proj4

Generated by Doxygen 1.12.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Matrix< T > Class Template Reference

Klasa reprezentująca macierz.

`#include <Matrix.hpp>`

Public Member Functions

- Matrix (void)
  Konstruktor domyślny.
- Matrix (int n)
  Konstruktor tworzący macierz o rozmiarze n x n.
- Matrix (int n, int *t)
  Konstruktor tworzący macierz o rozmiarze n x n i wypełniający ją danymi z tablicy.
- Matrix (Matrix const &m)
  Konstruktor kopiujący.
- void wypisz (void)
  Wypisuje zawartość macierzy.
- Matrix & alokuj (int n)
  Alokuje pamięć dla macierzy o rozmiarze n x n.
- int pokaz (int x, int y) const
  Zwraca wartość elementu macierzy na pozycji (x, y).
- Matrix & dowroc (void)
  Odwraca macierz (transpozycja).
- Matrix & losuj (void)
  Wypełnia macierz losowymi wartościami od 0 do 9.
- Matrix & losuj (int x)
  Wypełnia x losowych elementów macierzy wartościami od 0 do 9.
- Matrix & diagonalna (const int *t)
  Wypełnia przekątną macierzy wartościami z tablicy t, pozostałe elementy są równe 0.
- Matrix & diagonalna_k (int k, const int *t)
  Wypełnia przekątną macierzy wartościami z tablicy t, przesuniętą o k pozycji.
- Matrix & kolumna (int x, const int *t)
  Wypełnia kolumnę x wartościami z tablicy t.

- Matrix & wiersz (int x, const int *t)

  Wypełnia wiersz x wartościami z tablicy t.
- Matrix & przekatna (void)

  Wypełnia macierz: 1 na przekątnej, 0 poza przekątną.
- Matrix & pod_przekatna (void)

  Wypełnia macierz: 1 pod przekątną, 0 nad przekątną i na przekątnej.
- Matrix & nad_przekatna (void)

  Wypełnia macierz: 1 nad przekątną, 0 pod przekątną i na przekątnej.
- Matrix & wstaw (int x, int y, int wartosc)

  Wstawia wartość do macierzy na podaną pozycję.
- Matrix & szachownica (void)

  Tworzy wzór szachownicy w macierzy.
- Matrix & operator+ (Matrix const &m)

  Dodaje do macierzy inną macierz element po elemencie.
- Matrix & operator* (Matrix const &m)

  Mnoży macierz przez inną macierz.
- Matrix & operator+ (int a)

  Dodaje do każdego elementu macierzy liczbę całkowitą.
- Matrix & operator* (int a)

  Mnoży każdy element macierzy przez liczbę całkowitą.
- Matrix & operator- (int a)

  Odejmuje od każdego elementu macierzy liczbę całkowitą.
- Matrix & operator++ (int)

  Operator postinkrementacji - zwiększa każdy element o 1.
- Matrix & operator-- (int)

  Operator postdekrementacji - zmniejsza każdy element o 1.
- Matrix & operator+= (int a)

  Dodaje liczbę całkowitą do każdego elementu macierzy (A += a).
- Matrix & operator-= (int a)

  Odejmuje od każdego elementu macierzy liczbę całkowitą (A -= a).
- Matrix & operator*= (int a)

  *Mnoży każdy element macierzy przez liczbę całkowitą (A *= a).*
- Matrix & operator() (double a)

  Dodaje do każdego elementu macierzy część całkowitą z liczby a.
- bool operator== (const Matrix &m) const

  Sprawdza, czy macierz jest równa innej macierzy.
- bool operator> (const Matrix &m) const

  Sprawdza, czy każda wartość bieżącej macierzy jest większa niż w macierzy m.
- bool operator< (const Matrix &m) const

  Sprawdza, czy każda wartość bieżącej macierzy jest mniejsza niż w macierzy m.

