**Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра обчислювальної техніки**

**Лабораторна робота №1**

з дисципліни  
«Алгоритми і структури даних»

Виконав: Перевірила:

студент групи ІМ-31 Молчанова А. А.  
Литвиненко Сергій Андрійович  
номер у списку групи: 12

Київ 2024

**Завдання**

Дане натуральне число n. Знайти суму перших n членiв ряду чисел, заданого рекурентною формулою. Розв’язати задачу **трьома способами**:

1. у програмі використати рекурсивну функцiю, яка виконує обчислення i членiв ряду, i суми на рекурсивному спуску;
2. у програмі використати рекурсивну функцію, яка виконує обчислення i членiв ряду, i суми на рекурсивному поверненнi;
3. у програмі використати рекурсивну функцію, яка виконує обчислення членiв ряду на рекурсивному спуску, а обчислення суми на рекурсивному поверненні.

При проєктуваннi програм слiд врахувати наступне:

1. програми повинні працювати коректно для довiльного цiлого додатного n включно з n = 1;
2. видимiсть змiнних має обмежуватися тими ділянками, де вони потрібні;
3. функції повинні мати властивість модульності;
4. у кожному з трьох способів рекурсивна функція має бути одна (за по-

треби, можна також використати додаткову функцiю-обгортку (wrapper function));

1. у другому способі можна використати запис (struct) з двома полями

(але в інших способах у цьому немає потреби i це вважатиметься надлишковим);

1. програми мають бути написані мовою програмування С.

**Варіант 12**

**Текст програм**

1. Рекурсивну функція, яка виконує обчислення i членiв ряду, i суми на рекурсивному спуску.

#include <stdio.h>

double wrapMyCos(double x, unsigned i, unsigned n, double previos, double totalSum) {

if (i >= n) return totalSum;

const double next = -previos \* (x \* x) / (4 \* i \* i - 2 \* i);

const double sum = totalSum + next;

return wrapMyCos(x, i + 1, n, next, sum);

}

double myCos(double x, unsigned n) {

return wrapMyCos(x, 1, n, 1.0, 1.0);

}

int main(int argc, char const \*argv[]) {

const unsigned n = 5;

double x;

printf("Enter x: ");

scanf("%lf", &x);

const double res = myCos(x, n);

printf("cos(%lf) = %lf\n", x, res);

return 0;

}

1. Рекурсивна функція, яка виконує обчислення i членiв ряду, i суми на рекурсивному поверненнi.

#include <stdio.h>

double wrapMyCos(double x, unsigned n, double\* previos) {

if (n <= 1) return \*previos;

const double totalSum = wrapMyCos(x, n - 1, previos);

const unsigned i = n - 1;

const double next = -\*previos \* (x \* x) / (4 \* i \* i - 2 \* i);

const double sum = totalSum + next;

\*previos = next;

return sum;

}

double myCos(double x, unsigned n) {

double previos = 1.0;

return wrapMyCos(x, n, &previos);

}

int main(int argc, char const \*argv[]) {

const unsigned n = 5;

double x;

printf("Enter x: ");

scanf("%lf", &x);

const double res = myCos(x, n);

printf("cos(%lf) = %lf\n", x, res);

return 0;

}

1. У програмі використати рекурсивну функцію, яка виконує обчислення членiв ряду на рекурсивному спуску, а обчислення суми на рекурсивному поверненні.

#include <stdio.h>

double wrapMyCos(double x, unsigned i, unsigned n, double previos) {

if (i >= n) return 1.0;

const double next = -previos \* (x \* x) / (4 \* i \* i - 2 \* i);

const double totalSum = wrapMyCos(x, i + 1, n, next);

const double sum = next + totalSum;

return sum;

}

double myCos(double x, unsigned n) {

return wrapMyCos(x, 1, n, 1.0);

}

int main(int argc, char const \*argv[]) {

const unsigned n = 5;

double x;

printf("Enter x: ");

scanf("%lf", &x);

const double res = myCos(x, n);

printf("cos(%lf) = %lf\n", x, res);

return 0;

}

1. Циклічний варіант вирішення задачі.

#include <stdio.h>

double myCos(double x, unsigned n) {

double res = 1.0;

double previos = res;

for (int i = 1; i < n; i++) {

previos = -previos \* (x \* x) / (4 \* i \* i - 2 \* i);

res += previos;

}

return res;

}

int main(int argc, char const \*argv[]) {

const unsigned n = 5;

double x;

printf("Enter x: ");

scanf("%lf", &x);

const double res = myCos(x, n);

printf("cos(%lf) = %lf\n", x, res);

return 0;

}

**Тестування програм**

1. Рекурсивну функція, яка виконує обчислення i членiв ряду, i суми на рекурсивному спуску.









1. Рекурсивна функція, яка виконує обчислення i членiв ряду, i суми на рекурсивному поверненнi.

