WAMS 2024 Programowanie - Projekt

Sprawozdanie

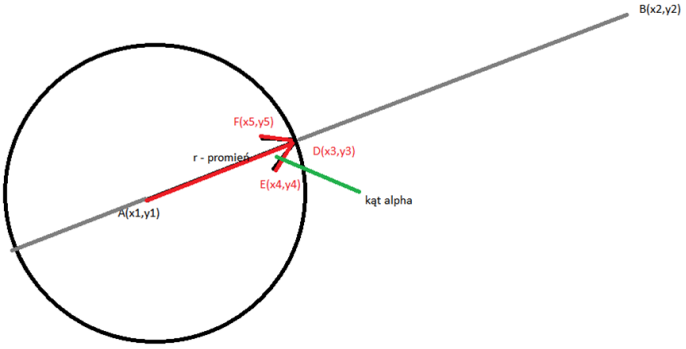
Gerard Kalinowski

Nr indeksu: 79605
Zadanie : WAMS – programowanie – projektowe 1
Numer zadania: 1

Spis treści

1. Treść Polecenia

Korzystając z równania opisującego okręg o środku A i promieniu r, oraz równania prostych wyznacz punkty D, E, F opisujące strzałkę ukierunkowaną w stronę punktu B, nachyloną pod kątem α.



Dane Wejściowe (Input)

$A\ (x_A, y_A)$ - Współrzędne punktu A, będącego również środkiem okręgu

$B\ (x_B, y_B)$ - Współrzędne punktu końcowego

r - Promień koła o środku w punkcie A

alpha - Kąt nachylenia ramion strzałki względem promienia (10°-45°)

Dane Wyjściowe (Output)

$D\ (x_D, y_D)$ - Punkt opisujący czoło sztrzałki

$E\ (x_E, y_E)$ - Punkt opisujący jedno ramię strzałki

$F\ (x_F, y_F)$ - Punkt opisujący drugie ramię strzałki

Założenia zadania

- Środek okręgu A znajduje się w układzie współrzędnych w punkcie (x_A, y_A)
- Promień r musi być większy od zera: $r > 0$
- Punkt D leży na odcinku AB i jest punktem przecięcia tej linii z okręgiem
- Kąty $\angle ADE, \angle ADF$ mają wartość α
- Długość ramienia strzałki jest równa 1/5 promienia R
  $|DE| = |DF| = \frac{r}{5}$
  - Długość wektora $\overrightarrow{DE}$ jest pięciokrotnie mniejsza od wektora $\overrightarrow{AD}$
    $\|\overrightarrow{AD}\| = 5 \times \|\overrightarrow{DE}\|$
- Punkty E, F znajdują się wewnątrz okręgu
  - Punkt F jest symetrycznym odbiciem punktu E względem odcinka AD.
    $F = S_{AD}(E)$
  - Wektor $\overrightarrow{DE}$ należy obrócić o α w lewo i użyć jego negacji do wyznaczenia punktu E

Założenia programu

1. Użytkownik wprowadza Dane Wejściowe.
2. Program przeprowadza obliczenia za pomocą konkretnych wzorów, wszystkie poczynania wyprowadza do konsoli.
3. Finalne współrzędne punktów D,E,F (Dane Wyjściowe) wyprowadza do konsoli.

2. Wyprowadzenie wzorów

0. Ustanowienie współrzędnych punktów A, B, promienia r oraz kąta α:

   $A(x_A, y_A),\ B(x_B, y_B),\ r,\ \alpha \in (10^\circ, 45^\circ)$

1. Wektor kierunkowy prostej AB:

   a. Wyznaczenie wektora kierunkowego prostej AB, łączącej te dwa punkty:

   $\overrightarrow{AB} = (x_B - x_A, y_B - y_A)$

   b. Długość wektora:

   $\|\overrightarrow{AB}\| = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$

   c. Normalizacja wektora:

   $\hat{u} = (\frac{x_{\overrightarrow{AB}}}{\|\overrightarrow{AB}\|}, \frac{y_{\overrightarrow{AB}}}{\|\overrightarrow{AB}\|})$

   $\hat{u} = (\frac{x_B - x_A}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}}, \frac{y_B - y_A}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}})$

2. Wyznaczenie współrzędnych dla punktu $D(x_D, y_D)$:

   a. Znalezienie punktu D na odcinku AB w odległości d od punktu A wzdłuż wektora $\overrightarrow{AB}$

   $D = A + d \times \hat{u}$

   b. Współrzędne punktu D, gdzie $d = r$, bo punkt leży na okręgu:

   $$D = \begin{pmatrix} x_D = x_A \\ y_D = y_A \end{pmatrix} + r \times \begin{pmatrix} x_{\hat{u}} \\ y_{\hat{u}} \end{pmatrix}$$

   $$D = \begin{pmatrix} x_D = x_A + r \times \frac{x_B - x_A}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}} \\ y_D = y_A + r \times \frac{y_B - y_A}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}} \end{pmatrix}$$

3. Wyznaczenie współrządnych dla punktu $E(x_E, y_E)$:

   a. Obliczenie wektora $\overrightarrow{AD}$

   $$\overrightarrow{AD} = \begin{pmatrix} x_{\overrightarrow{AD}} = x_D - x_A \\ y_{\overrightarrow{AD}} = y_D - y_A \end{pmatrix}$$

   b. Obliczenie długości wektora $\|\overrightarrow{AD}\| = r$

   $\|\overrightarrow{AD}\| = \sqrt{(x_D - x_A)^2 + (y_D - y_A)^2}$

   c. Obrócenie wektora $\overrightarrow{DE}$ o α w lewo, gdzie również $\|\overrightarrow{AD}\| = 5 \times \|\overrightarrow{DE}\|$:

   $$\overrightarrow{DE} = \begin{pmatrix} x_{\overrightarrow{DE}} = \frac{x_{\overrightarrow{AD}} \cos(\alpha) - y_{\overrightarrow{AD}} \sin(\alpha)}{5} \\ y_{\overrightarrow{DE}} = \frac{x_{\overrightarrow{AD}} \sin(\alpha) + y_{\overrightarrow{AD}} \cos(\alpha)}{5} \end{pmatrix}$$

   $$\overrightarrow{DE} = \begin{pmatrix} x_{\overrightarrow{DE}} = \frac{(x_D - x_A)\cos(\alpha) - (y_D - y_A)\sin(\alpha)}{5} \\ y_{\overrightarrow{DE}} = \frac{(x_D - x_A)\sin(\alpha) + (y_D - y_A)\cos(\alpha)}{5} \end{pmatrix}$$

