Matrix

4

Wygenerowano za pomocą Doxygen 1.12.0

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 2

Indeks plików

2.1 Lista plików

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

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja klasy matrix

Klasa reprezentujaca macierz kwadratowa.

`#include <Matrix.h>`

Metody publiczne

- matrix (void)
- matrix (int n)
- matrix (int n, int ∗t)
- matrix (matrix &m)
- ∼matrix (void)
- matrix & alokuj (int n)
- matrix & wstaw (int x, int y, int wartosc)
- int pokaz (int x, int y)
- matrix & odwroc (void)
- matrix & losuj (void)
- matrix & losuj (int x)
- matrix & diagonalna (int ∗t)
- matrix & diagonalna_k (int k, int ∗t)
- matrix & kolumna (int x, int ∗t)
- matrix & wiersz (int y, int ∗t)
- matrix & przekatna (void)
- matrix & pod_przekatna (void)
- matrix & nad_przekatna (void)
- matrix & szachownica (void)
- matrix & operator+ (matrix &m)
- matrix & operator∗ (matrix &m)
- matrix & operator+ (int a)
- matrix & operator∗ (int a)
- matrix & operator- (int a)
- matrix & operator++ (int)
- matrix & operator-- (int)
- matrix & operator+= (int a)
- matrix & operator-= (int a)
- matrix & operator∗= (int a)
- matrix & operator+= (double a)
- bool operator== (const matrix &m)
- bool operator> (const matrix &m)
- bool operator< (const matrix &m)
- matrix & wczytaj (const std::string &nazwa)

Atrybuty prywatne

- int n
- int ∗∗ tab

Przyjaciele

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

3.1.1 Opis szczegÃ³Åowy

Klasa reprezentujaca macierz kwadratowa.

Klasa ta pozwala na tworzenie i manipulowanie macierzami kwadratowymi o dynamicznie alokowanej pamieci. Zawiera metody umozliwiajace wykonywanie roznych operacji na macierzach, takich jak dodawanie, mnozenie, wstawianie wartosci, czy odwracanie macierzy.

Definicja w linii 14 pliku Matrix.h.

3.1.2 Dokumentacja konstruktora i destruktora

3.1.2.1 matrix() [1/4]

```
matrix::matrix (
            void )
```

Konstruktor domylny bez alokacji pamiêci

nie przyjmuje ¿adnych argumentów i nie alokuje pamiêci na macierz

Definicja w linii 3 pliku Matrix.cpp.

3.1.2.2 matrix() [2/4]

```
matrix::matrix (
            int n)
```

Konstruktor przeci¹¿eniowy alokuje macierz o wymiarach n na n przyjmuje jeden argument typu int, który okrela rozmiar macierzy

Definicja w linii 9 pliku Matrix.cpp.

3.1.2.3 matrix() [3/4]

```
matrix::matrix (
            int n,
            int * t)
```

Konstruktor przeci¹¿eniowy alokuje pamiêæ i przepisuje dane z tabeli przyjmuje dwa argumenty: int n - rozmiar macierzy, int* t - tablica z danymi

Definicja w linii 22 pliku Matrix.cpp.

3.1.2.4 matrix() [4/4]

```
matrix::matrix (
            matrix & m)
```

Konstruktor kopiuj¹cy przyjmuje jeden argument typu matrix, który jest kopiowany

Definicja w linii 40 pliku Matrix.cpp.

3.1.2.5 ~matrix()

```
matrix::~matrix (
            void )
```

Definicja w linii 56 pliku Matrix.cpp.

3.1.3 Dokumentacja funkcji skÅadowych

3.1.3.1 alokuj()

```
matrix & matrix::alokuj (
            int n)
```

Metoda alokuj alokuje pamiêæ na macierz Jeli macierz nie ma zaalokowanej pamiêci to j¹ alokuje w wielkoci n na n Jeli ma zaalokowan¹ pamiêæ to rozwa¿a przypadki gdy jest jej mniej lub wiêcej ni¿ n na n przez co odpowiednio usuwa lub dodaje nadmiarowe lub brakuj¹ce komórki

Definicja w linii 67 pliku Matrix.cpp.

3.1.3.2 diagonalna()

```
matrix & matrix::diagonalna (
            int * t)
```

Metoda diagonalna po przek¹tnej s¹ wpisane dane z tabeli, a pozosta^{3}e elementy s¹ równe 0 przyjmuje jeden argument typu int*, który jest tablic¹ z danymi które zostan¹ wpisane na przek¹tn¹

Definicja w linii 186 pliku Matrix.cpp.

3.1.3.3 diagonalna_k()

```
matrix & matrix::diagonalna_k (
            int k,
            int * t)
```

Metoda diagonalna_k po przek¹tnej s¹ wpisane dane z tabeli przesuniête o k, a pozosta[3]e elementy s¹ równe 0 przyjmuje dwa argumenty: int k - przesuniêcie przek¹tnej, int* t - tablica z danymi

Definicja w linii 194 pliku Matrix.cpp.

3.1.3.4 kolumna()

```
matrix & matrix::kolumna (
            int x,
            int * t)
```

Metoda kolumna przepisuje dane z tabeli do kolumny, któr¹ wskazuje zmienna x przyjmuje dwa argumenty: int x - kolumna, int* t - tablica z danymi

Definicja w linii 219 pliku Matrix.cpp.

3.1.3.5 losuj() [1/2]

```
matrix & matrix::losuj (
            int x)
```

Metoda losuj wype[3]nia macierz cyframi od 0 do 9 elementy macierzy. Zmienna x okrela ile cyfr bêdziemy losowaæ. przyjmuje jeden argument typu int, który okrela ile cyfr bêdziemy losowaæ

Definicja w linii 177 pliku Matrix.cpp.

3.1.3.6 losuj() [2/2]

```
matrix & matrix::losuj (
            void )
```

Metoda losuj wype[3]nia macierz cyframi od 0 do 9 wszystkie elementy macierzy nie przyjmuje ¿adnych argumentów

Definicja w linii 165 pliku Matrix.cpp.

3.1.3.7 nad_przekatna()

```
matrix & matrix::nad_przekatna (
            void )
```

Metoda nad_przekatna uzupe[3]nia macierz: 1-nad przek¹tn¹, 0-pod przek¹tn¹ i po przek¹tnej, nie przyjmuje ¿adnych argumentów

Definicja w linii 261 pliku Matrix.cpp.

3.1.3.8 odwroc()

```
matrix & matrix::odwroc (
            void )
```

Metoda odwroc zamienia wiersze z kolumnami nie przyjmuje ¿adnych argumentów

Definicja w linii 134 pliku Matrix.cpp.

3.1.3.9 operator∗() [1/2]

```
matrix & matrix::operator* (
            int a)
```

Metoda operator∗ mno¿y macierz przez liczbê przyjmuje jeden argument typu int, który jest mno¿ony przez macierz

Definicja w linii 354 pliku Matrix.cpp.

