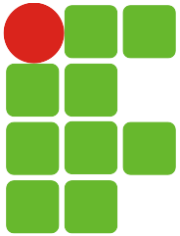
# Estrutura de Dados

**Profa. Marta Talitha Carvalho**

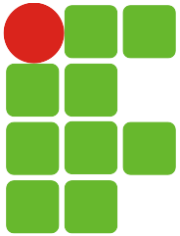
Aula 4: PILHA

# Pilha



- Uma das estruturas mais simples é a pilha. Possivelmente é a razão de ser bastante utilizada em programação, sendo inclusive implementada diretamente no hardware da maioria das máquinas modernas.

# Pilha



LIFO *Last In First Out*

o último componente inserido
é o primeiro a ser retirado

Pilha

FIFO *First In First Out*

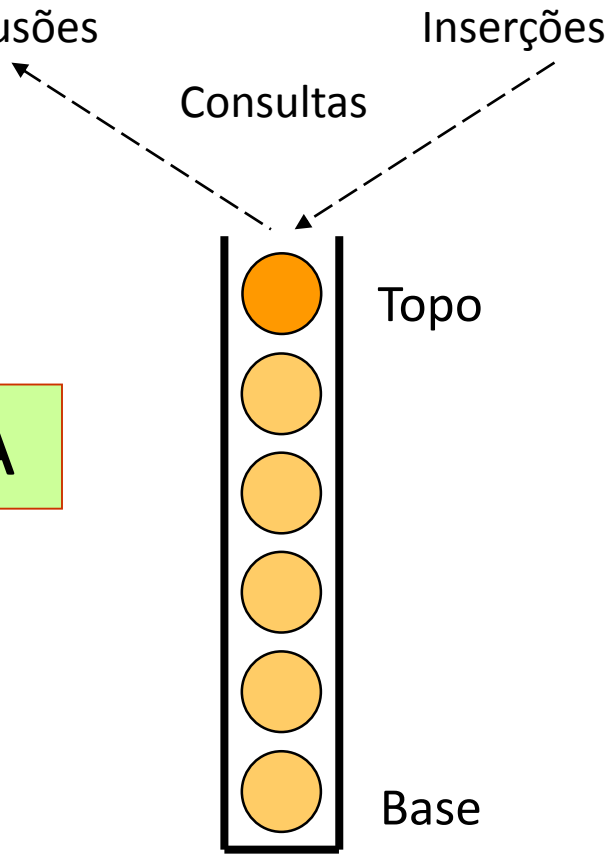
o primeiro componente inserido
é também o primeiro a ser retirado

