

```
/**
 * Arquivo: scannerTAD.h
 * Versão : 1.0
 * Data   : 2024-10-18 08:11
 * -----
 * Este arquivo cria a interface scannerTAD.h, um tipo abstrato de dado (TAD)
 * que exporta um scanner, um programa que "lê" uma linha de texto agrupando as
 * letras para formar palavras (tokens) que são reconhecidas como uma unidade
 * lógica coerente. O scanner divide a string do texto em seus tokens
 * componentes. Um token é definido como:
 *
 * 1. Uma sequência de caracteres alfanuméricos consecutivos (letras ou
 *    números) representando palavras; OU
 * 2. Uma string de apenas um caractere contendo um espaço; OU
 * 3. Uma string de apenas um caractere contendo um sinal de pontuação.
 *
 * Para utilizar esta interface, primeiro você deve criar uma instância de um
 * scannerTAD fazendo:
 *
 *     scanner = criar_scanner( );
 *
 * Todas as chamadas aos outros subprogramas nesta interface receberão esse
 * scanner como argumento, para identificar uma instância em particular do TAD.
 *
 * Para fazer com que o scanner receba a string a ser lida, faça:
 *
 *     ler_string(scanner, str);
 *
 * onde str é a string a partir da qual os tokens devem ser lidos. Para obter
 * cada token individualmente, faça:
 *
 *     token = obter_token(scanner);
 *
 * Para determinar se ainda resta algum token a ser lido, você deve utilizar o
 * predicado existe_outro_token(scanner). A função obter_token(scanner) retorna
 * a string vazia "" (só com o caractere \0) após a leitura do último token.
 *
 * O seguinte fragmento de código serve como um idioma para o processamento de
 * cada token na string de entrada:
 *
 *     scanner = criar_scanner( );
 *     ler_string(scanner, str);
 *     while (existe_outro_token(scanner))
 *     {
 *         token = obter_token(scanner);
 *         . . . processar o token . . .
 *         liberar_token(token);
 *     }
 *
 * Esta versão do scannerTAD.h também suporta as seguintes opções adicionais:
 *
 *     salvar_token
 *     configurar_tratamento_de_espacos
 *
 * Ao terminar o uso de um token específico, você deve liberá-lo da memória com
 * a função: (isso é necessário pois os tokens ficam na Heap)
 *
 *     liberar_token(token);
 *
 * Ao terminar o uso do scanner você deve liberá-lo com:
```

```
*
*      remover_scanner(scanner);
*
* Baseado em: Programming Abstractions in C, de Eric S. Roberts.
*      Capítulo 8: Abstract Data Types (pg. 347-358).
*
* 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
*/

/** Inicia Boilerplate da Interface */

#ifndef _SCANNERTAD_H
#define _SCANNERTAD_H

/** Includes */

#include "genlib.h"

/** Tipos de Dados */

/**
 * TIPO: scannerTAD
 * -----
 * Este é um tipo abstrato de dado utilizado para representar uma instância
 * individual de um scanner. Como qualquer outro TAD, os detalhes da estrutura
 * interna estão ocultos do cliente.
 */

typedef struct scannerTCD *scannerTAD;

/** Declarações de Subprogramas: */

/**
 * FUNÇÃO: criar_scanner
 * Uso: scanner = criar_scanner( );
 * -----
 * Esta função cria uma nova instância de um scanner. Todos os outros
 * subprogramas desta interface receberão o scanner criado como argumento,
 * para identificar qual instância particular do scanner o cliente está
 * utilizando. Isso permite que um mesmo cliente utilize simultaneamente
 * diversos scanners ao mesmo tempo.
 */

scannerTAD criar_scanner (void);

/**
 * PROCEDIMENTO: remover_scanner
 * Uso: remover_scanner(&scanner);
 * -----;
 * Este procedimento remove uma instância de um scanner e libera todas as
 * estruturas de memória associadas a ele. Note que este procedimento precisa
 * receber um PONTEIRO para um scanner.
 */
```

```
*/

void remover_scanner(scannerTAD *scanner);

/**
 * PROCEDIMENTO: ler_string
 * Uso: ler_string(scanner, str);
 * -----
 * Este procedimento inicializa a instância do scanner e faz com que essa
 * instância receba uma string (str) a ser processada para a extração dos
 * tokens.
 */

void ler_string (scannerTAD scanner, string str);

/**
 * FUNÇÃO: obter_token
 * Uso: token = obter_token(scanner);
 * -----
 * Esta função retorna o próximo token lido pelo scanner. Se não houver mais
 * nenhum token disponível, é retornada a string vazia "". O token retornado
 * por esta função é sempre alocado na Heap, o que significa que os clientes
 * devem liberar os tokens quando eles não forem mais necessários.
 */

string obter_token (scannerTAD scanner);

/**
 * PROCEDIMENTO: liberar_token
 * Uso: liberar_token(&token);
 * -----
 * Este procedimento recebe um PONTEIRO para um token armazenado na Heap e
 * libera essa área de memória.
 */

void liberar_token (string *token);

/**
 * PREDICADO: existe_outro_token
 * Uso: if (existe_outro_token( )) . . .
 * -----
 * Este predicado retorna TRUE enquanto existirem tokens adicionais para o
 * scanner ler e retornar.
 */

bool existe_outro_token (scannerTAD scanner);

/**
 * PROCEDIMENTO: salvar_token
 * Uso: salvar_token(scanner, token);
 * -----
 * Este procedimento armazena o token na estrutura de dados do scanner de tal
 * forma que, na próxima vez que obter_token for chamado, esse token armazenado
 * será retornado sem que nenhuma leitura adicional dos caracteres da string
 * lida pelo scanner.
 */

void salvar_token (scannerTAD scanner, string token);

/**
```

```
* PROCEDIMENTO: configurar_tratamento_de_espacos
* Uso: configurar_tratamento_de_espacos(scanner, opcao);
* -----
* Este procedimento controla se o scanner ignorará caracteres em branco ou se
* irá considerá-los tokens válidos. Por padrão o scanner trata caracteres em
* branco (espaços, tabs, etc.) como qualquer outro caractere de pontuação e
* retorna tokens para esses caracteres. Em algumas aplicações (parsers,
* compiladores, etc.) isso não é desejável. Para fazer com que o scanner
* ignore caracteres em branco:
*
*     configurar_tratamento_de_espacos(scanner, 1);
*
* Para fazer com que o scanner volte a considerar os espaços em branco como
* tokens válidos, faça:
*
*     configurar_tratamento_de_espacos(scanner, 0);
*
* As opções para o controle do tratamento de espaços são representadas por
* números inteiros:
*
*     0 = considerar os espaços em branco como tokens
*     1 = ignorar os espaços em branco
*
* Atenção: qualquer valor diferente de 0 ou 1 é entendido como inválido e,
* nesse caso, o scanner volta para seu comportamento padrão (considerar os
* espaços como tokens válidos).
*
* Em código de produção o melhor seria não utilizar números inteiros e criar
* uma enumeração para isso, mas deixar como inteiros simplifica didaticamente
* a implementação no momento.
*/

void configurar_tratamento_de_espacos (scannerTAD scanner, int opcao);

/**
 * FUNÇÃO: obter_tratamento_de_espacos
 * Uso: opcao = obter_tratamento_de_espacos(scanner);
 * -----
 * Esta função retorna a opção atual de tratamento de espaços configurada para o
 * scanner.
 */

int obter_tratamento_de_espacos (scannerTAD scanner);

/**/ Finaliza Boilerplate da Interface ***/

#endif
```