

```
/**
 * Arquivo: stackTAD1.h
 * Versão : 1.0
 * Data   : 2024-10-15 00:51
 * -----
 * Este arquivo define uma interface para um tipo abstrato de dado do tipo
 * piha (stack). Esta NÃO É UMA PILHA GENÉRICA, ou seja, não pode ser usada
 * diretamente para armazenar dados de qualquer tipo, ao contrário: esta
 * interface limita o stack a um único tipo de dado (double por padrão). É
 * possível alterar o tipo de dado (basta alterar o tipo de dado na definição do
 * "elementoT" nesta interface), mas isso exigiria algumas alterações em funções
 * que estabelecem alguns comportamentos que dependem do conhecimento do tipo
 * double (pop e ver_elemento).
 *
 * Baseado em: Programming Abstractions in C, de Eric S. Roberts.
 *             Capítulo 8: Abstract Data Types (pg. 327-347).
 *
 * Prof.: Abrantes Araújo Silva Filho (Computação Raiz)
 *        www.computacaoraiz.com.br
 *        www.youtube.com.br/computacaoraiz
 *        github.com/computacaoraiz
 *        twitter.com/ComputacaoRaiz
 *        www.linkedin.com/company/computacaoraiz
 *        www.abrantes.pro.br
 *        github.com/abrantesasf
 */

/**** Inicialização do Boilerplate da Interface: ****/

#ifndef _STACKTAD_H
#define _STACKTAD_H

/**** Includes: ****/

#include "genlib.h"

/**** Tipos de Dados: ****/

/* Seção 1: tipos concretos de dados */

/**
 * TIPO: elementoT
 * -----
 * O tipo "elementoT" é utilizado nesta interface para indicar o tipo de dado
 * dos valores que serão armazenados no stack, ou seja, representa o tipo de
 * dado dos elementos. Nesta interface, por padrão, o tipo de dado armazenado é
 * "double", mas isso pode ser alterado ao se editar a linha da definição, a
 * seguir (é necessário compilar a implementação da interface se esse tipo
 * padrão for alterado).
 */

typedef double elementoT;

/* Seção 2: tipos abstratos de dados */

/**
 * TIPO ABSTRATO: stackTAD
 * -----
 * O tipo stackTAD representa um tipo abstrato que é uma pilha, para armazenar
 * os elementos do tipo "elementoT". Como o stackTAD é definido apenas como um
```

```
* ponteiro para uma estrutura concreta que não está definida nesta interface,  
* os clientes não têm acesso à implementação interna.  
*/
```

```
typedef struct stackTCD *stackTAD;
```

```
/**/ Declarações de Subprogramas: */
```

```
/**  
 * FUNÇÃO: criar_stackTAD  
 * Uso: stackTAD = criar_stackTAD( );  
 * -----  
 * Esta função aloca e retorna um novo stack, que está inicialmente vazio. Se  
 * ocorrer algum erro na criação do stack, será retornado NULL.  
 */
```

```
stackTAD criar_stackTAD (void);
```

```
/**  
 * PROCEDIMENTO: remover_stackTAD  
 * Uso: remover_stackTAD(&stackTAD);  
 * -----  
 * Este procedimento libera o armazenamento alocado para o stackTAD. Note que o  
 * argumento é um PONTEIRO para um stack.  
 */
```

```
void remover_stackTAD (stackTAD *stack);
```

```
/**  
 * PROCEDIMENTO: push  
 * Uso: push(stack, elemento);  
 * -----  
 * Este procedimento coloca o elemento especificado no topo do stack.  
 */
```

```
void push (stackTAD stack, elementoT elemento);
```

```
/**  
 * FUNÇÃO: pop  
 * Uso: elemento = pop(stack);  
 * -----  
 * Esta função retira o elemento que está no topo da pilha e retorna esse  
 * elemento. O valor a ser retirado é sempre o último que entrou na pilha por  
 * último. Se o stack for null ou estiver vazio, é exibida uma mensagem de erro  
 * e o valor retornado será um valor NAN, ou seja, o valor padrão que representa  
 * um "Not a Number" conforme o padrão IEEE-754. Seu programa deve verificar se  
 * o valor retornado é, ou não, um NaN e tomar ações apropriadas:  
 *  
 *     if (isnan(elemento = pop(stack)))  
 *         ... código para tratar o erro do retorno do pop ...  
