Pilhas

- Uma estrutura do tipo Pilha pode ser implementada como Vetor ou Lista Encadeada:
  - Vetor



  - Lista encadeada

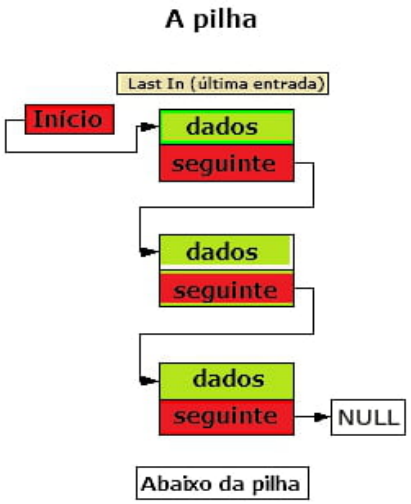


Pilhas

- Utilizar vetor para representar uma Pilha.
  - Não é eficiente, pois o vetor terá tamanho fixo e estático
  - Operações de inserir e remover da pilha serão lentas, pois o vetor precisa alterar a posição de todos os elementos armazenados
- Utilizar lista encadeada para representar uma Pilha
  - A estrutura será dinâmica, ou seja, sem tamanho fixo
  - Operações de inserir e remover serão rápidas, pois é necessário alterar apenas os elementos próximos;
  - Todos os elementos estarão protegidos de alteração na memória do computador
  - Controle total da Pilha através de ponteiros.

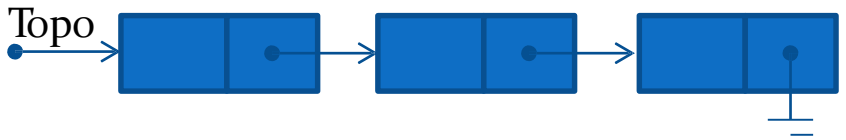
Pilhas

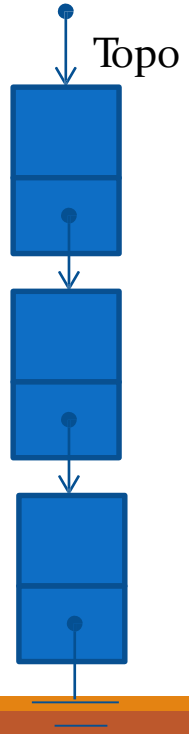
- Assim, vamos aprender a representar uma Pilha através de listas encadeadas, que são diversos elementos alocados dinamicamente conectados entre si.
- Toda as conexões entre elementos e como a lista deve representar o formato de uma Pilha é um trabalho manual do programador.



Pilha Dinâmica (Dynamic Stack)

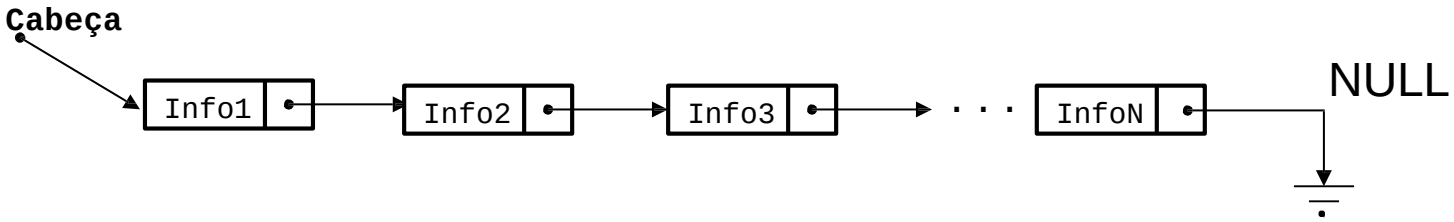
- Pilha implementada através de uma lista linear encadeada.
- É uma lista linear encadeada em que as operações de inserção e retirada de um elemento é realizada em uma das extremidades da lista, chamada de topo.
- O topo da pilha será o início da lista
  - Empilhar = inserir antes do primeiro da lista
  - Desmpilhar = remover o primeiro da lista





Lista Encadeada Dinamicamente

- Elementos são armazenados em nós, e cada nó tem um apontador para o próximo nó na lista.