Fila

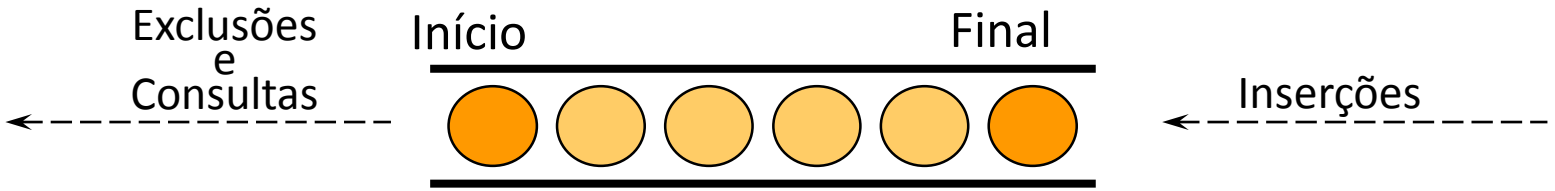
# Pilha

ANALOGIA

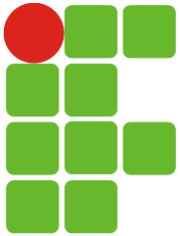
PILHA




FILA



# Qual é a diferença entre uma Lista e uma Pilha?

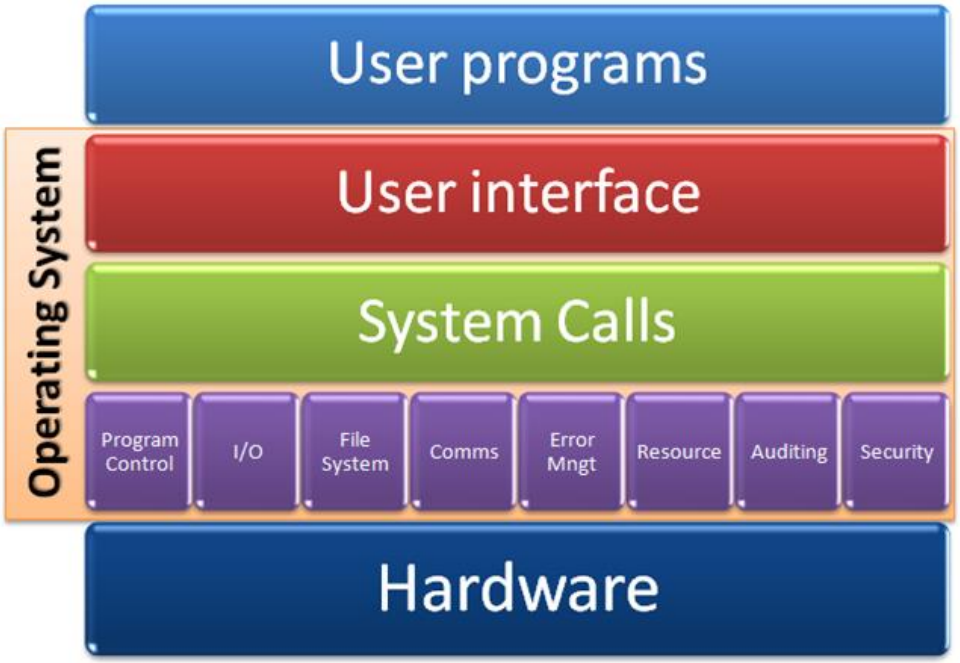


- As operações de uma Pilha são mais **restritas** do que as de uma Lista. Por exemplo, você pode adicionar ou remover um elemento em qualquer posição de uma Lista mas em uma Pilha você só pode adicionar ou remover do topo.

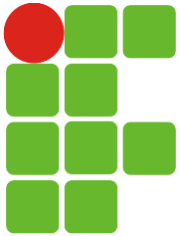
Lista é uma estrutura mais poderosa e mais genérica do que uma Pilha. A Pilha possui apenas um **subconjunto de operações** da Lista.

# Pilha

- Exemplos:
  - um PC moderno usa pilhas ao nível de arquitetura.
  - Design básico de um sistema operacional para manipular interrupções e **chamadas** de função do sistema operacional.
  - Executar uma Máquina virtual java e a própria linguagem Java possui uma classe denominada "Stack".



