Pilhas

Prof. Denio Duarte
duarte@uffs.edu.br
Prof. Claunir Pavan
claunir.pavan@uffs.edu.br

Pilha

- Pilhas são estruturas de dados baseadas em lista simples.
  - Abordagem LIFO: last-in first-out, ou seja, o último a entrar é o primeiro a sair
- Operações básicas
  - **Inserir** um novo item no topo da pilha (push)
  - **Remover** um item do topo da pilha (pop)
- É comum também podermos realizar outras operações úteis em uma pilha:
  - **Inicializar** a pilha
  - **Testar** se a pilha está **vazia**
  - **Destruir** a pilha (tipicamente, liberar a memória alocada para a estrutura de dados)

Pilha

- Usos
  - Controle de chamada de funções dentro dos programas, a última função chamada é a primeira a ser destruída
  - Controle de prioridade nas operações matemáticas: (, {, [e operadores

Exemplo de funcionamento

Entrada:

L A * S T I * N ...

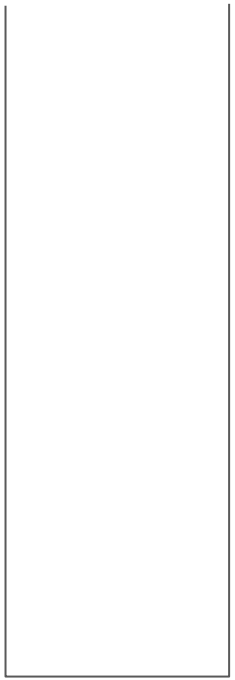
Processamento:

- Se a entrada é uma letra: insere na pilha
- Se é um * : remove da pilha

Saída:

[redacted]

Pilha



Pilha vazia

Exemplo de funcionamento

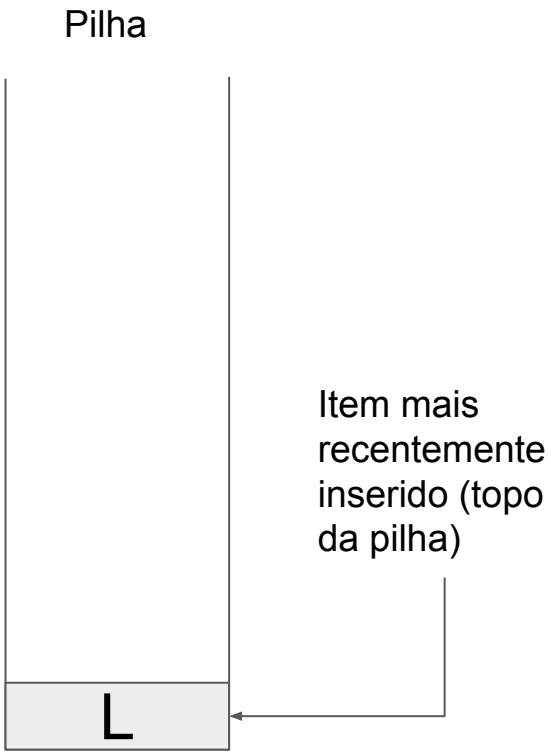
Entrada:

A * S T I * N ...

Processamento:

- Se a entrada é uma letra: insere na pilha
- Se é um * : remove da pilha

Saída:



Exemplo de funcionamento

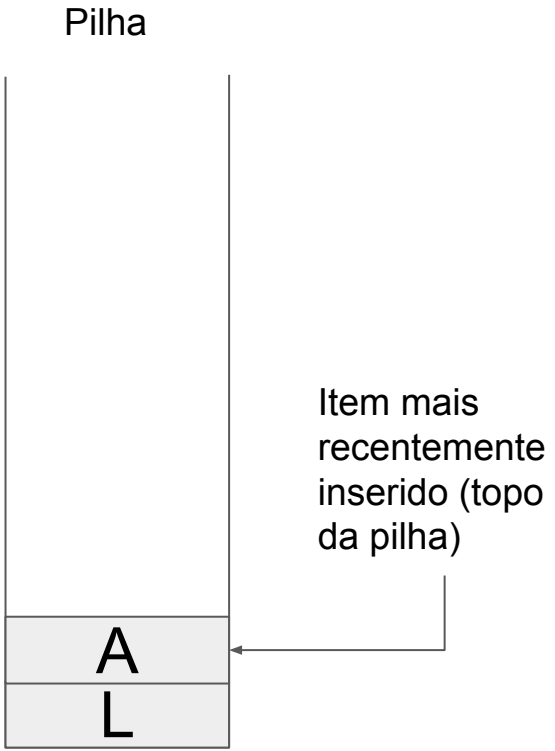
Entrada:

* S T I * N ...

