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

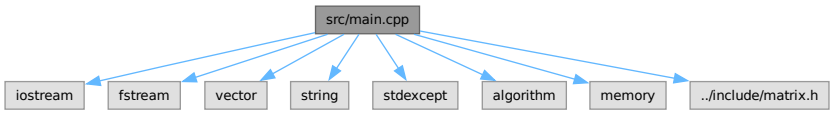
Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku src/main.cpp

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

Wykres zależności załączania dla main.cpp:



Funkcje

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

  Wczytaj macierz z pliku tekstowego Funkcja otwiera plik tekstowy i wczytuje wymiary macierzy oraz jej elementy.
- void wypisz_fragment (const matrix &m, size_t max_rows=5, size_t max_cols=5)

  Wypisz fragment macierzy na standardowe wyjscie Funkcja wyswietla pierwsze max_rows wierszy i max_cols kolumn macierzy.
- int main ()

  Glowna funkcja programu Program testuje wszystkie operacje i funkcjonalnosci biblioteki matrix.

4.1.1 Dokumentacja funkcji

4.1.1.1 main()

```
int main ()
```

Glowna funkcja programu Program testuje wszystkie operacje i funkcjonalnosci biblioteki matrix.

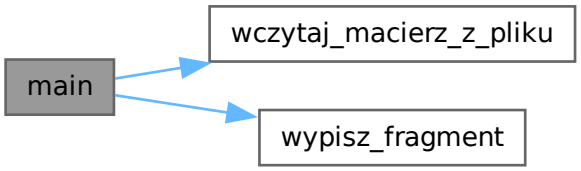
Wczytuje macierze z plikow, testuje operatory arytmetyczne, porownania i inne operacje dostepne w klasie matrix. Wyswietla wyniki testow w formacie czytelnym dla uzytkownika.

Zwraca

Kod zakonczenia programu (0 = sukces, 1 = blad)

Definicja w linii 68 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)
```

Wczytaj macierz z pliku tekstowego Funkcja otwiera plik tekstowy i wczytuje wymiary macierzy oraz jej elementy.

Spodziewany format pliku:

- Linia 1: liczba wierszy liczba kolumn
- Nastepne linie: elementy macierzy oddzielone spacjami

Parametry

filename	Sciezka do pliku zawierajacego dane macierzy

Zwraca

Macierz wczytana z pliku

Wyjątki

std::runtime_error	Jesli plik nie mogl byc otwarty

Definicja w linii 21 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)
```

Wypisz fragment macierzy na standardowe wyjscie Funkcja wyswietla pierwsze max_rows wierszy i max_cols kolumn macierzy.

Jesli macierz jest wieksza, dodaje "..." wskazujace na ukryte elementy.

Parametry

m	Macierz do wypisania
max_rows	Maksymalna liczba wierszy do wyswietlenia (domyslnie 5)
max_cols	Maksymalna liczba kolumn do wyswietlenia (domyslnie 5)

