

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

Macierz Matrix przy użyciu Github Copilot

Autor:
Kamil Gruca
Jakub Hajduk

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań	3
2. Analiza problemu	7
2.1. Przedstawienie Macierzy	7
2.2. GitHub Copilot	10
2.3. Zastosowanie macierzy kwadratowej	10
3. Projektowanie	11
3.1. Narzędzia użyte w projekcie	11
3.2. Sposób użycia narzędzi	11
4. Implementacja	13
4.1. Struktura kodu	13
4.2. Rozłożenie metody wstaw	14
4.3. Rozłożenie metody przekatna	15
4.4. Rozłożenie konstruktora alokującego pamięć	16
4.5. Problemy podczas pracy z GitHub Copilot	17
4.6. Powstałe wyniki	17
4.7. Efekt programu	18
4.8. Drzewo commitów z Githuba	19
5. Wnioski	20
Literatura	21
Spis rysunków	22
Spis listingów	23

1. Ogólne określenie wymagań

Celem tego projektu jest stworzenie klasy `matrix`, która umożliwia pracę z macierzami kwadratowymi o wymiarach $(n \times n)$, gdzie (n) oznacza wielkość macierzy. Macierz będzie przechowywana w dynamicznej pamięci (na stercie), co pozwala na łatwe zarządzanie rozmiarem danych w trakcie działania programu. W klasie będzie użytych wiele funkcji, które umożliwią wykonywanie różnych operacji matematycznych i manipulacje na macierzach. Wszystkie te funkcje będą zaimplementowane bez użycia gotowych (wyspecjalizowanych) bibliotek.

W trakcie realizacji projektu klasa `matrix` zostanie umieszczona w osobnym pliku. Podobnie funkcja `main`, odpowiedzialna za testowanie wszystkich metod klasy, również znajdzie się w oddzielnym pliku. Taki podział pozwala na łatwiejsze zarządzanie kodem, rozwijanie go i testowanie. Macierz będzie testowana na przykładach o rozmiarze co najmniej 30×30 . Program zostanie także odpowiednio zabezpieczony w taki sposób, aby uniemożliwić operacje matematycznie niewykonalne, np. mnożenie macierzy o nieprawidłowych rozmiarach.

Klasa `matrix` będzie posiadała następujące funkcje:

Konstruktory:

- `matrix(void)`: Konstruktor domyślny bez alokacji pamięci.
- `matrix(int n)`: Konstruktor, który alokuje pamięć dla macierzy o wymiarach $n \times n$.
- `matrix(int n, int* t)`: Konstruktor, który alokuje pamięć i wypełnia macierz danymi z tabeli `t`.
- `matrix(matrix& m)`: Konstruktor kopiujący.

Destruktor:

- `matrix(void)`: Zwolnienie pamięci zajmowanej przez macierz.

Funkcje zarządzania pamięcią:

- `matrix& alokuj(int n)`: Alokuje lub dostosowuje pamięć macierzy do wymiarów $n \times n$.

Operacje na elementach macierzy:

- `matrix& wstaw(int x, int y, int wartosc)`: Wstawia wartość do macierzy na pozycji (x, y) .

- `int pokaz(int x, int y)`: Zwraca wartość elementu na pozycji (x, y).

Transformacje macierzy:

- `matrix& dowroc(void)`: Zamienia wiersze z kolumnami.
- `matrix& losuj(void)`: Wypełnia macierz losowymi cyframi od 0 do 9.
- `matrix& losuj(int x)`: Wypełnia losowo x elementów macierzy cyframi od 0 do 9.

Specjalne układy macierzy:

- `matrix& diagonalna(int* t)`: Tworzy macierz diagonalną z danymi z tabeli t.
- `matrix& diagonalna_k(int k, int* t)`: Tworzy macierz diagonalną przesuniętą o k w górę lub w dół.
- `matrix& kolumna(int x, int* t)`: Uzupełnia kolumnę x danymi z tabeli t.
- `matrix& wiersz(int y, int* t)`: Uzupełnia wiersz y danymi z tabeli t.
- `matrix& przekatna(void)`: Tworzy macierz, gdzie 1 jest na przekątnej, a reszta to 0.
- `matrix& pod_przekatna(void)`: Uzupełnia macierz wartościami 1 pod przekątną.
- `matrix& nad_przekatna(void)`: Uzupełnia macierz wartościami 1 nad przekątną.
- `matrix& szachownica(void)`: Tworzy wzór szachownicy dla macierzy.

