Matrix Lib C++

1.0

Wygenerowano za pomocą Doxygen 1.15.0

Rozdział 1

Struktura katalogów

1.1 Katalogi

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików wraz z ich krótkimi opisami:

Rozdział 3

Dokumentacja katalogów

3.1 Dokumentacja katalogu src

Pliki

- plik main.cpp
- plik matrix_core.cpp
- plik matrix_operators.cpp
- plik matrix_utils.cpp

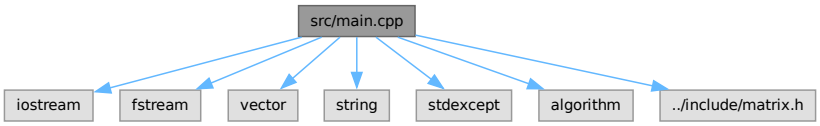
Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku src/main.cpp

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <stdexcept>
#include <algorithm>
#include "../include/matrix.h"
```

Wykres zależności załączania dla main.cpp:



Funkcje

- matrix wczytaj_macierz_z_pliku (const std::string &filename)
- void wypisz_fragment (const matrix &m, size_t max_rows=5, size_t max_cols=5)
- int main ()

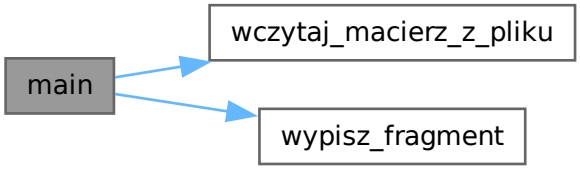
4.1.1 Dokumentacja funkcji

4.1.1.1 main()

```
int main ()
```

Definicja w linii 41 pliku main.cpp.

Oto graf wywołań dla tej funkcji:



4.1.1.2 wczytaj_macierz_z_pliku()

```
matrix wczytaj_macierz_z_pliku (
            const std::string & filename)
```

Definicja w linii 10 pliku main.cpp.

Oto graf wywoływań tej funkcji:

4.1.1.3 wypisz_fragment()

```
void wypisz_fragment (
            const matrix & m,
            size_t max_rows = 5,
            size_t max_cols = 5)
```

Definicja w linii 27 pliku main.cpp.

Oto graf wywoływań tej funkcji:

4.2 main.cpp

Idź do dokumentacji tego pliku.

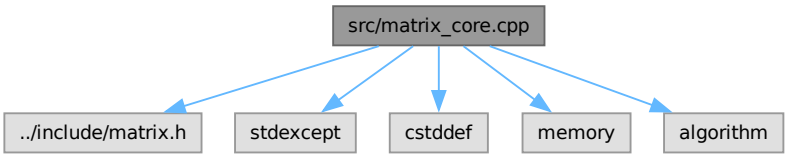
```
00001 #include <iostream>
00002 #include <fstream>
00003 #include <vector>
00004 #include <string>
00005 #include <stdexcept>
00006 #include <algorithm> // min
00007 #include "../include/matrix.h"
00008
00009 // Wczytywanie macierzy z pliku tekstowego
00010 matrix wczytaj_macierz_z_pliku(const std::string& filename) {
00011     std::ifstream fin(filename);
00012     if (!fin) throw std::runtime_error("Nie można otworzyć pliku: " + filename);
00013
00014     size_t rows, cols;
00015     fin » rows » cols;
00016
00017     matrix result(rows, cols, 0.0);
00018     for (size_t r = 0; r < rows; ++r) {
00019         for (size_t c = 0; c < cols; ++c) {
00020             fin » result.data[r][c];
00021         }
00022     }
00023     return result;
00024 }
00025
00026 // Wypisanie fragmentu macierzy (np. 5x5)
00027 void wypisz_fragment(const matrix& m, size_t max_rows = 5, size_t max_cols = 5) {
00028     size_t rmax = std::min(static_cast<size_t>(m.rows), max_rows);
00029     size_t cmax = std::min(static_cast<size_t>(m.cols), max_cols);
00030     for (size_t r = 0; r < rmax; ++r) {
00031         std::cout « "[ ";
00032         for (size_t c = 0; c < cmax; ++c) {
00033             std::cout « m(r, c) « " ";
00034         }
00035         if (cmax < static_cast<size_t>(m.cols)) std::cout « "...";
00036         std::cout « "]\n";
00037     }
00038     if (rmax < static_cast<size_t>(m.rows)) std::cout « "...\n";
00039 }
00040
00041 int main() {
00042     try {
00043         std::cout « "Wczytywanie macierzy A z pliku...\n";
00044         matrix A = wczytaj_macierz_z_pliku("data/input_matrix_A.txt");
00045         std::cout « "Wczytywanie macierzy B z pliku...\n";
00046         matrix B = wczytaj_macierz_z_pliku("data/input_matrix_B.txt");
00047
00048         std::cout « "\n--- Testowanie A+B ---\n";
00049         matrix AB = A + B;
00050         wypisz_fragment(AB);
00051
00052         std::cout « "\n--- Testowanie A*B ---\n";
00053         matrix AMB = A * B;
00054         wypisz_fragment(AMB);
00055
00056         std::cout « "\n--- Testowanie A+10 ---\n";
00057         matrix Aplus10 = A + 10;
00058         wypisz_fragment(Aplus10);
00059
00060         std::cout « "\n--- Testowanie A*5 ---\n";
00061         matrix Amul5 = A * 5;
00062         wypisz_fragment(Amul5);
00063
00064         std::cout « "\n--- Testowanie 10+A ---\n";
00065         matrix tenPlusA = 10 + A;
00066         wypisz_fragment(tenPlusA);
00067
00068         std::cout « "\n--- Testowanie operator() ---\n";
00069         std::cout « "A(0,0) przed: " « A(0,0) « "\n";
00070         A(0,0) = 123.456;
00071         std::cout « "A(0,0) po: " « A(0,0) « "\n";
00072
00073         std::cout « "\n--- Testowanie operator+= (skalarem) ---\n";
00074         matrix Ap = A;
00075         Ap += 3; // dodaj 3 do wszystkich elementów
00076         wypisz_fragment(Ap);
00077
00078         std::cout « "\n--- Testowanie operator-= (skalarem) ---\n";
00079         matrix Am = A;
00080         Am -= 2; // odejmij 2 od wszystkich elementów
00081         wypisz_fragment(Am);
00082
```