Definicja w linii 44 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>
00007 #include <memory>
00008 #include "../include/matrix.h"
00009
00010 using namespace std;
00011
00021 matrix wczytaj_macierz_z_pliku(const std::string& filename) {
00022     std::ifstream fin(filename);
00023     if (!fin) throw std::runtime_error("Nie można otworzyć pliku: " + filename);
00024
00025     size_t rows, cols;
00026     fin » rows » cols;
00027
00028     matrix result(rows, cols, 0.0);
00029     for (size_t r = 0; r < rows; ++r) {
00030         for (size_t c = 0; c < cols; ++c) {
00031             fin » result(r, c);
00032         }
00033     }
00034     return result;
00035 }
00036
00044 void wypisz_fragment(const matrix& m, size_t max_rows = 5, size_t max_cols = 5) {
00045     size_t rmax = std::min(static_cast<size_t>(m.get_rows()), max_rows);
00046     size_t cmax = std::min(static_cast<size_t>(m.get_cols()), max_cols);
00047
00048     for (size_t r = 0; r < rmax; ++r) {
00049         std::cout « "[ ";
00050         for (size_t c = 0; c < cmax; ++c) {
00051             std::cout « m(r, c) « " ";
00052         }
00053         if (cmax < static_cast<size_t>(m.get_cols()))
00054             std::cout « "...";
00055         std::cout « "]\n";
00056     }
00057     if (rmax < static_cast<size_t>(m.get_rows()))
00058         std::cout « "...\n";
00059 }
00060
00068 int main() {
00069     try {
00070         std::cout « "=== TESTOWANIE BIBLIOTEKI MATRIX ===\n\n";
00071
00072         std::cout « "Wczytywanie macierzy A z pliku...\n";
00073         matrix A = wczytaj_macierz_z_pliku("data/input_matrix_A.txt");
00074         std::cout « "Wymiary A: " « A.get_rows() « "x" « A.get_cols() « "\n\n";
00075
00076         std::cout « "Wczytywanie macierzy B z pliku...\n";
00077         matrix B = wczytaj_macierz_z_pliku("data/input_matrix_B.txt");
00078         std::cout « "Wymiary B: " « B.get_rows() « "x" « B.get_cols() « "\n\n";
00079
00080         std::cout « "--- Testowanie A+B ---\n";
00081         try {
00082             matrix AB = A + B;
00083             wypisz_fragment(AB);
00084         } catch (const std::exception& e) {
00085             std::cout « "Błąd: " « e.what() « "\n";
00086         }
00087
00088         std::cout « "\n--- Testowanie A*B ---\n";
00089         try {
00090             matrix AMB = A * B;
00091             wypisz_fragment(AMB);
00092         } catch (const std::exception& e) {
00093             std::cout « "Błąd: " « e.what() « "\n";
00094         }
00095
00096         std::cout « "\n--- Testowanie A+10 ---\n";
00097         matrix Aplus10 = A + 10;
00098         wypisz_fragment(Aplus10);
00099
00100         std::cout « "\n--- Testowanie A*5 ---\n";
00101         matrix Amul5 = A * 5;
00102         wypisz_fragment(Amul5);
00103
00104         std::cout « "\n--- Testowanie 10+A ---\n";
00105         matrix tenPlusA = 10 + A;
```

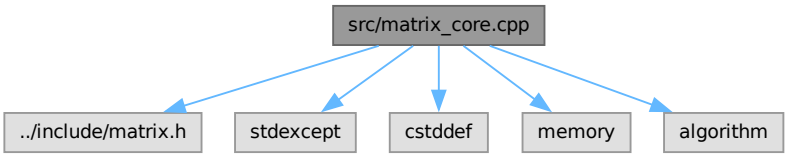
```
00106         wypisz_fragment(tenPlusA);
00107
00108         std::cout « "\n--- Testowanie 5*A ---\n";
00109         matrix fiveTimesA = 5 * A;
00110         wypisz_fragment(fiveTimesA);
00111
00112         std::cout « "\n--- Testowanie A-3 ---\n";
00113         matrix Aminus3 = A - 3;
00114         wypisz_fragment(Aminus3);
00115
00116         std::cout « "\n--- Testowanie 10-A ---\n";
00117         matrix tenMinusA = 10 - A;
00118         wypisz_fragment(tenMinusA);
00119
00120         std::cout « "\n--- Testowanie operator() (dostęp) ---\n";
00121         std::cout « "A(0,0) przed: " « A(0, 0) « "\n";
00122         A(0, 0) = 123.456;
00123         std::cout « "A(0,0) po: " « A(0, 0) « "\n";
00124
00125         std::cout « "\n--- Testowanie operator+= (skalarem) ---\n";
00126         matrix Ap = A;
00127         Ap += 3;
00128         wypisz_fragment(Ap);
00129
00130         std::cout « "\n--- Testowanie operator-= (skalarem) ---\n";