   $\|\overrightarrow{AD}\| = 5 \times \|\overrightarrow{DE}\|$

   $\|\overrightarrow{DE}\| = \sqrt{(x_{\overrightarrow{DE}})^2 + (y_{\overrightarrow{DE}})^2}$

   d. Negacja wektora $\overrightarrow{DE}$:

   $$-\overrightarrow{DE} = \overrightarrow{DE}' = \begin{pmatrix} x_{\overrightarrow{DE}'} = \frac{-(x_D - x_A)\cos(\alpha) + (y_D - y_A)\sin(\alpha)}{5} \\ y_{\overrightarrow{DE}'} = \frac{-(x_D - x_A)\sin(\alpha) - (y_D - y_A)\cos(\alpha)}{5} \end{pmatrix}$$

   e. Wyznaczenie punktu E:

   $$E = \begin{pmatrix} x_E = x_D + x_{\overrightarrow{DE}'} \\ y_E = y_D + y_{\overrightarrow{DE}'} \end{pmatrix}$$

   $$E = \begin{pmatrix} x_E = x_D + \frac{-(x_D - x_A)\cos(\alpha) + (y_D - y_A)\sin(\alpha)}{5} \\ y_E = y_D + \frac{-(x_D - x_A)\sin(\alpha) - (y_D - y_A)\cos(\alpha)}{5} \end{pmatrix}$$

4. Wyznaczenie punktu F, jako symetrycznego odbicia punktu E, względem prostej AB

   a. Wzór funkcji liniowej f prostej AB

$$f: \begin{cases} y_A = mx_A + b \\ y_B = mx_B + b \end{cases}$$

$$b = y_A - \frac{y_B - y_A}{x_B - x_A} x_A$$

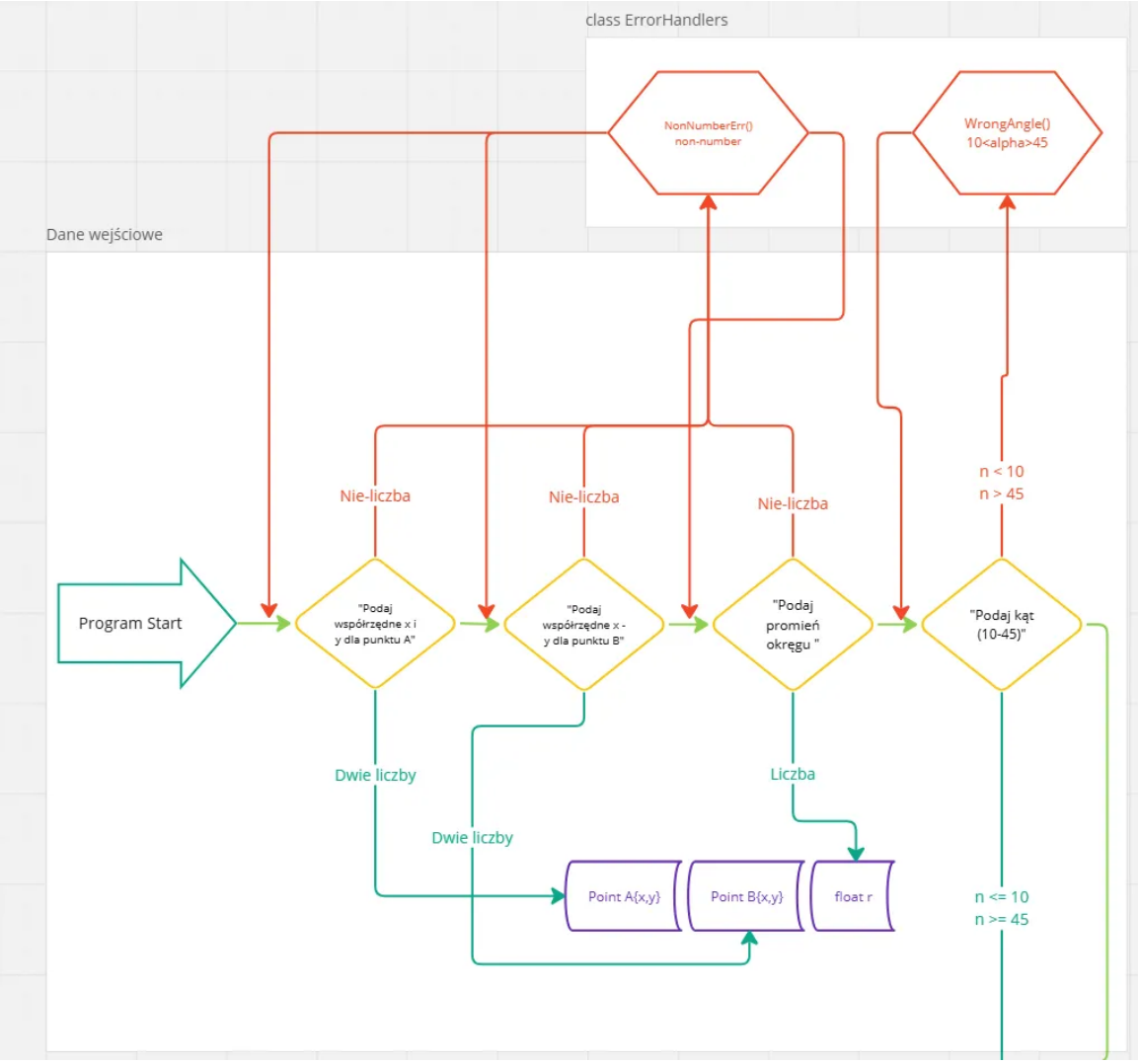
b. Współczynnik kierunkowy funkcji:

