

REV	DATA	ZMIANY
0.1	13.01.2023	<i>Wiktor Pantak (wpantak@agh.edu.pl)</i>
0.2	14.01.2023	<i>Wiktor Pantak (wpantak@agh.edu.pl)</i>
0.3	18.01.2023	<i>Wiktor Pantak (wpantak@agh.edu.pl)</i>
0.4	22.01.2023	<i>Wiktor Pantak (wpantak@agh.edu.pl)</i>

# Symboliczne Obliczanie Pochodnych Wyrażenia

Języki Programowania Obiektowego

Autor: Wiktor Pantak

Akademia Górniczo-Hutnicza

Kraków 2023

## Spis treści

---

### Spis treści

1.	WSTĘP: .....	4
2.	PODSTAWOWE ZAŁOŻENIA PROJEKTU: .....	5
3.	FUNKCJONALNOŚĆ:.....	6
4.	ANALIZA PROBLEMU: .....	7
5.	PROJEKT TECHNICZNY:.....	9
6.	OPIS REALIZACJI:.....	11
7.	OPIS WYKONANYCH TESTÓW – LISTA BUGÓW, UZUPEŁNIEN, ITP.: .....	12
8.	PODRĘCZNIK UŻYTKOWNIKA:.....	13
9.	METODOLOGIA ROZWOJU I UTRZYMANIA SYSTEM: .....	15
10.	BIBLIOGRAFIA: .....	16

## Lista oznaczeń

CLI	Command-Line Interface
$\sin()$	sinus
$\cos()$	cosinus
$\text{tg}()$	tangens
$x^n$	x podniesione do potęgi n
$\ln()$	logarytm naturalny
$\log_a()$	logarytm o podstawie a
UML	Unified Model Language

Tabela 1 Lista oznaczeń

## 1. Wstęp:

Dokument obejmuje opracowanie systemu symbolicznego obliczania pochodnych wyrażeń jednej zmiennej. Celem tego projektu jest umożliwienie użytkownikowi obliczenia podanych przez niego wyrażeń jednej zmiennej.

## 2. Podstawowe założenia projektu:

- Wykorzystanie tablic matematycznych dotyczących pochodnych funkcji elementarnych.
- Komunikacja programu z użytkownikiem poprzez CLI.
- Opracowanie modułu wykonującego obliczenie pochodnej wyrażeń podanych przez użytkownika, gdzie niewiadomą jest "x".
- Wykorzystanie języka C++, do stworzenia programu.

### 3. Funkcjonalność:

Przewidzianą funkcjonalnością programu jest obliczanie pochodnej wyrażenia z zmienną "x", podanej przez użytkownika w formie tekstu oraz wyświetlenie wyniku również w formie tekstowej w CLI. W późniejszych wersjach programu, planowane jest wprowadzenie obliczanie wartości pochodnej dla podanej przez użytkownika wartości argumentu.

## 4. Analiza problemu:

Założmy, że istnieje dana funkcja  $f(x)$  oraz argument  $x_0$ , który jest określony w otoczeniu  $f(x)$ . Oznaczenie pochodnej przyjmujemy jako:

$$f'(x)$$

Wtedy pochodną definiujemy jako granicę:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Zwaną jako ilorazem różnicowym, czyli przyrostem wartości funkcji względem przyrostu argumentu funkcji. Można wykorzystać tę zależność do wyznaczenia współczynnika kierunkowego funkcji.

Przykładowe wyznaczenie pochodnej funkcji  $f(x)=x$ :

$$f'(x) = \lim_{h \rightarrow 0} \frac{x_0 + h - x_0}{h} = \lim_{h \rightarrow 0} \frac{h}{h} = 1$$

W celu wyliczenia pochodnych wyrażeń jednego argumentu (zmiennej) można wykorzystać wyznaczone wzory pochodnych funkcji elementarnych oraz podstawowe wzory zależności różniczkowanych funkcji.