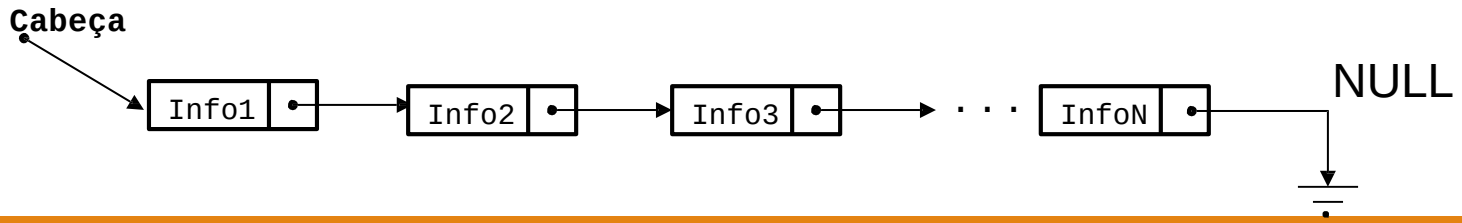


- Ou seja, as células não existem nativamente como uma posição, variável ou algo parecido com um vetor.
  - Nós criamos as células como uma *struct* alocada dinamicamente e controlamos a sequência linear lógica entre essas células

- O mais importante é sempre manter armazenado a posição da primeira célula, para que seja possível acessar as demais a frente desta na ordem linear.

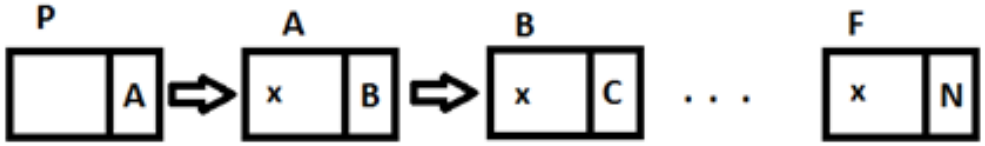
Lista Encadeada Dinamicamente

- Lista encadeada
  - Sequência encadeada de elementos, chamados de nós da lista
  - Nó da lista é representado por dois campos:
    - a informação armazenada e
    - o ponteiro para o próximo elemento da lista
  - A lista é representada por um ponteiro para o primeiro nó (Cabeça)
  - O ponteiro do último elemento é NULL



Lista Encadeada Dinamicamente

- Lista encadeada
  - Uma lista encadeada (= linked list = lista ligada) é uma sequência de células;
  - Cada célula contém um objeto de algum tipo e o endereço da célula seguinte. A estrutura de cada célula de uma lista deve ser do mesmo tipo dos dados armazenados.



P: primeiro elemento (nó) da lista
x: dado armazenado
F: último elemento (nó) da lista
N: NULL

Lista Encadeada Dinamicamente

- Iremos definir uma nova forma de representar dados e sua sequencia de armazenamento.
  - Pela semelhança das operações de uma lista e um vetor, podemos entender que iremos definir uma nova forma de representar vetores.

- Todas as células são do mesmo tipo e armazenam algum tipo de dado.

- Reparem que o ponteiro para o primeiro elemento (Cabeça) precisa conseguir enxergar o elemento tendo nível de acesso maior que as demais células

Operações com uma Pilha Dinâmica

- Criação da pilha
  - Declarar o ponteiro para o topo da pilha
- Inicialização da pilha
  - Determina o status inicial da pilha, a fim de prepará-la para a inserção de dados.
  - Ponteiro para o topo é NULL – aponta para nada
- Empilhamento
  - Consiste em inserir um valor no topo da pilha (início da lista).
- Verificar se a pilha está vazia
  - Caracterizada pelo ponteiro para topo estar com NULL.
- Desempilhamento
  - Consiste em retirar um valor do topo da pilha. É preciso verificar previamente se a pilha está vazia.
- Mostrar o topo

