

Programowanie Matematyczne

Komputery jak i języki programowania początkowo były tworzone w celu ułatwienia wykonywania skomplikowanych obliczeń matematycznych. Naukowcy i inżynierowie różnych branż bardzo często w swej pracy wykorzystują wyniki obliczeń wykonanych na setkach a nawet tysiącach wartości początkowych. Często, aby wyznaczyć np. dopuszczalną masę elementu maszyny czy budowli należy rozwiązać układ kilkuset równań opisujących zachodzące zależności. Do takich właśnie celów były początkowo projektowane komputery. Język C/C++ daje znakomite możliwości rozwiązywania zagadnień matematycznych. W rozdziale tym zostaną przedstawione rozwiązania podstawowych problemów algebraicznych, na których opierają się metody macierzowe rozwiązywania układów równań liniowych. Wprowadzenie w tematykę zastosowań matematyki jest jednym z kluczowych etapów w trakcie nauki programowania. Większość operacji, jakie w tle wykonują programy komputerowe wiąże się właśnie w wykorzystaniem wiedzy, jaką przekazuje matematyka.

Zrozumienie zasad implementowania rozwiązań metod obliczeniowych pozwoli lepiej zrozumieć kolejny rozdział książki, w który przedstawiono specjalistyczne metody wykorzystania możliwości, jakie daje język C/C++ w rozwiązywaniu skomplikowanych zadań matematycznych czy symulacji obiektów. W rozdziale tym zostaną omówione podstawy metod macierzowych rozwiązywania układów równań. Jest to jeden z najważniejszych problemów matematycznych, technicznych czy ekonomicznych zastosowań możliwości obliczeniowych komputerów. W dzisiejszym świecie wiele rozwiązań czy decyzji, jakie podejmuje komputer opiera się o rozwiązanie układu równań wielu zmiennych, które stanowią model badanego obiektu. W ten właśnie sposób są opisane warunki, jakimi należy się kierować przy wyborze odpowiedniej wartości temperatury, ciśnienia, napięcia czy innych charakterystyk układów technicznych. Analogicznie są opisywane rozwiązania problemów ekonomicznych. Przedstawienie dziedziny metod macierzowych jest wkładem w rozwój prac nad opracowaniem rozwiązań problemów inżynierskich, ekonomicznych czy czysto naukowych. W literaturze [10, 12, 16, 18, 26, 36, 48, 68, 92, 94, 108, 110] można znaleźć wiele odniesień do znaczenia metod macierzowych w zastosowaniach informatyki w technice i ekonomii.

Autorzy chcą przedstawić efektywne algorytmy, które są w stanie operować na macierzach. Przedstawione w tym rozdziale metody macierzowe są procedurami, które stanowią najczęściej elementy pośrednie całych algorytmów symulacji stanów badanych obiektów. Dlatego proponowane rozwiązania są opracowane w taki sposób, aby zwiększyć efektywność. W literaturze [4, 6, 8, 16-17, 26, 35, 44, 56, 60] dużą uwagę poświęca się efektywności budowanych algorytmów. Znaczna część algorytmów przedstawionych w książkach [30, 36, 68, 79, 92, 108, 110] pokazuje możliwości szybkiej i dokładnej transformacji macierzy. Autorzy zdecydowali się pokazać podobne rozwiązania postawionych w literaturze [30, 92, 110] problemów. W rozdziale tym zostaną pokazane praktyczne przykłady zastosowania prostych metod macierzowych do transponowania macierzy, odwracania macierzy czy metoda eliminacji niewiadomych. Pomimo, iż są to znane problemy matematyki i w literaturze [36, 92, 108, 110] można spotkać propozycje algorytmów realizujących takie metody w podobny sposób, to jednak niestety większość z przedstawionych propozycji nie pokazuje kodu realizującego te metody. W tym i kolejnym rozdziale zostaną zaprezentowane przykłady programów, które realizują omawiane metody.

Jest to ułatwienie dla inżynierów i naukowców, którzy w swej codziennej pracy korzystają z rozwiązań omawianych w tej książce. Stosując przedstawione propozycje kodów gotowych procedur w dużo łatwiejszy sposób będą oni mogli budować algorytmy symulujące pracę obiektów czy rozwiązujące zagadnienia naukowe. Autorzy mają nadzieję, że przedstawione rozwiązania staną się ułatwieniem dla programistów szukających efektywnych i przetestowanych algorytmów opracowanych dla metod macierzowych.

Zastosowania geometryczne

Na początku pokażemy prosty przykład wyznaczenia pola trójkąta znając jedynie współrzędne jego wierzchołków w układzie kartezjańskim. Jest to zadanie proste i ma za cel zainteresować Czytelnik o mniejszym doświadczeniu możliwościami zastosowania języka C/C++ w matematyce i informatyce. Do rozwiązania tego zadania będzie potrzebna znajomość kilku prostych wzorów matematycznych, znanych Czytelnikowi z kursu algebry liniowej czy geometrii analitycznej. Czytelnik zechce się zapoznać ze wzorem opisującym pole trójkąta za pomocą długości wektorów w następujący sposób:

$$P_{ABC} = \frac{1}{2} \cdot |d(\overrightarrow{AB}, \overrightarrow{AC})|.$$

Wzór ten oznacza, że rozpatrujemy parę wektorów \overrightarrow{AB} oraz \overrightarrow{AC} rozpiętych na punktach $A = (x_A, y_A)$, $B = (x_B, y_B)$, $C = (x_C, y_C)$ które charakteryzują odpowiednie współrzędne wektorowe

$$\overrightarrow{AB} = [u_x, u_y] = [x_B - x_A, y_B - y_A],$$

$$\overrightarrow{AC} = [v_x, v_y] = [x_C - x_A, y_C - y_A].$$

Współrzędne wektorowe możemy wykorzystać do wyznaczenia długości boków trójkąta. Zatem wzór opisujący pole trójkąta można zapisać w postaci odpowiedniego wyznacznika

$$P_{ABC} = \frac{1}{2} \cdot |d(\overrightarrow{AB}, \overrightarrow{AC})| = \frac{1}{2} \cdot \begin{vmatrix} x_B - x_A & y_B - y_A \\ x_C - x_A & y_C - y_A \end{vmatrix}$$

Korzystając ze wzoru pozwalającego na wyliczenie długości odcinka pomiędzy dowolnymi punktami płaszczyzny możemy wyznaczyć wszystkie długości boków trójkąta.

