Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка програмної системи управління ігровими рушіями на основі скриптової мови

Виконав: студент IV курсу
групи 2ПІ-18б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Доценко Д. В.
(прізвище та ініціали)

Керівник: к.т.н., доц. Рейда О. М.
(прізвище та ініціали)

Рецензент: д.т.н., проф. Василевський О.М.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«____» _____ 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
_____

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Доценка Дмитра Володимировича

1. Тема роботи – «Розробка програмної системи управління ігровими рушіями на основі скриптової мови»

Керівник роботи: Рейда Олександр Миколайович , к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 року №66.

2. Строк подання студентом роботи 13 червня 2022 року.

3. Вихідні дані до роботи: Середовище розробки – CLion, Мови розробки – C++, Операційна система – Windows 10.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; розробка архітектури та алгоритмів програмного додатка; розробка програмного додатку; тестування додатку, висновки; список використаних джерел, додатки, графічна частина.

5. Перелік графічного матеріалу: актуальність розробки, головні задачі дослідження, новизна отриманих результатів, практична цінність, діаграма класів, блок-схема алгоритму виклику функцій, обробка ключових слів, ініціалізація даних, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Рейда О.М., к.т.н., доцент кафедри ПЗ		

7. Дата видачі завдання 25 березня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	26.03.2022 – 10.04.2022	Вик.
2	Розробка архітектури та алгоритмів роботи системи	11.04.2022 – 19.04.2022	Вик.
3	Вибір середовища та мови розробки	20.04.2022 – 27.04.2022	Вик.
4	Розробка програмного продукту	28.04.2022 – 10.05.2022	Вик.
5	Тестування роботи системи	11.05.2022 – 19.05.2022	Вик.
6	Оформлення матеріалів до захисту БДР	20.05.2022 – 10.06.2022	Вик.

Студент _____ **Доценко Д. В.**

(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Рейда О.М.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 60 сторінок формату А4, на яких є 41 рисунок, 2 таблиць, список використаних джерел містить 10 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз скриптових мов програмування, що використовуються в ігрових рушіях, розроблено структуру даних, алгоритми функціювання та інтерпретатор коду та спроектовано систему додання базових функцій. Сформульовано мету досліджень – розробка скриптової мови програмування для підвищення рівня інтегрування авторського коду у системі керування ігрового рушія для подальшого модифікування та оновлення.

Запропоновано метод використання функціональної мови програмування сценаріїв, що інтерпретується для керування систем ігрових рушіїв, особливість якої полягає в легкій інтеграції в існуючий проект, метод підвищення рівня інтегрування авторського коду у системі керування ігрового рушія за рахунок зменшення необхідної кількості операцій для виконання та надання додаткових можливостей. Розроблено розроблено скриптову мову програмування, що може бути ефективно інтегрована в ігровий рушій з можливістю швидкого доповнення мови програмування та виконуватись самостійно.

Розроблено алгоритми та програмний засіб для генерування програмного коду.

Отримані в бакалаврській дипломній роботі результати можна використати для побудови доповнень і модифікацій ігрових рушіїв..

Ключові слова: скриптова мова програмування, ігровий рушій, легка інтеграція.

ABSTRACT

The bachelor's thesis consists of 60 A4 pages format, which have 41 figures, 2 tables, a list of used sources contains 10 titles.

The bachelor's thesis contain a detailed analysis of scripting languages used in game engines and developed a data structure, functional algorithms and code interpreter and designed a system for adding basic functions. The purpose of the research is formulated – development of a scripting language for increasing the level of integration of author's code in the game engine control system for further modification and updating.

Proposed the method of using the functional scripting programming language, which is interpreted for the management of game engine systems, the feature of which is easy integration into the existing project. The method of increasing the level of integration of author's code in the control system of the game engine by reducing the required number of operations to perform and provide additional capabilities. Developed a scripting language that can be effectively integrated into the game engine with the ability to quickly modify the programming language and run independently.

Algorithms and software for generating software code have been developed.

The results obtained in the bachelor's thesis can be used to build additions and modifications to game engines.

Keywords: scripting language, game engine, easy integration.

ЗМІСТ

ВСТУП

Для забезпечення швидкодії і підвищення ефективності системи управління ігровими рушіями на основі скриптової мови необхідно використати велику кількість ресурсів, часу на реалізацію та перевірку, що призводить до зниження ефективності роботи по удосконаленню і модернізації ігрових рушіїв.

Таким чином, існуючі методи і засоби системи управління ігровими рушіями на основі скриптової мови характеризуються суттєвим використанням ресурсів і обчислювальною складністю, що в значній мірі впливає на конкурентну спроможність ігрового рушія.

Тому актуальними є питання підвищення швидкості інтегрування авторського коду для використання у ігровому рушієві, що стимулює зацікавленість в використанні додатка та надає можливість для поліпшення програмного продукту силами користувачів.

Надаючи можливість створення модифікацій розробник збільшує час активного використання та популярність продукту, оскільки кожний користувач зможе створити ідеальні для себе умови подальшого використання. Також це збільшує обмін інформації між користувачами програми [1].

Більшість всесвітньо відомих ігор підтримують модифікації, для прикладу Minecraft, Terraria, Dota2, CS:GO, серія ігор TES, починаючи з 3, з найбільш відомою Skyrim, Warcraft 3. Навіть не зважаючи, що більшість з цих ігор вийшло більше 10 років тому вони досі відомі і підтримують велику кількість активних користувачів за відсутності регулярних оновлень, що добавляють новий контент. Для прикладу в грі Skyrim, 2011 року виходу, за останні 5 років вийшло 3 оновлення, що в більшості спрямовані на виправлення помилок та додання предметів для гравця, підтримує стабільну кількість користувачів [2] та знаходиться на 51-му місці в списку поточної кількості гравців, на момент написання [3].

