AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

ZAAWANSOWANE PROGRAMOWANIE

Macierz Kwadratowa z zastosowaniem Copilot

Autor:
Jakub Piwko
Łukasz Nowak

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2023

Spis treści

1. Ogólne określenie wymagań **4**

2. Analiza problemu **7**

3. Projektowanie **10**

3.1. Visual Studio Code 10

3.2. Github + Git 11

4. Implementacja **13**

4.1. Klasa Matrix 13

4.2. Konstruktor Domyślny 13

4.3. Konstruktor z Parametrem 13

4.4. Konstruktor Kopiujący 14

4.5. Konstruktor Alokacji Pamięci 14

4.6. Destruktor 15

4.7. Metoda Wypisania Macierzy 15

4.8. Metoda Alokacji Pamięci 16

4.9. Metoda Wstawiania Wartości 16

4.10. Metoda Zwracająca Wartość Elementu (x,y) 16

4.11. Metoda Zamieniająca Wiersze z Kolumnami 17

4.12. Metoda Wypełniająca Macierz cyframi od 0 do 9 17

4.13. Metoda Wypełniająca Wybraną Część Macierzy cyframi od 0 do 9 18

4.14. Metoda Wypełniająca Macierz po Przekątnej 18

4.15. Metoda Wypełniająca Macierz po Wskazanej Przekątnej 19

4.16. Metoda Wypełniająca Wskazaną Kolumnę Wartościami 20

4.17. Metoda Wypełniająca Wskazany Wiersz Wartościami 20

4.18. Metoda Wypełniająca Przekątną liczbami 1 20

4.19. Metoda Wypełniająca pod Przekątną liczbami 1 21

4.20. Metoda Wypełniająca nad Przekątną liczbami 1 21

4.21. Metoda Wypełniająca Macierz Szachownicą 22

4.22. Metoda Sumująca Macierze 23

4.23. Metoda Mnożąca Macierze 23

4.24. Metoda Dodająca wartość do elementów macierzy 24
4.25. Metoda Mnożąca wartość elementów macierzy 24
4.26. Metoda Odejmująca int od macierzy 24
4.27. Metoda Dodająca integer od macierzy(odwrócone) 25
4.28. Metoda Mnożąca integer przez elementy macierzy(odwrócone) 25
4.29. Metoda odejmująca integer przez elementy macierzy(odwrócone) 26
4.30. Metody ++A oraz A++ 26
4.31. Metody –A oraz A– 27
4.32. Metoda operator+= 28
4.33. Metoda operator-= 28
4.34. Metoda operator+= typu double 28
4.35. Metoda operator== 29
4.36. Metoda operator¿ 29
4.37. Metoda operator¡ 30

5. Wnioski **31**

Literatura **32**

Spis rysunków **33**

Spis tabel **34**

Spis listingów **35**

1. Ogólne określenie wymagań

Celem tego zadania jest napisanie klasy matrix, która wykonuje działania na macierzy kwadratowej. Wykonanie tego zadania polega na użyciu funkcjonalności Github Copilot do jego rozwiązania oraz obeserwacji i analizy rozwiązań zaproponowanych przez narzędzie Copilot.

Klasa matrix ma zawierać poniższe funkcjonalności:
• matrix(void); //konstruktor domyślny bez alokacji pamięci
• matrix(int n); //przeciążeniowy alokuje macierz o wymiarach n na n,
• matrix(int n , int* t); //przeciążeniowy alokuje pamięć i przepisuje dane z tabeli
• matrix(matrix& m); //konstruktor kopiujący
• matrix(void); //destruktor
• matrix& alokuj(int n); //jeśli macierz nie ma zaalokowanej pamięci to ją alokuje w wielkości n na n, jeśli macierz ma zaalokowaną pamięć to sprawdza czy rozmiar alokacji jest równy zdeklarowanemu rozmiarowi. W przypadku gdy tej pamięci jest mniej, pamięć ma zostać zwolniona i zaalokowana ponownie w żądanym rozmiarze. W przypadku gdy tej pamięci jest więcej pozostawić alokację bez zmian.
• matrix& wstaw(int x, int y, int wartosc); //wiersz, kolumna, wartość
• int pokaz(int x, int y); //zwraca wartość elementu x, y
• matrix& odwroc(void); //zamienia wiersze z kolumnami
• matrix& losuj(void); //wypełniamy cyframi od 0 do 9 wszystkie elementy macierzy
• matrix& losuj(int x); //wypełniamy cyframi od 0 do 9 elementy macierzy. Zmienna x określa ile cyfr będziemy losować. Następnie algorytm losuje, w które miejsca wstawi wylosowane cyfry
• matrix& diagonalna(int* t); //po przekątnej są wpisane dane z tabeli, a pozostałe elementy są równe 0
• matrix& diagonalna_k(int k, int* t); // po przekątnej są wpisane dane z tabeli, a pozostałe elementy są równe 0. Parametr k może oznaczać: 0 - przekątna przechodząca przez środek (czyli tak jak metoda diagonalna), cyfra dodatnia przesuwa diagonalną do góry macierzy o podaną cyfrę, cyfra ujemna przesuwa diagonalną w dół o podaną cyfrę
• matrix& kolumna(int x, int* t); //przepisuje dane z tabeli do kolumny, którą wskazuje zmienna x
• matrix& wiersz(int y, int* t); //przepisuje dane z tabeli do wiersza, który wskazuje zmienna x