Friends

- Matrix operator+ (int a, Matrix const &m)

  Dodaje liczbę całkowitą do każdego elementu macierzy (postać a + m).
- Matrix operator* (int a, Matrix const &m)

  *Mnoży każdy element macierzy przez liczbę całkowitą (postać a * m).*
- Matrix operator- (int a, Matrix const &m)

  Odejmuje każdy element macierzy od liczby a (postać a - m).
- std::ostream & operator<< (std::ostream &o, Matrix const &m)

  Wypisuje macierz do strumienia wyjściowego.

3.1.1 Detailed Description

template<typename T>
class Matrix< T >

Klasa reprezentująca macierz.

Template Parameters

T	Typ danych przechowywanych w macierzy.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Matrix() [1/4]

```
template<typename T >
Matrix< T >::Matrix (
            void )  [inline]
```

Konstruktor domyślny.

3.1.2.2 Matrix() [2/4]

```
template<typename T >
Matrix< T >::Matrix (
            int n)  [inline]
```

Konstruktor tworzący macierz o rozmiarze n x n.

Parameters

n	Rozmiar macierzy.

3.1.2.3 Matrix() [3/4]

```
template<typename T >
Matrix< T >::Matrix (
            int n,
            int * t)  [inline]
```

Konstruktor tworzący macierz o rozmiarze n x n i wypełniający ją danymi z tablicy.

Parameters

n	Rozmiar macierzy.
t	Wskaźnik do tablicy z danymi.

3.1.2.4 Matrix() [4/4]

```
template<typename T >
Matrix< T >::Matrix (
            Matrix< T > const & m)  [inline]
```

Konstruktor kopiujący.

Parameters

m	Obiekt macierzy do skopiowania.

3.1.3 Member Function Documentation

3.1.3.1 alokuj()

```
template<typename T >
Matrix & Matrix< T >::alokuj (
            int n)  [inline]
```

Alokuje pamięć dla macierzy o rozmiarze n x n.

Parameters

n	Rozmiar macierzy.

Returns

Referencja do obiektu macierzy.

3.1.3.2 diagonalna()

```
template<typename T >
Matrix & Matrix< T >::diagonalna (
            const int * t)  [inline]
```

Wypełnia przekątną macierzy wartościami z tablicy t, pozostałe elementy są równe 0.

Parameters

t	Wskaźnik do tablicy z wartościami.

Returns

Referencja do obiektu macierzy.

3.1.3.3 diagonalna_k()

```
template<typename T >
Matrix & Matrix< T >::diagonalna_k (
            int k,
            const int * t)  [inline]
```

Wypełnia przekątną macierzy wartościami z tablicy t, przesuniętą o k pozycji.

Parameters

k	Przesunięcie przekątnej.
t	Wskaźnik do tablicy z wartościami.

Returns

Referencja do obiektu macierzy.

3.1.3.4 dowroc()

```
template<typename T >
Matrix & Matrix< T >::dowroc (
            void )  [inline]
```

Odwraca macierz (transpozycja).

Returns

Referencja do obiektu macierzy.

3.1.3.5 kolumna()

```
template<typename T >
Matrix & Matrix< T >::kolumna (
            int x,
            const int * t)  [inline]
```

Wypełnia kolumnę x wartościami z tablicy t.

Parameters

x	Numer kolumny.
t	Wskaźnik do tablicy z wartościami.

Returns

Referencja do obiektu macierzy.

3.1.3.6 losuj() [1/2]

```
template<typename T >
Matrix & Matrix< T >::losuj (
            int x)  [inline]
```

Wypełnia x losowych elementów macierzy wartościami od 0 do 9.

Parameters

x	Liczba elementów do wypełnienia.

Returns

Referencja do obiektu macierzy.

3.1.3.7 losuj() [2/2]

```
template<typename T >
Matrix & Matrix< T >::losuj (
            void )  [inline]
```

Wypełnia macierz losowymi wartościami od 0 do 9.

Returns

Referencja do obiektu macierzy.

3.1.3.8 nad_przekatna()

```
template<typename T >
Matrix & Matrix< T >::nad_przekatna (
            void )  [inline]
```

Wypełnia macierz: 1 nad przekątną, 0 pod przekątną i na przekątnej.

Returns

Referencja do obiektu macierzy.

