```c
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <SDL2/SDL.h>

// Declarações globais:
bool esta_rodando = false;                      // status de execução do programa

SDL_Window *janela = NULL;                      // ponteiro para uma janela
int largura = 800;                              // largura da janela
int altura = 600;                               // altura da janela

SDL_Renderer *renderizador = NULL;              // ponteiro para um renderizador

uint32_t *framebuffer = NULL;                   // ponteiro para o framebuffer
SDL_Texture *textura = NULL;                    // textura para o framebuffer


// Protótipos dos subprogramas:
bool inicializar_sdl (void);            // inicializa uma janela
bool configurar (void);                 // setup inicial da aplicação
void processar (void);                  // recebe e processa inputs do usuário
void atualizar (void);                  // atualiza o estado do programa
void renderizar (void);                 // renderiza a aplicação
void finalizar_sdl (void);              // faz a limpeza de estruturas da memória

void atualizar_textura (void);            // atualiza/copia textura->renderizador
void limpar_framebuffer (uint32_t cor);  // limpa o framebuffer



/**
 * MAIN
 */
int main(void)
{
    // Inicializa o SDL e garante que a rotina de finalização do SDL seja
    // chamada quando o programa for encerrado:
    esta_rodando = inicializar_sdl();
    atexit(finalizar_sdl);

    // Configura o framebuffer e cria textura para o renderizador:
    if(!configurar())
    {
        finalizar_sdl();
        fprintf(stderr, "Erro na configuração do ambiente.\n");
        return 1;
    }

    // Inicia o loop do programa:
    while (esta_rodando)
    {
        processar();
        atualizar();
        renderizar();
    }

    // Ao terminar o loop, finaliza o ambiente SDL:
    finalizar_sdl();
```

```
    // Retorno do status final:
    return 0;
}


/**
 * INICIALIZAR_SDL
 * Inicializa a biblioteca SDL, cria uma janela e um contexto de renderização
 * para a janela.
 *
 * Parâmetros:
 *    (void): a função não recebe nada.
 *
 * Retorno:
 *    (bool): TRUE se a inicialização foi realizada, FALSE caso contrário.
 *
 * Efeitos colaterais:
 *    a) Atribui janela para a variável SDL_Window *janela;
 *    b) Atribui contexto de renderização para SDL_Renderer *renderizador.
 */
bool inicializar_sdl (void)
{
    // Inicializa a biblioteca SDL, passando uma lista OR com as flags que
    // indicam quais sub-sistemas inicializar.
    // Doc: https://wiki.libsdl.org/SDL2/SDL_Init
    if (SDL_Init(SDL_INIT_EVERYTHING) != 0)
    {
        fprintf(stderr, "Erro na inicialização do SDL.\n");
        return false;
    }

    // Cria uma janela SDL com uma posição, tamanho e flags. A função que cria
    // a janela tem 6 parâmetros: título, x, y, w, h, flags. Se o título for
    // null, a janela não terá um título.
    // Doc: https://wiki.libsdl.org/SDL2/SDL_CreateWindow
    janela = SDL_CreateWindow(NULL,
                              SDL_WINDOWPOS_CENTERED,
                              SDL_WINDOWPOS_CENTERED,
                              largura,
                              altura,
                              SDL_WINDOW_BORDERLESS);
    if (!janela)
    {
        fprintf(stderr, "Erro na criação da janela SDL.\n");
        return false;
    }

    // Cria um contexto de renderização 2D para uma janela específica. É o
    // contexto de renderização que acompanha uma janela. Temos que passar
    // o ponteiro para a janela, o display onde a janela será exibida (-1
    // significa o display padrão), e as flags (0 significa que não tem
    // nenhuma flag especial).
    // Doc: https://wiki.libsdl.org/SDL2/SDL_CreateRenderer
    renderizador = SDL_CreateRenderer(janela, -1, 0);
    if (!renderizador)
    {
        fprintf(stderr, "Erro na criação do renderizador SDL.\n");
    }

    // Se chegamos aqui, conseguimos inicializar o SDL, criamos uma
```

```
    // janela e acoplamos um renderizador para essa janela.
    return true;
}


/**
 * CONFIGURAR
 * Cria o framebuffer e uma textura para um contexto de renderização.
 *
 * Parâmetros:
 *    (void): a função não recebe nada.
 *
 * Retorno:
 *    (bool): TRUE se a configuração foi feita, FALSE caso contrário.
 *
 * Efeitos colaterais:
 *    a) Atribui array para a variável uint32_t *framebuffer;
 *    b) Atribui textura de um renderizador na variável
 *       SDL_Texture *texture.
 */
bool configurar (void)
{
    // Cria o framebuffer, uma área que descerve a imagem da tela onde cada
    // pixel corresponde a uma localização na memória, como uma matriz de cores
    // de pixels com tamanho largura x altura, armazenada em um array por
    // prioridade de linha. Para acessar um determinado pixel, fazer:
    //    (largura * linha) + coluna
    framebuffer = (uint32_t *) malloc(sizeof(uint32_t) * largura * altura);
    if (!framebuffer)
    {
        fprintf(stderr, "Erro na alocação do framebuffer.\n");
        return false;
    }

    // Cria uma textura para um contexto de renderização.
    // Doc: https://wiki.libsdl.org/SDL2/SDL_CreateTexture
    textura = SDL_CreateTexture(
        renderizador,
        SDL_PIXELFORMAT_ARGB8888,
        SDL_TEXTUREACCESS_STREAMING,
        largura,
        altura);
    if (!textura)
    {
        fprintf(stderr, "Erro na criação da textura SDL.\n");
        return false;
    }

    // Se tudo foi configurado corretamente, retorna true:
    return true;
}


/**
 * FINALIZAR_SDL
 * Finaliza o ambiente SDL, limpando as estruturas de memória que foram criadas
 * pelo SDL ou pelo programador.
