Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 9 з дисципліни «Алгоритми та структури даних -1. Основи алгоритмізації»

«Дослідження арифметичних циклічних алгоритмів» Варіант 17

Виконав студент ІП-15 Куркчі Юрій Сергійовий
(шифр, прізвище, ім'я, по батькові)

Перевірив Вечерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2021

Лабораторна робота 9 Дослідження алгоритмів обходу масивів

Мета – дослідити алгоритми обходу масивів, набути практичних навичок використання цих алгоритмів під час складання програмних специфікацій.

Варіант 17

№	Опис варіанту
17	Задано матрицю дійсних чисел A[n,n], ініціалізувати матрицю обходом по рядках. На побічній діагоналі матриці знайти перший додатний і останній від'ємний елементи, та поміняти їх місцями з елементами головній діагоналі.

Постановка задачі

Заданий алгоритм повинен:

1. Створити двовимірний масив дійсних чисел розмірності nxn.
2. Ініціювати його обходом по рядках.
3. Знайти перший додатний і останній від'ємний елементи масиву на побічній діагоналі.
4. Поміняти їх місцями з відповідними елементами головної діагоналі.

Побудова математичної моделі

Таблиця змінних

Змінна	**Тип**	**Ім'я**	**Призначення**
Розмірність масиву	Цілий	n	Початкові дані
Перший рядок з додатній елементом на побічній діагоналі масиву	Цілий	fi	Проміжні дані
Перший стовпець з додатній елементом на побічній діагоналі масиву	Цілий	fj	Проміжні дані
Останій рядок з відємним елементом на побічній діагоналі масиву	Цілий	li	Проміжні дані
Останій стовпець з відємним елементом на побічній діагоналі масиву	Цілий	lj	Проміжні дані
Лічильник	Цілий	i	Проміжні дані
Лічильник	Цілий	j	Проміжні дані

Перше додатне число побічної діагоналі	Дійсний	first_el	Кінцеві дані
Останнє від'ємне число побічної діагоналі	Дійсний	last_el	Кінцеві дані
Додаткове значення для генерування дійсних чисел	Дійсний	k	Початкові дані
Значення для виявлення першого додатного числа побічної діагоналі	Дійсний	first	Проміжні дані
Значення додатніх значень	Дійсний	pos	Проміжні дані
Значення для виявлення останього негативного числа побічної діагоналі	Дійсний	last	Проміжні дані
Значення відємних значень	Дійсний	neg	Проміжні дані
Масив дійсних чисел	Дійсний[i][j]	A[][]	Кінцеві дані
Тимчасовий масив в функціях	Дійсний[i][j]	Matrix[][]	Проміжні дані

Використані функції

- rand()%n – повертає, випадковим чином згенероване, ціле число з діапазону [0, n).
- a%b – повертає остачу від ділення числа a на число b.

Власні функції

- Creat(ціле число) -створює двовимірний масив дійсних чисел заданої розмірності.
- input(двовимірний дійсний масив, ціле число) – повертає двовимірний масив дійсних чисел заданої розмірності ініційований обходом по рядках, кожен елемент якого згенерований випадковим чином.
- output(двовимірний дійсний масив, ціле число) – Виводить двовимірний масив.
- posit(двовимірний дійсний масив, ціле число) – повертає елемент, рядок та стовпець першого додатного елемента побічної діагоналі матриці.

- negat(двовимірний дійсний масив, ціле число) – повертає елемент ,рядок та стовпець останнього від'ємного елемента побічної діагоналі матриці.
- reset(двовимірний дійсний масив, ціле число) – повертає двовимірний масив дійсних чисел заданої розмірності, у якому елемент з заданою позицією переставлений місцями з відповідним елементом головної діагоналі.

Згенеруємо елементи двовимірного масиву за допомогою функції rand:

1. Згенеруємо елементи двовимірного масиву за допомогою функції rand:
2. matrix[i][j] = (pow(-1,rand() % 2)) * (i % 2 ? (i + 1) * n - j - 0.5 : k).
3. Знайдемо перший додатний та останній від'ємний елемент побічної діагоналі, починаючи пошук з правого верхнього кута: matrix[i][n-i-1]
4. Переставимо відшукані елементи з елементами головної діагоналі за допомогою індексів шуканих елементів.

Розв'язання

1. Визначимо основні дії.
2. Деталізуємо дію генерації масиву.
3. Деталізуємо дію пошуку першого додатного елемента побічної діагоналі.
4. Деталізуємо дію пошуку останнього від'ємного елемента побічної діагоналі.
5. Деталізуємо дію додатного елемента та останнього від'ємного елемента побічної діагоналі за допомогою умовної форми вибору.
6. Деталізуємо дію генерації масива обходом по рядках за допомогою ітераційної форми повторення та альтернативної форми вибору.
7. Деталізуємо дію пошуку першого додатного елемента побічної діагоналі за допомогою ітераційної форми повторення та умовної форми вибору.
8. Деталізуємо дію пошуку останнього від'ємного елемента побічної діагоналі за допомогою ітераційної форми повторення та умовної форми вибору.
9. Деталізуємо дію перестановки елемента з відповідним елементом головної діагоналі.

Псевдокод алгоритму

Крок 1

Початок

Введення n

Генерація масиву

Генерація елементів масиву

Пошук першого додатного елемента побічної діагоналі

Пошук останнього від'ємного елемента побічної діагоналі

Перестановка першого додатного та останнього від'ємного елемента побічної діагоналі

Виведення масиву

Видалення масиву

Кінець

Підпрограма **creat**(n)

Генерація масива обходом по рядках

Все підпрограма

Підпрограма input(matrix[][], n)

Генерація елементів масива

Все підпрограма

Підпрограма output(matrix[][], n)

Вивід масива

Все підпрограма

Підпрограма posit(matrix[][], n)

Пошук першого додатного елемента побічної діагоналі

Все підпрограма

Підпрограма negat (matrix[][], n)

Пошук останнього від’ємного елемента побічної діагоналі

Все підпрограма

Підпрограма reset (matrix[][], n)

Перестановка елемента з відповідним елементом головної діагоналі

Все підпрограма

Підпрограма Delete(matrix[][], n)

Видалення масиву

Все підпрограма

Крок 10

Початок

Введення n

A[n][n]=creat(n)

Input(A, n**)**

posit (A, n)

negat(A, n)

reset (A, n)

output(A, n)

Delete(A, n)

Кінець

Підпрограма creat(n)

```
**matrix = new *[n];
```

повторити
для i **від** 0 **до** n

```
    matrix [i] = new [n];
```

Все підпрограма

Підпрограма
input(matrix[][], n)
k=0,5
повторити
для i **від** 0 **до** n
повторити
для j **від** 0 **до** n

```
  matrix[i][j] =(pow(-1,rand()%2))*(i%2?(i+1)*n-j0.5:k)
```

```
k++
```

все повторити
все повторити
Все підпрограма

Підпрограма output(matrix[][], n)