Processamento:

- Se a entrada é uma letra: insere na pilha
- Se é um * : remove da pilha

Saída:



Exemplo de funcionamento

Entrada:

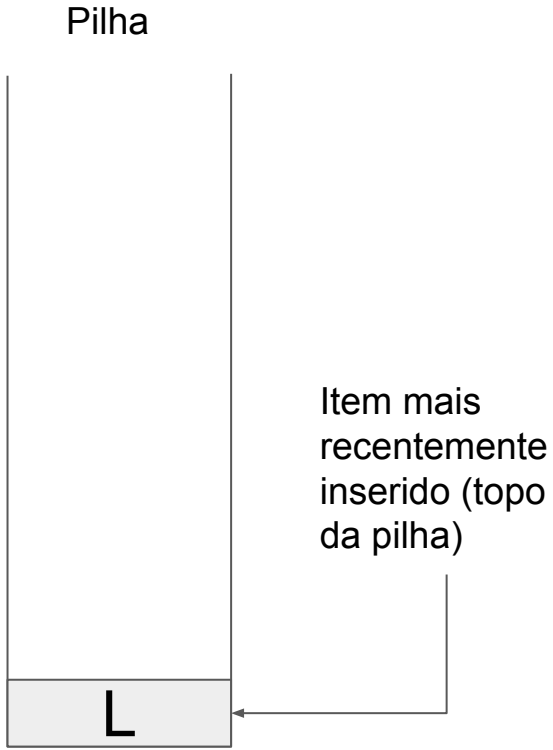
S T I * N ...

Processamento:

- Se a entrada é uma letra: insere na pilha
- Se é um * : remove da pilha

Saída:

A



Exemplo de funcionamento

Entrada:

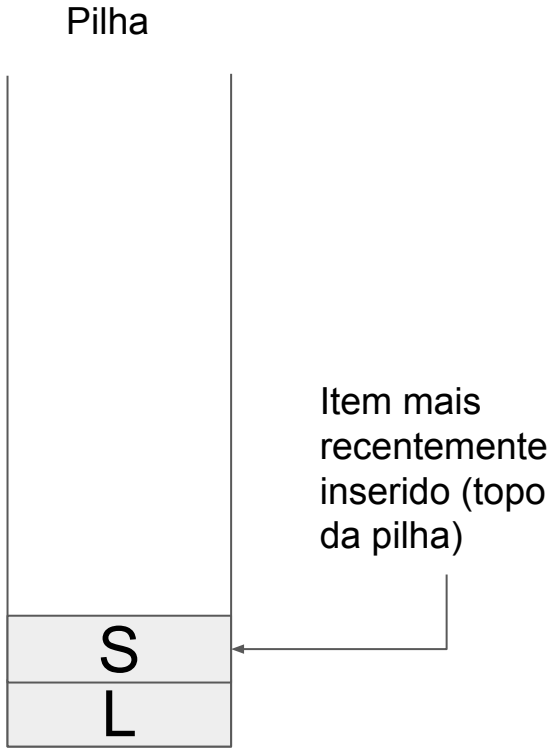
T I * N ...