 *
 * Parâmetros:
 *    (void): o procedimento não recebe nada.
```

```
 *
 * Retorno:
 *     (void): não retorna nada.
 */
void finalizar_sdl (void)
{
    // Limpa áreas de memória:
    if (framebuffer)
    {
        free(framebuffer);
        framebuffer = NULL;
    }

    // Finaliza o ambiente SDL.
    // Docs: https://wiki.libsdl.org/SDL2/SDL_DestroyTexture
    //       https://wiki.libsdl.org/SDL2/SDL_DestroyRenderer
    //       https://wiki.libsdl.org/SDL2/SDL_DestroyWindow
    //       https://wiki.libsdl.org/SDL2/SDL_Quit
    SDL_DestroyTexture(textura);
    SDL_DestroyRenderer(renderizador);
    SDL_DestroyWindow(janela);
    SDL_Quit();
}


/**
 * PROCESSAR
 * Faz o processamento de todos os inputs do jogo.
 *
 * Parâmetros:
 *     (void): o procedimento não recebe nada.
 *
 * Retorno:
 *     (void): o procedimento não retorna nada.
 */
void processar (void)
{
    // Structure a ser preenchida com os próximos eventos da fila, ou NULL:
    SDL_Event evento;

    // Pool para os eventos pendentes atuais:
    // Doc: https://wiki.libsdl.org/SDL2/SDL_PollEvent
    SDL_PollEvent(&evento);

    // Testa o evento recebido:
    switch (evento.type)
    {
        case SDL_QUIT:                                  // botão x da janela
            esta_rodando = false;
            break;
        case SDL_KEYDOWN:                               // tecla ESC
            if (evento.key.keysym.sym == SDLK_ESCAPE)
                esta_rodando = false;
            break;
    }
}


// TODO
void atualizar (void)
```

```
{
    ;
}


/**
 * RENDERIZAR
 * Exibe as imagens na janela, frame a frame. Faz isso através dos seguintes
 * passos: limpeza do renderizador, ajuste da cor de desenho do renderizador,
 * limpeza do framebuffer, atualização da textura com os dados do framebuffer e
 * cópia da textura para o renderizador, e, finalmente, a renderização da
 * imagem na janela.
 *
 * Parâmetros:
 *    (void): o procedimento não recebe nada.
 *
 * Retorno:
 *    (tipo): o procedimento não recebe nada.
 */
void renderizar (void)
{
    // Limpa o renderizador atual.
    // Doc: https://wiki.libsdl.org/SDL2/SDL_RenderClear
    SDL_RenderClear(renderizador);

    // Cor utilizada para as operações de desenho do renderizador atual (Rect,
    // Line, Clear) do renderizador atual. Tem 5 parâmetros: o renderer, os
    // valores RGB e o valor do alpha (transparência).
    // Doc: https://wiki.libsdl.org/SDL2/SDL_SetRenderDrawColor
    SDL_SetRenderDrawColor(renderizador, 255, 0, 0, 255);

    // Limpa o framebuffer passando uma cor padrão.
    limpar_framebuffer(0xFFFFFF00);

    // Atualiza a textura com os dados do framebuffer e copia a textura
    // para o renderizador de destino:
    atualizar_textura();

    // E agora, finalmente, vamos renderizar:
    SDL_RenderPresent(renderizador);
}


/**
 * ATUALIZAR_TEXTURA
 * Atualiza a textura atual com os novos dados dos pixels (armazenados no
 * framebuffer), e copia essa textura atualizada para o renderizador de
 * destino.
 *
 * Parâmetros:
 *    (void): o procedimento não recebe nada.
 *
 * Retorno:
 *    (void): o procedimento não retorna nada.
 *
 * Efeitos colaterais:
 *    a) Atualiza a textura em SDL_Texture *textura;
 *    b) Atualiza o renderizador em SDL_Renderer *renderizador.
 */
void atualizar_textura (void)
```

```
{
    // Atualiza uma dada textura (ou área retangular dessa textura) com os novos
    // dados dos pixels (armazenados no framebuffer).
    // Doc: https://wiki.libsdl.org/SDL2/SDL_UpdateTexture
    SDL_UpdateTexture(
        textura,
        NULL,
        framebuffer,
        (int) (largura * sizeof(uint32_t)));

    // Copia uma parte (ou toda) da textura para o alvo de renderização atual.
    // Doc: https://wiki.libsdl.org/SDL2/SDL_RenderCopy
    SDL_RenderCopy(
        renderizador,
        textura,
        NULL, NULL);
}


/**
 * LIMPAR_FRAMEBUFFER
 * Limpa o framebuffer preenchendo-o com uma cor qualquer.
 *
 * Parâmetros:
 *    (uint32_t): um número inteiro em hexadeimal no formato 0xAARRGGBB, onde AA
 *                é o valor do Alpha, RR o valor de Red, GG o valor de Green e
 *                BB o valor de Blue (variando de 00 até FF).
 *
 * Retorno:
 *    (void): o procedimento não retorna nada.
 *
 * Efeitos colaterais:
 *    Zerar o framebuffer com uma cor específica.
 */
void limpar_framebuffer (uint32_t cor)
{
    for (int l = 0; l < altura; l++)
    {
        for (int c = 0; c < largura; c++)
        {
            framebuffer[largura * l + c] = cor;
        }
    }
}
```