



CENTRO DE CIÊNCIA E TECNOLOGIA
LABORATÓRIO DE CIÊNCIAS MATEMÁTICAS
UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE

Pilhas

Representação como TAD

Disciplina: Estrutura de Dados I

Prof. Fermín Alfredo Tang Montané

Curso: Ciência da Computação

TAD Pilha (Stack ADT)

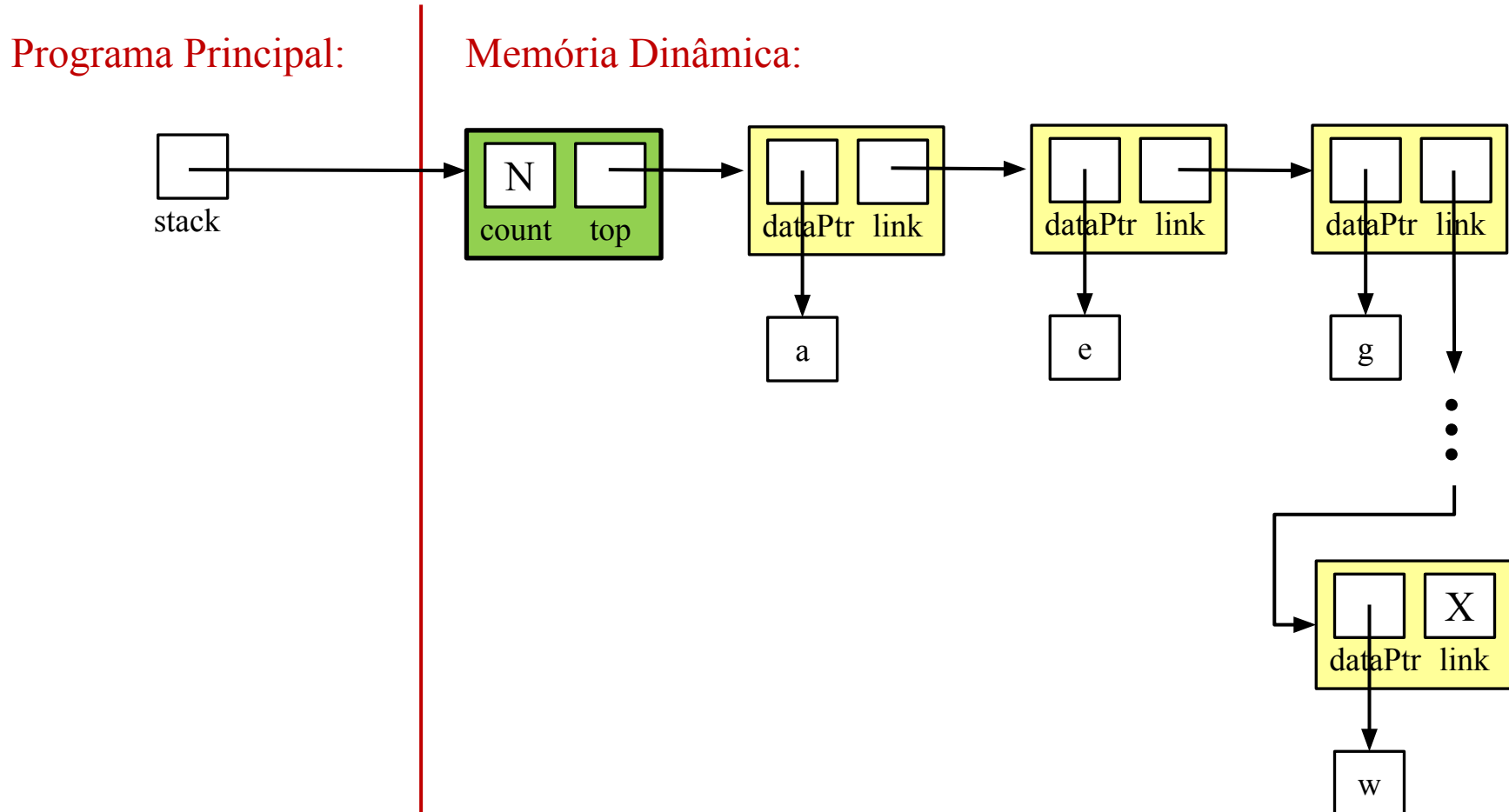
Estrutura

- A implementação de pilhas como TAD visa garantir que a implementação da pilha seja independente de sua aplicação, permitindo a reutilização do código do TAD sem qualquer modificação.
- Já o código da aplicação que requer a estrutura da dados pilha, pode precisar de adaptação para se adequar ao tipo de dado utilizado.
- Na implementação da TAD em vez de se armazenar um dado em cada nó da estrutura, armazena-se o ponteiro a esse dado.
- O programa de aplicação terá a responsabilidade de alocar memória para o dado e de passar o seu endereço ao TAD pilha.

TAD Pilha (Stack ADT)

Conceitos Estruturais

- A estrutura do TAD Pilha é mostrada na figura.



TAD Pilha (Stack ADT)

Estrutura da Pilha

- A definição do tipo para a pilha é mostrada na figura:

P3-06.h

```
1 // Stack ADT Type Definitions
2 typedef struct node
3 {
4     void*      dataPtr;
5     struct node* link;
6 } STACK_NODE;
7
8 typedef struct
9 {
10     int      count;
11     STACK_NODE* top;
12 } STACK;
```

// Definição do tipo nó da pilha (STACK_NODE)

// Ponteiro genérico

// Ponteiro de ligação

// Definição do tipo (cabeçalho) pilha (STACK)

// Contador de elementos

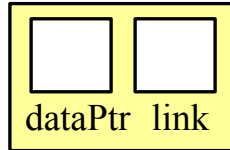
// Ponteiro ao primeiro nó da pilha

- Estas definições devem ser incluídas em um arquivo cabeçalho (header file) de maneira que qualquer função que precise definir uma pilha possa fazê-lo facilmente.

TAD Pilha (Stack ADT)

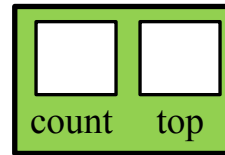
Estrutura da Pilha - Figura

STACK_NODE



```
typedef struct node
{
    void*      dataPtr;
    struct node* link;
} STACK_NODE;
```

STACK



```
typedef struct
{
    int      count;
    STACK_NODE* top;
} STACK;
```

- A estrutura do nó consiste em:
 - i) um ponteiro genérico (ao dado);
 - ii) um ponteiro de ligação com outro nó.
- A estrutura do cabeçalho consiste em:
 - i) um contador do número de elementos presentes na pilha;
 - ii) um ponteiro ao topo da pilha.