3.1.3.10 operator∗() [2/2]

```
matrix & matrix::operator* (
            matrix & m)
```

Metoda operator- odejmuje dwie macierze przyjmuje jeden argument typu matrix, który jest odejmowany od macierzy

Definicja w linii 326 pliku Matrix.cpp.

3.1.3.11 operator∗=()

```
matrix & matrix::operator*= (
            int a)
```

Metoda operator∗= ka¿dy element w macierzy mno¿ymy o a przyjmuje jeden argument typu int, który jest mno¿ony przez macierz

Definicja w linii 461 pliku Matrix.cpp.

3.1.3.12 operator+() [1/2]

```
matrix & matrix::operator+ (
            int a)
```

Metoda operator+ dodaje liczbê do macierzy przyjmuje jeden argument typu int, który jest dodawany do macierzy

Definicja w linii 342 pliku Matrix.cpp.

3.1.3.13 operator+() [2/2]

```
matrix & matrix::operator+ (
            matrix & m)
```

Metoda operator+ dodaje dwie macierze przyjmuje jeden argument typu matrix, który jest dodawany do macierzy

Definicja w linii 314 pliku Matrix.cpp.

3.1.3.14 operator++()

```
matrix & matrix::operator++ (
            int )
```

Metoda operator++ wszystkie liczby powiêkszone o 1 nie przyjmuje ¿adnych argumentów

Definicja w linii 415 pliku Matrix.cpp.

3.1.3.15 operator+=() [1/2]

```
matrix & matrix::operator+= (
            double a)
```

Metoda operator+= ka¿dy element w macierzy powiêkszamy o a przyjmuje jeden argument typu double, który jest dodawany do macierzy

Definicja w linii 472 pliku Matrix.cpp.

3.1.3.16 operator+=() [2/2]

```
matrix & matrix::operator+= (
            int a)
```

Metoda operator+= ka¿dy element w macierzy powiêkszamy o a przyjmuje jeden argument typu int, który jest dodawany do macierzy

Definicja w linii 439 pliku Matrix.cpp.

3.1.3.17 operator-()

```
matrix & matrix::operator- (
            int a)
```

Metoda operator- odejmuje liczbê od macierzy przyjmuje jeden argument typu int, który jest odejmowany od macierzy

Definicja w linii 366 pliku Matrix.cpp.

3.1.3.18 operator--()

```
matrix & matrix::operator-- (
            int )
```

Metoda operator-- wszystkie liczby pomniejszone o 1 nie przyjmuje ¿adnych argumentów

Definicja w linii 427 pliku Matrix.cpp.

3.1.3.19 operator-=()

```
matrix & matrix::operator-= (
            int a)
```

Metoda operator-= ka¿dy element w macierzy pomniejszamy o a przyjmuje jeden argument typu int, który jest odejmowany od macierzy

Definicja w linii 450 pliku Matrix.cpp.

3.1.3.20 operator<()

```
bool matrix::operator< (
            const matrix & m)
```

Metoda operator< sprawdza, czy ka¿dy element macierzy spe[3]nia nierównoæ ??(??, ??) < ??(??, ??).Jeli tak, to mo¿emy powiedzieæ, ¿e macierz jest mniejsza, w przeciwnym wypadku nie mo¿emy stwierdziæ,¿e macierz jest mniejsza. przyjmuje jeden argument typu matrix, który jest porównywany z macierz[1]

Definicja w linii 533 pliku Matrix.cpp.

3.1.3.21 operator==()

```
bool matrix::operator== (
            const matrix & m)
```

Metoda operator== sprawdza, czy ka¿dy element macierzy spe[3]nia równoæ ??(??, ??) = ??(??, ??) A = | 1 2 | B = | 1 2| |3 4| |3 4 | jeli nie, to nie mo¿emy mówiæ, ¿e macierze s[1] równe przyjmuje jeden argument typu matrix, który jest porównywany z macierz[1]

Definicja w linii 497 pliku Matrix.cpp.

3.1.3.22 operator>()

```
bool matrix::operator> (
            const matrix & m)
```

Metoda operator> sprawdza, czy ka¿dy element macierzy spe[3]nia nierównoæ ??(??, ??) > ??(??, ??).Jeli tak, to mo¿emy powiedzieæ, ¿e macierz jest wiêksza, w przeciwnym wypadku nie mo¿emy stwierdziæ,¿e macierz jest wiêksza. przyjmuje jeden argument typu matrix, który jest porównywany z macierz[1]

Definicja w linii 515 pliku Matrix.cpp.

3.1.3.23 pod_przekatna()

```
matrix & matrix::pod_przekatna (
            void )
```

Metoda pod_przekatna uzupe[3]nia macierz: 1-pod przek[1]tn[1], 0-nad przek[1]tn[1] i po przek[1]tnej, nie przyjmuje ¿adnych argumentów

Definicja w linii 243 pliku Matrix.cpp.

3.1.3.24 pokaz()

```
int matrix::pokaz (
            int x,
            int y)
```

Metoda pokaz zwraca wartoæ elementu x, y przyjmuje dwa argumenty: int x - wiersz, int y - kolumna

Definicja w linii 130 pliku Matrix.cpp.

3.1.3.25 przekatna()

```
matrix & matrix::przekatna (
            void )
```

Metoda przekatna uzupe[3]nia macierz: 1-na przek[1]tnej, 0-poza przek[1]tn[1], nie przyjmuje ¿adnych argumentów

Definicja w linii 235 pliku Matrix.cpp.

3.1.3.26 szachownica()

```
matrix & matrix::szachownica (
            void )
```

Metoda szachownica uzupelnia macierz w ten sposób dla n=4: 0101 1010 0101 1010 nie przyjmuje ¿adnych argumentów

Definicja w linii 279 pliku Matrix.cpp.

3.1.3.27 wczytaj()

```
matrix & matrix::wczytaj (
            const std::string & nazwa)
```

Metoda wczytaj wczytuje macierz z pliku o nazwie nazwa przyjmuje jeden argument typu string, który jest nazw[1] pliku

Definicja w linii 551 pliku Matrix.cpp.

3.1.3.28 wiersz()

```
matrix & matrix::wiersz (
            int y,
            int * t)
```

Metoda wiersz przepisuje dane z tabeli do wiersza, który wskazuje zmienna x przyjmuje dwa argumenty: int y - wiersz, int* t - tablica z danymi

Definicja w linii 227 pliku Matrix.cpp.

3.1.3.29 wstaw()

```
matrix & matrix::wstaw (
            int x,
            int y,
            int wartosc)
```

Metoda wstaw wstawia wartoæ do macierzy przyjmuje trzy argumenty: int x - wiersz, int y - kolumna, int wartosc - wartoæ

Definicja w linii 125 pliku Matrix.cpp.

3.1.4 Dokumentacja przyjaciÃ^{3}Å i powiÄzanych symboli

3.1.4.1 operator*

```
matrix operator* (
            int a,
            matrix & m)  [friend]
```

Metoda operator* mno¿y macierz przez liczbê przyjmuje jeden argument typu double, który jest mno¿ony przez macierz

Definicja w linii 391 pliku Matrix.cpp.