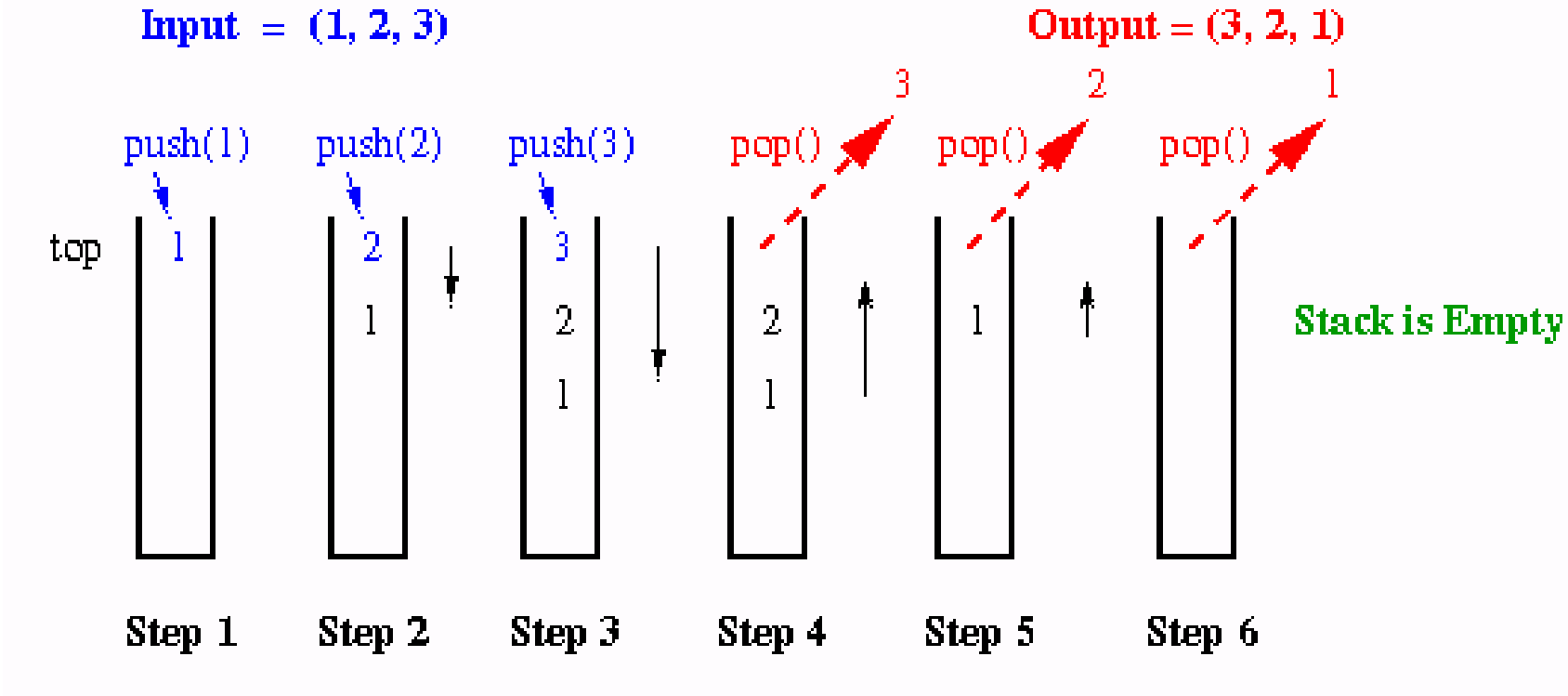
# Pilha



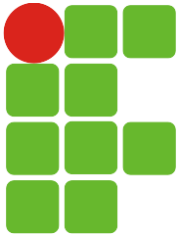
- Existem 2 operações básicas que devem ser implementadas numa estrutura pilha:
  - Operação empilhar (inserir elemento no topo)
  - Operação desempilhar (remover do topo)

- É comum nos referirmos a essas duas operações pelos termos em inglês:
  - push => empilhar
  - pop => desempilhar

# Pilha


Input = (1, 2, 3)
Output = (3, 2, 1)
push(1)
push(2)
push(3)
pop()
pop()
pop()
3
2
1
top
1
2
1
3
2
1
2
1
1
Stack is Empty
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6

