Testy programu przeprowadzone przez **Krzysztof Kozubek** (18-12-2014),
Napisanego programu przez **Małgorzatę Marie Kaczmarczyk.**

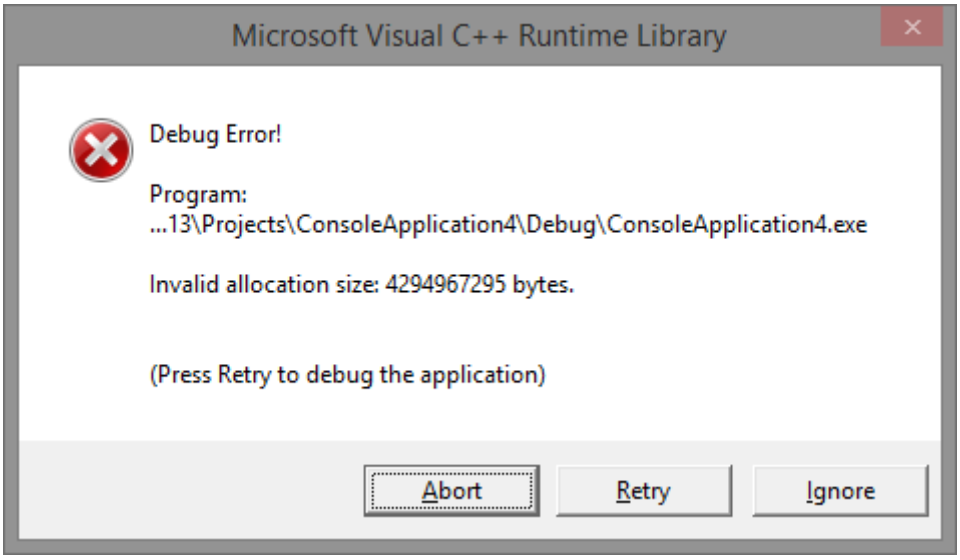
Pierwsze wrażenie programu było, będzie ciężko znaleźć błędy w programie.
Kod jest napisany bardzo przejrzyście i czytelnie, występują komentarze tam gdzie trzeba, nie jest ich też za dużo.
Wybór języka programowania jest odpowiedni do podanego zadania.

Co działa:

-> Wpisywanie wartości wielkości macierzy przyjmuje tylko liczby
-> Wpisywanie wartości macierzy może być podawane w dowolny sposób(potwierdzanie każdej liczby osobno enterem lub wpisanie wszystkich oddzielonych spacją i potwierdzenie spacją)
-> Wartości macierzy które są podawane ponad jej rozmiar nie są brane pod uwagę
-> Program z zasady liczy poprawnie
-> Wypisywane są wszystkie macierze(A, L i U oraz macierz P)

Co nie działa:

-> nie jest możliwe wyliczanie rozkładu dla macierzy o rozmiarze 1x1
-> wypisuje wyniki niepoprawne
-> program zostaje zatrzymany błędem, jeśli długość ciągu znaków przyjmowania wartości jest większy od 50 znaków



-> Podana wartość rozmiaru macierzy może być wartością niecałkowitą(rzeczywistą)
-> Podawane wartości macierzy większe od 1 < |x| (-1 < x < 1) posiadają jedynie precyzje do 4 miejsca po przecinku (przy wypisaniu, **Wprowadzona macierz:**)

Wprowadz macierz:
8.0001 8.00001
8.000001 8.0000001

Wprowadzona macierz:
8.0001 8.00001
8 8

-> Gdy wyznacznik jest równy 0, program próbuje znaleźć rozkład.
-> Zdarza się wypisywać nie poprawne wyniki:

Wprowadzona macierz:
-55 32
9 0

Macierz L:
1 0
-0.163636 1

Macierz U:
-55 32
0 5.23636

Macierz permutacji:
1 0
0 1

Sprawdzenie:
(-0.163636) * (-55) = 8,99998
Przyjmując, że wartości wypisywane są skracane do 6 miejsca po przecinku:
(-0,16363636363636363636363636363636) * (-55) = 8,99999999999999999999999999999998

Wynik zostaje przyjęty jako prawdziwy przez przyjętą precyzję:

```
int main()
{
	const double precision = 0.01;

	cout << "Faktoryzacja LU" << endl;
```

-> Występuje dzielenie przez 0(#IND, #INF)

Co należało by poprawić:

-> Błędy, które zostały wcześniej wspomniane oraz
-> Sprawdzać czy macierz nie jest singularna
-> Program powinien mieć większą precyzję liczenia
-> Należało by dodać ile liczb zostało do dodania
-> **„-”** nie powinien być akceptowany jako wartość macierzy
-> **„-0”** powinno być zamieniane na **„0”**
-> Zmienić język, z polskiego na angielski
-> Dodatek: można wprowadzić graficzny interfejs…

