Nazwisko i imię: **Soczyński Krzysztof**

Kierunek: **Informatyka w inżynierii komputerowej**

Rok: **I** Semestr: **I** Forma studiów: **niestacjonarne**

Grupa: **Lk4**

Przedmiot: **Podstawy programowania w C/C++**

Zajęcia: **Laboratorium komputerowe 2** Data: **27.11.2022**

1. Treść zadania
    1. *Napisz program obliczający e^x oraz $(a^x)^x$ … n-krotnie.*
    2. *Napisz program mnożący dwie macierze.*

2. Kod programu

1.

```cpp
#include <cmath>
#include <iostream>
#include <numbers>

template <typename T> T myPow(T a, const int x) {
	T result{ 1 };
	for (int i{ 0 }; i < abs(x); ++i) {
		result *= a;
	}

	return x > 0 ? result : T(1) / result;
}

unsigned long long myFactorial(const unsigned long long x) {
	if (x == 0 or x == 1) return 1;

	unsigned long long result{ 1ULL };
	for (auto i{ 2ULL }; i <= x; ++i) {
		result *= i;
	}

	return result;
}

long double myExp(const unsigned long long x) {
	long double result{ 1.0L };
	long double previousResult{ 0.0L };
	constexpr long double errorRange(0.01L / 100.0L );

	for (int i{ 1 }; i < 170 && !(previousResult / result >= 1.0L - errorRange
&& previousResult / result <= 1.0L + errorRange); ++i) {
		const auto powResult = myPow(x, i);
		const auto factorialResult = myFactorial(i);

		previousResult = result;
		result += static_cast<long double>(powResult) / static_cast<long
double>(factorialResult);
	}

	return result;
}

template<typename T> T multipow(T a, int x, const unsigned int n) {
	T result = { 1 };
	for (auto i{ 0U }; i < n; ++i) {
		result = myPow(a, x);
		a = result;
	}

	return result;
}
```

```cpp
int main() {
	int x{}, n{};
	std::cout << "Liczymy exp(x). Podaj x: ";
	std::cin >> x;
	std::cout << "Wynik cmath: " << exp(x) << '\n';
	std::cout << "Wynik myExp: " << myExp(x) << '\n';
	std::cout << "Wynik myPow: " << myPow(std::numbers::e_v, x) << '\n';

	std::cout << "Policzymy n-krotnie 1.1^x. Podaj n: ";
	std::cin >> n;
	std::cout << "Podaj x: ";
	std::cin >> x;
	std::cout << "Wynik: " << multipow(1.1L, x, n) << '\n';

	std::cout << std::endl;
	return 0;
}
```

2.

```
#pragma once

#include <stdexcept>
#include <vector>

template<typename T>
class Matrix {
public:
	Matrix() = default;
	Matrix(size_t rows, size_t columns) : values{
std::vector<std::vector<T>>(rows, std::vector<T>(columns)) } {}
	Matrix(const std::vector<std::vector<T>>& val) : values(val) {}
	Matrix(Matrix&) = default;
	Matrix(Matrix&&) = default;

	[[nodiscard]] Matrix operator *(Matrix& other) {
		if (empty() or other.empty()) throw std::runtime_error("EMPTY
MATRIX!");
		if (numColumns() != other.numRows()) throw std::runtime_error("NON
MATCHING SIZES!");

		std::vector<std::vector<T>> result;
		result.resize(numRows(), std::vector<T>(other.numColumns()));

		for (int i{ 0 }; i < numRows(); ++i) {
			for (int j{ 0 }; j < other.numColumns(); ++j) {
				for (int k{ 0 }; k < numColumns(); ++k) {
					result[i][j] += values[i][k] * other[k][j];
				}
			}
		}
		return result;
	}

	std::vector<T>& operator [](size_t i) { return values[i]; }

	[[nodiscard]] bool empty() const { return values.empty(); }
	[[nodiscard]] size_t numRows() const { return values.size(); }
	[[nodiscard]] size_t numColumns() const {
		if (numRows() == 0) {
			return 0;
		}
		return values[0].size();
	}

private:
	std::vector<std::vector<T>> values;
};
```

```cpp
#include "Matrix.hpp"

#include <iostream>

int main() {
    Matrix<float> matrix1{{ {2,2,2}, {3,3,3}, {4,4,4} }};
    Matrix<float> matrix2{ {{3,3},{4,4},{5,5}} };

    try {
        auto resultMatrix = matrix1 * matrix2;

        for (size_t i = 0; i < resultMatrix.numRows(); ++i) {
            auto& row = resultMatrix[i];

            for (const auto& el : row) {
                std::cout << el << ' ';
            }
            std::cout << '\n';
        }
    } catch (std::runtime_error& ex) {
        std::cout << ex.what();
    }

    std::cout << std::endl;
}
```

