Балтийский государственный технический университет
«ВОЕНМЕХ» им. Д. Ф. Устинова

Факультет «О» Естественнонаучный

Кафедра О7 «Информационные системы и программная инженерия»

Практическая работа №5
по дисциплине «Программирование на ЯВУ»
на тему «SDL»

Выполнил:
Студент Костров Г. Ю.
Группа О712Б
Преподаватель: Васюков В.М.

Санкт-Петербург
2022 г

Постановка задачи:

Построить график функции

y = exp(-2) / (x-5)

при x€[-5,3]

Изобразить на экране движение корабля по морю с помощью клавиш управления курсором: при нажатии на клавишу «>» начинается движение вправо, при нажатии на клавишу «<» – влево, при нажатии на клавишу «^» корабль останавливается.

Текст программы:

// Задание 1

```
#include "SDL.h"
#include "stdio.h"
#include "iostream"
#include "math.h"
#include "cmath"


#define SCREEN_WIDTH 1280
#define SCREEN_HEIGHT 720

float map(float minX, float maxX, float minPxX, float maxPxX, float x)
{
    return (x - minX) / (maxX - minX) * (maxPxX - minPxX) + minPxX;
}

int main(int argc, char **argv)
{
    SDL_Init(SDL_INIT_EVERYTHING);
    const auto Width = 1280;
    const auto Heigth = 720;
    SDL_Window *w = SDL_CreateWindow("Graphs", 63, 126, Width, Heigth,
SDL_WINDOW_BORDERLESS);
    SDL_Renderer *r = SDL_CreateRenderer(w, -1, SDL_RENDERER_ACCELERATED);
    for (auto done = false; !done;)
    {
        SDL_Event e;
        while (SDL_WaitEvent(&e))
        {
            switch (e.type)
            {
            case SDL_QUIT:
                done = true;
                break;
            }
            SDL_SetRenderDrawColor(r, 0xff, 0xff, 0xff, 0xff);
            SDL_RenderClear(r);
            SDL_SetRenderDrawColor(r, 0xee, 0xee, 0xee, 0xff);

            // Линии, которые делят окно на квадратики)
            {
            for (float x = -10; x < 10; ++x) {
```

```
                SDL_RenderDrawLine(r,
                                   map(-10, 10, 0, Width, x), map(-5, 5, Heigth,
0, -5),
                                   map(-10, 10, 0, Width, x), map(-5, 5, Heigth,
0, +5));
            }
            for (float y = -5; y < 5; ++y) {
                SDL_RenderDrawLine(r,
                                   map(-10, 10, 0, Width, -10), map(-5, 5, Heigth,
0, y),
                                   map(-10, 10, 0, Width, 10), map(-5, 5, Heigth,
0, y));
            }
            }

            // Ось абцисс и ординат + функция
            {
                SDL_SetRenderDrawColor(r, 0xee, 0xc0, 0xc0, 0xff);
                SDL_RenderDrawLine(r,
                                   map(-10, 10, 0, Width, -10), map(-5, 5, Heigth, 0,
0),
                                   map(-10, 10, 0, Width, 10), map(-5, 5, Heigth, 0,
0));
                SDL_RenderDrawLine(r,
                                   map(-10, 10, 0, Width, 0), map(-5, 5, Heigth, 0, -
5),
                                   map(-10, 10, 0, Width, 0), map(-5, 5, Heigth, 0,
+5));
                SDL_SetRenderDrawColor(r, 0x00, 0x00, 0x00, 0xff);
            }

            auto oldX = -1;
            auto oldY = 0;
            for (float x = -5; x <= 3; x += 0.01) // x от -5 до 3 по условию
задачи
            {
                auto y = exp(-2) / (x - 5); // ТУТ МОЖНО ИЗМЕНЯТЬ ФУНКЦИЮ
                auto newX = map(-10, 10, 0, Width, x);
                auto newY = map(-5, 5, Heigth, 0, y);
                if (oldX >= 0)
                    SDL_RenderDrawLine(r, oldX, oldY, newX, newY);
                oldX = newX;
                oldY = newY;
            }
            SDL_RenderPresent(r);
        }
        SDL_Quit();
    }
    return 0;
}
```

