Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра микро- и наноэлектроники

**Лабораторная работа №7**

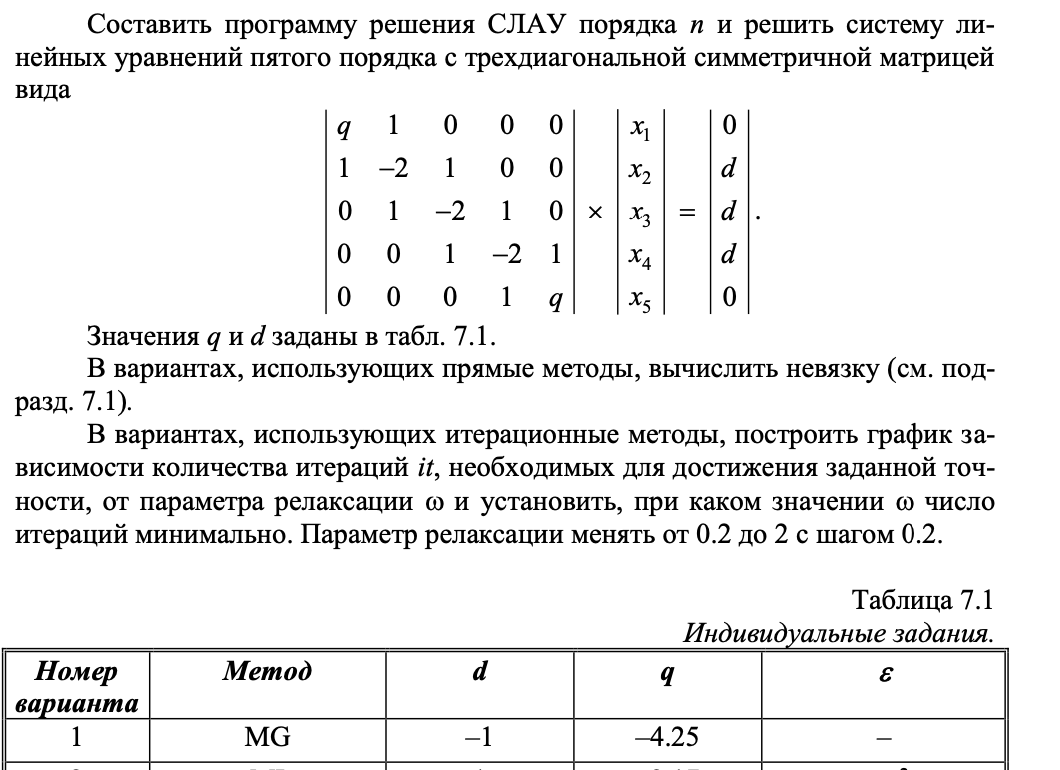
**«Алгоритмы поиска корней уравнений»**

**Вариант 1**

|  |  |
| --- | --- |
| Выполнил: | студент группы 142701  Ахремчик Никита Андреевич |
| Проверил: |  |

Минск 2022

**Задачи:** изучить прямые и численные методы нахождения корней системы линейных алгебраических уравнений (СЛАУ).

**Индивидуальное задание**

Текст программы:

#**include** <iostream>

#**include** <stdlib.h>

**using** **namespace** std;

**const** **float** d = -1;

**const** **float** q = -4.25;

**const** **int** N = 5;

// вывод системы уравнений

**void** **print**(**double** \*\*matrix, **double** \*y, **int** n){

puts("");

printf("Ваши СЛАУ:\n");

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

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

printf("%0.f\*x%i", matrix[i][j], j);

**if** (j<(n-1)) printf(" + ");

}

printf(" = %0.f\n", y[i]);

}

puts("");

}

**double** \*\***basicMatrix**(){

**double** \*\*matrix;

matrix = **new** **double** \*[N];

**for** (**int** i=0; i<N; i++){

matrix[i] = **new** **double** [N];

**for** (**int** j=0; j<N; j++){

**if**((i==0 && j==0) || (i==4 && j==4)) matrix[i][j] = q;

**else** **if**(i==j) matrix[i][j] = -2;

**else** **if**(i==0 && j==1) matrix[i][j] = 1;

**else** **if**((i==1 && j==0) || (i==1 && j==2)) matrix[i][j] = 1;

**else** **if**((i==2 && j==1) || (i==2 && j==3)) matrix[i][j] = 1;

**else** **if**((i==3 && j==2) || (i==3 && j==4)) matrix[i][j] = 1;

**else** **if**(i==4 && j==3) matrix[i][j] = 1;

**else** matrix[i][j] = 0;

}

}

**return** matrix;

}

**void** **printMatrix**(**double** \*\*matrix, **int** n){

printf("Результат:\n");