• matrix& przekatna(void); //uzupełnia macierz: 1-na przekątnej, 0-poza przekątną
• matrix& pod_przekatna(void); //uzupełnia macierz: 1-pod przekątną, 0-nad przekątną i po przekątnej
• matrix& nad_przekatna(void); //uzupełnia macierz: 1-nad przekątną, 0-pod przekątną i po przekątnej
• matrix& szachownica(void); //uzupełnia macierz w ten sposób dla n=4: 0101 1010 0101 1010
• matrix& operator+(matrix& m); //A+B
• matrix& operator*(matrix& m); //A*B
• matrix& operator+(int a); //A+int
• matrix& operator*(int a); //A*int
• matrix& operator-(int a); //A-int
• friend matrix operator+(int a, matrix& m); //int+A
• friend matrix operator*(int a, matrix& m); //int*A
• friend matrix operator-(int a, matrix& m); //int-A
• matrix& operator++(int); //A++ wszystkie liczby powiększone o 1
• matrix& operator–(int); //A– wszystkie liczby pomniejszone o 1
• matrix& operator+=(int a); //każdy element w macierzy powiększamy o „a”
• matrix& operator-=(int a); //każdy element w macierzy pomniejszamy o „a”
• matrix& operator*=(int a); //każdy element w macierzy mnożymy o „a”
• matrix& operator(double); //wszystkie cyfry są powiększone o część całkowitą z wpisanej cyfry
• friend ostream& operator«(ostream& o, matrix& m); //wypisanie macierzy
• bool operator==(const matrix& m); //sprawdza, czy każdy element macierzy spełnia równość A(n,m) == B(n,m)
A=(1 2) B=(1 2)
(3 4) (3 4)
jeśli nie, to nie możemy mówić że macierze są równe,
• bool operator¿(const matrix& m); //operator większości sprawdza, czy każdy element macierzy spełnia nierówność A(n,m) ¿ B(n,m). Jeśli tak, to możemy powiedzieć że macierz jest większa, w przeciwnym wypadku nie możemy stwierdzić, że macierz jest większa.
• bool operator¡(const matrix& m); //tak jak wyżej tylko operator mniejszości. Na marginesie macierzy możemy nie dać rady określić że jest równa, mniejsza i większa, wtedy mówimy że jest różna

Klasa matrix musi być napisana w osobnym pliku. Funkcja main (też osobny plik)

musi uruchamiać wszystkie metody celem sprawdzenia ich poprawności. Dobrym sposobem będzie wczytanie macierzy lub tabel z pliku aby nie wpisywać ich za każdym razem z klawiatury. Macierz powinna być testowana co najmniej na n=30 lub więcej.

2. Analiza problemu

Opis ogólny

Macierze są tabelarycznymi zbiorami liczb ułożonymi w rzędach i kolumnach. Każdy element macierzy jest identyfikowany przez dwie indeksy: indeks wiersza (i) i indeks kolumny (j). Macierze są szeroko stosowane w matematyce, naukach przyrodniczych, inżynierii i innych dziedzinach.

Reprezentacja macierzy

Macierz A o wymiarach $m \times n$ może być reprezentowana w prostokątnej notacji:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

gdzie m to liczba wierszy, n to liczba kolumn.

Rodzaje macierzy

- **Macierz Kwadratowa:** Ma tyle samo wierszy co kolumn.
- **Macierz Transponowana:** Otrzymuje się przez zamianę wierszy na kolumny.
- **Macierz Jednostkowa:** Diagonalna macierz, gdzie elementy na głównej przekątnej są równe 1, a pozostałe są równe 0.
- **Macierz Zerowa:** Wszystkie elementy są równe 0.

Operacje na Macierzach

- **Dodawanie i Odejmowanie:** Macierze o tych samych wymiarach można dodawać i odejmować element po elemencie.
- **Mnożenie przez Skalar:** Każdy element macierzy jest mnożony przez daną liczbę.
- **Mnożenie Macierzy:** Zostało wcześniej opisane.

- **Inwersja Macierzy:** Nie wszystkie macierze mają odwrotność, ale te, które mają, można odwrócić.

Zastosowania Mnożenia Macierzy

Mnożenie macierzy jest powszechnie stosowane w różnych dziedzinach nauki i inżynierii. Poniżej przedstawiamy kilka przykładów:

- **Grafika Komputerowa:** Transformacje 3D, animacje.
- **Nauka Danych:** Analiza regresji, przekształcenia danych.
- **Fizyka:** Modelowanie układów dynamicznych.
- **Inżynieria:** Analiza struktur, obwodów elektrycznych.
- **Ekonomia:** Modele ekonometryczne.

Opis Algorytmu Mnożenia Macierzy

Mnożenie macierzy jest matematycznym procesem, w którym każdy element wynikowej macierzy jest sumą iloczynów odpowiadających mu elementów z odpowiednich wiersza i kolumny macierzy wejściowych.

$$C(i,j) = \sum_{k=1}^{p} A(i,k) \cdot B(k,j)$$

gdzie C to wynikowa macierz o wymiarach $m \times n$, A to macierz $m \times p$, a B to macierz $p \times n$.