# Pilha

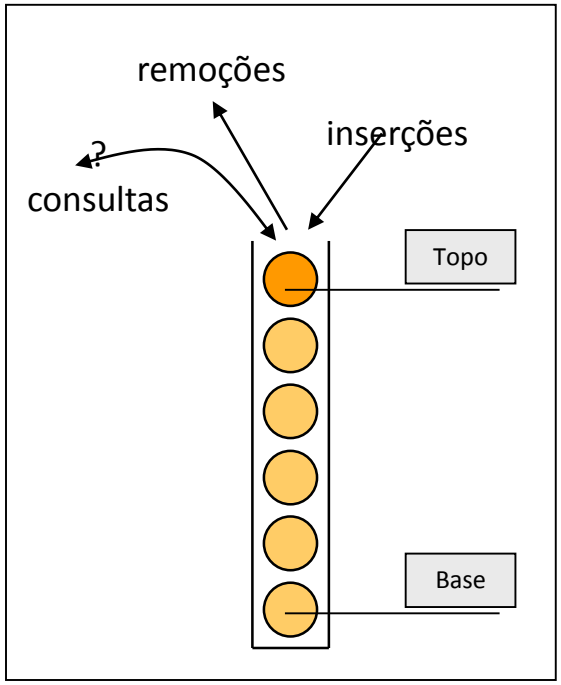


- Vamos considerar a implementação de 5 operações:
  - Cria uma estrutura de pilha.(inicializa)
  - Insere elemento no topo (push).
  - Remove elemento do topo (pop).
  - Verifica se a pilha está vazia.
  - Libera estrutura pilha.

# Pilha

- O Arquivo pilha.h que representa a interface do tipo, pode conter o seguinte código:

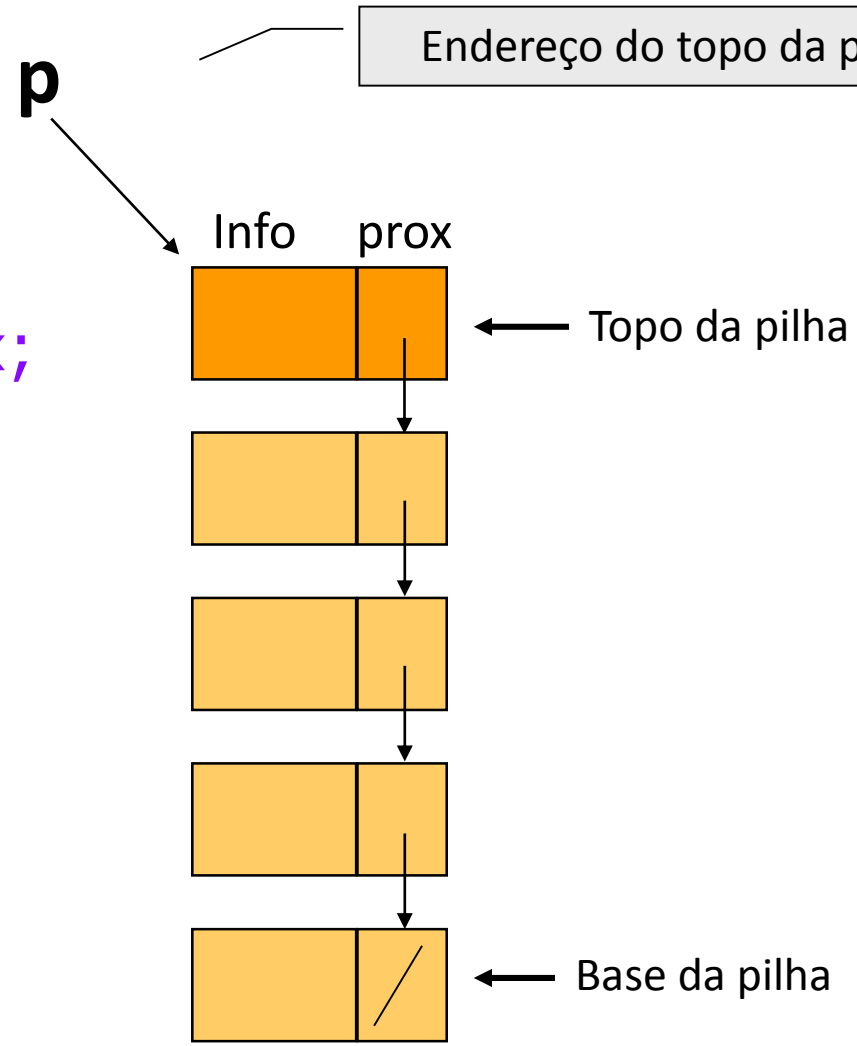
```
Typedef struct pilha Pilha;

Pilha* cria(void);
void push(Pilha* p, float v);
float pop(Pilha* p);
int vazia(Pilha* p );
void libera(Pilha* p);
```