Zakładając zatem  $f, g, h: \mathbf{R} \rightarrow \mathbf{R}$  będą różniczkowalne na zbiorze otwartym  $U$ , gdzie  $c$  jest stałą, można wykorzystać wtedy podane wzory:

Funkcja	Pochodna	Uwagi
$f \pm g$	$f' \pm g'$	
$c \cdot f$	$c \cdot f'$	
$f \cdot g$	$f' \cdot g + f \cdot g'$	
$f \cdot g \cdot h$	$f' \cdot g \cdot h + f \cdot g' \cdot h + f \cdot g \cdot h'$	
$\frac{f}{g}$	$\frac{f' \cdot g - f \cdot g'}{g^2}$	$g(x) \neq 0$
$\ln(f)$	$\frac{f'}{f}$	
$g(f)$	$g'(f) \cdot f'$	

Tabela 2 Podstawowe wzory.

Pochodne funkcji elementarnych, w podanej tabelce x jest zmienną, a wszystkie pozostałe litery są stałymi:

Funkcja	Pochodna	Uwagi
c	0	
x	1	
$x^n$	$nx^{n-1}$	
$ax+b$	a	
$ax^2+bx+c$	$2ax+b$	
$ax^{-1}$	$-ax^{-2}$	$x \neq 0$
$\sin(x)$	$\cos(x)$	
$\cos(x)$	$-\sin(x)$	
$\operatorname{tg}(x)$	$\frac{1}{\cos^2(x)}$	$x \neq \pi/2 + k\pi, k \in \mathbb{Z}$
$e^x$	$e^x$	
$a^x$	$a^x \ln(a)$	$a > 0$
$x^x$	$x^x(1 + \ln(x))$	$x > 0$
$\ln_a(x)$	$\frac{1}{x}$	$x > 0$
$\log_a(x)$	$\frac{1}{(x \cdot \ln(a))}$	

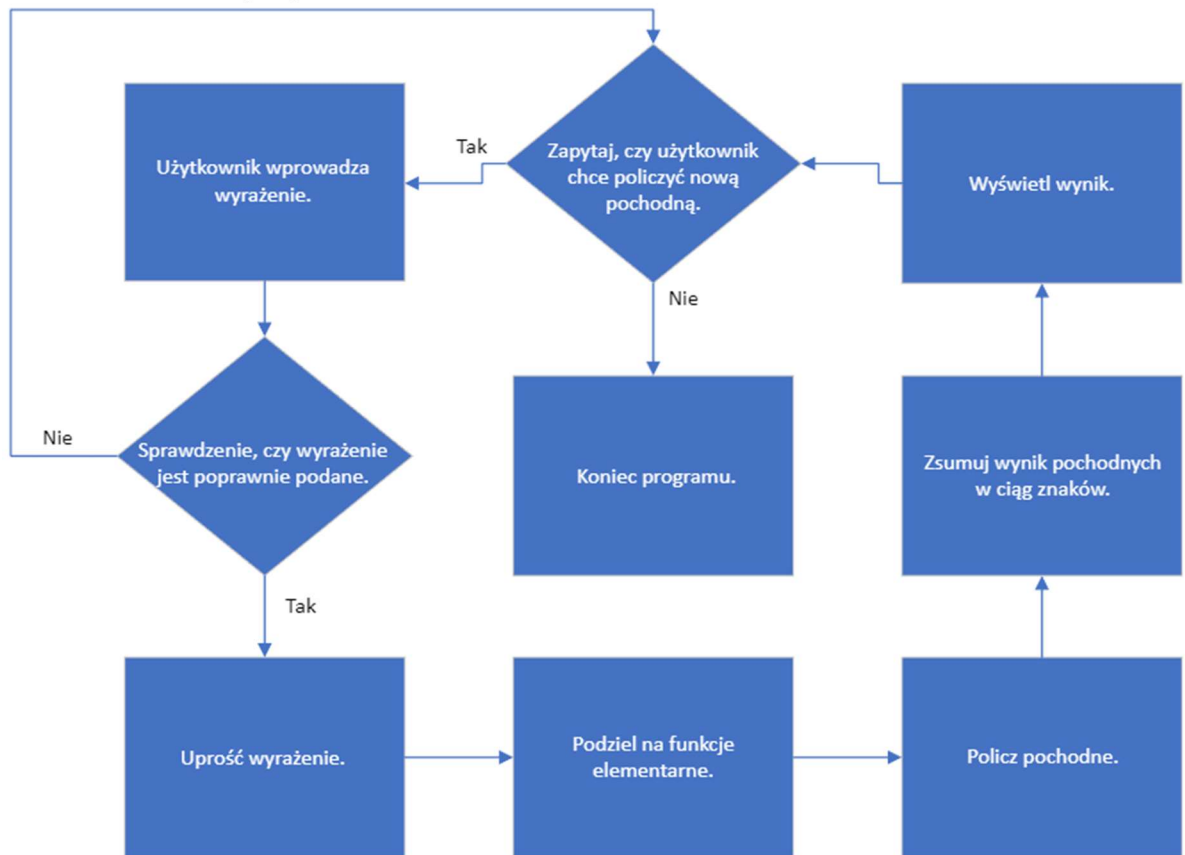
Tabela 3 Pochodne funkcji elementarnych, które są planowane do zaimplementowania w programie.