Operacje matematyczne:

- `matrix& operator+(matrix& m)`: Dodawanie macierzy.
- `matrix& operator*(matrix& m)`: Mnożenie macierzy.
- `matrix& operator+(int a)`: Dodawanie liczby do każdego elementu macierzy.
- `matrix& operator*(int a)`: Mnożenie każdego elementu macierzy przez liczbę.
- `matrix& operator-(int a)`: Odejmowanie liczby od każdego elementu macierzy.
- `friend matrix operator+(int a, matrix& m)`: Dodaje liczbę całkowitą a do każdego elementu macierzy m. Funkcja zwraca nową macierz z wynikiem operacji.

- `friend matrix operator*(int a, matrix& m)`: Mnoży każdy element macierzy `m` przez liczbę całkowitą `a`. Wynikiem jest nowa macierz.
- `friend matrix operator-(int a, matrix& m)`: Odejmuje liczbę całkowitą `a` od każdego elementu macierzy `m`. Funkcja zwraca nową macierz z wynikiem.
- `matrix& operator++(int)`: Inkrementacja każdego elementu macierzy.
- `matrix& operator--(int)`: Dekrementacja każdego elementu macierzy.
- `matrix& operator+=(int a)`: Dodanie liczby do macierzy (z modyfikacją oryginalnej macierzy).
- `matrix& operator-=(int a)`: Odejmowanie liczby od macierzy.
- `matrix& operator*=(int a)`: Mnożenie macierzy przez liczbę.
- `matrix& operator(double)`: Wszystkie cyfry są powiększone o część całkowitą z wpisanej cyfry.

Operatory logiczne:

- `bool operator==(const matrix& m)`: Porównanie macierzy.
- `bool operator>(const matrix& m)`: Sprawdzenie, czy macierz jest większa od innej.
- `bool operator<(const matrix& m)`: Sprawdzenie, czy macierz jest mniejsza od innej.

Inne operacje:

- `friend ostream& operator<<(ostream& o, matrix& m)`: Wyświetlanie macierzy w czytelnej formie.

Przewidywane wyniki.

Po zakończeniu projektu użytkownik będzie mógł:

- tworzyć macierze o różnych rozmiarach i wypełniać je danymi.
- wykonywać podstawowe operacje matematyczne na macierzach, takie jak dodawanie, mnożenie.
- generować specjalne układy macierzy, takie jak macierze diagonalne, szachownicowe czy z losowymi wartościami.

- wykorzystywać klasyczne operatory w dzięki ich przeciążeniu.

Rola GitHub Copilot w projekcie.

W trakcie pracy nad tym projektem użyjemy narzędzia GitHub Copilot, które jest bardzo pomocne dla programistów podczas pisania kodu. GitHub Copilot korzysta ze sztucznej inteligencji, aby:

- proponować fragmenty kodu na podstawie kontekstu i komentarzy.
- przyspieszać implementację powtarzalnych struktur kodu.
- pomagać w znajdowaniu błędów i sugerować poprawki.

Dzięki GitHub Copilot będziemy mogli sprawniej zaimplementować klasę `matrix`, korzystając z sugestii dotyczących funkcji, struktur danych czy też operacji na macierzach. Narzędzie to działa jako rozszerzenie w edytorach kodu, takich jak Visual Studio Code lub Visual Studio 2022, gdzie na bieżąco podpowiada odpowiednie fragmenty kodu.

2. Analiza problemu

2.1. Przedstawienie Macierzy

Macierz¹ to w matematyce tabela, która zawiera liczby. Można ją sobie wyobrazić jako prostokątną tablicę, w której znajdują się liczby ustawione w wierszach i kolumnach. Przedstawia to rysunek nr 2.1 (s. 7):

$$\begin{array}{c}
 1 \\
 2 \\
 3 \\
 \vdots \\
 m
 \end{array}
 \begin{bmatrix}
 \overset{1}{a_{11}} & \overset{2}{a_{12}} & \dots & \overset{n}{a_{1n}} \\
 \overset{2}{a_{21}} & \overset{2}{a_{22}} & \dots & \overset{n}{a_{2n}} \\
 \overset{3}{a_{31}} & \overset{2}{a_{32}} & \dots & \overset{n}{a_{3n}} \\
 \vdots & \vdots & \vdots & \vdots \\
 \overset{m}{a_{m1}} & \overset{2}{a_{m2}} & \dots & \overset{n}{a_{mn}}
 \end{bmatrix}$$

Rys. 2.1. Macierz[1]

Macierz składa się z wierszy i kolumn:

- Wiersze to poziome grupy liczb.
- Kolumny to pionowe grupy liczb.

Możemy mówić o macierzy, wskazując jej wiersz i kolumnę, co pozwala precyzyjnie określić położenie każdej liczby w macierzy. Na przykład, jeśli mamy macierz 3x3 (trzy wiersze i trzy kolumny), to każda liczba w tej macierzy będzie miała swoje miejsce na przecięciu odpowiedniego wiersza i kolumny.

