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

Кафедра И5 «Информационные системы и программная инженерия»

Лабораторная работа №5

по дисциплине «Программирование на языке высокого уровня»
по теме «Графический режим ввода-вывода в языках C и C++ с использованием SDL»

Выполнил:
Студент Альков В. С.
Группа И407Б

Преподаватель:
Кимсанбаев К. А.

Санкт-Петербург
2021 г.

Задача: Написать две программы согласно номеру индивидуального варианта. В первой построить график функции на интервале, соответствующем выбранному уровню сложности: низкий – на указанном в задании интервале, средний – на интервале, заданном с клавиатуры, но не включающем точек разрыва функции, высокий – на произвольном интервале. Масштаб по осям координат и положение осей определить так, чтобы график занимал весь экран. Приращение аргумента выбрать таким, чтобы непрерывные участки функции отображались плавной кривой.

Во второй программе смоделировать непрерывное движение заданного объекта, выход из программы осуществлять при нажатии клавиши «Esc».

1. Построить график функции $y = \tan\left(x - \frac{\pi}{3}\right)$ при $x \in \left[0, \frac{2\pi}{3}\right]$.

2. Написать программу движения окружности в равнобедренном прямоугольном треугольнике, катеты которого параллельны границам экрана. Движение происходит под некоторым углом с «отражением от стенки».

Задание 1

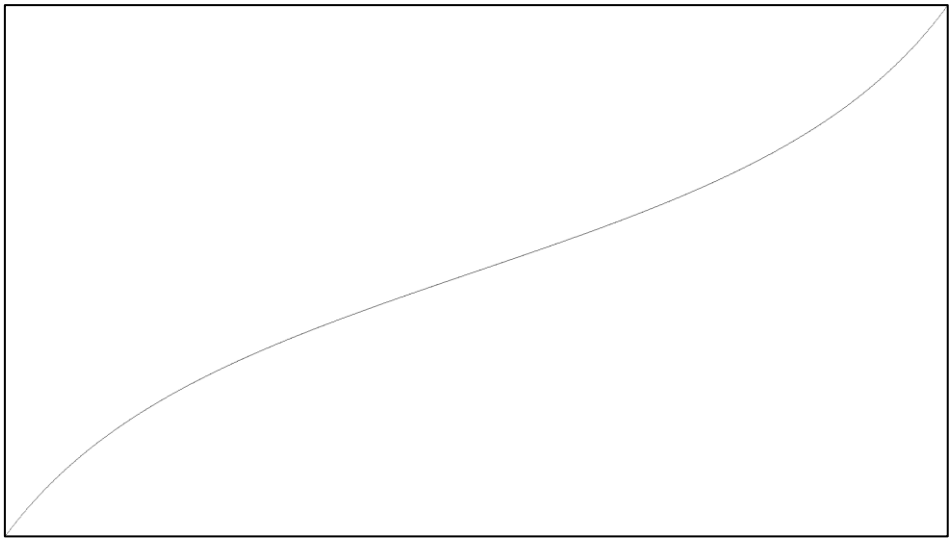
Текст программы

```
#include <iostream>
#include <math.h>
#include <SDL2/SDL.h>
#define XMIN -6.9 /*диапозон графика*/
#define XMAX 5.9
/*отрисовка графика по точкам*/
void draw_by_pixels(SDL_Renderer *render, double xmin, double xmax);
/*функция*/
double f(double x)
{
    return tan(x-M_PI/3);
}
int main(int argc, char *argv[])
{
    SDL_Init(SDL_INIT_EVERYTHING);
    SDL_Window *win = SDL_CreateWindow("y = tan(x-PI/3)", 0, 0, 0, 0,
SDL_WINDOW_FULLSCREEN_DESKTOP);
    if (win == nullptr)
    {
        std::cout << "SDL_CreateWindow Error: " << SDL_GetError() << std::endl;
        return 1;
    }
    SDL_Renderer *ren = SDL_CreateRenderer(win, -1, 0);
    if (ren == nullptr)
    {
        std::cout << "SDL_CreateRenderer Error: " << SDL_GetError() <<
std::endl;
        return 1;
    }
    /*установка цвета*/
    SDL_SetRenderDrawColor(ren, 255, 255, 255, 255);
    /*очистка рендера*/
    SDL_RenderClear(ren);
    /*отрисовка графика*/
    draw_by_pixels(ren, XMIN, XMAX);
    SDL_RenderPresent(ren);
    SDL_Event windowEvent;
    while (true)
```