Processamento:

- Se a entrada é uma letra: insere na pilha
- Se é um * : remove da pilha

Saída:

A



Exemplo de funcionamento

Entrada:

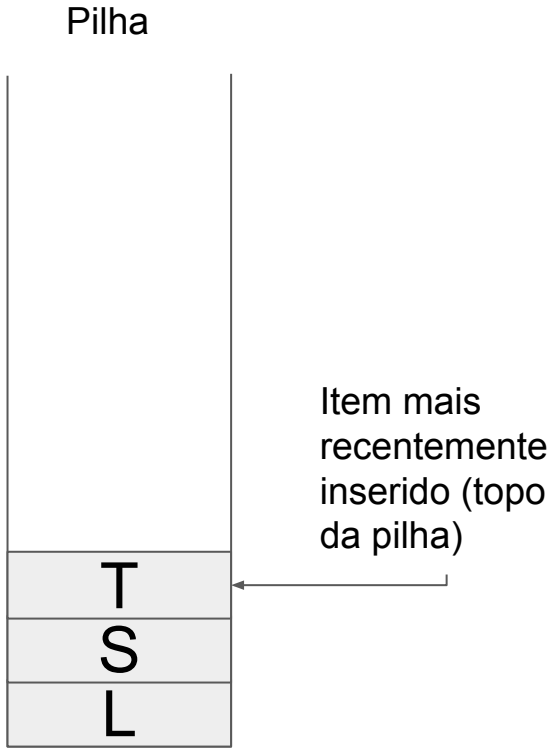
I * N ...

Processamento:

- Se a entrada é uma letra: insere na pilha
- Se é um * : remove da pilha

Saída:

A



Exemplo de funcionamento

Entrada:

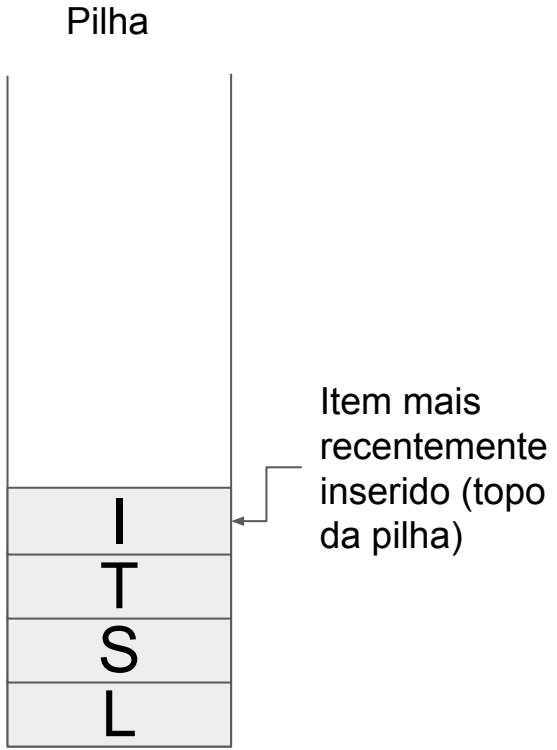
* N ...

Processamento:

- Se a entrada é uma letra: insere na pilha
- Se é um * : remove da pilha

Saída:

A



Exemplo de funcionamento

Entrada:

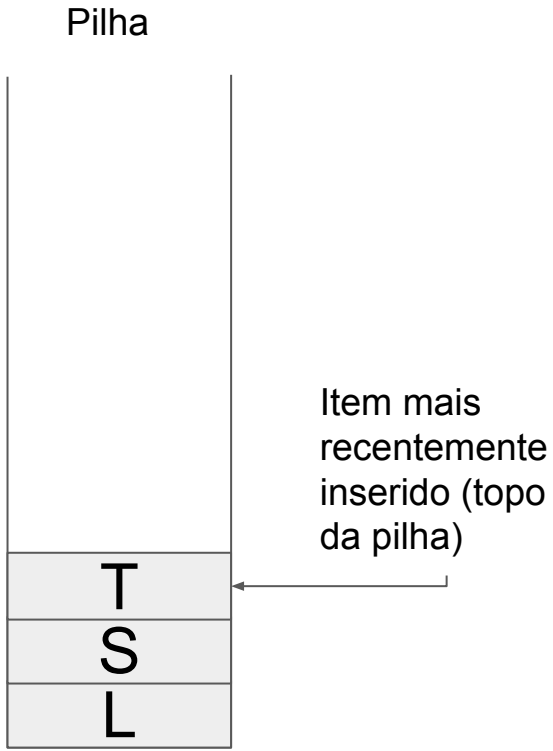
N ...