00131         matrix Am = A;
00132         Am -= 2;
00133         wypisz_fragment(Am);
00134
00135         std::cout « "\n--- Testowanie operator*= (skalarem) ---\n";
00136         matrix As = A;
00137         As *= 2;
00138         wypisz_fragment(As);
00139
00140         std::cout « "\n--- Testowanie inkrementacji ++ (postfix) ---\n";
00141         matrix Aincr = A;
00142         Aincr++;
00143         wypisz_fragment(Aincr);
00144
00145         std::cout « "\n--- Testowanie dekrementacji -- (postfix) ---\n";
00146         matrix Adecr = A;
00147         Adecr--;
00148         wypisz_fragment(Adecr);
00149
00150         std::cout « "\n--- Testowanie operator() z double ---\n";
00151         matrix Adouble = A;
00152         Adouble(3.7); // dodaj 3 do wszystkich elementów
00153         wypisz_fragment(Adouble);
00154
00155         std::cout « "\n--- Testowanie operatora porównania == ---\n";
00156         matrix C = A;
00157         std::cout « "A == C: " « ((A == C) ? "PRAWDA" : "FAŁSZ") « "\n";
00158         std::cout « "A == B: " « ((A == B) ? "PRAWDA" : "FAŁSZ") « "\n";
00159
00160         std::cout « "\n--- Testowanie operatora > ---\n";
00161         matrix D(3, 3, 0.5);
00162         matrix E(3, 3, 0.3);
00163         std::cout « "D(0.5) > E(0.3): " « ((D > E) ? "PRAWDA" : "FAŁSZ") « "\n";
00164
00165         std::cout « "\n--- Testowanie operatora < ---\n";
00166         std::cout « "E(0.3) < D(0.5): " « ((E < D) ? "PRAWDA" : "FAŁSZ") « "\n";
00167
00168         std::cout « "\n--- Wypisywanie     macierzy A (operator«) ---\n";
00169         std::cout « A;
00170
00171         std::cout « "\n=== KONIEC TESTÓW ===\n";
00172         return 0;
00173
00174     } catch (const std::exception& e) {
00175         std::cerr « "Błąd krytyczny: " « e.what() « std::endl;
00176         return 1;
00177     }
00178 }
```

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.

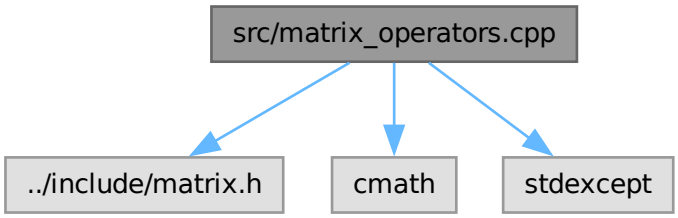
```
00001 #include "../include/matrix.h"
00002 #include <stdexcept>
00003 #include <cstddef>
00004 #include <memory>
00005 #include <algorithm>
00006
00018 matrix::matrix() noexcept : rows(0), cols(0), data(nullptr) {}
00019
00040 matrix::matrix(std::size_t r, std::size_t c, double value)
00041     : rows(static_cast<int>(r)), cols(static_cast<int>(c)) {
00042     alokuj(r * c);
00043     for (std::size_t i = 0; i < static_cast<std::size_t>(rows); ++i) {
00044         for (std::size_t j = 0; j < static_cast<std::size_t>(cols); ++j) {
00045             data[i][j] = value;
00046         }
00047     }
00048 }
00049
00069 matrix::matrix(std::initializer_list<std::initializer_list<double» init)
00070     : rows(static_cast<int>(init.size())),
00071       cols(init.size() ? static_cast<int>(init.begin()->size()) : 0) {
00072     alokuj(rows * cols);
00073     std::size_t r = 0;
00074     for (const auto& row : init) {
00075         if (row.size() != static_cast<std::size_t>(cols))
00076             throw std::runtime_error("Niezgodne długości wierszy w initializer_list");
00077         std::size_t c = 0;
00078         for (double val : row) {
00079             data[r][c] = val;
00080             ++c;
00081         }
00082         ++r;
00083     }
00084 }
00085
00108 matrix::matrix(const matrix& other)
00109     : rows(other.rows), cols(other.cols) {
00110     alokuj(rows * cols);
00111     for (int i = 0; i < rows; ++i) {
00112         for (int j = 0; j < cols; ++j) {
00113             data[i][j] = other.data[i][j];
00114         }
00115     }
00116 }
00117
00141 matrix& matrix::operator=(const matrix& other) {
00142     if (this == &other) {
00143         return *this;
00144     }
00145
00146     // Zwolnij starą pamięć
00147     data.reset();
00148
00149     rows = other.rows;
00150     cols = other.cols;
00151
00152     alokuj(rows * cols);
```