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

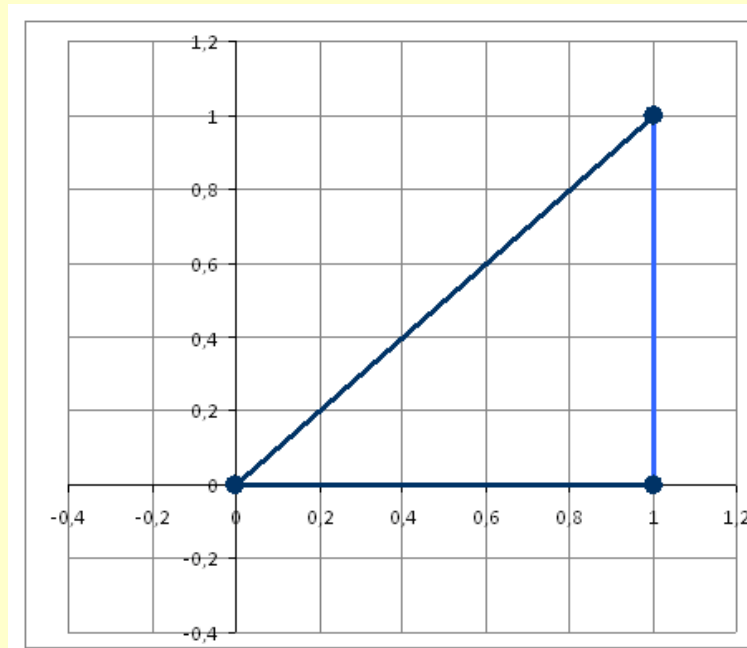
Mając na uwadze powyższe wzory możemy się zastanowić, w jaki sposób można je wykorzystać w praktyce, aby móc łatwo i efektywnie wyznaczać pole figur geometrycznych. Poprzez efektywność programu będziemy zawsze rozumieli nie tylko szybkość i precyzję obliczeniową danej procedury. Ważnym jest, aby była ona zbudowana w taki sposób, że programista będzie w stanie przenosić różne jej fragmenty do innych swoich programów. Taką budowę powinniśmy oprzeć o odpowiednio zbudowane funkcje, które potrafiłyby współdziałać ze sobą wymieniając się danymi tak, aby w efekcie program zwracał poszukiwaną wartość.

Przykład 53

Napisać program pobierający z wejścia standardowego współrzędne trzech wierzchołków trójkąta. Następnie wyznaczający pole trójkąta rozpiętego na tych wierzchołkach i zapisujący wartość wyznaczoną oraz pobrane współrzędne do pliku wyjściowego wyniki.txt.

Dla przykładowych obliczeń został przyjęty pokazany na kolejnej ilustracji trójkąt o trzech wierzchołkach w punktach $A=(0,0)$, $B(1, 1)$, $C=(1,0)$.

Trójkąt o zadanych wierzchołkach



Zatem stosując wzory na pole trójkąta otrzymamy następujące wyniki obliczeń.

$$\overrightarrow{AB} = [x_B - x_A, y_B - y_A] = [1 - 0, 1 - 0] = [1, 1]$$

$$\overrightarrow{AC} = [x_C - x_A, y_C - y_A] = [1 - 0, 0 - 0] = [1, 0]$$

$$\overrightarrow{BC} = [x_C - x_B, y_C - y_B] = [1 - 0, 0 - 1] = [0, -1]$$

$$|AB| = \sqrt{u_x^2 + u_y^2} = \sqrt{1 + 1} = \sqrt{2}$$

$$|AC| = \sqrt{v_x^2 + v_y^2} = \sqrt{1 + 0} = \sqrt{1} = 1$$

$$|BC| = \sqrt{w_x^2 + w_y^2} = \sqrt{0 + 1} = 1$$

$$P_{ABC} = \frac{1}{2} \cdot |d(\overrightarrow{AB}, \overrightarrow{AC})| = \frac{1}{2} \cdot |1 \cdot 0 - 1 \cdot 1| = 0,5$$

Czytelnik zechce zapoznać się z kodem programu realizującego zadanie pokazanym poniżej i porównać wyliczone wartości ze zwróconymi przez program a pliku *wyniki.txt*, którego wartość pokazano na ilustracji.

```

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>

double bok(double x1,double y1,double x2, double y2)
{
    double dlugosc;
    dlugosc = sqrt( pow((x2-x1),2)+pow((y2-y1),2));
    return(dlugosc);
}

double pole(double ax, double ay, double bx,
            double by, double cx, double cy)
{
    double powierzchnia;
    powierzchnia = fabs((bx-ax)*(cy-ay)-(by-ay)*(cx-ax)) /2;
    return powierzchnia ;
}

int main(int argc, char *argv[])
{
    double ax, ay, bx, by, cx, cy, ab;
    printf("Program oblicza pole trojkata\n");
    printf("na podstawie jego wierzchołkow, prosze podac ");

```

```
// Deklaracja bibliotek.
```

```

/* Biblioteka matematyczna zawierająca
funkcje takie jak: sort(), pow(), fabs(). */
/* Deklaracja funkcji obliczającej
długości odpowiednich boków trójkąta
według wzorów (10.1.5). */

```

```
/* Deklaracja funkcji obliczającej pole
trójkąta według wzoru (10.1.4). */
```

```
// Funkcja główna programu.
```

```
printf("ich współrzędne:\n\n ");
printf("Podaj współrzędne punktu A:\n");
printf("x: ");
scanf("%lf", &ax);
printf("y: ");
scanf("%lf", &ay);
printf("Podaj współrzędne punktu B:\n");
printf("lx: ");
scanf("%lf", &bx);
printf("y: ");
scanf("%lf", &by);
printf("Podaj współrzędne punktu C:\n");
printf("x: ");
scanf("%lf", &cx);
printf("y: ");
scanf("%lf", &cy);
printf("\n");
FILE *plik;
if ((plik=fopen("wyniki.txt", "w")) == NULL)
{
    printf("Nie moge otworzyc pliku plik tekstowy wyniki.txt do zapisu!\n");
    exit(1);
}
```