повторити
для i **від** 0 **до** n
повторити
для j **від** 0 **до** n
вивести matrix[i][j]
все повторити
все повторити

Все підпрограма

Підпрограма
positive(matrix[][], n)

```
first = -1
повторити
 для i від 1 до n
        pos = 0;
        повторити
        для i від 1 до n
                якщо j == n-i-1
                        то
                        якщо
                        matrix[n-j-1][n-i-1]>0
                        &&
                        first == -1
                                то
                                first=matrix[n-j-1][n-i-1]
                                first_el=first
                                fi=n-j-1
                                fj=n-i-1
                                все якщо
                        все якщо
                все якщо
        все повторити
все повторити
Все підпрограма
```

Підпрограма negat(matrix[][], n)

```
Last = 0
повторити
 для i від 1 до n
        neg = 0;
        повторити
        для i від 1 до n
                якщо j == n-i-1
                        то
                        якщо
                        matrix[n-j-1][n-i-1]<0
```

```
                                    то
                                    last=matrix[n-j-1][n-i-1]
                                    li=n-j-1
                                    lj=n-i-1
                                    все якщо
                              все якщо
                        все якщо
                  все повторити
все повторити

last _el=last
```

Все підпрограма

```
Підпрограма reset(matrix[][], n)
повторити
 для i від 1 до n
        повторити
        для i від 1 до n
                якщо j == i && i == f
                        то matrix[i][fj] = matrix[i][j]
                  matrix[i][j] = first_el
                якщо j == i && i == li
                        то matrix[i][lj] = matrix[i][j]
                matrix[i][j] = last_el
                все якщо
        все повторити
все повторити
Все підпрограма
```

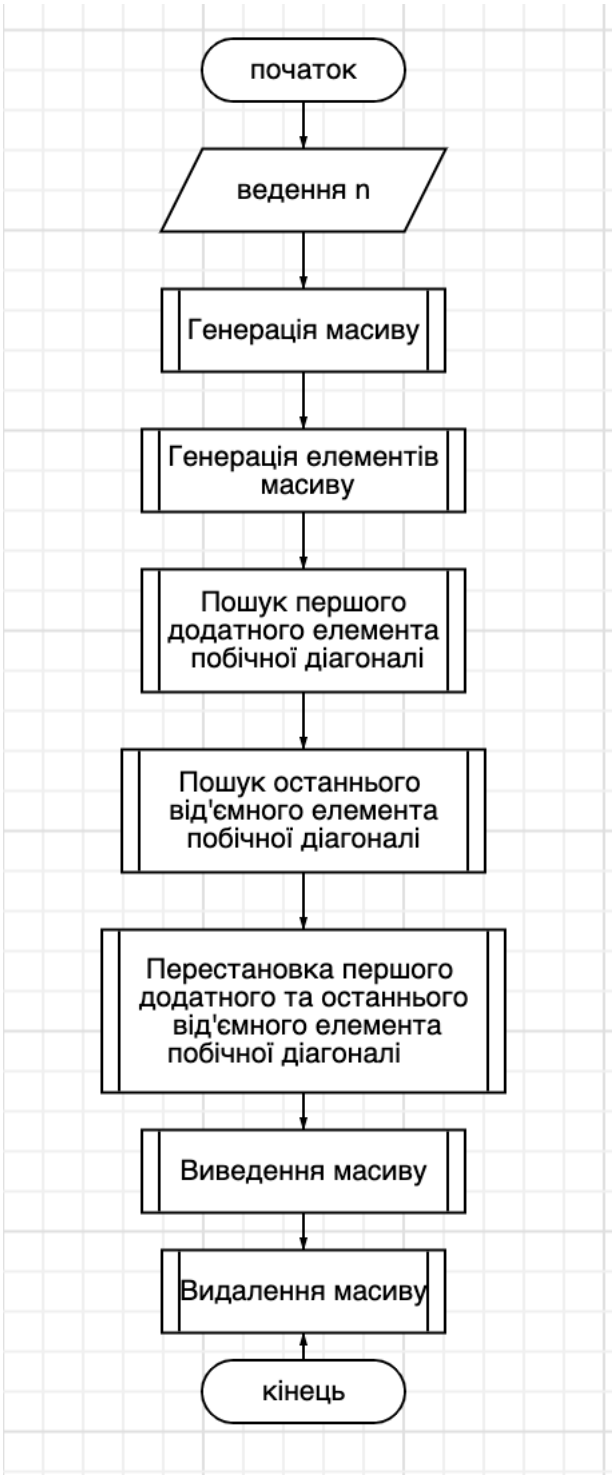
```
Підпрограма Delete(matrix[][], n)

повторити
для i від 0 до n
    delete[] matrix[i]
повторити
delete[] matrix
```

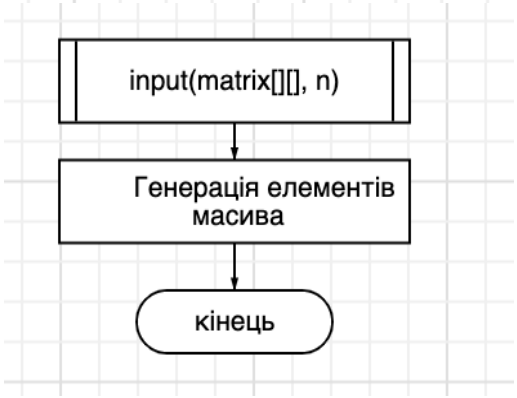
Все підпрограма

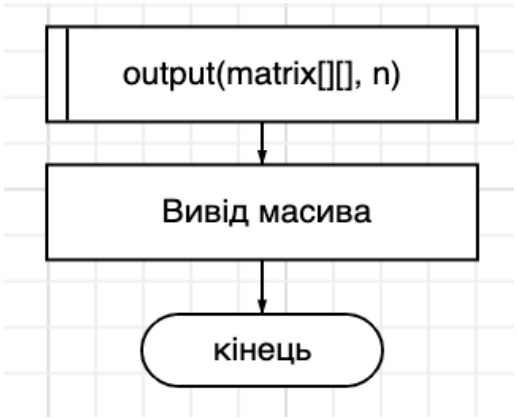
Блок-схема алгоритму

Крок 1

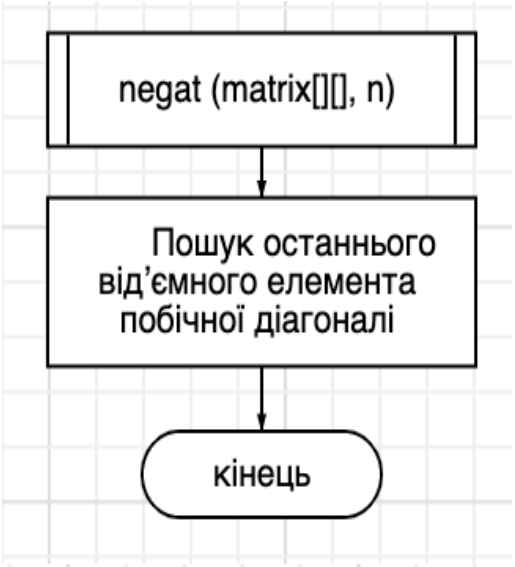

початок
ведення n
Генерація масиву
Генерація елементів масиву
Пошук першого додатного елемента побічної діагоналі
Пошук останнього від'ємного елемента побічної діагоналі
Перестановка першого додатного та останнього від'ємного елемента побічної діагоналі
Виведення масиву
Видалення масиву
кінець