Kod który był testowany:

```
#include <iostream>
#include <string.h>
#include <cmath>
#include <cstdlib>

using namespace std;

void PrintMatrix(double** m, int n);
int CheckInput(char* input);
double FindMax(double** m, int i, int n);
void IdentityMatrix(double**m, int n);
bool Compatibility(double** l, double** u, double** a0, double** p, int n, double
precision);
void Delete(double** m, int n);

int main()
{
	const double precision = 0.01;

	cout << "Faktoryzacja LU" << endl;
	int n;

	char input[50];
	bool ok = false;
	do{
		cout << "Podaj rozmiar macierzy: ";
		cin >> input;
		if (!CheckInput(input) || atoi(input) < 2)
		{
			cout << "Nieprawidlowy rozmiar macierzy." << endl;
		}
		else ok = true;
	} while (!ok);
	n = atoi(input);

	double** a;
	double** l;
	double** u;
	double** p;

	double** a0;

	a = new double*[n];
	l = new double*[n];
	u = new double*[n];
	p = new double*[n];

	a0 = new double*[n];


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

		a0[i] = new double[n];
	}
```

```
	IdentityMatrix(p, n); //inicjalizacja macierzy permutacji jako diagonalnej

	cout << "Wprowadz macierz:" << endl;
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			ok = false;
			do{
				cin >> input;
				if (!CheckInput(input))
				{
					cout << "Nieprawidlowa wartosc A[" << i << "][" << j
<< "]." << endl;
				}
				else ok = true;
			} while (!ok);

			a0[i][j] = atof(input);
			a[i][j] = a0[i][j];
		}



	}

	cout << "Wprowadzona macierz:" << endl;
	PrintMatrix(a, n);

	for (int i = 0; i < n; i++)
	{
		int pivot = FindMax(a, i, n);

		for (int j = 0; j < n; j++)
		{
			double temp = a[pivot][j];
			a[pivot][j] = a[i][j];
			a[i][j] = temp;

			temp = p[pivot][j];
			p[pivot][j] = p[i][j];
			p[i][j] = temp;
		}

		for (int j = i + 1; j < n; j++)
		{
			a[j][i] /= a[i][i];

			for (int k = i + 1; k < n; k++)
			{
				a[j][k] -= a[j][i] * a[i][k];
			}
		}
	}

	// macierz L:
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < i; j++)
		{
			l[i][j] = a[i][j];
```

```
			}
			l[i][i] = 1;
			for (int j = i + 1; j < n; j++)
			{
				l[i][j] = 0;
			}
		}

		//macierz U:
		for (int i = 0; i < n; i++)
		{
			for (int j = 0; j < i; j++)
			{
				u[i][j] = 0;
			}

			for (int j = i; j < n; j++)
			{
				u[i][j] = a[i][j];
			}
		}

		cout << "Macierz L:" << endl;
		PrintMatrix(l, n);
		cout << "Macierz U:" << endl;
		PrintMatrix(u, n);
		cout << "Macierz permutacji:" << endl;
		PrintMatrix(p, n);

		//sprawdzenie poprawnosci rozkladu
		if (Compatibility(l, u, a0, p, n, precision)) cout << "A == L U - wynik zgodny!"
<< endl;
		else cout << "Wynik niepoprawny" << endl;

		Delete(a, n);
		Delete(l, n);
		Delete(u, n);
		Delete(p, n);
		Delete(a0, n);

		return 0;
}

//wypisuje macierz na ekran
void PrintMatrix(double** m, int n)
{
		for (int i = 0; i < n; ++i)
		{
			for (int j = 0; j < n; ++j)
			{
				cout << m[i][j] << " \t";
			}
			cout << endl;
		}
		cout << endl;
}

// sprawdza, czy wszystkie znaki wejscia sa cyframi
int CheckInput(char* input)
{
		int i;
		for (i = 0; i < strlen(input); i++)
```

```
	{
		if ((input[i] < '0' || input[i] > '9') && input[i] != '.')
		{
			if (i > 0 || input[i] != '-') return 0;
		}
	}
	return 1;
}

// znajduje maksymalną wartość w i-tej kolumnie macierzy m
double FindMax(double** m, int i, int n)
{
	int jMax = i;
	for (int j = i; j < n; j++)
	{
		if (abs(m[i][j]) > abs(m[i][jMax])) jMax = j;
	}
	return jMax;
}

// inicjalizuje macierz jako identycznościową
void IdentityMatrix(double**m, int n)
{
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			m[i][j] = (i == j) ? 1 : 0;
		}
	}
}

//sprawdza, czy iloczyn dwóch macierzy równa się trzeciej
bool Compatibility(double** l, double** u, double** a0, double** p, int n, double
precision)
{
	double** aP;
	double** aLU;

	aP = new double*[n];
	aLU = new double*[n];

	for (int i = 0; i < n; ++i)
	{
		aP[i] = new double[n];
		aLU[i] = new double[n];
	}

	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			for (int k = 0; k < n; k++)
			{
				aLU[i][j] += l[i][k] * u[k][i]; //aLU = l * u
				aP[i][j] += p[i][k] * a0[k][i]; //aP = p * a
			}
		}
	}
	int compatible = 0;
	for (int i = 0; i < n; i++)
	{
```

```
                for (int j = 0; j < n; j++)
                {
                        if (abs(aP[i][j] - aLU[i][j]) <= precision) compatible++;
                }
        }

        Delete(aP, n);
        Delete(aLU, n);
        if (compatible == n * n) return true;
        else return false;
}

// usuwa z pamięci tablicę dwuwymiarową
void Delete(double** m, int n)
{
        for (int i = 0; i < n; ++i)
        {
                delete[] m[i];
        }
        delete m;
}
```