Przykład Mnożenia Macierzy

Rozważmy dwie macierze:

$$A = \begin{bmatrix} 2 & 3 \\ 4 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 1 \\ 2 & 3 \end{bmatrix}$$

Wynikowa macierz C o wymiarach 2x2 będzie miała elementy obliczone według wzoru:

$$C(i,j) = \sum_{k=1}^{2} A(i,k) \cdot B(k,j)$$

Ostateczny wynik:

$$C = \begin{bmatrix} 16 & 9 \\ 14 & 7 \end{bmatrix}$$

3. Projektowanie

Do wykonania tego zadania użyte zostało oprogramowanie:

3.1. Visual Studio Code

Visual Studio Code to lekki, wieloplatformowy edytor kodu źródłowego stworzony przez firmę Microsoft. Jest on rozwijany jako projekt o kodzie źródłowym dostępnym publicznie i dostępny do pobrania bezpłatnie. Poniżej znajdują się główne cechy i charakterystyki Visual Studio Code:

Darmowy i otwarty kod źródłowy:

VS Code jest dostępny za darmo do użytku i jego kod źródłowy jest dostępny publicznie. To oznacza, że społeczność programistyczna może przeglądać, zgłaszać błędy, wnosić poprawki i aktywnie uczestniczyć w rozwoju edytora.

Wsparcie dla wielu języków programowania:

VS Code obsługuje wiele popularnych języków programowania, takich jak JavaScript, TypeScript, Python, Java, C#, C++, HTML, CSS i wiele innych. Dostępne są również rozszerzenia dla wielu innych języków, co czyni go uniwersalnym narzędziem dla różnych programistycznych zastosowań.

Inteligentne podpowiedzi i uzupełnianie kodu:

VS Code oferuje inteligentne funkcje uzupełniania kodu, sugerując odpowiednie składniki, metody czy zmienne na podstawie kontekstu kodu.

Integracja z systemem kontroli wersji:

Edytor umożliwia łatwą integrację z systemami kontroli wersji, takimi jak Git, co ułatwia śledzenie zmian w kodzie i współpracę zespołową.

Rozszerzenia i motywy:

Dzięki systemowi rozszerzeń użytkownicy mogą dostosować środowisko pracy zgodnie z własnymi preferencjami. Istnieje wiele dostępnych motywów, które pozwalają dostosować wygląd edytora.

Integracja z narzędziami do budowania i debugowania:

Visual Studio Code integruje się z różnymi narzędziami do budowania i debugowania aplikacji, co ułatwia proces tworzenia, testowania i odpluskwiania kodu.

Terminal wbudowany:

W edytorze dostępny jest wbudowany terminal, co pozwala programistom wykonywać polecenia bezpośrednio z poziomu edytora.

Obsługa rozszerzeń:

Rozszerzenia dostarczają dodatkowe funkcje i wsparcie dla różnych technologii. Społeczność aktywnie tworzy i udostępnia rozszerzenia, co sprawia, że VS Code jest

elastycznym narzędziem dla wielu dziedzin programowania.

Wieloplatformowość:

VS Code jest dostępny na platformy Windows, macOS i Linux, co sprawia, że jest przyjazny dla programistów pracujących na różnych systemach operacyjnych.

Społeczność i dokumentacja:

Visual Studio Code ma aktywną społeczność, która udziela wsparcia, a oficjalna dokumentacja jest obszerna, co ułatwia naukę i korzystanie z tego narzędzia.

Podsumowując, Visual Studio Code jest popularnym edytorem kodu źródłowego z szerokimi możliwościami dostosowania, wsparciem dla wielu języków programowania oraz aktywną społecznością programistyczną. W celu dalszego zapoznania się z tym oprogramowaniem proszę odwiedzić stronę [1].

3.2. Github + Git

Git to system kontroli wersji używany do śledzenia zmian w kodzie źródłowym i współpracy nad projektem programistycznym.

Kontrola Wersji:

Git umożliwia programistom śledzenie zmian w kodzie źródłowym, przechowywanie historii zmian i przywracanie poprzednich wersji projektu.

Rozproszony System Kontroli Wersji:

Git jest rozproszonym systemem, co oznacza, że każdy programista ma pełną kopię repozytorium na swoim komputerze, co ułatwia niezależne pracę i umożliwia pracę offline.

Branching i Merging:

Git pozwala na tworzenie oddzielnych gałęzi (branch) projektu, co umożliwia pracę nad różnymi funkcjonalnościami niezależnie. Następnie można połączyć (merge) te gałęzie w jedną całość.

Współpraca:

Git ułatwia współpracę zespołową, umożliwiając wielu programistom równoczesną pracę nad projektem i rozwiązywanie konfliktów wersji.

Platforma Niezależna:

Git jest dostępny na różnych platformach (Windows, macOS, Linux) i można go wykorzystywać w różnych narzędziach i środowiskach programistycznych.

Bezpieczeństwo:

Repozytoria Git są zabezpieczone i chronione, co zapewnia bezpieczne przechowywanie kodu źródłowego.

Społeczność:

Git ma ogromną społeczność użytkowników i jest szeroko używany w całym świecie, co oznacza, że dostępne są liczne źródła wsparcia i dokumentacji. Więcej informacji na temat Github można zdobyć na stronie do której link znajduje się w bibliografii [2].