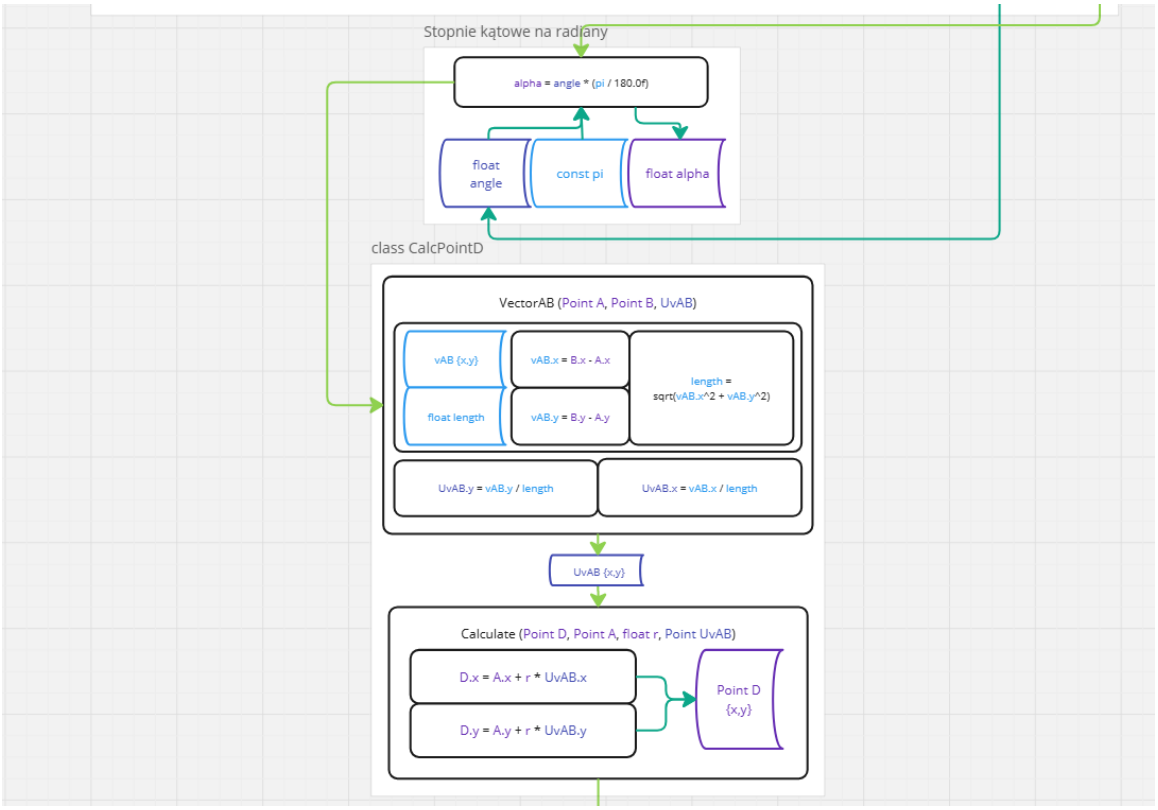
$$m = \frac{y_B - y_A}{x_B - x_A}$$

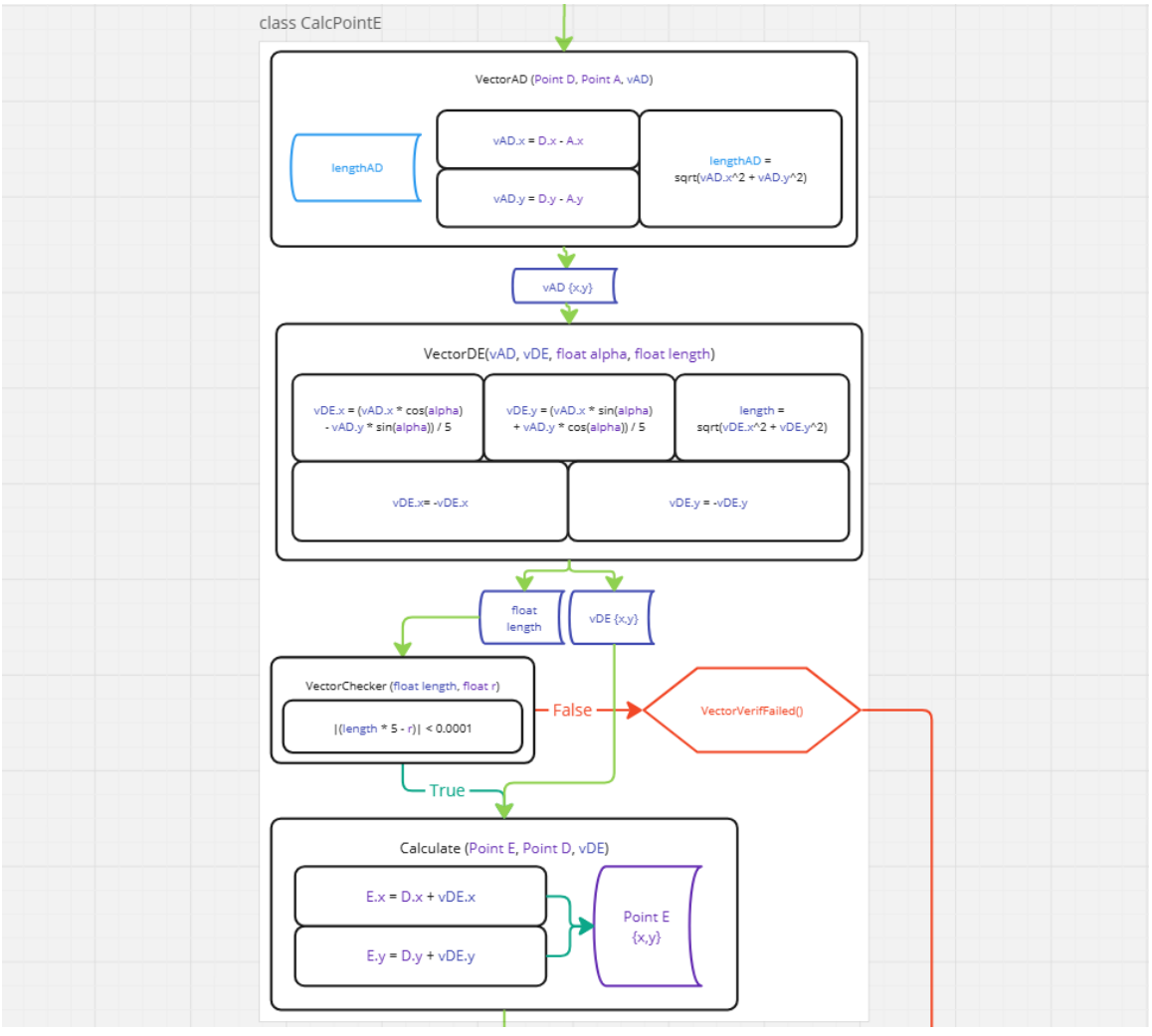
c. Wyznaczenie punktu F ze wzoru na współrzędne punktu będącego symetrycznym odbiciem względej prostej:

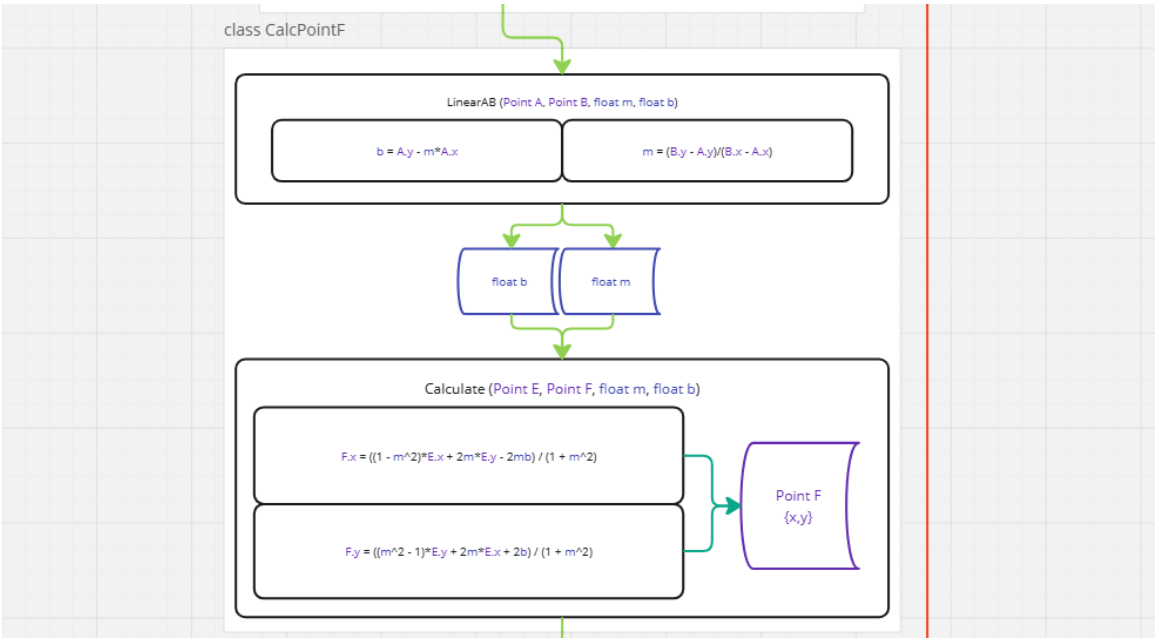
$$F = \begin{pmatrix} x_F = \frac{(1-m^2)x_E + 2my_E - 2mb}{1+m^2} \\ y_F = \frac{(m^2-1)y_E + 2mx_E + 2b}{1+m^2} \end{pmatrix}$$