TAD Pilha (Stack ADT)

Operações de suporte do TAD Pilha

- Devemos ter operações de suporte ao funcionamento da Pilha.
- Serão apresentadas as seguintes operações:
 - Criar Pilha (Create Stack);
 - Inserir Pilha (Push Stack);
 - Remover Pilha (Pop Stack);
 - Topo Pilha (Stack Top);
 - Pilha Vazia (Empty Stack);
 - Pilha Cheia (Full Stack);
 - Contador Pilha (Stack Count);
 - Destruir Pilha (Destroy Stack).
- Essas operações serão implementadas em C.

TAD Pilha (Stack ADT)

Operações de suporte do TAD Pilha

- O protótipo destas funções é inicialmente declarado em um arquivo de interface. `stacksADT.h`

```
1 // Header file for stack ADT.
2
3 #include <stdlib.h>
4 #include <stdbool.h>
5
6 #include "P3-06.h" /* Stack ADT Definitions */
7
8 /* ADT Prototype Declarations */
9 STACK* createStack (void);
10 bool pushStack (STACK* stack, void* dataInPtr);
11 void* popStack (STACK* stack);
12 void* stackTop (STACK* stack);
13 bool emptyStack (STACK* stack);
14 bool fullStack (STACK* stack);
15 int stackCount (STACK* stack);
16 STACK* destroyStack (STACK* stack);
17
18 #include "P3-07.h" /* Create Stack */
19 #include "P3-08.h" /* Push Stack */
20 #include "P3-09.h" /* Pop Stack */
21 #include "P3-10.h" /* Retrieve Stack Top */
22 #include "P3-11.h" /* Empty Stack */
23 #include "P3-12.h" /* Full Stack */
24 #include "P3-13.h" /* Stack Count */
25 #include "P3-14.h" /* DestroyStack */
```

TAD Pilha (Stack ADT)

Criar Pilha (Create Stack)

P3-07.h

```
1  /* ===== createStack =====
2      This algorithm creates an empty stack.
3      Pre  Nothing
4      Post Returns pointer to a null stack
5              -or- NULL if overflow
6  */
7  STACK* createStack (void)
8  {
9      // Local Definitions
10     STACK* stack;
11
12     // Statements
13     stack = (STACK*) malloc( sizeof (STACK));
14     if (stack)
15     {
16         stack->count = 0;
17         stack->top    = NULL;
18     } // if
19     return stack;
20 }
```

Cria uma pilha vazia.

Entrada: Nenhuma.

Saída: Retorna um ponteiro a um
cabeçalho (pilha) ou um ponteiro nulo, caso
não houver memória

// Retorna um ponteiro a (cabeçalho) pilha

// Ponteiro a (cabeçalho) pilha

// Alocação de memória (cabeçalho) pilha

// Contador é zerado

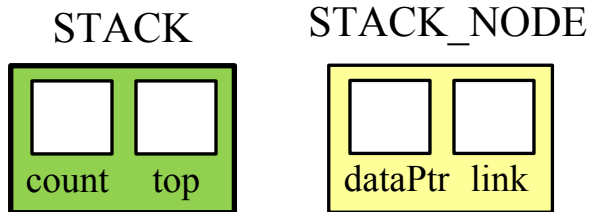
// Ponteiro ao topo é nulo

// Retorna ponteiro a (cabeçalho) pilha

TAD Pilha (Stack ADT)

Criar Pilha (Create Stack) - Figura

Definição de Estruturas:



```
STACK* pilha;  
...  
pilha = createStack();
```

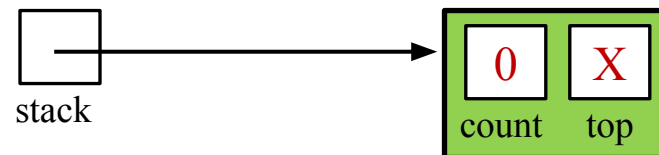
1 STACK* stack;



2 stack=(STACK*)malloc(sizeof(STACK));



3 stack->count:=0;
stack->top:=NULL;



4 return stack;

TAD Pilha (Stack ADT)

P3-08.h

Inserir Pilha (Push Stack)

```
1  /* ===== pushStack =====
2  This function pushes an item onto the stack.
3      Pre      stack is a pointer to the stack
4              dataPtr pointer to data to be inserted
5      Post     Data inserted into stack
6      Return   true  if successful
7              false if underflow
8  */
9  bool pushStack (STACK* stack, void* dataInPtr)
10 {
11     // Local Definitions
12     STACK_NODE* newPtr;
13
14     // Statements
15     newPtr = (STACK_NODE* ) malloc(sizeof( STACK_NODE));
16     if (!newPtr)
17         return false;
18
19     newPtr->dataPtr = dataInPtr;
20
21     newPtr->link     = stack->top;
22     stack->top       = newPtr;
23
24     (stack->count)++;
25     return true;
26 }
```

Inserir um elemento na pilha (no início).

Entrada: Ponteiro a uma pilha;

Ponteiro genérico (a um dado)

Saída: Dado inserido na pilha;

Retorna: true, caso sucesso;

false, caso underflow

// Retorna um valor true ou false

// Ponteiro a um nó da pilha

// Alocar memória para um nó da pilha

// Se alocação falhar

// Retorna false

// Liga o dado inserido ao novo nó

// Liga o topo da pilha ao novo nó

// Atualiza topo da pilha com novo nó

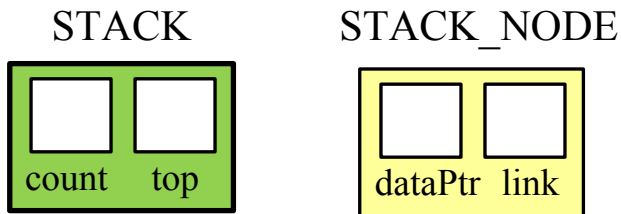
// Incrementa o contador da pilha

// Retorna true

TAD Pilha (Stack ADT)

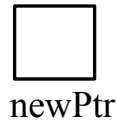
Inserir Pilha (Push Stack) - Figura

Definição de Estruturas:

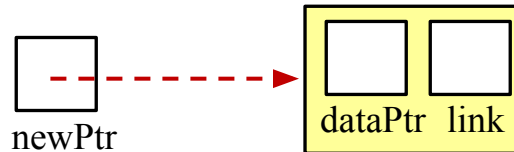


```
STACK* pilha;  
int* num;  
...  
pilha = createStack();  
...  
pushStack (pilha, num);
```

