```c
/**
 * Arquivo: arraybuf.c
 * Versão : 1.0
 * Data   : 2024-10-19 07:13
 * -------------------------
 * Este arquivo implementa a interface buffer.h utilizando como estrutura de
 * dados principal um array. Nesta implementação o buffer terá tamanho limitado
 * por uma constante simbólica.
 *
 * Baseado em: Programming Abstractions in C, de Eric S. Roberts.
 *             Capítulo 9: Efficiency and ATDs (pg. 381-386).
 *
 * 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 "strlib.h"

/*** Constantes Simbólicas ***/

#define TAMBUFFER 100

/*** Tipos de Dados ***/

/**
 * Tipo: struct bufferTCD
 * ----------------------
 * Nesta representação interna do buffer os caracteres estão armazenados em um
 * array. Também existirão variáveis para armazenar o tamanho da string do
 * buffer (o que nos permite ignorar o '\0' como marcador de final) e a posição
 * do cursor no buffer. A posição do cursor indicará o índice da posição no
 * array onde o próximo caractere será inserido.
 *
 *     texto      array com tamanho TAMBUFFER para os caracteres
 *     tamanho    quantidade de caracteres no buffer
 *     cursor     posição atual do cursor de edição
 */

struct bufferTCD
{
    char texto[TAMBUFFER];
    int tamanho;
    int cursor;
};

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

```c
/**
 * Função: criar_buffer
 * Uso: buffer = criar_buffer( );
 * ------------------------------
 * Cria e retorna um novo bufferTAD. Se a memória não puder ser alocada para o
 * buffer, retorna NULL.
 */

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

    B->tamanho = 0;
    B->cursor = 0;

    return B;
}

/**
 * Procedimento: liberar_buffer
 * Uso: liberar_buffer(buffer);
 * ----------------------------
 * Libera a memória alocada para um buffer. Recebe um PONTEIRO para um buffer,
 * ou seja, um ponteiro para um ponteiro para struct bufferTCD.
 */

void liberar_buffer (bufferTAD *buffer)
{
    if (*buffer != NULL)
    {
        free(*buffer);
        *buffer = NULL;
    }
}

/**
 * Procedimentos: mover_cursor_para_frente
 *                mover_cursor_para_tras
 * Uso: mover_cursor_para_frente(buffer);
 *      mover_cursor_para_tras(buffer);
 * ---------------------------------------
 * Movimentam o cursor para frete e para trás, um caractere de cada vez.
 */

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

    if (buffer->cursor < buffer->tamanho)
        buffer->cursor++;
}
```

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

    if (buffer->cursor > 0)
        buffer->cursor--;
}

/**
 * Procedimentos: mover_cursor_para_final
 *                mover_cursor_para_inicio
 * Uso: mover_cursor_para_final(buffer);
 *      mover_cursor_para_inicio(buffer);
 * ----------------------------------------
 * Movem o cursor para o início e para o final do buffer.
 */

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

    buffer->cursor = buffer->tamanho;
}

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

    buffer->cursor = 0;
}

/**
 * Procedimento: inserir_caractere
 * Uso: inserir_caractere(buffer, c);
 * ----------------------------------
 * Insere o caractere "c" no buffer "buffer", na posição atual do cursor,
 * deslocando os caracteres após a posição de inserção 1 posição para a
 * direita. Se ocorrer buffer overflow, ocorrerá um erro.
 */

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

```c
    else if (buffer->cursor == TAMBUFFER)
    {
        fprintf(stderr, "Erro: buffer overflow.\n");
        exit(1);
    }

    for (int i = buffer->tamanho; i > buffer->cursor; i--)
        buffer->texto[i] = buffer->texto[i - 1];

    buffer->texto[buffer->cursor] = c;
    buffer->tamanho++;
    buffer->cursor++;
}

/**
 * Procedimento: apagar_caractere
 * Uso: apagar_caractere(buffer);
 * ------------------------------
 * Apaga o caractere imediatamente posterior ao cursor.
 */

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

    if (buffer->cursor < buffer->tamanho)
    {
        for (int i = buffer->cursor + 1; i < buffer->tamanho; i++)
            buffer->texto[i - 1] = buffer->texto[i];
        buffer->tamanho--;
    }
}

/**
 * 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 < buffer->tamanho; i++)
        printf(" %c", buffer->texto[i]);

    printf("\n");

    for (int i = 0; i < buffer->cursor; i++)
        printf("  ");
    printf("^\n");
}
```