

ESTRUTURA DE DADOS I

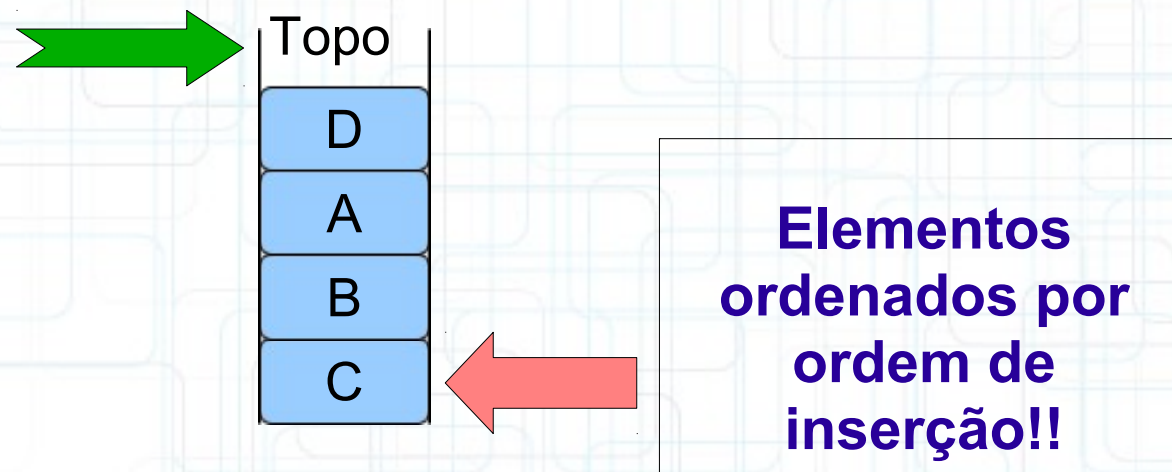
Aula 10 – Pilhas

Profa. Me. Carmen Dalla Rosa Bittencourt
bittencourt.carmen@gmail.com

Pilhas

Definição:

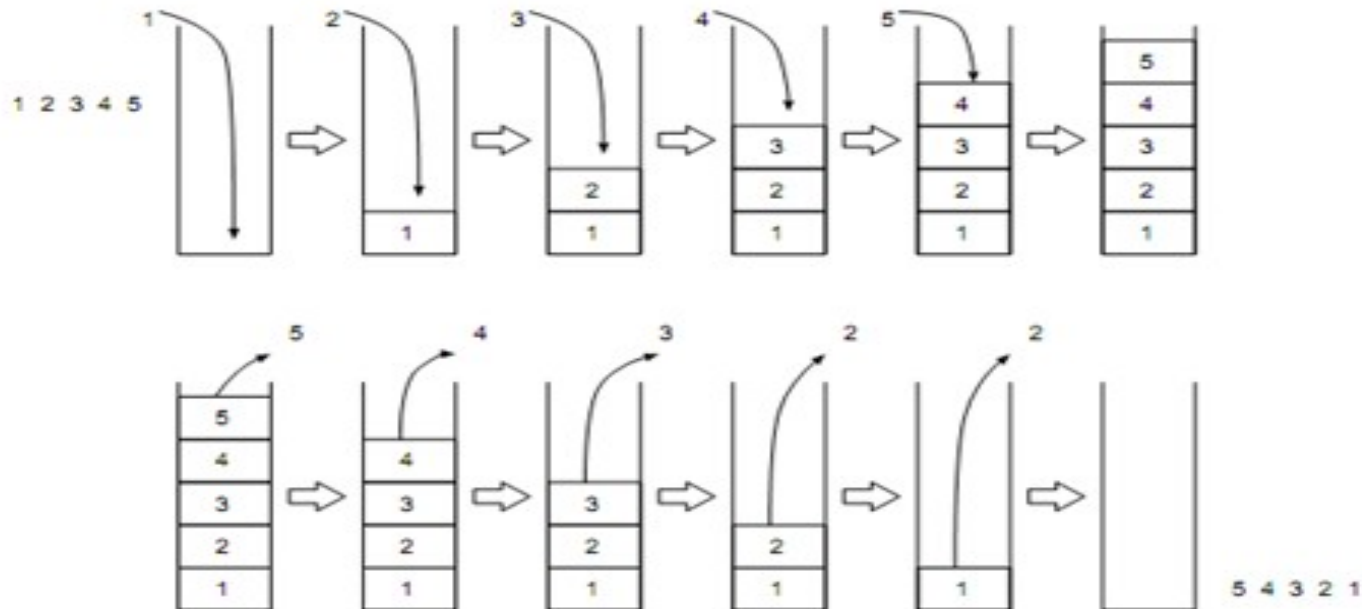
- É um conjunto ordenado de itens na qual todas as inserções e retiradas são feitas em uma das extremidades denominada **Topo**.
- A ideia fundamental da pilha é que todo o acesso a seus elementos é feito através do seu topo (um único ponto de acesso).



Pilhas

- O último elemento inserido é o primeiro a ser retirado, ou seja, os elementos são removidos na ordem inversa da sua inserção.
- Em outras palavras, o primeiro elemento a ser inserido na pilha é o último a ser removido. Essa política é conhecida pela sigla LIFO (*Last In First Out*). Em português Último a Entrar Primeiro a Sair.