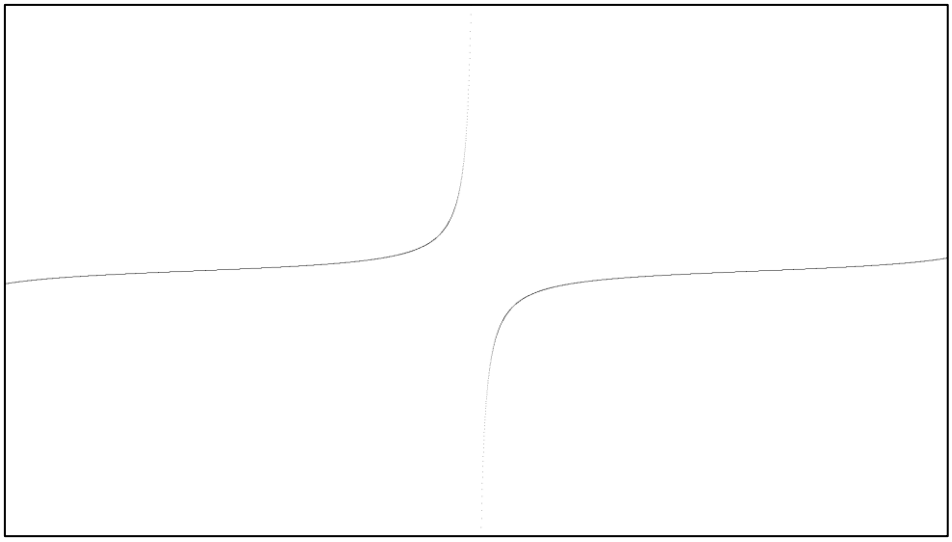
```
    {
        /*обработка событий*/
        if (SDL_PollEvent(&windowEvent))
        {
            /*событие выхода*/
            if (windowEvent.key.keysym.sym==SDLK_ESCAPE)
                break;
        }
    }
    SDL_DestroyWindow(win);
    SDL_DestroyRenderer(ren);
    SDL_Quit();
    return 0;
}
void draw_by_pixels(SDL_Renderer *render, double xmin, double xmax)
{
    Sint32 width, height;
    /*находим размер области для рисования*/
    SDL_GetRendererOutputSize(render, &width, &height);
    SDL_SetRenderDrawColor(render, 0, 0, 0, 255);
    /*крайние точки по Y*/
    double ymin, ymax;
    /*mx, my- масштаб отрисовки по X и Y*/
    double mx=width/fabs(xmin-xmax), my, dx=0.001, x, y;
    Sint16 x0scr, y0scr, xscr, yscr;
    x0scr=floor(-1*xmin*mx);
    y0scr=height/2;
    ymin = ymax = f(xmin);
    for(x=xmin;x<=xmax;x+=dx)
    {
        /*проверяем определен ли тангенс*/
        if(cos(x-M_PI/3))
        {
            /*находим крайние точки по Y*/
            if(ymin>f(x)&& fabs(f(x))<mx/10)
                ymin = f(x);
            if(ymax<f(x)&& fabs(f(x))<mx/10)
                ymax = f(x);
        };
    };
    /*вычисляем масштаб по Y*/
    my=height/fabs(ymin-ymax);
    /*отрисовывем график*/
    for(x=xmin;x<=xmax;x+=dx)
    {
            if(cos(x-M_PI/3))
            {
                y=f(x);
                xscr=x0scr+floor(x*mx);
                yscr=y0scr-floor(y*my);
                SDL_RenderDrawPoint(render, xscr, yscr);
            }
    }
}
```