3.1.4.2 operator+

```
matrix operator+ (
            int a,
            matrix & m)  [friend]
```

Metoda operator+ dodaje liczbê do macierzy przyjmuje jeden argument typu double, który jest dodawany do macierzy

Definicja w linii 378 pliku Matrix.cpp.

3.1.4.3 operator-

```
matrix operator- (
            int a,
            matrix & m)  [friend]
```

Metoda operator- odejmuje liczbê od macierzy przyjmuje jeden argument typu double, który jest odejmowany od macierzy

Definicja w linii 403 pliku Matrix.cpp.

3.1.4.4 operator<<

```
std::ostream & operator<< (
            std::ostream & o,
            matrix & m)  [friend]
```

Metoda operator<< wypisanie macierzy przyjmuje dwa argumenty: std::ostream& o - strumieñ wyjciowy, matrix& m - macierz

Definicja w linii 485 pliku Matrix.cpp.

3.1.5 Dokumentacja atrybutÃ³w skÅadowych

3.1.5.1 n

```
int matrix::n  [private]
```

Definicja w linii 17 pliku Matrix.h.

3.1.5.2 tab

```
int** matrix::tab  [private]
```

Definicja w linii 18 pliku Matrix.h.

Dokumentacja dla tej klasy zostaÅa wygenerowana z plikÃ³w:

- Matrix/Matrix.h
- Matrix/Matrix.cpp

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku Matrix/main.cpp

```
#include "Matrix.h"
#include <iostream>
```

Funkcje

- int main ()

4.1.1 Dokumentacja funkcji

4.1.1.1 main()

```
int main ()
```

Definicja w linii 4 pliku main.cpp.

4.2 main.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "Matrix.h"
00002 #include <iostream>
00003
00004 int main()
00005 {
00006         // Testowanie
00007         matrix m1; // Konstruktor domyślny
00008         std::cout<<"Test konstruktora domyślnego: \n" << m1 << std::endl;
00009         // Działa poprawnie
00010         matrix m2(30); // Konstruktor przeciążeniowy
00011         std::cout << "Test konstruktora przeciążeniowego: \n" << m2 << std::endl;
00012         // Działa poprawnie
00013         matrix m3(3, new int[9] {1, 2, 3, 4, 5, 6, 7, 8, 9}); // Konstruktor przeciążeniowy
00014         std::cout << "Test konstruktora przeciążeniowego z tablicą: \n" << m3 << std::endl;
00015         // Działa poprawnie
00016         matrix m4(m3); // Konstruktor kopiujący
00017         std::cout << "Test konstruktora kopiującego: \n" << m4 << std::endl;
00018         // Działa poprawnie
```

```
00019         matrix m5;
00020         m5.alokuj(30); // Testowanie metody alokuj
00021         std::cout « "Test metody alokuj: \n" « m5 « std::endl;
00022         // Działa poprawnie
00023         matrix m6;// Testowanie metody wczytaj
00024         m6.wczytaj("matrix.txt"); // Wczytanie macierzy z pliku
00025         std::cout « "Test metody wczytaj: \n" « m6 « std::endl;
00026         // Działa poprawnie
00027         matrix m7(m6);
00028         m7.wstaw(1, 1, 23); // Testowanie metody wstaw
00029         std::cout « "Test metody wstaw: \n" « m7 « std::endl;
00030         // Działa poprawnie
00031         matrix m8(m7);
00032         std::cout « "Test metody pokaz: \n" « m8.pokaz(1, 1) « std::endl; // Testowanie metody pokaz
00033         // Działa poprawnie
00034         matrix m9(m6);
00035         m9.odwroc(); // Testowanie metody odwroc
00036         std::cout « "Test metody odwroc: \n" « m9 « std::endl;
00037         // Działa poprawnie
00038         matrix m10(30);
00039         m10.losuj(); // Testowanie metody losuj
00040         std::cout « "Test metody losuj: \n" « m10 « std::endl;
00041         // Działa poprawnie
00042         matrix m11(30);
00043         m11.losuj(10); // Testowanie metody losuj
00044         std::cout « "Test metody losuj ze zmienna: \n" « m11 « std::endl;
00045         // Działa poprawnie
00046         matrix m12(30);
00047         m12.diagonalna(new int[30] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
      20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30}); // Testowanie metody diagonalna
00048         std::cout « "Test metody diagonalna: \n" « m12 « std::endl;
00049         // Działa poprawnie
00050         matrix m13(30);
00051         m13.diagonalna_k(2, new int[30] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
      18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30}); // Testowanie metody diagonalna_k
00052         std::cout « "Test metody diagonalna_k: \n" « m13 « std::endl;
00053         // Działa poprawnie
00054         matrix m14(30);
00055         m14.kolumna(1, new int[30] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
      20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30}); // Testowanie metody kolumna
00056         std::cout « "Test metody kolumna: \n" « m14 « std::endl;
00057         // Działa poprawnie
00058         matrix m15(30);
00059         m15.wiersz(1, new int[30] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
      20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30}); // Testowanie metody wiersz
00060         std::cout « "Test metody wiersz: \n" « m15 « std::endl;
00061         // Działa poprawnie
00062         matrix m16(30);
00063         m16.przekatna(); // Testowanie metody przekatna
00064         std::cout « "Test metody przekatna: \n" « m16 « std::endl;
00065         // Działa poprawnie
00066         matrix m17(30);
00067         m17.pod_przekatna(); // Testowanie metody pod_przekatna
00068         std::cout « "Test metody pod_przekatna: \n" « m17 « std::endl;
00069         // Działa poprawnie
00070         matrix m18(30);
00071         m18.nad_przekatna(); // Testowanie metody nad_przekatna
00072         std::cout « "Test metody nad_przekatna: \n" « m18 « std::endl;
00073         // Działa poprawnie
00074         matrix m19(30);
00075         m19.szachownica(); // Testowanie metody szachownica
00076         std::cout « "Test metody szachownica: \n" « m19 « std::endl;
00077         // Działa poprawnie
00078         matrix m20(m15);
00079         matrix m21(m14);
00080         m21 = m20 + m19; // Testowanie operatora +
00081         std::cout « "Test operatora +: \n" « m21 « std::endl;
00082         // Działa poprawnie
00083         matrix m22(m15);
00084         matrix m23(m14);
00085         m23 = m22 * m19; // Testowanie operatora *
00086         std::cout « "Test operatora *: \n" « m23 « std::endl;
00087         // Działa poprawnie
00088         matrix m24(30);
00089         m24 = m24 + 5; // Testowanie operatora +
00090         std::cout « "Test operatora +: \n" « m24 « std::endl;
00091         // Działa poprawnie
00092         matrix m25(30);
00093         m25 = m25 * 5; // Testowanie operatora *
00094         std::cout « "Test operatora *: \n" « m25 « std::endl;
00095         // Działa poprawnie
00096         matrix m26(30);
00097         m26 = m26 - 5; // Testowanie operatora -
00098         std::cout « "Test operatora -: \n" « m26 « std::endl;
00099         // Działa poprawnie
00100         matrix m27(30);
00101         m27 += 5; // Testowanie operatora +=
```