Operações com pilhas



Pilhas

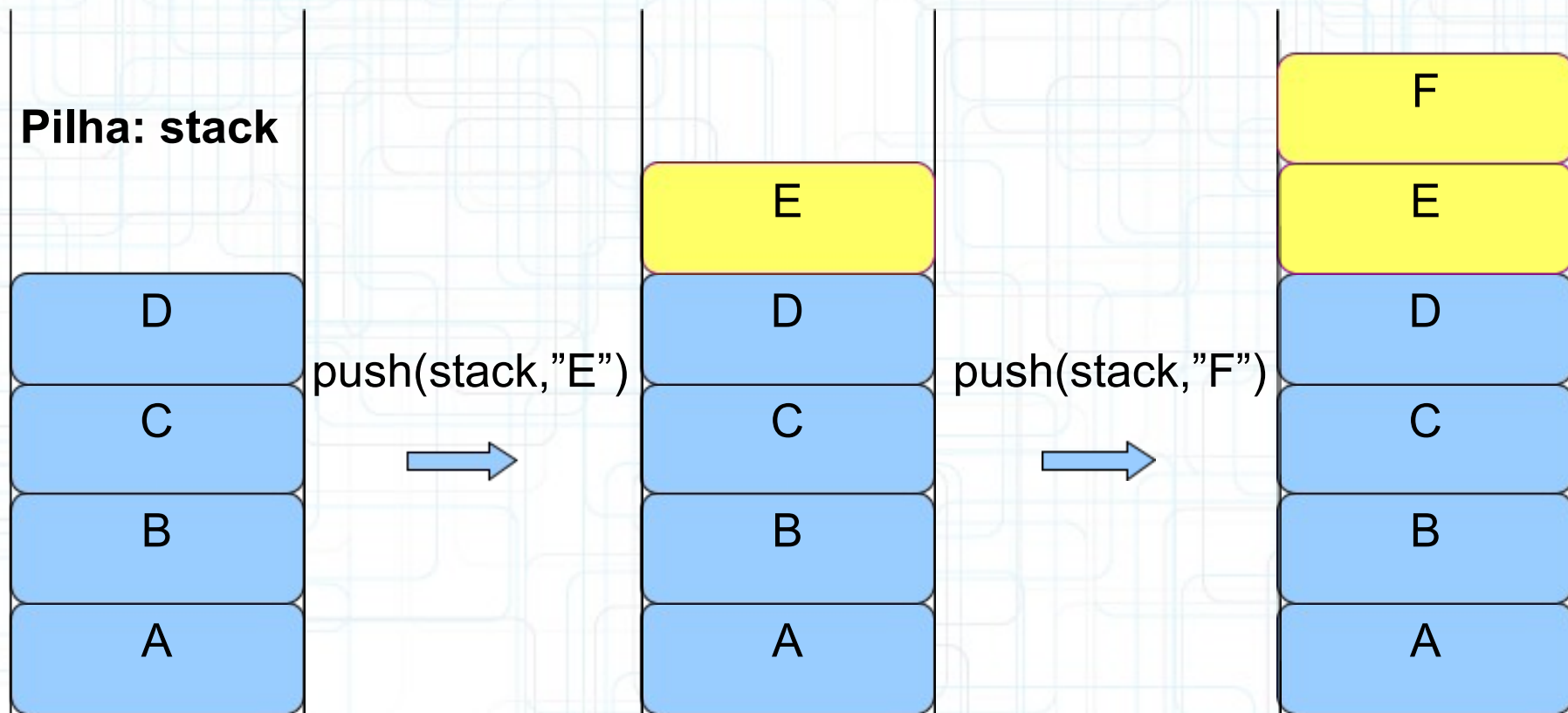
Operações:

- a) **Empilhar** (*push*)
 - Função que inclui um item na pilha (no topo).
- b) **Desempilhar** (*pop*)
 - Função que remove o elemento do topo da pilha.
- c) **TopoPilha** (*stacktop*)
 - Função que retorna o elemento corrente do topo da pilha.
- d) **PilhaVazia** (*empty*)
 - Função que retorna se a pilha está vazia.
 - Retorna *true(1)* se a pilha estiver vazia
 - Retorna *false(0)* se a pilha **não** estiver vazia.
- e) **Inicializar** (*init*)
 - Inicializa a pilha como vazia.