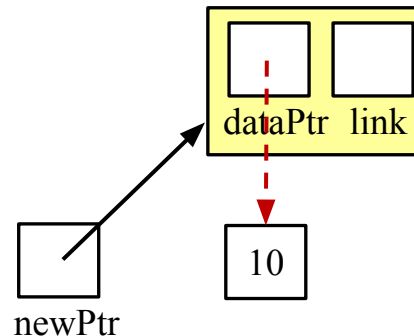
0 STACK_NODE* newPtr;



1 newPtr=(STACK_NODE*)malloc(sizeof(STACK_NODE));



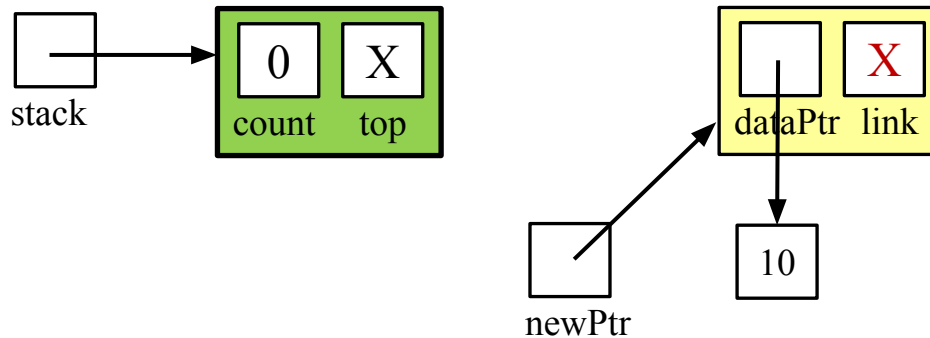
2 newPtr ->dataPtr=dataInPtr;



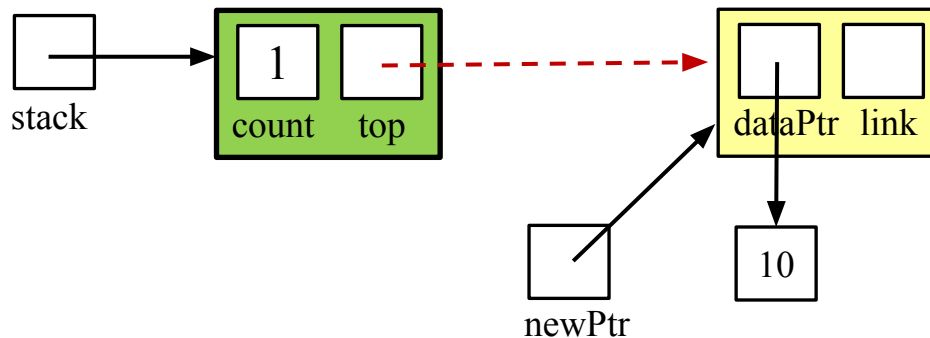
TAD Pilha (Stack ADT)

Inserir Pilha (Push Stack) - Figura

3 newPtr ->link=stack->top;



5 stack->top=newPtr;
(stack->count)++;

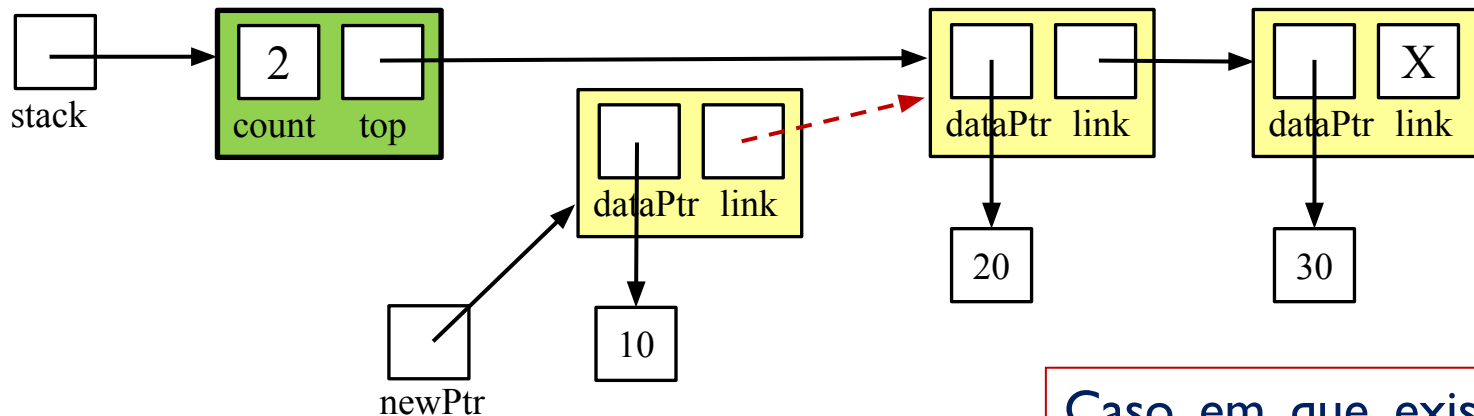


Caso não existem elementos na pilha.

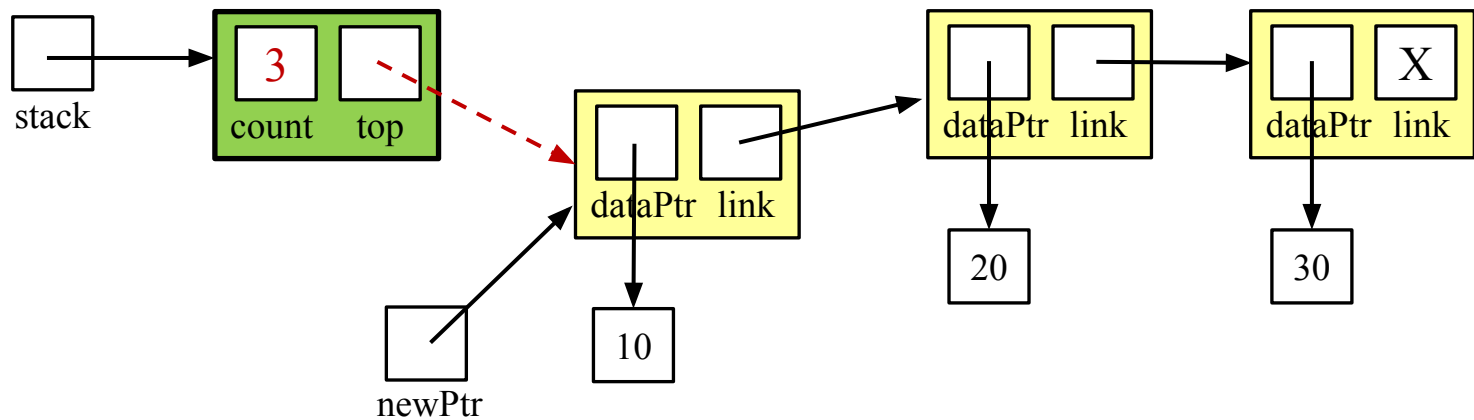
TAD Pilha (Stack ADT)

Inserir Pilha (Push Stack) - Figura

3 newPtr ->link=stack->top;



5 stack->top=newPtr;
(stack->count)++;



Caso em que existem elementos na pilha.

