Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний

інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 6 з дисципліни

«Алгоритми та структури даних-1.

Основи алгоритмізації»

«Дослідження рекурсивних алгоритмів»

Варіант 5

Виконав студент ІП-12, Василишин Михайло Михайлович

(шифр, прізвище, ім'я, по батькові)

Перевірив Василишин Михайло Михайлович

( прізвище, ім'я, по батькові)

Київ 2021

**Лабораторна робота №6** «Дослідження рекурсивних алгоритмів»

**Мета** – дослідити особливості роботи рекурсивних алгоритмів та набути практичних навичок їх використання під час складання програмних специфікацій підпрограм.

**Варіант** – 5

**Задача №5**.

Обчислити суму 6 елементів геометричної прогресії, що зростає: початкове значення – 2, крок – 2

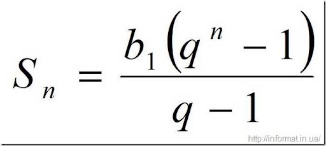
**Розв’язок**

1. **Постановка задачі.**

Результатом розв’язку є числова величина, що дорівнює сумі вказаних в умові задачі елементів геометричної прогресії . Оскільки всі необхідні дані для обчислення вказані в умові задачі, то побудова математичної моделі зводиться до деталізації алгоритму знаходження суми елементів геометричної прогресії математичними та алгоритмічними засобами. Для визначення результату будуть використані проміжні змінні. Початкових даних для розв’язку не потрібно.

1. **Побудова математичної моделі.** Складемо таблицю імен змінних.

|  |  |  |  |
| --- | --- | --- | --- |
| Змінна | Тип | Ім’я | Призначення |
| Перший член | Дійсний | b1 | Початкове дане |
| Крок прогресії | Дійсний | q | Початкове дане |
| Кількість елементів прогресії | Цілий | n | Початкове дане |
| Число у парному степені | Дійсний | temp | Проміжне дане |
| Значення, яке повертає функція | Дійсний | res | Проміжне дане |
| Відповідь до задачі | Дійсний | s | Результату |

Знаходження суми членів геометричної прогресії може бути реалізоване за допомогою формули: 

Бачимо, що всі змінні відомі з умови задачі, результат визначається шляхом використання арифметичних операцій та піднесенням до степеня. Піднесення до степеня може організувуватись рекурсивною функцією, де не враховуючи умову виходу буде такий перехід: . Помітимо, що кількість множень – O(n). Для оптимізації алгоритму використаємо алгоритм повторюваного піднесення до квадрата(*exponentiating by squaring)* . Опишемо цей алгоритм, щоб показати кількість множень для нього.

Помітимо, що для будь якого числа а та парного числа n виконується рівність:  a^n = (a^{n/2})^2 = a^{n/2} \cdot a^{n/2} . Ця рівність є основою для вказаного алгоритму, адже для парного n ми зменшуємо кількість множень в 2 рази!

Виокремемо випадок, коли n – непарне. Помітимо, що:  a^n = a^{n-1} \cdot a ,

де має парний степінь, міркування для якого уже описані. Тому чітко прослідковується рекурсивна структура алгоритму.

Зрозуміло, що кількість множень залежить від парності n, тому кількість множень с = Z + 2\*(E-1), де Z – кількість нулів у двійковій інтерпритації числа, а E – кількість одиниць. Таким чином, кількість множень – O(log(n)), що менше ніж O(n), а тому будемо використовувати алгоритм повторюваного піднесення до квадрата. Рекурсивні переходи визначуються відповідно до вказаних рівностей для парних і непарних n, умова виходу – (n = 1).

**Крок 1.** Визначимо основні дії.

**Крок 2.** Деталізуємо знаходження суми членів геометричної прогресії

**Крок 3.** Деталізуємо функцію funcPow()

1. **Псевдокод алгоритму**

Крок 1

**початок**

Знаходження суми членів геометричної прогресії

**кінець**

**функція** funcPow()

Крок 2

**початок**

b1 = 2.0

q = 2.0

n = 6

S = b1\*(funcPow(q,n)-1) / (q-1);

**вивід** S

**кінець**

**функція** funcPow()

Крок 3

**початок**

b1 = 2.0

q = 2.0

n = 6

S = b1\*(funcPow(q,n)-1) / (q-1);