3.1.3.9 operator()()

```
template<typename T >
Matrix & Matrix< T >::operator() (
            double a)  [inline]
```

Dodaje do każdego elementu macierzy część całkowitą z liczby a.

Parameters

a	Liczba typu double, bierzemy floor(a) i dodajemy do każdego elementu.

Returns

Zwraca referencję do bieżącej macierzy (this).

Dla przykładu, jeśli a = 3.7 to floor(a) = 3 i dodajemy 3 do każdego elementu.

3.1.3.10 operator*() [1/2]

```
template<typename T >
Matrix & Matrix< T >::operator* (
            int a)  [inline]
```

Mnoży każdy element macierzy przez liczbę całkowitą.

Parameters

a	Liczba, przez którą mnożymy elementy macierzy.

Returns

Zwraca referencję do bieżącej macierzy (this).

3.1.3.11 operator∗() [2/2]

```
template<typename T >
Matrix & Matrix< T >::operator* (
            Matrix< T > const & m)  [inline]
```

Mnoży macierz przez inną macierz.

Parameters

m	Macierz, przez którą mnożymy.

Returns

Zwraca referencję do bieżącej macierzy (this).

Wykonuje klasyczne mnożenie macierzy kwadratowych. Jeśli rozmiary się nie zgadzają, nic nie zmienia. Wynikiem jest A = A ∗ m.

3.1.3.12 operator∗=()

```
template<typename T >
Matrix & Matrix< T >::operator*= (
            int a)  [inline]
```

Mnoży każdy element macierzy przez liczbę całkowitą (A ∗= a).

Parameters

a	Liczba przez którą mnożymy.

Returns

Zwraca referencję do bieżącej macierzy (this).

3.1.3.13 operator+() [1/2]

```
template<typename T >
Matrix & Matrix< T >::operator+ (
            int a)  [inline]
```

Dodaje do każdego elementu macierzy liczbę całkowitą.

Parameters

a	Liczba, którą dodajemy do elementów macierzy.

Returns

Zwraca referencję do bieżącej macierzy (this).

3.1.3.14 operator+() [2/2]

```
template<typename T >
Matrix & Matrix< T >::operator+ (
            Matrix< T > const & m)  [inline]
```

Dodaje do macierzy inną macierz element po elemencie.

Parameters

m	Druga macierz, którą dodajemy.

Returns

Zwraca referencję do bieżącej macierzy (this).

Dodaje wszystkie elementy macierzy m do bieżącej macierzy A. Jeśli rozmiary się nie zgadzają, nic nie robi.

3.1.3.15 operator++()

```
template<typename T >
Matrix & Matrix< T >::operator++ (
            int )  [inline]
```

Operator postinkrementacji - zwiększa każdy element o 1.

Returns

Zwraca referencję do bieżącej macierzy (this).

Po wykonaniu A++ każdy element w A jest większy o 1.

3.1.3.16 operator+=()

```
template<typename T >
Matrix & Matrix< T >::operator+= (
            int a)  [inline]
```

Dodaje liczbę całkowitą do każdego elementu macierzy (A += a).

Parameters

a	Liczba do dodania.

Returns

Zwraca referencję do bieżącej macierzy (this).

3.1.3.17 operator-()

```
template<typename T >
Matrix & Matrix< T >::operator- (
            int a)  [inline]
```

Odejmuje od każdego elementu macierzy liczbę całkowitą.

Parameters

a	Liczba, którą odejmujemy od elementów macierzy.

Returns

Zwraca referencję do bieżącej macierzy (this).

3.1.3.18 operator--()

```
template<typename T >
Matrix & Matrix< T >::operator-- (
            int )  [inline]
```

Operator postdekrementacji - zmniejsza każdy element o 1.

Returns

Zwraca referencję do bieżącej macierzy (this).

Po wykonaniu A– każdy element w A jest mniejszy o 1.

3.1.3.19 operator-=()

```
template<typename T >
Matrix & Matrix< T >::operator-= (
            int a)  [inline]
```

Odejmuje od każdego elementu macierzy liczbę całkowitą (A -= a).

Parameters

a	Liczba do odjęcia.

Returns

Zwraca referencję do bieżącej macierzy (this).