// Задание 2

```
//Using SDL, SDL_image, SDL_ttf, standard IO, strings, and string streams
#include <SDL.h>
#include <SDL_image.h>
#include <SDL_ttf.h>
#include <cstdio>
#include <string>
#include <sstream>
#include <vector>

//Screen dimension constants
const int SCREEN_WIDTH = 1280;
```

```
const int SCREEN_HEIGHT = 960;

//Texture wrapper class
class LTexture {
public:
    //Initializes variables
    LTexture();

    //Deallocates memory
    ~LTexture();

    //Loads image at specified path
    bool loadFromFile(std::string path);

#if defined(SDL_TTF_MAJOR_VERSION)

    //Creates image from font string
    bool loadFromRenderedText(std::string textureText, SDL_Color textColor);

#endif

    //Deallocates texture
    void free();

    //Set color modulation
    void setColor(Uint8 red, Uint8 green, Uint8 blue);

    //Set blending
    void setBlendMode(SDL_BlendMode blending);

    //Set alpha modulation
    void setAlpha(Uint8 alpha);

    //Renders texture at given point
    void render(int x, int y, SDL_Rect *clip = NULL, double angle = 0.0, SDL_Point
*center = NULL,
                SDL_RendererFlip flip = SDL_FLIP_NONE);

    //Gets image dimensions
    int getWidth();

    int getHeight();

private:
    //The actual hardware texture
    SDL_Texture *mTexture;

    //Image dimensions
    int mWidth;
    int mHeight;
};

//Starts up SDL and creates window
bool init();

//Loads media
bool loadMedia();

//Frees media and shuts down SDL
void close();

//The window we'll be rendering to
SDL_Window *gWindow = NULL;
```

```
//The window renderer
SDL_Renderer *gRenderer = NULL;

//Globally used font
TTF_Font *gFont = NULL;

//Scene textures
LTexture gPromptTextTexture;
LTexture gInputTextTexture;

std::string nameOfFilesForClock[4] = {"clock.bmp", "hour.bmp", "minute.bmp",
"second.bmp"};

std::vector<LTexture> clockTextures(4);

LTexture::LTexture() {
    //Initialize
    mTexture = NULL;
    mWidth = 0;
    mHeight = 0;
}

LTexture::~LTexture() {
    //Deallocate
    free();
}

bool LTexture::loadFromFile(std::string path) {
    //Get rid of preexisting texture
    free();

    //The final texture
    SDL_Texture *newTexture = NULL;

    //Load image at specified path
    SDL_Surface *loadedSurface = IMG_Load(path.c_str());
    if (loadedSurface == NULL) {
        printf("Unable to load image %s! SDL_image Error: %s\n", path.c_str(),
IMG_GetError());
    } else {
        //Color key image
        SDL_SetColorKey(loadedSurface, SDL_TRUE, SDL_MapRGB(loadedSurface->format,
0, 0xFF, 0xFF));

        //Create texture from surface pixels
        newTexture = SDL_CreateTextureFromSurface(gRenderer, loadedSurface);
        if (newTexture == NULL) {
            printf("Unable to create texture from %s! SDL Error: %s\n",
path.c_str(), SDL_GetError());
        } else {
            //Get image dimensions
            mWidth = loadedSurface->w;
            mHeight = loadedSurface->h;
        }

        //Get rid of old loaded surface
        SDL_FreeSurface(loadedSurface);
    }

    //Return success
    mTexture = newTexture;
    return mTexture != NULL;
```

```
}

#if defined(SDL_TTF_MAJOR_VERSION)

bool LTexture::loadFromRenderedText(std::string textureText, SDL_Color textColor)
{
    //Get rid of preexisting texture
    free();

    //Render text surface
    SDL_Surface *textSurface = TTF_RenderText_Solid(gFont, textureText.c_str(),
textColor);
    if (textSurface != NULL) {
        //Create texture from surface pixels
        mTexture = SDL_CreateTextureFromSurface(gRenderer, textSurface);
