```c
/**
 * Arquivo: lsebuff.c
 * Versão : 1.0
 * Data   : 2024-10-27 16:55
 * ------------------------
 * Este arquivo implementa a interface buffer.h, utilizando uma lista encadeada
 * simples para o armazenamento dos caracteres do buffer.
 *
 * Baseado em: Programming Abstractions in C, de Eric S. Roberts.
 *             Capítulo 9: Efficiency and ADTs (pg. 391-407).
 *
 * 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 "strlib.h"

/*** Constantes Simbólicas ***/

/*** Variáveis Globais ***/

/*** Tipos de Dados ***/

/**
 * Tipo: celulaTCD, celulaTAD
 * --------------------------
 * Define uma célula (nó) de uma lista encadeada simples. Não há um verdadeiro
 * tipo abstrato aqui pois este tipo é usado apenas dentro da implementação, com
 * o tipo concreto visível. Mesmo assim é definido um tipo celulaTAD que é um
 * ponteiro para uma celulaTCD, para facilitar algumas operações. As operações
 * de criar e remover células são privadas a este arquivo, e estão definidas
 * mais abaixo.
 */

typedef struct celulaTCD
{
    char letra;
    struct celulaTCD *proximo;
} celulaTCD;

typedef struct celulaTCD *celulaTAD;

/**
 * Tipo: bufferTCD
 * ---------------
 * É a representação concreta para o TAD buffer, definido na interface. Nesta
 * representação usamos uma lista simplesmente encadeada para armazenar os
 * caracteres do buffer. Os elementos do buffer são:
```

```
 *
 *     inicio    : ponteiro para o início da lista
 *     cursor    : ponteiro para a posição atual do cursor
 *
 * Para simplificar as operações na lista esta imiplementação adota a estratégia
 * de manter uma "dummy cell" no início de cada lista, de forma que o buffer
 * vazio tem a seguinte representação:
 *
 *     +-------+          +------+
 *     | I o---+--------=>|      |      I: ponteiro início
 *     +-------+    /     +------+
 *     | C o---+---/      | NULL |      C: ponteiro cursor
 *     +-------+          +------+
 */

struct bufferTCD
{
    celulaTAD inicio;
    celulaTAD cursor;
};

/*** Declarações de Suprogramas Privados ***/

static celulaTAD criar_celula (void);
static void remover_celula (celulaTAD *celula);

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

/**
 * Função: criar_buffer
 * Uso: buffer = criar_buffer( );
 * ------------------------------
 * Esta função aloca um novo buffer vazio para o editor de texto, representado
 * internamente por uma lista encadeada.
 */

bufferTAD
criar_buffer (void)
{
    bufferTAD B = calloc(1, sizeof(struct bufferTCD));
    if (B == NULL)
    {
        fprintf(stderr, "Erro: impossível alocar buffer.\n");
        return NULL;
    }

    celulaTAD temp = criar_celula();
    if (temp == NULL)
    {
        fprintf(stderr, "Erro: a célula não foi criada.\n");
        free(B);
        B = NULL;
        return NULL;
    }

    B->inicio = B->cursor = temp;
    B->inicio->proximo = NULL;

    temp = NULL;
```

```
    return B;
}

/**
 * Procedimento: liberar_buffer
 * Uso: liberar_buffer(buffer);
 * ----------------------------
 * Este procedimento libera todas as células do buffer bem como o buffer em si.
 * Note que o loop não é exatamente o idioma padrão para processar todas as
 * células em uma lista encadeada, pois não é válido liberar uma célula e,
 * depois, verificar seu ponteiro "próximo". Para evitar visitar campos em uma
 * estrutura depois que ela foi liberada, temos que copiar o ponteiro antes de
 * liberar essa estrutura da memória.
 */

void
liberar_buffer (bufferTAD *buffer)
{
    if (buffer == NULL || *buffer == NULL)
    {
        fprintf(stderr, "Erro: buffer inválido.\n");
    }
    else
    {
        celulaTAD atual, proxima;
        atual = (*buffer)->inicio;
        while (atual != NULL)
        {
            proxima = atual->proximo;
            remover_celula(&atual);
            atual = proxima;
        }
        free(*buffer);
        *buffer = NULL;
    }
}

/**
 * Procedimento: inserir_caractere
 * Uso: inserir_caractere(buffer, c);
 * ----------------------------------
 * Insere o caractere 'c' no buffer "buffer", na posição indicada pelo cursor (o
 * cursor aponta para a CÉLULA IMEDIATAMENTE ANTES DA POSIÇÃO DO CURSOR). Após a
 * inserção do caractere os ponteiros são ajustados.
 */

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

    // 1: cria nova célula na memória e retorna um ponteiro para a essa célula:
    celulaTAD pc = criar_celula();
    if (pc == NULL)
    {
        fprintf(stderr, "Erro: impossível alocar célula.\n");
```

```
        exit(1);
    }

    // 2: copia o caractere para a nova célula:
    pc->letra = c;

    // 3: copia o "próximo" do cursor para o ponteiro "próximo" da nova célula:
    pc->proximo = buffer->cursor->proximo;

    // 4: altera o "próximo" do cursor para que apontar para o nova célula:
    buffer->cursor->proximo = pc;

    // 5: faz o cursor para apontar para a nova célula:
    buffer->cursor = pc;

    // 6: remove o ponteiro temporário (não estritamente necessário):
    pc = NULL;
}

/**
 * Procedimento: apagar_caractere
 * Uso: apagar_caractere(buffer);
 * ------------------------------
 * Recebe o buffer como argumento e remove o caractere apontado pelo cursor. A
 * operação é facilitada pelo uso da "dummy cell", que faz para o cursor apontar
 * para a célula imediatamente anterior à posição do cursor.
 */

void
apagar_caractere (bufferTAD buffer)
{
    celulaTAD temp;

    if (buffer->cursor->proximo != NULL)
    {
        temp = buffer->cursor->proximo;
        buffer->cursor->proximo = temp->proximo;
        remover_celula(&temp);
    }

    temp = NULL;
}

/**
 * Procedimentos: mover_cursor_para_frente
 *                mover_cursor_para_tras
 * Uso: mover_cursor_para_frente(buffer);
 *      mover_cursor_para_tras(buffer);
 * ---------------------------------------
 */

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