Результат работы программы

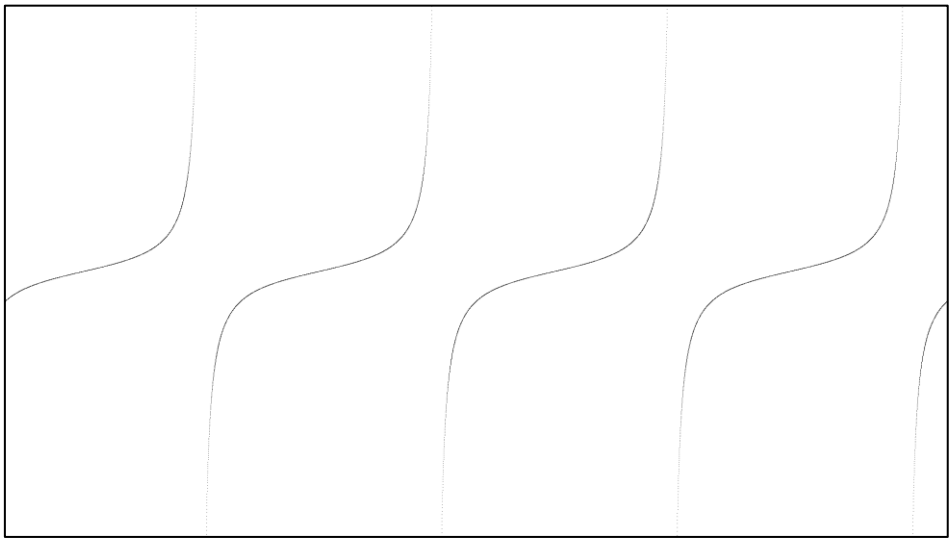
При $x \in \left[0, \frac{2\pi}{3}\right]$



При $x \in \left[-\pi, \frac{2\pi}{3}\right]$



При $x \in \left[-3\pi, \frac{2\pi}{2}\right]$



Задание 2

Текст программы

```
#include <iostream>
#include <math.h>
#include <SDL2/SDL.h>
#define DEFAULTWIDTH 500
#define DEFAULTHEIGHT 500
#define ANGLE1 5*M_PI/3
#define ANGLE2 M_PI/6
#include <vector>

static SDL_Renderer* renderer;

/*класс линия*/
class Line
{
public:
    /*координаты концов*/
    int x1, x2, y1, y2;
    double angle;
    Line()
    {
        x1=x2=y1=y2=angle=0;
    }
    /*установка координат*/
    void setPos(int pos1, int pos2, int pos3, int pos4)
    {
        x1 = pos1;
        x2 = pos2;
        y1 = pos3;
        y2 = pos4;
    }
    /*отрисовка*/
    void draw()
    {
        SDL_RenderDrawLine(renderer, x1, x2, y1, y2);
    }
    /*тип объекта*/
    virtual int getType(){return -1;};
};

/*горизонтальная линия*/
class HortisontalLine : public Line
{
public:
    /*тип объекта*/
    int getType(){return 0;}
};

/*вертикальная линия*/
class VerticalLine : public Line
{
public:
    /*тип объекта*/
    int getType(){return 1;}
};

/*наклонная линия*/
class ObliqueLine : public Line
{
public:
```

```
    /*тип объекта*/
    int getType(){return 2;}
};

/*коллизия линии и круга*/
bool lineCircle(double x1, double y1, double x2, double y2, double cx, double
cy, double r);
/*коллизия точки и круга*/
bool pointCircle(double px, double py, double cx, double cy, double r);
/*коллизия линии и точки*/
bool linePoint(double x1, double y1, double x2, double y2, double px, double
py);
/*расстояние между точками*/
double dist(double x1,double y1, double x2, double y2);

/*класс круг*/
class Circle
{
public:
    /*angle - угол, под которым движется круг*/
    double x, y, radius, angle;
    /*lines - указатели на линии для проверки коллизии*/
    std::vector<Line*> lines;
    Circle(int r, std::vector<Line*> l)
    {
        radius = r;
        angle = 0;
        lines = l;
    }
    /*установка координат круга*/
    void setPos(double posx, double posy)
    {
        x=posx;
        y=posy;
    }
    /*движение круга*/
    void move()
    {
        x+=cos(angle);
        y-=sin(angle);
        if(angle >= 2*M_PI)
            angle-=2*M_PI;
        if(angle <= -2*M_PI)
            angle+=2*M_PI;
        int count = 0, index, size = lines.size();
        /*находим кол-во коллизий*/
        for(int i=0; i<size; i++)
        {
            if(lineCircle(lines[i]->x1, lines[i]->x2, lines[i]->y1, lines[i]-
>y2, x, y, radius))
            {
                index = i;
                count++;
            }
        }
        /*если коллизия с одним объектом, меняем угол движения в зависимости от
типа линии*/
        if(count == 1)
        {
            /*если коллизия с горизонтальной линией*/
            if(lines[index]->getType()==0)
                angle= -angle + 2*M_PI;
            /*если коллизия с вертикальной линией*/
```