        if (mTexture == NULL) {
            printf("Unable to create texture from rendered text! SDL Error: %s\n",
SDL_GetError());
        } else {
            //Get image dimensions
            mWidth = textSurface->w;
            mHeight = textSurface->h;
        }

        //Get rid of old surface
        SDL_FreeSurface(textSurface);
    } else {
        printf("Unable to render text surface! SDL_ttf Error: %s\n",
TTF_GetError());
    }


    //Return success
    return mTexture != NULL;
}

#endif

void LTexture::free() {
    //Free texture if it exists
    if (mTexture != NULL) {
        SDL_DestroyTexture(mTexture);
        mTexture = NULL;
        mWidth = 0;
        mHeight = 0;
    }
}

void LTexture::setColor(Uint8 red, Uint8 green, Uint8 blue) {
    //Modulate texture rgb
    SDL_SetTextureColorMod(mTexture, red, green, blue);
}

void LTexture::setBlendMode(SDL_BlendMode blending) {
    //Set blending function
    SDL_SetTextureBlendMode(mTexture, blending);
}

void LTexture::setAlpha(Uint8 alpha) {
    //Modulate texture alpha
    SDL_SetTextureAlphaMod(mTexture, alpha);
}
```

```
void LTexture::render(int x, int y, SDL_Rect *clip, double angle, SDL_Point
*center, SDL_RendererFlip flip) {
    //Set rendering space and render to screen
    SDL_Rect renderQuad = {x, y, mWidth, mHeight};

    //Set clip rendering dimensions
    if (clip != NULL) {
        renderQuad.w = clip->w;
        renderQuad.h = clip->h;
    }

    //Render to screen
    SDL_RenderCopyEx(gRenderer, mTexture, clip, &renderQuad, angle, center, flip);
}

int LTexture::getWidth() {
    return mWidth;
}

int LTexture::getHeight() {
    return mHeight;
}

bool init() {
    //Initialization flag
    bool success = true;

    //Initialize SDL
    if (SDL_Init(SDL_INIT_VIDEO) < 0) {
        printf("SDL could not initialize! SDL Error: %s\n", SDL_GetError());
        success = false;
    } else {
        //Set texture filtering to linear
        if (!SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "1")) {
            printf("Warning: Linear texture filtering not enabled!");
        }

        //Create window
        gWindow = SDL_CreateWindow("Time", SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED, SCREEN_WIDTH,
                                   SCREEN_HEIGHT, SDL_WINDOW_SHOWN);
        if (gWindow == NULL) {
            printf("Window could not be created! SDL Error: %s\n",
SDL_GetError());
            success = false;
        } else {
            //Create vsynced renderer for window
            gRenderer = SDL_CreateRenderer(gWindow, -1, SDL_RENDERER_ACCELERATED |
SDL_RENDERER_PRESENTVSYNC);
            if (gRenderer == NULL) {
                printf("Renderer could not be created! SDL Error: %s\n",
SDL_GetError());
                success = false;
            } else {
                //Initialize renderer color
                SDL_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0xFF);

                //Initialize PNG loading
                int imgFlags = IMG_INIT_PNG;
                if (!(IMG_Init(imgFlags) & imgFlags)) {
                    printf("SDL_image could not initialize! SDL_image Error:
%s\n", IMG_GetError());
                    success = false;
```

```
                }

                //Initialize SDL_ttf
                if (TTF_Init() == -1) {
                    printf("SDL_ttf could not initialize! SDL_ttf Error: %s\n",
TTF_GetError());
                    success = false;
                }
            }
        }
    }

    return success;
