МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

Лабораторная работа № 2

**Алгоритмы отсечения**

Выполнил студент 595 гр.

\_\_\_\_\_\_\_\_\_\_\_\_\_ Лаптев А.В.

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Калачев А.В.

Лабораторная работа защищена

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Барнаул 2023

**Задание 1.**

# Дополните программу, разработанную при выполнении Задания 1 в Лабораторной работе №1, функцией:

bool PolyClip( int &x1, int &y1, // координаты начала отрезка

int &x2, int &y2, // координаты конца отрезка

TPoint \*P, // массив вершин многоугольника

int VertexCount // количество вершин многоугольника),

реализующей алгоритм Павлидиса отсечения отрезка выпуклым многоугольным окном.

Функция должна возвращать true, если в результате отсечения отрезка получен фрагмент отрезка, требующий отображения, и false — в противном случае. Тип TPoint определен следующим образом:

typedef struct { int x; int y; } TPoint;

Следует обратить внимание, что аргументы x1, y1, x2, y2 передаются в функцию по ссылке, т.е. служат одновременно как входными, так и выходными параметрами и по завершении работы функции содержат координаты отсеченного отрезка. Ниже приведен типичный пример использования функции PolyClip.

// Описание пятиугольника

TPoint pent[] = { {50,200}, {120, 300}, {300,250}, {350,100}, {80,50} };

// Описание исходного отрезка

int x1=0, y1=0, x2=640, y2=480;

// Результат отсечения (есть/нет видимая часть)

bool toDraw;

// Отобразить отсекающие окно

YX\_FillPolygon(pent,sizeof(pent)/sizeof(TPoint),BLUE);

// Отобразить отсекаемый отрезок серым цветом

Brezenham\_Line(x1,y1,x2,y2,WHITE);

// Произвести отсечение отрезка

toDraw = PolyClip(x1,y1,x2,y2,pent,sizeof(pent)/sizeof(TPoint));

// Если есть видимая часть, отобразить ее белым цветом

if(toDraw) Brezenham\_Line(x1,y1,x2,y2,LIGHTWHITE);

В качестве теста корректности разработанной функции программа должна быть способна:

* считывать описание многоугольного отсекающего окна из файла следующего формата.

<N>

<x1> <y1>

<x2> <y2>

…

<xN> <yN>

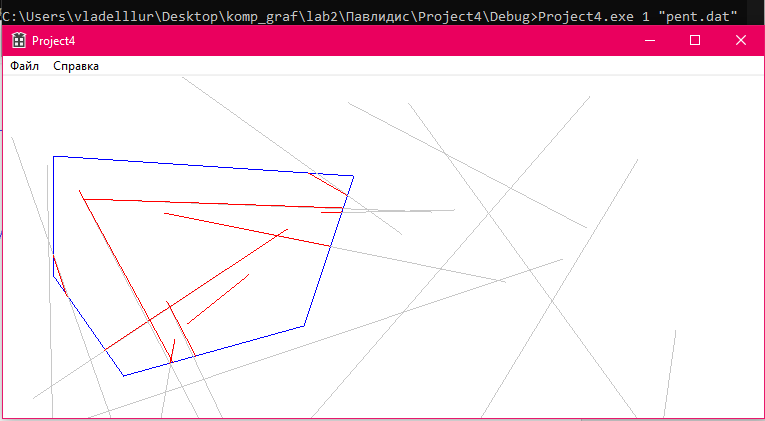
Здесь N — количество вершин в многоугольнике, xi ,yi — координаты вершин многоугольника. В файле может содержаться описание только одного многоугольника. Количество вершин в многоугольнике не может превосходить 50. Примеры описаний простых тестовых многоугольников можно найти в папке Lab2\Polygons.

* работать в двух режимах:
* отображать набор случайных отрезков на фоне отсекающего многоугольного окна;
* отображать отрезок с заданными координатами концов на фоне отсекающего многоугольного окна.

