```c
/**
 * Arquivo: stackbuff.c
 * Versão : 1.0
 * Data   : 2024-10-18 17:27
 * ------------------------
 * Este arquivo implemeta a interface buffer.h, utilizando pilhas para o
 * armazenamento dos caracteres do buffer.
 *
 * Baseado em: Programming Abstractions in C, de Eric S. Roberts.
 *             Capítulo 9: Efficiency and ADTs (pg. 386-391).
 *
 * 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
 */

/*** Includes: ***/

#include "buffer.h"
#include "genlib.h"
#include <stdio.h>
#include <stdlib.h>
#include "simpio.h"
#include "stackTAD.h"
#include "strlib.h"

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

/**
 * Tipo: bufferTCD
 * ---------------
 * Nesta representação de buffer os caracteres são armazenados em uma de duas
 * pilhas. Os caracteres que estão ANTES do cursor são armazenados na pilha
 * "antes"; os caracteres que estão DEPOIS do cursor são armazenados na pilha
 * "depois". O cursor não é representado explicitamente: ele é mantido de
 * forma implícita, como o limite entre as duas pilhas. Um buffer com as letras
 * ABCDE, com o cursor entre as letras C e D, seria armazenado como:
 *
 *       C
 *       B        D
 *       A        E
 *     -----    ------
 *     antes    depois
 */

struct bufferTCD
{
    stackTAD antes;
    stackTAD depois;
};

/*** Definições de Subprogramas Exportados: ***/

/**
 * FUNÇÃO: criar_buffer
```

```
 * Uso: buffer = criar_buffer( );
 * ----------------------------
 * Esta função aloca memória de modo dinâmico, em quantidade suficiente para a
 * representação interna do bufferTAD, e inicializa o buffer para representar
 * um buffer vazio.
 */

bufferTAD criar_buffer (void)
{
    bufferTAD buffer = malloc(sizeof(struct bufferTCD));
    if (buffer == NULL)
    {
        fprintf(stderr, "Erro: impossível alocar buffer.\n");
        return NULL;
    }

    buffer->antes = criar_stackTAD();
    buffer->depois = criar_stackTAD();
    if (buffer->antes == NULL || buffer->depois == NULL)
    {
        fprintf(stderr, "Erro: impossível alocar pilhas do buffer.\n");
        return NULL;
    }

    return buffer;
}

/**
 * PROCEDIMENTO: liberar_buffer
 * Uso: liberar_buffer(buffer);
 * ----------------------------
 * Este procedimento libera o espaço de armazenamento alocado para o buffer. O
 * argumento deve ser um PONTEIRO para o buffer (um ponteiro para ponteiro para
 * struct bufferTCD).
 */

void liberar_buffer (bufferTAD *buffer)
{
    if (*buffer != NULL)
    {
        remover_stackTAD(&((*buffer)->antes));
        remover_stackTAD(&((*buffer)->depois));
        free(*buffer);
        buffer = NULL;
    }
}

/**
 * PROCEDIMENTOS: mover_cursor_para_frente
 *                mover_cursor_para_tras
 * Uso: mover_cursor_para_frente(buffer);
 *      mover_cursor_para_tras(buffer);
 * ---------------------------------------
 * Estes procedimentos movem o cursor para frente e para trás, no buffer, um
 * caractere por vez. Se "mover_cursor_para_frete" for chamada no final do
 * buffer, ou se "mover_cursor_para_tras" for chamada no início do buffer, os
 * procedimentos não têm efeito nenhum.
 */

void mover_cursor_para_frente (bufferTAD buffer)
```

```
{
    if (buffer == NULL)
    {
        fprintf(stderr, "Erro: movimentação em buffer null.\n");
        exit(1);
    }

    if (!vazia(buffer->depois))
        push(buffer->antes, pop(buffer->depois));
}

void mover_cursor_para_tras (bufferTAD buffer)
{
    if (buffer == NULL)
    {
        fprintf(stderr, "Erro: movimentação em buffer null.\n");
        exit(1);
    }

    if (!vazia(buffer->antes))
        push(buffer->depois, pop(buffer->antes));
}

/**
 * PROCEDIMENTOS: mover_cursor_para_final
 *                mover_cursor_para_inicio
 * Uso: mover_cursor_para_final(buffer);
 *      mover_cursor_para_inicio(buffer);
 * ---------------------------------------
 * Estes procedimentos movem o cursor apra o final ou para o início do buffer,
 * respectivamente.
 */

void mover_cursor_para_final (bufferTAD buffer)
{
    if (buffer == NULL)
    {
        fprintf(stderr, "Erro: movimentação em buffer null.\n");
        exit(1);
    }

    while (!vazia(buffer->depois))
        push(buffer->antes, pop(buffer->depois));
}

void mover_cursor_para_inicio (bufferTAD buffer)
{
    if (buffer == NULL)
    {
        fprintf(stderr, "Erro: movimentação em buffer null.\n");
        exit(1);
    }

    while (!vazia(buffer->antes))
        push(buffer->depois, pop(buffer->antes));
}

/**
 * PROCEDIMENTO: inserir_caractere
 * Uso: inserir_caractere(buffer, c);
```

```
 * ---------------------------------
 * Insere o caractere "c" no buffer "buffer", na posição atual do cursor. Após
 * a inserção o cursos é posicionado após o caractere inserido, para permitir
 * inserções consecutivas.
 */

void inserir_caractere (bufferTAD buffer, char c)
{
    if (buffer == NULL)
    {
        fprintf(stderr, "Erro: inserção em buffer null.\n");
        exit(1);
    }

    push(buffer->antes, c);
}

/**
 * PROCEDIMENTO: apagar_caractere
 * Uso: apagar_caractere(buffer);
 * ------------------------------
 * Apaga o caractere imediatamente posterior ao cursor. Se o cursor já está no
 * final do buffer, não causa nenhum efeito.
 */

void apagar_caractere (bufferTAD buffer)
{
    if (buffer == NULL)
    {
        fprintf(stderr, "Erro: remoção em buffer null.\n");
        exit(1);
    }

    if (!vazia(buffer->depois))
        (void) pop(buffer->depois);
}

/**
 * PROCEDIMENTO: exibir_buffer
 * Uso: exibir_buffer(buffer);
 * ---------------------------
 * Exibe o conteúdo atual do buffer no terminal.
 */

void exibir_buffer (bufferTAD buffer)
{
    for (int i = 0; i < qtd_elementos(buffer->antes); i++)
        printf(" %c", ver_elemento(buffer->antes, i));

    for (int i = qtd_elementos(buffer->depois) - 1; i >= 0; i--)
        printf(" %c", ver_elemento(buffer->depois, i));

    printf("\n");

    for (int i = 0; i < qtd_elementos(buffer->antes); i++)
        printf("  ");

    printf("^\n");
}
```