Interface do tipo Pilha Dinâmica

```
struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};

typedef struct elemento* Pilha;

Pilha* cria_Pilha();
void libera_Pilha(Pilha* pi);
int consulta_topo_Pilha(Pilha* pi, struct aluno *al);
int insere_Pilha(Pilha* pi, struct aluno al);
int remove_Pilha(Pilha* pi);
int tamanho_Pilha(Pilha* pi);
int Pilha_vazia(Pilha* pi);
int Pilha_cheia(Pilha* pi);
void imprime_Pilha(Pilha* pi);
```

Interface do tipo Pilha Dinâmica

- A interface é o arquivo com extensão (.h), chamado de header ou cabeçalho que descreve as funções que uma estrutura irá conter.

- A separação da estrutura em cabeçalho (.h) e depois sua implementação em arquivo (.c) é o princípio do uso de bibliotecas.
  - Assim como “stdlib.h” ou “stdio.h”

- Notem que o projeto da pilha com código em anexo já está estruturado desta forma.

Descrição do Nó

Criar a pilha é alocar um nó com nível de acesso maior e retonar ele, esse irá sempre apontar para o topo:

```
struct aluno{
    int matricula;
    char nome[30];
    float n1,n2,n3;
};
```

A descrição de um nó:

```
struct elemento{
    struct aluno dados;  /* dado que é armazenado na pilha*/
    struct elemento *prox;
};
```

Módulo do tipo Pilha Dinâmica

- O módulo é o arquivo com extensão (.c), que especifica os códigos dos protótipos de funções feitas no header (.h)
- Com o código implementado, sempre que a função for chamada o compilador saberá como ela se comporta.
- A seguir vamos acompanhar a criação do código do modulo da pilha dinâmica, com implementação das funções prototipadas no header.

Criar a Pilha

- Criar a pilha é alocar um nó com nível de acesso maior e retonar ele, esse irá sempre apontar para o topo:

```
Pilha* pi =(Pilha*) malloc(sizeof(Pilha));
if(pi != NULL)
     *pi = NULL;
 return pi;
```

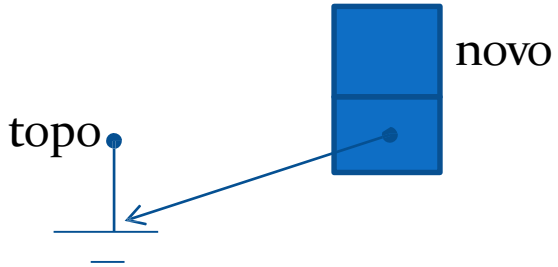
- Determina o status inicial da pilha, a fim de prepará-la para a inserção de dados.
- Inicialmente o ponteiro para o topo não aponta para nada:

***pi = NULL;**

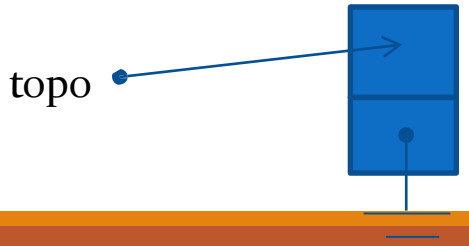
Empilhar (Push)

- Consiste em inserir um valor no topo da pilha.
  - Cria-se um novo nó

  - Novo nó aponta para o nó para o qual o topo aponta

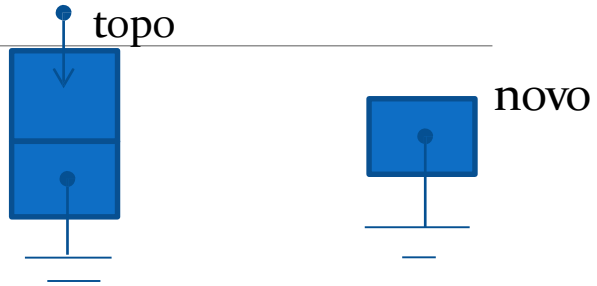


  - topo aponta para o novo nó

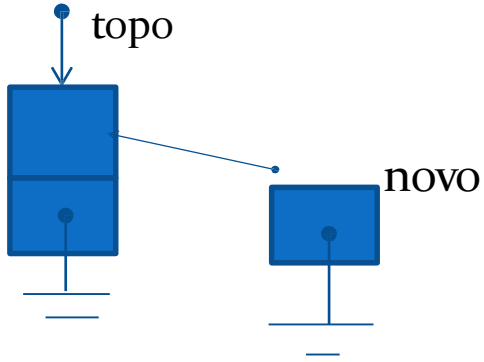


Empilhar (Push)