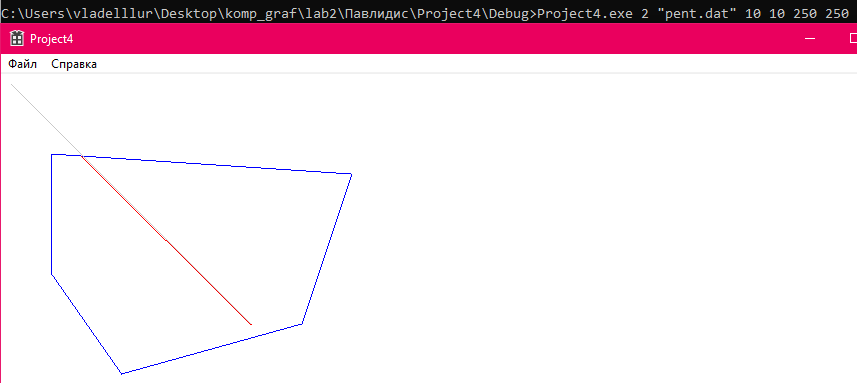
При этом в каждом из режимов отсеченные части отрезков (части, попадающие во внутренность окна) должны выделяться цветом, отличным от цвета исходных отрезков.

Имя файла с описанием отсекающего окна, номер режима работы, а также координаты отсекаемого отрезка (во втором режиме) должны передаваться в программу через командную строку.

**Тест программы:**



*Рис.1. Случайные отрезки.*

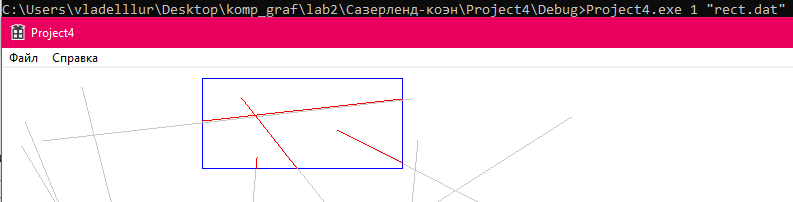


*Рис.2. Заданный отрезок.*

**Задание 2.**

Выполните задание 1 для алгоритма средней точки Сазерленда-Коэна отсечения отрезка прямоугольным окном.

**Тест программы:**



Вывод: Таким образом, мы реализовали алгоритмы отсечения Павлидиса и Сазерленда-Коэна на C++ в виде функций. Программы прошли тест работоспособности для выпуклых окон успешно.

# Приложение А

Код функции, реализующей алгоритм Павлидиса:

long int Sign(long int num)

{

    return (num < 0) ? -1 : ((num>0)? 1 : 0);

}

long int DetFunc(long int xi, int yi, int x1, int y1, int x2, int y2)

{

    return xi \* (y1 - y2) + yi \* (x2 - x1) + (x1 \* y2 - y1 \* x2);

}

long int DetFunc(TPoint Pi, TPoint P1, TPoint P2)

{

    return (Pi.x \* (P1.y - P2.y)) + (Pi.y \* (P2.x - P1.x)) + ((P1.x \* P2.y - P1.y \* P2.x));

}

TPoint Intersection(TPoint P1, TPoint P2, TPoint P3, TPoint P4)

{

    TPoint ret;

    if (P3.x == P4.x)

    {

        ret.x = P3.x;

        ret.y = (-P1.x \* P2.y + P2.x \* P1.y - ret.x\*(P1.y-P2.y)) / (P2.x - P1.x);

        return ret;

    }

    else

    {

        if (P3.y == P4.y)

        {

            ret.y = P3.y;

            ret.x = (-P1.x \* P2.y + P2.x \* P1.y - ret.y \* (P2.x - P1.x)) / (P1.y-P2.y);

            return ret;

        }

    }

    ret.y = -((P1.x \* P2.y - P1.y \* P2.x) \* (P3.y - P4.y) - (P3.x \* P4.y - P3.y \* P4.x) \* (P1.y - P2.y)) / ((P2.x - P1.x) \* (P3.y - P4.y) - (P4.x - P3.x) \* (P1.y - P2.y));

    ret.x = (-ret.y \* (P4.x - P3.x) - (P3.x \* P4.y - P3.y \* P4.x)) / (P3.y - P4.y);

    return ret;

}

bool PolyClip(HWND hWnd, int& x1, int& y1, int& x2, int& y2, TPoint\* P, int VertexCount)