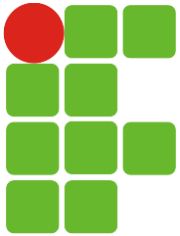
# Pilha - ESTRUTURA

- Estrutura:

```
struct no{
        float info;
        struct no* prox;
};
struct pilha{
        No* prim;
}
```



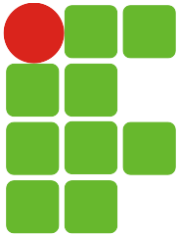
# Pilha



- Função criar:

```
Pilha* cria(void) {
        Pilha* p = (Pilha*) malloc (sizeof(Pilha));
        p->prim = NULL;
        return p;
}
```

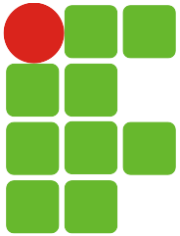
# Pilha



- Função auxiliar insere e retira no início :

```
No* ins_ini(No* l, float v) {
        No* p = (No*) malloc(sizeof(No));
        p->info = v;
        p->prox = l;
        retur p;
}

No* ret_ini(No* l) {
        No* p = l->prox;
        free(l);
        return p;
}
```

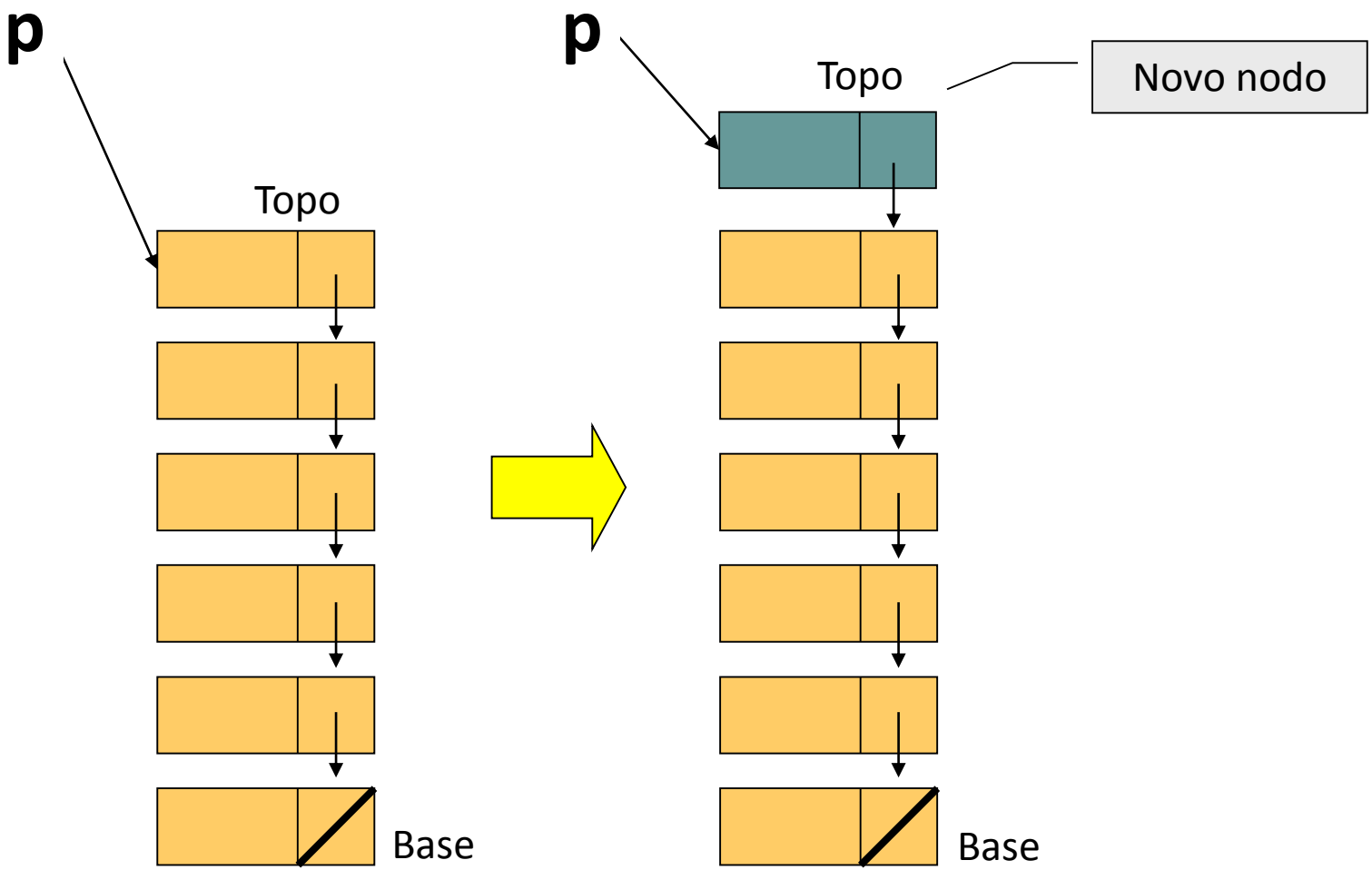
# Pilha



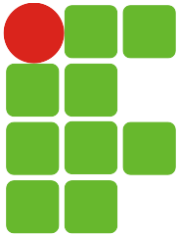
- As funções que manipulam a pilha fazem uso das funções auxiliares:

```
void push(Pilha* p, float v){
        p->prim = ins_ini(p->prim,v);
}

float pop(Pilha* p){
        float v;
        if (vazia(p)){
                printf("Pilha vazia \n");
                exit(1); //aborta o programa
        }
        v = p->prim->info;
        p->prim = ret_ini(p->prim);
        return v;
}
```