**вивід** S

**кінець**

**початок функції funcPow( q, n)**

res = 0

**якщо** n == 0

**то** res = 1

**інкше**

**якщо** n%2 == 1

**то** res =q\* funcPow(q,n-1)

**все якщо**

**якщо** n%2 == 0

**то**

temp = funcPow(q,n/2)

res = temp\*temp

**все якщо**

**все якщо**

**повернути** res

**кінець функції funcPow( q, n)**

1. **Блок-схема**

Крок 1



Крок 2



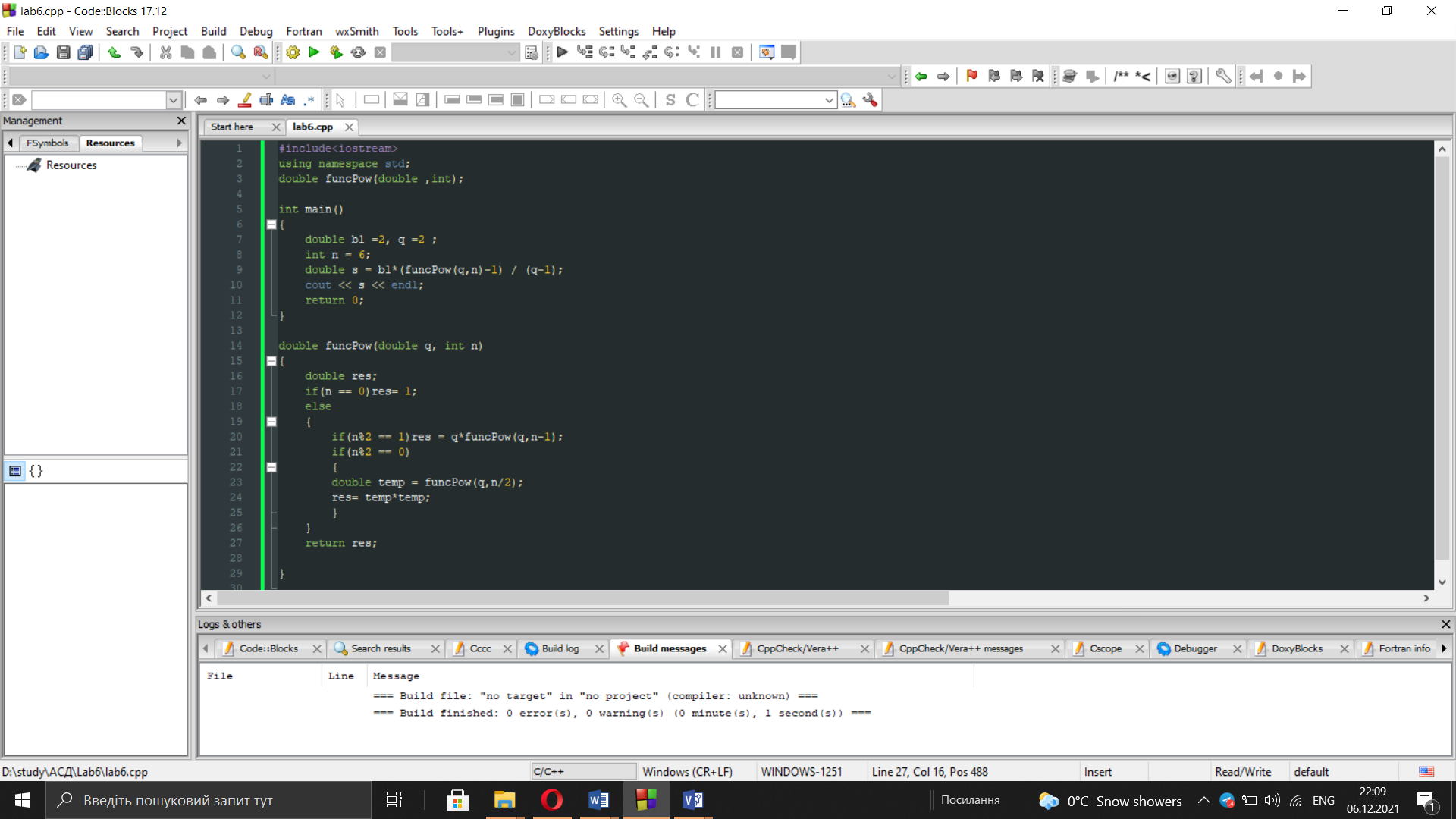
Крок 3

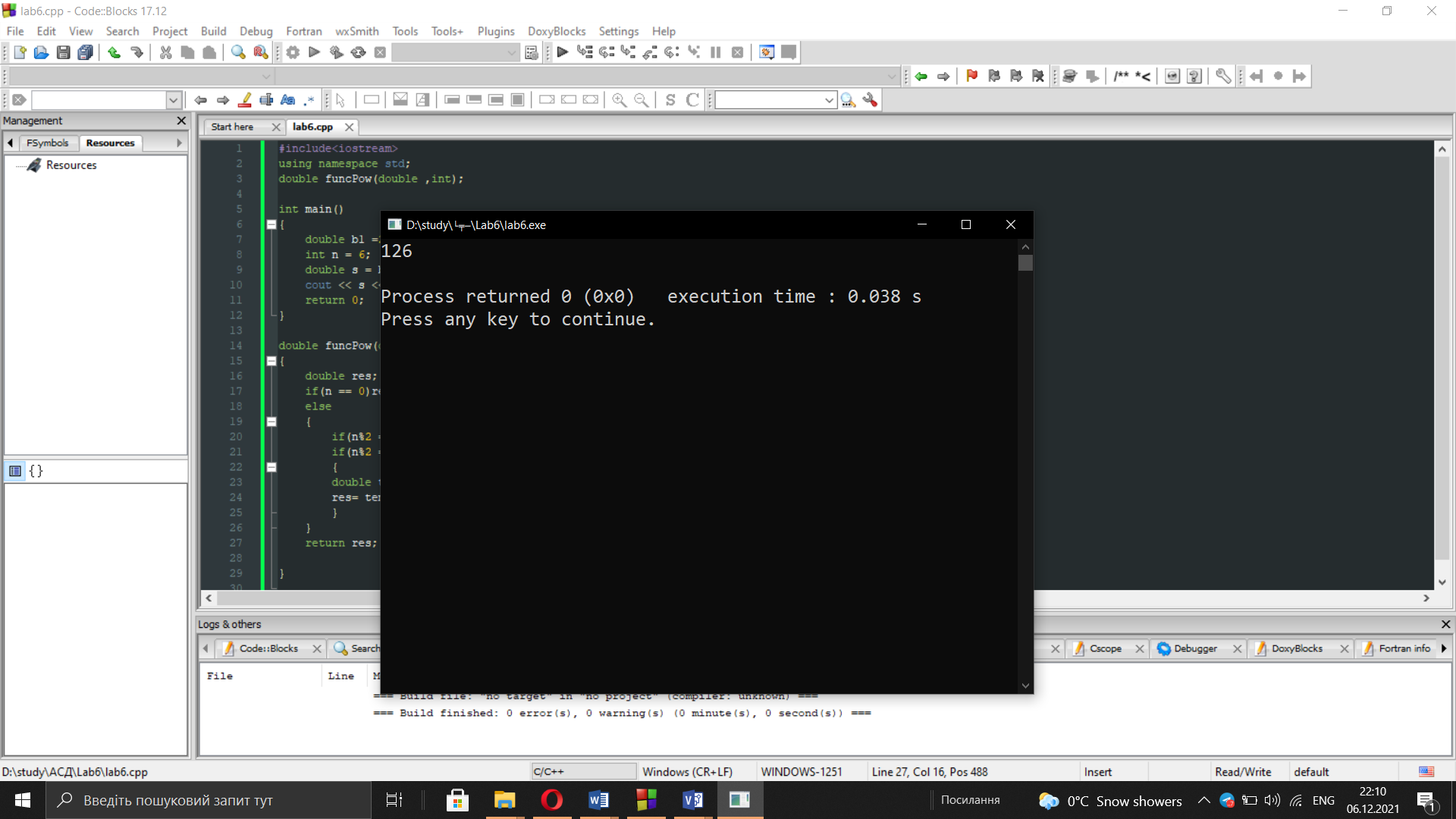


1. **Випробування алгоритму**

Перевіримо привильність роботи алгоритму на одному тестовому наборі(адже вхідних даних немає).

|  |  |
| --- | --- |
| Блок | Дія |
|  | Початок |
| 1 | funcPow(2.0,6) = funcPow(2.0,3)\*funcPow(2.0,3) |
| 2 | funcPow(2.0,3) = 2.0\*funcPow(2.0,2) |
| 3 | funcPow(2.0,2) = funcPow(2.0,1)\*funcPow(2.0,1) |
| 4 | funcPow(2.0,1) = 1.0 \* funcPow(2.0,0) |
| 5 | funcPow(2.0,0) = 1.0 |
|  | (Повернення по стеку) |
| 6 | funcPow(2.0,1) = 1.0\*2.0 = 2.0 |
| 7 | funcPow(2.0,2) = 2.0\*2.0 = 4.0 |
| 8 | funcPow(2.0,3) = 2.0\*4.0 = 8.0 |
| 9 | funcPow(2.0,6) = 8.0\*8.0 = 64.0 |
| 10 | S = 2.0 \* (64.0 -1) / (2.0-1) = 126.0 |
|  | Кінець |





Результати перевірки збігаються із результатом виведення програми, що реалізовує алгоритм.

1. **Висновки**

На даній лабораторній роботі було досліджено особливості роботи рекурсивних алгоритмів та набуто практичних навичок їх використання під час складання програмних специфікацій підпрограм. Особливістю моєї роботи було використання оптимізованого алгоритму повторюваного піднесення до квадрата для визначення цілого степеню числа напритовагу звичайної рекурсивної функції з складністю O(n). Таким чином було вдосконалено навички з розв’язування задач за допомогою алгоритму повторюваного піднесення до квадрата та закріплено його практичне застосування. Отримано досвід з побудови альтернативних оптимізнованих алгоритмів в контексті рекурсивних функцій.