```
00102         std::cout « "Test operatora +=: \n" « m27 « std::endl;
00103         // Działa poprawnie
00104         matrix m28(30);
00105         m28 -= 5; // Testowanie operatora -=
00106         std::cout « "Test operatora -=: \n" « m28 « std::endl;
00107         // Działa poprawnie
00108         matrix m29(30);
00109         m29 *= 5; // Testowanie operatora *=
00110         std::cout « "Test operatora *=: \n" « m29 « std::endl;
00111         // Działa poprawnie
00112         matrix m30(30);
00113         m30 += 5.5; // Testowanie operatora +=
00114         std::cout « "Test operatora +=: \n" « m30 « std::endl;
00115         // Działa poprawnie
00116         matrix m31(30);
00117         matrix m32(30);
00118         if (m31 == m32) // Testowanie operatora ==
00119         {
00120             std::cout « "Macierze są równe" « std::endl;
00121         }
00122         else
00123         {
00124             std::cout « "Macierze nie są równe" « std::endl;
00125         }
00126         // Działa poprawnie
00127         matrix m33(30);
00128         matrix m34(30);
00129         if (m33 > m34) // Testowanie operatora >
00130         {
00131             std::cout « "Macierz 1 jest większa od macierzy 2" « std::endl;
00132         }
00133         else
00134         {
00135             std::cout « "Macierz 1 nie jest większa od macierzy 2" « std::endl;
00136         }
00137         // Działa poprawnie
00138         matrix m35(30);
00139         matrix m36(30);
00140         if (m35 < m36) // Testowanie operatora <
00141         {
00142             std::cout « "Macierz 1 jest mniejsza od macierzy 2" « std::endl;
00143         }
00144         else
00145         {
00146             std::cout « "Macierz 1 nie jest mniejsza od macierzy 2" « std::endl;
00147         }
00148         // Działa poprawnie
00149         matrix m37(m6);
00150         m37++; // Testowanie operatora ++
00151         std::cout « "Test operatora ++: \n" « m37 « std::endl;
00152         // Działa poprawnie
00153         matrix m38(m6);
00154         m38--; // Testowanie operatora --
00155         std::cout « "Test operatora --: \n" « m38 « std::endl;
00156         // Działa poprawnie
00157         std::cout « std::endl « m10«std::endl;// Testowanie operatora«
00158         // Działa poprawnie
00159
00160
00161
00162 }
00163
00164
```

4.3 Dokumentacja pliku Matrix/Matrix.cpp

```
#include "Matrix.h"
```

Funkcje

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

4.3.1 Dokumentacja funkcji

4.3.1.1 operator∗()

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

Metoda operator∗ mno¿y macierz przez liczbê przyjmuje jeden argument typu double, który jest mno¿ony przez macierz

Definicja w linii 391 pliku Matrix.cpp.

4.3.1.2 operator+()

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

Metoda operator+ dodaje liczbê do macierzy przyjmuje jeden argument typu double, który jest dodawany do macierzy

Definicja w linii 378 pliku Matrix.cpp.

4.3.1.3 operator-()

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

Metoda operator- odejmuje liczbê od macierzy przyjmuje jeden argument typu double, który jest odejmowany od macierzy

Definicja w linii 403 pliku Matrix.cpp.

4.3.1.4 operator<<()

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

Metoda operator<< wypisanie macierzy przyjmuje dwa argumenty: std::ostream& o - strumieñ wyjciowy, matrix& m - macierz

Definicja w linii 485 pliku Matrix.cpp.

4.4 Matrix.cpp

Idź do dokumentacji tego pliku.

```
00001 #include "Matrix.h" //<-- plik nagłówkowy
00002
00003 matrix::matrix(void)  //<-- konstruktor domyślny bez alokacji pamięci
00004 {
00005     n = 0;  //<-- rozmiar macierzy
00006     tab = nullptr;  //<-- wskaźnik do macierzy
00007 }
00008
00009 matrix::matrix(int n)  //<-- konstruktor przeciążeniowy alokuje macierz o wymiarach n na n
00010 {
00011     this->n = n; //<-- rozmiar macierzy
00012     tab = new int* [n];  //<-- wskaźnik do macierzy
00013     for (int i = 0; i < n; i++) //<-- pętla przechodząca przez elementy macierzy
00014     {
00015         tab[i] = new int[n];  //<-- przypisywanie elementu macierzy do elementu macierzy
00016         for (int j = 0; j < n; j++) //<-- pętla przechodząca przez elementy macierzy
00017         {
00018             tab[i][j] = 0; //<-- inicjalizacja elementu macierzy wartością 0
00019         }
00020     }
00021 }
00022 matrix::matrix(int n, int* t)  //<-- konstruktor przeci��eniowy alokuje pami�� i przepisuje dane z
      tabeli
00023 {
00024     this->n = n; //<-- rozmiar macierzy
00025     tab = new int* [n];  //<-- wska�nik do macierzy
00026     for (int i = 0; i < n; i++)  //<-- p�tla przechodz�ca przez elementy macierzy
00027     {
00028         tab[i] = new int[n];  //<-- nowa macierz
00029     }
00030     int k = 0; //<-- wska�nik do macierzy
00031     for (int i = 0; i < n; i++)  //<-- p�tla przechodz�ca przez elementy macierzy
00032     {
00033         for (int j = 0; j < n; j++)  //<-- p�tla przechodz�ca przez elementy macierzy
00034         {
00035             tab[i][j] = t[k];  //<-- przypisanie elementu macierzy do elementu macierzy
00036             k++;  //<-- zwiększenie wskaźnika
00037         }
00038     }
00039 }
00040 matrix::matrix(matrix& m) {
00041     this->n = m.n; //<-- przypisanie elementu macierzy do elementu macierzy
00042     if (m.tab != nullptr) { //<-- sprawdzenie czy tab jest równe nullptr
00043         tab = new int* [n]; //<-- nowa macierz
00044         for (int i = 0; i < n; i++) { //<-- p�tla przechodz�ca przez elementy macierzy
00045             tab[i] = new int[n]; //<-- nowa macierz
00046             for (int j = 0; j < n; j++) { //<-- p�tla przechodz�ca przez elementy macierzy
00047                 tab[i][j] = m.tab[i][j]; //<-- przypisanie elementu macierzy do elementu macierzy
00048             }
00049         }
00050     }
00051     else { //<-- je�li tab nie jest r�wne nullptr
00052         tab = nullptr; //<-- przypisanie elementu macierzy do elementu macierzy
00053     }
00054 }
00055
00056 matrix::~matrix(void) //<-- destruktor
00057 {
00058     if (tab) {  // <--- Sprawdzenie czy macierz ma zaalokowana pamiec
00059     for (int i = 0; i < n; ++i) {  // <--- Petla zwalniajaca pamiec
00060         delete[] tab[i]; // <--- Zwalnianie pamieci
00061     }
00062     delete[] tab; // <--- Zwalnianie pamieci
00063     tab = nullptr;  // <--- Ustawianie wskaznika na nullptr
00064     n = 0; // <--- Zerowanie rozmiaru
00065 }
00066 }
00067 matrix& matrix::alokuj(int n) //<-- je�li macierz nie ma zaalokowanej pami�ci to j� alokuje w
      wielko�ci n na n, je�li macierz ma zaalokowan pami�� to sprawdza czy rozmiar alokacji jest r�wny
      zdeklarowanemu rozmiarowi.W przypadku gdy tej pami�ci jest mniej, pami�� ma zosta zwolniona i
      zaalokowana ponownie w ��danym rozmiarze. W przypadku gdy tej pami�ci jest wi�cej pozostawi alokacj
      bez zmian.
00068 {
00069     if (tab == nullptr) //<-- sprawdzenie czy tab jest r�wne nullptr
00070     {
00071         this->n = n; //<-- przypisanie elementu macierzy do elementu macierzy
00072         tab = new int* [n]; //<-- nowa macierz
00073         for (int i = 0; i < n; i++) //<-- p�tla przechodz�ca przez elementy macierzy
00074         {
00075             tab[i] = new int[n]; //<-- nowa macierz
00076         }
00077     }
```