3.1.3.20 operator<()

```
template<typename T >
bool Matrix< T >::operator< (
            const Matrix< T > & m) const  [inline]
```

Sprawdza, czy każda wartość bieżącej macierzy jest mniejsza niż w macierzy m.

Parameters

m	Macierz do porównania.

Returns

true jeśli wszystkie elementy A są mniejsze od odpowiadających elementów m, w przeciwnym razie false.

3.1.3.21 operator==()

```
template<typename T >
bool Matrix< T >::operator== (
            const Matrix< T > & m) const  [inline]
```

Sprawdza, czy macierz jest równa innej macierzy.

Parameters

m	Macierz do porównania.

Returns

true jeśli są równe, false w przeciwnym razie.

Porównuje elementy o tych samych indeksach.

3.1.3.22 operator>()

```
template<typename T >
bool Matrix< T >::operator> (
            const Matrix< T > & m) const  [inline]
```

Sprawdza, czy każda wartość bieżącej macierzy jest większa niż w macierzy m.

Parameters

m	Macierz do porównania.

Returns

true jeśli wszystkie elementy A są większe od odpowiadających elementów m, w przeciwnym razie false.

3.1.3.23 pod_przekatna()

```
template<typename T >
Matrix & Matrix< T >::pod_przekatna (
            void )  [inline]
```

Wypełnia macierz: 1 pod przekątną, 0 nad przekątną i na przekątnej.

Returns

Referencja do obiektu macierzy.

3.1.3.24 pokaz()

```
template<typename T >
int Matrix< T >::pokaz (
            int x,
            int y) const  [inline]
```

Zwraca wartość elementu macierzy na pozycji (x, y).

Parameters

x	Wiersz.
y	Kolumna.

Returns

Wartość elementu macierzy.

3.1.3.25 przekatna()

```
template<typename T >
Matrix & Matrix< T >::przekatna (
            void )  [inline]
```

Wypełnia macierz: 1 na przekątnej, 0 poza przekątną.

Returns

Referencja do obiektu macierzy.

3.1.3.26 szachownica()

```
template<typename T >
Matrix & Matrix< T >::szachownica (
            void )  [inline]
```

Tworzy wzór szachownicy w macierzy.

Returns

Zwraca referencję do bieżącej macierzy (this).

Metoda ustawia w macierzy wzór podobny do szachownicy, gdzie pola naprzemiennie mają wartość 0 lub 1.

3.1.3.27 wiersz()

```
template<typename T >
Matrix & Matrix< T >::wiersz (
            int x,
            const int * t)  [inline]
```

Wypełnia wiersz x wartościami z tablicy t.

Parameters

x	Numer wiersza.
t	Wskaźnik do tablicy z wartościami.

Returns

Referencja do obiektu macierzy.

3.1.3.28 wstaw()

```
template<typename T >
Matrix & Matrix< T >::wstaw (
            int x,
            int y,
            int wartosc)  [inline]
```

Wstawia wartość do macierzy na podaną pozycję.

Parameters

x	Indeks wiersza.
y	Indeks kolumny.
wartosc	Wartość, którą wstawiamy do macierzy.

Returns

Zwraca referencję do bieżącej macierzy (this).

Metoda wstawia liczbę do macierzy w odpowiednie miejsce. Dzięki temu można zmienić zawartość macierzy w dowolnym miejscu.

3.1.3.29 wypisz()

```
template<typename T >
void Matrix< T >::wypisz (
            void )  [inline]
```

Wypisuje zawartość macierzy.

3.1.4 Friends And Related Symbol Documentation

3.1.4.1 operator*

```
template<typename T >
Matrix operator* (
            int a,
            Matrix< T > const & m)  [friend]
```

Mnoży każdy element macierzy przez liczbę całkowitą (postać a * m).

Parameters

a	Liczba przez którą mnożymy.
m	Macierz, której elementy mnożymy.

Returns

Nowa macierz po pomnożeniu.

3.1.4.2 operator+

```
template<typename T >
Matrix operator+ (
            int a,
            Matrix< T > const & m)  [friend]
```

Dodaje liczbę całkowitą do każdego elementu macierzy (postać a + m).