# Pilha



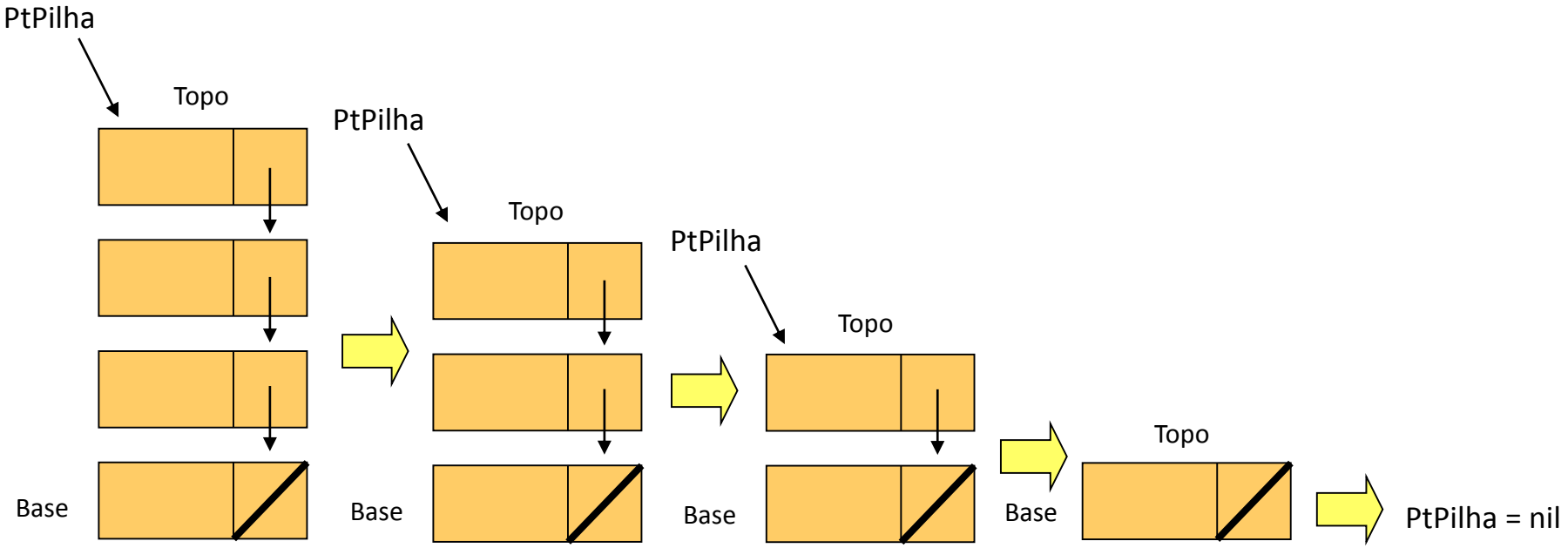
# Pilha



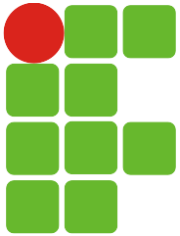
```
void libera(Pilha* p){
        No* n = p->prim;
        No* temp;
        while(n!=NULL){
                temp = n->prox;
                free(n);
                n = temp;
        }
        free(p);
}
```

# Pilha

- Operação: Liberar uma pilha

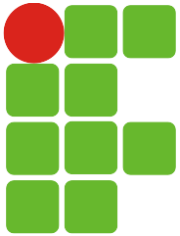


# Pilha – Exercício 1



- Faça as seguintes funções:
  - Verificar se está vazia .
  - Imprimir na ordem do topo para a base.

# Pilha



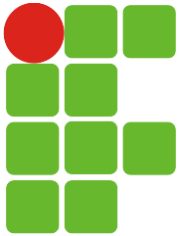
Desde pequenos aprendemos a escrever expressões aritméticas em que as operações com dois argumentos são escritas na seguinte ordem: o primeiro argumento, o símbolo de operação, o segundo argumento. Esta maneira de escrever expressões é denominada notação infixa. Um dos problemas que ela apresenta é a necessidade de regras de prioridade e de parênteses para indicar exatamente as operações. Por exemplo, as expressões *infixas*:

**5 + 7 * 3  e  ( 5 + 7 ) * 3**

representam valores distintos (26 e 36). A fim de evitar o uso de parênteses, existe uma outra notação, denominada *pós-fixa*, em a ordem é: o primeiro argumento, o segundo argumento, o símbolo de operação. Por exemplo, as expressões acima seriam escritas nesta notação como:

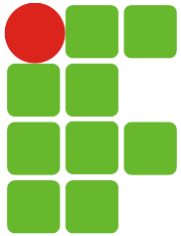
**5 7 3 * +  e  5 7 + 3 ***

# Pilha



- Um bom exemplo de aplicação de pilha é o funcionamento das calculadoras HP. Elas trabalham com expressões pós-fixadas, então para avaliarmos uma expressão como (1-2) * (4 +5), podemos digitar: 1 2 – 4 5 + *
- Cada operando é empilhada numa pilha de valores. Quando se encontra um operador, desempilha-se o número apropriado de operandos (dois p/ operadores binários e um p/ operadores unários). Realiza-se a operação devida e empilha-se o resultado. Deste modo, na expressão acima são empilhados os valores 1 e 2. Quando aparece o operador -, 1 e 2 são desempilhados e o resultado da operação, no caso -1 (=1-2), é colocado no topo da pilha. A seguir, 4 e 5 são empilhados. Operador seguinte, +, desempilha 4 e 5 e empilha o resultado da soma, 9. Nesta hora, estão na pilha dois resultados parciais, -1 na base e 9 no topo. O operador *, então desempilha os dois e coloca -9 (=-1 * 9) no topo da pilha.