4. Implementacja

4.1. Klasa Matrix

Klasa Matrix posiada 2 zmienne, zmienną n która określa wielkość macierzy oraz zmienną **matrix która jest wskaźnikiem do wskaźników na int, który będzie wskazywał na dynamicznie alokowaną pamięć dla macierzy. W publicznej części klasy znajdują się wszystkie metody, które opisane zostaną indywidualnie w dalszej części dokumentacji.

```
    class Matrix {
private:
    int n;
    int** matrix;
public:
                    -=- Metody -=-
    -=- Opisane w dalszej czesci dokumentacji -=-
};
```

Listing 1. Klasa Matrix

4.2. Konstruktor Domyślny

Konstruktor opisany na listingu 2 wykorzystywany jest w sytuacji tworzenia obiektu klasy matrix, jakim jest macierz, w przypadku braku podania parametru.

```
    //konstruktor domyslny
    Matrix::Matrix(void){
    this->n = 0;
    this->matrix = new int*[n];
    for(int i = 0; i < n; i++){
        this->matrix[i] = new int[n];
    }
    }
```

Listing 2. Konstruktor Domyślny

4.3. Konstruktor z Parametrem

Konstruktor opisany na listingu 3 wykorzystywany jest w sytuacji tworzenia obiektu klasy matrix, jakim jest macierz, w przypadku podania parametru.

```
    //konstruktor z parametrem
    Matrix::Matrix(int n){
```

```
3     this->n = n;
4     this->matrix = new int*[n];
5     for(int i = 0; i < n; i++){
6         this->matrix[i] = new int[n];
7     }
8     }
```

Listing 3. Konstruktor z Parametrem

4.4. Konstruktor Kopiujący

Konstruktor opisany na listingu 4 wykorzystywany jest w sytuacji tworzenia obiektu klasy matrix z zamiarem przypisania wartości innego obiektu tej samej klasy. Kopiuje on zawartość jednej macierzy do innej.

```
1     //konstruktor kopiujacy
2     Matrix::Matrix(const Matrix& m){
3     this->n = m.n;
4     this->matrix = new int*[n];
5     for(int i = 0; i < n; i++){
6         this->matrix[i] = new int[n];
7     }
8     for(int i = 0; i < n; i++){
9         for(int j = 0; j < n; j++){
10            this->matrix[i][j] = m.matrix[i][j];
11        }
12    }
13    }
```

Listing 4. Konstruktor Kopiujący

4.5. Konstruktor Alokacji Pamięci

Konstruktor opisany na listingu 5 wykorzystywany aby przypisać odpowiedni obszar pamięci dla obiektu klasy matrix.

```
1     //konstruktor alokuje pamiec i przepisuje dane z tabeli
2     Matrix::Matrix(int n, int* tab){
3     this->n = n;
4     this->matrix = new int*[n];
5     for(int i = 0; i < n; i++){
6         this->matrix[i] = new int[n];
7     }
8     int k = 0;
```

```
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            this->matrix[i][j] = tab[k];
            k++;
        }
    }
    }
```

Listing 5. Konstruktor Alokacji Pamięci

4.6. Destruktor

Destruktor opisany na listingu 6 wykorzystywany aby zwolnić pamięć obiektu klasy matrix ze stosu.

```
    //destruktor
    Matrix::~Matrix(){
    for(int i = 0; i < n; i++){
        delete[] matrix[i];
    }
    delete[] matrix;
    }
```

Listing 6. Destruktor

4.7. Metoda Wypisania Macierzy

Metoda Wypisania Macierzy przedstawiona na listingu 7 jest przeciążeniem operatora ostream, dzięki czemu jest w stanie wypisywać macierz za pomocą polecenia cout.

```
    //wypisanie macierzy
    ostream& operator<<(ostream& o, Matrix& m) {
    for (int i = 0; i < m.n; i++) {
        for (int j = 0; j < m.n; j++) {
            o << m.matrix[i][j] << " ";
        }
        o << endl;
    }
    return o;
    }
```

Listing 7. Metoda Wypisania Macierzy

4.8. Metoda Alokacji Pamięci

Metoda Alokacji Pamięci przedstawiona na listingu 8 pozwala na modyfikacje alokowanej pamięci obiektu klasy matrix.

```
1    //metoda alokujaca pamiec na macierz
2    Matrix& Matrix::alokuj(int n){
3    this->n = n;
4    this->matrix = new int*[n];
5    for(int i = 0; i < n; i++){
6        this->matrix[i] = new int[n];
7    }
8    return *this;
9    }
```

Listing 8. Metoda Alokacji Pamięci

4.9. Metoda Wstawiania Wartości

Metoda Wstawiania Wartości przedstawiona na listingu 9 pozwala na wstawianie wartości do obiektu klasy matrix w podanym przez użytkownika miejscu.

```
1    //metoda wstawiajaca wartosc
2    Matrix& Matrix::wstaw(int x, int y, int value){
3    this->matrix[x][y] = value;
4    return *this;
5    }
```

Listing 9. Metoda Wstawiająca wartość

4.10. Metoda Zwracająca Wartość Elementu (x,y)

Metoda Zwracająca Wartość Elementu (x,y) przedstawiona na listingu 10 pozwala na zwracanie wartości obiektu klasy matrix w podanym przez użytkownika miejscu.

```
1    //zwraca wartosc elementu x, y
2    int Matrix::pokaz(int x, int y){
3    return this->matrix[x][y];
4    }
```

Listing 10. Metoda Zwracająca Wartość Elementu (x y)