TAD Pilha (Stack ADT)

P3-09.h

Remover Pilha (Pop Stack)

Remove o elemento no topo da pilha.
Entrada: um ponteiro a uma pilha
Saída: ponteiro a um dado ou;
ponteiro nulo.

```
1  /* ===== popStack =====
2  This function pops item on the top of the stack.
3  Pre  stack is pointer to a stack
4  Post Returns pointer to user data if successful
5       NULL if underflow
6  */
7  void* popStack (STACK* stack)
8  {
9  // Local Definitions
10     void*      dataOutPtr;
11     STACK_NODE* temp;
12
13 // Statements
14 if (stack->count == 0)
15     dataOutPtr = NULL;
16 else
17     {
18         temp      = stack->top;
19         dataOutPtr = stack->top->dataPtr;
20         stack->top = stack->top->link;
21         free (temp);
22         (stack->count)--;
23     } // else
24 return dataOutPtr;
25 }
```

// Retorna um ponteiro genérico

// Ponteiro genérico (a um dado)
// Ponteiro ao um nó (de ligação)

// Se contador estiver zerado
// Ponteiro a um dado será nulo
// Se não

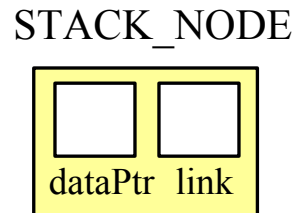
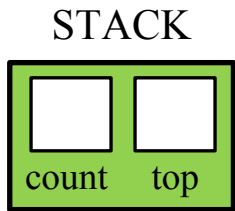
// Copia ponteiro ao primeiro nó
// Copia ponteiro ao primeiro dado
// Atualiza o ponteiro ao primeiro nó
// Libera memória referente ao primeiro nó
// Diminui o contador do elementos

// Retorna ponteiro a um dado

TAD Pilha (Stack ADT)

Remover Pilha (Pop Stack) - Figura

Definição de Estruturas:



```
STACK*   pilha;
int*     num;
```

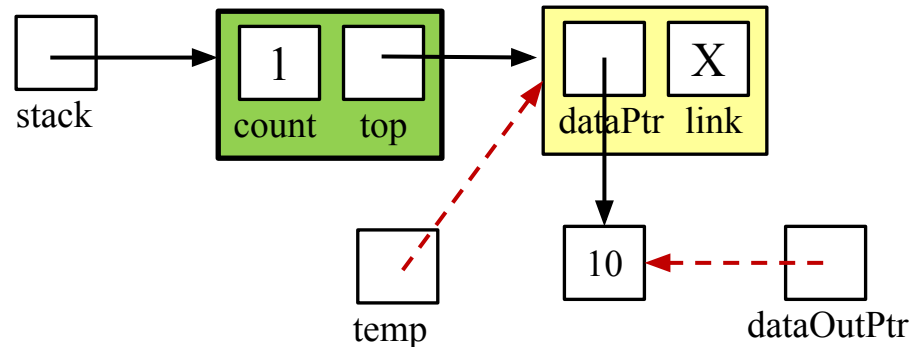
```
...
pilha = createStack();
```

```
...
push (pilha, num);
```

```
...
num=(int*)popStack (pilha);
```

0 void* dataOutPtr;
STACK_NODE* temp;

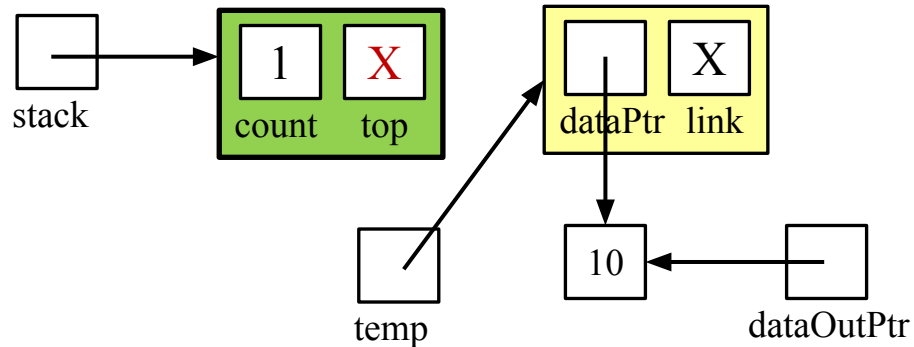
1 temp=stack->top;
dataOutPtr= stack->top->dataPtr;



TAD Pilha (Stack ADT)

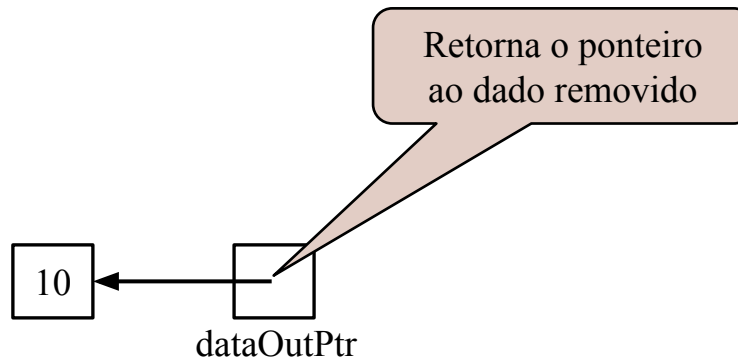
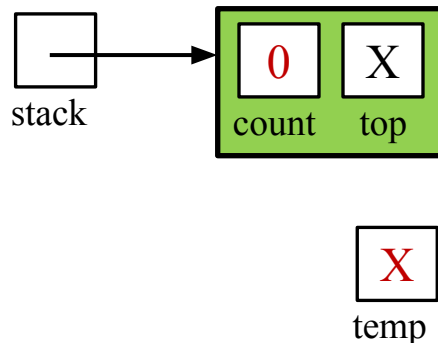
Remover Pilha (Pop Stack) - Figura

2 `stack->top=stack->top->link;`



Caso exista um único elemento na pilha.