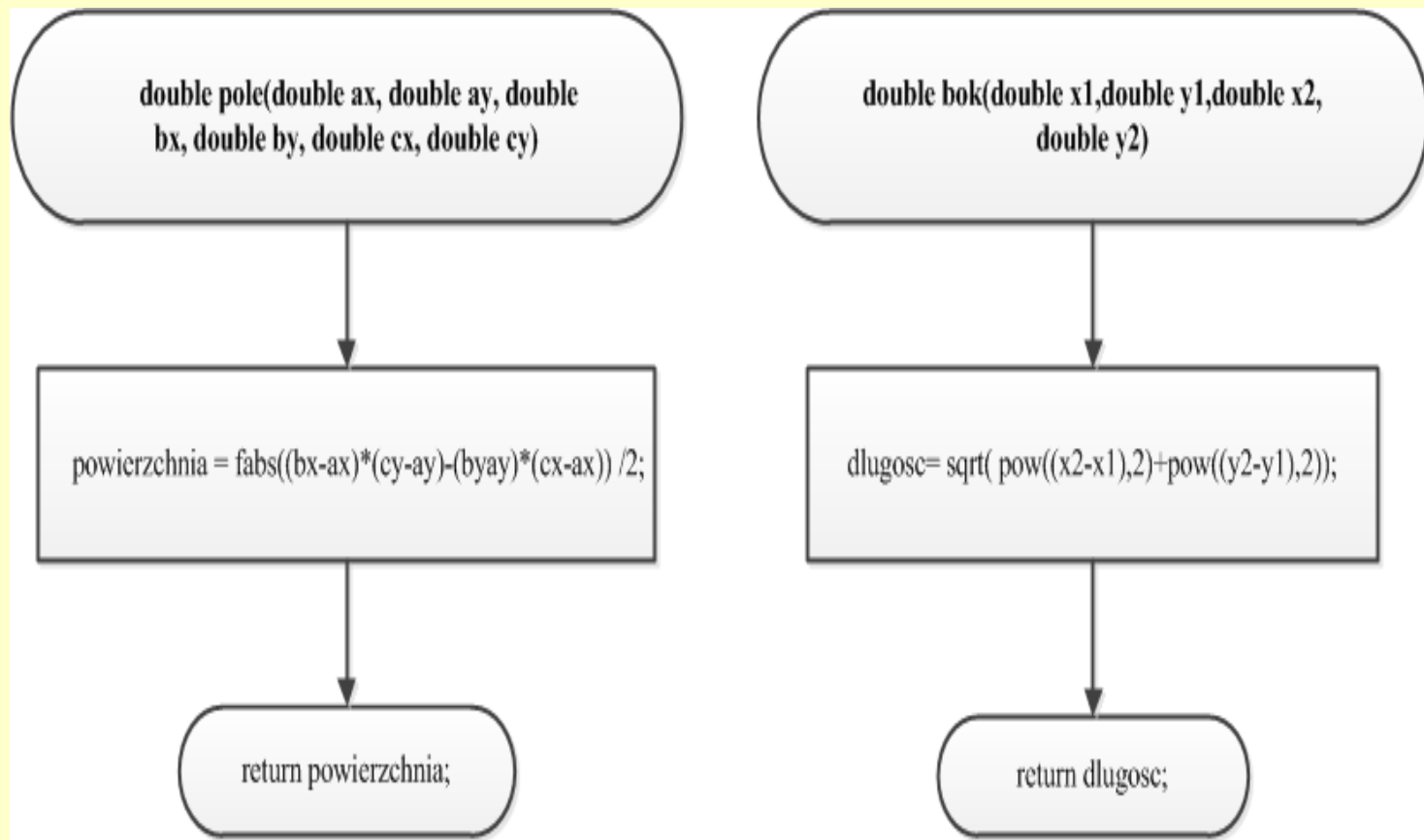
```
/* Procedura pobierająca
współrzędne punktów.*/
```

```
/* Deklaracja pliku do zapisu
danych.*/
/* Próba otwarcia pliku do
zapisu.*/
```

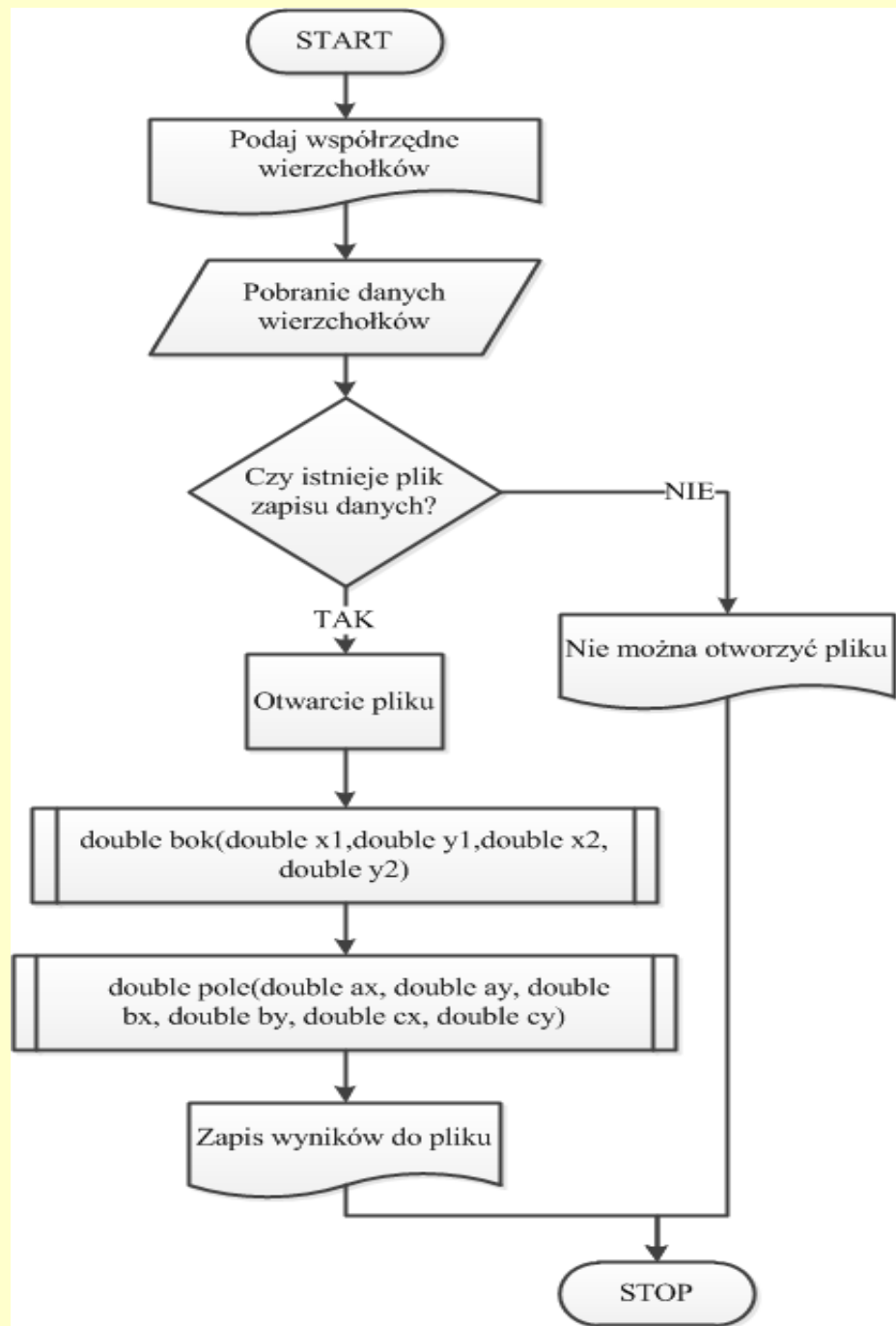
```
fprintf(plik, "Wspolrzedne punktow:\n");
printf("A(%.2f, %.2f)\nB(%.2f, %.2f)\nC(%.2f,%.2f)\n",
        ax, ay, bx, by, cx, cy);
fprintf(plik, "Dlugosc odcinka AB: %.2f\n",
        bok(ax, ay, bx, by));
fprintf(plik, "Dlugosc odcinka BC: %.2f\n",
        bok(bx, by, cx, cy));
fprintf(plik, "Dlugosc odcinka CA: %.2f\n",
        bok(cx, cy, ax, ay));
fprintf(plik, "Pole trojkata: %.2f\n" ,
        pole(ax,ay, bx, by, cx, cy));
fclose (plik);
_getch();
return 0;
}
```

```
/* Zamknięcie otwartego pliku
danych.*/
```