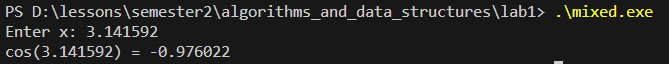


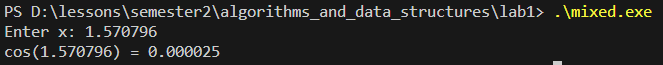




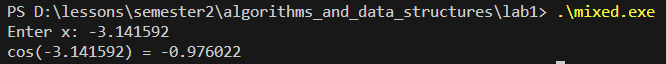


1. Рекурсивна функція, яка виконує обчислення членiв ряду на рекурсивному спуску, а обчислення суми на рекурсивному поверненні.

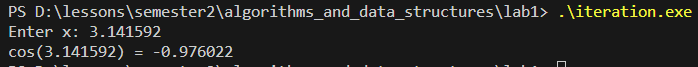




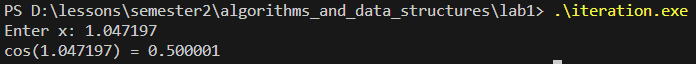




1. Циклічний варіант вирішення задачі.

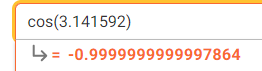


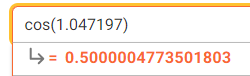
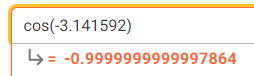






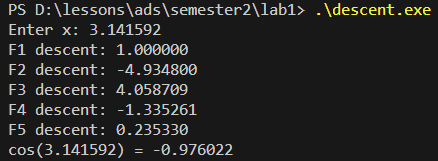
Перевірка на калькуляторі:

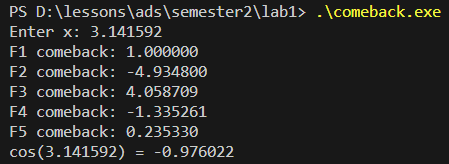
 

Щоб перевірити правильнiсть обчислень елементiв ряду, я вивів на екран відповідну величину та порівняв її з власними розрахунками.

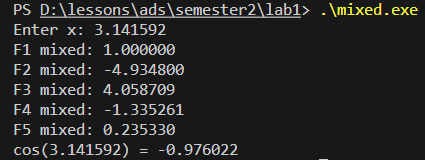
1. Рекурсивну функція, яка виконує обчислення i членiв ряду, i суми на рекурсивному спуску.



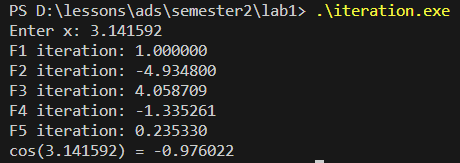
1. Рекурсивна функція, яка виконує обчислення i членiв ряду, i суми на рекурсивному поверненнi.



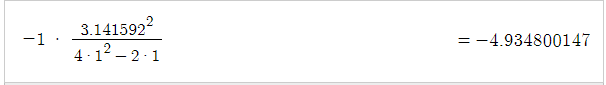
1. Рекурсивна функція, яка виконує обчислення членiв ряду на рекурсивному спуску, а обчислення суми на рекурсивному поверненні.

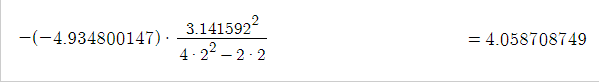


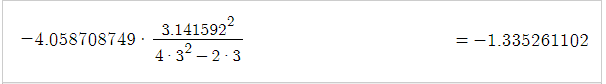
1. Циклічний варіант вирішення задачі.

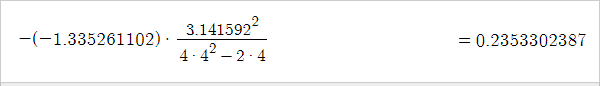


За власними розрахунками:









Сума також співпадає з обчисленнями програми.



Отже, програми працюють правильно.

**Графік похибки обчислення функції**

**Висновок**

Протягом виконання лабораторної роботи я навчився розробляти рекурсивні алгоритми. Було розроблено 4 програми для апроксимації функції cos. Точність обчислення функції за допомогою апроксимації залежить від кількості складених доданків, тож чим більший параметр n в функції, що апроксимує, тим більш точним є результат обчислення. Даний метод апроксимації розкладає функцію cos в околі нуля, тож чим ближче значення аргументу знаходиться до нуля, тим менше ітерацій або рекурсивних викликів знадобиться для обчислення точного значення функції. І навпаки, для того, щоб обчислити значення функції в точці, що знаходиться далеко від нуля, треба більша кількість ітерацій або рекурсивних викликів, що гарно видно на графіку, представленому вище.

Рекурсивні алгоритми інколи є більш наглядними та короткими, ніж їх ітеративний варіант, особливо коли задача поставлена рекурентним відношенням. Вони є незамінними в тих випадках, коли алгоритм майже неможливо представити ітераційним способом. Проте, рекурсивні алгоритми часто є повільнішими за їх ітеративний аналог, адже витрачаються додаткові ресурси для виклику функції та поверненню з неї. Також рекурсивні алгоритми вимагають більше пам’яті через зберігання контексту функції.