```
00153     for (int i = 0; i < rows; ++i) {
00154         for (int j = 0; j < cols; ++j) {
00155             data[i][j] = other.data[i][j];
00156         }
00157     }
00158 
00159     return *this;
00160 }
00161 
00183 void matrix::alokuj(std::size_t /*n*/) {
00184     data = std::make_unique<double*[]>(rows);
00185     for (int i = 0; i < rows; ++i) {
00186         data[i] = new double[cols];
00187     }
00188     for (int i = 0; i < rows; ++i) {
00189         for (int j = 0; j < cols; ++j) {
00190             data[i][j] = 0.0;
00191         }
00192     }
00193 }
00194 
00219 double& matrix::operator()(std::size_t r, std::size_t c) {
00220     return data[r][c];
00221 }
00222 
00247 double matrix::operator()(std::size_t r, std::size_t c) const {
00248     return data[r][c];
00249 }
```

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 228 pliku matrix_operators.cpp.

4.5.1.2 operator+()

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

Definicja w linii 198 pliku matrix_operators.cpp.

4.5.1.3 operator-()

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

Definicja w linii 258 pliku matrix_operators.cpp.

4.5.1.4 operator<<()

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

Definicja w linii 467 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
00037 matrix& matrix::operator+(matrix& m) {
00038     if (rows != m.rows || cols != m.cols)
00039         throw std::runtime_error("Nieprawidłowe wymiary dla dodawania");
00040     auto result = std::make_unique<matrix>(rows, cols);
00041     for(int i = 0; i < rows; i++) {
00042         for(int j = 0; j < cols; j++) {
00043             result->data[i][j] = this->data[i][j] + m.data[i][j];
00044         }
00045     }
00046     return *result.release();
00047 }
00048
00076 matrix& matrix::operator*(matrix& m) {
00077     if (cols != m.rows)
```