Schemat blokowy funkcji obliczania długości boku i pola trójkąta

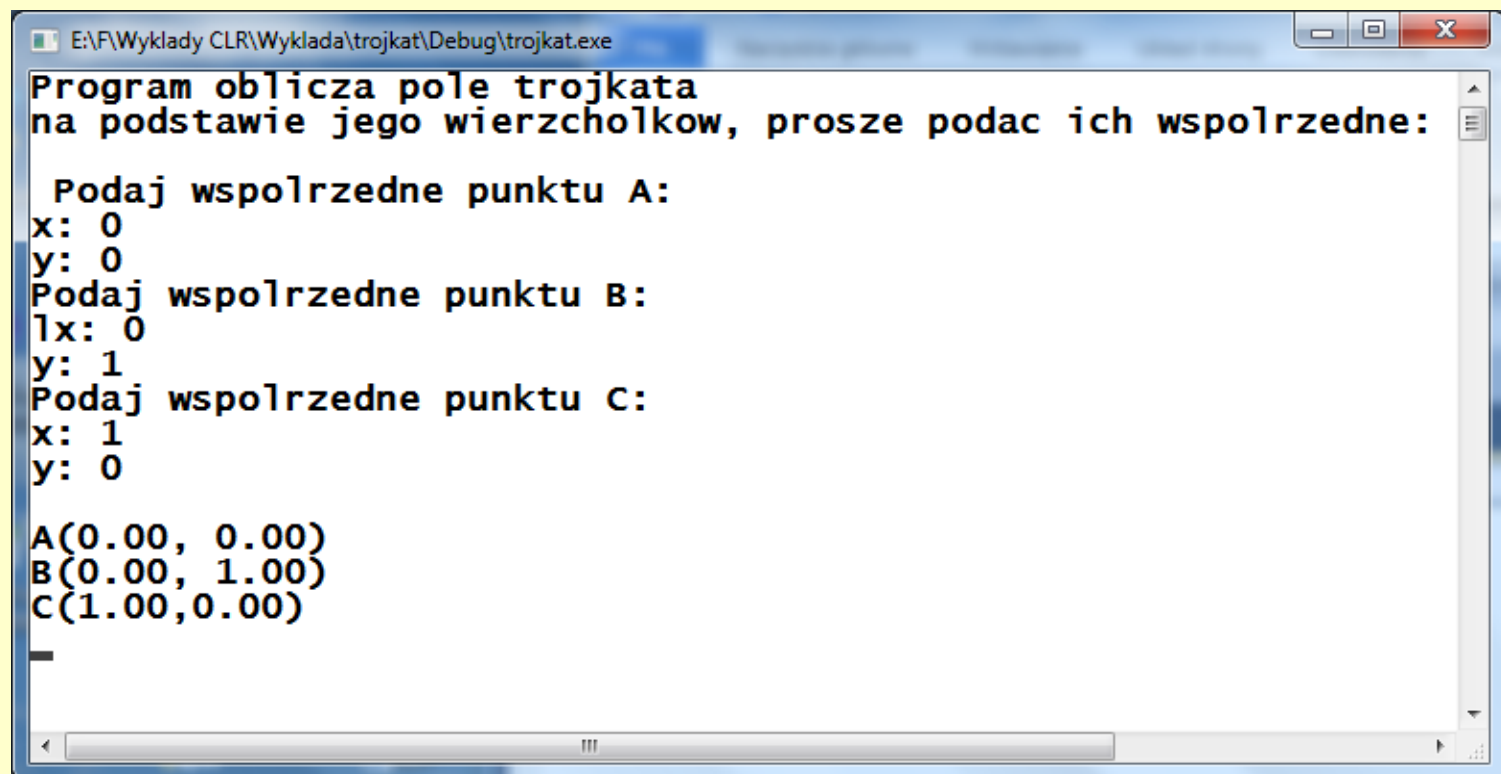


Schemat
blokowy
programu
obliczania
pola
trójkąta.



Program obliczający pole trójkąta o zadanych wierzchołkach

Powyższe schematy blokowe pokazują, w jaki sposób poszczególne procedury pobierają dane oraz współpracują ze sobą realizując zadanie postawione przed programem. Działanie programu oraz wynik zapisane do pliku zostały przedstawione na poniższej ilustracji.



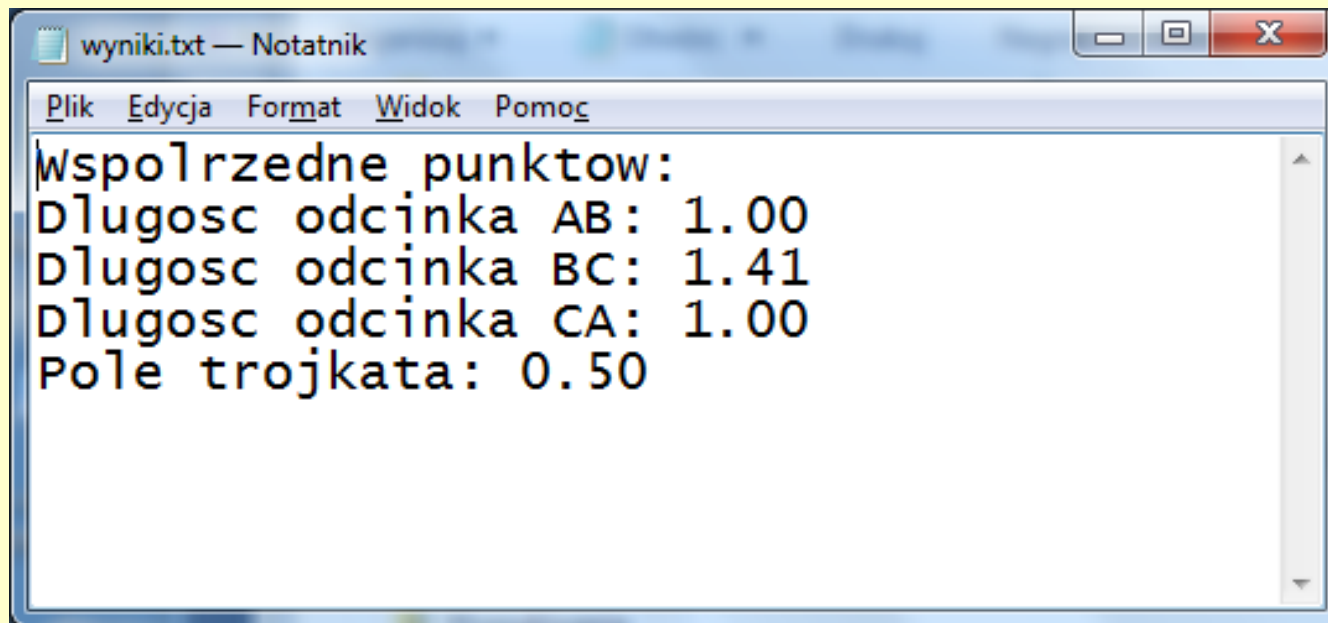
```
E:\F\Wyklady CLR\Wyklada\trojkat\Debug\trojkat.exe
Program oblicza pole trojkata
na podstawie jego wierzchołkow, prosze podac ich wspolrzedne:

Podaj wspolrzedne punktu A:
x: 0
y: 0
Podaj wspolrzedne punktu B:
lx: 0
y: 1
Podaj wspolrzedne punktu C:
x: 1
y: 0

A(0.00, 0.00)
B(0.00, 1.00)
C(1.00,0.00)
```