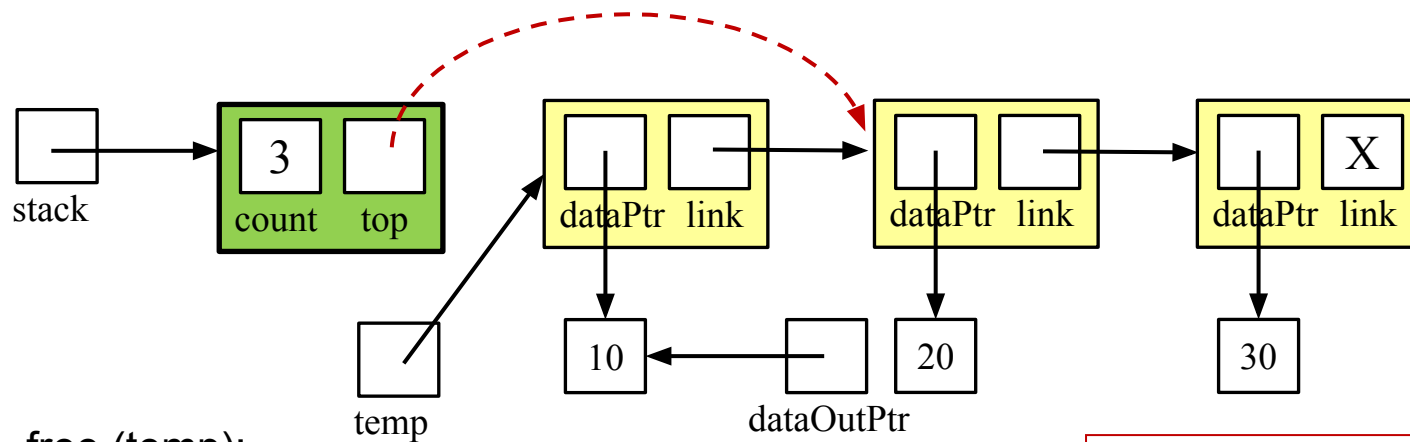
3 `free (temp);`
`(stackCount)--;`
`return dataOutPtr;`



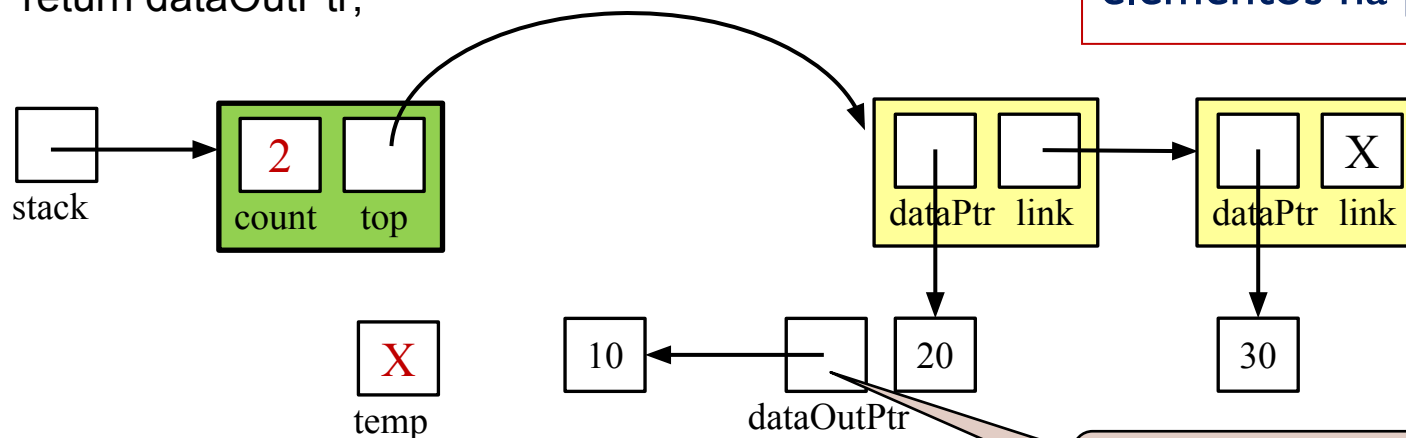
TAD Pilha (Stack ADT)

Remover Pilha (Pop Stack) - Figura

2 `stack->top=stack->top->link;`



3 `free (temp);`
`(stackCount)--;`
`return dataOutPtr;`



Caso existam vários elementos na pilha.

Retorna o ponteiro
ao dado removido

TAD Pilha (Stack ADT)

Topo Pilha (Stack Top)

P3-10.h

```
1  /* ===== stackTop =====
2  Retrieves data from the top of stack without
3  changing the stack.
4  Pre  stack is a pointer to the stack
5  Post Returns data pointer if successful
6       null pointer if stack empty
7  */
```

Da acesso ao dado no topo da pilha sem modificar a pilha.

Entrada: um ponteiro a uma pilha
Saída: ponteiro a um dado ou;
ponteiro nulo.

```
8  void* stackTop (STACK* stack)
9  {
10 // Statements
11     if (stack->count == 0)
12         return NULL;
13     else
14         return stack->top->dataPtr;
15 } // stackTop
```

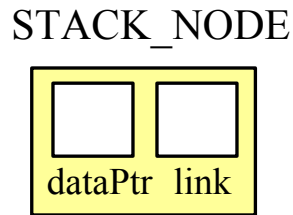
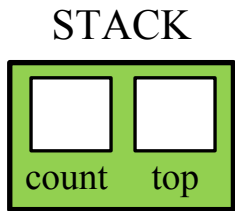
// Retorna um ponteiro genérico

// Se o contador estiver zerado
// Retorna ponteiro nulo
// Se não
// Retorna ponteiro ao dado no topo

TAD Pilha (Stack ADT)