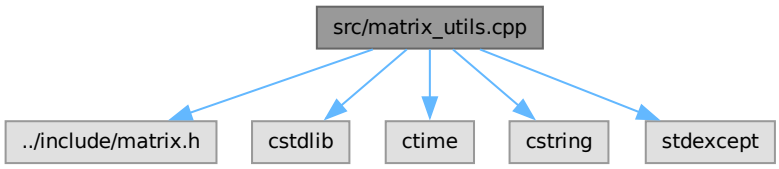
```
00078         throw std::runtime_error("Nieprawidłowe wymiary dla mnożenia");
00079     auto result = std::make_unique<matrix>(rows, m.cols);
00080     for(int i = 0; i < rows; i++) {
00081         for(int j = 0; j < m.cols; j++) {
00082             result->data[i][j] = 0;
00083             for(int k = 0; k < cols; k++) {
00084                 result->data[i][j] += this->data[i][k] * m.data[k][j];
00085             }
00086         }
00087     }
00088     return *result.release();
00089 }
00090
00110 matrix& matrix::operator+(int a) {
00111     auto result = std::make_unique<matrix>(rows, cols);
00112     for(int i = 0; i < rows; i++) {
00113         for(int j = 0; j < cols; j++) {
00114             result->data[i][j] = this->data[i][j] + a;
00115         }
00116     }
00117     return *result.release();
00118 }
00119
00139 matrix& matrix::operator*(int a) {
00140     auto result = std::make_unique<matrix>(rows, cols);
00141     for(int i = 0; i < rows; i++) {
00142         for(int j = 0; j < cols; j++) {
00143             result->data[i][j] = this->data[i][j] * a;
00144         }
00145     }
00146     return *result.release();
00147 }
00148
00168 matrix& matrix::operator-(int a) {
00169     auto result = std::make_unique<matrix>(rows, cols);
00170     for(int i = 0; i < rows; i++) {
00171         for(int j = 0; j < cols; j++) {
00172             result->data[i][j] = this->data[i][j] - a;
00173         }
00174     }
00175     return *result.release();
00176 }
00177
00198 matrix operator+(int a, matrix& m) {
00199     matrix result(m.rows, m.cols);
00200     for(int i = 0; i < m.rows; i++) {
00201         for(int j = 0; j < m.cols; j++) {
00202             result.data[i][j] = a + m.data[i][j];
00203         }
00204     }
00205     return result;
00206 }
00207
00228 matrix operator*(int a, matrix& m) {
00229     matrix result(m.rows, m.cols);
00230     for(int i = 0; i < m.rows; i++) {
00231         for(int j = 0; j < m.cols; j++) {
00232             result.data[i][j] = a * m.data[i][j];
00233         }
00234     }
00235     return result;
00236 }
00237
00258 matrix operator-(int a, matrix& m) {
00259     matrix result(m.rows, m.cols);
00260     for(int i = 0; i < m.rows; i++) {
00261         for(int j = 0; j < m.cols; j++) {
00262             result.data[i][j] = a - m.data[i][j];
00263         }
00264     }
00265     return result;
00266 }
00267
00288 matrix& matrix::operator++(int) {
00289     for(int i = 0; i < rows; i++) {
00290         for(int j = 0; j < cols; j++) {
00291             this->data[i][j]++;
00292         }
00293     }
00294     return *this;
00295 }
00296
00317 matrix& matrix::operator--(int) {
00318     for(int i = 0; i < rows; i++) {
00319         for(int j = 0; j < cols; j++) {
00320             this->data[i][j]--;
00321         }
```

```
00322     }
00323     return *this;
00324 }
00325
00345 matrix& matrix::operator+=(int a) {
00346     for(int i = 0; i < rows; i++) {
00347         for(int j = 0; j < cols; j++) {
00348             this->data[i][j] += a;
00349         }
00350     }
00351     return *this;
00352 }
00353
00373 matrix& matrix::operator-=(int a) {
00374     for(int i = 0; i < rows; i++) {
00375         for(int j = 0; j < cols; j++) {
00376             this->data[i][j] -= a;
00377         }
00378     }
00379     return *this;
00380 }
00381
00401 matrix& matrix::operator*=(int a) {
00402     for(int i = 0; i < rows; i++) {
00403         for(int j = 0; j < cols; j++) {
00404             this->data[i][j] *= a;
00405         }
00406     }
00407     return *this;
00408 }
00409
00432 matrix& matrix::operator()(double value) {
00433     int intPart = static_cast<int>(value);
00434     for(int i = 0; i < rows; i++) {
00435         for(int j = 0; j < cols; j++) {
00436             this->data[i][j] += intPart;
00437         }
00438     }
00439     return *this;
00440 }
00441
00467 ostream& operator«(ostream& o, matrix& m) {
00468     for(int i = 0; i < m.rows; i++) {
00469         for(int j = 0; j < m.cols; j++) {
00470             o « m.data[i][j];
00471             if(j < m.cols - 1) o « " ";
00472         }
00473         o « endl;
00474     }
00475     return o;
00476 }
00477
00503 bool matrix::operator==(const matrix& m) {
00504     if(rows != m.rows || cols != m.cols) return false;
00505     for(int i = 0; i < rows; i++) {
00506         for(int j = 0; j < cols; j++) {
00507             if(this->data[i][j] != m.data[i][j]) return false;
00508         }
00509     }
00510     return true;
00511 }
00512
00541 bool matrix::operator>(const matrix& m) {
00542     if(rows != m.rows || cols != m.cols) return false;
00543     for(int i = 0; i < rows; i++) {
00544         for(int j = 0; j < cols; j++) {
00545             if(this->data[i][j] <= m.data[i][j]) return false;
00546         }
00547     }
00548     return true;
00549 }
00550
00579 bool matrix::operator<(const matrix& m) {
00580     if(rows != m.rows || cols != m.cols) return false;
00581     for(int i = 0; i < rows; i++) {
00582         for(int j = 0; j < cols; j++) {
00583             if(this->data[i][j] >= m.data[i][j]) return false;
00584         }
00585     }
00586     return true;
00587 }
```