```
00078     else //<-- je♦li tab nie jest r♦wne nullptr
00079     {
00080         if (this->n < n)  //<-- sprawdzenie czy n jest wi♦ksze od this->n
00081         {
00082             for (int i = 0; i < this->n; i++) //<-- p♦tla przechodz♦ca przez elementy macierzy
00083             {
00084                 delete[] tab[i]; //<-- usuwanie elementu macierzy
00085             }
00086             delete[] tab; //<-- usuwanie elementu macierzy
00087             this->n = n; //<-- przypisanie elementu macierzy do elementu macierzy
00088             tab = new int* [n];     //<-- nowa macierz
00089             for (int i = 0; i < n; i++) //<-- p♦tla przechodz♦ca przez elementy macierzy
00090             {
00091                 tab[i] = new int[n]; //<-- nowa macierz
00092             }
00093         }
00094         else if (this->n > n)  //<-- sprawdzenie czy n jest mniejsze od this->n
00095         {
00096             for (int i = 0; i < n; i++) //<-- p♦tla przechodz♦ca przez elementy macierzy
00097             {
00098                 delete[] tab[i]; //<-- usuwanie elementu macierzy
00099             }
00100             delete[] tab; //<-- usuwanie elementu macierzy
00101             this->n = n; //<-- przypisanie elementu macierzy do elementu macierzy
00102             tab = new int* [n]; //<-- nowa macierz
00103             for (int i = 0; i < n; i++) //<-- p♦tla przechodz♦ca przez elementy macierzy
00104             {
00105                 tab[i] = new int[n]; //<-- nowa macierz
00106             }
00107         }
00108         else  //<-- sprawdzenie czy n jest r♦wne this->n
00109         {
00110             for (int i = 0; i < this->n; i++) //<-- p♦tla przechodz♦ca przez elementy macierzy
00111             {
00112                 delete[] tab[i]; //<-- usuwanie elementu macierzy
00113             }
00114             delete[] tab; //<-- usuwanie elementu macierzy
00115             this->n = n; //<-- przypisanie elementu macierzy do elementu macierzy
00116             tab = new int* [n]; //<-- nowa macierz
00117             for (int i = 0; i < n; i++) //<-- p♦tla przechodz♦ca przez elementy macierzy
00118             {
00119                 tab[i] = new int[n]; //<-- nowa macierz
00120             }
00121         }
00122     }
00123     return *this; //<-- zwracamy macierz
00124 }
00125 matrix& matrix::wstaw(int x, int y, int wartosc) //<-- wiersz, kolumna, warto♦♦
00126 {
00127     tab[x][y] = wartosc; //<-- przypisanie elementu macierzy do elementu macierzy
00128     return *this; //<-- zwracamy macierz
00129 }
00130 int matrix::pokaz(int x, int y) //<-- zwraca warto♦♦ elementu x, y
00131 {
00132     return tab[x][y]; //<-- zwracamy element macierzy
00133 }
00134 matrix& matrix::odwroc(void) //<-- zamienia wiersze z kolumnami
00135 {
00136     if (tab == nullptr || n <= 0) {
00137         // Jeśli tab jest niezainicjalizowany lub n jest nieprawidłowe, zwracamy macierz bez zmian
00138         return *this;
00139     }
00140
00141     int** temp = new int* [n]; //<-- nowa macierz
00142     for (int i = 0; i < n; i++) //<-- pętla przechodząca przez elementy macierzy
00143     {
00144         temp[i] = new int[n]; //<-- nowa macierz
00145     }
00146
00147     for (int i = 0; i < n; i++) //<-- pętla przechodząca przez elementy macierzy
00148     {
00149         for (int j = 0; j < n; j++) //<-- pętla przechodząca przez elementy macierzy
00150         {
00151             temp[i][j] = tab[j][i]; //<-- zamiana wierszy z kolumnami
00152         }
00153     }
00154
00155     for (int i = 0; i < n; i++) //<-- pętla przechodząca przez elementy macierzy
00156     {
00157         delete[] tab[i]; //<-- usuwanie elementu macierzy
00158     }
00159     delete[] tab; //<-- usuwanie elementu macierzy
00160
00161     tab = temp; //<-- przypisanie elementu macierzy do elementu macierzy
00162     return *this; //<-- zwracamy macierz
00163 }
00164
```