Pilhas

Operações: Empilhar: função *push(pilha,valor)*

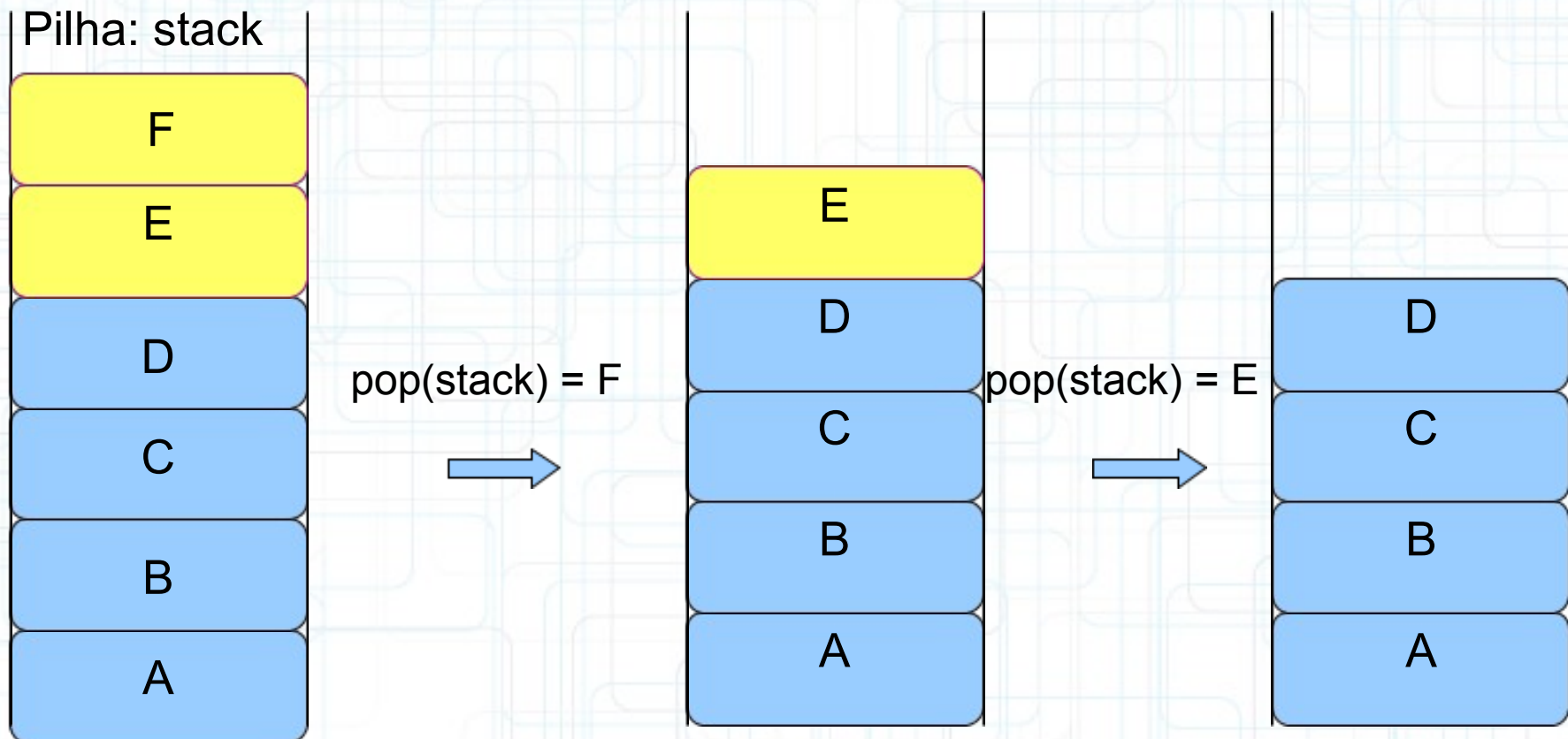
- Chamar a função (push) passando como parâmetros a variável pilha e o valor a empilhar. No exemplo abaixo a variável pilha é **stack** e os valores empilhados são **"E"** e **"F"**.



Pilhas

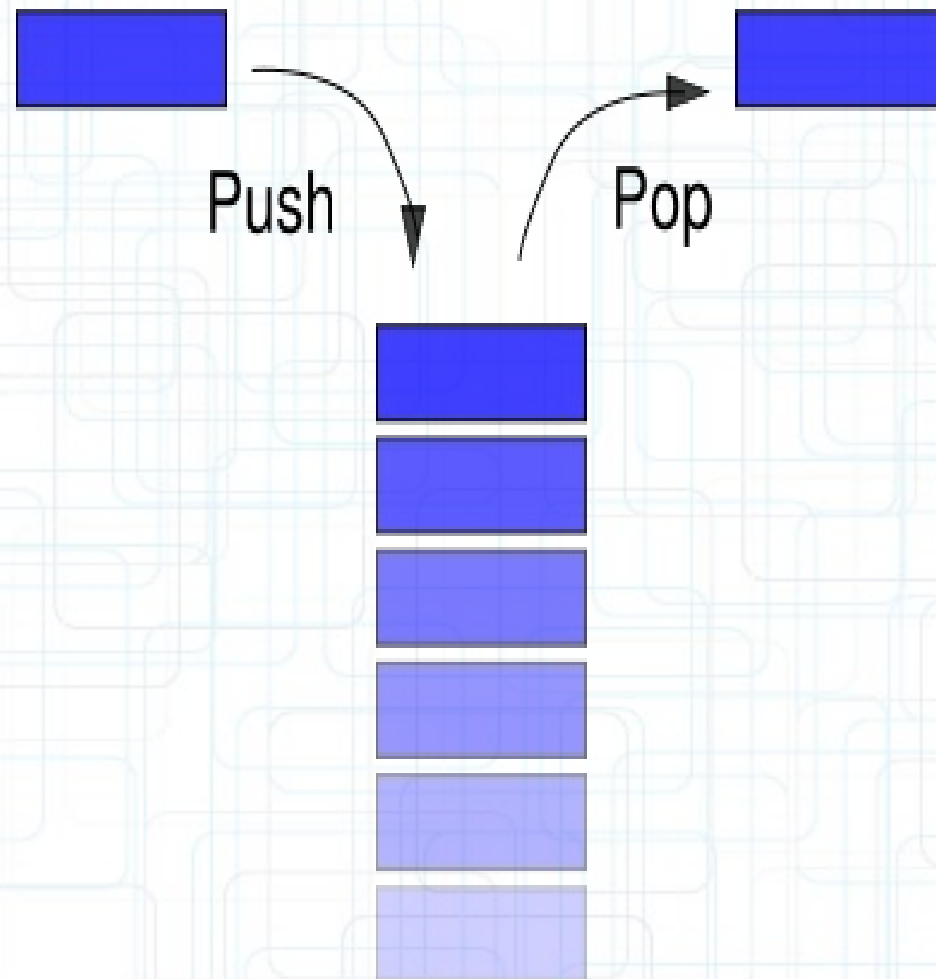
Operações: Desempilhar: função *pop(pilha)*

- Chamar a função (pop) passando como parâmetro a variável pilha. A função retornará o valor desempilhado. No exemplo abaixo a variável pilha é **stack** e o retorno da chamada da função é "**F**" e posteriormente "**E**".