4.7 Dokumentacja pliku src/matrix_utils.cpp

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

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 #include <stdexcept>
00006
00007 using namespace std;
00008
00037 matrix& matrix::wstaw(int x, int y, int wartosc) {
00038     data[x][y] = wartosc;
00039     return *this;
00040 }
00041
00068 int matrix::pokaz(int x, int y) {
00069     return data[x][y];
00070 }
00071
00100 matrix& matrix::dowroc() {
00101     if (rows != cols) {
00102         throw std::runtime_error("Transpozycja in-place wymaga macierzy kwadratowej");
00103     }
00104
00105     for (int i = 0; i < rows; ++i) {
00106         for (int j = i + 1; j < cols; ++j) {
00107             double temp = data[i][j];
00108             data[i][j] = data[j][i];
00109             data[j][i] = temp;
00110         }
00111     }
00112     return *this;
00113 }
00114
00140 matrix& matrix::losuj() {
00141     srand(time(0));
00142     for (int i = 0; i < rows; ++i) {
00143         for (int j = 0; j < cols; ++j) {
00144             data[i][j] = rand() % 10;
00145         }
00146     }
00147     return *this;
00148 }
00149
00178 matrix& matrix::losuj(int x) {
00179     srand(time(0));
00180     for (int i = 0; i < x; ++i) {
00181         int row = rand() % rows;
```

```
00182         int col = rand() % cols;
00183         int value = rand() % 10;
00184         data[row][col] = value;
00185     }
00186     return *this;
00187 }
00188
00221 matrix& matrix::diagonalna(int* t) {
00222     // Wyzeruj całą macierz
00223     for (int i = 0; i < rows; ++i) {
00224         for (int j = 0; j < cols; ++j) {
00225             data[i][j] = 0;
00226         }
00227     }
00228
00229     // Wstaw wartości na głównej przekątnej
00230     int minDim = (rows < cols) ? rows : cols;
00231     for (int i = 0; i < minDim; ++i) {
00232         data[i][i] = t[i];
00233     }
00234     return *this;
00235 }
00236
00283 matrix& matrix::diagonalna_k(int k, int* t) {
00284     // Wyzeruj całą macierz
00285     for (int i = 0; i < rows; ++i) {
00286         for (int j = 0; j < cols; ++j) {
00287             data[i][j] = 0;
00288         }
00289     }
00290
00291     if (k >= 0) {
00292         // k >= 0: przesuwa w prawo (górę)
00293         for (int i = 0; i < rows; ++i) {
00294             int j = i + k;
00295             if (j < cols) {
00296                 data[i][j] = t[i];
00297             }
00298         }
00299     } else {
00300         // k < 0: przesuwa w dół
00301         int absk = -k;
00302         for (int i = absk; i < rows; ++i) {
00303             int j = i - absk;
00304             data[i][j] = t[i - absk];
00305         }
00306     }
00307     return *this;
00308 }
00309
00339 matrix& matrix::kolumna(int x, int* t) {
00340     for (int i = 0; i < rows; ++i) {
00341         data[i][x] = t[i];
00342     }
00343     return *this;
00344 }
00345
00375 matrix& matrix::wiersz(int y, int* t) {
00376     for (int j = 0; j < cols; ++j) {
00377         data[y][j] = t[j];
00378     }
00379     return *this;
00380 }
00381
00408 matrix& matrix::przekatna() {
00409     for (int i = 0; i < rows; ++i) {
00410         for (int j = 0; j < cols; ++j) {
00411             if (i == j) {
00412                 data[i][j] = 1;
00413             } else {
00414                 data[i][j] = 0;
00415             }
00416         }
00417     }
00418     return *this;
00419 }
00420
00444 matrix& matrix::pod_przekatna() {
00445     for (int i = 0; i < rows; ++i) {
00446         for (int j = 0; j < cols; ++j) {
00447             if (i > j) {
00448                 data[i][j] = 1;
00449             } else {
00450                 data[i][j] = 0;
00451             }
00452         }
00453     }
```