Plik danych wejściowych o zadanych wierzchołkach

Natomiast sam plik danych, jakie zostaną zwrócone przez program wyglądać będzie w następujący sposób.

A screenshot of a Windows Notepad window titled 'wyniki.txt — Notatnik'. The window has a menu bar with 'Plik', 'Edycja', 'Format', 'Widok', and 'Pomoc'. The text inside the window is as follows:

```
współrzędne punktów:  
Długość odcinka AB: 1.00  
Długość odcinka BC: 1.41  
Długość odcinka CA: 1.00  
Pole trójkąta: 0.50
```

Układ równań liniowych

Podstawowym celem tego rozdziału jest podanie metod pozwalających efektywnie wyznaczać rozwiązanie układu równań liniowych postaci:

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + \cdots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + \cdots + a_{nn}x_n = b_n \end{cases} \quad (1)$$

Rozwiązaniem układu (1) nazywamy taki n – elementowyciąg $x_1^*, x_2^*, \dots, x_n^* \in R$, spełniający zależność:

$$\begin{cases} b_1 - a_{11}x_1^* - \cdots - a_{1n}x_n^* = 0 \\ b_2 - a_{21}x_1^* - \cdots - a_{2n}x_n^* = 0 \\ \dots \\ b_n - a_{n1}x_1^* - \cdots - a_{nn}x_n^* = 0 \end{cases} \quad (2)$$

Równanie (1) możemy zapisać następująco:

$$\sum_{j=1}^n a_{ij}x_j = b_i \quad \text{dla } i = 1, \dots, n \quad (3)$$

lub w postaci macierzowej:

$$\begin{aligned} Ax &= b, & \text{gdzie } A &\in R^{n \times n}, \\ x, b &\in R^n \end{aligned} \quad (4)$$

Zależność (3) – warunku, że ciąg $x_1^*, x_2^*, \dots, x_n^* \in R$ jest rozwiązaniem możemy zapisać:

$$b_i - \sum_{j=1}^n a_{ij}x_j^* = 0 \quad \text{dla } i = 1, \dots, n \quad (5)$$

lub w postaci macierzowej:

$$\begin{array}{l} b - Ax^* = 0, \quad \text{gdzie } A \in R^{n \times n}, \\ x^*, b \in R^n \end{array} \quad (6)$$

Metoda Gaussa rozwiązywania układów równań liniowych

Założmy obecnie, że układ równań liniowych (3) ma dokładnie jedno rozwiązanie. Algorytm eliminacji Gaussa wyznaczania rozwiązania możemy podzielić na dwa etapy:

1. Doprowadzenie układu równań w którym macierz układu jest macierzą trójkątną górną tzn. elementy pod przekątną są równe zero i na przekątnej różne od zera.
2. Wyznaczenie rozwiązania układu równań (1) z układu równań wyznaczonego w punkcie 1. tzn. z układu równań w którym macierz jest macierzą trójkątną górną.

Przedstawmy metodę eliminacji Gaussa dla układu o dwóch niewiadomych:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1 \\ a_{21}x_1 + a_{22}x_2 = b_2 \end{cases} \quad (7)$$

Oznaczmy układ równań (7) następująco:

$$\begin{cases} a_{11}^1x_1 + a_{12}^1x_2 = b_1^1 \\ a_{21}^1x_1 + a_{22}^1x_2 = b_2^1 \end{cases} \quad (8)$$

Oznaczenie to pozwala zapisać kolejne przekształcenia układu równań liniowych i indeks górny jeden oznacza formalnie przepisanie układu równań liniowych do macierzy co możemy zapisać:

$$\begin{bmatrix} a_{11}^1 & a_{12}^1 & b_1^1 \\ a_{21}^1 & a_{22}^1 & b_2^1 \end{bmatrix} \quad (9)$$

W pierwszym kroku wybieramy element w kolumnie pierwszej według którego dokonamy eliminacji niewiadomych pod główną przekątną. W obliczeniach na papierze wygodnie jest wybrać element o wartości jeden, jeśli taki istnieje. W obliczeniach numerycznych wybieramy element największy, co do modułu i przestawiamy wiersze (jeśli dokonujemy zamiany). Obecnie możemy założyć, że element na głównej przekątnej jest różny od zera, gdyż układ równań posiada dokładnie jedno rozwiązanie. Następnie dokonujemy eliminacji pierwszej niewiadomej z równania drugiego za pomocą wzorów:

$$a_{22}^2 = a_{22}^1 - \frac{a_{21}^1 a_{12}^1}{a_{11}^1}$$

$$b_2^2 = b_2^1 - \frac{a_{21}^1 b_1^1}{a_{11}^1}$$

$$a_{21}^2 = 0$$

Przekształcenie to jest równoważne pomnożeniu wiersza pierwszego wiersza równania (7) przez $-\frac{a_{21}^1}{a_{11}^1}$ i dodaniu do wiersza drugiego. W wyniku przekształcenia otrzymujemy:

$$\begin{cases} a_{11}^1 x_1 + a_{12}^1 x_2 = b_1^1 \\ \left(a_{22}^1 - \frac{a_{21}^1 a_{12}^1}{a_{11}^1} \right) x_2 = b_2^1 - \frac{a_{21}^1 b_1^1}{a_{11}^1} \end{cases} \quad (11)$$

$$\begin{cases} a_{11}^1 x_1 + a_{12}^1 x_2 = b_1^1 \\ a_{22}^2 x_2 = b_2^2 \end{cases} \quad (12)$$

$$\begin{bmatrix} a_{11}^1 & a_{12}^1 & b_1^1 \\ 0 & a_{22}^2 & b_2^2 \end{bmatrix} \quad (13)$$