Pilhas

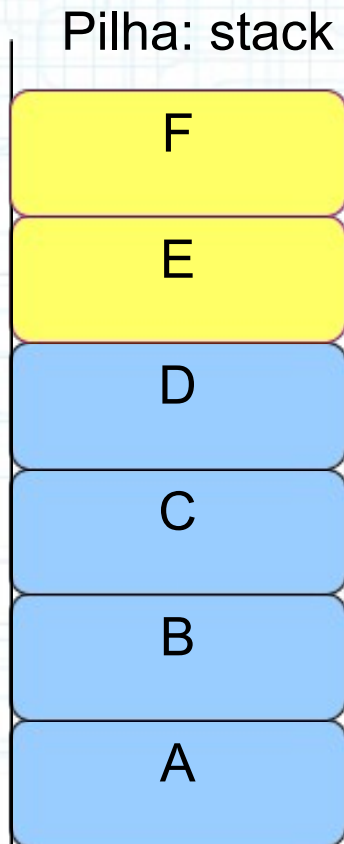
Empilhar (push) x Desempilhar (pop)



Pilhas

Operações: TopoPilha: função *stacktop(pilha)*

- Chamar a função (*stacktop*) passando como parâmetro a variável pilha. A função retornará o valor corrente do topo da pilha. No exemplo abaixo o valor retornado pela função é "F".

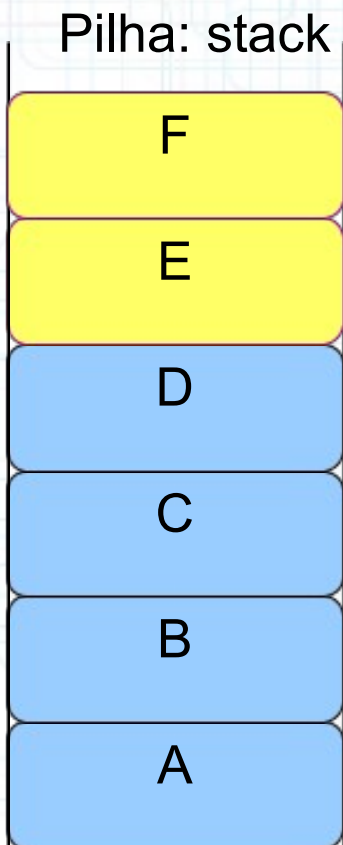


`stacktop(stack)` → F

Pilhas

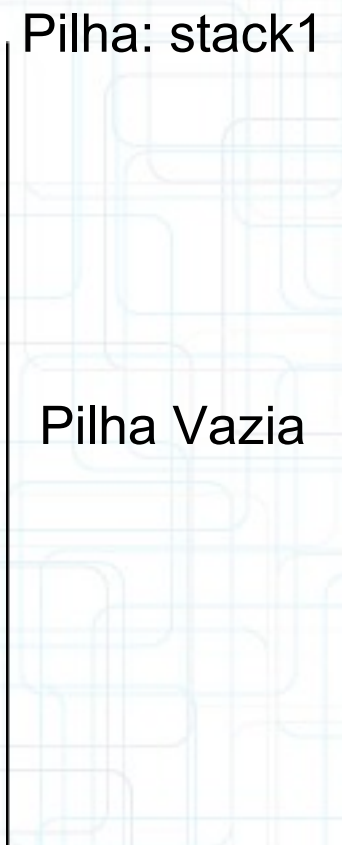
Operações: Pilha Vazia: função *empty(pilha)*

- Chamar a função (*empty*) passando como parâmetro a variável pilha. A função retornará 1(*true*) se a pilha estiver vazia e 0(*false*) se existir pelo menos um elemento.



`empty(stack)` → false

false = há 6 elementos na pilha. Se houver pelo menos um, o resultado é false.



`empty(stack1)` → true

true = não há nenhum elemento na pilha.

Pilhas

Tipos:

- a) **Estática:**
 - Tamanho máximo fixo e constante na implementação.
 - Utiliza-se vetor para implementar.
 - Utilizada quando se sabe previamente o número máximo de elementos que se deseja armazenar.
- b) **Dinâmica:**
 - Cresce indefinidamente.
 - Utiliza-se ponteiros para implementar.
 - Utilizada quando o número máximo de elementos é desconhecido (limitado a quantidade de memória disponível).

Pilha Dinâmica

- Implementação em C:
 - **Pilha Dinâmica**: Estruturas básicas utilizadas

```
struct ELEMENTO_PILHA{  
    int info;  
    struct ELEMENTO_PILHA *prox;  
};  
  
struct PILHA {  
    struct ELEMENTO_PILHA *topo;  
};
```

- Não há constante para informar o tamanho máximo da pilha, ela cresce conforme a necessidade e a disponibilidade de memória.
- A variável **topo** é um ponteiro para o elemento que está no topo da pilha.