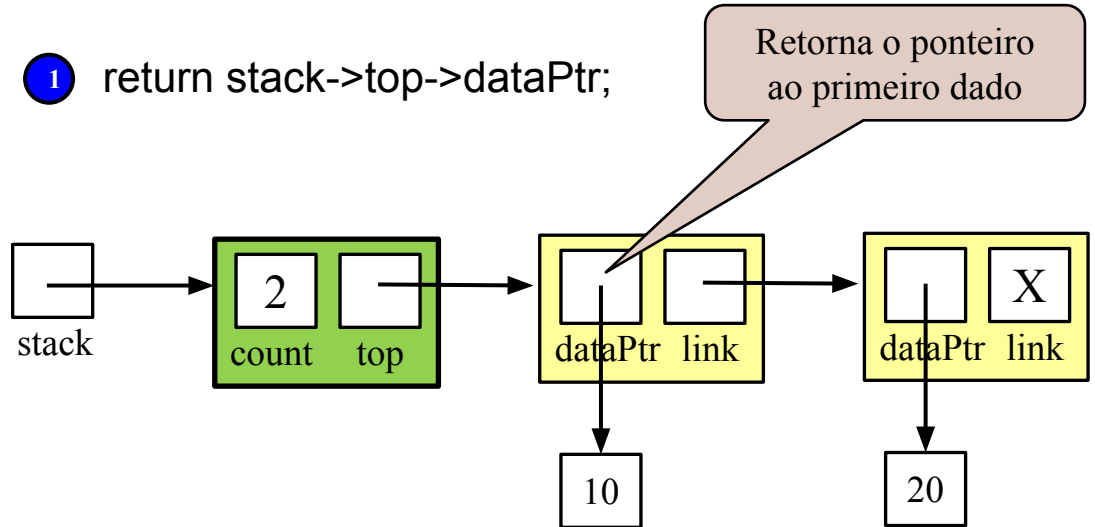
Remover Pilha (Pop Stack) - Figura

Definição de Estruturas:



```
STACK*   pilha;  
int*     num;  
...  
pilha = createStack();  
...  
push (pilha, num);  
...  
num=(int*)popStack (pilha);  
...  
num=(int*)stackTop (pilha);
```

① return stack->top->dataPtr;



TAD Pilha (Stack ADT)

Pilha Vazia (Empty Stack)

P3-11.h

```
1  /* ===== emptyStack =====
2     This function determines if a stack is empty.
3     Pre  stack is pointer to a stack
4     Post returns 1 if empty; 0 if data in stack
5  */
6  bool emptyStack (STACK* stack)
7  {
8  // Statements
9     return (stack->count == 0);
10 } // emptyStack
```

Determina se a pilha está vazia.
Entrada: um ponteiro a uma pilha
Saída: retorna 1 se vazia;
retorna 0 caso contrário.

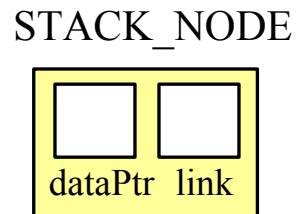
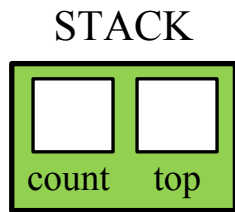
// Retorna um ou zero

// Retorna resultado da comparação.

TAD Pilha (Stack ADT)

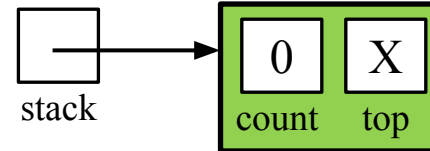
Pilha Vazia (Empty Stack) - Figura

Definição de Estruturas:



```
STACK*   pilha;  
int*     num;  
bool    vazio;  
...  
pilha = createStack();  
...  
vazio=emptyStack(pilha);
```

① return (stack->count==0);



Verifica o contador,
retorna True

TAD Pilha (Stack ADT)

Pilha Cheia (Full Stack)

P3-12.h

```
1  /* ===== fullStack =====
2  This function determines if a stack is full.
3
4  Full is defined as heap full.
5  Pre    stack is pointer to a stack head node
6  Return true if heap full
7         false if heap has room
8  */
9  bool fullStack (STACK* stack)
10 {
11 // Local Definitions
12 STACK_NODE* temp;
13 // Statements
14 if ((temp =
15     (STACK_NODE*)malloc (sizeof(*(stack->top))))
16     {
17     free (temp);
18     return false;
19 } // if
20
21 // malloc failed
22 return true;
23 }
```

Determina se a pilha está cheia.

Entrada: um ponteiro a (cabeçalho) pilha;
Saída: retorna true se memória cheia;
retorna false caso contrário.

// Retorna um valor true ou false

// Ponteiro a um nó da pilha

// Se a tentativa de alocação funcionar

// Libera temp

// Retorna false

// Caso contrário

// Retorna true

TAD Pilha (Stack ADT)

Contador Pilha (Stack Count)

P3-13.h

```
1  /* ===== stackCount =====
2     Returns number of elements in stack.
3     Pre  stack is a pointer to the stack
4     Post count returned
5  */
6  int stackCount (STACK* stack)
7  {
8  // Statements
9     return stack->count;
10 }
```

Retorna o número de elementos na pilha.
Entrada: um ponteiro a (cabeçalho) pilha
Saída: retorna o contador

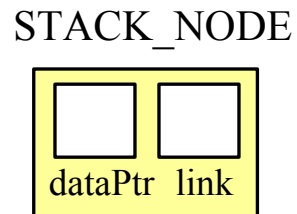
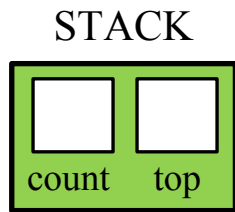
// Retorna um inteiro

// Retorna o contador da pilha

TAD Pilha (Stack ADT)

Pilha Vazia (Empty Stack) - Figura

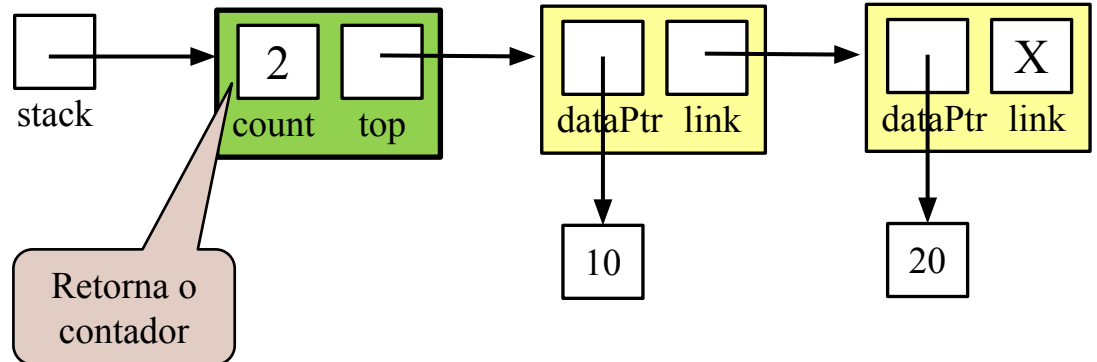
Definição de Estruturas:



```
STACK* pilha;  
int* num;  
bool vazio;  
...  
pilha = createStack();
```

```
...  
vazio=emptyStack(pilha);
```