Parameters

a	Liczba do dodania.
m	Macierz, do której dodajemy liczbę.

Returns

Nowa macierz z dodaną liczbą a.

3.1.4.3 operator-

```
template<typename T >
Matrix operator- (
            int a,
            Matrix< T > const & m)  [friend]
```

Odejmuje każdy element macierzy od liczby a (postać a - m).

Parameters

a	Liczba od której odejmujemy.
m	Macierz, której elementy odejmujemy.

Returns

Nowa macierz po odjęciu.

3.1.4.4 operator<<

```
template<typename T >
std::ostream & operator<< (
            std::ostream & o,
            Matrix< T > const & m)  [friend]
```

Wypisuje macierz do strumienia wyjściowego.

Parameters

o	Strumień wyjściowy.
m	Macierz do wypisania.

Returns

Referencja do strumienia wyjściowego.

Wypisuje elementy macierzy w postaci wierszy. Każdy element jest oddzielony spacją.

Chapter 4

File Documentation

4.1 src/main.cpp File Reference

```
#include <iostream>
#include "Matrix.hpp"
#include <fstream>
```

Functions

- int main (int argc, char ∗argv[ ])

4.1.1 Function Documentation

4.1.1.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

4.2 src/Matrix.hpp File Reference

```
#include <random>
#include <vector>
#include <print>
#include <cmath>
#include <iostream>
```

Classes

- class Matrix< T >

  Klasa reprezentująca macierz.