Pilha Dinâmica

- Implementação em C:
 - **Pilha Dinâmica:**
 - Para declararmos um variável do tipo pilha utilizaremos o código: **struct PILHA p1;**
 - Com isso criamos um variável denominada de **p1** em que seu tipo é **PILHA**.
 - **p1** possui um ponteiro para um estrutura do tipo ELEMENTO_PILHA. Esse ponteiro irá apontar para o elemento do topo.
 - A estrutura ELEMENTO_PILHA armazenará os dados dos elementos da pilha.
 - A variável **info** armazena o dado da pilha.
 - O ponteiro **prox** irá apontar para o próximo elemento dentro da pilha.

Pilha Dinâmica

- Implementação em C:
 - **Pilha Dinâmica**:
 - O ponteiro **prox** permitirá que se “navegue” entre os elementos da pilha.
 - Os elementos da pilha serão criados dinamicamente (funções malloc(), sizeof() e free()).

Pilha Dinâmica

- Implementação em C das Operações básicas:
 - **Inicializar: função `init(pilha)`**
 - A função recebe a referência (endereço) da estrutura;
 - O ponteiro para o topo deverá apontar para NULL.
 - Deve ser a primeira função a ser chamada antes de se utilizar a pilha criada.

```
void init(PILHA *p1) {  
    p1->topo = NULL;  
}
```

Utilização na função chamadora (main): `init(&p1);`

Pilha Dinâmica

- **PilhaVazia: função `empty(pilha)`**
 - A variável topo representa a posição do elemento superior da pilha.
 - Conforme descrito na função init, em que a pilha é criada vazia, uma pilha não terá elementos quando topo apontar para NULL.

```
int empty (PILHA *p1) {  
    if (p1->topo == NULL){  
        return 1;                // Pilha vazia  
    }else {  
        return 0;                // Pilha NÃO vazia  
    }  
}
```

Utilização na função chamadora (main):

```
if (empty(&p1)==1){  
    printf("a pilha está vazia");  
}else{  
    printf("a pilha não está vazia");  
}
```

Pilha Dinâmica

- **Empilhar: função push(pilha,valor)**

- A cada inclusão a variável topo deve apontar para esse novo elemento.
- Os elementos serão armazenados em novos elemento do tipo estrutura ELEMENTO_PILHA.

```
void push(PILHA *ps, int elemento) {  
    struct ELEMENTO_PILHA *p;  
    p=(struct ELEMENTO_PILHA*) malloc (sizeof(struct ELEMENTO_PILHA));  
    p->info = elemento;  
    p->prox = ps->topo;  
    ps->topo = p;  
}
```

Utilização (main): push(&p1,4); // Empilhando o valor 4.
p=(struct ELEMENTO_PILHA*)malloc(sizeof(struct ELEMENTO_PILHA));
Conversão para ponteiro de struct ELEMENTO_PILHA
retornado pela Alocação de memória do Tamanho
de um struct ELEMENTO_PILHA

Função malloc()

- malloc(size) aloca dinamicamente uma parte da memória, de tamanho size, e retorna um ponteiro para um item de tipo char.

Ex:

```
int *p;  
p = (int *) malloc (sizeof(int));
```

Esses comandos criam dinamicamente a variável inteira *p. Malloc cria um objeto com o tamanho de sizeof(int), ou seja, aloca armazenamento para um inteiro.

Malloc retorna também um ponteiro para o armazenamento que ele aloca. Para forçar esse ponteiro apontar para um inteiro, usamos o operador de conversão (int *).

Função malloc()

- Ex:

```
int *p, *q, x;  
p = (int *) malloc (sizeof(int));  
*p = 3;  
q = p;  
printf("%d %d \n", *p, *q);  
x = 7;  
*q = x;  
printf("%d %d \n", *p, *q);  
p = (int *) malloc (sizeof(int));  
*p = 5;  
printf("%d %d \n", *p, *q);
```

Função malloc()

- Obs:

Se malloc() for chamada duas vezes sucessivas e seu valor for atribuído à mesma variável, como:

```
p = (int *) malloc (sizeof(int));  
*p = 3;  
p = (int *) malloc (sizeof(int));  
*p = 7;
```

a primeira cópia de *p é perdida porque seu endereço não foi salvo.

O espaço alocado para variáveis dinâmicas só pode ser acessado por meio de um ponteiro. A menos que o ponteiro para a primeira variável seja salvo em outro ponteiro, essa variável será perdida.

Pilha Dinâmica

- Implementação em C:
 - **Desempilhar: função valor = pop(pilha)**
 - Remove o elemento do topo da pilha. E retorna esse elemento para o programa de chamada.
 - A função deve evitar o stack underflow. Ele acontece quando se tenta desempilhar uma pilha vazia.

```
int pop(PILHA *ps) {  
    int valorTopo;  
    struct ELEMENTO_PILHA *aux;  
    if (empty(ps)) {  
        printf("\n stack underflow ! \n");  
        exit(1);  
    } else {  
        valorTopo = ps->topo->info;  
        aux = ps->topo;  
        ps->topo = ps->topo->prox;  
        free(aux);  
        return valorTopo;  
    }  
}
```

Utilização (main): printf("Elemento desempilhado =%d",pop(&p1));

Função free()

- free() é usada para liberar o armazenamento de um variável alocada dinamicamente.

Ex:

```
int *p;  
p = (int *) malloc (sizeof(int));  
free(p);
```

Após o comando free(p), quaisquer referências futuras a *p é invalidada (a menos, que um novo valor seja atribuído a p por um comando de atribuição ou por uma nova chamada a malloc).