```
00083         std::cout « "\n--- Testowanie operator*= (skalarem) ---\n";
00084         matrix As = A;
00085         As *= 2;
00086         wypisz_fragment(As);
00087 
00088         std::cout « "\n--- Testowanie inkrementacji ++ (postfix) ---\n";
00089         matrix Aincr = A;
00090         Aincr++;
00091         wypisz_fragment(Aincr);
00092 
00093         std::cout « "\n--- Testowanie dekrementacji -- (postfix) ---\n";
00094         matrix Adecr = A;
00095         Adecr--;
00096         wypisz_fragment(Adecr);
00097 
00098         std::cout « "\n--- Testowanie operatora porównania == ---\n";
00099         std::cout « ((A == B) ? "PRAWDA" : "FAŁSZ") « "\n";
00100 
00101         std::cout « "\n--- Testowanie operatora porównania != ---\n";
00102         std::cout « ((!(A == B)) ? "PRAWDA" : "FAŁSZ") « "\n";
00103 
00104         std::cout « "\n--- Testowanie operatora > ---\n";
00105         std::cout « ((A > B) ? "PRAWDA" : "FAŁSZ") « "\n";
00106 
00107         std::cout « "\n--- Testowanie operatora < ---\n";
00108         std::cout « ((A < B) ? "PRAWDA" : "FAŁSZ") « "\n";
00109 
00110         std::cout « "\n--- Test zabezpieczeń (try-catch) ---\n";
00111         try {
00112             matrix C(2, 2, 1.0);
00113             matrix wynik = A * C; // jeśli wymiary nie pasują, powinien być wyjątek
00114             wypisz_fragment(wynik);
00115         } catch (const std::exception& e) {
00116             std::cout « "Złapano wyjątek: " « e.what() « "\n";
00117         }
00118 
00119     } catch (const std::exception& e) {
00120         std::cerr « "Błąd krytyczny: " « e.what() « std::endl;
00121         return 1;
00122     }
00123 
00124     std::cout « "\n--- KONIEC TESTÓW ---\n";
00125     return 0;
00126 }
```

4.3 Dokumentacja pliku src/matrix_core.cpp

```
#include "../include/matrix.h"
#include <stdexcept>
#include <cstddef>
#include <memory>
#include <algorithm>
```

Wykres zależności załączania dla matrix_core.cpp:



4.4 matrix_core.cpp

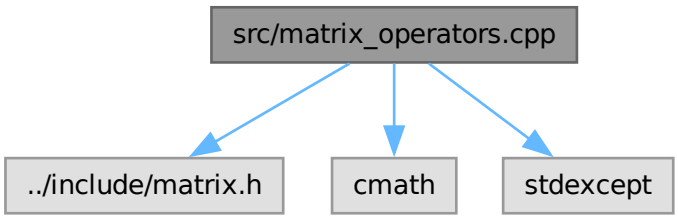
Idź do dokumentacji tego pliku.