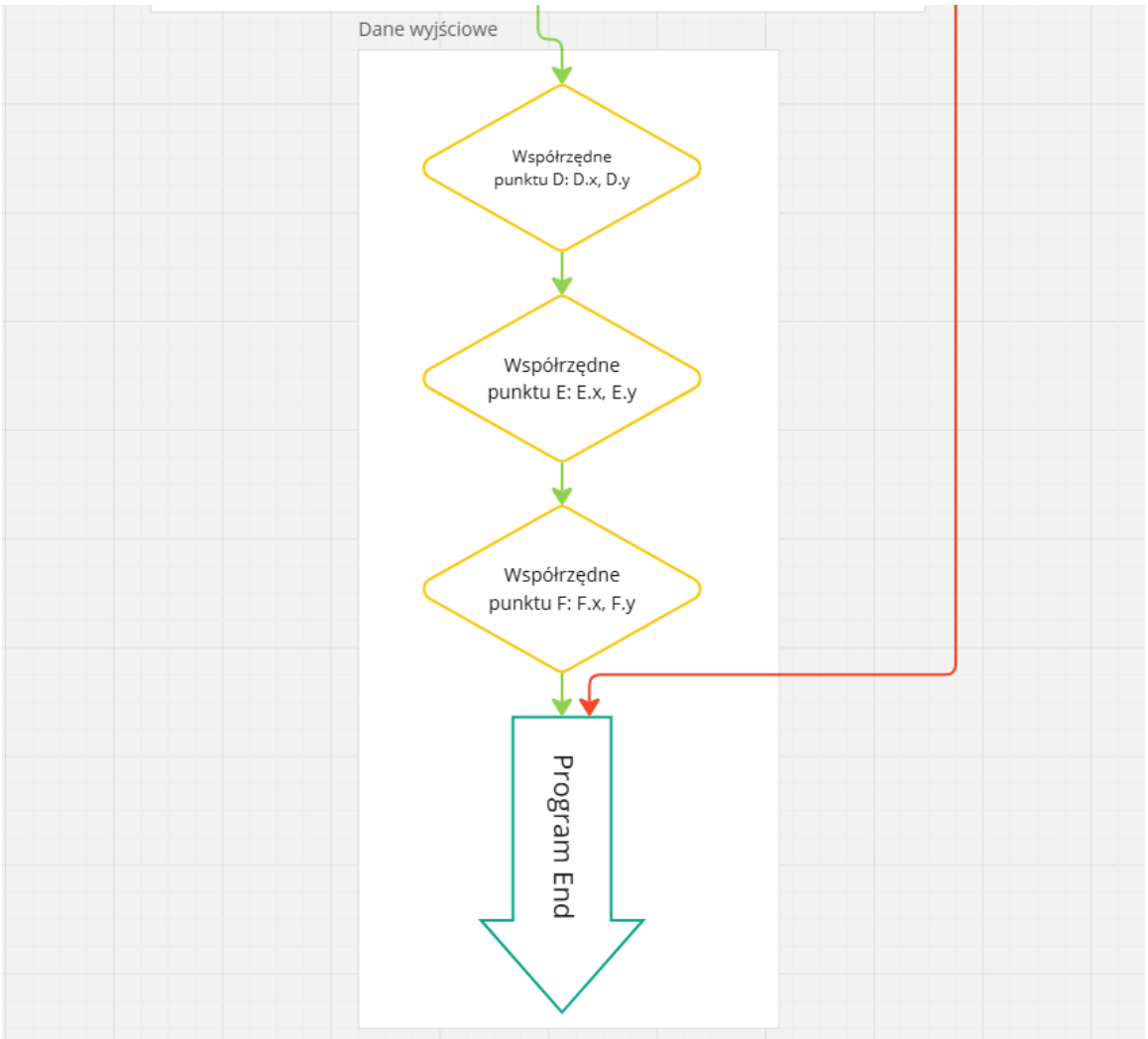
3. Schemat blokowy




Stopnie kątowe na radiany
alpha = angle * (pi / 180.0f)
float angle
const pi
float alpha
class CalcPointD
VectorAB (Point A, Point B, UvAB)
vAB {x,y}
float length
vAB.x = B.x - A.x
vAB.y = B.y - A.y
length = sqrt(vAB.x^2 + vAB.y^2)
UvAB.y = vAB.y / length
UvAB.x = vAB.x / length
UvAB {x,y}
Calculate (Point D, Point A, float r, Point UvAB)
D.x = A.x + r * UvAB.x
D.y = A.y + r * UvAB.y
Point D {x,y}


class CalcPointE
VectorAD (Point D, Point A, vAD)
lengthAD
vAD.x = D.x - A.x
vAD.y = D.y - A.y
lengthAD = sqrt(vAD.x^2 + vAD.y^2)
vAD {x,y}
VectorDE(vAD, vDE, float alpha, float length)
vDE.x = (vAD.x * cos(alpha) - vAD.y * sin(alpha)) / 5
vDE.y = (vAD.x * sin(alpha) + vAD.y * cos(alpha)) / 5
length = sqrt(vDE.x^2 + vDE.y^2)
vDE.x= -vDE.x
vDE.y = -vDE.y
float length
vDE {x,y}
VectorChecker (float length, float r)
|(length * 5 - r)| < 0.0001
False
VectorVeriffFailed()
True
Calculate (Point E, Point D, vDE)
E.x = D.x + vDE.x
E.y = D.y + vDE.y
Point E {x,y}


class CalcPointF
LinearAB (Point A, Point B, float m, float b)
b = A.y - m*A.x
m = (B.y - A.y)/(B.x - A.x)
float b
float m
Calculate (Point E, Point F, float m, float b)
F.x = ((1 - m^2)*E.x + 2m*E.y - 2mb) / (1 + m^2)
F.y = ((m^2 - 1)*E.y + 2m*E.x + 2b) / (1 + m^2)
Point F
{x,y}


Dane wyjściowe
Współrzędne punktu D: D.x, D.y
Współrzędne punktu E: E.x, E.y
Współrzędne punktu F: F.x, F.y
Program End