Kiedy mówimy, że macierz jest kwadratowa, mamy na myśli to, że liczba wierszy jest równa liczbie kolumn. Przykładowo, jeśli mamy macierz, która ma 3 wiersze, to musi ona mieć również 3 kolumny. Taką macierz nazywamy macierzą kwadratową o wymiarach 3x3. W przypadku, gdy mamy 2 wiersze i 2 kolumny, to również jest to macierz kwadratowa, ale w wymiarze 2x2.

¹Więcej na stronie [https://pl.wikipedia.org/wiki/Macierz\[1\]](https://pl.wikipedia.org/wiki/Macierz[1]).

Macierze kwadratowe są szczególnie ważne, ponieważ mają pewne unikalne właściwości, takie jak przekątna – linia, która łączy elementy znajdujące się na przecięciu pierwszego wiersza i pierwszej kolumny, drugiego wiersza i drugiej kolumny, itd. W matematyce macierze kwadratowe są szczególnie przydatne w obliczeniach związanych z równościami liniowymi, transformacjami geometrycznymi i wielu innych zastosowaniach. Przedstawia to rysunek nr 2.2 (s. 8):

$$A = \underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}}_{n \text{ kolumn}} \left. \vphantom{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}} \right\} n \text{ wierszy}$$

Jeżeli liczba wierszy i liczba kolumn są sobie równe, wtedy A nazywamy macierzą kwadratową wymiaru n .

Rys. 2.2. Macierz kwadratowa[2]

Macierze kwadratowe są także wykorzystywane w macierzach diagonalnych (gdzie wszystkie elementy poza główną przekątną są zerami) oraz w wielu innych zastosowaniach, takich jak macierze symetryczne, macierze ortogonalne czy macierze jednostkowe.

Macierz diagonalna² (wspomniana wcześniej) to szczególny rodzaj macierzy kwadratowej, w której wszystkie elementy poza główną przekątną są zerami. Oznacza to, że tylko elementy na tej przekątnej (od lewego górnego rogu do prawego dolnego rogu) mogą mieć różne wartości, a wszystkie inne elementy muszą być równe zeru. Obrazuje to rysunek nr 2.3 (s. 9) znajdujący się poniżej:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

Macierz kwadratowa A jest **macierzą diagonalną**,

jeżeli jest postaci:

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}$$

Wszystkie wyrazy poza główną przekątną są równe 0.

Rys. 2.3. Macierz diagonalna[2]

Macierze diagonalne są prostsze w obliczeniach, ponieważ większość ich elementów to zera. Oznacza to, że nie musimy zajmować się liczeniem tych zerowych wartości, a skupiamy się tylko na elementach znajdujących się na przekątnej, co upraszcza wszelkie obliczenia. Gdy macierz diagonalna jest mnożona przez inną macierz, obliczenia są znacznie szybsze, ponieważ zera w pozostałych miejscach nie wpływają na wynik mnożenia – tylko wartości na przekątnej mają znaczenie.

²Więcej na stronie https://pl.wikipedia.org/wiki/Macierz_diagonalna[2].

2.2. GitHub Copilot

GitHub Copilot³ to narzędzie, które pomaga programistom pisać kod szybciej i łatwiej. Został stworzony przez firmę GitHub we współpracy z OpenAI i działa jak inteligentny asystent programistyczny. Można go zainstalować jako rozszerzenie do popularnych edytorów kodu, takich jak Visual Studio. Dzięki GitHub Copilot, programiści nie muszą pisać wszystkiego ręcznie. Zamiast tego, Copilot podpowiada, co można napisać dalej, na podstawie tego, co zostało już wprowadzone. Działa to w sposób bardzo podobny do automatycznego uzupełniania, jakie mamy np. w telefonach komórkowych podczas pisania wiadomości.

GitHub Copilot wspiera wiele popularnych języków programowania, takich jak Python, JavaScript, TypeScript czy C++, więc można go używać w różnych projektach, niezależnie od języka, w którym się pracuje. Dzięki temu jest naprawdę wszechstronny. Co więcej, Copilot nie tylko sugeruje kod, ale również pomaga w generowaniu testów czy dokumentacji.

Pomimo swoich zalet, GitHub Copilot nie jest idealny. Czasami może zaproponować coś, co nie do końca pasuje do tego, co chcieliśmy zrobić. Ponadto, ponieważ opiera się na sztucznej inteligencji, może zdarzyć się, że zasugeruje coś, co jest błędne lub nieoptymalne. Dlatego ważne jest, aby sprawdzać każdą sugestię i dostosować ją do swoich potrzeb.