4.11. Metoda Zamieniająca Wiersze z Kolumnami

Metoda Zamieniająca Wiersze z Kolumnami przedstawiona na listingu 11 pozwala na zamianę wierszy oraz kolumn miejscami.

```
    //zamienia wiersze z kolumnami
    Matrix& Matrix::odwroc(void){
    int** temp = new int*[n];
    for(int i = 0; i < n; i++){
        temp[i] = new int[n];
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            temp[i][j] = this->matrix[j][i];
        }
    }
    for(int i = 0; i < n; i++){
        delete[] this->matrix[i];
    }
    delete[] this->matrix;
    this->matrix = temp;
    return *this;
}
```

Listing 11. Metoda Zamieniająca Wiersze z Kolumnami

4.12. Metoda Wypełniająca Macierz cyframi od 0 do 9

Metoda Wypełniająca Macierz cyframi od 0 do 9 przedstawiona na listingu 12 wypełnia macierz losowymi cyframi w przedziale od 0 do 9.

```
    //wypelnia cyframi od 0 do 9 wszystkie elementy tablicy
    Matrix& Matrix::losuj(void){
    srand(time(NULL));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            this->matrix[i][j] = rand() % 10;
        }
    }
    return *this;
    }
```

Listing 12. Metoda Wypełniająca Macierz cyframi od 0 do 9

4.13. Metoda Wypełniająca Wybraną Część Macierzy cyframi od 0 do 9

Metoda Wypełniająca Wybraną Część Macierzy cyframi od 0 do 9 przedstawiona na listingu 13 wypełnia macierz losowymi cyframi w przedziale od 0 do 9 na obszarze o długości wyznaczonej przez zmienną x.

```
1   //wypelniamy cyframi od 0 do 9 elementy macierzy. Zmienna x
    okresla ile cyfr bedziemy losowac Nastepnie algorytm losuje, w
    ktore miejsca wstawi wylosowane cyfry
2   Matrix& Matrix::losuj(int x){
3   srand(time(NULL));
4   int* tab = new int[x];
5   for(int i = 0; i < x; i++){
6       tab[i] = rand() % 10;
7   }
8   int k = 0;
9   for(int i = 0; i < n; i++){
10      for(int j = 0; j < n; j++){
11          this->matrix[i][j] = tab[k];
12          k++;
13          if(k == x){
14              k = 0;
15          }
16      }
17  }
18  delete[] tab;
19  return *this;
20  }
```

Listing 13. Metoda Wypełniająca Wybraną Część Macierzy cyframi od 0 do 9

4.14. Metoda Wypełniająca Macierz po Przekątnej

Metoda Wypełniająca Macierz po Przekątnej przedstawiona na listingu 14 wypełnia macierz po przekątnej wartościami z tabeli, a pozostałą część macierzy wypełnia zerami.

```
1   //po przekatnej sa wpisane dane z tabeli, a pozostale elementy
    sa rowne 0,
2   Matrix& Matrix::diagonalna(int* tab){
3   int k = 0;
4   for(int i = 0; i < n; i++){
5       this->matrix[i][i] = tab[k];
6       k++;
```

```
    }
    return *this;
    }
```

Listing 14. Metoda Wypełniająca Macierz po Przekątnej

4.15. Metoda Wypełniająca Macierz po Wskazanej Przekątnej

Metoda Wypełniająca Macierz po Wskazanej Przekątnej przedstawiona na listingu 15 wypełnia macierz po przekątnej wskazanej przez zmienną k, wartościami z tabeli, a pozostałą część macierzy wypełnia zerami.

```
    //po przekatnej sa wpisane dane z tabeli, a pozostale elementy
    sa rowne 0. Parametr k moze oznaczac: 0 - przekatna przechodzaca
     przez srodek (czyli tak jak metoda diagonalna), cyfra dodatnia
    przesuwa diagonalna do gory macierzy o podana cyfre, cyfra
    ujemna przesuwa diagonalna w dol o podana cyfre
    Matrix& Matrix::diagonalna_k(int* tab, int k){
    int l = 0;
    if(k == 0){
        for(int i = 0; i < n; i++){
            this->matrix[i][i] = tab[l];
            l++;
        }
    }
    else if(k > 0){
        for(int i = 0; i < n; i++){
            if(i + k < n){
                this->matrix[i][i + k] = tab[l];
                l++;
            }
        }
    }
    else if(k < 0){
        for(int i = 0; i < n; i++){
            if(i + k >= 0){
                this->matrix[i][i + k] = tab[l];
                l++;
            }
        }
    }
    return *this;
```

```
}
```

Listing 15. Metoda Wypełniająca Macierz po Wskazanej Przekątnej

4.16. Metoda Wypełniająca Wskazaną Kolumnę Wartościami

Metoda Wypełniająca Wskazaną Kolumnę Wartościami przedstawiona na listingu 16 wypełnia wskazaną przez zmienną x kolumnę wartościami z tabeli.

```
    //przepisuje dane z tabeli do kolumny, ktora wskazuje zmienna x
    Matrix& Matrix::kolumna(int x, int* t){
    for(int i = 0; i < n; i++){
        this->matrix[i][x] = t[i];
    }
    return *this;
    }
```