}

bool loadMedia() {
    //Loading success flag
    bool success = true;

    //Open the font
    gFont = TTF_OpenFont("courier.ttf", 28);
    if (gFont == NULL) {
        printf("Failed to load font! SDL_ttf Error: %s\n", TTF_GetError());
        success = false;
    } else {
        //Render the prompt
        SDL_Color textColor = {0, 0, 0, 0xFF};
        if (!gPromptTextTexture.loadFromRenderedText("Enter time in format **:**
:", textColor)) {
            printf("Failed to render prompt text!\n");
            success = false;
        }
    }
    for (int i = 0; i < 4; i++) {
        if (!clockTextures.at(i).loadFromFile(nameOfFilesForClock[i])) {
            printf("Failed to load texture!\n");
            success = false;
        }
    }
    return success;
}

void close() {
    //Free loaded images
    gPromptTextTexture.free();
    gInputTextTexture.free();

    //Free global font
    TTF_CloseFont(gFont);
    gFont = NULL;

    //Destroy window
    SDL_DestroyRenderer(gRenderer);
    SDL_DestroyWindow(gWindow);
    gWindow = NULL;
    gRenderer = NULL;

    //Quit SDL subsystems
    TTF_Quit();
    IMG_Quit();
    SDL_Quit();
}
```

```
int main(int argc, char *args[]) {
    //Start up SDL and create window
    if (!init()) {
        printf("Failed to initialize!\n");
    } else {
        //Load media
        if (!loadMedia()) {
            printf("Failed to load media!\n");
        } else {
            //Main loop flag
            bool quit = false;

            //Event handler
            SDL_Event e;

            //Set text color as black
            SDL_Color textColor = {0, 0, 0, 0xFF};

            //The current input text.
            std::string inputText = "text";
            gInputTextTexture.loadFromRenderedText(inputText.c_str(), textColor);

            //Enable text input
            SDL_StartTextInput();

            int hour, minute, second;
            int angleOfSeconds = 0;
            bool inputTime = false;
            //While application is running
            while (!quit) {
                //The rerender text flag
                bool renderText = false;

                //Handle events on queue
                while (SDL_PollEvent(&e) != 0) {
                    //User requests quit
                    if (e.type == SDL_QUIT) {
                        quit = true;
                    }
                        //Special key input
                    else if (e.type == SDL_KEYDOWN) {
                        //Handle backspace
                        if (e.key.keysym.sym == SDLK_BACKSPACE &&
inputText.length() > 0) {
                            //lop off character
                            inputText.pop_back();
                            renderText = true;
                        }
                            //Handle copy
                        else if (e.key.keysym.sym == SDLK_c && SDL_GetModState() &
KMOD_CTRL) {
                            SDL_SetClipboardText(inputText.c_str());
                        }
                            //Handle paste
                        else if (e.key.keysym.sym == SDLK_v && SDL_GetModState() &
KMOD_CTRL) {
                            inputText = SDL_GetClipboardText();
                            renderText = true;
                        }
                    }
                        //Special text input event
                    else if (e.type == SDL_TEXTINPUT) {
                        //Not copy or pasting
```

```
                        if (!(SDL_GetModState() & KMOD_CTRL &&