creat(n)
Генерація масива обходом по рядках
кінець


input(matrix[][], n)
Генерація елементів масива
кінець


output(matrix[][], n)
Вивід масива
кінець

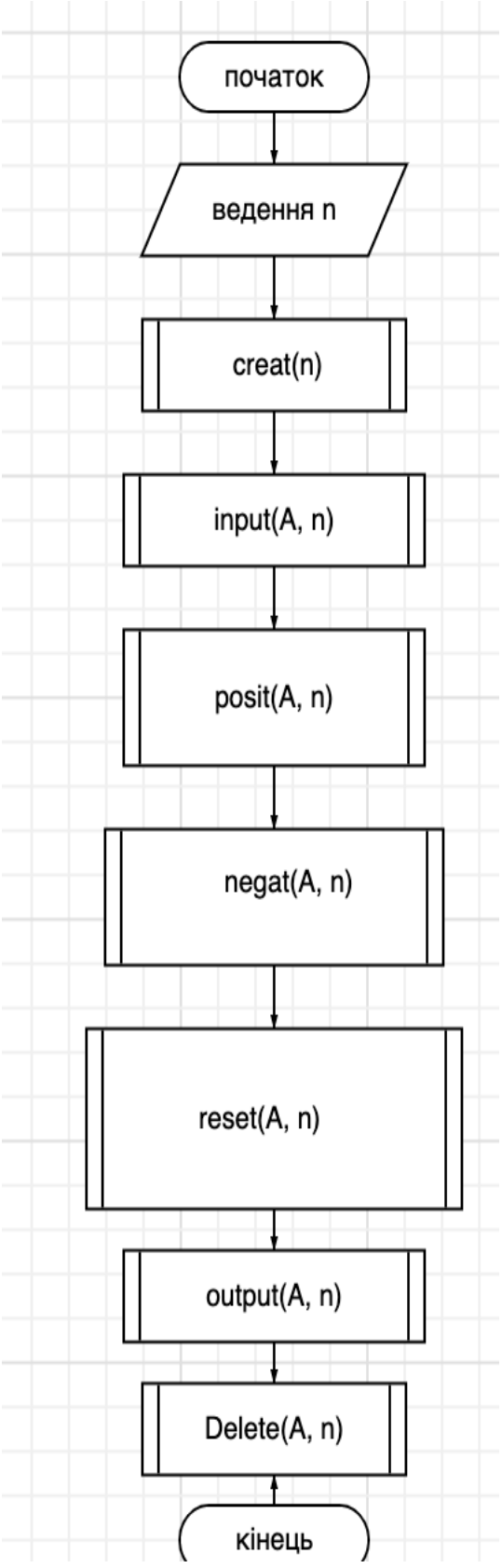

posit(matrix[][], n)
Пошук першого додатного елемента побічної діагоналі
кінець

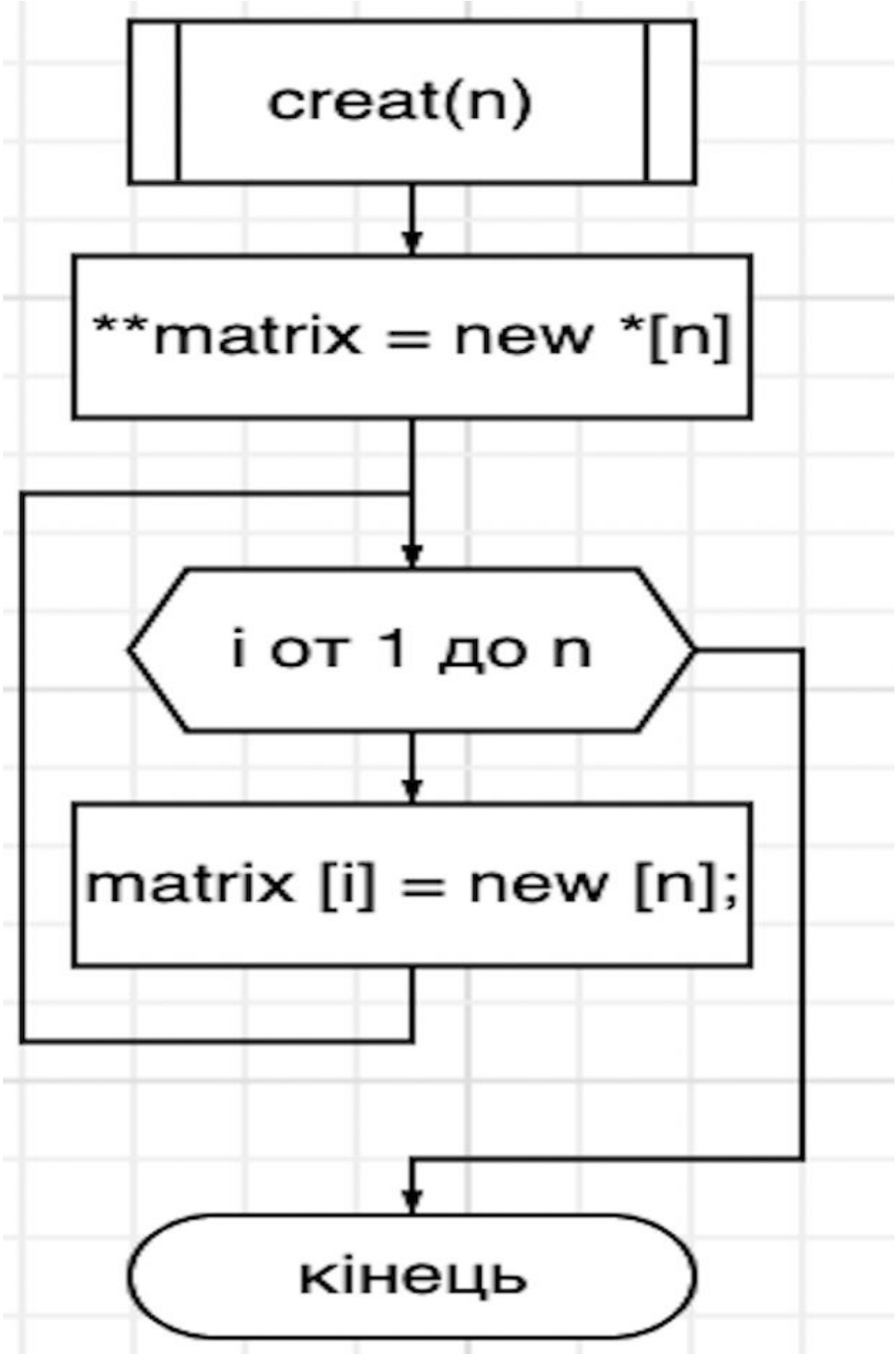

negat (matrix[][], n)
Пошук останнього від'ємного елемента побічної діагоналі
кінець