2.3. Zastosowanie macierzy kwadratowej

Macierze kwadratowe są szeroko wykorzystywane w różnych dziedzinach nauki i technologii. Na przykład:

- **Fizyka:** W fizyce często modeluje się różne zjawiska za pomocą równań, które można zapisać w postaci macierzy. Operacje na macierzach pozwalają na szybkie obliczenia, np. w obliczaniach związanych z mechaniką kwantową czy teorią względności.
- **Informatyka:** W informatyce macierze wykorzystywane są do przetwarzania obrazów czy grafów.
- **Ekonomia:** W ekonomii macierze używane są do przedstawiania różnych modeli finansowych i analizowania danych, takich jak przepływ dóbr i usług między krajami.

³Aby dowiedzieć się więcej wejdź na stronę <https://github.com/features/copilot>[3].

3. Projektowanie

3.1. Narzędzia użyte w projekcie

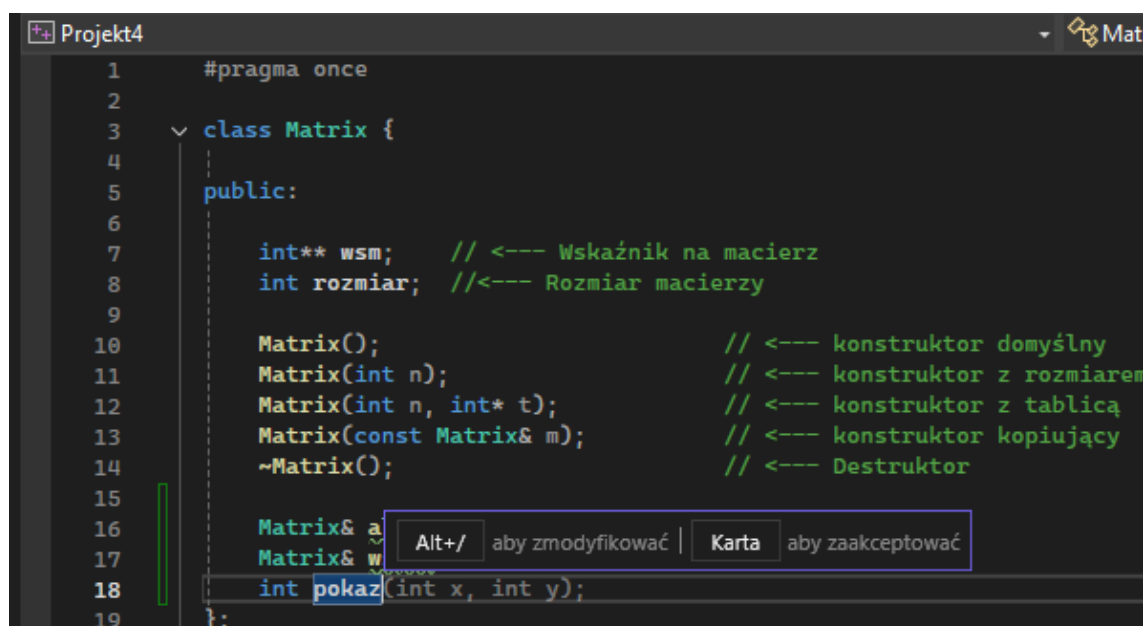
W ramach realizacji projektu będziemy korzystać z kilku narzędzi i technologii, które pomogą nam w tworzeniu i testowaniu działań na macierzach. Głównym językiem programowania będzie C++, ponieważ oferuje on dużą kontrolę nad zarządzaniem pamięcią i jest powszechnie wykorzystywany do tworzenia wydajnych algorytmów, w tym operacji na dużych zbiorach danych, takich jak macierze. Do kompilacji oraz testowania kodu użyjemy środowisko Microsoft Visual Studio 2022, które zapewnia wszystkie niezbędne narzędzia do pracy z C++. Visual Studio 2022 ułatwia proces debugowania oraz umożliwia integrację z różnymi bibliotekami, co będzie szczególnie przydatne przy implementacji i testowaniu złożonych operacji na macierzach.

W projekcie będziemy korzystać z GitHub Copilot, narzędzia wspomagającego programowanie, które wykorzystuje sztuczną inteligencję do generowania kodu. Dzięki temu będziemy mogli szybciej pisać i testować funkcje naszej klasy Matrix, ponieważ Copilot będzie sugerował nam fragmenty kodu i pomagał w rozwiązywaniu problemów, które mogą pojawić się podczas implementacji. Jest to szczególnie przydatne w przypadku pisania metod związanych z macierzami i algorytmami matematycznymi.