```
00001 #include "../include/matrix.h"
00002 #include <stdexcept>
00003 #include <cstddef>
00004 #include <memory>
00005 #include <algorithm>
00006
00007 matrix::matrix() noexcept : rows(0), cols(0), data(nullptr) {}
00008
00009 matrix::matrix(std::size_t r, std::size_t c, double value)
00010     : rows(static_cast<int>(r)), cols(static_cast<int>(c)) {
00011     alokuj(r * c);
00012     for (std::size_t i = 0; i < static_cast<std::size_t>(rows); ++i) {
00013         for (std::size_t j = 0; j < static_cast<std::size_t>(cols); ++j) {
00014             data[i][j] = value;
00015         }
00016     }
00017 }
00018
00019 matrix::matrix(std::initializer_list<std::initializer_list<double» init)
00020     : rows(static_cast<int>(init.size())),
00021       cols(init.size() ? static_cast<int>(init.begin()->size()) : 0) {
00022     alokuj(rows * cols);
00023     std::size_t r = 0;
00024     for (const auto& row : init) {
00025         if (row.size() != static_cast<std::size_t>(cols))
00026             throw std::runtime_error("Niezgodne długości wierszy w initializer_list");
00027         std::size_t c = 0;
00028         for (double val : row) {
00029             data[r][c] = val;
00030             ++c;
00031         }
00032         ++r;
00033     }
00034 }
00035
00036 void matrix::alokuj(std::size_t /*n*/) {
00037     data = new double*[rows];
00038     for (std::size_t i = 0; i < static_cast<std::size_t>(rows); ++i) {
00039         data[i] = new double[cols];
00040     }
00041     for (std::size_t i = 0; i < static_cast<std::size_t>(rows); ++i) {
00042         for (std::size_t j = 0; j < static_cast<std::size_t>(cols); ++j) {
00043             data[i][j] = 0.0;
00044         }
00045     }
00046 }
```

4.5 Dokumentacja pliku src/matrix_operators.cpp

```
#include "../include/matrix.h"
#include <cmath>
#include <stdexcept>
```

Wykres zależności załączania dla matrix_operators.cpp:



Funkcje

- matrix operator+ (int a, matrix &m)
- matrix operator∗ (int a, matrix &m)
- matrix operator- (int a, matrix &m)
- ostream & operator<< (ostream &o, matrix &m)

4.5.1 Dokumentacja funkcji

4.5.1.1 operator∗()

```
matrix operator* (
            int a,
            matrix & m)
```

Definicja w linii 79 pliku matrix_operators.cpp.

4.5.1.2 operator+()

```
matrix operator+ (
            int a,
            matrix & m)
```

Definicja w linii 68 pliku matrix_operators.cpp.

4.5.1.3 operator-()

```
matrix operator- (
            int a,
            matrix & m)
```

Definicja w linii 90 pliku matrix_operators.cpp.

4.5.1.4 operator<<()