```
    if (buffer->cursor->proximo != NULL)
    {
        buffer->cursor = buffer->cursor->proximo;
    }
}

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

    celulaTAD temp;

    if (buffer->cursor != buffer->inicio)
    {
        temp = buffer->inicio;
        while (temp->proximo != buffer->cursor)
        {
            temp = temp->proximo;
        }
        buffer->cursor = temp;
    }
}

/**
 * Procedimentos: mover_cursor_para_inicio
 *                mover_cursor_para_final
 * Uso: mover_cursor_para_inicio(buffer);
 *      mover_cursor_para_final(buffer);
 * ----------------------------------------
 */

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

    buffer->cursor = buffer->inicio;
}

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

    while (buffer->cursor->proximo != NULL)
    {
        mover_cursor_para_frente(buffer);
```

```
    }
}

/**
 * Procedimento: exibir_buffer
 * Uso: exibif_buffer(buffer);
 * ---------------------------
 * Exite o conteúdo atual do buffer no terminal.
 */

void
exibir_buffer (bufferTAD buffer)
{
    if (buffer == NULL)
    {
        fprintf(stderr, "Erro: impressão de buffer null.\n");
    }
    else
    {
        celulaTAD tmp;

        for (tmp = buffer->inicio->proximo; tmp != NULL; tmp = tmp->proximo)
        {
            printf(" %c", tmp->letra);
        }
        printf("\n");
        for (tmp = buffer->inicio; tmp != buffer->cursor; tmp = tmp->proximo)
        {
            printf("  ");
        }
        printf("^\n");
    }
}

/*** Definições de Subprogramas Privados ***/

/**
 * Função: criar_celula
 * Uso: celulaTAD = criar_celula( );
 * ---------------------------------
 * Cria uma célula da lista encadeada e retorna um ponteiro para a célula. Em
 * caso de erro, retorna NULL.
 */

static celulaTAD
criar_celula (void)
{
    celulaTAD temp = calloc(1, sizeof(struct celulaTCD));
    if (temp == NULL)
    {
        fprintf(stderr, "Erro: impossível criar célular.\n");
        return NULL;
    }

    return temp;
}

/**
 * Procedimento: remover_celula
 * Uso: remover_celula(&celula);
```

```
 * ----------------------------
 * Recebe um ponteiro para uma celulaTAD e faz a liberação de memória dessa
 * célula. Se o ponteiro recebido for NULL, imprime uma mensagem em stderr.
 */

static void
remover_celula (celulaTAD *celula)
{
    if (celula == NULL || *celula == NULL)
    {
        fprintf(stderr, "Erro: não há celula para liberar.\n");
    }
    else
    {
        free(*celula);
        *celula = NULL;
    }
}
```