4. Kod programu

```
#include <iostream>
// potrzebne dla VectorChecker()
#include <cmath>
// potrzebne do GetValidatedInput()
#include <sstream>
#include <string>
#include <limits>

using namespace std;

/*//=============================================================================
    Definiowanie zmiennych punktów jako zbioru współrzędnych
    Przyda się również w sytuacji wektorów
*///=============================================================================

struct Point {
    float x;
    float y;
};

/*//=============================================================================
    Metody błędów:
        NonNumerErr() - Zła liczba podana jako input
        WrongAngle() - Wartość kąta alpha wykracza poza założenia programu
        VectorVerifFailed() - WektorDE nie spełnia założeń zadania (VectorDE * 5 = r)
*///=============================================================================

class ErrorHandlers {
    public:

    float GetValidatedInput(const std::string& prompt) {
        std::string input;
        float value;

        while (true) {
            cout << prompt;
            getline(cin, input);

            // Konwersja tekstu na liczbę
            stringstream stream(input);
            if (stream >> value && stream.eof()) {
                return value;
            } else {
                cout << "!!! To nie jest liczba, albo użyłeś przecinka jako separatora dziesiętnego -
Użyj kropki !!!\n\n";
            }
        }
    }

    void SamePoints() {
        cout << "!!! Punkty A i B nie mogą leżeć w tym samym miejscu !!!\n\n";
    }

    void DivZero(){
        cout << "!!! Punkty A i B leżą na osi OX !!!\n";
        cout << "!!! Nie będzie możliwe wyznaczenie punktu F (Dzielenie przez zero) !!!\n\n";
    }

    void WrongAngle() {
        cout << "!!! Kąt ma być w zakresie 10-45 stopni !!!\n\n";
```

```
    }

    void WrongRadius() {
        cout << "!!! Podaj liczbę większą niż 0 !!!\n\n";
    }

    void VectorVerifFailed() {
        cout << "!!! Długość wektora DE nie spełnia warunku zadania - błąd w obliczeniach !!!\n\n";
        exit(100);
    }
};

/*//=============================================================================
    Wyznaczenie punktu D - operacje:
        VectorAB() - metoda wyznaczająca wektor AB:
            - Współrzędne x, y wektora AB
            - Długość wektora AB
            - Normalizacja wektora AB

        Calculate() - metoda wyznaczająca współrzędne punktu D:
            - Współrzędne x, y punktu D
*///=============================================================================

class CalcPointD {
    public:

    Point UvAB;

    void VectorAB(Point* A, Point* B,  Point* UvAB){
        // Współrzędne Wektora AB
        Point vAB;
        vAB.x = B->x - A->x;
        vAB.y = B->y - A->y;
        cout << "-----------------------------------------------------------------------\n";
        cout << "== Wektor AB: (" << vAB.x << "),(" << vAB.y << ")\n";

        // Długość Wektora AB
        float length;
        length = sqrt(pow(vAB.x,2) + pow(vAB.y,2));
        cout << "== Długość Wektora AB = " << length << endl;

        // Normalizacja (Unifikacja) Wektora AB
        UvAB->x = vAB.x / length;
        UvAB->y = vAB.y / length;
        cout << "== Znormalizowany Wektor AB: (" << UvAB->x << "),(" << UvAB->y << ")\n";
    };

    void Calculate(Point* D, Point* A, float r, Point* UvAB){
        // Współrzędne punktu D
        D->x = A->x + (r * UvAB->x);
        D->y = A->y + (r * UvAB->y);
        //cout << "== PointD: (" << D->x << "),(" << D->y << ")\n";
        cout << "-----------------------------------------------------------------------\n";
    };
};

/*//=============================================================================
    Wyznaczenie punktu E - operacje:
        VectorAD() - metoda wyznaczająca wektor AD:
            - Współrzędne x, y wektora AD
            - Długość wektora AD

        VectorDE() - metoda wyznaczająca wektor DE:
            - Współrzędne x, y wektora DE (Obrócenie wektora AD o kąt alpha w lewo)
```

```
            - Długość wektora DE (Do późniejszej weryfikacji z założeniami zadania)
            - Negacja współrzędnych wektora DE (Aby punkt E znajdował się wewnątrz okręgu o środku A)

        VectorChecker() - metoda sprawdzająca założenie WektorDE * 5 = AD = r
            - Jeśli prawda -> idź dalej (Floating Point error w granicy 0.0001 jest dopuszczalny)
            - Jeśli fałsz -> terminacja programu (kod 100)

        Calculate() - wyznaczenie współrzędnych punktu E:
            - Współrzędne x, y punktu E
*///============================================================================

class CalcPointE {
    public:

    Point vAD, vDE;
    float length;

    void VectorAD(Point* D, Point* A, Point* vAD){
        // Współrzędne Wektora AD
        vAD->x = D->x - A->x;
        vAD->y = D->y - A->y;
        cout << "== Wektor AD: (" << vAD->x << "), (" << vAD->y << ")" << endl;

        // Długość Wektora AD
        float lengthAD;
        lengthAD = sqrt(pow(vAD->x,2) + pow(vAD->y,2));
        cout << "== Długość Wektora AD = " << lengthAD << endl;
    };

    void VectorDE(Point* vAD, Point* vDE, float alpha, float& length){
        // Obrócenie Wektora AD o alpha kątów w lewo, pomniejszenie wartości pięciokrotnie
        vDE->x = (vAD->x * cos(alpha) - vAD->y * sin(alpha)) / (5);
        vDE->y = (vAD->x * sin(alpha) + vAD->y * cos(alpha)) / (5);

        // Wyliczenie długości wektora w celu weryfikacji
        length = sqrt(pow(vDE->x,2) + pow(vDE->y,2));

        // Negacja Wektora DE, aby punkt E znajdował się wewnątrz okręgu
        vDE->x = -vDE->x;
        vDE->y = -vDE->y;
        cout << "== Wektor DE: (" << vDE->x << "), (" << vDE->y << ")" << endl;
    };

    void VectorChecker(float& length, float r){
        // Floating point error handler
        float epsilon = 0.0001;

        // Jeżeli długość Wektora DE * 5 jest względnie taka sama z promieniem r to przejdź dalej
        if (fabs((length * 5) - r) < epsilon) {
            cout << "== Długość wektora DE spełnia warunek zadania!\n";
        }
        // W przeciwnym razie, przerwij działanie programu
        else {
            ErrorHandlers ErrHand;
            ErrHand.VectorVerifFailed();
        };
    };

    void Calculate(Point* E, Point* D, Point* vDE){
        // Współrzędne punktu E
        E->x = D->x + vDE->x;
        E->y = D->y + vDE->y;
        //cout << "== PointE: " << E->x << ", " << E->y << endl;
        cout << "------------------------------------------------------------------\n";
```

```
    };
};

/*//=============================================================================
    Wyznaczenie punktu F - operacje:
        (Punkt F jest symetrycznym odbiciem punktu E, względem odcinka AB (prosta f).
        Prosta f ma wzór ogólny f: y = mx + b)

        (Wzór na współrzędne punktu F został wyznaczony następująco:
            - Wyznaczenie współczynnika kierunkowego na podstawie punktów A i B
                m = (By - Ay) / (Bx - Ax)
            - Wyznaczenie wyrazu wolnego b dla f: Ay = m * Ax + b
                b = Ay - (m * Ax)
            - Równanie prostej przechodzącej przez punkt E, prostopadłej do prostej f
                g: y - Ey = -x/m + Ex/m
            - Wyznaczenie punktu przecięcia prostych f i g
                P: f = g
                P: m * x + b = -x/m + Ey + Ex/m
                P:  {Px = (Ex + m * Ey - m * b) / (m^2 + 1)} // podstawienie Px do wzoru prostej g
                    {Py = m * Px + b}                       // podstawienie Py do wzoru prostej f
            - Odbicie symetryczne punktu E (odległość punktu E od P == odległość punkt F od P)
                F:  {Fx = 2Px - Ex}
                    {Fy = 2Py - Ey}
            - Po podstawieniu wzorów na Px w równaniu punktu F, wychodzą nam wzory w metodzie Calcula
te())

        - LinearAB() - metoda wyznaczająca współczynnik kierunkowy m i wyraz wolny b:
            - Zmienne m, b

        - Calculate() - metoda wyznaczająca współrzęde punktu F:
            - Współrzędne x, y punktu F
*///=============================================================================
class CalcPointF {
    public:
    float b, m;

    void LinearAB(Point* A, Point* B, float& m, float& b){
        // Współ. kierunkowy m oraz wyraz wolny b funkcji f: y = mx + b (na której leży odcinek AD)
        m = (B->y - A->y) / (B->x - A->x);
        b = A->y - (m * A->x);
        cout << "== Współczynnik kierunkowy m = " << m << endl;
        cout << "== Wyraz wolny b = " << b << endl;
    };

    void Calculate(Point* E, Point* F, float& m, float& b){
        // Współrzędne punktu F
        F->x = ((1 - pow(m,2)) * E->x + (2 * m * E->y) - (2 * m * b)) / (1 + pow(m,2));
        F->y = (((pow(m,2) - 1) * E->y) + (2 * m * E->x) + (2 * b)) / (1 + pow(m,2));
        //cout << "== PointF: " << F->x << ", " << F->y << endl;
    };
};

int main() {

// Error Handlery zawarte w klasie ErrorHandlers
    ErrorHandlers ErrHand;

// TEST INPUT ================================
    // Point A = {2,3};
    // Point B = {4,6};
    // float r = 2;
    // float angle = 30;
// TEST INPUT ================================
```

```
// INPUT ===================================
// Punkty A i B
    Point A, B;
    while (true){

        A.x = ErrHand.GetValidatedInput("Podaj współrzędną x dla Punktu A: ");
        A.y = ErrHand.GetValidatedInput("Podaj współrzędną y dla Punktu A: ");
        cout << "Współrzędne punktu A: (" << A.x << ", " << A.y << ")\n";

        B.x = ErrHand.GetValidatedInput("Podaj współrzędną x dla Punktu B: ");
        B.y = ErrHand.GetValidatedInput("Podaj współrzędną y dla Punktu B: ");
        cout << "Współrzędne punktu B: (" << B.x << ", " << B.y << ")\n";


        if(A.x == B.x && A.y == B.y){
            ErrHand.SamePoints();
            continue;
        }

        if(A.x == 0 && B.x == 0){
            ErrHand.DivZero();
            continue;
        }

        break;
    }

// Promień okręgu - nie może być ujemny
    float r;
    while (true){
        r = ErrHand.GetValidatedInput("Podaj promień okręgu: ");

        if (r <= 0) {
            ErrHand.WrongRadius();
            continue;
        }

        cout << "Promień okręgu to: " << r << endl;
        break;
    }

// Kąt pomiędzy ramieniem strzałki a promieniem - w zakresie angle <10,45>
    float angle;
    while (true){
        angle = ErrHand.GetValidatedInput("Podaj kąt (10-45): ");

        if (angle < 10 || angle > 45){
            ErrHand.WrongAngle();
            continue;
        }

        cout << "Wybrany kąt to: " << angle << endl;
        break;
    }

// ========================================

// Przekształcenie wartości kątowej na radiany
    const float pi = 3.14159265358979323846f;
    float alpha = angle * (pi / 180.0f);
// ========================================

// Wyznaczenie punktu D
    Point D;
```

```
    CalcPointD CalcPointD;

    CalcPointD.VectorAB(&A, &B, &CalcPointD.UvAB);
    CalcPointD.Calculate(&D, &A, r, &CalcPointD.UvAB);

// Wyznaczenie Punktu E
    Point E;
    CalcPointE CalcPointE;

    CalcPointE.VectorAD(&D, &A, &CalcPointE.vAD);
    CalcPointE.VectorDE(&CalcPointE.vAD, &CalcPointE.vDE, alpha, CalcPointE.length);
    CalcPointE.VectorChecker(CalcPointE.length, r);
    CalcPointE.Calculate(&E, &D, &CalcPointE.vDE);

// Wyznaczenie Punktu F
    Point F;
    CalcPointF CalcPointF;

    CalcPointF.LinearAB(&A, &B, CalcPointF.m, CalcPointF.b);
    CalcPointF.Calculate(&E, &F, CalcPointF.m, CalcPointF.b);

// Output danych wyjściowych
    cout << "======================================================================\n";
    cout << "== Współrzędne punktu D: (" << D.x << ", " << D.y << ")" << endl;
    cout << "== Współrzędne punktu E: (" << E.x << ", " << E.y << ")" << endl;
    cout << "== Współrzędne punktu F: (" << F.x << ", " << F.y << ")" << endl;
    cout << "======================================================================\n";

// KONIEC :D
    return 0;
}
```