```
ostream & operator<< (
            ostream & o,
            matrix & m)
```

Definicja w linii 162 pliku matrix_operators.cpp.

4.6 matrix_operators.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "../include/matrix.h"
00002 #include <cmath>
00003 #include <stdexcept>
00004
00005 // Dodawanie macierzy A+B
00006 matrix& matrix::operator+(matrix& m) {
00007     if (rows != m.rows || cols != m.cols)
00008         throw std::runtime_error("Nieprawidłowe wymiary dla dodawania");
00009     matrix* result = new matrix(rows, cols);
00010     for(int i = 0; i < rows; i++) {
00011         for(int j = 0; j < cols; j++) {
00012             result->data[i][j] = this->data[i][j] + m.data[i][j];
00013         }
00014     }
00015     return *result;
00016 }
00017
00018 // Mnożenie macierzy A*B
00019 matrix& matrix::operator*(matrix& m) {
00020     if (cols != m.rows)
00021         throw std::runtime_error("Nieprawidłowe wymiary dla mnożenia");
00022     matrix* result = new matrix(rows, m.cols);
00023     for(int i = 0; i < rows; i++) {
00024         for(int j = 0; j < m.cols; j++) {
00025             result->data[i][j] = 0;
00026             for(int k = 0; k < cols; k++) {
00027                 result->data[i][j] += this->data[i][k] * m.data[k][j];
00028             }
00029         }
00030     }
00031     return *result;
00032 }
00033
00034 // Dodaj 'a' do każdego elementu
00035 matrix& matrix::operator+(int a) {
00036     matrix* result = new matrix(rows, cols);
00037     for(int i = 0; i < rows; i++) {
00038         for(int j = 0; j < cols; j++) {
00039             result->data[i][j] = this->data[i][j] + a;
00040         }
00041     }
00042     return *result;
00043 }
00044
00045 // Pomnóż każdy element przez 'a'
00046 matrix& matrix::operator*(int a) {
00047     matrix* result = new matrix(rows, cols);
00048     for(int i = 0; i < rows; i++) {
00049         for(int j = 0; j < cols; j++) {
00050             result->data[i][j] = this->data[i][j] * a;
00051         }
00052     }
00053     return *result;
00054 }
00055
00056 // Odejmij 'a' od każdego elementu
00057 matrix& matrix::operator-(int a) {
00058     matrix* result = new matrix(rows, cols);
00059     for(int i = 0; i < rows; i++) {
00060         for(int j = 0; j < cols; j++) {
00061             result->data[i][j] = this->data[i][j] - a;
00062         }
00063     }
00064     return *result;
00065 }
00066
00067 // int + A
00068 matrix operator+(int a, matrix& m) {
00069     matrix result(m.rows, m.cols);
00070     for(int i = 0; i < m.rows; i++) {
00071         for(int j = 0; j < m.cols; j++) {
00072             result.data[i][j] = a + m.data[i][j];
00073         }
00074     }
00075     return result;
00076 }
00077
00078 // int * A
00079 matrix operator*(int a, matrix& m) {
00080     matrix result(m.rows, m.cols);
00081     for(int i = 0; i < m.rows; i++) {
00082         for(int j = 0; j < m.cols; j++) {
```