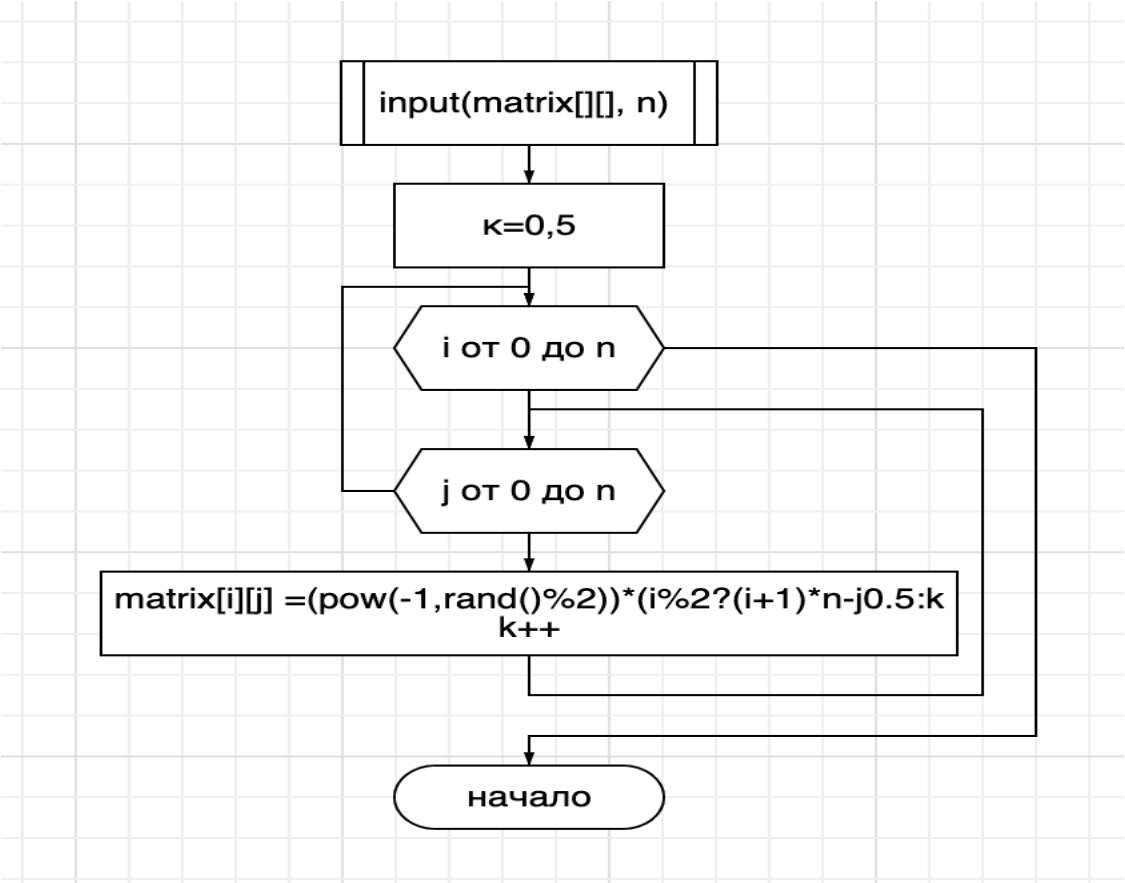

reset (matrix[][], n)
Перестановка елемента з відповідним елементом головної діагоналі
кінець