Listing 16. Metoda Wypełniająca Wskazaną Kolumnę Wartościami

4.17. Metoda Wypełniająca Wskazany Wiersz Wartościami

Metoda Wypełniająca Wskazany Wiersz Wartościami przedstawiona na listingu 17 wypełnia wskazany przez zmienną y wiersz wartościami z tabeli.

```
    //przepisuje dane z tabeli do wiersza, ktory wskazuje zmienna y
    Matrix& Matrix::wiersz(int y, int* t){
    for(int i = 0; i < n; i++){
        this->matrix[y][i] = t[i];
    }
    return *this;
}
```

Listing 17. Metoda Wypełniająca Wskazany Wiersz Wartościami

4.18. Metoda Wypełniająca Przekątną liczbami 1

Metoda Wypełniająca Przekątną liczbami 1 przedstawiona na listingu 18 wypełnia przekątną macierzy liczbami 1, a pozostałe miejsca liczbą 0.

```
   //uzupelnia macierz: 1-na przekatnej, 0-poza przekatna
    Matrix& Matrix::przekatna(void){
    for(int i = 0; i < n; i++){
        this->matrix[i][i] = 1;
    }
```

```
6     return *this;
7     }
```

Listing 18. Metoda Wypełniająca Przekątną liczbami 1

4.19. Metoda Wypełniająca pod Przekątną liczbami 1

Metoda Wypełniająca pod Przekątną liczbami 1 przedstawiona na listingu 19 wypełnia pod przekątną macierzy liczbami 1, a pozostałe miejsca liczbą 0.

```
1   //uzupelnia macierz: 1-pod przekatna, 0-nad przekatna i po
    przekatnej
2    Matrix& Matrix::pod_przekatna(void){
3    for(int i = 0; i < n; i++){
4        for(int j = 0; j < n; j++){
5            if(i > j){
6                this->matrix[i][j] = 1;
7            }
8            else{
9                this->matrix[i][j] = 0;
10           }
11       }
12   }
13   return *this;
14   }
```

Listing 19. Metoda Wypełniająca pod Przekątną liczbami 1

4.20. Metoda Wypełniająca nad Przekątną liczbami 1

Metoda Wypełniająca nad Przekątną liczbami 1 przedstawiona na listingu 20 wypełnia nad przekątną macierzy liczbami 1, a pozostałe miejsca liczbą 0.

```
1   //uzupelnia macierz: 1-pod przekatna, 0-nad przekatna i po
    przekatnej
2    //uzupelnia macierz: 1-nad przekatna, 0-pod przekatna i po
    przekatnej
3    Matrix& Matrix::nad_przekatna(void){
4    for(int i = 0; i < n; i++){
5        for(int j = 0; j < n; j++){
6            if(i < j){
7                this->matrix[i][j] = 1;
8            }
9            else{
10               this->matrix[i][j] = 0;
```

```
            }
        }
    }
    return *this;
    }
```

Listing 20. Metoda Wypełniająca nad Przekątną liczbami 1

4.21. Metoda Wypełniająca Macierz Szachownicą

Metoda Wypełniająca macierz w postaci szachownicy przedstawiona na listingu 21 wypełnia macierz liczbami 1 oraz 0 w taki sposób, aby przypominała wyglądem szachownice.

```
  //uzupelnia macierz w ten sposob dla n=4:
    //0101
    //1010
    //0101
    //1010
    Matrix& Matrix::szachownica(void){
    int k = 0;
    for(int i = 0; i < n; i++){
        if(i % 2 == 0){
            for(int j = 0; j < n; j++){
                if(j % 2 == 0){
                    this->matrix[i][j] = 0;
                }
                else{
                    this->matrix[i][j] = 1;
                }
            }
        }
        else{
            for(int j = 0; j < n; j++){
                if(j % 2 == 0){
                    this->matrix[i][j] = 1;
                }
                else{
                    this->matrix[i][j] = 0;
                }
            }
        }
    }
    return *this;
```

```
}
```

Listing 21. Metoda Wypełniająca macierz 1 i 0 w postaci szachwnicy

4.22. Metoda Sumująca Macierze

Metoda Sumująca Macierze przedstawiona na listingu 22 dodaje do siebie poszczególne wartości elementów macierzy.

```
//A+B
Matrix& Matrix::operator+(Matrix& m){
    Matrix* temp = new Matrix(n);
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            temp->matrix[i][j] = this->matrix[i][j] + m.matrix[i][j
   ];
        }
    }
    return *temp;
}
```

Listing 22. Metoda Sumująca macierze

4.23. Metoda Mnożąca Macierze

Metoda Mnożąca Macierze przedstawiona na listingu 23 wyświetla iloczyn elementów macierzy. Należy zauważyć że aby pomnożyć macierze ilość elementów w kolumnie lub wierszu jednej macierzy musi być równy wierszowi lub kolumnie w drugiej.

```
//A*B
Matrix& Matrix::operator*(Matrix& m){
    Matrix* temp = new Matrix(n);
    int suma = 0;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            suma = 0;
            for(int k = 0; k < n; k++){
                suma += this->matrix[i][k] * m.matrix[k][j];
            }
            temp->matrix[i][j] = suma;
        }
    }
    return *temp;
}
```