Função free()

- Ex:

```
int *p, *q;  
p = (int *) malloc (sizeof(int));  
*p = 5;  
q = (int *) malloc (sizeof(int));  
*q = 8;  
free(p);  
p = q;  
printf("%d %d \n", *p, *q);  
x = 7;  
*q = x;  
q = (int *) malloc (sizeof(int));  
*q = 6;  
printf("%d %d \n", *p, *q);
```


Pilha Dinâmica

- Implementação em C:
 - **TopoPilha: função valor = stacktop(pilha)**
 - Retorna o elemento do topo da pilha.
 - A função deve verificar se existe algum elemento na pilha.
 - A pilha não é alterada. Somente retorna o elemento do topo.

```
int stacktop(PILHA *ps) {  
    if (empty(ps)) {  
        printf("\n underflow ! \n");  
        exit(1);  
    } else {  
        return ps->topo->info;  
    }  
}
```

Utilização (main): printf("Topo da pilha = %d", stacktop(&s1));

Pilha Dinâmica - Aplicações

EXEMPLO 1

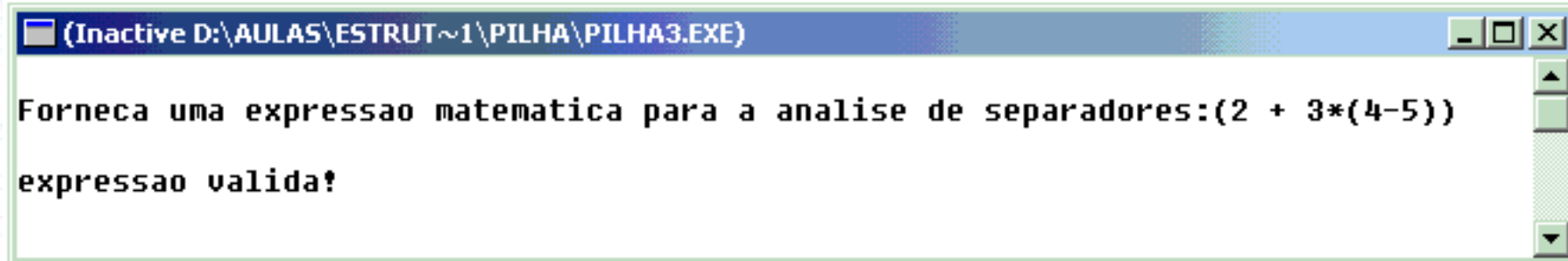
Deseja-se desenvolver um programa que analise uma expressão matemática e identifique se os elementos separadores de abertura '[', '{' e '(' são encerrados de forma correta com os elementos separadores de encerramento ')', '}' e ']'.

Por simplicidade, o programa não verifica a ordem de emprego desses elementos de abertura. Ou seja, expressões tais como $2 * (3 - [4 + \{ 2 + 3 \}])$ e $2 * \{ 3 - (4 + [2 + 3]) \}$ são consideradas válidas.

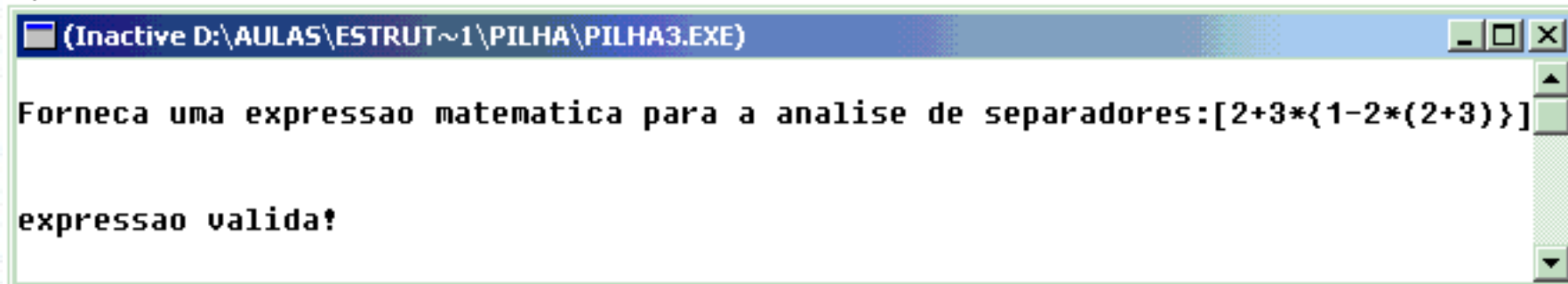
Por outro lado, expressões tais como $2 * (3 - [4 + 5])$ são consideradas inválidas, pois o último elemento aberto '[', posicionado antes do número 4, está sendo encerrado com o ')', posicionado após o número 5.