1 return stack->count;



TAD Pilha (Stack ADT)

Destruir Pilha (Destroy Stack)

P3-14.h

```
1  /* ===== destroyStack =====
2      This function releases all nodes to the heap.
3      Pre  A stack
4      Post returns null pointer
5  */
6  STACK* destroyStack (STACK* stack)
7  {
8      // Local Definitions
9      STACK_NODE* temp;
10
11     // Statements
12     if (stack)
13     {
14         // Delete all nodes in stack
15         while (stack->top != NULL)
16         {
17             // Delete data entry
18             free (stack->top->dataPtr);
19
20             temp = stack->top;
21             stack->top = stack->top->link;
22             free (temp);
23         } // while
24
25         // Stack now empty. Destroy stack head node
26         free (stack);
27     } // if stack
28     return NULL;
29 }
```

Libera todos os nós na memória heap
Entrada: um ponteiro a (cabeçalho) pilha
Saída: retorna ponteiro nulo

// Retorna um ponteiro a (cabeçalho) pilha

// Ponteiro a um nó da pilha

// Se o ponteiro (cabeçalho) pilha não for nulo

// Apaga todos os nós na pilha

// Enquanto ponteiro ao nó topo não for nulo

// Apaga o dado da memória

// Guarda o ponteiro ao nó topo

// Redefine o nó topo com seu sucessor

// Apaga o antigo nó topo da memória

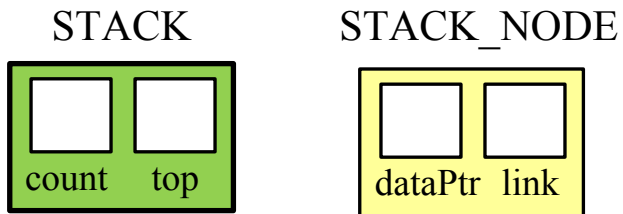
// Apaga o nó cabeçalho

// retorna nulo

TAD Pilha (Stack ADT)

Destruir Pilha (Destroy Stack) - Figura

Definição de Estruturas:

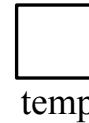


STACK* pilha;
int* num;

...
pilha = createStack();

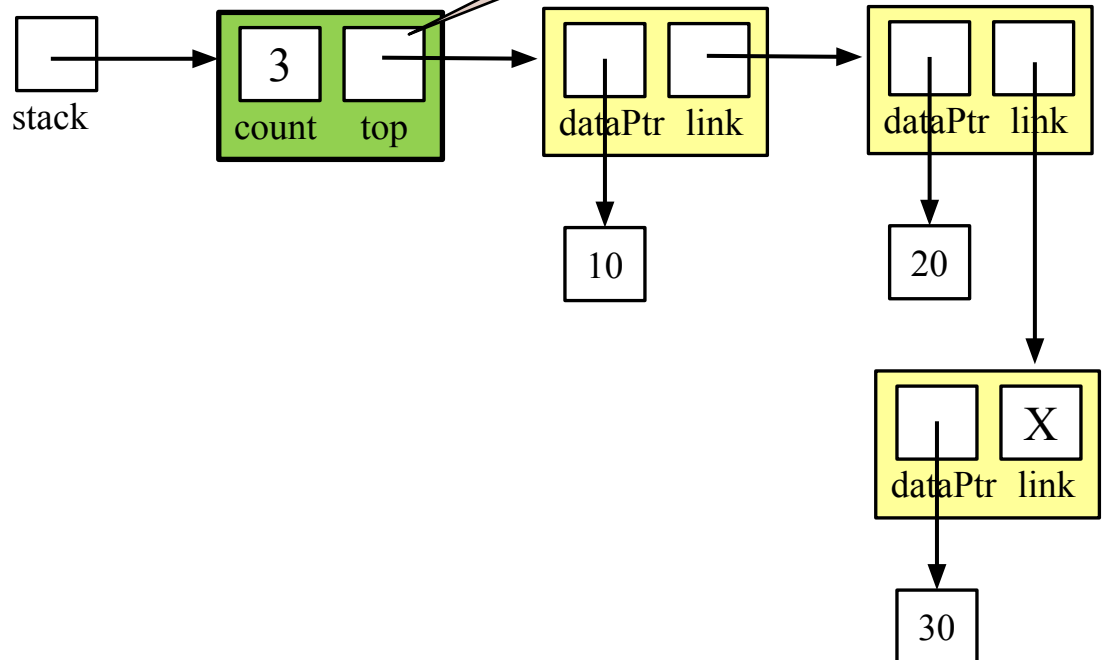
...
pilha=destroyStack(pilha);

0 STACK_NODE* temp;



1 while (stack->top != NULL);