# Pilha

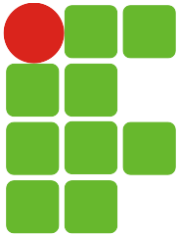


- Como exemplo de aplicação de uma estrutura pilha, vamos implementar uma calculadora pós-fixada.

**calc.h**

```
typedef struct calc Calc
Calc* cria_calc(char* f);
void operando(Calc* c , float v);
void operador(Calc* c, char op);
void libera_calc(calc* c);
```

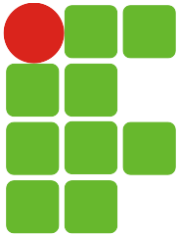
# Pilha



- Implementação:

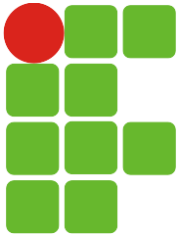
```
struct calc{
        char f[21]; //formato para impressão
        Pilha* p; //pilha de operandos
}

Calc* cria_calc(char* formato){
        Calc* c = (Calc*) malloc(sizeof(Calc));
        strcpy(c->f, formato);
        c->p=cria(); //cria pilha vazia
        return c;
}
```

# Pilha – Programa Principal



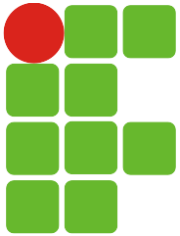
```
int main()
{
        Calc* calculadora;
        char op;
        float valor;
        printf("Calculadora pos-fixada\n\n");
        printf("Digite 's' para sair.\n");
        printf("Digite 'r' para raiz quadrada.\n");
        printf("Digite '^' para exponenciacao.\n\n");
        // Inicializa a calculadora
        calculadora = cria_calc("%.2f\n");
        // Loop para utilizar a calculadora
        do
        {
                scanf(" %c", &op);
```

# Pilha



```
                if(op=='+' || op=='-' || op=='*' || op=='/' || op=='^' || op=='r')
                {
                        operador(calculadora, op);
                }
                else
                {
                        ungetc(op, stdin);
                        if(scanf("%f", &valor)==1)
                        {
                                operando(calculadora, valor);
                        }
                }
        }
        while(op!='s');
        // Libera calculadora
        libera_calc(calculadora);

        return 0;
}
```

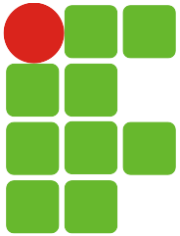
# Pilha



- Implementação:

```
void operando(Calc* c, float v){

        //empilha operando
        push(c->p,v);

        //empilha topo da pilha
        printf(c->f,v);
}
```

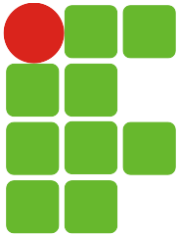
# Pilha



- Implementação:

```
void operador(Calc* c, char op){
        float v1,v2,v;
        if (vazia(c->p))    //desempilha
                v2=0;
        else
                v2=pop(c->p);
        if (vazia(c->p))
                v1=0;
        else
                v1=pop(c->p);
        switch(op) { //faz operação
                case `+ `: v= v1+v2; break;
                case `- `: v= v1 – v2; break;
                case `*`: v = v1 * v2; break;
                case `/ `: v = v1/v2; break;
        }
        push (c->p,v); //empilha resultado
        printf(c->f,v); //imprime resultado
}
```

# Pilha – Exercício 2



- **Faça a implementação do método Libera calculadora de memória .**

- **Faça o seguinte teste no programa principal:**

**Digite : 4 5 8 * +**

**Digite: 7/**

**Resultado: 6.28**