Celem programu jest podzielenie podanego przez użytkownika wyrażenia na mniejsze części, aby móc wykorzystać wyżej podane pochodne wyrażeń podstawowych, a następnie wyświetlić otrzymany wynik. Dla podanego wyrażenia: " $2x^3 + \log_{10}(3x)$ " będziemy chcieli podzielić je na 2 wyrazy: " $2x^3$ " oraz " $\log_{10}(3x)$ " policzyć z nich pochodne, połączyć je i wyświetlić wynik: " $6x^2 + \frac{3}{(3x \cdot \ln(10))}$ ".



## 5. Projekt techniczny:

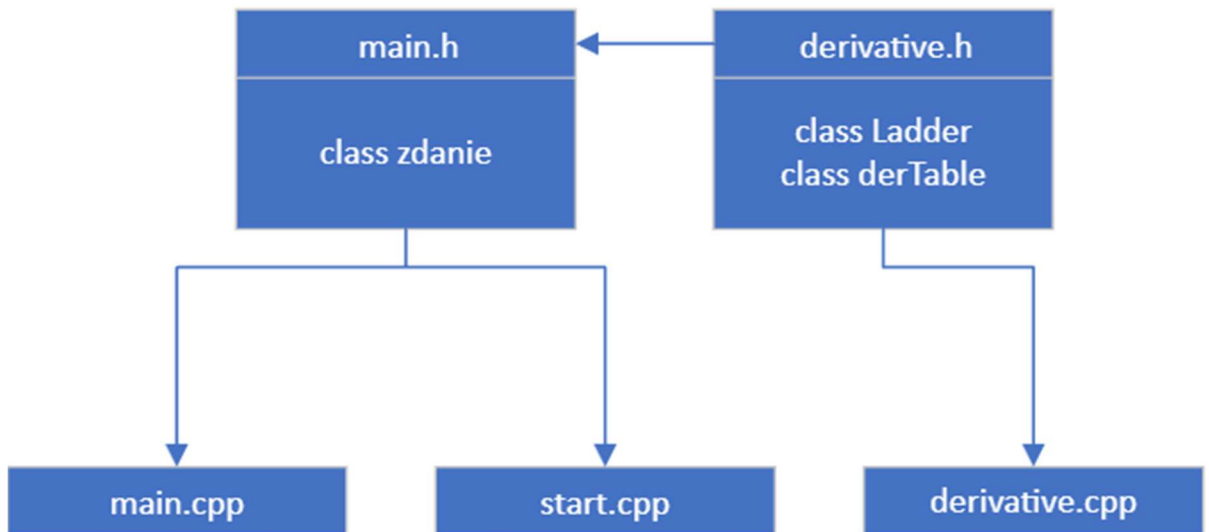
Schemat działania programu:



Rysunek 1 UML schematu działania programu.