Processamento:

- Se a entrada é uma letra: insere na pilha
- Se é um * : remove da pilha

Saída:

A I



Exemplo de funcionamento

Entrada:

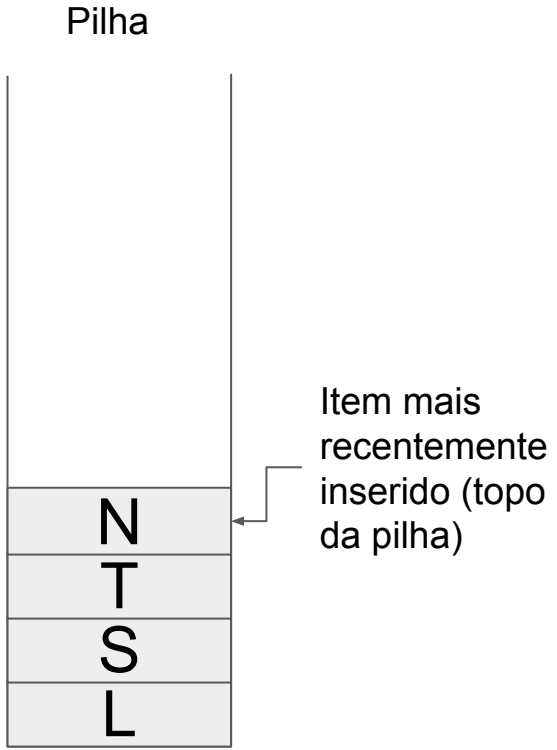
...

Processamento:

- Se a entrada é uma letra: insere na pilha
- Se é um * : remove da pilha

Saída:

A I

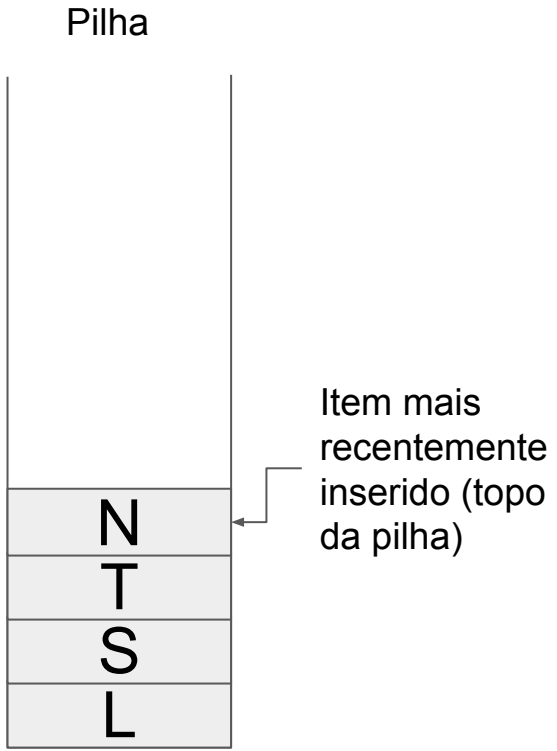


Exemplo de funcionamento

Entrada:

...

O processamento continuaria...



Exemplo de aplicação

Problema: Verificar se as chaves, colchetes e parênteses de um programa estão corretamente aninhados.

Entrada (após pré-processamento):

( ) { [ ] }

Saída:

Correto

Entrada (após pré-processamento):

( ) [{] }

Saída:

Incorreto

Exemplo de aplicação

Entrada:

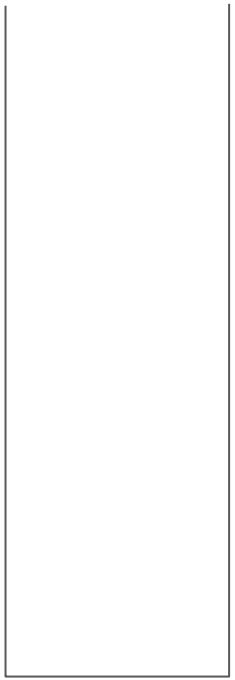
( ) { [ ] }

Solução:

- Se é (, { ou [: insere na pilha
- Se é), } ou]: remove da pilha e compara se deu match, isto é, topo igual à entrada