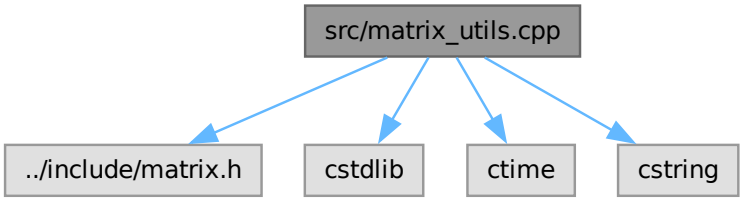
```
00083             result.data[i][j] = a * m.data[i][j];
00084         }
00085     }
00086     return result;
00087 }
00088
00089 // int - A
00090 matrix operator-(int a, matrix& m) {
00091     matrix result(m.rows, m.cols);
00092     for(int i = 0; i < m.rows; i++) {
00093         for(int j = 0; j < m.cols; j++) {
00094             result.data[i][j] = a - m.data[i][j];
00095         }
00096     }
00097     return result;
00098 }
00099
00100 // A++ (wszystkie elementy +1)
00101 matrix& matrix::operator++(int) {
00102     for(int i = 0; i < rows; i++) {
00103         for(int j = 0; j < cols; j++) {
00104             this->data[i][j]++;
00105         }
00106     }
00107     return *this;
00108 }
00109
00110 // A-- (wszystkie elementy -1)
00111 matrix& matrix::operator--(int) {
00112     for(int i = 0; i < rows; i++) {
00113         for(int j = 0; j < cols; j++) {
00114             this->data[i][j]--;
00115         }
00116     }
00117     return *this;
00118 }
00119
00120 // Powiększ każdy element o 'a'
00121 matrix& matrix::operator+=(int a) {
00122     for(int i = 0; i < rows; i++) {
00123         for(int j = 0; j < cols; j++) {
00124             this->data[i][j] += a;
00125         }
00126     }
00127     return *this;
00128 }
00129
00130 // Pomniejsz każdy element o 'a'
00131 matrix& matrix::operator-=(int a) {
00132     for(int i = 0; i < rows; i++) {
00133         for(int j = 0; j < cols; j++) {
00134             this->data[i][j] -= a;
00135         }
00136     }
00137     return *this;
00138 }
00139
00140 // Pomnóż każdy element o 'a'
00141 matrix& matrix::operator*=(int a) {
00142     for(int i = 0; i < rows; i++) {
00143         for(int j = 0; j < cols; j++) {
00144             this->data[i][j] *= a;
00145         }
00146     }
00147     return *this;
00148 }
00149
00150 // Dodaj część całkowitą z double do wszystkich elementów
00151 matrix& matrix::operator()(double value) {
00152     int intPart = static_cast<int>(value);
00153     for(int i = 0; i < rows; i++) {
00154         for(int j = 0; j < cols; j++) {
00155             this->data[i][j] += intPart;
00156         }
00157     }
00158     return *this;
00159 }
00160
00161 // Wypisywanie
00162 ostream& operator«(ostream& o, matrix& m) {
00163     for(int i = 0; i < m.rows; i++) {
00164         for(int j = 0; j < m.cols; j++) {
00165             o « m.data[i][j];
00166             if(j < m.cols - 1) o « " ";
00167         }
00168         o « endl;
00169     }
```

```
00170     return o;
00171 }
00172
00173 // Czy każdy element jest równy?
00174 bool matrix::operator==(const matrix& m) {
00175     if(rows != m.rows || cols != m.cols) return false;
00176     for(int i = 0; i < rows; i++) {
00177         for(int j = 0; j < cols; j++) {
00178             if(this->data[i][j] != m.data[i][j]) return false;
00179         }
00180     }
00181     return true;
00182 }
00183
00184 // Czy każdy element jest większy?
00185 bool matrix::operator>(const matrix& m) {
00186     if(rows != m.rows || cols != m.cols) return false;
00187     for(int i = 0; i < rows; i++) {
00188         for(int j = 0; j < cols; j++) {
00189             if(this->data[i][j] <= m.data[i][j]) return false;
00190         }
00191     }
00192     return true;
00193 }
00194
00195 // Czy każdy element jest mniejszy?
00196 bool matrix::operator<(const matrix& m) {
00197     if(rows != m.rows || cols != m.cols) return false;
00198     for(int i = 0; i < rows; i++) {
00199         for(int j = 0; j < cols; j++) {
00200             if(this->data[i][j] >= m.data[i][j]) return false;
00201         }
00202     }
00203     return true;
00204 }
```

4.7 Dokumentacja pliku src/matrix_utils.cpp

```
#include "../include/matrix.h"
#include <cstdlib>
#include <ctime>
#include <cstring>
```

Wykres zależności załączania dla matrix_utils.cpp:



4.8 matrix_utils.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "../include/matrix.h"
00002 #include <cstdlib>
00003 #include <ctime>
00004 #include <cstring>
```