```
00454     return *this;
00455 }
00456
00480 matrix& matrix::nad_przekatna() {
00481     for (int i = 0; i < rows; ++i) {
00482         for (int j = 0; j < cols; ++j) {
00483             if (i < j) {
00484                 data[i][j] = 1;
00485             } else {
00486                 data[i][j] = 0;
00487             }
00488         }
00489     }
00490     return *this;
00491 }
00492
00520 matrix& matrix::szachownica() {
00521     for (int i = 0; i < rows; ++i) {
00522         for (int j = 0; j < cols; ++j) {
00523             if ((i + j) % 2 == 0) {
00524                 data[i][j] = 0;
00525             } else {
00526                 data[i][j] = 1;
00527             }
00528         }
00529     }
00530     return *this;
00531 }
```

Rozdział 5

Przykłady

5.1 /home/ninja/ANS/Programowanie zaawansowane/matrix-copilot/src/matrix_core.cpp

Konstruktor z parametrami - tworzy macierz o podanych wymiarach.

Konstruktor z parametrami - tworzy macierz o podanych wymiarachAlokuje pamięć dla macierzy o wymiarach r×c i inicjalizuje wszystkie elementy wartością `value`.

Parametry

r	liczba wierszy (konwertowana do int)
c	liczba kolumn (konwertowana do int)
value	wartość początkowa wszystkich elementów (domyślnie 0.0)

Wyjątki

std::bad_alloc	jeśli alokacja pamięci się nie powiedzie

Warunek końcowy

rows == r, cols == c, wszystkie elementy == value @complexity O(r × c) - dla alokacji i inicjalizacji

```
matrix m(3, 3, 1.5);  // macierz 3×3 wypełniona wartościami 1.5
```

zaawansowane/matrix-copilot/src/matrix_core.cpp

5.2 /home/ninja/ANS/Programowanie zaawansowane/matrix-copilot/src/matrix_operators.cpp

Operator dodawania macierzy - A + B.

Operator dodawania macierzy - A + BDodaje dwie macierze tego samego rozmiaru, zwracając nową macierz zawierającą sumę elementów. Operacja odbywa się element po elemencie.

Parametry

m	macierz do dodania (prawy operand)

Zwraca

referencja na nową macierz zawierającą sumę A + B

Wyjątki

std::runtime_error	jeśli wymiary macierzy nie są zgodne (rows lub cols się różnią)

Warunek wstępny

this->rows == m.rows && this->cols == m.cols

Warunek końcowy

wynikowa macierz[i][j] = this[i][j] + m[i][j] @complexity O(n × m) gdzie n = rows, m = cols

```
matrix A(2, 2);
A(0, 0) = 1; A(0, 1) = 2;
A(1, 0) = 3; A(1, 1) = 4;

matrix B(2, 2);
B(0, 0) = 5; B(0, 1) = 6;
B(1, 0) = 7; B(1, 1) = 8;

matrix C = A + B;  // C[0][0] = 6, C[0][1] = 8, ...
```

Ostrzeżenie

Używa `release()` na unique_ptr - być ostrożnym z zarządzaniem pamięcią!

zaawansowane/matrix-copilot/src/matrix_operators.cpp

5.3 /home/ninja/ANS/Programowanie zaawansowane/matrix-copilot/src/matrix_utils.cpp

Wstawia wartość do komórki macierzy.

Wstawia wartość do komórki macierzyUstawia wartość elementu na pozycji (x, y) na podaną wartość. Brak sprawdzenia granic - nieprawidłowe indeksy mogą spowodować niezdefiniowane zachowanie.

Parametry

x	indeks wiersza (0-based)
y	indeks kolumny (0-based)
wartosc	wartość do wstawienia (konwertowana z int na double)

Zwraca

referencja na bieżącą macierz (umożliwia łańcuchowanie operacji)

Warunek wstępny

0 <= x < rows && 0 <= y < cols

Warunek końcowy

data[x][y] == wartosc @complexity O(1)

```
matrix m(3, 3);
m.wstaw(0, 0, 5).wstaw(1, 1, 10);  // łańcuchowanie operacji
```

Ostrzeżenie

Brak sprawdzenia granic! Użycie nieprawidłowych indeksów spowoduje niezdefiniowane zachowanie

Zobacz również

pokaz()

zaawansowane/matrix-copilot/src/matrix_utils.cpp

Skorowidz