5. Testy

Sytuacja 1

- Wartości

A (x)	A (y)	B (x)	B (y)	r	alpha
2	3	4	6	2	30

- Rezultat - Działania wykonane prawidłowo, program nie zwrócił błędu (1)

  Output:

  == Współrzędne punktu D: (3.1094, 4.6641)
  == Współrzędne punktu E: (3.08366, 4.26493)
  == Współrzędne punktu F: (2.75084, 4.48681)

Sytuacja 2

- Wartości

A (x)	A (y)	B (x)	B (y)	r	alpha
,2	,x	A	2-c	--2	3-

- Rezultat - Program nie przejdzie dalej póki nie zostaną podane poprawne wartości, powraca do ustalenia wartości błędnej zmiennej

  Output: *To nie jest liczba, albo użyłeś przecinka jako separatora dziesiętnego - Użyj kropki*

Sytuacja 3

- Wartości

A (x)	A (y)	B (x)	B (y)	r	alpha
1	1	1	1	1	10

- Rezultat - Program wraca do ustalenia współrzędnych A

  Output: *Punkty A i B nie mogą leżeć w tym samym miejscu*

Sytuacja 4

- Wartości

A (x)	A (y)	B (x)	B (y)	r	alpha
0	0	0	0.1	0.1	10

- Rezultat - Program wraca do ustalenia współrzędnych A

  Output:

  Punkty A i B leżą na osi OX. Nie będzie możliwe wyznaczenie punktu F (Dzielenie przez zero)

Sytuacja 5

- Wartości

A (x)	A (y)	B (x)	B (y)	r	alpha
1	2	3	4	0	10

- Rezultat - Program wraca do ustalenia wartości r (tak samo dla wartości ujemnych)