W ten sposób przekształciliśmy układ równań do układu równań w którym macierz układu równań jest macierzą trójkątną górną. Przechodzimy do punktu 2. wyznaczenia rozwiązania. Wyznaczamy x_2 :

$$x_2 = \frac{b_2^1}{a_{22}^1}$$

Następnie wyznaczamy x_1 :

$$x_1 = \frac{b_1^1 - a_{12}^1 x_2}{a_{11}^1}$$

Przedstawmy rozwiązanie układu na przykładzie:

$$\begin{cases} 2x_1 + x_2 = 3 \\ x_1 + 2x_2 = 3 \end{cases}$$

Do obliczeń papierowych wygonie jest przestawić wiersze:

$$\begin{cases} x_1 + 2x_2 = 3 \\ 2x_1 + x_2 = 3 \end{cases}$$

Dokonujemy przekształcenia:

$$a_{22}^2 = a_{22}^1 - \frac{a_{21}^1 a_{12}^1}{a_{11}^1} = 1 - \frac{2 \cdot 2}{1} = -3$$
$$b_2^2 = b_2^1 - \frac{a_{21}^1 b_1^1}{a_{11}^1} = 3 - \frac{2 \cdot 3}{1} = -3$$

Zapisujemy układ równań:

$$\begin{cases} x_1 + 2x_2 = 3 \\ -3x_2 = -3 \end{cases}$$

Wyznaczamy $x_2 = 1$ i wstawiamy do równania pierwszego:

$$x_1 = \frac{b_1^1 - a_{12}^1 x_2}{a_{11}^1} = \frac{3 - 2 \cdot 1}{1} = 1$$

Metoda eliminacji niewiadomych (metoda Gaussa) daje się uogólnić na dowolny wymiar macierzy. Rozpatrzmy układ równań posiadający dokładnie jedno rozwiązanie:

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\ \cdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n = b_n \end{cases}$$

Oznaczamy układ równań:

$$\begin{cases} a_{11}^1x_1 + \cdots + a_{1n}^1x_n = b_1^1 \\ \cdots \\ a_{n1}^1x_1 + \cdots + a_{nn}^1x_n = b_n^1 \end{cases}$$

Wybieramy w pierwszej kolumnie element największy co do modułu:

$$|a_{t1}^1| = \max_{1 \leq i \leq n} |a_{i1}^1|$$

Jeśli jest to konieczne przestawiamy wiersz pierwszy z wierszem t .

Możemy założyć, że $a_{11}^1 \neq 0$, gdyż układ posiada dokładnie jedno rozwiązanie i w pierwszej kolumnie istnieje element niezerowy. Dokonujemy przekształcenia:

$$a_{i1}^2 = \frac{a_{i1}^1}{a_{11}^1} \quad \text{dla } i = 2, \dots, n$$

$$a_{ij}^2 = a_{ij}^1 - a_{i1}^2 \cdot a_{1j}^1 \quad \text{dla } i = 2, \dots, n \quad j = 2, \dots, n$$

$$b_i^2 = b_i^1 - a_{i1}^2 \cdot b_1^1 \quad \text{dla } i = 2, \dots, n$$

W wyniku otrzymujemy macierz zawierającą zera w pierwszej kolumnie. Ponieważ wiadomo, że w pierwszej kolumnie występują zera możemy zapamiętać w niej obliczane elementy:

$$a_{i1}^2 = \frac{a_{i1}^1}{a_{11}^1} \quad \text{dla } i = 2, \dots, n$$

Na kroku k nasz układ wygląda następująco:

$$\left\{ \begin{array}{l} a_{11}^1 x_1 + \dots + a_{1k}^1 + \dots + a_{1n}^1 x_n = b_1^1 \\ \dots \\ a_{kk}^k x_k + \dots + a_{kn}^k x_n = b_k^k \\ \dots \\ a_{nk}^k x_k + \dots + a_{nn}^k x_n = b_n^k \end{array} \right.$$

Wybieramy w k – *tej* kolumnie element największy co do modułu:

$$|a_{tk}^k| = \max_{k \leq i \leq n} |a_{ik}^k|$$

Jeśli jest to konieczne k – *ty* wiersz pierwszy z wierszem t .

Możemy założyć, że $a_{kk}^k \neq 0$, gdyż układ posiada dokładnie jedno rozwiązanie i w k – *tej* kolumnie istnieje element niezerowy. Dokonujemy przekształcenia:

$$a_{ik}^{k+1} = \frac{a_{ik}^k}{a_{kk}^k} \quad \text{dla } i = k + 1, \dots, n$$

$$a_{ij}^{k+1} = a_{ij}^k - a_{ik}^{k+1} \cdot a_{kj}^k \quad \text{dla } i = k + 1, \dots, n \\ j = k + 1, \dots, n$$

$$b_i^{k+1} = b_i^k - a_{ik}^{k+1} \cdot b_k^k \quad \text{dla } i = k + 1, \dots, n$$

W wyniku otrzymujemy macierz zawierającą zera w $k - tej$ kolumnie. Ponieważ wiadomo, że w $k - tej$ kolumnie występują zera możemy zapamiętać w niej obliczane elementy:

$$a_{ik}^{k+1} = \frac{a_{ik}^k}{a_{kk}^k} \quad dla \ i = k + 1, \dots, n$$

Po $n - 1$ krokach otrzymujemy układ równań:

$$\left\{ \begin{array}{l} a_{11}^1 x_1 + a_{12}^1 x_2 \dots + a_{1n}^1 x_n = b_1^1 \\ a_{22}^2 x_2 + \dots + a_{2n}^2 x_n = b_2^2 \\ \dots \\ a_{nn}^n x_n = b_n^n \end{array} \right.$$

Możemy założyć, że $a_{nn}^n \neq 0$, gdyż układ posiada dokładnie jedno rozwiązanie i możemy obliczyć:

$$x_n = \frac{b_n^n}{a_{nn}^n}$$

Pozostałe niewiadome obliczamy ze wzoru:

$$x_i = \frac{b_i^i - \sum_{j=i+1}^n a_{ij}^i x_j}{a_{ii}^i}$$

Złożoność obliczeniowa algorytmu eliminacji niewiadomych rozwiązywania układu równań liniowych.

Ponieważ w każdym kroku k przekształcenia wykonujemy $n - k + 1$ operacji dodawania i mnożenia (mnożenie i dzielenie traktujemy tak samo) to do wyznaczenia rozwiązania układu równań liniowych potrzebujemy

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n \approx \frac{1}{3}n^3$$

Operacji dodawania i mnożenia. Należy podkreślić , że metoda wyznacznikowa wymaga n^4 operacji dodawania i mnożenia.