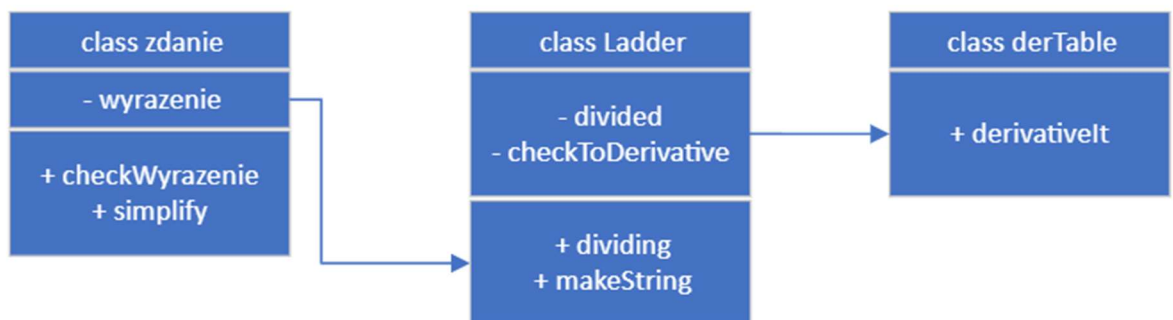
Przy uruchomieniu programu, użytkownik zostanie poproszony o podanie wyrażenia. Jeżeli wyrażenie zostało wprowadzone poprawnie wykona się policzenie pochodnej, a następnie wyświetli się otrzymany wynik. W innym przypadku użytkownik zostanie poinformowany o błędzie w wprowadzonym wyrażeniu i zostanie zapytany, czy chce wprowadzić kolejne.

Relacja bibliotek i plików źródłowych:



Rysunek 2 UML zależności bibliotek oraz plików źródłowych.

Hierarchia klas:



Rysunek 3 UML hierarchi klas projektu.

Do klasy Ladder wysyła się poprzez getter wyrażenie z klasy zdanie, a zmienne divided oraz checkToDerivative z klasy Ladder do klasy derTable, gdzie fragmenty wyrażenia są różniczkowane. Wyrażenie jest uzupełniane w funkcji loop(), która jest zadeklarowana w main.h i opisana w pliku start.cpp .

## 6. Opis realizacji:

Kod przygotowano w środowisku Microsoft Visual Studio Community 2022 ver. 17.4.4 oparty na języku C++20. Wykorzystano kompilator Microsoft® C/C++ ver. Kompilatora optymalizującego 19.33.31630 dla x64.

Program był testowany wykorzystując CLI MicrosoftPowerShell, na komputerze o parametrach:

- system: Windows 11 Home ver. 22H2
- processor: Intel Core i5-9300 CPU 2,40GHz
- 8GB RAM

Wykorzystano github jako system kontroli źródeł: <https://github.com/piktorwa/JPO> .  
Do budowy projektu wykorzystano CMake ver. 3.24.2 .

## 7. Opis wykonanych testów – lista bugów, uzupełnień, itp.:

Funkcja	Uwagi
zdanie::checkWyrazenie()	Poprawnie usuwa spacje z wyrażenia oraz ujednolica liczby zmiennie przecinkowe (zamienia przecinki na kropki). Poprawnie sprawdza ilość nawiasów, nie sprawdza czy nawiasy są podane w odpowiedniej kolejności.
zdanie::simplify()	Poprawnie wymnaża 2 liczby oddzielone znakiem mnożenia “*” oraz zmienną “x”. Nie wprowadzono sumy oraz odejmowania dwóch liczb obok siebie. Nie sprawdza się, czy w wyrażeniu znajdują się inne możliwe kombinacje do uproszczenia oprócz dwóch zachodzących po sobie liczbach.
Ladder::dividing()	Wprowadzono rozdzielanie sumy, różnicy oraz iloczynu pochodnych. Nie wprowadzono rozdzielania ilorazu oraz $\ln()$ . Nie przeprowadzono żadnych testów w celu sprawdzenia poprawności zaimplementowanych rozwiązań
Ladder::makeString()	Wprowadzono sumowanie vector <string> do string, aby następnie zwrócić stworzoną zmienną typu string, jako wynik wyliczonej pochodnej. Nie wykonano testów.
derTable::derivativeIt()	Stworzono funkcję. Nie wprowadzono żadnej funkcji elementarnej, przez co nie możliwe jest policzenie żadnej pochodnej wprowadzonej przez użytkownika.