4.3 Matrix.hpp

Go to the documentation of this file.

```
00001 #include <random>
00002 #include <vector>
00003 #include <print>
00004 #include <cmath>
00005 #include <iostream>
00006
00012 template <typename T>
00013 class Matrix {
00014 private:
00015     std::vector<std::vector<T» data;
00016
00017 public:
00021     Matrix(void) {}
00022
00028     Matrix(int n) : data(n, std::vector<T>(n)) {}
00029
00036     Matrix(int n, int* t) : data(n, std::vector<T>(n)) {
00037         for(auto& row : data){
00038             row = {t, t + n};
00039             t += n;
00040         }
00041     }
00042
00048     Matrix(Matrix const& m) : data(m.data) {}
00049
00053     void wypisz(void) {
00054         for(const auto& row : data){
00055             for(const auto& elem : row) {
00056                 std::print("{} ", elem);
00057             }
00058             std::println();
00059         }
00060     }
00061
00068     Matrix& alokuj(int n){
00069         if(data.size() == 0){
00070             data.resize(n, std::vector<T>(n));
00071         } else {
00072             if(data.size() < n){
00073                 data.resize(n, std::vector<T>(n));
00074             }
00075         }
00076         return *this;
00077     }
00078
00086     int pokaz(int x, int y) const {
00087         return data[x][y];
00088     }
00089
00095     Matrix& dowroc(void) {
00096         Matrix<T> temp(data.size());
00097
00098         for(int i = 0; i < data.size(); i++){
00099             for(int j = 0; j < data.size(); j++){
00100                 temp.data[i][j] = data[j][i];
00101             }
00102         }
00103
00104         *this = temp;
00105         return *this;
00106     }
00107
00113     Matrix& losuj(void) {
00114         std::random_device rd;
00115         std::mt19937 gen(rd());
00116         std::uniform_int_distribution dis(0, 9);
00117
00118         for(int i = 0; i < data.size(); i++){
00119             for(int j = 0; j < data.size(); j++){
00120                 data[i][j] = dis(gen);
00121             }
00122         }
00123         return *this;
00124     }
00125
00132     Matrix& losuj(int x) {
00133         if (data.size() == 0) return *this;
00134
00135         std::random_device rd;
00136         std::mt19937 gen(rd());
00137         std::uniform_int_distribution<> dis(0, 9);
00138
```

```
00139         int n = (int)data.size();
00140         for (int i = 0; i < x; i++) {
00141             int a = dis(gen) % n; // zabezpieczenie przed wyjściem poza zakres
00142             int b = dis(gen) % n; // j.w.
00143             data[a][b] = dis(gen);
00144         }
00145         return *this;
00146     }
00147
00154     Matrix& diagonalna(const int* t) {
00155         for(int i = 0; i < data.size(); i++){
00156             for(int j = 0; j < data.size(); j++){
00157                 if(i == j){
00158                     data[i][j] = t[i];
00159                 } else {
00160                     data[i][j] = 0;
00161                 }
00162             }
00163         }
00164         return *this;
00165     }
00166
00174     Matrix& diagonalna_k(int k, const int* t) {
00175         for(int i = 0; i < data.size(); i++){
00176             for(int j = 0; j < data.size(); j++){
00177                 if(i == j + k){
00178                     data[i][j] = t[j];
00179                 } else {
00180                     data[i][j] = 0;
00181                 }
00182             }
00183         }
00184         return *this;
00185     }
00186
00194     Matrix& kolumna(int x, const int* t) {
00195         for(int i = 0; i < data.size(); i++){
00196             data[i][x] = t[i];
00197         }
00198         return *this;
00199     }
00200
00208     Matrix& wiersz(int x, const int* t) {
00209         for(int i = 0; i < data.size(); i++){
00210             data[x][i] = t[i];
00211         }
00212         return *this;
00213     }
00214
00220     Matrix& przekatna(void) {
00221         for(int i = 0; i < data.size(); i++){
00222             for(int j = 0; j < data.size(); j++){
00223                 if(i == j){
00224                     data[i][j] = 1;
00225                 } else {
00226                     data[i][j] = 0;
00227                 }
00228             }
00229         }
00230         return *this;
00231     }
00232
00238     Matrix& pod_przekatna(void) {
00239         for(int i = 0; i < data.size(); i++){
00240             for(int j = 0; j < data.size(); j++){
00241                 if(i > j){
00242                     data[i][j] = 1;
00243                 } else {
00244                     data[i][j] = 0;
00245                 }
00246             }
00247         }
00248         return *this;
00249     }
00250
00256     Matrix& nad_przekatna(void) {
00257         for(int i = 0; i < data.size(); i++){
00258             for(int j = 0; j < data.size(); j++){
00259                 if(i < j){
00260                     data[i][j] = 1;
00261                 } else {
00262                     data[i][j] = 0;
00263                 }
00264             }
00265         }
00266         return *this;
00267     }
```

```
00268
00269     //.............................................................
00270
00281     Matrix& wstaw(int x, int y, int wartosc) {
00282         data[x][y] = wartosc;
00283         return *this;
00284     }
00285
00293     Matrix& szachownica(void) {
00294         for (int i = 0; i < (int)data.size(); i++) {
00295             for (int j = 0; j < (int)data.size(); j++) {
00296                 data[i][j] = ((i + j) % 2 == 0) ? 0 : 1;
00297             }
00298         }
00299         return *this;
00300     }
00301
00310     Matrix& operator+(Matrix const& m) {
00311         // Sprawdzenie wymiarów