  Output: *Podaj liczbę większą niż 0*

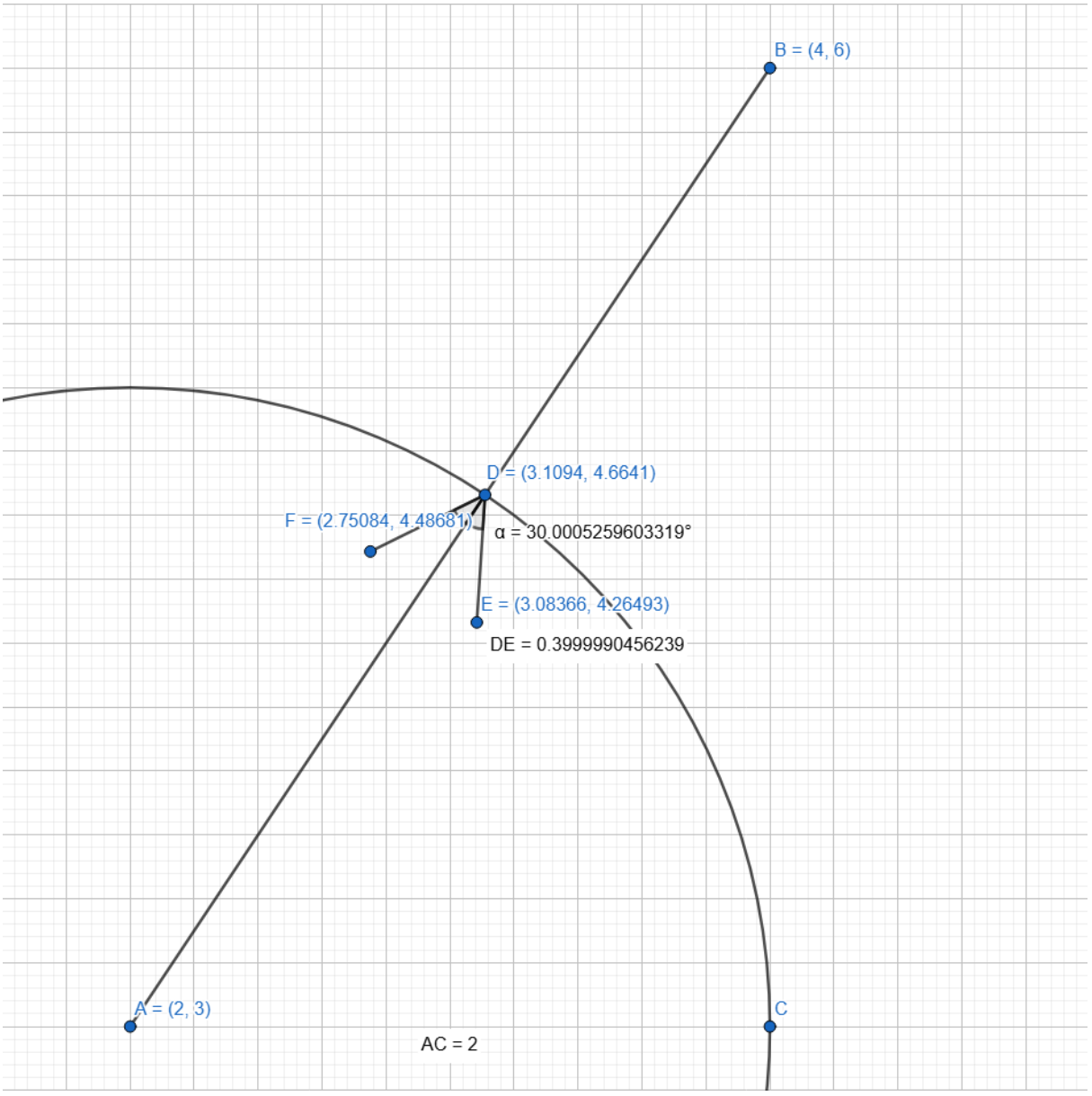
Sytuacja 6

- Wartości

A (x)	A (y)	B (x)	B (y)	r	alpha
1	2	3	4	1	2

- Rezultat - Program wraca do ustalenia wartości kąta (tak samo dla wartości przekraczających 45, oraz wartości ujemnych)

  Output: *Kąt ma być w zakresie 10-45 stopni*



(1) - Przedstawienie wizualne wyników z sytuacji nr 1 w programie GeoGebra

6. Wnioski

Rozwiązaniem zastosowanym w napisaniu tego programu było podzielenie funkcji na klasy.
Każda klasa (CalcPointD, CalcPointE, CalcPointF) zawiera metody wyliczające wartości potrzebne do wyznaczenia punktów z Danych Wyjściowych.

Program zawiera komentarze objaśniające właściwości metod każdej klasy.

Do określenia współrzędnych punktów użyto struktury {float x, float y} w celu zachowania przejrzystości kodu.

Klasa ErrorHandlers zawiera w sobie metody służące jako sposób, na ewentualne błędne dane wejściowe:

- getValidatedInput() - Przeprowadza konwersję danych wejściowych string → float, po czym weryfikuje czy nie wystąpił błąd w strumieniu danych (stream.eof())
- SamePoints() - Informuje użytkownika o przewidywanym błędzie w kalkulacjach (’nan’) w wyniku nieróżności punktów A i B
- DivZero() - Informuje użytkownika o przewidywanym błędzie w kalkulacjach (’nan’) w wyniku położenia punktów A i B na osi OX
- WrongAngle() - Informuje użytkownika o niespełnieniu założeń zadania - wykroczeniu wartości kąta poza zakres 10-45 stopni
- WrongRadius() - Informuje użytkownika o niespełnieniu założeń zadania - podanie zerowej lub ujemnej wartości promienia
- VectorVerifFailed() - W przypadku niespełnieniu założenia zadania (Długość Ramienia Strzałki (Wektor DE) jest pięciokrotnie mniejszy od promienia okręgu (r)) terminuje działanie programu (kod 100)

To rozwiązanie pomaga w poradzeniu sobie z błędami wynikającymi z inputu użytkownika - niedopuszczenie do nieskończonych pętli czy błędnych wyników

Kod zawiera przykładowe wartości (użyte w sytuacji nr 1, testach) które przyspieszą proces debuggowania. Analogicznie jest z outputem do konsoli, gdzie program nie tylko wydaje współrzędne punktów, ale i wartości wszystkich zmiennych pośrednich - współrzędne wektorów, ich długości itp.

W wyniku limitów liczb zmiennoprzecinkowych wyniki mają rację bytu po zaokrągleniu do części dziesięciotysięcznych.