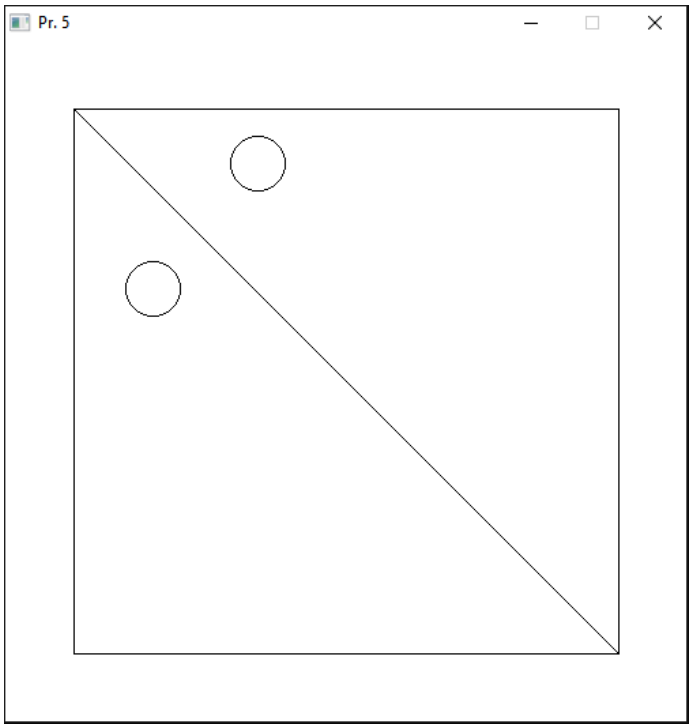
```
            if(lines[index]->getType()==1)
                angle= -angle + 3*M_PI;
            /*если коллизия с наклонной линией*/
            if(lines[index]->getType()==2)
                angle= -angle + 2*M_PI - 2*lines[index]->angle;
        }
        else if(count>1) /*если с двумя, значит угол будет обратным*/
            angle= angle + M_PI;
    };
    /*отрисовка круга по точкам*/
    int draw()
    {
        int offsetx, offsety, d;
        int status;

        offsetx = 0;
        offsety = radius;
        d = radius -1;
        status = 0;

        while (offsety >= offsetx) {
            status += SDL_RenderDrawPoint(renderer, x + offsetx, y + offsety);
            status += SDL_RenderDrawPoint(renderer, x + offsety, y + offsetx);
            status += SDL_RenderDrawPoint(renderer, x - offsetx, y + offsety);
            status += SDL_RenderDrawPoint(renderer, x - offsety, y + offsetx);
            status += SDL_RenderDrawPoint(renderer, x + offsetx, y - offsety);
            status += SDL_RenderDrawPoint(renderer, x + offsety, y - offsetx);
            status += SDL_RenderDrawPoint(renderer, x - offsetx, y - offsety);
            status += SDL_RenderDrawPoint(renderer, x - offsety, y - offsetx);

            if (status < 0) {
                status = -1;
                break;
            }
            if (d >= 2*offsetx) {
                d -= 2*offsetx + 1;
                offsetx +=1;
            }
            else if (d < 2 * (radius - offsety)) {
                d += 2 * offsety - 1;
                offsety -= 1;
            }
            else {
                d += 2 * (offsety - offsetx - 1);
                offsety -= 1;
                offsetx += 1;
            }
        }
        return status;
    }
};

int main(int argc, char *argv[])
{
    SDL_Init(SDL_INIT_EVERYTHING);
    SDL_DisplayMode displayMode;
    if (SDL_GetDesktopDisplayMode(0, &displayMode) != 0)
    {
        SDL_Log("SDL_GetDesktopDisplayMode failed: %s", SDL_GetError());
        return 1;
    }
    int    windowPosX    =    displayMode.w/2-DEFAULTWIDTH/2,windowPosY    =
displayMode.h/2-DEFAULTHEIGHT/2;
```