Listing 23. Metoda Mnożąca macierze

4.24. Metoda Dodająca wartość do elementów macierzy

Metoda Dodająca wartość typu integer do każdego elementu Macierze przedstawiona na listingu 24 dodaje wartość typu int do elementów macierzy.

```
//A+int
Matrix& Matrix::operator+(int a){
    Matrix* temp = new Matrix(n);
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            temp->matrix[i][j] = this->matrix[i][j] + a;
        }
    }
    return *temp;
}
```

Listing 24. Metoda Dodająca zmienną typu int do elementów macierzy

4.25. Metoda Mnożąca wartość elementów macierzy

Metoda Mnożąca wartość typu integer do każdego elementu Macierzy przedstawiona na listingu 25 mnoży elementy macierzy przez wskazaną zmienną.

```
//A+int
Matrix& Matrix::operator+(int a){
    Matrix* temp = new Matrix(n);
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            temp->matrix[i][j] = this->matrix[i][j] + a;
        }
    }
    return *temp;
}
```

Listing 25. Metoda Mnożąca zmienną typu int przez elementy macierzy

4.26. Metoda Odejmująca int od macierzy

Metoda Odejmująca wartość typu integer przedstawiona na listingu 26 odejmuje zmienne typu int od każdego elementu macierzy.

```
//A-int
Matrix& Matrix::operator-(int a){
    Matrix* temp = new Matrix(n);
    for(int i = 0; i < n; i++){
```

```
        for(int j = 0; j < n; j++){
            temp->matrix[i][j] = this->matrix[i][j] - a;
        }
    }
    return *temp;
}
```

Listing 26. Metoda Odejmująca zmienną typu int od elementów macierzy

4.27. Metoda Dodająca integer od macierzy(odwrócone)

Metoda Dodająca wartość typu integer przedstawiona na listingu 27 dodaje zmienne typu int od każdego elementu macierzy. Ta metoda przedstawia "odwrócony"sposób dodawania gdzie do zmiennej/wartości typu int dodajemy obiekt będący macierzą.

```
//int+A
Matrix operator+(int a, Matrix& m){
    Matrix* temp = new Matrix(m.n);
    for(int i = 0; i < m.n; i++){
        for(int j = 0; j < m.n; j++){
            temp->matrix[i][j] = m.matrix[i][j] + a;
        }
    }
    return *temp;
}
```

Listing 27. Metoda Dodająca zmienną typu int do elementów macierzy

4.28. Metoda Mnożąca integer przez elementy macierzy(odwrócone)

Metoda Mnożąca przedstawiona na listingu 28 mnoży zmienne typu int przez każdy element macierzy. Ta metoda przedstawia "odwrócony"sposób mnożenia, gdzie przez zmiennej/wartości typu int mnożymy elementy obiektu będący macierzą.

```
//int*A
Matrix operator*(int a, Matrix& m){
    Matrix* temp = new Matrix(m.n);
    for(int i = 0; i < m.n; i++){
        for(int j = 0; j < m.n; j++){
            temp->matrix[i][j] = m.matrix[i][j] * a;
        }
    }
    return *temp;
```

```
}
```

Listing 28. Metoda Dodająca zmienną typu int do elementów macierzy

4.29. Metoda odejmująca integer przez elementy macierzy(odwrócone)

Metoda Odejmująca przedstawiona na listingu 29 odejmuje zmienne typu int przez każdy element macierzy. Ta metoda przedstawia "odwrócony"sposób odejmowania, gdzie przez zmiennej/wartości typu int odejmujemy elementy obiektu będący macierzą.

```
//int-A
Matrix operator-(int a, Matrix& m){
    Matrix* temp = new Matrix(m.n);
    for(int i = 0; i < m.n; i++){
        for(int j = 0; j < m.n; j++){
            temp->matrix[i][j] = m.matrix[i][j] - a;
        }
    }
    return *temp;
}
```

Listing 29. Metoda Odejmująca zmienną typu int od elementów macierzy

4.30. Metody ++A oraz A++

Metoda przedstawiona na listingu 30 opisują operacje równą dodaniu do każdego elementu macierzy 1, to znaczy zwiększeniu wartosci każdego elementu o 1.

```
//A++
Matrix& Matrix::operator++(int){
    Matrix* temp = new Matrix(n);
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            temp->matrix[i][j] = this->matrix[i][j];
        }
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            this->matrix[i][j]++;
        }
    }
    return *temp;
}
```

```
16
17 //++A
18 Matrix& Matrix::operator++(){
19     for(int i = 0; i < n; i++){
20         for(int j = 0; j < n; j++){
21             this->matrix[i][j]++;
22         }
23     }
24     return *this;
25 }
```

Listing 30. Metoda ++A oraz A++

4.31. Metody –A oraz A–

Metoda przedstawiona na listingu 31 opisują operacje równą odjęciu od każdego elementu macierzy 1, to znaczy zmniejszeniu wartości każdego elementu o 1.

```
1  //A--
2  Matrix& Matrix::operator--(int){
3      Matrix* temp = new Matrix(n);
4      for(int i = 0; i < n; i++){
5          for(int j = 0; j < n; j++){
6              temp->matrix[i][j] = this->matrix[i][j];
7          }
8      }
9      for(int i = 0; i < n; i++){
10         for(int j = 0; j < n; j++){
11             this->matrix[i][j]--;
12         }
13     }
14     return *temp;
15 }
16
17 //--A
18 Matrix& Matrix::operator--(){
19     for(int i = 0; i < n; i++){
20         for(int j = 0; j < n; j++){
21             this->matrix[i][j]--;
22         }
23     }
24     return *this;
25 }
```