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

printf("|");

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

**if**((i==0 && j==0) || (i==4 && j==4)) printf("%.2f ", matrix[i][j]);

**else** printf(" %.f ", matrix[i][j]);

}

printf("|");

puts("");

}

}

**void** **deleteMAtrix**(**double** \*\*matrix, **int** n){

**for**(**int** i=0; i<n; i++) **delete**[] matrix[i];

**delete**[] matrix;

}

**double** \***basicY**(){

// инициализируем значения

**double** \*y;

y = **new** **double**[N];

**for** (**int** i=0; i<N; i++){

**if**(i==0 || i==4) y[i] = 0;

**else** y[i] = d;

}

**return** y;

}

**void** **printY**(**double** \*y, **int** n){

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

printf("y[%i]=%.f\n", i, y[i]);

}

puts("");

}

**void** **deleteY**(**double** \*y){

**delete**[] y;

}

**double** \* **methodGauss**(**double** \*\*matrix, **double** \*y, **int** n){

**double** \*x, max;

x = **new** **double**[n];

**int** k, index;

k = 0;

**while** (k<n){

// поиск максимального значения строки a[i][k]

max = abs(matrix[k][k]);

index = k; // индекс максимального элемента строки

**for** (**int** i=(k+1); i<n; i++){

**if** (abs(matrix[i][k])>max){

max = abs(matrix[i][k]);

index = i;

}

}

// Перестановка строк

**if** (max == 0)

{

// нет ненулевых диагональных элементов

cout << "Решение получить невозможно из-за нулевого столбца ";

cout << index << " матрицы A" << endl;

**return** 0;

}

// перестановка значений для реализация алгоритма Гаусса

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

**double** temp = matrix[k][j];

matrix[k][j] = matrix[index][j];

matrix[index][j] = temp;

}

**double** temp = y[k];

y[k] = y[index];

y[index] = temp;

// приведение к треугольному виду

**for** (**int** i=k; i<n; i++){

**double** temp = matrix[i][k];

**if** (abs(temp) == 0) **continue**; // для нулевого коэффициента пропустить

**for** (**int** j=0; j<n; j++) matrix[i][j] = matrix[i][j] / temp;

y[i] = y[i] / temp;

**if** (i == k) **continue**; // уравнение не вычитать само из себя

**for** (**int** j=0; j<n; j++) matrix[i][j] = matrix[i][j] - matrix[k][j];

y[i] = y[i] - y[k];

}

k++;

}

// обратная подстановка

**for** (k=(n-1); k>=0; k--){

x[k] = y[k];

**for** (**int** i=0; i<k; i++) y[i] = y[i] - matrix[i][k] \* x[k];

}

**return** x;

}

**int** **main**(){

setlocale(LC\_ALL, "RUS");

printf("1 - Заполнить матрицу самостоятельно\n2 - Базовая матрица\n0 - Выход\nВаше значение: ");

**int** var;

**bool** flag=true;

cin >> var;

**double** \*\*matrix, \*y, \*x;

**while**(flag){

**switch**(var){

**case** 1:

**int** n;

printf("Введите количество уравнений: ");

cin >> n;

matrix = **new** **double** \*[n];

// инициализируем массив

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

matrix[i] = **new** **double** [n];

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

printf("a[%i][%i]= ", i, j);

cin >> matrix[i][j];

}

}

// инициализируем значения

y = **new** **double**[n];

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

printf("y[%i]= ", i);

cin >> y[i];

}

print(matrix, y, n);

x = methodGauss(matrix, y, n);

printf("Результат:\n");

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

printf("x[%i]=%.f\n", i, x[i]);

deleteMAtrix(matrix, n);

deleteY(y);

puts("");

printf("1 - Заполнить матрицу самостоятельно\n2 - Базовая матрица\n0 - Выход\nВаше значение: ");

cin >> var;

**break**;

**case** 2:

matrix = basicMatrix();

printf("Матрица:\n");

**for** (**int** i=0; i<N; i++){

printf("|");

**for**(**int** j=0; j<N; j++){

**if**((i==0 && j==0) || (i==4 && j==4)) printf("%.2f ", matrix[i][j]);

**else** printf(" %.f ", matrix[i][j]);

}

printf("|");

puts("");

}

y = basicY();

print(matrix, y, N);

x = methodGauss(matrix, y, N);

printf("Результат:\n");