3.2. Sposób użycia narzędzi

Aby uzyskać dostęp do GitHub Copilot i w pełni wykorzystać jego możliwości w projekcie, należy przejść przez kilka kroków. Po pierwsze, trzeba założyć konto na GitHubie, jeśli jeszcze go nie mamy. Do rejestracji na GitHubie może być wymagane potwierdzenie tożsamości, na przykład poprzez przesłanie zdjęcia swojej legitymacji studenckiej lub innego dokumentu tożsamości. Wtedy uzyskamy dostęp do specjalnych ofert dla studentów, takich jak GitHub Copilot, który jest darmowy dla zweryfikowanych użytkowników.

Kolejnym krokiem jest zainstalowanie wtyczki GitHub Copilot do Visual Studio 2022. Można to zrobić, przechodząc do Visual Studio Marketplace i wyszukując wtyczkę „GitHub Copilot”, a następnie klikając przycisk „Install”. Po zainstalowaniu wtyczki, należy połączyć ją z naszym kontem na GitHubie. Proces ten polega na zalogowaniu się do GitHub Copilot za pomocą naszych danych logowania do GitHub’a. Jeśli już mamy konto, wystarczy zaakceptować uprawnienia, które są wymagane do działania wtyczki.



Rys. 3.1. Uzupełnianie linii kodu przez GitHub Copilot

Po zainstalowaniu narzędzia, GitHub Copilot automatycznie zacznie sugerować nam fragmenty kodu podczas pisania, pomagając w szybkim oraz efektywnym pisaniu programu, co zostało przedstawione na rysunku nr 3.1 (s. 12). Dodatkowo, będziemy używać systemu kontroli wersji Git, który pozwala na śledzenie zmian w kodzie, zarządzanie wersjami i wspólne tworzenie aplikacji. Wszystkie pliki projektu będą przechowywane w repozytorium na GitHubie. Dzięki temu, w każdej chwili będziemy mogli sprawdzić historię zmian, przywrócić poprzednią wersję kodu, a także współpracować z innymi osobami pracującymi nad tym samym projektem.

4. Implementacja

4.1. Struktura kodu

W naszym projekcie stworzyliśmy klasę, która reprezentuje macierz kwadratową. Klasa ta umożliwia operacje na macierzach, takie jak wypełnianie macierzy losowymi liczbami, uzupełnianie macierzy po przekątnej czy powiększanie wartości macierzy. Program jest podzielony na trzy pliki, co ułatwia zarządzanie kodem i powoduje, że program jest łatwiejszy do rozbudowy.

Program został podzielony na 3 pliki, które odpowiadają za określoną funkcjonalność programu:

1. Plik `Matrix.h`:

W tym pliku znajduje się definicja klasy `Matrix`. Zawiera ona deklarację wszystkich metod, które będą operować na macierzy, oraz deklarację prywatnych zmiennych, takich jak tablica dwuwymiarowa przechowująca dane macierzy (wsm) oraz rozmiar macierzy (rozmiar). Plik ten pełni rolę interfejsu, czyli zapewnia dostęp do funkcji publicznych klasy i ustala strukturę obiektu `Matrix`. Dzięki temu plikowi użytkownik klasy może poznać, jakie funkcje są dostępne oraz jak się z nimi komunikować.

2. Plik `Matrix.cpp`:

W tym pliku znajduje się szczegółowa implementacja wszystkich metod klasy `Matrix`. Dla każdej metody znajdują się odpowiednie funkcje, które implementują logikę operacji na macierzy. Na przykład, w metodzie `wstaw` implementujemy sposób ustawiania wartości w macierzy na podstawie indeksów `x` i `y`, a w metodzie `przekatna` ustawiamy wartości na przekątnej na 1, a pozostałe na 0.

Dzięki oddzieleniu implementacji od definicji w pliku nagłówkowym (`Matrix.h`), łatwiej jest zarządzać kodem, modyfikować go lub dodawać nowe funkcje, przy okazji nie wpływając na inne części programu.

3. Plik `main.cpp`:

W tym pliku znajduje się kod główny programu, który wykorzystuje klasę `Matrix`. Program w tym pliku służy do testowania metod klasy i wykonywania operacji na macierzach. W tym pliku użytkownik może stworzyć obiekt klasy `Matrix`, wykonać operacje jak powiększanie każdego elementu o wartość wpisaną przez użytkownika, uzupełnianie macierzy według preferencji czy zamienianie wierszy z kolumnami.