3. Opis programu
    1. Dołączone biblioteki

- **<cmath>**
  biblioteka dająca dostęp do funkcji matematycznych zawartych w bibliotece standardowej C++

- **<iostream>**
  biblioteka pozwalająca wczytywać dane i wypisywać dane z/do strumienia wejścia/wyjścia

- **<numbers>**
  biblioteka dająca dostęp do stałych matematycznych (dostępna od C++20)

- **<stdexcept>**
  biblioteka dająca możliwość tworzenia **std::runtime_error**

- **<vector>**
  biblioteka udostępniająca tablicę dynamiczną

    2. Kod „linijka po linijce”
        1.

```
template <typename T> T myPow(T a, const int x) {
	T result{ 1 };
	for (int i{ 0 }; i < abs(x); ++i) {
		result *= a;
	}

	return x > 0 ? result : T(1) / result;
}
```

Funkcja obliczająca dowolną potęgę całkowitą liczby a. Użycie template pozwala przekazać do funkcji parametr dowolnego typu zamiast przeładowywać funkcję kilka razy. Korzystając z operatora warunkowego można w krótki sposób zapisać zwracanie zależne od tego czy potęga jest dodatnia czy ujemna.

```
unsigned long long myFactorial(const unsigned long long x) {
	if (x == 0 or x == 1) return 1;

	unsigned long long result{ 1ULL };
	for (auto i{ 2ULL }; i <= x; ++i) {
		result *= i;
	}

	return result;
}
```

Funkcja obliczająca silnię z liczby x.

```
long double myExp(const unsigned long long x) {
	long double result{ 1.0L };
	long double previousResult{ 0.0L };
	constexpr long double errorRange(0.01L / 100.0L );

	for (int i{ 1 }; i < 170 && !(previousResult / result >= 1.0L - errorRange
&& previousResult / result <= 1.0L + errorRange); ++i) {
		const auto powResult = myPow(x, i);
		const auto factorialResult = myFactorial(i);

		previousResult = result;
		result += static_cast<long double>(powResult) / static_cast<long
double>(factorialResult);
	}

	return result;
}
```

Funkcja obliczająca e^x korzystając ze wzoru $\sum_{k=0}^{\infty} \frac{x^k}{k!}$. Niedokładność wynikająca z zapisu liczb zmiennoprzecinkowych w komputerach niwelowana jest warunkiem sprawdzającym różnicę pomiędzy dwoma kolejnymi wynikami sumowania i gdy jest ona dostatecznie mała uznajemy wynik za poprawny.

```
template<typename T> T multipow(T a, int x, const unsigned int n) {
	T result = { 1 };
	for (auto i{ 0U }; i < n; ++i) {
		result = myPow(a, x);
		a = result;
	}

	return result;
}
```

Funkcja obliczająca n-krotne podniesienie a^x.

```
int main() {
	int x{}, n{};
	std::cout << "Liczymy exp(x). Podaj x: ";
	std::cin >> x;
	std::cout << "Wynik cmath: " << exp(x) << '\n';
	std::cout << "Wynik myExp: " << myExp(x) << '\n';
	std::cout << "Wynik myPow: " << myPow(std::numbers::e_v<double>, x) <<
'\n';

	std::cout << "Policzymy n-krotnie 1.1^x. Podaj n: ";
	std::cin >> n;
	std::cout << "Podaj x: ";
	std::cin >> x;
	std::cout << "Wynik: " << multipow(1.1L, x, n) << '\n';

	std::cout << std::endl;
	return 0;
}
```

Funkcja **`main()`** programu. Użytkownik podaje x – potęgę do której podniesiemy e. Następnie wyliczamy e^x trzema dostępnymi metodami: funkcją **`exp()`** z biblioteki standardowej języka oraz napisanymi funkcjami **`myExp()`** i **`myPow()`** i wyświetlamy wyniki dla porównania.