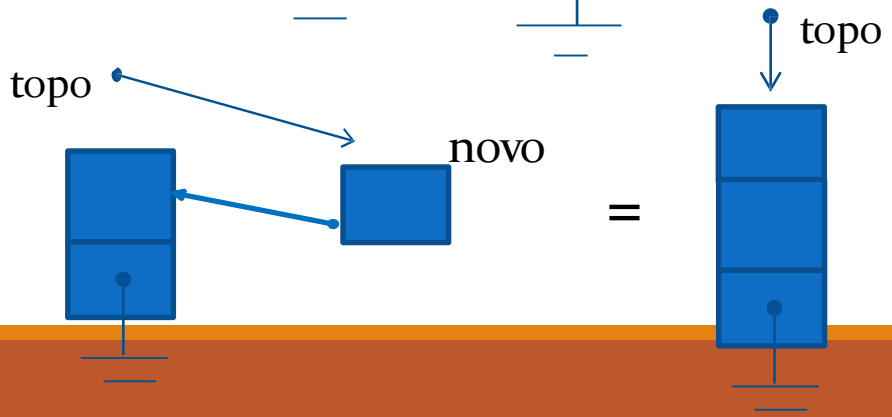
- Cria-se um novo nó



- Novo nó aponta para o nó para o qual o topo aponta



- topo aponta para o novo nó

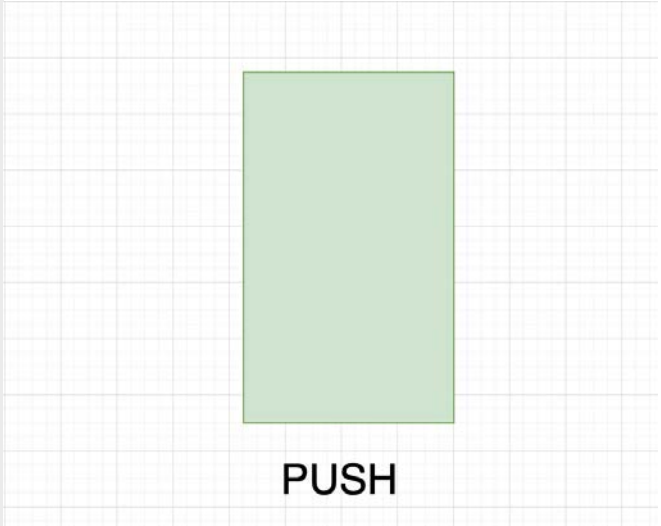


Empilhar (Push)

A descrição de um nó:

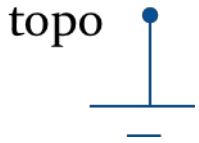
```
if(pi == NULL)
   return 0;
Elem* no;
no = (Elem*) malloc(sizeof(Elem));
if(no == NULL)
   return 0;
no->dados = al;
no->prox = (*pi);
*pi = no;

return 1;
```