```
00165 matrix& matrix::losuj(void) //<-- wype◆niamy cyframi od 0 do 9 wszystkie elementy macierzy
00166 {
00167     srand(time(NULL)); //<-- losowanie
00168     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00169     {
00170         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00171         {
00172             tab[i][j] = rand() % 10; //<-- losowanie elementu macierzy
00173         }
00174     }
00175     return *this;
00176 }
00177 matrix& matrix::losuj(int x) //<-- wype◆niamy cyframi od 0 do 9 elementy macierzy. Zmienna x okre◆la
      ile cyfr b◆dziemy losowa◆. Nast◆pnie algorytm losuje, w kt◆re miejsca wstawi wylosowane cyfry
00178 {
00179     srand(time(NULL)); //<-- losowanie
00180     for (int i = 0; i < x; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00181     {
00182         tab[rand() % n][rand() % n] = rand() % 10; //<-- losowanie elementu macierzy
00183     }
00184     return *this; //<-- zwracamy macierz
00185 }
00186 matrix& matrix::diagonalna(int* t) //<-- po przek◆tnej s◆ wpisane dane z tabeli, a pozosta◆e elementy
      s◆ r◆wne 0
00187 {
00188     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00189     {
00190         tab[i][i] = t[i]; //<-- przypisanie elementu macierzy do elementu macierzy
00191     }
00192     return *this; //<-- zwracamy macierz
00193 }
00194 matrix& matrix::diagonalna_k(int k, int* t) //<-- po przek◆tnej s◆ wpisane dane z tabeli, a pozosta◆e
      elementy s◆ r◆wne 0. Parametr k mo◆e oznacza◆ : 0 - przek◆tna przechodz◆ca przez ◆rodek (czyli tak jak
      metoda diagonalna), cyfra dodatnia przesuwa diagonaln◆ do g◆ry macierzy o podan◆ cyfr◆, cyfra ujemna
      przesuwa diagonaln◆ w d◆ o podan◆ cyfr◆
00195 {
00196     if (k == 0) //<-- sprawdzenie czy k jest r◆wne 0
00197     {
00198         for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00199         {
00200             tab[i][i] = t[i]; //<-- przypisanie elementu macierzy do elementu macierzy
00201         }
00202     }
00203     else if (k > 0) //<-- sprawdzenie czy k jest wi◆ksze od 0
00204     {
00205         for (int i = 0; i < n - k; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00206         {
00207             tab[i][i + k] = t[i]; //<-- przypisanie elementu macierzy do elementu macierzy
00208         }
00209     }
00210     else //<-- sprawdzenie czy k jest mniejsze od 0
00211     {
00212         for (int i = 0; i < n + k; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00213         {
00214             tab[i - k][i] = t[i]; //<-- przypisanie elementu macierzy do elementu macierzy
00215         }
00216     }
00217     return *this; //<-- zwracamy macierz
00218 }
00219 matrix& matrix::kolumna(int x, int* t) //<-- przepisuje dane z tabeli do kolumny, kt◆r◆ wskazuje
      zmienna x
00220 {
00221     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00222     {
00223         tab[i][x] = t[i]; //<-- przypisanie elementu macierzy do elementu macierzy
00224     }
00225     return *this;   //<-- zwracamy macierz
00226 }
00227 matrix& matrix::wiersz(int y, int* t) //<-- przepisuje dane z tabeli do wiersza, kt◆ry wskazuje
      zmienna x
00228 {
00229     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00230     {
00231         tab[y][i] = t[i]; //<-- przypisanie elementu macierzy do elementu macierzy
00232     }
00233     return *this; //<-- zwracamy macierz
00234 }
00235 matrix& matrix::przekatna(void) //<-- uzupełnia macierz: 1-na przekątnej, 0-poza przekątną,
00236 {
00237     for (int i = 0; i < n; i++) //<-- pętla przechodząca przez elementy macierzy
00238     {
00239         tab[i][i] = 1; //<-- przypisanie 1 do elementu macierzy
00240     }
00241     return *this; //<-- zwracamy macierz
00242 }
00243 matrix& matrix::pod_przekatna(void) //<-- uzupełnia macierz: 1-pod przekątną, 0-nad przekątną i po
      przekątnej,
```

```
00244 {
00245     for (int i = 0; i < n; i++) //<-- pętla przechodząca przez elementy macierzy
00246     {
00247         for (int j = 0; j < n; j++) //<-- pętla przechodząca przez elementy macierzy
00248         {
00249             if (i > j) //<-- sprawdzenie czy i jest większe od j
00250             {
00251                 tab[i][j] = 1; //<-- przypisanie 1 do elementu macierzy
00252             }
00253             else //<-- jeśli i nie jest większe od j
00254             {
00255                 tab[i][j] = 0; //<-- przypisanie 0 do elementu macierzy
00256             }
00257         }
00258     }
00259     return *this; //<-- zwracamy macierz
00260 }
00261 matrix& matrix::nad_przekatna(void) //<-- uzupełnia macierz: 1-nad przekątną, 0-pod przekątną i po
    przekątnej,
00262 {
00263     for (int i = 0; i < n; i++) //<-- pętla przechodząca przez elementy macierzy
00264     {
00265         for (int j = 0; j < n; j++) //<-- pętla przechodząca przez elementy macierzy
00266         {
00267             if (i < j) //<-- sprawdzenie czy i jest mniejsze od j
00268             {
00269                 tab[i][j] = 1; //<-- przypisanie 1 do elementu macierzy
00270             }
00271             else //<-- jeśli i nie jest mniejsze od j
00272             {
00273                 tab[i][j] = 0; //<-- przypisanie 0 do elementu macierzy
00274             }
00275         }
00276     }
00277     return *this; //<-- zwracamy macierz
00278 }
00279 matrix& matrix::szachownica(void) //<-- uzupelnia macierz w ten spos\xb b dla n=4: 0101 1010 0101 1010
00280 {
00281     for (int i = 0; i < n; i++) //<-- p\xbtla przechodz\xbca przez elementy macierzy
00282     {
00283         if (i % 2 == 0) // <-- sprawdzenie czy i jest parzyste
00284         {
00285             for (int j = 0; j < n; j++) //<-- p\xbtla przechodz\xbca przez elementy macierzy
00286             {
00287                 if (j % 2 == 0) //<-- sprawdzenie czy j jest parzyste
00288                 {
00289                     tab[i][j] = 0; //<-- przypisanie 0 do elementu macierzy
00290                 }
00291                 else //<-- je\xbli j nie jest parzyste
00292                 {
00293                     tab[i][j] = 1; //<-- przypisanie 1 do elementu macierzy
00294                 }
00295             }
00296         }
00297         else //<-- je\xbli i nie jest parzyste
00298         {
00299             for (int j = 0; j < n; j++) //<-- p\xbtla przechodz\xbca przez elementy macierzy
00300             {
00301                 if (j % 2 == 0) //<-- sprawdzenie czy j jest parzyste
00302                 {
00303                     tab[i][j] = 1; //<-- przypisanie 1 do elementu macierzy
00304                 }
00305                 else //<-- je\xbli j nie jest parzyste
00306                 {
00307                     tab[i][j] = 0; //<-- przypisanie 0 do elementu macierzy
00308                 }
00309             }
00310         }
00311     }
00312     return *this; //<-- zwracamy macierz
00313 }
00314 matrix& matrix::operator+(matrix& m) //<-- dodawanie macierzy do macierzy
00315 {
00316     matrix* temp = new matrix(n); //<-- nowa macierz
00317     for (int i = 0; i < n; i++) //<-- p\xbtla przechodz\xbca przez elementy macierzy
00318     {
00319         for (int j = 0; j < n; j++) //<-- p\xbtla przechodz\xbca przez elementy macierzy
00320         {
00321             temp->tab[i][j] = tab[i][j] + m.tab[i][j]; //<-- dodanie elementu macierzy do elementu
    macierzy
00322         }
00323     }
00324     return *temp; //<-- zwracamy macierz
00325 }
00326 matrix& matrix::operator*(matrix& m) //<-- mno\xbenie macierzy przez macierz
00327 {
00328     matrix* temp = new matrix(n); //<-- nowa macierz
```