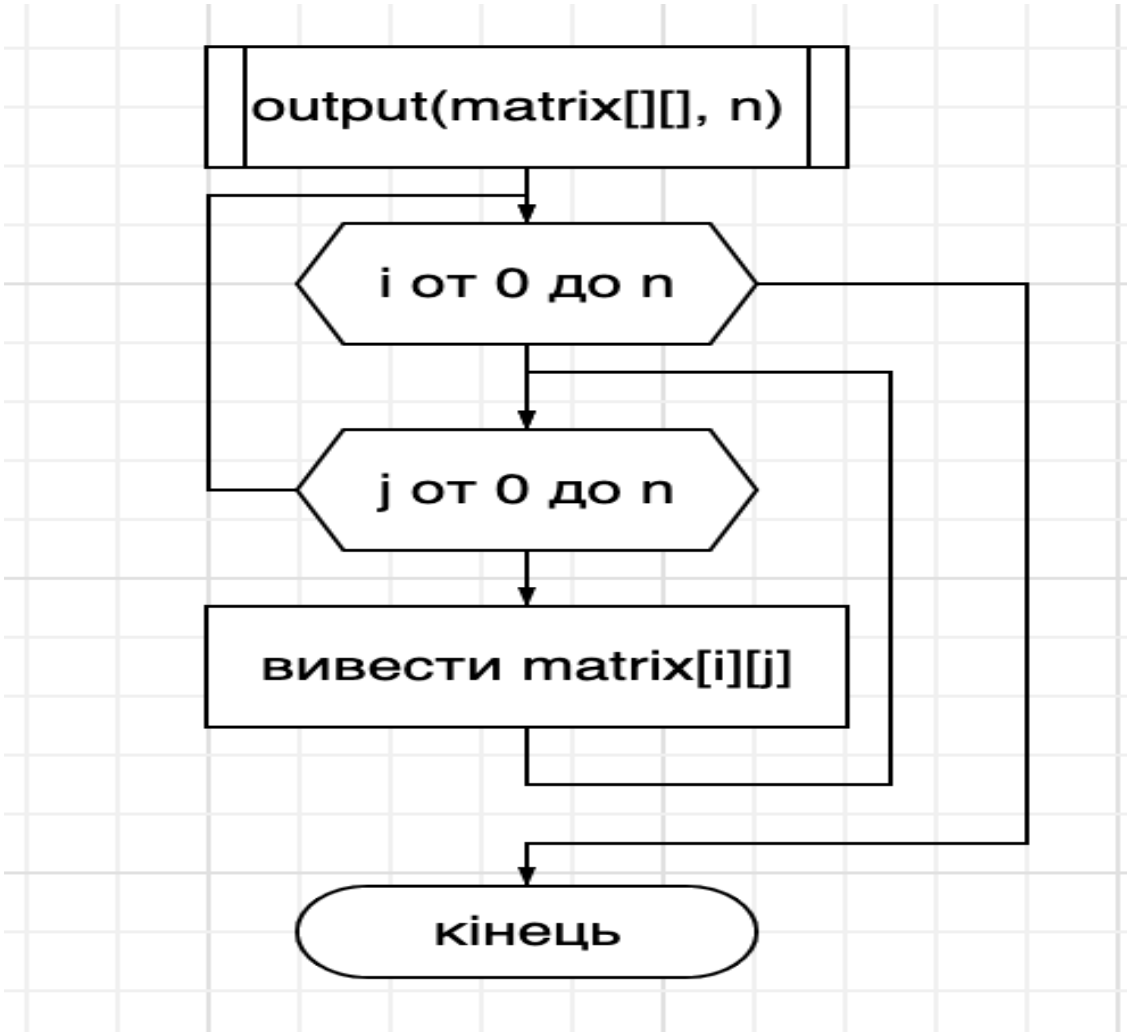

Delete(matrix[][], n)
Видалення масиву
Видалення масиву
кінець

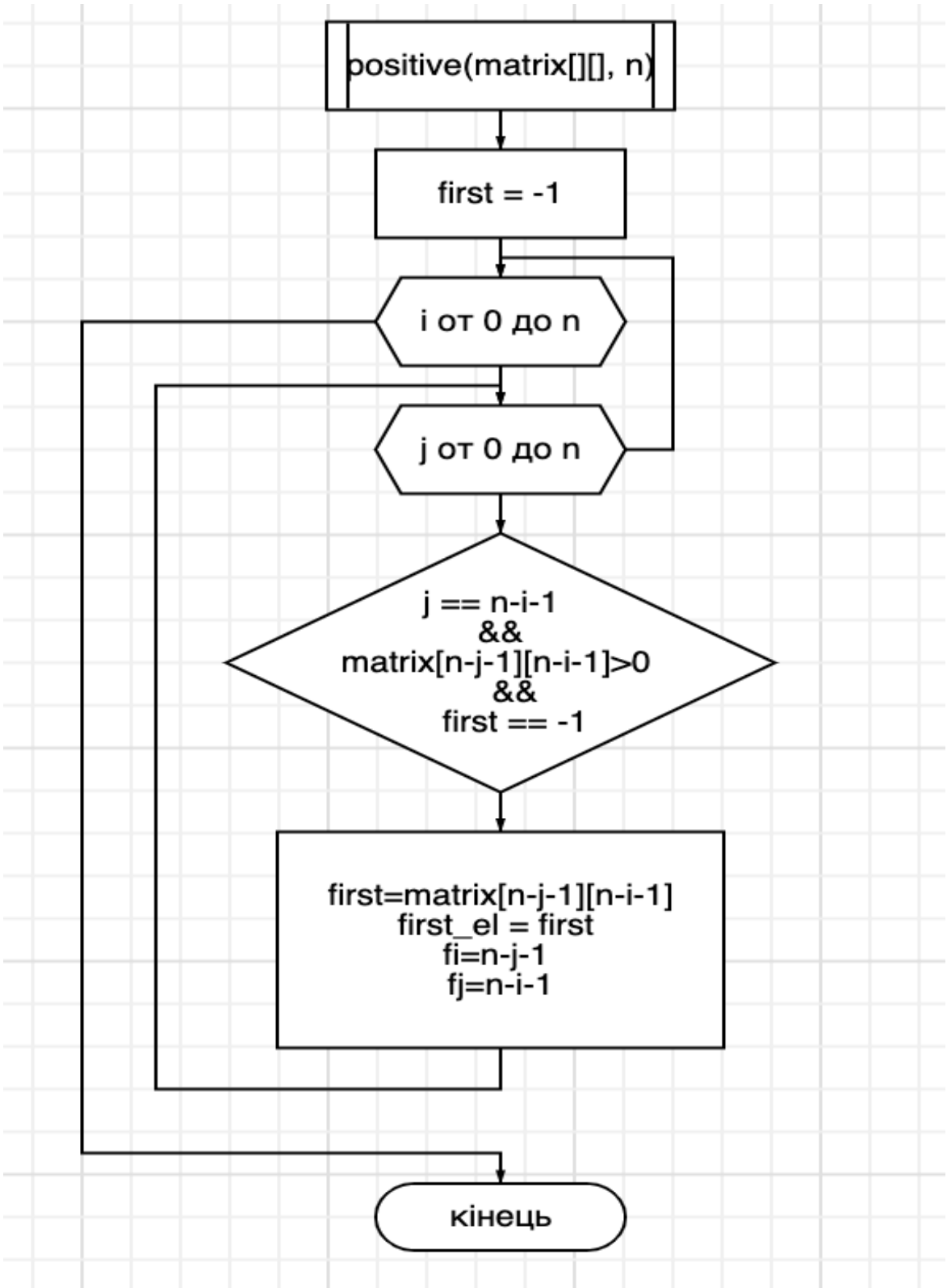
Крок 10

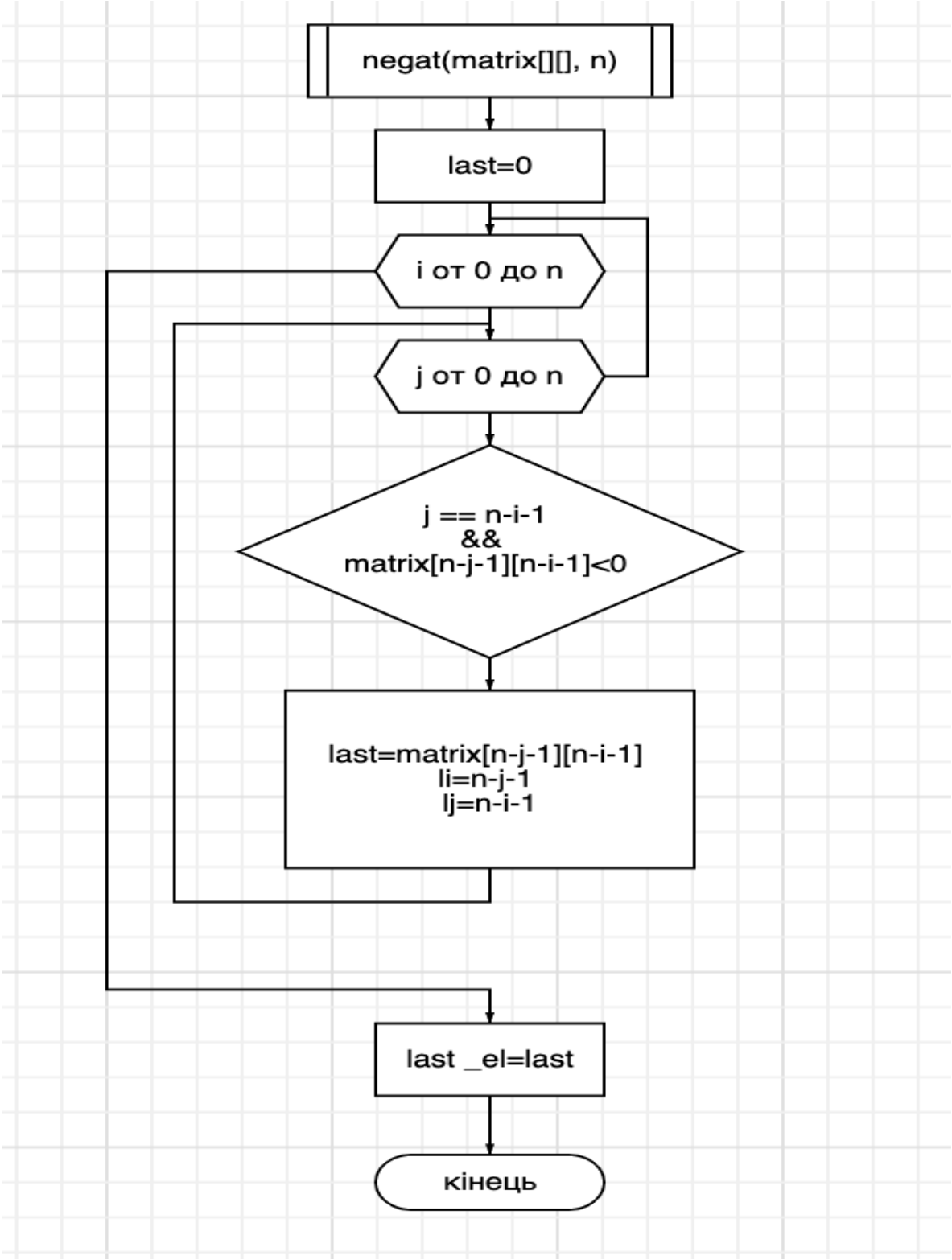

початок
ведення n
creat(n)
input(A, n)
posit(A, n)
negat(A, n)
reset(A, n)
output(A, n)