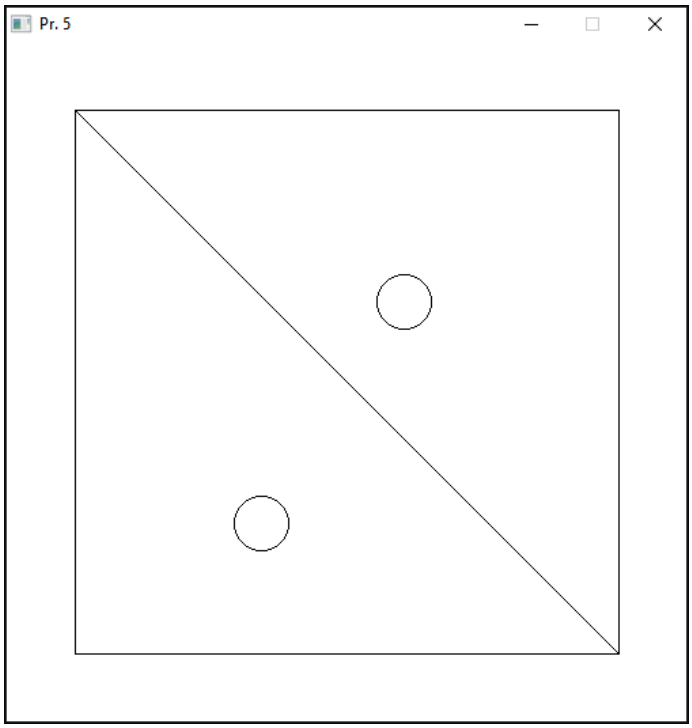
```
    SDL_Window* window = SDL_CreateWindow("Pr. 5",    windowPosX, windowPosY,
DEFAULTWIDTH, DEFAULTWIDTH, 0);
    if (window == nullptr)
    {
        std::cout << "SDL_CreateWindow Error: " << SDL_GetError() << std::endl;
        return 1;
    }
    renderer = SDL_CreateRenderer(window, -1, 0);
    if (renderer == nullptr)
    {
        std::cout  <<  "SDL_CreateRenderer  Error:  "  <<  SDL_GetError()  <<
std::endl;
        return 1;
    }
    SDL_Event windowEvent;

    /*создаем линии для треугольников*/
    std::vector<Line*> lines;
    lines.push_back(new HortisontalLine());
    lines.push_back(new VerticalLine());
    lines.push_back(new ObliqueLine());
    lines.push_back(new HortisontalLine());
    lines.push_back(new VerticalLine());

    int offsetX = DEFAULTWIDTH*0.1, offsetY = DEFAULTHEIGHT*0.1;
    lines[0]->setPos(offsetX,   DEFAULTHEIGHT-offsetY,   DEFAULTWIDTH-offsetX,
DEFAULTHEIGHT-offsetY);
    lines[1]->setPos(offsetX, offsetY, offsetX, DEFAULTHEIGHT-offsetY);
    lines[2]->setPos(offsetX,  offsetY,  DEFAULTWIDTH-offsetX,  DEFAULTHEIGHT-
offsetY);
    /*угол наклонной прямой для отражения*/
    lines[2]->angle = M_PI/4;
    lines[3]->setPos(offsetX, offsetY, DEFAULTWIDTH-offsetX, offsetY);
    lines[4]->setPos(DEFAULTWIDTH-offsetX,   offsetY,   DEFAULTWIDTH-offsetX,
DEFAULTHEIGHT-offsetY);
    /*задаем значения кругам*/
    Circle c(DEFAULTWIDTH*0.04, lines);
    Circle c2(DEFAULTWIDTH*0.04, lines);
    c.angle = ANGLE1;
    c.setPos(DEFAULTWIDTH-2*offsetX, 2*offsetY);
    c2.angle = ANGLE2;
    c2.setPos(2*offsetX, DEFAULTHEIGHT-2*offsetY);
    int size = lines.size();
    while (true)
    {
        SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
        SDL_RenderClear(renderer);
        SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
        for(int i=0; i<size; i++)
            lines[i]->draw();
        c.move();
        c.draw();
        c2.move();
        c2.draw();
        SDL_RenderPresent(renderer);

        if (SDL_PollEvent(&windowEvent))
        {
            if (windowEvent.key.keysym.sym==SDLK_ESCAPE)
                break;
            if (windowEvent.type == SDL_QUIT)
                break;
        }
```

```
        SDL_Delay(1);
    }