**for** (**int** i=0; i<N; i++)

printf("x[%i]=%.f\n", i, x[i]);

deleteMAtrix(matrix, N);

deleteY(y);

puts("");

printf("1 - Заполнить матрицу самостоятельно\n2 - Базовая матрица\n0 - Выход\nВаше значение: ");

cin >> var;

**break**;

**case** 0:

flag = false;

printf("\nНажмите ENTER........");

**break**;

}

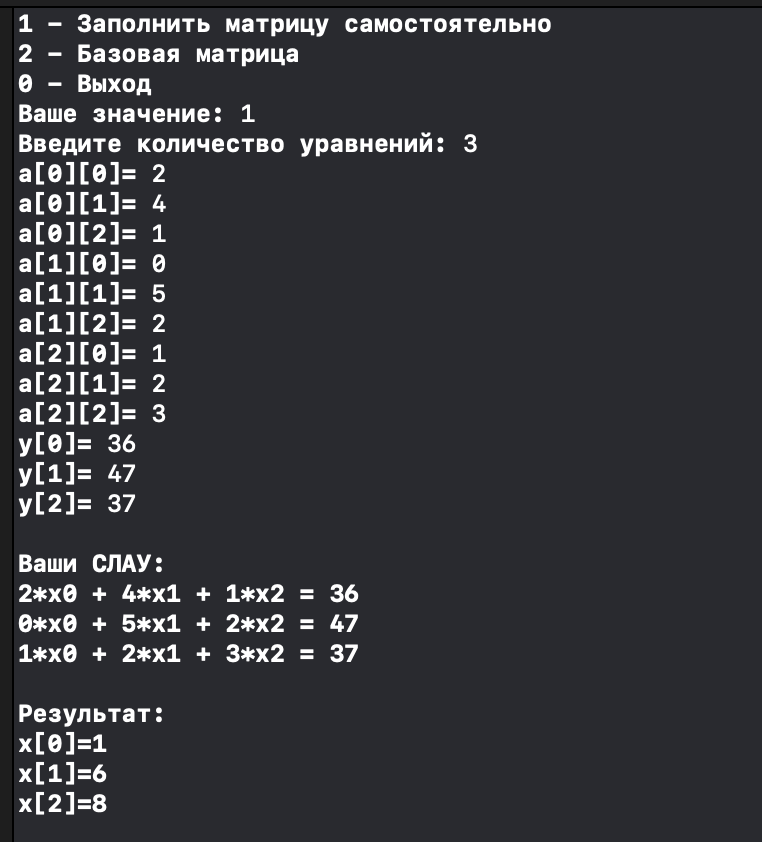
}

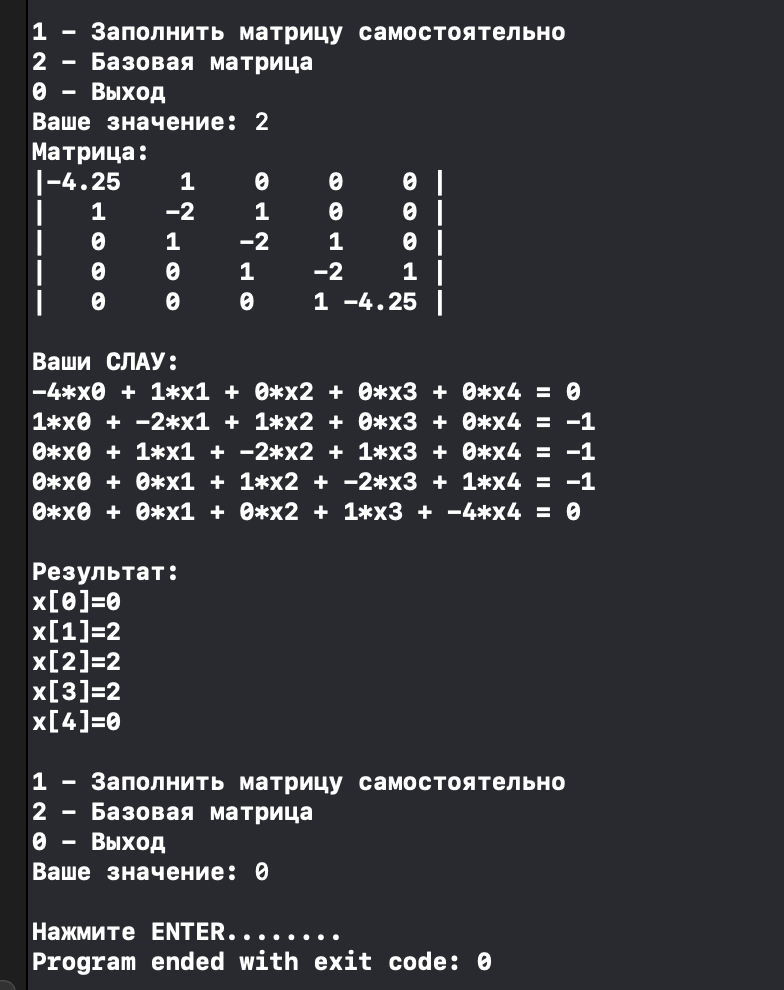
cin.get(); cin.get();