Listing 31. Metoda –A oraz A–

4.32. Metoda operator+=

Metoda przedstawiona na listingu ?? opisują operacje równą dodaniu do każdego elementu macierzy wartość typu integer nie poprzez metodę klasy lecz za pomocą operatora +=.

```
//kazdy element w macierzy powiekszamy o a
Matrix& Matrix::operator+=(int a){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            this->matrix[i][j] += a;
        }
    }
    return *this;
}
```

Listing 32. Operator +

4.33. Metoda operator-=

Metoda przedstawiona na listingu ?? opisują operacje równą odjęciu od każdego elementu macierzy wartość typu integer nie poprzez metodę klasy lecz za pomocą operatora -=.

```
//kazdy element w macierzy pomniejszamy o a
Matrix& Matrix::operator-=(int a){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            this->matrix[i][j] -= a;
        }
    }
    return *this;
}
```

Listing 33. Operator-

4.34. Metoda operator+= typu double

Metoda przedstawiona na listingu ?? opisują operacje równą dodaniu do każdego elementu macierzy wartość typu double, a dokładnie jej całkowitą część, nie poprzez metodę klasy lecz za pomocą operatora +=.

```
//wszystkie cyfry sa powiekszone o czesc calkowita z wpisanej cyfry
Matrix& Matrix::operator+(double a){
    int b = (int)a;
```

```
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            this->matrix[i][j] += b;
        }
    }
    return *this;
}
```

Listing 34. Operator+

4.35. Metoda operator==

Metoda przedstawiona na listingu **??** opisują operacje sprawdzania czy dwa obiekty klasy matrix są równe.

```

bool Matrix::operator==(const Matrix& m){
    bool flag = true;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(this->matrix[i][j] != m.matrix[i][j]){
                flag = false;
            }
        }
    }
    return flag;
}
```

Listing 35. Operator

4.36. Metoda operator¿

Metoda przedstawiona na listingu 36 opisują operacje sprawdzania czy jeden obiekt klasy matrix jest większy od drugiego.

```
//operator wiekszosci
bool Matrix::operator>(const Matrix& m){
    bool flag = true;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if(this->matrix[i][j] < m.matrix[i][j]){
                flag = false;
            }
        }
    }
```

```
11      return flag;
12  }
```

Listing 36. Operator¿

4.37. Metoda operator¡

Metoda przedstawiona na listingu 37 opisują operacje sprawdzania czy jeden obiekt klasy matrix jest mniejszy od drugiego.

```
1
2   //operator mniejszosci
3   bool Matrix::operator<(const Matrix& m){
4       bool flag = true;
5       for(int i = 0; i < n; i++){
6           for(int j = 0; j < n; j++){
7               if(this->matrix[i][j] > m.matrix[i][j]){
8                   flag = false;
9               }
10          }
11      }
12      return flag;
13  }
```

Listing 37. Operator¡

5. Wnioski

Bibliografia

[1] *Strona internetowa Visual Studio Code*. URL: https://code.visualstudio.com (term. wiz. 29. 11. 2023).

[2] *Strona internetowa Github*. URL: https://github.com (term. wiz. 13. 11. 2023).

Spis rysunków

Spis tabel

Spis listingów

1. Klasa Matrix 13
2. Konstruktor Domyślny 13
3. Konstruktor z Parametrem 13
4. Konstruktor Kopiujący 14
5. Konstruktor Alokacji Pamięci 14
6. Destruktor 15
7. Metoda Wypisania Macierzy 15
8. Metoda Alokacji Pamięci 16
9. Metoda Wstawiająca wartość 16
10. Metoda Zwracająca Wartość Elementu (x y) 16
11. Metoda Zamieniająca Wiersze z Kolumnami 17
12. Metoda Wypełniająca Macierz cyframi od 0 do 9 17
13. Metoda Wypełniająca Wybraną Część Macierzy cyframi od 0 do 9 . . 18
14. Metoda Wypełniająca Macierz po Przekątnej 18
15. Metoda Wypełniająca Macierz po Wskazanej Przekątnej 19
16. Metoda Wypełniająca Wskazaną Kolumnę Wartościami 20
17. Metoda Wypełniająca Wskazany Wiersz Wartościami 20
18. Metoda Wypełniająca Przekątną liczbami 1 20
19. Metoda Wypełniająca pod Przekątną liczbami 1 21
20. Metoda Wypełniająca nad Przekątną liczbami 1 21
21. Metoda Wypełniająca macierz 1 i 0 w postaci szachwnicy 22
22. Metoda Sumująca macierze 23
23. Metoda Mnożąca macierze 23
24. Metoda Dodająca zmienną typu int do elementów macierzy 24
25. Metoda Mnożąca zmienną typu int przez elementy macierzy 24
26. Metoda Odejmująca zmienną typu int od elementów macierzy 24
27. Metoda Dodająca zmienną typu int do elementów macierzy 25
28. Metoda Dodająca zmienną typu int do elementów macierzy 25
29. Metoda Odejmująca zmienną typu int od elementów macierzy 26
30. Metoda ++A oraz A++ 26
31. Metoda –A oraz A– 27

32. Operator + ... 28
33. Operator- 28
34. Operator+ 28
35. Operator 29
36. Operator¿ 29
37. Operator¡ 30