```
00329     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00330     {
00331         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00332         {
00333             temp->tab[i][j] = 0; //<-- zerowanie elementu macierzy
00334             for (int k = 0; k < n; k++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00335             {
00336                 temp->tab[i][j] += tab[i][k] * m.tab[k][j]; //<-- mno◆enie elementu macierzy przez
    element macierzy
00337             }
00338         }
00339     }
00340     return *temp; //<-- zwracamy macierz
00341 }
00342 matrix& matrix::operator+(int a) //<-- dodawanie liczby do macierzy
00343 {
00344     matrix* temp = new matrix(n); //<-- nowa macierz
00345     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00346     {
00347         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00348         {
00349             temp->tab[i][j] = tab[i][j] + a; //<-- dodanie elementu macierzy do a
00350         }
00351     }
00352     return *temp; //<-- zwracamy macierz
00353 }
00354 matrix& matrix::operator*(int a) //<-- mno◆enie macierzy przez liczb◆
00355 {
00356     matrix* temp = new matrix(n); //<-- nowa macierz
00357     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00358     {
00359         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00360         {
00361             temp->tab[i][j] = tab[i][j] * a; //<-- mno◆enie elementu macierzy przez a
00362         }
00363     }
00364     return *temp; //<-- zwracamy macierz
00365 }
00366 matrix& matrix::operator-(int a) //<-- odejmowanie liczby od macierzy
00367 {
00368     matrix* temp = new matrix(n); //<-- nowa macierz
00369     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00370     {
00371         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00372         {
00373             temp->tab[i][j] = tab[i][j] - a; //<-- odejmowanie elementu macierzy od a
00374         }
00375     }
00376     return *temp; //<-- zwracamy macierz
00377 }
00378 matrix operator+(int a, matrix& m) //<-- dodawanie liczby do macierzy
00379 {
00380     matrix temp(m); //<-- nowa macierz
00381     for (int i = 0; i < m.n; i++) //<-- pętla przechodząca przez elementy macierzy
00382     {
00383         for (int j = 0; j < m.n; j++) //<-- pętla przechodząca przez elementy macierzy
00384         {
00385             temp.tab[i][j] = a + temp.tab[i][j]; //<-- dodanie elementu macierzy do a
00386         }
00387     }
00388     return temp; //<-- zwracamy macierz
00389 }
00390
00391 matrix operator*(int a, matrix& m) //<-- mno◆enie macierzy przez liczb◆
00392 {
00393     matrix temp(m); //<-- nowa macierz
00394     for (int i = 0; i < m.n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00395     {
00396         for (int j = 0; j < m.n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00397         {
00398             temp.tab[i][j] =a* temp.tab[i][j]; //<-- mno◆enie elementu macierzy przez a
00399         }
00400     }
00401     return temp; //<-- zwracamy macierz
00402 }
00403 matrix operator-(int a, matrix& m) //<-- odejmowanie macierzy od liczby
00404 {
00405     matrix temp(m); //<-- nowa macierz
00406     for (int i = 0; i < m.n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00407     {
00408         for (int j = 0; j < m.n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00409         {
00410             temp.tab[i][j] =a- temp.tab[i][j] ; //<-- odejmowanie elementu macierzy od a
00411         }
00412     }
00413     return temp; //<-- zwracamy macierz
00414 }
```

```
00415 matrix& matrix::operator++(int) //<-- wszystkie liczby powi◆kszone o 1
00416 {
00417     matrix* temp = new matrix(n); //<-- nowa macierz
00418     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00419     {
00420         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00421         {
00422             temp->tab[i][j] = tab[i][j] + 1; //<-- powi◆kszenie elementu macierzy o 1
00423         }
00424     }
00425     return *temp; //<-- zwracamy macierz
00426 }
00427 matrix& matrix::operator--(int) //<-- wszystkie liczby pomniejszone o 1
00428 {
00429     matrix* temp = new matrix(n); //<-- nowa macierz
00430     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00431     {
00432         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00433         {
00434             temp->tab[i][j] = tab[i][j] - 1; //<-- pomniejszenie elementu macierzy o 1
00435         }
00436     }
00437     return *temp; //<-- zwracamy macierz
00438 }
00439 matrix& matrix::operator+=(int a) //<-- ka◆dy element w macierzy powi◆kszamy o „a”
00440 {
00441     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00442     {
00443         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00444         {
00445             tab[i][j] += a; //<-- powi◆kszenie elementu macierzy o a
00446         }
00447     }
00448     return *this; //<-- zwracamy macierz
00449 }
00450 matrix& matrix::operator-=(int a) //<-- ka◆dy element w macierzy pomniejszamy o „a”
00451 {
00452     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00453     {
00454         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00455         {
00456             tab[i][j] -= a; //<-- pomniejszenie elementu macierzy o a
00457         }
00458     }
00459     return *this; //<-- zwracamy macierz
00460 }
00461 matrix& matrix::operator*=(int a) //<-- ka◆dy element w macierzy mno◆ymy o „a”
00462 {
00463     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00464     {
00465         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00466         {
00467             tab[i][j] *= a; //<-- mno◆enie elementu macierzy przez a
00468         }
00469     }
00470     return *this; //<-- zwracamy macierz
00471 }
00472 matrix& matrix::operator+=(double a) //<-- wszystkie cyfry s◆ powi◆kszone o cz◆◆ ca◆kowit◆ z wpisanej
    cyfry
00473 {
00474     int b = a; //<-- zmienna b przechowuje cz◆◆ ca◆kowit◆ z wpisanej cyfry
00475     for (int i = 0; i < n; i++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00476     {
00477         for (int j = 0; j < n; j++) //<-- p◆tla przechodz◆ca przez elementy macierzy
00478         {
00479             tab[i][j] += b; //<-- dodanie cz◆ci ca◆kowitej do elementu macierzy
00480         }
00481
00482     }
00483     return *this; //<-- zwracamy macierz
00484 }
00485 std::ostream& operator«(std::ostream& o, matrix& m) //<-- wypisanie macierzy
00486 {
00487     for (int i = 0; i < m.n; i++) //<-- p◆tla wypisuj◆ca elementy macierzy
00488     {
00489         for (int j = 0; j < m.n; j++) //<-- p◆tla wypisuj◆ca elementy macierzy
00490         {
00491             o « m.tab[i][j] « " "; //<-- wypisanie elementu macierzy
00492         }
00493         o « std::endl; //<-- przej◆cie do nowej linii
00494     }
00495     return o; //<-- zwracamy strumie◆ wyj◆ciowy
00496 }
00497 bool matrix::operator==(const matrix& m) //<-- sprawdza, czy ka◆dy element macierzy spe◆nia r◆wno◆◆
    ??(??, ??) = ??(??, ??) A = | 1 2 | B = | 1 2| |3 4| |3 4 | je◆li nie, to nie mo◆emy m◆wi◆, ◆e
    macierze s◆ r◆wne
00498 {
```