2.

```
template<typename T>
class Matrix {
public:
	Matrix() = default;
	Matrix(size_t rows, size_t columns) : values{
std::vector<std::vector<T>>(rows, std::vector<T>(columns)) } {}
	Matrix(const std::vector<std::vector<T>>& val) : values(val) {}
	Matrix(Matrix&) = default;
	Matrix(Matrix&&) = default;

	[[nodiscard]] Matrix operator *(Matrix& other) noexcept(false) {
		if (empty() or other.empty()) throw std::runtime_error("EMPTY
MATRIX!");
		if (numColumns() != other.numRows()) throw std::runtime_error("NON
MATCHING SIZES!");

		std::vector<std::vector<T>> result;
		result.resize(numRows(), std::vector<T>(other.numColumns()));

		for (int i{ 0 }; i < numRows(); ++i) {
			for (int j{ 0 }; j < other.numColumns(); ++j) {
				for (int k{ 0 }; k < numColumns(); ++k) {
					result[i][j] += values[i][k] * other[k][j];
				}
			}
		}
		return result;
	}

	std::vector<T>& operator [](size_t i) { return values[i]; }

	[[nodiscard]] bool empty() const { return values.empty(); }
	[[nodiscard]] size_t numRows() const { return values.size(); }
	[[nodiscard]] size_t numColumns() const {
		return numRows() == 0 ? 0 : values[0].size();
	}

private:
	std::vector<std::vector<T>> values;
};
```

Definicja bardzo prostej klasy **`Matrix`**. Wartości przechowywany w macierzy dzięki użyciu szablonu mogą być dowolnego typu. Klasa udostępnia pięć konstruktorów:

- domyślny
- przyjmujący rozmiary macierzy
- przyjmujący **`vector<vector<T>>`** przechowujący dane do zapisania w macierzy
- domyślny konstruktor kopiujący
- domyślny konstruktor przenoszenia

Przeładowanie operatora * pozwala na wykonanie mnożenia macierzy. Operacja może wyrzucić wyjątek **`std:runtime_error`** w sytuacji kiedy któraś z mnożonych macierzy jest pusta lub nie został spełniony warunek zgodności ilości kolumn macierzy A z ilością wierszy macierzy B.
Przeładowanie operatora [] pozwala uzyskać dostęp do danych zawartych w macierzy.
Funkcja **`empty()`** zwraca **`true`** jeżeli macierz jest pusta.
Funkcja **`numRows()`** zwraca liczbę wierszy macierzy.

Funkcja **numColumns()** zwraca liczbę kolumn macierzy.
[[nodiscard]] informuje kompilator, że wartość zwracana z funkcji nie powinna zostać zignorowana, a jeżeli taka sytuacji nastąpi kompilator zgłosi ostrzeżenie.
noexcept(false) oznacza, że funkcja może potencjalnie wyrzucić wyjątek.

```
int main() {
    Matrix<float> matrix1{{ {2,2,2}, {3,3,3}, {4,4,4} }};
    Matrix<float> matrix2{ {{3,3},{4,4},{5,5}} };

    try {
        auto resultMatrix = matrix1 * matrix2;

        for (size_t i = 0; i < resultMatrix.numRows(); ++i) {
            auto& row = resultMatrix[i];

            for (const auto& el : row) {
                std::cout << el << ' ';
            }
            std::cout << '\n';
        }
    } catch (std::runtime_error& ex) {
        std::cout << ex.what();
    }

    std::cout << std::endl;
}
```

Funkcja **main()** programu. Tworzymy dwa obiekty stworzonej klasy **Matrix**. W bloku try-catch wykonujemy mnożenie macierzy oraz wyświetlamy macierz wynikową. Try-catch pozwala przechwycić wyjątki i obsłużyć je nie doprowadzając do crasha programu.

4. Wyjście programu
    1.

    2.