Program do zadania Nr.1:

```cpp
#include <iostream>
#include <chrono>

using namespace std;

/**
 * n - liczba równań (wiersze macierzy)
 * b - przekątna nad główną (ponumerowana: [0; n-2])
 * c - główna przekątna macierzy (ponumerowana: [0; n-1])
 * a - przekątna poniżej głównej (ponumerowana: [1; n-1])
 * f - prawa strona (kolumna(wyniki))
 * x - rozwiązanie, tablica X będzie zawierać odpowiedź
 */

void solveMatrix (int n, const double *a, double *c, const double *b, double *f, double *x){
    /* Forward substitution */
    double m;
    /* Modify coefficients */
    for (int i = 1; i < n; i++){
        m = a[i]/c[i-1];
        c[i] = c[i] - m*b[i-1];
        f[i] = f[i] - m*f[i-1];
    }
    /* back substitute */
    x[n-1] = f[n-1]/c[n-1];
    for (int i = n - 2; i >= 0; i--)
        x[i]=(f[i]-b[i]*x[i+1])/c[i];
}

int main(){
    int n = 7;
    double lower[] = {1, 1, 1, 1, 1 ,1 ,1};
    double diagonal[] = {4, 4, 4, 4, 4, 4, 4};
    double upper[] = {1, 1, 1, 1, 1, 1, 1};
    double rhs[] = {1, 2, 3, 4, 5, 6, 7};
    double X[n];
    cout.precision( prec: 16);
    auto start = chrono::high_resolution_clock::now();
    solveMatrix(n,upper,diagonal,lower,rhs,X);
    auto finish = chrono::high_resolution_clock::now();
    int counter = 1;
    for (double i : X){
        cout << "x" << counter << " --> " << i << endl;
        counter++;
    }
    cout << "\n" << chrono::duration_cast<chrono::nanoseconds>(finish-start).count() << "ns\n";
}
```

Wyniki do zadania Nr.1:

```
C:\Users\homen\CLionProjects\C++\cmake-build-debug\C__.exe
x1 --> 0.1667893961708395
x2 --> 0.3328424153166421
x3 --> 0.501840942562592
x4 --> 0.6597938144329897
x5 --> 0.858983799705449
x6 --> 0.9042709867452137
x7 --> 1.523932253313697

300ns

Process finished with exit code 0
```

Komentarze do zadania Nr.1:

W tym zadaniu było użyto Tridiagonal matrix algorithm lub zwanym jeszcze jak algorytm Thomasa, który służy do rozwiązywania układów równań liniowych w postaci A*x = B, gdzie A jest macierzą trójdiagonalną.

Rozwiązanie jest wykonywane w dwóch etapach, podobnie jak w metodzie Gaussa: podstawianie do przodu i podstawianie wstecz.

Podstawianie do przodu składa się z obliczenia nowych współczynników w następujący sposób, oznaczających nowe współczynniki liczbami pierwszymi:

$$c'_i = \begin{cases} \dfrac{c_i}{b_i} & ; \quad i = 1 \\ \dfrac{c_i}{b_i - a_i c'_{i-1}} & ; \quad i = 2, 3, \dots, n-1 \end{cases}$$

oraz

$$d'_i = \begin{cases} \dfrac{d_i}{b_i} & ; \quad i = 1 \\ \dfrac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}} & ; \quad i = 2, 3, \dots, n. \end{cases}$$

Rozwiązanie jest następnie otrzymywane przez podstawianie wstecz:

$$x_n = d'_n$$

$$x_i = d'_i - c'_i x_{i+1} \qquad ; \ i = n-1, n-2, \dots, 1.$$

Powyższa metoda nie modyfikuje oryginalnych wektorów współczynników, ale musi również śledzić za nowymi współczynnikami. Jeżeli wektory współczynników można modyfikować, to algorytm o większej wydajności (który został użyty w programie (zadanie Nr.1)) to:

For $i = 2, 3, \dots, n,$ do

$$w = \frac{a_i}{b_{i-1}}$$

$$b_i := b_i - w c_{i-1}$$

$$d_i := d_i - w d_{i-1}$$

Po czym następuje podstawianie wstecz

$$x_n = \frac{d_n}{b_n}$$

$$x_i = \frac{d_i - c_i x_{i+1}}{b_i}, \quad \text{for } i = n-1, n-2, \dots, 1$$