{

    TPoint\* line = new TPoint[2];

    line[0].x = x1; line[0].y = y1;

    line[1].x = x2; line[1].y = y2;

    long int& i = \* (new long int);

    long int\* Si = new long int[VertexCount];

    bool ex1 = true;

    std::vector<int>\* zero = new std::vector<int>;

    std::vector<int>\* notzero = new std::vector<int>;

    for (i = 0; i < VertexCount; i++)

    {

        Si[i] = DetFunc(P[i], line[0], line[1]);

        if (Si[i] == 0)

        {

            zero->push\_back(i);

            continue;

        }

        notzero->push\_back(i);

    }

    for (i = 0; i < notzero->size(); i++)

    {

        if (Sign(Si[(\*notzero)[0]]) != Sign(Si[(\*notzero)[i]]))

        {

            ex1 = false;

            break;

        }

    }

    if (ex1 && (zero->size()==0))

    {

        return false;

    }

    if ((zero->size() == 1)&&ex1)

    {

        x1 = P[(\*zero)[0]].x;

        x2 = P[(\*zero)[0]].x;

        y1 = P[(\*zero)[0]].y;

        y2 = P[(\*zero)[0]].y;

        return true;

    }

    if ((zero->size() == 2) && ex1)

    {

        x1 = P[zero->at(0)].x;

        x2 = P[zero->at(1)].x;

        y1 = P[zero->at(0)].y;

        y2 = P[zero->at(1)].y;

        return true;

    }

    notzero->clear();

    notzero->shrink\_to\_fit();

    delete notzero;

    zero->clear();

    zero->shrink\_to\_fit();

    delete zero;

    bool first = true, second = true;

    SLine\* SL = new SLine[VertexCount];

    int firstmainsign = DetFunc(line[0], P[0], P[1]);

    int secondmainsign = DetFunc(line[1], P[0], P[1]);

    SL[0].firstpoint = firstmainsign;

    SL[0].secondpoint = secondmainsign;

    for (i = 1; i < VertexCount - 1; i++)

    {

        SL[i].firstpoint = DetFunc(line[0], P[i], P[i+1]);

        SL[i].secondpoint = DetFunc(line[1], P[i], P[i + 1]);

        if (Sign(firstmainsign) != Sign(SL[i].firstpoint))

        {

            first = false;

        }

        if (Sign(secondmainsign) != Sign(SL[i].secondpoint))

        {

            second = false;

        }

    }

    SL[i].firstpoint = DetFunc(line[0], P[i], P[0]);

    SL[i].secondpoint = DetFunc(line[1], P[i], P[0]);

    if (Sign(firstmainsign) != Sign(SL[i].firstpoint))

    {

        first = false;

    }

    if (Sign(secondmainsign) != Sign(SL[i].secondpoint))

    {

        second = false;

    }

    bool flag = true;

    std::vector<PIntersection> pi;

    PIntersection\* buf = new PIntersection;

    for (i = 0; i < VertexCount - 1; i++)

    {

        if (Sign(Si[i]) != Sign(Si[i + 1]))

        {

            if (Sign(SL[i].firstpoint) != Sign(SL[i].secondpoint))

            {

                buf->start = i;

                buf->finish = i + 1;

                pi.push\_back(\*buf);

            }

        }

        if (Sign(SL[i].firstpoint) != Sign(SL[i+1].firstpoint))

            flag = false;

        if (Sign(SL[i].secondpoint) != Sign(SL[i + 1].secondpoint))

            flag = false;

    }

    if (Sign(Si[i]) != Sign(Si[0]))     {

        if (Sign(SL[i].firstpoint) != Sign(SL[i].secondpoint))

        {

            buf->start = i;

            buf->finish = 0;

            pi.push\_back(\*buf);

        }

    }

    if (Sign(SL[i].firstpoint) != Sign(SL[0].firstpoint))

        flag = false;

    if (Sign(SL[i].secondpoint) != Sign(SL[0].secondpoint))

        flag = false;

    if (pi.size() == 0)

    {

        if (flag)

        {

            return true;

        }

        return false;

    }

    delete buf;

    delete[] SL;

    delete[] Si;

    TPoint& intrsec = \*(new TPoint);

    for (i = 0; i < pi.size(); i++)