Verificar se a Pilha está Vazia

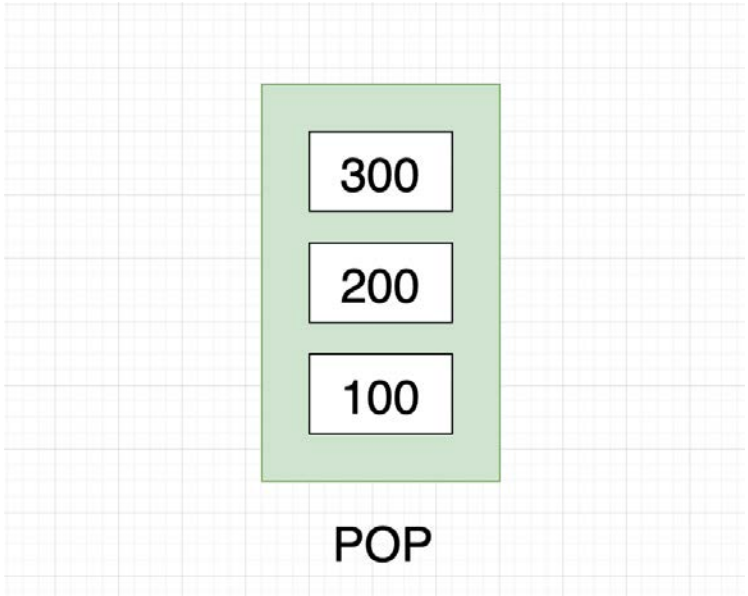
- A pilha estará vazia, quando o ponteiro para o topo estiver apontando para nada (estiver aterrado)

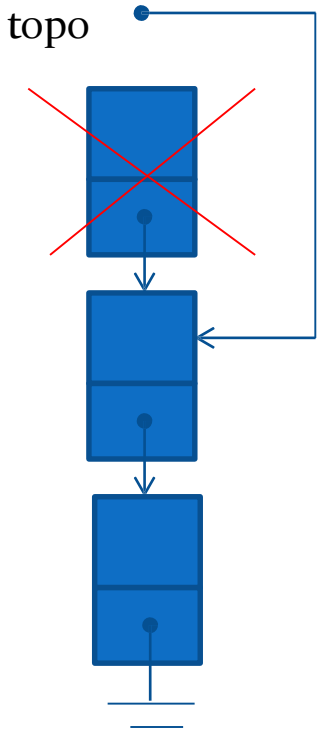



```
int Pilha_vazia(Pilha* pi){
    if(pi == NULL)
        return 1;
    if(*pi == NULL)
        return 1;
    return 0;
}
```

Desempilhar (Pop)

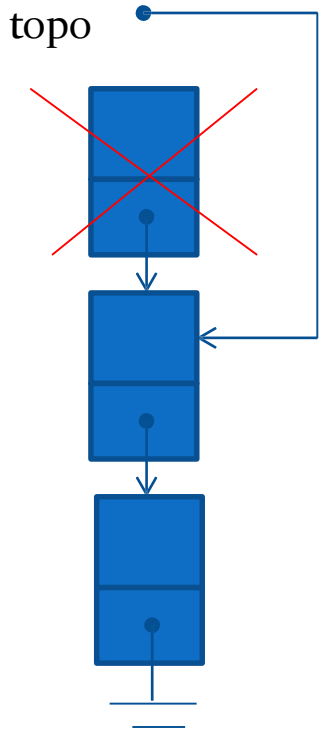
- O ponteiro que indica o topo irá apontar para o próximo do topo;
- Antigo topo deve ser removido





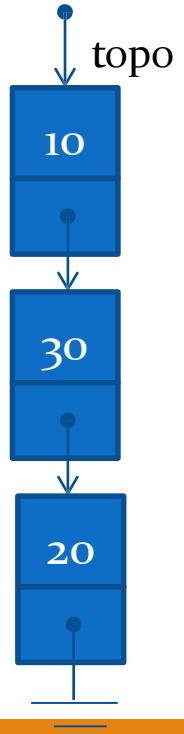
Desempilhar (Pop)

```
int remove_Pilha(Pilha* pi){
    if(pi == NULL)
        return 0;
    if((*pi) == NULL)
        return 0;
    Elem *no = *pi;
    *pi = no->prox;
    free(no);
    return 1;
}
```