Saída:

Pilha



Pilha vazia

Exemplo de aplicação

Entrada:

) { [ ] }

Solução:

- Se é (, { ou [: insere na pilha
- Se é), } ou]: remove da pilha e compara se deu match, isto é, topo igual à entrada

Saída:

Exemplo de aplicação

Entrada:

{ [ ] }

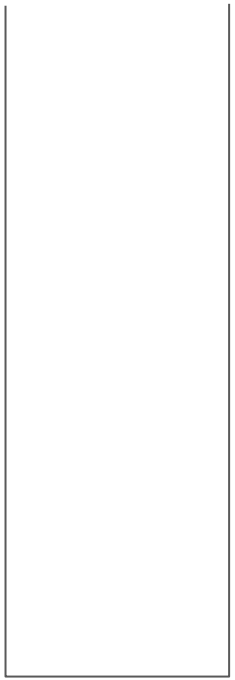
Solução:

- Se é (, { ou [: insere na pilha
- Se é), } ou]: remove da pilha e compara se deu match, isto é, topo igual à entrada

Saída:

[redacted]

Pilha



Pilha vazia

Exemplo de aplicação

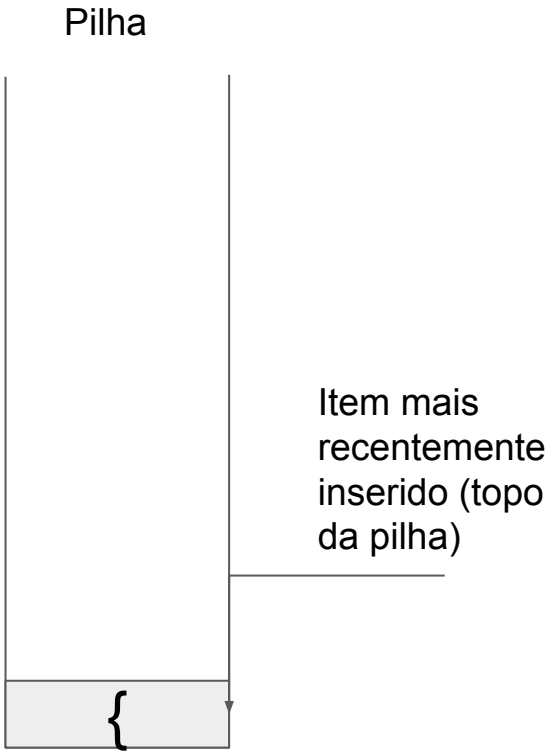
Entrada:

[ ] }

Solução:

- Se é (, { ou [: insere na pilha
- Se é), } ou]: remove da pilha e compara se deu match, isto é, topo igual à entrada

Saída:



Exemplo de aplicação

Entrada:

] }

Solução:

- Se é (, { ou [: insere na pilha
- Se é), } ou]: remove da pilha e compara se deu match, isto é, topo igual à entrada

Saída:

Exemplo de aplicação

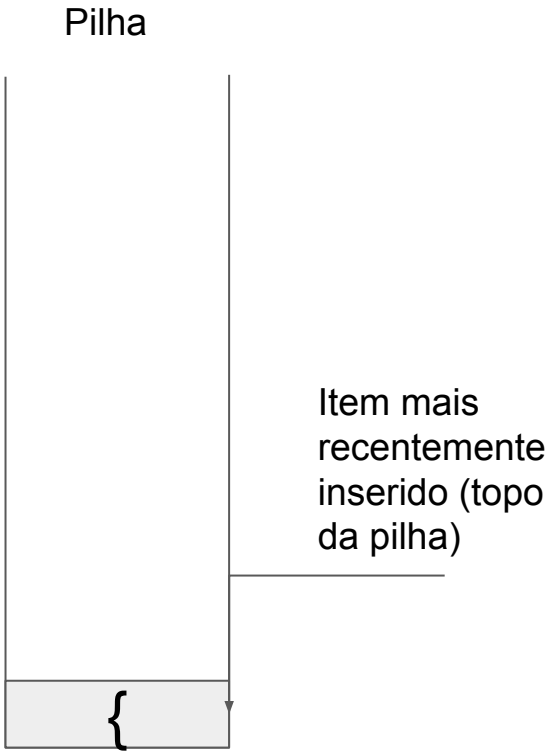
Entrada:

}

Solução:

- Se é (, { ou [: insere na pilha
- Se é), } ou]: remove da pilha e compara se deu match, isto é, topo igual à entrada

Saída:



Exemplo de aplicação

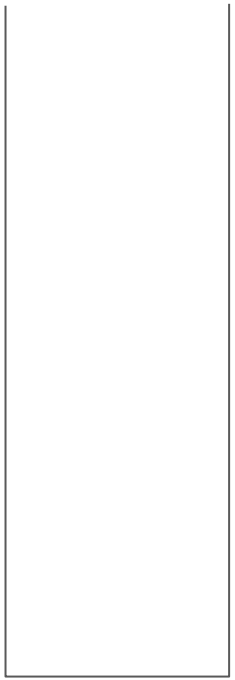
Entrada:

Solução:

- Se é (, { ou [: insere na pilha
- Se é), } ou]: remove da pilha e compara se deu match, isto é, topo igual à entrada

Saída:

Pilha



Pilha vazia

Exemplo de aplicação

Entrada:

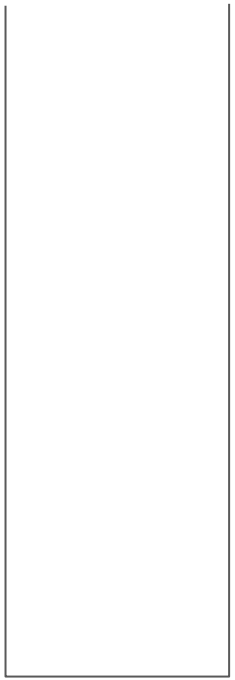
Solução:

- Se é (, { ou [: insere na pilha
- Se é), } ou]: remove da pilha e compara se deu match, isto é, topo igual à entrada

Saída:

Correto (pilha e entrada vazias)

Pilha



Pilha vazia

Implementação

- Comumente, uma pilha é implementada de duas maneiras: usando um vetor ou usando uma lista encadeada simples
- Usando um vetor
  - Desvantagem: É necessário definir um tamanho máximo da pilha; uso ineficiente da memória total alocada
  - Vantagem: Inserção e remoção de itens não requerem alocação e liberação de memória
- Usando uma lista encadeada simples
  - Desvantagem: Inserção e remoção de itens requerem alocação e liberação de memória
  - Vantagem: Uso mais eficiente da memória total alocada

Implementação - lista encadeada simples

- Podemos declarar os seguintes tipos:

```
typedef int Item;
```

```
typedef struct elemStack {
    Item item;
    struct elemStack *next;
} ElemStack;
```

```
typedef struct {
    ElemStack *top;
} Stack;
```

Implementação - lista encadeada simples

- Podemos declarar os seguintes tipos:

```c
typedef int Item;
```

```c
typedef struct elemStack {
    Item item;
    struct elemStack *next;
} ElemStack;
```

```c
typedef struct {
    ElemStack *top;
} Stack;
```

```c
Stack *stack;
```

ou

```c
typedef int Item;
```

```c
typedef struct elemStack {
    Item item;
    struct elemStack *next;
} ElemStack;
```

```c
// sem estrutura para o sentinela
ElemStack *stack, *top;
```