    {

        intrsec = Intersection(line[0], line[1], P[pi[i].start], P[pi[i].finish]);

        if (!first)

        {

            x1 = intrsec.x;

            y1 = intrsec.y;

            first = true;

            continue;

        }

        if (!second)

        {

            x2 = intrsec.x;

            y2 = intrsec.y;

            second = true;

            continue;

        }

    }

    pi.clear();

    pi.shrink\_to\_fit();

    delete& intrsec;

    delete& i;

    delete[] line;

    return true;

}

Код WM\_PAINT

int otrezok[2][2];

            char title[512];

            WideCharToMultiByte(CP\_ACP, 0, argv[2], -1, title, sizeof(title), 0, 0);

            long int filebuf;

            FILE\* file = fopen(title, "rb");

            fscanf(file, "%d\n", &filebuf);

            const unsigned int sizeofmatrix = filebuf;

            TPoint\* pent = new TPoint[sizeofmatrix];

            for (int i = 0; i < sizeofmatrix; i++)

            {

                fscanf(file, "%d ", &filebuf);

                pent[i].x = filebuf;

                fscanf(file, "%d\n", &filebuf);

                pent[i].y = filebuf;

            }

            fclose(file);

            int i;

            for (i=0; i < sizeofmatrix-1; i++)

            {

                Brezenhem(hdc, pent[i].x, pent[i].y, pent[i + 1].x, pent[i + 1].y, 0, 0, 255);

            }

            Brezenhem(hdc, pent[i].x, pent[i].y, pent[0].x, pent[0].y, 0, 0, 255);

            if (!(lstrcmpW(argv[1], L"1")))

            {

                srand(int(time(NULL)));

                for (i = 0; i < 30; i++)

                {

                    {

                        otrezok[0][0] = rand() % 400;

                        otrezok[1][0] = rand() % (400);

                        otrezok[0][1] = rand() % (400);

                        otrezok[1][1] = rand() % (400);

                    }

                    Brezenhem(hdc, otrezok[0][0], otrezok[1][0], otrezok[0][1], otrezok[1][1], 200, 200, 200);

                    if (PolyClip(hWnd, otrezok[0][0], otrezok[1][0], otrezok[0][1], otrezok[1][1], pent, sizeofmatrix))

                    {

                        Brezenhem(hdc, otrezok[0][0], otrezok[1][0], otrezok[0][1], otrezok[1][1], 255, 0, 0);

                    }

                }

            }

            else

            {

                char buf[11];

                if (!(lstrcmpW(argv[1], L"2")))

                {

                    {

                        if (!WideCharToMultiByte(CP\_ACP, 0, argv[3], -1, buf, sizeof(buf), 0, 0))

                        {

                            MessageBox(hWnd, L"ERROR", argv[0], MB\_OK);

                            exit(-1);

                        }

                        otrezok[0][0] = atoi(buf);

                        if (!WideCharToMultiByte(CP\_ACP, 0, argv[4], -1, buf, sizeof(buf), 0, 0))

                        {

                            MessageBox(hWnd, L"ERROR", argv[0], MB\_OK);

                            exit(-1);

                        }

                        otrezok[1][0] = atoi(buf);

                        if (!WideCharToMultiByte(CP\_ACP, 0, argv[5], -1, buf, sizeof(buf), 0, 0))

                        {

                            MessageBox(hWnd, L"ERROR", argv[0], MB\_OK);

                            exit(-1);

                        }

                        otrezok[0][1] = atoi(buf);

                        if (!WideCharToMultiByte(CP\_ACP, 0, argv[6], -1, buf, sizeof(buf), 0, 0))

                        {

                            MessageBox(hWnd, L"ERROR", argv[0], MB\_OK);

                            exit(-1);

                        }

                        otrezok[1][1] = atoi(buf);

                    }

                    Brezenhem(hdc, otrezok[0][0], otrezok[1][0], otrezok[0][1], otrezok[1][1], 200, 200, 200);

                    if (PolyClip(hWnd, otrezok[0][0], otrezok[1][0], otrezok[0][1], otrezok[1][1], pent, sizeofmatrix))