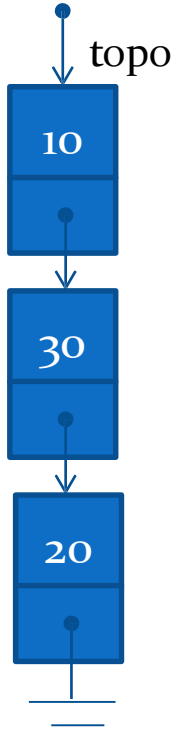
Mostrar o Topo

- Deve-se verificar se existem elementos na pilha (ou seja, se a pilha não está vazia)
- Numa pilha convencional só se pode vizualizar os dados do elemento que esta no topo (no exemplo o 10)



Mostrar o Topo

```
int consulta_topo_Pilha(Pilha* pi, struct aluno *al){
    if(pi == NULL)
        return 0;
    if((*pi) == NULL)
        return 0;
    *al = (*pi)->dados;
    return 1;
}
```



Pilha Dinâmica

- Exemplo de uso

```
int main(){
    struct aluno a[4] = {{2,"Andre",9.5,7.8,8.5},
                         {4,"Ricardo",7.5,8.7,6.8},
                         {1,"Bianca",9.7,6.7,8.4},
                         {3,"Ana",5.7,6.1,7.4}};
    Pilha* pi = cria_Pilha();
    printf("Tamanho: %d\n\n\n\n",tamanho_Pilha(pi));
    int i;
    for(i=0; i < 4; i++)
        insere_Pilha(pi,a[i]);

    imprime_Pilha(pi);
    printf("Tamanho: %d\n\n\n\n",tamanho_Pilha(pi));
```

Pilha Dinâmica

- Exemplo de uso

```
 for(i=0; i < 4; i++)
        remove_Pilha(pi);
    printf("Tamanho: %d\n\n\n\n",tamanho_Pilha(pi));
    imprime_Pilha(pi);

    for(i=0; i < 4; i++)
        insere_Pilha(pi,a[i]);

    printf("Tamanho: %d\n\n\n\n",tamanho_Pilha(pi));
    imprime_Pilha(pi);

    libera_Pilha(pi);
    system("pause");
    return 0;
}
```

Testando a Pilha Dinâmica

- Executem o arquivo main do projeto de pilha dinâmica em anexo.
- Observe o resultado de saida e faça modificações no main para compreender melhor o comportamento da estrutura de dados Pilha.
- Insiram novos elementos, vejam o topo da pilha e retorne o seu tamanho com as funções especìficas para esse fim.

Material Auxiliar

- Excelentes vídeo aulas sobre estrutura de dados em C

Aulas do Professor Dr. André Backes
https://www.youtube.com/channel/UCUc6UwvpQfOLDE7e52-OCMw

Referências Oficiais

- BACKES, André. **Linguagem C, Completa e Descomplicada**. Editora Elsevier. 1ª Edição. Rio de Janeiro, 2013.

- CORMEM, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. **Algoritmos: teoria e prática**. 3. ed. São Paulo, SP: Campus, 2012.

- PEREIRA, Silvio L. **Estrutura de Dados Fundamentais: Conceitos e Aplicações**. 12. ed. São Paulo, SP: Érica, 2010.

Referências Oficiais

- BOAVENTURA NETTO, Paulo O. JURKIEWICS, Samuel. **Grafos: Introdução e prática**. 5. ed. São Paulo, SP: Blucher, 2012.
- FORBELLONE, André L. V.; EBERSPASHER, Henri F. **Lógica de programação: a construção de algoritmos e estruturas de dados**. 3. ed. São Paulo, SP: Prentice Hall, 2005.
- GOLDBARG, Marco; GOLDBARG, Elizabeth. **Grafos: Conceitos, Algoritmos e Aplicações**. 1. ed. São Paulo, SP: Campus, 2012.
- SZWARCFITER, Jayme L.; MARKENSON, Lilian. **Estruturas de dados e seus algoritmos**. 3. ed. Rio de Janeiro, RJ: LTC, 2010.
- TOSCANI, Laira V.; VELOSO, Paulo A. S. **Complexidade de algoritmos :análise, projeto e métodos**. 3. ed. Porto Alegre, RS: Bookman, 2012.