                              (e.text.text[0] == 'c' || e.text.text[0] == 'C' ||
e.text.text[0] == 'v' ||
                               e.text.text[0] == 'V'))) {
                            //Append character
                            inputText += e.text.text;
                            renderText = true;
                        }
                    }
                    if (e.type == SDL_KEYUP && e.key.repeat == 0 &&
e.key.keysym.sym == SDLK_RETURN &&
                        !inputText.empty() && (inputText.length() == 5)) {
                        if (std::isdigit(inputText[0]) &&
std::isdigit(inputText[1]) && std::isdigit(inputText[3]) &&
std::isdigit(inputText[4]) && inputText[2] == ':') {
                            if ((std::stoi(inputText.substr(0, 2)) < 24) &&
(std::stoi(inputText.substr(3, 2)) < 60)) {
                                hour = std::stoi(inputText.substr(0, 2));
                                minute = std::stoi(inputText.substr(3, 2));
                                if (hour >= 12)
                                    hour -= 12;
                                inputTime = true;
                                SDL_StopTextInput();
                            }
                        }
                    }
                }
                //Clear screen
                SDL_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0xFF);
                SDL_RenderClear(gRenderer);
                if (!inputTime) {
                    //Rerender text if needed
                    if (renderText) {
                        //Text is not empty
                        if (!inputText.empty()) {
                            //Render new text
                            gInputTextTexture.loadFromRenderedText(inputText,
textColor);
                        }
                            //Text is empty
                        else {
                            //Render space texture
                            gInputTextTexture.loadFromRenderedText(" ",
textColor);
                        }
                    }
                    //Render text textures
                    gPromptTextTexture.render((SCREEN_WIDTH -
gPromptTextTexture.getWidth()) / 2, 0);
                    gInputTextTexture.render((SCREEN_WIDTH -
gInputTextTexture.getWidth()) / 2,
                                             gPromptTextTexture.getHeight());
                } else {
                    clockTextures.at(0).render((SCREEN_WIDTH -
clockTextures.at(0).getWidth()) / 2, (SCREEN_HEIGHT -
clockTextures.at(0).getHeight()) / 2);
                    clockTextures.at(3).render((SCREEN_WIDTH -
clockTextures.at(0).getWidth()) / 2 + 195, (SCREEN_HEIGHT -
clockTextures.at(0).getHeight()) / 2 + 20,
                                               nullptr, angleOfSeconds++);
                    if (angleOfSeconds % 360 == 0)
                        minute += 1;
                    clockTextures.at(1).render((SCREEN_WIDTH -
```

```
clockTextures.at(0).getWidth()) / 2 + 200, (SCREEN_HEIGHT -
clockTextures.at(0).getHeight()) / 2 + 120, nullptr, hour * 30);
                    clockTextures.at(2).render((SCREEN_WIDTH -
clockTextures.at(0).getWidth()) / 2 + 200, (SCREEN_HEIGHT -
clockTextures.at(0).getHeight()) / 2 + 80, nullptr, minute * 6);

                }

                //Update screen
                SDL_RenderPresent(gRenderer);
            }

            //Disable text input
            SDL_StopTextInput();
        }
    }

    //Free resources and close SDL
    close();

    return 0;
}
```