Pilha Dinâmica - Aplicações

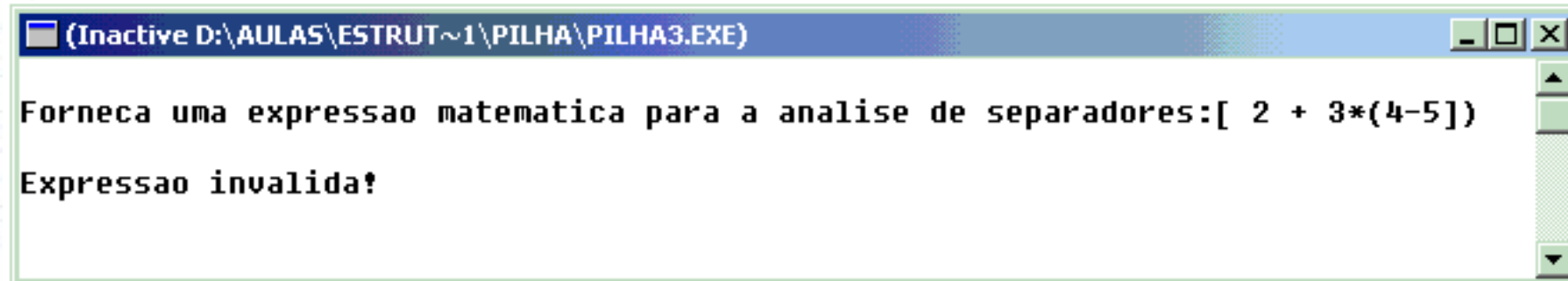
a)



b)



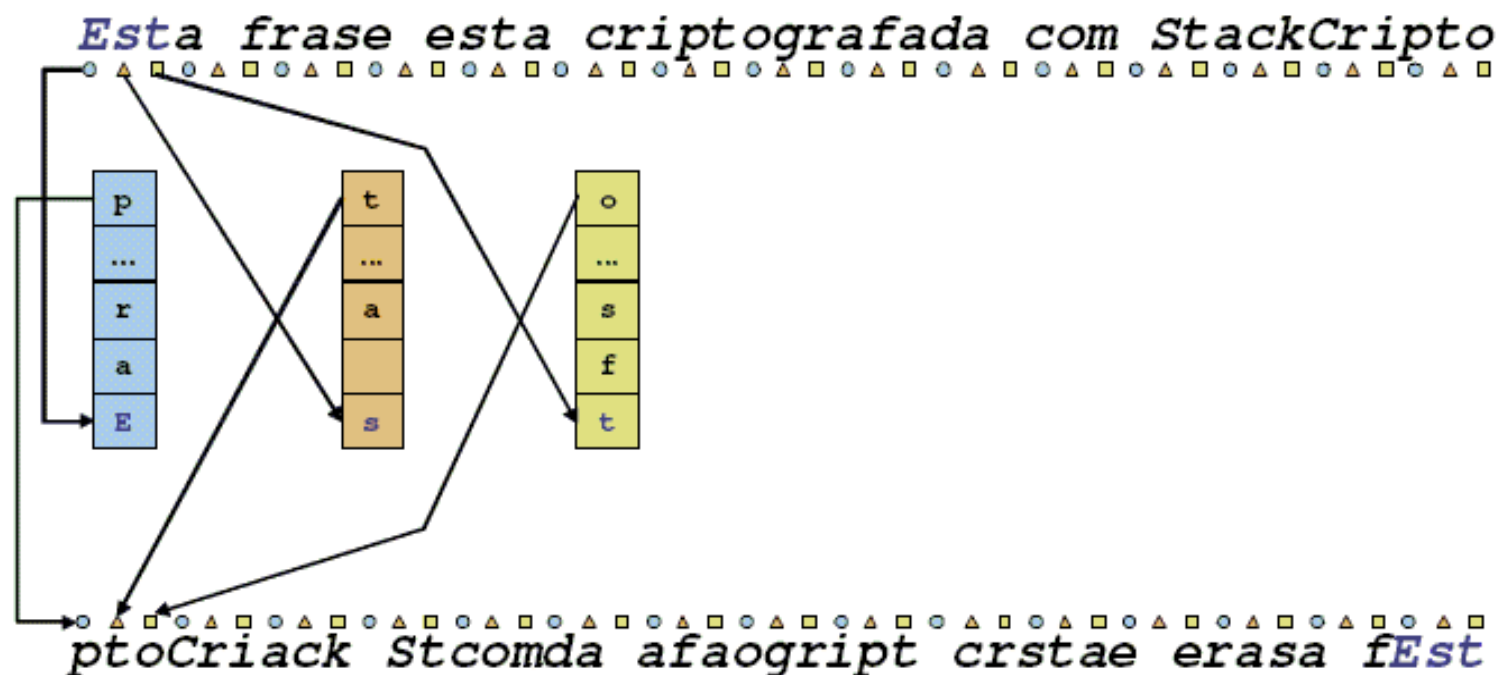
c)