```
00499     if (n != m.n)  //<-- sprawdzenie czy macierze maj♦ taki sam rozmiar
00500     {
00501         return false; //<-- je♦li nie to zwracamy false
00502     }
00503     for (int i = 0; i < n; i++) //<-- p♦tla sprawdzaj♦ca elementy macierzy
00504     {
00505         for (int j = 0; j < n; j++) //<-- p♦tla sprawdzaj♦ca elementy macierzy
00506         {
00507             if (tab[i][j] != m.tab[i][j]) //<-- sprawdzenie czy elementy s♦ r♦wne
00508             {
00509                 return false; //<-- je♦li nie to zwracamy false
00510             }
00511         }
00512     }
00513     return true; //<-- je♦li wszystkie elementy s♦ r♦wne to zwracamy true
00514 }
00515 bool matrix::operator>(const matrix& m) //<-- operator wi♦kszo♦ci sprawdza, czy ka♦dy element macierzy
    spe♦nia nier♦wno♦♦ ??(??, ??) > ??(??, ??).Je♦li tak, to mo♦emy powiedzie♦, ♦e macierz jest wi♦ksza, w
    przeciwnym wypadku nie mo♦emy stwierdzi♦,♦e macierz jest wi♦ksza.
00516 {
00517     if (n != m.n)  //<-- sprawdzenie czy macierze maj♦ taki sam rozmiar
00518     {
00519         return false; //<-- je♦li nie to zwracamy false
00520     }
00521     for (int i = 0; i < n; i++) //<-- p♦tla sprawdzaj♦ca elementy macierzy
00522     {
00523         for (int j = 0; j < n; j++)  //<-- p♦tla sprawdzaj♦ca elementy macierzy
00524         {
00525             if (tab[i][j] <= m.tab[i][j]) //<-- sprawdzenie czy elementy s♦ wi♦ksze od siebie
00526             {
00527                 return false;  //<-- je♦li nie to zwracamy false
00528             }
00529         }
00530     }
00531     return true;  //<-- je♦li wszystkie elementy s♦ wi♦ksze to zwracamy true
00532 }
00533 bool matrix::operator<(const matrix& m)  //<-- tak jak wy♦ej tylko operator mniejszo♦ci. Na marginesie
    macierzy mo♦emy nie da♦ rady okre♦li♦, ♦e jest r♦wna, mniejsza i wi♦ksza, wtedy m♦wimy ♦e jest r♦na
00534 {
00535     if (n != m.n)  //<-- sprawdzenie czy macierze maj♦ taki sam rozmiar
00536     {
00537         return false;  //<-- je♦li nie to zwracamy false
00538     }
00539     for (int i = 0; i < n; i++)  //<-- p♦tla sprawdzaj♦ca elementy macierzy
00540     {
00541         for (int j = 0; j < n; j++) //<-- p♦tla sprawdzaj♦ca elementy macierzy
00542         {
00543             if (tab[i][j] >= m.tab[i][j]) //<-- sprawdzenie czy elementy s♦ mniejsze od siebie
00544             {
00545                 return false;  //<-- je♦li nie to zwracamy false
00546             }
00547         }
00548     }
00549     return true; //<-- je♦li wszystkie elementy s♦ mniejsze to zwracamy true
00550 }
00551 matrix& matrix::wczytaj(const std::string& nazwa)  //<-- wczytuje macierz z pliku o nazwie „nazwa”
00552 {
00553     std::ifstream plik(nazwa);  //<-- otwarcie pliku
00554     if (!plik.is_open())  //<-- sprawdzenie czy plik zosta♦ otwarty
00555     {
00556         std::cout « "Nie udalo sie otworzyc pliku" « std::endl; //<-- komunikat o b♦♦dzie
00557         return *this;  //<-- zwr♦cenie macierzy
00558     }
00559     int n;  //<-- rozmiar macierzy
00560     plik » n; //<-- wczytanie rozmiaru macierzy
00561     this->alokuj(n);  //<-- alokacja pami♦ci
00562     if (n <= 0)  //<-- sprawdzenie czy rozmiar macierzy jest wi♦kszy od zera
00563     {
00564         std::cout « "Niepoprawny rozmiar macierzy" « std::endl; //<-- komunikat o b♦♦dzie
00565         plik.close();  //<-- zamkni♦cie pliku
00566         return *this;  //<-- zwr♦cenie macierzy
00567     }
00568     for (int i = 0; i < n; i++)   // <--- Petla wczytujaca wartosci do macierzy
00569     {
00570         for (int j = 0; j < n; j++) // <--- Petla wczytujaca wartosci do macierzy
00571         {
00572             plik » tab[i][j]; // <--- Wczytanie wartosci do macierzye
00573         }
00574     }
00575
00576     plik.close(); //<-- zamkni♦cie pliku
00577     return *this; //<-- zwr♦cenie macierzy
00578 }
```

4.5 Dokumentacja pliku Matrix/Matrix.h

```
#include <string>
#include <iostream>
#include <fstream>
```

Komponenty

- class matrix

  Klasa reprezentujaca macierz kwadratowa.

4.6 Matrix.h

IdÅº do dokumentacji tego pliku.

```
00001 #pragma once  //<-- zabezpieczenie przed wielokrotnym do³¹czaniem pliku nag³ówkowego
00002 #include <string>  //<-- biblioteka do obs³ugi stringów
00003 #include <iostream>     //<-- biblioteka do obs³ugi strumieni wejcia/wyjcia
00004 #include <fstream> //<-- biblioteka do obs³ugi plików
00014 class matrix
00015 {
00016 private:
00017     int n; //<-- rozmiar macierzy
00018     int** tab; //<-- wskanik do macierzy
00019 public:
00025     matrix(void);
00030     matrix(int n);
00035     matrix(int n, int* t);
00040     matrix(matrix& m);
00041     ~matrix(void); //<--destruktor,
00047     matrix& alokuj(int n);
00052     matrix& wstaw(int x, int y, int wartosc);
00057     int pokaz(int x, int y);
00062     matrix& odwroc(void);
00067     matrix& losuj(void);
00072     matrix& losuj(int x);
00077     matrix& diagonalna(int* t);
00082     matrix& diagonalna_k(int k, int* t);
00087     matrix& kolumna(int x, int* t);
00092     matrix& wiersz(int y, int* t);
00097     matrix& przekatna(void);
00102     matrix& pod_przekatna(void);
00107     matrix& nad_przekatna(void);
00112     matrix& szachownica(void);
00117     matrix & operator+(matrix & m);
00122     matrix& operator*(matrix& m);
00127     matrix& operator+(int a);
00132     matrix& operator*(int a);
00137     matrix& operator-(int a);
00142     friend matrix operator+(int a, matrix& m);
00147     friend matrix operator*(int a, matrix& m);
00152     friend matrix operator-(int a, matrix& m);
00157     matrix& operator++(int);
00162     matrix& operator--(int);
00167     matrix& operator+=(int a);
00172     matrix& operator-=(int a);
00177     matrix& operator*=(int a);
00182     matrix& operator+=(double a);
00187     friend std::ostream& operator«(std::ostream& o, matrix& m);
00192     bool operator==(const matrix& m);
00197     bool operator>(const matrix& m);
00202     bool operator<(const matrix& m);
00207     matrix& wczytaj(const std::string& nazwa);
00208
00209 };
00210
```

Skorowidz