4.2. Rozłożenie metody wstaw

Funkcja wstaw przypisuje wartość do elementu macierzy w określonej pozycji, jeśli indeksy są prawidłowe. Funkcja ta została przedstawiona na listingu nr 1 (s. 14).

```
1 Matrix& Matrix::wstaw(int x, int y, int wartosc) {  
2     if (x >= 0 && x < rozmiar && y >= 0 && y < rozmiar) {  
3         wsm[x][y] = wartosc;  
4     }  
5     return *this;  
6 }
```

Listing 1. Metoda Matrix& wstaw

Szczegółowe omówienie metody przedstawionej na listingu 1 (s. 14):

- Definicja metody wstaw, która przyjmuje indeksy x, y oraz wartość do przypisania. (Linia 1)
- Sprawdza, czy indeksy x i y są w granicach rozmiaru macierzy. (Linia 2)
- Jeśli indeksy są prawidłowe, przypisuje wartość wartosc do elementu macierzy w pozycji [x][y]. (Linia 3)
- Zwraca referencję do bieżącego obiektu macierzy. (Linia 5)

4.3. Rozłożenie metody przekatna

Funkcja przekatna ustawia wszystkie elementy macierzy na 0, z wyjątkiem elementów na głównej przekątnej, które ustawiane są na 1. Funkcja ta została przedstawiona na listingu nr 2 (s. 15).

```
1 Matrix& Matrix::przekatna(void) {  
2     for (int i = 0; i < rozmiar; ++i) {  
3         for (int j = 0; j < rozmiar; ++j) {  
4             wsm[i][j] = (i == j) ? 1 : 0;  
5         }  
6     }  
7     return *this;  
8 }
```

Listing 2. Metoda Matrix& przekatna

Szczegółowe omówienie metody przedstawionej na listingu 2 (s. 15):

- Rozpoczyna pętlę po wierszach macierzy. (Linia 2)
- Rozpoczyna pętlę po kolumnach macierzy. (Linia 3)
- Ustawia element macierzy w pozycji [i][j] na 1, jeśli $i == j$ (czyli na diagonalu), w przeciwnym przypadku ustawia go na 0. (Linia 4)
- Zwraca referencję do bieżącego obiektu macierzy. (Linia 7)

4.4. Rozłożenie konstruktora alokującego pamięć

Konstruktor klasy Matrix służy do tworzenia obiektu macierzy kwadratowej o rozmiarze $n \times n$ oraz do alokacji pamięci jego elementów. Wszystkie elementy macierzy są początkowo ustawione na wartość 0. Konstruktor ten został przedstawiony na Listingu nr. 3 (s. 16)

```
1 Matrix::Matrix(int n) : wsm(nullptr), rozmiar(0) {
2     if (n > 0) {
3         rozmiar = n;
4         wsm = new int* [rozmiar];
5         for (int i = 0; i < rozmiar; ++i) {
6             wsm[i] = new int[rozmiar];
7             for (int j = 0; j < rozmiar; ++j) {
8                 wsm[i][j] = 0;
9             }
10        }
11    }
12 }
```

Listing 3. Metoda Matrix(int n)

Szczegółowe omówienie Konstruktora przedstawionego na 3 (s. 16):

- Matrix::Matrix(int n): Jest to konstruktor klasy Matrix, który przyjmuje jeden argument n określający rozmiar, wsm(nullptr): Ustawia wskaźnik wsm na nullptr, rozmiar(0): Ustawia początkowy rozmiar macierzy na 0. (Linia 1)
- Instrukcja sprawdza czy podany rozmiar n jest większy od zera. (Linia 2)
- Alokowana jest tablica wskaźników wsm, która będzie przechowywać wskaźniki do wierszy macierzy. (Linia 4)
- Pętla for iteruje przez każdy wiersz (indeks i) i dla każdego z nich alokujemy pamięć dla kolumn. (Linia 5-6)
- Dla każdej komórki w macierzy (w każdym wierszu i i każdej kolumnie j) ustawiamy wartość na 0. (Linia 7-8)

4.5. Problemy podczas pracy z GitHub Copilot

Podczas pisania metod takich jak przekątna, GitHub Copilot nie radził sobie z wypełnianiem odpowiednich pól, gdyż nie wiedział, że metoda "przekątna" wypełnia pola cyfrą 1 w miejscu przekątnej, natomiast poza przekątną cyfrą 0.

Również miał spore problemy podczas pisania metody diagonalna_k. Zamiast uzupełniać kod w odpowiedni sposób, by przesunął kolumny w górę bądź w dół w zależności od wartości podanej przez użytkownika sugerował kod, który przesunął w prawo lub w lewo (wierszami). Mimo podania pierwszych definicji przedstawianej funkcji dalej popełniał błędy i do końca nie wiedział, jak ta metoda ma działać.

Copilot również nie wyróżniał się kreatywnością, jeśli chodzi o dodawanie komentarzy czy też testowanie napisanych funkcji w pliku main.cpp. Komentarze sugerował takie same lub bardzo podobne do stworzonych wcześniej funkcji, a także testy były bardzo podobne, mimo, że funkcje bardzo różniły się swoim działaniem.