```
00005
00006 using namespace std;
00007
00008 // 1. Wstawia wartość do komórki (wiersz x, kolumna y)
00009 matrix& matrix::wstaw(int x, int y, int wartosc) {
00010     data[x][y] = wartosc;
00011     return *this;
00012 }
00013
00014 // 2. Zwraca wartość z komórki (wiersz x, kolumna y)
00015 int matrix::pokaz(int x, int y) {
00016     return data[x][y];
00017 }
00018
00019 // 3. Dokonuje transpozycji macierzy w miejscu (zamienia wiersze z kolumnami)
00020 matrix& matrix::dowroc() {
00021     // Dla macierzy kwadratowej n x n
00022     if (rows != cols) {
00023         throw std::runtime_error("Transpozycja in-place wymaga macierzy kwadratowej");
00024     }
00025
00026     for (int i = 0; i < rows; ++i) {
00027         for (int j = i + 1; j < cols; ++j) {
00028             int temp = data[i][j];
00029             data[i][j] = data[j][i];
00030             data[j][i] = temp;
00031         }
00032     }
00033     return *this;
00034 }
00035
00036 // 4. Wypełnia wszystkie pola losowymi cyframi 0-9
00037 matrix& matrix::losuj() {
00038     srand(time(0));
00039     for (int i = 0; i < rows; ++i) {
00040         for (int j = 0; j < cols; ++j) {
00041             data[i][j] = rand() % 10;
00042         }
00043     }
00044     return *this;
00045 }
00046
00047 // 5. Losuje 'x' par współrzędnych i wstawia w nie losowe cyfry 0-9
00048 matrix& matrix::losuj(int x) {
00049     srand(time(0));
00050     for (int i = 0; i < x; ++i) {
00051         int row = rand() % rows;
00052         int col = rand() % cols;
00053         int value = rand() % 10;
00054         data[row][col] = value;
00055     }
00056     return *this;
00057 }
00058
00059 // 6. Przepisuje tablicę 't' na główną przekątną, reszta zerami
00060 matrix& matrix::diagonalna(int* t) {
00061     // Wyzeruj całą macierz
00062     for (int i = 0; i < rows; ++i) {
00063         for (int j = 0; j < cols; ++j) {
00064             data[i][j] = 0;
00065         }
00066     }
00067
00068     // Wstaw wartości na głównej przekątnej
00069     int minDim = (rows < cols) ? rows : cols;
00070     for (int i = 0; i < minDim; ++i) {
00071         data[i][i] = t[i];
00072     }
00073     return *this;
00074 }
00075
00076 // 7. Przepisuje tablicę 't' na przekątną przesuniętą o 'k'
00077 // k=0 główna, k>0 w górę (w prawo), k<0 w dół
00078 matrix& matrix::diagonalna_k(int k, int* t) {
00079     // Wyzeruj całą macierz
00080     for (int i = 0; i < rows; ++i) {
00081         for (int j = 0; j < cols; ++j) {
00082             data[i][j] = 0;
00083         }
00084     }
00085
00086     if (k >= 0) {
00087         // k > 0: przesuwa w prawo (górę)
00088         for (int i = 0; i < rows; ++i) {
00089             int j = i + k;
00090             if (j < cols) {
00091                 data[i][j] = t[i];
```

```
00092             }
00093         }
00094     } else {
00095         // k < 0: przesuwa w dół
00096         int absk = -k;
00097         for (int i = absk; i < rows; ++i) {
00098             int j = i - absk;
00099             data[i][j] = t[i - absk];
00100         }
00101     }
00102     return *this;
00103 }
00104
00105 // 8. Wpisuje tablicę 't' w kolumnę o indeksie x
00106 matrix& matrix::kolumna(int x, int* t) {
00107     for (int i = 0; i < rows; ++i) {
00108         data[i][x] = t[i];
00109     }
00110     return *this;
00111 }
00112
00113 // 9. Wpisuje tablicę 't' w wiersz o indeksie y
00114 matrix& matrix::wiersz(int y, int* t) {
00115     for (int j = 0; j < cols; ++j) {
00116         data[y][j] = t[j];
00117     }
00118     return *this;
00119 }
00120
00121 // 10. Wstawia 1 na przekątnej, 0 poza nią
00122 matrix& matrix::przekatna() {
00123     for (int i = 0; i < rows; ++i) {
00124         for (int j = 0; j < cols; ++j) {
00125             if (i == j) {
00126                 data[i][j] = 1;
00127             } else {
00128                 data[i][j] = 0;
00129             }
00130         }
00131     }
00132     return *this;
00133 }
00134
00135 // 11. Wstawia 1 pod przekątną, 0 na przekątnej i nad nią
00136 matrix& matrix::pod_przekatna() {
00137     for (int i = 0; i < rows; ++i) {
00138         for (int j = 0; j < cols; ++j) {
00139             if (i > j) {
00140                 data[i][j] = 1;
00141             } else {
00142                 data[i][j] = 0;
00143             }
00144         }
00145     }
00146     return *this;
00147 }
00148
00149 // 12. Wstawia 1 nad przekątną, 0 na przekątnej i pod nią
00150 matrix& matrix::nad_przekatna() {
00151     for (int i = 0; i < rows; ++i) {
00152         for (int j = 0; j < cols; ++j) {
00153             if (i < j) {
00154                 data[i][j] = 1;
00155             } else {
00156                 data[i][j] = 0;
00157             }
00158         }
00159     }
00160     return *this;
00161 }
00162
00163 // 13. Wypełnia wzorem szachownicy (0 i 1) zależnie od parzystości sumy indeksów
00164 matrix& matrix::szachownica() {
00165     for (int i = 0; i < rows; ++i) {
00166         for (int j = 0; j < cols; ++j) {
00167             if ((i + j) % 2 == 0) {
00168                 data[i][j] = 0;
00169             } else {
00170                 data[i][j] = 1;
00171             }
00172         }
00173     }
00174     return *this;
00175 }
```

Skorowidz