Delete(A, n)
кінець

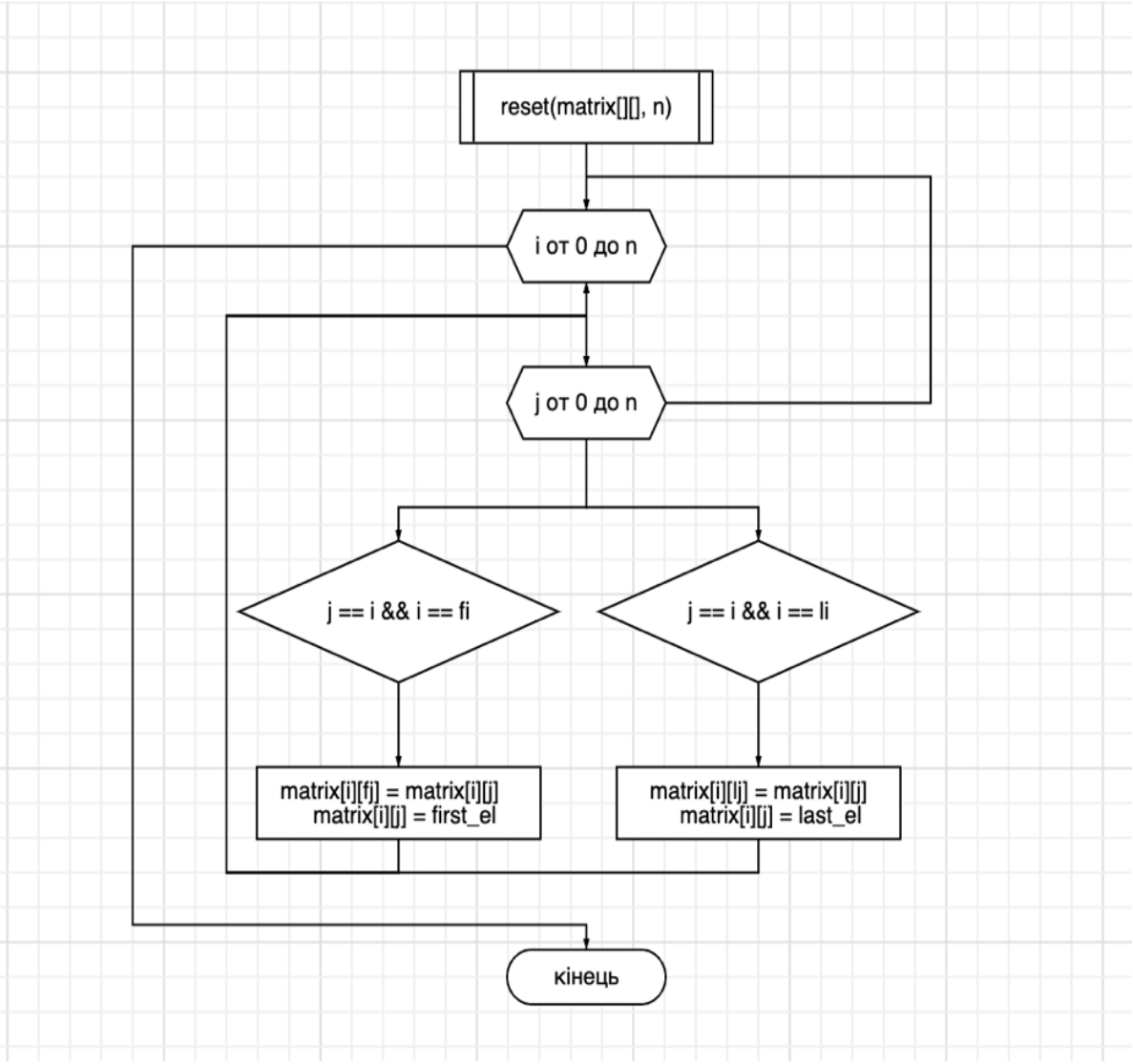

creat(n)
**matrix = new *[n]
i от 1 до n
matrix [i] = new [n];
кінець

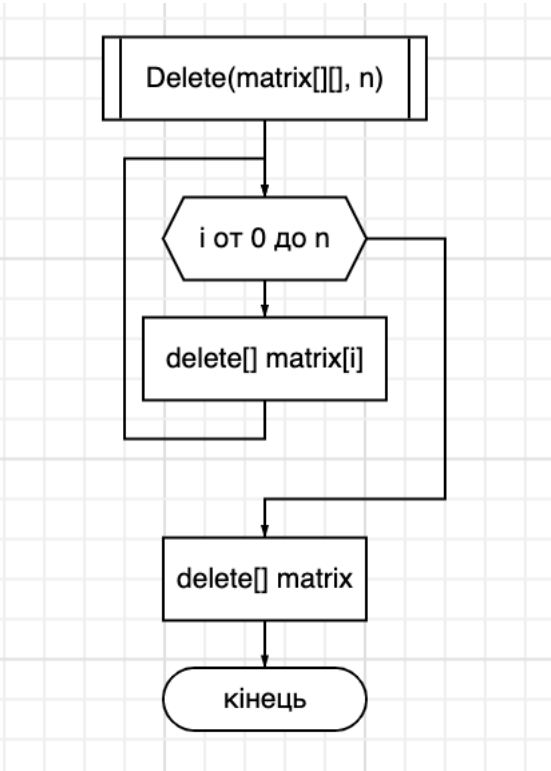

input(matrix[][], n)
к=0,5
i от 0 до n
j от 0 до n
matrix[i][j] =(pow(-1,rand()%2))*(i%2?(i+1)*n-j0.5:k
k++
начало