                    {

                        Brezenhem(hdc, otrezok[0][0], otrezok[1][0], otrezok[0][1], otrezok[1][1], 255, 0, 0);

                    }

                }

            }

            delete[] pent;

# Приложение Б

Код функции алгоритма Сазерленда-Коэна:

XYCode GetXYCode(long int x\_min, long int x\_max, long int y\_min, long int y\_max, int x1, int y1, int x2, int y2)

{

    XYCode code;

    ZeroMemory(&code, sizeof(code));

    code.buf = 8;

    if (x1 < x\_min)

    {

        code.p1 = code.p1 | code.buf;

    }

    if (x2 < x\_min)

    {

        code.p2 = code.p2 | code.buf;

    }

    code.buf = code.buf >> 1; //0100

    if (x1 > x\_max)

    {

        code.p1 = code.p1 | code.buf;

    }

    if (x2 > x\_max)

    {

        code.p2 = code.p2 | code.buf;

    }

    code.buf = code.buf >> 1;// 0010

    if (y1 < y\_min)

    {

        code.p1 = code.p1 | code.buf;

    }

    if (y2 < y\_min)

    {

        code.p2 = code.p2 | code.buf;

    }

    code.buf = code.buf >> 1; // 0001

    if (y1 > y\_max)

    {

        code.p1 = code.p1 | code.buf;

    }

    if (y2 > y\_max)

    {

        code.p2 = code.p2 | code.buf;

    }

    return code;

}

TPoint CutLine(int x1, int y1, int \_x2, int \_y2)

{

    TPoint ret;

    int del = (\_y2 - y1)/2;

    ret.y = \_y2 - del;

    del = (\_x2 - x1) / 2;

    ret.x = \_x2 - del;

    return ret;

}

bool QuadClip(int& x1, int& y1, int& x2, int& y2, TPoint\* P, int VertexCount)

{

    XYCode& code = \*(new XYCode), &codebackup = \*(new XYCode);

    ZeroMemory(&code, sizeof(code));

    memset(&code, 0, sizeof(code));

    unsigned int& i = \*(new unsigned int);

    long int x\_max, x\_min; x\_max = max(P[0].x, max(P[1].x, P[2].x)); x\_min = min(P[0].x, min(P[1].x, P[2].x));

    long int y\_max, y\_min; y\_max = max(P[0].y, max(P[1].y, P[2].y)); y\_min = min(P[0].y, min(P[1].y, P[2].y));

    code = GetXYCode(x\_min, x\_max, y\_min, y\_max, x1, y1, x2, y2);

    codebackup = GetXYCode(x\_min, x\_max, y\_min, y\_max, x1, y1, x2, y2);

    if ((code.p1 == code.p2)&&(code.p1 == 0000))

    {

        return true;

    }

    if ((code.p1&code.p2)!=0000)

    {

        return false;

    }

    TPoint point, pre\_point, dis\_one;

    point.x = x2;

    point.y = y2;

    if (code.p1 != 0000)

    {

        while (true)

        {

            pre\_point.x = point.x;

            pre\_point.y = point.y;

            point = CutLine(x1, y1, point.x, point.y);

            code = GetXYCode(x\_min, x\_max, y\_min, y\_max, x1, y1, point.x, point.y);

            if (code.p2 != 0000)

            {

                x1 = point.x;

                y1 = point.y;

                point.x = pre\_point.x;

                point.y = pre\_point.y;

            }

            if ((abs(x1 - point.x) <= 1) && (abs(y1 - point.y) <= 1))

            {

                break;

            }

        }

    }

    point.x = x1;

    point.y = y1;

    if (codebackup.p2 != 0000)

    {

        while (true)

        {

            pre\_point.x = point.x;

            pre\_point.y = point.y;

            point = CutLine(x2, y2, point.x, point.y);

            code = GetXYCode(x\_min, x\_max, y\_min, y\_max, x2, y2, point.x, point.y);

            if (code.p2 != 0000)

            {

                x2 = point.x;

                y2 = point.y;

                point.x = pre\_point.x;

                point.y = pre\_point.y;

            }

            if ((abs(x2 - point.x) <= 1) && (abs(y2 - point.y) <= 1))

            {

                break;

            }

        }

    }

    return true;

    delete& i;

    delete& code;

}