Tabela 4 Stworzone funkcje w projekcie wraz z uwagami.

## 8. Podręcznik użytkownika:

Sposób wpisywania danych:

Funkcja	CLI	Uwagi
+	+	
-	-	
·	*	
÷	/	
potęga	^	
,	.	Można stosować zamiennie przecinka i kropki, gdyż program podmienia wszystkie przecinki na kropki.
c	liczba z przedziału $0 \leq c \leq 9$	
x	x	
$x^n$	$x^n$	
$ax+b$	<b>ax+b</b>	Można wykorzystać znak mnożenia *.
$ax^2+bx+c$	$ax^2+bx+c$	Można wykorzystać znak mnożenia *.
$ax^{-1}$	$ax^{(-1)}$	Można wykorzystać znak mnożenia *.
$\sin(x)$	$\sin(x)$	
$\cos(x)$	$\cos(x)$	
$\operatorname{tg}(x)$	$\operatorname{tg}(x)$	
$e^x$	$e^x$	
$a^x$	$a^x$	
$x^x$	$x^x$	
$\ln(x)$	$\ln(x)$	
$\log_a(x)$	$\log_a(x)$	

*Tabela 5 Schemat wpisywania wyrażeń przez CLI.*

Po uruchomieniu programu, użytkownik zostanie poproszony o podanie wyrażenia, następnie jeżeli wyrażenie zostało podane poprawnie, zostanie policzona pochodna i wyświetlona. Przykładowe wyrażenie wpisane w CLI, według składni podanej w tabelce 5:

`"3x^2+sin(x)+cos(x^2*ln(x))"`

Kolejnym krokiem programu będzie zapytanie użytkownika, czy chce podać nowe wyrażenie:  
**Czy chcesz podać nowe wyrażenie? (t/n)**

Jeżeli użytkownik wpisze n, program zamknie się po 1 sekundzie. W przypadku wpisania jakiegokolwiek innego znaku, będzie wyświetlona prośba o podanie nowego wyrażenia, a następnie będzie wykonana operacja opisana w pierwszym akapicie tego punktu.

## 9. Metodologia rozwoju i utrzymania system:

Głównym celem rozwoju i utrzymania systemu jest dokończenie programu, aby osiągnąć cel postawiony na samym początku projektu, czyli obliczanie pochodnej wyrażenia jednej zmiennej. Następnym krokiem może być rozszerzenie projektu o wyliczanie wartości pochodnej wyrażenia dla argumentu podanego przez użytkownika.

W kolejnych rozszerzeniach można stworzyć obliczanie pochodnych wielu zmiennych lub wykorzystać otrzymany już kod w stworzeniu systemu obliczania pochodnych cząstkowych.

## 10. Bibliografia:

1. Cyganek B. : Introduction to programming with C++ for engineers, Wiley, 2021.
2. [https://pl.wikisource.org/wiki/Tablica\\_pochodnych](https://pl.wikisource.org/wiki/Tablica_pochodnych)
3. [https://www.cs.csustan.edu/~john/Courses/Previous\\_Semesters/CS3100\\_DataStructures/2003\\_04\\_Fall/Asg04/DiffAsg.html](https://www.cs.csustan.edu/~john/Courses/Previous_Semesters/CS3100_DataStructures/2003_04_Fall/Asg04/DiffAsg.html)
4. <https://www.geeksforgeeks.org/>
5. <https://cplusplus.com/>
6. <https://www.matemaks.pl/pochodne.html>