00312         if (data.size() != m.data.size()) {
00313             return *this;
00314         }
00315
00316         for (int i = 0; i < (int)data.size(); i++) {
00317             for (int j = 0; j < (int)data.size(); j++) {
00318                 data[i][j] += m.data[i][j];
00319             }
00320         }
00321         return *this;
00322     }
00323
00333     Matrix& operator*(Matrix const& m) {
00334         int n = (int)data.size();
00335         if (n == 0 || n != (int)m.data.size()) {
00336             return *this;
00337         }
00338
00339         Matrix<T> result(n);
00340         for (int i = 0; i < n; i++) {
00341             for (int j = 0; j < n; j++) {
00342                 T sum = 0;
00343                 for (int k = 0; k < n; k++) {
00344                     sum += data[i][k] * m.data[k][j];
00345                 }
00346                 result.data[i][j] = sum;
00347             }
00348         }
00349         *this = result;
00350         return *this;
00351     }
00352
00358     Matrix& operator+(int a) {
00359         for (auto& row : data) {
00360             for (auto& elem : row) {
00361                 elem += a;
00362             }
00363         }
00364         return *this;
00365     }
00366
00372     Matrix& operator*(int a) {
00373         for (auto& row : data) {
00374             for (auto& elem : row) {
00375                 elem *= a;
00376             }
00377         }
00378         return *this;
00379     }
00380
00386     Matrix& operator-(int a) {
00387         for (auto& row : data) {
00388             for (auto& elem : row) {
00389                 elem -= a;
00390             }
00391         }
00392         return *this;
00393     }
00394
00401     friend Matrix operator+(int a, Matrix const& m) {
00402         Matrix result(m);
00403         for (auto& row : result.data) {
00404             for (auto& elem : row) {
00405                 elem = a + elem;
00406             }
00407         }
00408         return result;
00409     }
```

```
00410
00417     friend Matrix operator*(int a, Matrix const& m) {
00418         Matrix result(m);
00419         for (auto& row : result.data) {
00420             for (auto& elem : row) {
00421                 elem = a * elem;
00422             }
00423         }
00424         return result;
00425     }
00426
00433     friend Matrix operator-(int a, Matrix const& m) {
00434         Matrix result(m);
00435         for (auto& row : result.data) {
00436             for (auto& elem : row) {
00437                 elem = a - elem;
00438             }
00439         }
00440         return result;
00441     }
00442
00449     Matrix& operator++(int) {
00450         for (auto& row : data) {
00451             for (auto& elem : row) {
00452                 elem += 1;
00453             }
00454         }
00455         return *this;
00456     }
00457
00464     Matrix& operator--(int) {
00465         for (auto& row : data) {
00466             for (auto& elem : row) {
00467                 elem -= 1;
00468             }
00469         }
00470         return *this;
00471     }
00472
00478     Matrix& operator+=(int a) {
00479         for (auto& row : data) {
00480             for (auto& elem : row) {
00481                 elem += a;
00482             }
00483         }
00484         return *this;
00485     }
00486
00492     Matrix& operator-=(int a) {
00493         for (auto& row : data) {
00494             for (auto& elem : row) {
00495                 elem -= a;
00496             }
00497         }
00498         return *this;
00499     }
00500
00506     Matrix& operator*=(int a) {
00507         for (auto& row : data) {
00508             for (auto& elem : row) {
00509                 elem *= a;
00510             }
00511         }
00512         return *this;
00513     }
00514
00522     Matrix& operator()(double a) {
00523         int val = (int)std::floor(a);
00524         for (auto& row : data) {
00525             for (auto& elem : row) {
00526                 elem += val;
00527             }
00528         }
00529         return *this;
00530     }
00531
00540     friend std::ostream& operator<<(std::ostream& o, Matrix const& m) {
00541         for (const auto& row : m.data) {
00542             for (const auto& elem : row) {
00543                 o << elem << " ";
00544             }
00545             o << "\n";
00546         }
00547         return o;
00548     }
00549
00557     bool operator==(const Matrix& m) const {
```

```
00558         if (data.size() != m.data.size() || data[0].size() != m.data[0].size()) return false;
00559         for (int i = 0; i < (int)data.size(); i++) {
00560             for (int j = 0; j < (int)data[i].size(); j++) {
00561                 if (data[i][j] != m.data[i][j]) return false;
00562             }
00563         }
00564         return true;
00565     }
00566 
00572     bool operator>(const Matrix& m) const {
00573         if (data.size() != m.data.size() || data[0].size() != m.data[0].size()) return false;
00574         for (int i = 0; i < (int)data.size(); i++) {
00575             for (int j = 0; j < (int)data[i].size(); j++) {
00576                 if (!(data[i][j] > m.data[i][j])) return false;
00577             }
00578         }
00579         return true;
00580     }
00581 
00587     bool operator<(const Matrix& m) const {
00588         if (data.size() != m.data.size() || data[0].size() != m.data[0].size()) return false;
00589         for (int i = 0; i < (int)data.size(); i++) {
00590             for (int j = 0; j < (int)data[i].size(); j++) {
00591                 if (!(data[i][j] < m.data[i][j])) return false;
00592             }
00593         }
00594         return true;
00595     }
00596 };
```

Index