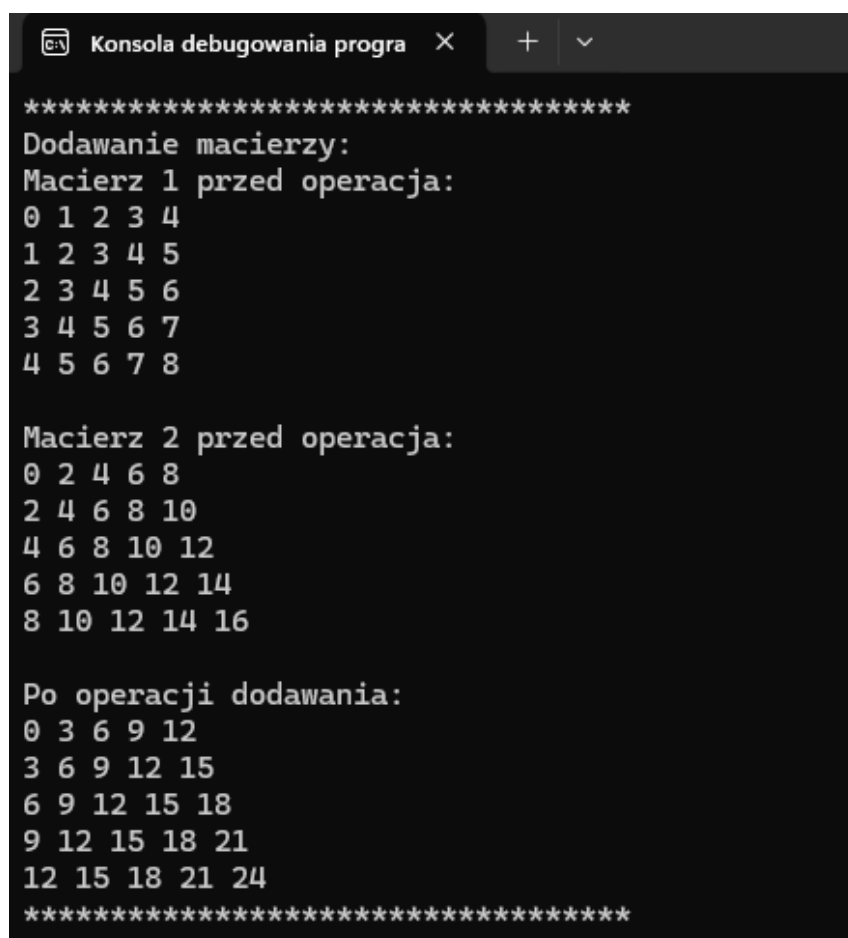
4.6. Powstałe wyniki

W ramach naszej pracy stworzyliśmy w pełni działający program realizujący wszystkie podstawowe operacje związane z macierzami za pomocą klasy Matrix. Użytkownik może łatwo wykonywać takie operacje, jak dodawanie, mnożenie, transponowanie czy losowe wypełnianie macierzy. Program umożliwia również wczytywanie macierzy z pliku. GitHub Copilot bardzo nam pomógł w trakcie realizacji pracy nad projektem. Narzędzie sugerowało nam fragmenty kodu i podpowiadało nam gotowe rozwiązania, dzięki czemu mogliśmy zaoszczędzić czas i skupić się na działaniu oraz logice programu, a nie na drobnych szczegółach implementacji.

4.7. Efekt programu

Efekt programu został przedstawiony na rysunku nr. 4.1 (s. 18), oraz na rysunku nr. 4.2 (s. 19). Rysunki przedstawiają fragment skompilowanego programu, możemy na nich zobaczyć takie funkcje jak:

- Dodawanie macierzy
- Dekrementacja macierzy
- Szachownica



```
*****
Dodawanie macierzy:
Macierz 1 przed operacja:
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8

Macierz 2 przed operacja:
0 2 4 6 8
2 4 6 8 10
4 6 8 10 12
6 8 10 12 14
8 10 12 14 16

Po operacji dodawania:
0 3 6 9 12
3 6 9 12 15
6 9 12 15 18
9 12 15 18 21
12 15 18 21 24
*****
```