Przedstawiony algorytm eliminacji niewiadomych niestety nie będzie działał poprawnie w przypadku napotkania zera na głównej przekątnej w macierzy współczynników. Aby algorytm mógł działać poprawnie należy zastosować procedurę wyboru elementu głównego w kolumnie. Następnie zamienić wiersze tak, aby element o największej wartości modułowej znalazł się na głównej przekątnej. Wybór elementu największego, co do modułu dokonujemy numerycznie zmniejszając występujące w czasie obliczeń błędy zaokrągleń

$$\left\{ \begin{array}{l} a_{11}^1 x_0 + a_{12}^1 x_1 + \dots + a_{1n}^1 x_n = b_1^1 \\ a_{k1}^k x_0 + a_{k2}^k x_1 + \dots + a_{kn}^k x_n = b_k^k \\ \vdots \\ a_{t1}^t x_0 + a_{t2}^t x_1 + \dots + a_{tn}^t x_n = b_t^t \\ a_{n1}^n x_0 + a_{n2}^n x_1 + \dots + a_{nn}^n x_n = b_n^n \end{array} \right. ,$$

Wybór elementu głównego

gdzie kierujemy się kryterium maksymalizującym $|a_{tk}^k| = \max_{k \leq i \leq n} |a_{ik}^k|$. Przestawianie wierszy nie zmienia rozwiązania układu i jest wykonane na każdym kroku eliminacji niewiadomych. Jeżeli wybrany element główny jest równy zero oznacza to, że wszystkie elementy w kolumnie rozpoczynając od przekątnej są równe zero. W takim wypadku przerywamy eliminację niewiadomych, gdyż układ nie posiada dokładnie jednego rozwiązania.

W obliczeniach prowadzonych na liczbach typu *double* sytuacja nie jest tak oczywista. Na ogół otrzymujemy małą liczbę, którą należy już uznać za zero. Przerwanie procedury dokonujemy, zatem gdy $|a_{tk}^k| = \max_{k \leq i \leq n} |a_{ik}^k| < \varepsilon$ gdzie ε jest np. dowolnie małą ustaloną stałą. Dobór liczby zależy od rozwiązywanego zadania i najczęściej odbieramy go na podstawie własnego doświadczenia. Wskazówką może być rząd wielkości elementów macierzy A oraz wektora prawych stron układu równań b . Jeżeli macierz i wektor zawierają liczby jedno albo dwu cyfrowe najlepiej, jeśli przyjmujemy , co w zapisie kodu programu przyjmie $\varepsilon = 1e - 14$.

Oczywiście dobór wartości nie jest uniwersalny i zawsze można pokazać przykład, dla którego należałoby dobrać inny współczynnik przerwania eliminacji.

Przykład 55

Napisać funkcję obliczającą rozwiązanie układu równań liniowych za pomocą metody eliminacji niewiadomych, która będzie zwracać obliczony wyznacznik macierzy A .

Wyznacznik macierzy A jest równy iloczynowi elementów znajdujących się na głównej przekątnej pomnożonego przez -1 podniesione do potęgi ilości przestawień dokonanych podczas eliminacji niewiadomych według wzoru :

$$W = (-1)^p \prod_{i=1}^n a_{ii}^i$$

Implementacja przedstawionego algorytmu została pokazana w poniższym kodzie. Czytelnik zechce zapoznać się z zaprezentowanym rozwiązaniem.

```
// LU_Cplus.cpp : main project file.
```

```
#include<stdio.h>
```

```
#include<math.h>
```

```
#include<conio.h>
```

```
double eliminacja(int n, double **a, double *b)
```

```
{
```

```
    double w = 1.0;
```

```
    double h, s;
```

```
    int t;
```

```
    for (int k = 0; k < n - 1; k++)
```

```
    {
```

```
        h = abs(a[k][k]);
```

```
        t = k;
```

```
// Przyjęcie, że element największy co do modułu znajduje
```

```
// się na głównej przekątnej.
```

```
    for (int j = k + 1; j < n; j++)
// Znaleźnienie największego elementu w kolumnie.
    if (abs(a[j][k]) > h)
    {
        h = abs(a[j][k]);
        t = j;
    }
    if (h < 1e-14) return 0.0;
// Sprawdzenie, czy macierz jest osobliwa.
    if (t > k)
// Sprawdzenie, czy trzeba zamienić wiersze.
    {
        for (int j = 0; j < n; j++)
// Zamiana wierszy.
        {
            s = a[k][j]; a[k][j] = a[t][j]; a[t][j] = s;
        }
        s = b[k]; b[k] = b[t]; b[t] = s;
// Zamiana wyrazów wolnych.
        w = -w;
// Zmiana znaku wyznacznika.
    }
```

```
        h = a[k][k];
        for (int i = k + 1; i < n; i++)
// Podzielenie kolumny przez element znajdujący się na
// głównej przekątnej.
            a[i][k] /= h;
        w *= h;
// Pomnożenie wyznacznika przez element na głównej przekątnej.
        for (int i = k + 1; i < n; i++)
            // Eliminacja Gaussa
            {
                for (int j = k + 1; j < n; j++)
                    a[i][j] -= a[i][k] * a[k][j];
                b[i] -= a[i][k] * b[k];
            }
    }
    if (abs(a[n - 1][n - 1]) < 1e-14) return 0.0;
// Sprawdzenie osobliwości macierzy.
    b[n - 1] /= a[n - 1][n - 1];
// Obliczenie n - tej niewiadomej.
    w *= a[n - 1][n - 1];
// Obliczenie wyznacznika.
```



```
    for (int i = n - 2; i >= 0; i--)  
// Rozwiązanie układu równań liniowych.  
    {  
        s = 0.0;  
        for (int j = i + 1; j < n; j++)  
            s += a[i][j] * b[j];  
        b[i] = (b[i] - s) / a[i][i];  
    }  
    return w;  
}  
int main()  
{  
    int n;  
    printf("Podaj n:");  
    scanf("%d", &n);  
    if (n < 2)  
    {  
        printf("Niepoprawna liczba");  
        _getch();  
        return -1;  
    }  
}
```

```
double **a = new double*[n];
double *b = new double[n];

for (int i = 0; i < n; i++)
    a[i] = new double[n];

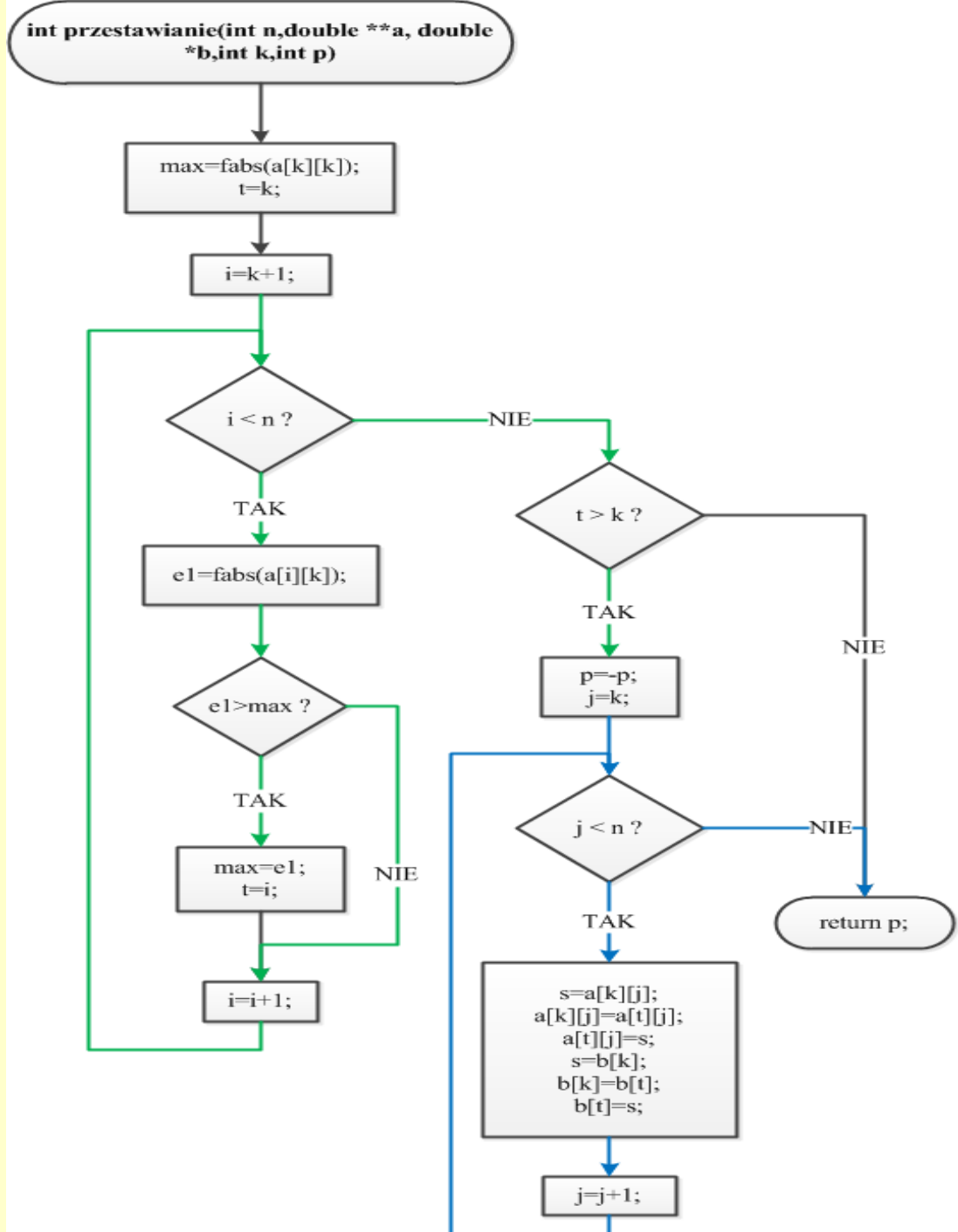
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        printf("a[%d][%d]=", i + 1, j + 1);
        scanf("%le", a[i] + j);

    }
    printf("b[%d]=", i + 1);
    scanf("%le", b + i);
}

double r = eliminacja(n, a, b);
```

```
printf("Rozwiazanie ukkladu rownan - wyznacznik=%g\n", r);  
if (r != 0.0)  
{  
    for (int i = 0; i < n; i++)  
        printf("x[%d]=%g\n", i + 1, b[i]);  
}  
else  
    printf("Macierz osobliwa");  
  
_getch();  
return 0;  
}
```