Implementação - lista encadeada simples

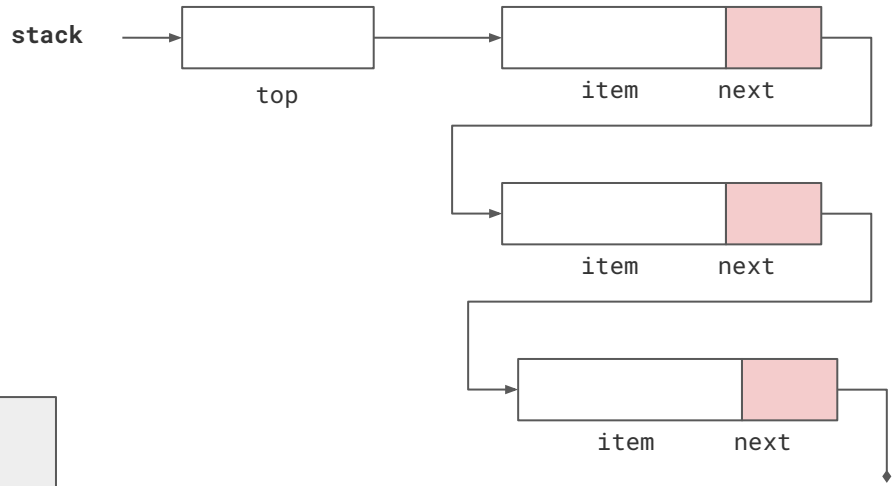
- Podemos declarar os seguintes tipos:

```c
typedef int Item;
```

```c
typedef struct elemStack {
    Item item;
    struct elemStack *next;
} ElemStack;
```

```c
typedef struct {
    ElemStack *top;
} Stack;
```

```c
Stack *stack;
```



Implementação - lista encadeada simples

- Operações:

```
void push(Stack *s, Item item)

void pop (Stack *s, Item *item)

void initStack(Stack *s)

int isEmptyStack(Stack *s)

void freeStack(Stack *s)
```

Implementação - lista encadeada simples

- Operação de inserir um novo elemento na pilha:

```
void push(Stack *s, Item item) {
    ElemStack *aux;

    // Cria um novo elemento da lista encadeada que representa a pilha e
    // armazena neste novo elemento o item a ser inserido na pilha
    aux = (ElemStack *)malloc(sizeof(ElemStack));
    aux->item = item;

    // Insere o novo elemento no início da lista encadeada que representa a
    // pilha
    aux->next = s->top;
    s->top = aux;
}
```

Implementação - lista encadeada simples

- Operação de remover um elemento da pilha:

```
void pop(Stack *s, Item *item) {
    ElemStack *aux;

    // Verificar se a pilha está vazia!

    // Armazena o item a ser removido da pilha
    *item = s->top->item; // ATENÇÃO: Depende da definição do tipo do item

    // Armazena o primeiro elemento da lista encadeada que representa a pilha
    // e remove este primeiro elemento da lista
    aux = s->top;
    s->top = s->top->next;

    // Libera a memória alocada para o elemento removido
    free(aux);
}
```

Implementação - lista encadeada simples

- Operação de inicializar a pilha:

```
void initStack(Stack *s) {
    s->top = NULL;
}
```

Implementação - lista encadeada simples

- Operação de testar se a pilha está vazia:

```
int isEmptyStack(Stack *s) {
    return (s->top == NULL);
}
//ou
int isEmptyStack(Stack *s) {
    if (s->top == NULL) return 1;
    else return 0;
}
```

Implementação - lista encadeada simples

- Operação de destruir a pilha (liberar a memória alocada para a pilha):

```
void freeStack(Stack *s) {
    ElemStack *aux;

    while (!isEmptyStack(s)) {
        // Armazena o primeiro elemento da lista encadeada que representa a
        // pilha e remove este primeiro elemento da lista
        aux = s->top;
        s->top = s->top->next;

        // Libera a memória alocada para o elemento removido
        free(aux);
    }
}
```

Implementação - lista encadeada simples

- Usando a pilha:

```c
int main() {
    Stack stack;
    Item item;

    initStack(&stack);

    for (int i = 0; i < 10; i++) {
        item = i;

        printf("Inserindo na pilha o item %d.\n", item);
        push(&stack, item);
    }

    // Continua no proximo slide
```

Implementação - lista encadeada simples

- Usando a pilha:

```
    // Continuacao do slide anterior

    while (isEmptyStack(&stack) == 0) {
        pop(&stack, &item);
        printf("Item %d removido da pilha.\n", item);
    }

    freeStack(&stack); // Sem efeito se a pilha já estiver vazia

    return 0;
}
```

Exercícios

1. Faça um programa que receba uma string, caractere por caractere
   - Cada caractere é colocado em uma pilha
   - No fim da entrada, esvazie a pilha imprimindo os caracteres armazenados