Rys. 4.1. Fragment skompilowanego programu

```
Konsola debugowania progra X + v

*****
Macierz dekrementacja:
Przed operacja:
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8

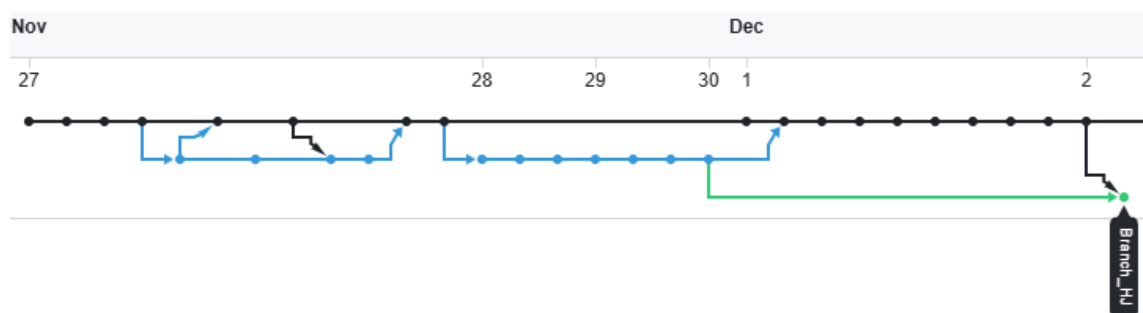
Po operacji:
-1 0 1 2 3
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
*****

Szachownica:
Macierz wygląda następująco:
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
```

Rys. 4.2. Fragment skompilowanego programu

4.8. Drzewo commitów z Githuba

Na rysunku nr. 4.3 (s. 19) zostało przedstawione Network Graph (drzewo commitów), na którym można zobaczyć kiedy i na jakiej gałęzi (branch) był commit, możemy na nim również dostrzec operacje scalania między gałęziami co było kluczowe w naszym projekcie.



Rys. 4.3. Network Graph

5. Wnioski

Podczas realizacji projektu polegającego na napisaniu klasy Matrix do obsługi kwadratowych macierzy, korzystanie z GitHub Copilot było bardzo dużym ułatwieniem, jednakże pokazało to pewne swoje ograniczenia. Przede wszystkim Copilot pomagał głównie w szybszym pisaniu kodu, zwłaszcza w miejscach, gdzie potrzebne były powtarzalne struktury, takie jak konstruktory, destruktory czy metody operujące na elementach macierzy. Jego podpowiedzi były poprawne i trafne przy generowaniu standardowych operacji, takich jak dodawanie, mnożenie czy transpozycja macierzy.

Jednak narzędzie ma swoje wady. Często proponował kod, który wyglądał poprawnie, ale nie zawsze uwzględniał określone działanie danej funkcji (np. miał duże trudności z przekątną czy też z metodą `diagonalna.k`). Przez to wymagało to dokładnego sprawdzania podpowiedzi, a czasami ich całkowitej modyfikacji, aby dostosować je do naszych założeń projektowych. Copilot nie zawsze rozumiał szerszy kontekst projektu, co oznaczało, że bardziej złożone operacje lub skomplikowane algorytmy musieliśmy napisać samodzielnie, ale gdy już Copilot zrozumiał działanie danej funkcji, sam dopowiadał poprawny kod.

W pracy z Copilotem bardzo ważne było zrozumienie kodu i jego ewentualnego poprawiania, gdy Copilot sobie nie radził. Narzędzie to przyspiesza proces pisania, ale wciąż wymaga, aby programista kontrolował jego działanie. Dzięki temu mogliśmy się skupić bardziej na tych skomplikowanych częściach projektu, przez co oszczędzaliśmy czas na prostszych funkcjach.

Copilot na ten moment nie zastąpi programistów, gdyż wymaga on pewnych poprawek i dalszego rozwijania. Może natomiast działać on jako wsparcie, szczególnie w pisaniu powtarzalnego lub też niezbyt skomplikowanego kodu. Aby jednak stworzyć wydajny, dobrze przemyślany projekt, nadal jest niezbędna wiedza programistyczna i doświadczenie człowieka. W przyszłości narzędzia takie jak Copilot mogą zyskać na znaczeniu, ale bardziej jako asystenci podczas pisania kodu, a nie samodzielni twórcy kodu.

Bibliografia

- [1] *Strona internetowa Wikipedia*. URL: <https://pl.wikipedia.org/wiki/Macierz> (term. wiz. 24.11.2024).
- [2] *Strona internetowa smurf.mimuw.edu.pl*. URL: <http://smurf.mimuw.edu.pl/node/1213> (term. wiz. 24.11.2024).
- [3] *Strona internetowa GitHub*. URL: <https://github.com/features/copilot> (term. wiz. 24.11.2024).

Spis rysunków

2.1. Macierz[1]	7
2.2. Macierz kwadratowa[2]	8
2.3. Macierz diagonalna[2]	9
3.1. Uzupełnianie linii kodu przez GitHub Copilot	12
4.1. Fragment skompilowanego programu	18
4.2. Fragment skompilowanego programu	19
4.3. Network Graph	19

Spis listingów

1.	Metoda Matrix& wstaw	14
2.	Metoda Matrix& przekatna	15
3.	Metoda Matrix(int n)	16