Schemat blokowy metody eliminacji niewiadomych



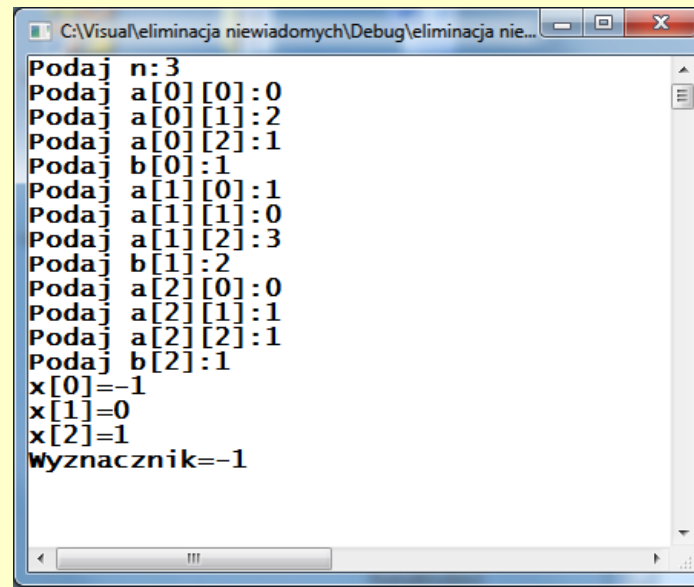
Metoda przestawiania elementów macierzy jest zastosowana w schemacie blokowym całego algorytmu, co na powyższym schemacie blokowym zostało oznaczone odpowiednim elementem. Czytelnik zechce przeanalizować oba schematy i zastanowi się, w jaki sposób uzupełnić powyższy schemat, aby omawiał cały kod programu rozwiązywania układów równań liniowych. Sprawdźmy działanie przedstawionego programu na przykładowym układzie równań

$$\begin{cases} 2x_1 + x_2 = 1 \\ x_0 + 3x_2 = 2 \\ x_1 + x_2 = 1 \end{cases}$$

Mamy, zatem następujące równanie w postaci macierzowej

$$\begin{bmatrix} 0 & 2 & 1 \\ 1 & 0 & 3 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}.$$

Rozwiązaniem układu równań jest wektor $\bar{x} = [-1, 0, 1]$, a wyznacznik macierzy jest w tym przypadku równy -1. Czytelnik może zaobserwować działanie programu na zamieszczonej poniżej ilustracji.



```
C:\Visual\eliminacja niewiadomych\Debug\eliminacja nie...
Podań n: 3
Podań a[0][0]: 0
Podań a[0][1]: 2
Podań a[0][2]: 1
Podań b[0]: 1
Podań a[1][0]: 1
Podań a[1][1]: 0
Podań a[1][2]: 3
Podań b[1]: 2
Podań a[2][0]: 0
Podań a[2][1]: 1
Podań a[2][2]: 1
Podań b[2]: 1
x[0]=-1
x[1]=0
x[2]=1
Wyznacznik=-1
```

Zajmijmy się teraz wyznaczeniem złożoności czasowej algorytmu eliminacji niewiadomych. W każdym z kroków eliminacji $k=0, \dots, n-2$ musimy wykonać $(n-k)^2$ operacji dodawania i mnożenia liczb. Natomiast w ostatnim kroku wykonujemy tylko jedno dzielenie. Stąd stosujemy wzór na ilość działań podstawowych wyrażony równaniem:

$$S(n) = 1^2 + 2^2 + \dots + n^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

Równanie określa czas działania metody eliminacji niewiadomych na $\vartheta(n^3)$.