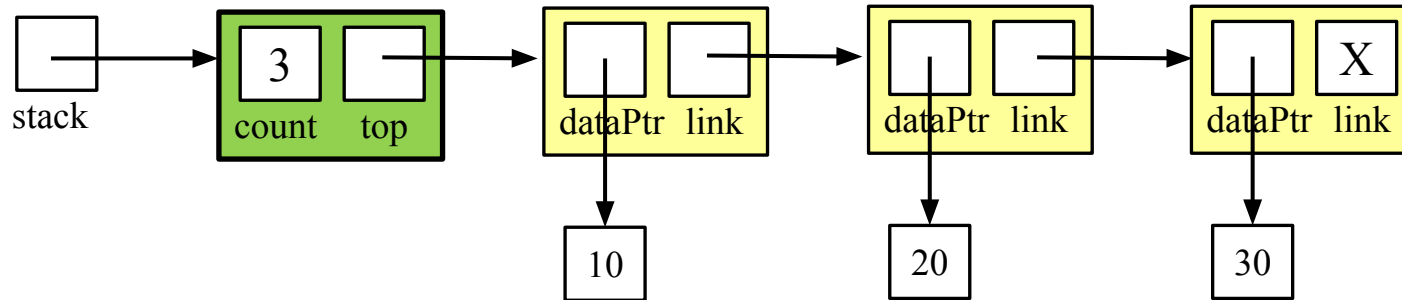
Verifica o ponteiro
ao topo



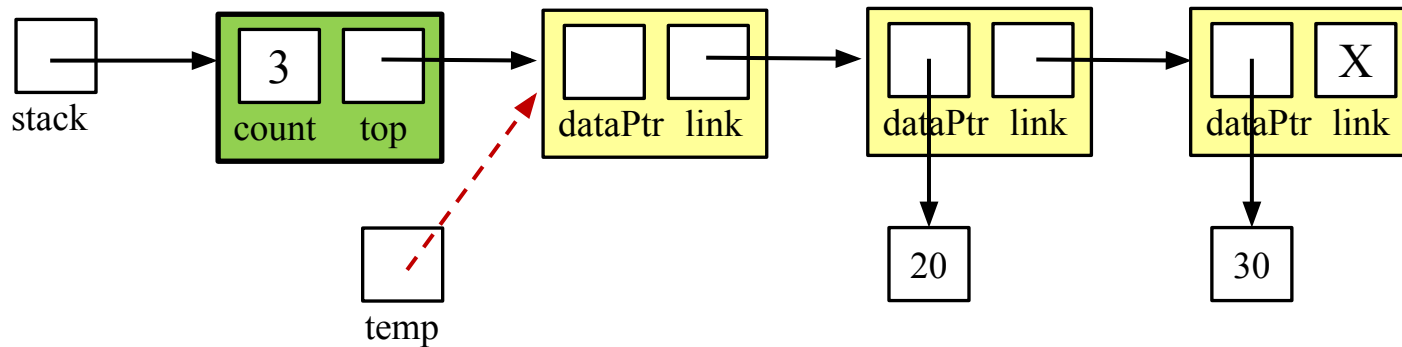
TAD Pilha (Stack ADT)

Destruir Pilha (Destroy Stack) - Figura

1 while (stack->top!=NULL);



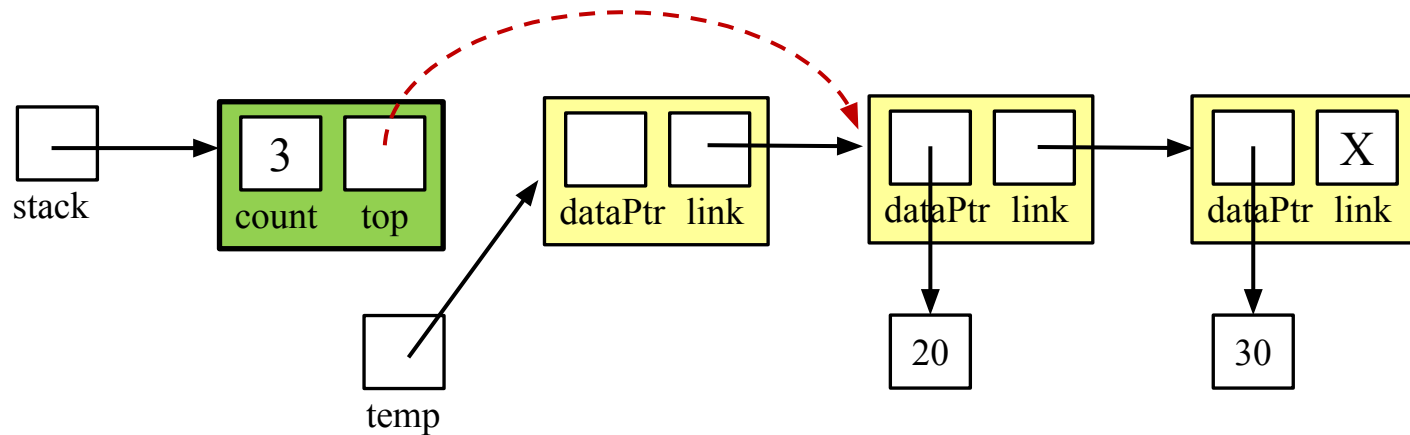
2 free (stack->top->dataPtr);
temp=stack-top;



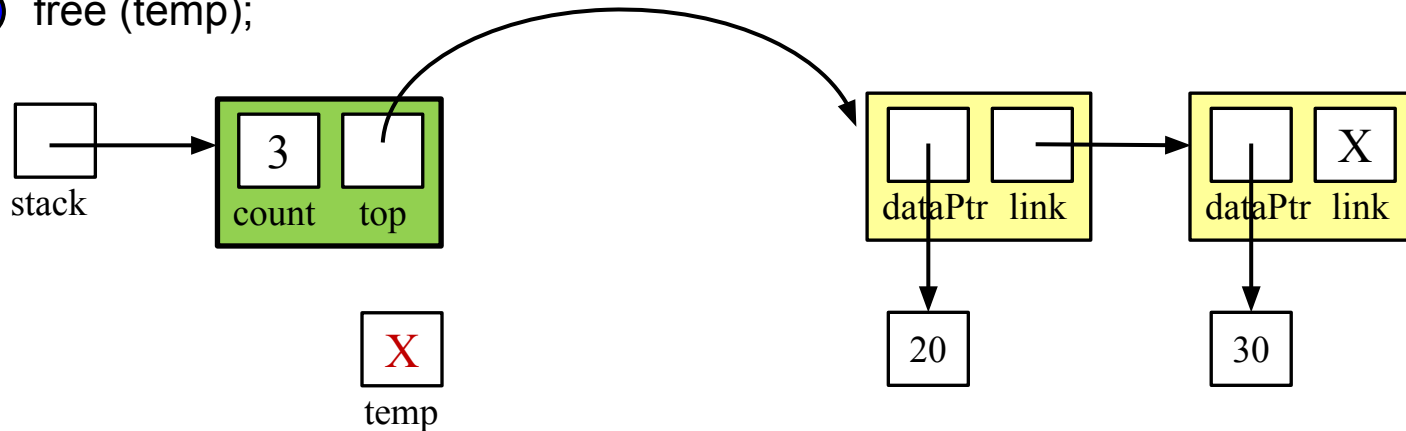
TAD Pilha (Stack ADT)

Destruir Pilha (Destroy Stack) - Figura

- 1 while (stack->top!=NULL);
- 3 stack->top=stack->top->link;



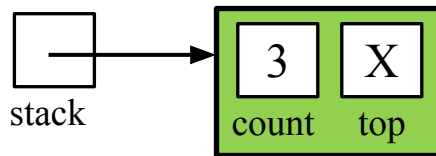
- 4 free (temp);



TAD Pilha (Stack ADT)

Destruir Pilha (Destroy Stack) - Figura

- Após ter realizado três iterações temos:



1 while (stack->top!=NULL);

5 free (stack);



Referências

- Gilberg, R.F. e Forouzan, B. A. Data Structures_A Pseudocode Approach with C. Capítulo 3. Stacks. Segunda Edição. Editora Cengage, Thomson Learning, 2005