output(matrix[][], n)
i от 0 до n
j от 0 до n
вивести matrix[i][j]
кінець


positive(matrix[][], n)
first = -1
i от 0 до n
j от 0 до n
j == n-i-1
&&
matrix[n-j-1][n-i-1]>0
&&
first == -1
first=matrix[n-j-1][n-i-1]
first_el = first
fi=n-j-1
fj=n-i-1
кінець


negat(matrix[][], n)
last=0
i от 0 до n
j от 0 до n
j == n-i-1
&&
matrix[n-j-1][n-i-1]<0
last=matrix[n-j-1][n-i-1]
li=n-j-1
lj=n-i-1
last _el=last
кінець


reset(matrix[][], n)
i от 0 до n
j от 0 до n
j == i && i == fi
j == i && i == li
matrix[i][fj] = matrix[i][j]
matrix[i][j] = first_el
matrix[i][lj] = matrix[i][j]
matrix[i][j] = last_el
кінець


Delete(matrix[][], n)
i от 0 до n
delete[] matrix[i]
delete[] matrix
кінець

Код програми

```
You, seconds ago | 1 author (You)
#include <iomanip>
#include <ctime>
#include <iostream>
#include <math.h>


using namespace std;


double **creat(int);
void input(double**, int);
void output(double**, int);
void posit(double**, int);
void negat(double**, int);
void reset(double**, int);
void Delete(double**, int);


double** A;
double first_el;
double last_el;
int n, fi , fj, li, lj;



int main() {

srand(time(NULL));

    cout << "\n\nSet the dimension of the matrix A(n x n)\n" << "n: ";
    cin >> n;
    cout << "\n";


    double **A = creat(n);

```

```
26  int main() {
27
28  srand(time(NULL));
29
30      cout << "\n\nSet the dimension of the matrix A(n x n)\n" << "n: ";
31      cin >> n;
32      cout << "\n";
33
34
35      double **A = creat(n);
36
37
38      input(A, n);
39
40
41      cout << "\nМатриця A: \n";
42
43      output(A, n);
44
45      posit(A, n);
46
47      negat(A, n);
48
49      reset(A, n);
50
51      cout << "Змінена матриця A: ";
52
53      output(A, n);
54
55      Delete(A, n);
56
57      return 0;      You, seconds ago • Uncommitted changes
58  }
59
```

```
62  double **creat(int n){
63
64      double** matrix = new double *[n];
65      for (int i = 0; i < n; ++i) {
66
67       matrix [i] = new double [n];
68      }
69      return matrix;
70  }
71        You, seconds ago • Uncommitted changes
72
73  void input(double** matrix, int n){
74
75      double k = 0.5;
76  for (int i = 0; i < n; i++) {
77
78          for (int j = 0; j < n; j++){
79
80              matrix[i][j] = (pow(-1,rand() % 2)) * (i % 2 ? (i + 1) * n - j - 0.5 : k);
81              k++;
82          }
83      }
84  }
85
86
87  void output(double **matrix, int n){
88
89      for (int i = 0; i < n; i++) {
90
91          cout << "\n";
92          for (int j = 0; j < n; j++) {
93
94              cout << setw(5) << matrix[i][j] << "  ";
95          }
96      }
97      cout << "\n" << endl;
98  }
```

```
void posit(double **matrix, int n){

    double first = -1;
   for (int i = 0; i < n; i++) {

       double pos = 0;
       for (int j = 0; j < n; j++) {

           if(j == n-i-1){

               if(matrix[n - j - 1][n - i - 1] > 0 ){

                   pos = matrix[n - 1 - j][n - i - 1];
                   cout << "Додатнье значення на побічній діагоналі A[" << i + 1 << "][" << j + 1 << "]= " << pos << "
                   if(pos > 0 && first == -1){

                       first = matrix[n - j - 1][n - i - 1];
                       first_el = first;
                       fi = n - j - 1;
                       fj = n - i - 1;

                   }
               }
           }
       }
    }
    cout << "\n Перше додатнье значення на побічній діагоналі: " << first_el << "\n";
           fi = fi;
           fj = fj;
    cout << "координати: [" << fi + 1 << "][" << fj + 1 << "]" << "\n\n";

}
```

```
void negat(double **matrix, int n){

   double last = 0;
  for (int i = 0; i < n; i++) {

      double neg = 0;
      for (int j = 0; j < n; j++) {

          if(j == n-i-1){

              if(matrix[n - j - 1][n - i - 1] < 0 ){

                  neg = matrix[n - 1 - j][n - i - 1];
                  cout << "Негативні значення на побічній діагоналі A[" << i + 1 << "][" << j + 1 << "]= " << neg <<
                  last=neg;
                  li = n - j - 1;
                  lj = n - i - 1;
                  }
              }
          }
      }
      last_el = last;
      cout << "\n Останнє негативне значення на побічній діагоналі: " << last_el << "\n";
      li = li;
      lj = lj;
      cout << "координати: [" << li + 1 << "][" << lj + 1 << "]" << "\n\n\n";
   }
```

```
void reset(double **matrix, int n){

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            if(j == i && i == fi){
            matrix[i][fj] = matrix[i][j];
            matrix[i][j] = first_el;
           cout << "Елемент ["<< i + 1 << "]["<<  fj + 1 << "]: "<< matrix[i][j] << " міняється з "  << " ["<< i + 1 <<
            }

            if(j == i && i == li){
            matrix[i][lj] = matrix[i][j];
            matrix[i][j] = last_el;


            cout << "Елемент ["<< i + 1 << "]["<<  lj + 1 << "]: "<< matrix[i][j] << " міняється з " << " ["<< i + 1 <<
            }
        }
    }
}



void Delete(double **Matrix, int n) {

    for (int i = 0; i < n; ++i) {
        delete[] Matrix[i];
    }
    delete[] Matrix;
}
```