    for(int i = 0; i<size; i++)
        delete lines[i];
    lines.clear();
    SDL_DestroyWindow(window);
    SDL_DestroyRenderer(renderer);
    SDL_Quit();
    return 0;
}

/*коллизия линии и круга*/
bool lineCircle(double x1, double y1, double x2, double y2, double cx, double
cy, double r)
{

    /*если концы линии в пересекают круг, то есть пересечение*/
    bool inside1 = pointCircle(x1,y1, cx,cy,r);
    bool inside2 = pointCircle(x2,y2, cx,cy,r);
    if (inside1 || inside2) return true;

    /*длина линии*/
    double len = dist(x1, y1, x2, y2);

    double dot = ( ((cx-x1)*(x2-x1)) + ((cy-y1)*(y2-y1)) ) / pow(len,2);

    /*ближайшие точки*/
    double closestX = x1 + (dot * (x2-x1));
    double closestY = y1 + (dot * (y2-y1));

    /*находится ли ближайшая точка на линии*/
    bool onSegment = linePoint(x1,y1,x2,y2, closestX,closestY);
    if (!onSegment) return false;

    /*если расстояние до ближайшей точки <= радиуса, то есть пересечение*/
    if (dist(closestX, closestY, cx, cy) <= r)
        return true;

    return false;
}

/*коллизия точки и круга*/
bool pointCircle(double px, double py, double cx, double cy, double r)
{
  /*если расстояние между точкой и центром круга <= радиуса, то точка внутри
круга*/
    if (dist(px, py, cx, cy) <= r)
        return true;
    return false;
}

/*коллизия точки и линии*/
bool linePoint(double x1, double y1, double x2, double y2, double px, double
py)
{
    /*расстояние от точки до концов линии*/
    double d1 = dist(px,py, x1,y1);
    double d2 = dist(px,py, x2,y2);
    /*длина линии*/
    double lineLen = dist(x1,y1, x2,y2);
    /*буфферная зона, в которой будет пересечение*/
    double buffer = 0.1;
```

```
    /*если сумма расстояний примерно совпадает с длиной линии, то есть
пересечение*/
    if (d1+d2 >= lineLen - buffer && d1+d2 <= lineLen+buffer)
        return true;
    return false;
}

/*расстояние между точками*/
double dist(double x1,double y1, double x2, double y2)
{
    double distX = x1 - x2;
    double distY = y1 - y2;
    return sqrt( (distX*distX) + (distY*distY) );
}
```

Результаты работы программы





В результате проделанной работы мы приобрели навыки рисования простых фигур; передвижения объектов, с помощью перерисовки объектов; определения коллизии объектов.