Результат работы программы:

Рисунок 1:Задание 1.

Рисунок 2: Задание 2.

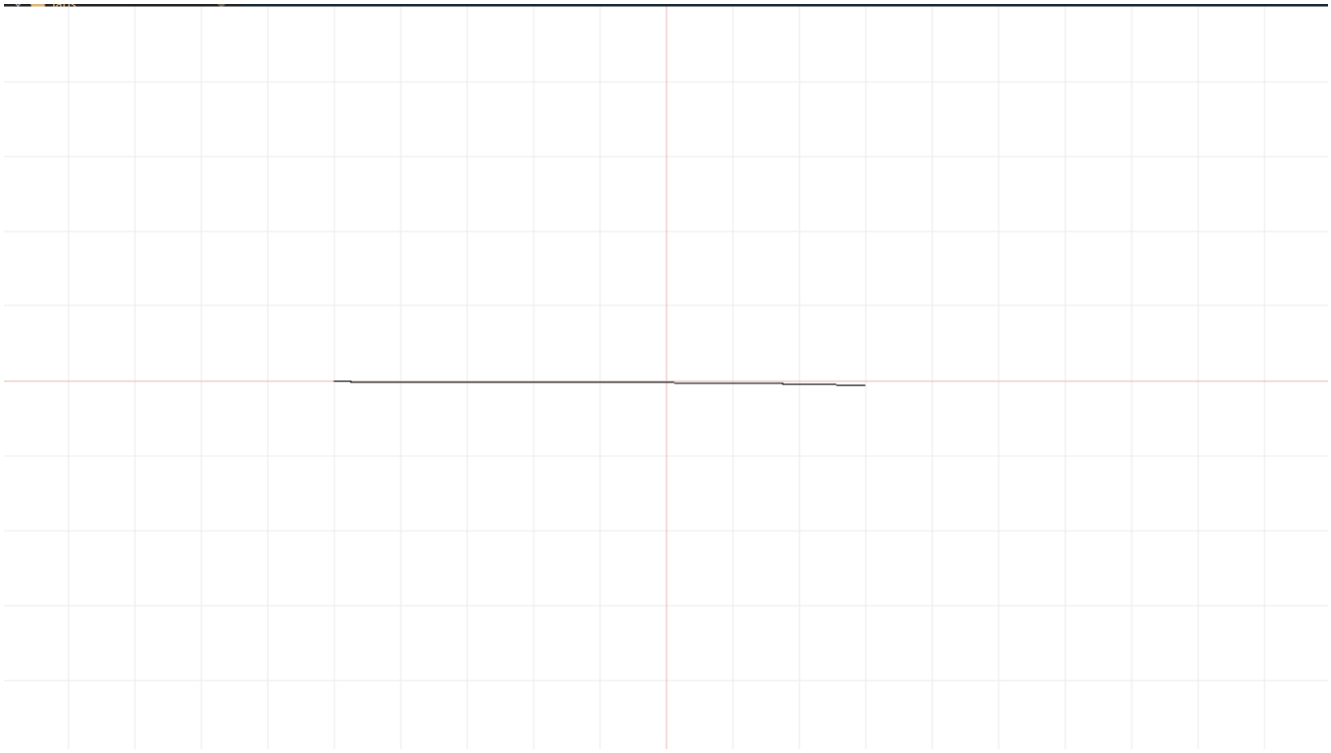

Рисунок 1

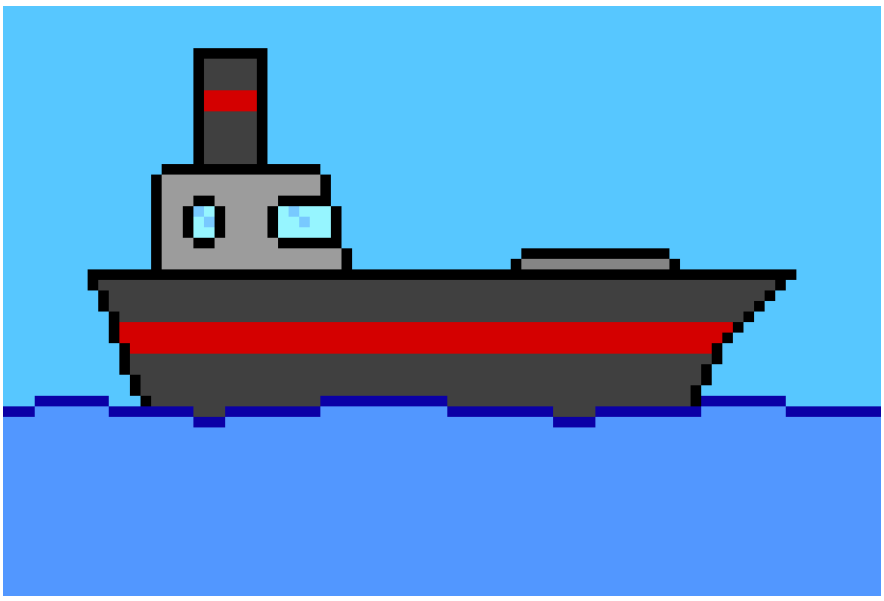

Рисунок 2