Pilha Dinâmica - Aplicações

EXEMPLO 2

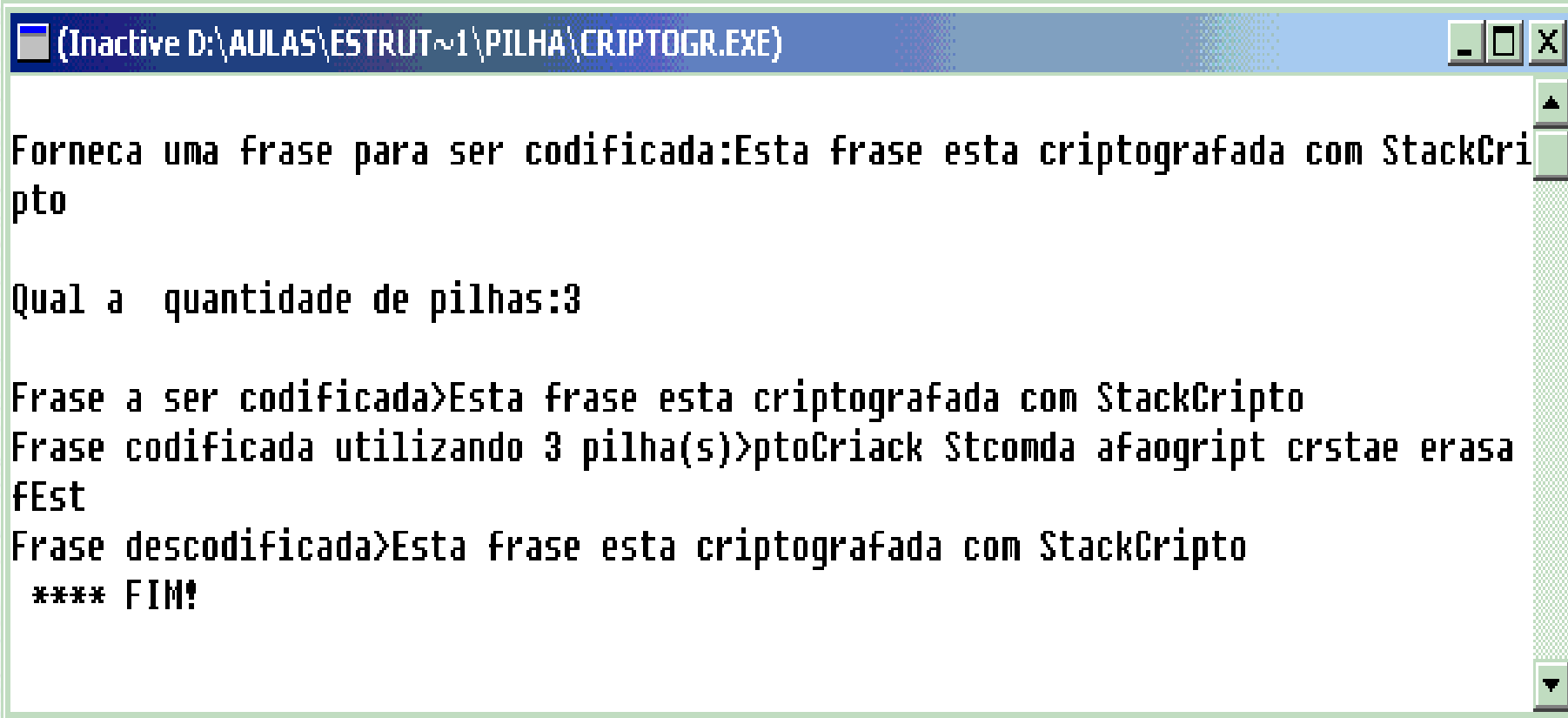
A próxima tarefa é criar uma classe que use pilhas para fazer criptografia, chama-se *StackCripto*. Esta classe possui como atributos apenas um parâmetro de quantas pilhas serão utilizadas para fazer a criptografia (construtor). Como métodos, ela possui o `cript(frase)` que retorna um array de caracteres cifrados e o método `decrypt(frase)`, que retorna um array de caracteres com a frase descryptografada. O processo de criptografia é simples, basta dispor os caracteres em pilhas e depois juntá-los novamente (6 pontos). Exemplo (repare que os métodos `cript` e `decrypt` são o mesmo algoritmo):



Desenvolver uma “classe” principal (arquivo com o `main`), que solicite ao usuário a digitação de uma frase e mostre a mesma criptografada e depois descryptografada (1 pontos), utilizando um objeto *StackCripto* e os seus métodos `cript` e `decrypt`.

Pilha Dinâmica - Aplicações

EXEMPLO 2



```
(Inactive D:\AULAS\ESTRUT~1\PILHA\CRIPTOGR.EXE)

Forneça uma frase para ser codificada:Esta frase esta criptografada com StackCri
pto

Qual a quantidade de pilhas:3

Frase a ser codificada>Esta frase esta criptografada com StackCripto
Frase codificada utilizando 3 pilha(s)>ptoCriack Stcomda afaogript crstae erasa
fEst
Frase decodificada>Esta frase esta criptografada com StackCripto
**** FIM!
```


Pilha Dinâmica - Aplicações

EXEMPLO 3 - Calculadora pós-fixada

Um bom exemplo de aplicação de pilha é o funcionamento das calculadoras da HP (Hewlett-Packard). Elas trabalham com expressões pós-fixadas, então para avaliarmos uma expressão como $(1-2)*(4+5)$ podemos digitar $1\ 2\ -\ 4\ 5\ +\ *$. O funcionamento dessas calculadoras é muito simples. Cada operando é empilhado numa pilha de valores. Quando se encontra um operador, desempilha-se o número apropriado de operandos (dois para operadores binários e um para 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. O operador seguinte, +, desempilha o 4 e o 5 e empilha o resultado da soma, 9. Nesta hora, estão na pilha os 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 Dinâmica - Exercícios

1) Considerando os conceitos relacionados a estruturas do tipo pilha, indique a situação das estruturas após a execução da seguinte sequência de funções:

```
init(stack1);  
init(stack2);  
push(stack1,"A");  
push(stack2,"B");  
push(stack1,"C");  
push(stack2,"D");  
push(stack2,"E");  
stacktop(stack1);  
pop(stack2);
```

stack1

stack2

Atividade complementar

- Ler na apostila de apoio (C_UFMG.pdf) as explicações sobre as funções :
 - malloc(),
 - sizeof(),
 - free() e
 - o operador de conversão.