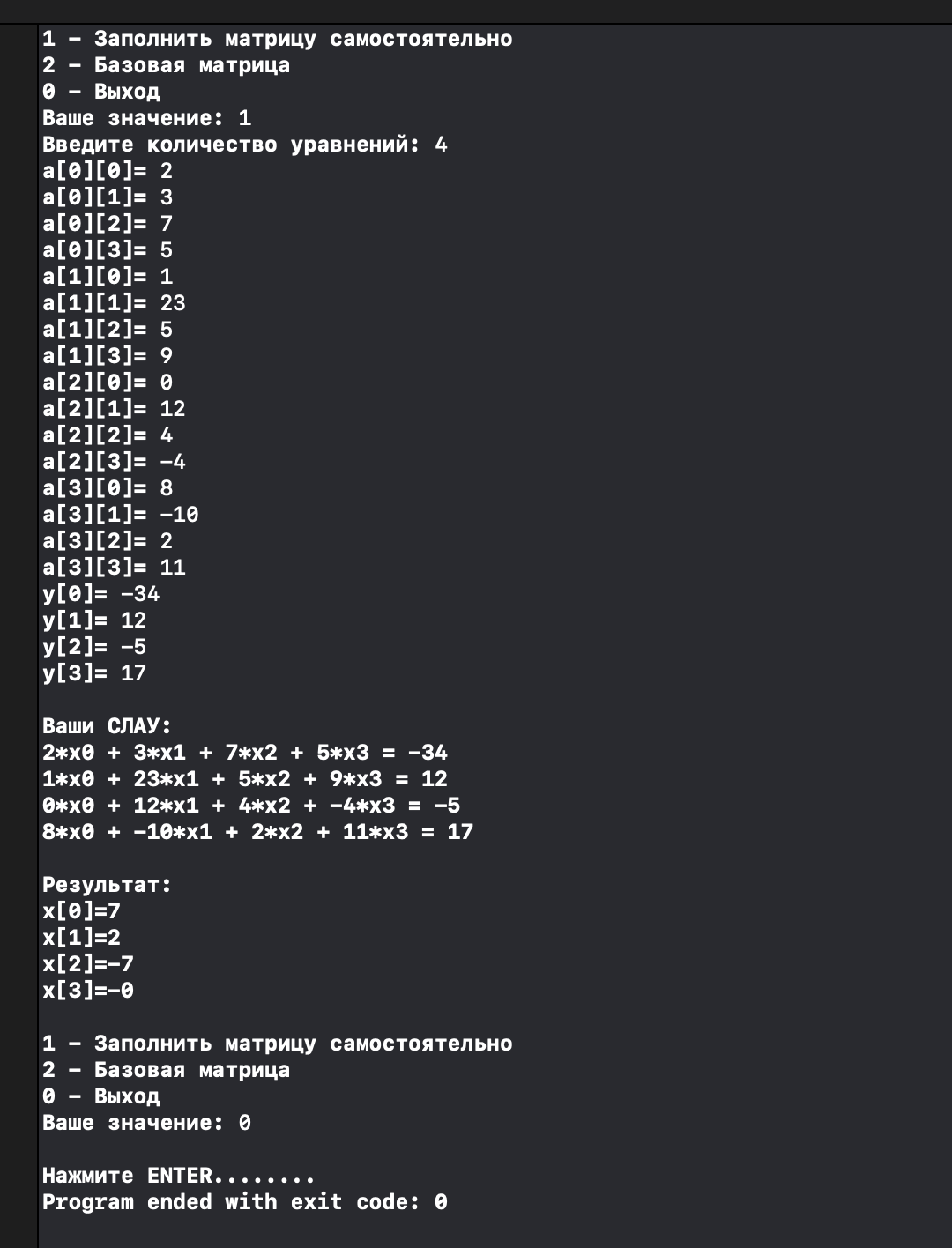
**return** 0;

}

**Результат вывода программы:**







**Выводы:** Проведя данную лабораторную работу, я научился работать с использованием алгоритмы поиска корней уравнений в С++. Также закрепил знания по прошлым темам.