Тестування програми

```
Set the dimension of the matrix A(n x n)
n: 3


Матриця A:

  -0.5     1.5    -2.5
  -5.5    -4.5    -3.5
   6.5     7.5    -8.5

Додатнье значення на побічній діагоналі A[3][1]= 6.5

 Перше додатнье значення на побічній діагоналі: 6.5
координати:  [3][1]

Негативні значення на побічній діагоналі A[1][3]= -2.5
Негативні значення на побічній діагоналі A[2][2]= -4.5

 Останнє негативне значення на побічній діагоналі: -4.5
координати:  [2][2]


Елемент [2][2]: -4.5 міняється з   [2][2]: -4.5


Елемент [3][1]: 6.5 міняється з   [3][3]: -8.5

Змінена матриця A:
  -0.5     1.5    -2.5
  -5.5    -4.5    -3.5
  -8.5     7.5     6.5
```

```
Set the dimension of the matrix A(n x n)
n: 4


Матриця A:

  0.5   -1.5    2.5   -3.5
 -7.5    6.5   -5.5    4.5
  8.5   -9.5   10.5   11.5
-15.5   14.5  -13.5  -12.5


 Перше додатнье значення на побічній діагоналі: 0
координати: [1][1]

Негативні значення на побічній діагоналі A[1][4]= -3.5
Негативні значення на побічній діагоналі A[2][3]= -5.5
Негативні значення на побічній діагоналі A[3][2]= -9.5
Негативні значення на побічній діагоналі A[4][1]= -15.5

 Останнє негативне значення на побічній діагоналі: -15.5
координати: [4][1]


Елемент [1][1]: 0 міняється з  [1][1]: 0

Елемент [4][1]: -15.5 міняється з  [4][4]: -12.5


Змінена матриця A:
    0   -1.5    2.5   -3.5
 -7.5    6.5   -5.5    4.5
  8.5   -9.5   10.5   11.5
-12.5   14.5  -13.5  -15.5
```

```
Set the dimension of the matrix A(n x n)
n: 5


Матриця A:

 -0.5    1.5    2.5    3.5   -4.5
 -9.5    8.5    7.5   -6.5    5.5
 10.5   11.5  -12.5   13.5   14.5
 19.5  -18.5   17.5  -16.5   15.5
 20.5   21.5   22.5   23.5  -24.5

Додатнье значення на побічній діагоналі A[5][1]= 20.5

 Перше додатнье значення на побічній діагоналі: 20.5
координати: [5][1]

Негативні значення на побічній діагоналі A[1][5]= -4.5
Негативні значення на побічній діагоналі A[2][4]= -6.5
Негативні значення на побічній діагоналі A[3][3]= -12.5
Негативні значення на побічній діагоналі A[4][2]= -18.5

 Останнє негативне значення на побічній діагоналі: -18.5
координати: [4][2]


Елемент [4][2]: -18.5 міняється з  [4][4]: -16.5


Елемент [5][1]: 20.5 міняється з  [5][5]: -24.5

Змінена матриця A:
 -0.5    1.5    2.5    3.5   -4.5
 -9.5    8.5    7.5   -6.5    5.5
 10.5   11.5  -12.5   13.5   14.5
 19.5  -16.5   17.5  -18.5   15.5
-24.5   21.5   22.5   23.5   20.5
```

```
Set the dimension of the matrix A(n x n)
n: 6


Матриця A:

  0.5   -1.5   -2.5   -3.5   -4.5    5.5
 11.5   10.5    9.5   -8.5    7.5   -6.5
 12.5  -13.5   14.5  -15.5  -16.5  -17.5
-23.5   22.5  -21.5  -20.5  -19.5   18.5
 24.5  -25.5  -26.5   27.5  -28.5   29.5
-35.5   34.5  -33.5  -32.5  -31.5   30.5

Додатнье значення на побічній діагоналі A[1][6]= 5.5
Додатнье значення на побічній діагоналі A[2][5]= 7.5

 Перше додатнье значення на побічній діагоналі: 5.5
координати: [1][6]

Негативні значення на побічній діагоналі A[3][4]= -15.5
Негативні значення на побічній діагоналі A[4][3]= -21.5
Негативні значення на побічній діагоналі A[5][2]= -25.5
Негативні значення на побічній діагоналі A[6][1]= -35.5

 Останнє негативне значення на побічній діагоналі: -35.5
координати: [6][1]


Елемент [1][6]: 5.5 міняється з  [1][1]: 0.5

Елемент [6][1]: -35.5 міняється з  [6][6]: 30.5


Змінена матриця A:
  5.5   -1.5   -2.5   -3.5   -4.5    0.5
 11.5   10.5    9.5   -8.5    7.5   -6.5
 12.5  -13.5   14.5  -15.5  -16.5  -17.5
-23.5   22.5  -21.5  -20.5  -19.5   18.5
 24.5  -25.5  -26.5   27.5  -28.5   29.5
 30.5   34.5  -33.5  -32.5  -31.5  -35.5
```

```
Set the dimension of the matrix A(n x n)
n: 7


Матриця A:

  0.5   -1.5    2.5    3.5    4.5   -5.5   -6.5
-13.5  -12.5   11.5  -10.5    9.5    8.5   -7.5
-14.5   15.5  -16.5  -17.5   18.5  -19.5  -20.5
 27.5  -26.5  -25.5   24.5   23.5  -22.5  -21.5
-28.5   29.5  -30.5  -31.5  -32.5   33.5  -34.5
 41.5   40.5  -39.5   38.5   37.5  -36.5  -35.5
-42.5  -43.5  -44.5   45.5   46.5   47.5  -48.5

Додатнье значення на побічній діагоналі A[2][6]= 8.5
Додатнье значення на побічній діагоналі A[3][5]= 18.5
Додатнье значення на побічній діагоналі A[4][4]= 24.5
Додатнье значення на побічній діагоналі A[6][2]= 40.5

 Перше додатнье значення на побічній діагоналі: 8.5
координати: [2][6]

Негативні значення на побічній діагоналі A[1][7]= -6.5
Негативні значення на побічній діагоналі A[5][3]= -30.5
Негативні значення на побічній діагоналі A[7][1]= -42.5

 Останнє негативне значення на побічній діагоналі: -42.5
координати: [7][1]


Елемент [2][6]: 8.5 міняється з  [2][2]: -12.5

Елемент [7][1]: -42.5 міняється з  [7][7]: -48.5


Змінена матриця A:
  0.5   -1.5    2.5    3.5    4.5   -5.5   -6.5
-13.5    8.5   11.5  -10.5    9.5  -12.5   -7.5
-14.5   15.5  -16.5  -17.5   18.5  -19.5  -20.5
 27.5  -26.5  -25.5   24.5   23.5  -22.5  -21.5
-28.5   29.5  -30.5  -31.5  -32.5   33.5  -34.5
 41.5   40.5  -39.5   38.5   37.5  -36.5  -35.5
-48.5  -43.5  -44.5   45.5   46.5   47.5  -42.5
```

Висновки

Протягом виконання цієї лабораторної роботи я набув навичок використання алгоритмів обходу масивів під час складання програмних специфікацій. Маючи розмірність та тип масиву, я склав алгоритм, який успішно генерує заданий масив обходом по рядках, знаходить перший додатний та останній від'ємний елементи його побічної діагоналі та переставляє їх місцями з відповідними елементами головної діагоналі.