 *     else  
 *         ... código para prosseguir com seu programa ...  
 *  
 * Obs.: você precisa de math.h para usar a função isnan(x);  
 */
```

```
elementoT pop (stackTAD stack);
```

```
/**  
 * PREDICADOS: vazia, cheia
```

```
* Uso: if (vazia(stack)) . . .
*       if (cheia(stack)) . . .
* -----
* Estes predicados retornam TRUE caso a pilha esteja vazia ou cheia,
* respectivamente. Se o stack for dinâmico, ou seja, sem um tamanho máximo
* pré-definido, o predicado "cheia" sempre retornará FALSE.
*/

bool vazia (stackTAD stack);
bool cheia (stackTAD stack);

/**
 * FUNÇÃO: tamanho
 * Uso: n = tamanho(stack);
 * -----
 * Esta função retorna o tamanho do stack, ou seja, quantos elementos este stack
 * pode armazenar. Três situações especiais podem ocorrer:
 *
 * a) A função retorna normalmente o tamanho do stack;
 * b) Ocorre um erro na função: será retornado o valor -1;
 * c) Se o stack é dinâmico, sem tamanho máximo, será retornado o valor 0.
 */
long int tamanho (stackTAD stack);

/**
 * FUNÇÃO: qtd_elementos
 * Uso: n = qtd_elementos(stack);
 * -----
 * Esta função retorna a quantidade de elementos atualmente dentro da pilha, ou
 * o valor -1 caso ocorra algum erro.
 */
long int qtd_elementos (stackTAD stack);

/**
 * FUNÇÃO: espaco_restante
 * Uso: n = espaco_restante(stack);
 * -----
 * Esta função é utilizada geralmente para DEBUG, não é um comportamento padrão
 * de um TAD pilha. Ela retorna a quantidade de "slots" ainda disponíveis na
 * pilha. Caso o stack seja dinâmico, ou seja, sem um tamanho máximo
 * pré-definido, retorna 0. Se houver algum erro, retorna -1.
 *
 * TODO: o retorno 0 (zero) pode indicar tanto uma pilha dinâmica sem limite
 *        pré-definido, ou uma pilha cheia. Isso está errado, é preciso ajustar
 *        esse comportamento.
 */

#ifdef debug
long int espaco_restante (stackTAD stack);
#endif

/**
 * FUNÇÃO: ver_elemento
 * Uso: elemento = ver_elemento(stack, posicao);
 * -----
 * Esta função é apenas para DEBUG, não é comportamento padrão de um TAD pilha.
 * Ela retorna o elemento especificado em uma posição qualquer da pilha, sem
 * fazer o pop de nenhum elemento. Se a posição especificada não for uma posição
 * válida, ou se o stack for null ou estiver vazio, é exibida uma mensagem de
```

```
* erro e o valor retornado será um valor NAN, ou seja, o valor padrão que
* representa um "Not a Number" conforme o padrão IEEE-754. Seu programa deve
* verificar se o valor retornado é, ou não, um NaN e tomar ações apropriadas:
*
*     if (isnan(elemento = ver_elemento(stack, posicao)))
*         ... código para tratar o erro ...
*     else
*         ... código para prosseguir com seu programa ...
*
* Obs.: você precisa de math.h para usar a função isnan(x);
*/

#ifdef debug
elementoT ver_elemento (stackTAD stack, int posicao);
#endif

/**
* PROCEDIMENTO: imprimir_stack
* Uso: imprimir_stack(stack, limite);
* -----
* Este procedimento é geralmente utilizado para DEBUG, não é comportamento
* padrão de um TAD pilha. Ele imprime os elementos atuais da pilha, até uma
* quantidade limite (para evitar que o terminal do usuário receba várias e
* várias linhas).
*/

#ifdef debug
void imprimir_stack (stackTAD stack, int limite);
#endif

/**** Finalização do Bolierplate da Interface: ****/

#endif
```