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Мета бакалаврської дипломної роботи полягає в розробці скриптової мови програмування для підвищення рівня інтегрування авторського коду у системі керування ігрового рушія для подальшого модифікування та оновлення.

Головні задачі дослідження є:

- провести аналіз існуючих скриптових мов програмування, що використовуються в ігрових рушіях, як метод створення модифікацій;
- запропонувати нові:
  - метод використання функціональної мови програмування сценаріїв, що інтерпретується для керування систем ігрових рушіїв, особливість якої полягає в легкій інтеграції в існуючий проект;
  - метод підвищення рівня інтегрування авторського коду у системі керування ігрового рушія;
- розробити область видимості об’єктів, системи їх ініціалізації та видалення з підтримкою рекурсивних функцій;
- розробити інтерпретатор скриптового коду з підтримкою пріоритетності операторів;
- провести експериментальні дослідження розроблених засобів текстурування.

Об’єкт дослідження – процес розробки програмної системи управління ігровими рушіями на основі скриптової мови.

Предмет дослідження є методи та засоби розробки системи управління ігровими рушіями на основі скриптової мови.

Методи дослідження. У процесі розробки використовувались методи: інтерпретації коду мовами програмування; управління комп’ютерною пам’яттю;

створення бінарного дерева; організації знакових операторів та теорія комп’ютерної системи числення.

Наукова новизна отриманих результатів:

- уперше запропоновано метод використання функціональної мови програмування сценаріїв, що інтерпретується для керування систем ігрових рушіїв, особливість якої полягає в легкій інтеграції в існуючий проект;
- уперше запропоновано метод підвищення рівня інтегрування авторського коду у системі керування ігрового рушія за рахунок зменшення необхідної кількості операцій для виконання та надання додаткових можливостей.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі розроблено скриптову мову програмування, що може бути ефективно інтегрована в ігровий рушій з можливістю швидкого доповнення мови програмування та виконуватись самостійно.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати аналіз способів ініціалізації даних.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на: Науково-технічна конференція підрозділів Вінницького національного технічного університету (Вінниця 2022).

Публікації. Основні результати досліджень опубліковано у матеріалах конференції.

1 ОБҐРУНТУВАННЯ РОЗРОБКИ

1.1 Аналіз предметної області

Скриптова мова програмування – це мова програмування, що в більшості випадків працює в реальному часі та автоматизує виконання операцій згідно «сценарію» [4].

Весь сценарій це набір команд, які використовуються для виконання базових операцій, наприклад додавання, ініціалізація об’єктів і так дальше, або виклику API функцій.

Мови програмування сценаріїв характеризуються легкістю вивчення та написання, сценарії представлені малими окремими файлами та виконуються від початку файлу до кінця без необхідності головної функції.

Використання мов програмування, що інтерпретується поширюються з часом все більше. Написання серверів на Python часта практика, що в нікого не визиває питань, навіть не зважаючи на швидкість виконання. Причина в законі Мура – спостереження згідно якому, що кожних 2 роки подвоюється кількість транзисторів на мікросхемі [5]. Тому з кожним роком використання низькорівневих мов програмування та деяких високорівневих зменшується зі збільшенням використання скриптових та швидких в написанні високорівневих мов програмування. На сьогоднішній день не сильно звертають увагу на використання процесора, а на можливість швидкої розробки. Що зумовлює популярність скриптових мов програмування.

Існує декілька типів скриптових мов програмування.

З’єднувальна мова програмування. В більшості випадків з’єднувальна мова програмування використовується, щоб поєднати різні компоненти системи, наприклад використання бази даних, робота з системою, виконання файлів, робота з сервером. Вони зумовлені необхідністю швидкого способу вирішення не значної проблеми.

Мови редагування тексту. Це мови програмування, що використовуються, як макроси текстовими редакторами, вони спеціалізуються на швидкій роботі з текстом та надають широкі можливості для його редагування.

Оболонкові мови програмування та менеджери процесів. Оболонкові скрипти та скрипти менеджерів процесів відповідають за роботу з операційною системою та контролем виконання програми.

Мови графічного інтерфейсу. Вони дозволяють взаємодіяти з вікнами, як користувач за допомогою коду скрипта, для прикладу натисну зелену кнопку, закрити спливаюче вікно. В більшості випадків використовуються для автоматизації дій користувача.

Мови вбудовані в додаток. Це окремі мови програмування, що створені виключно для потреби окремого програмного продукту, може бути представлена в вигляді модифікованої мови програмування, наприклад Emacs Lisp.

Мови з можливістю інтеграції. Це мови програмування, що були розроблені з задумкою інтеграції в інші програмні додатки. Розробник базової програми додає точки в яких виконуються певний скрипт, наприклад при завершені відкриття програми. Зазвичай використовується для швидкої модифікації продукту без зміни машинного коду. Самий відомий приклад JavaScript.

Більшість скриптових мов програмування, належать до декілька типів, наприклад TCL, що належить до зв’язувальних мов та мов з можливістю інтеграції.

Для створення бакалаврської дипломної роботи доцільно створити мову з можливістю інтеграції загального призначення.

1.2 Порівняльний аналіз аналогів

Для вирішення проблеми створення системи управління ігровими рушіями існує декілька аналогових мов програмування, що інтерпретується.

JavaScript – на сьогоднішній час сама відома скриптова мова програмування для розробки веб сторінок, але також може бути використаний в ігрових рушіях (див. рис. 1.1). Надає широкі можливості при написанні програм та має чудову швидкість виконання.

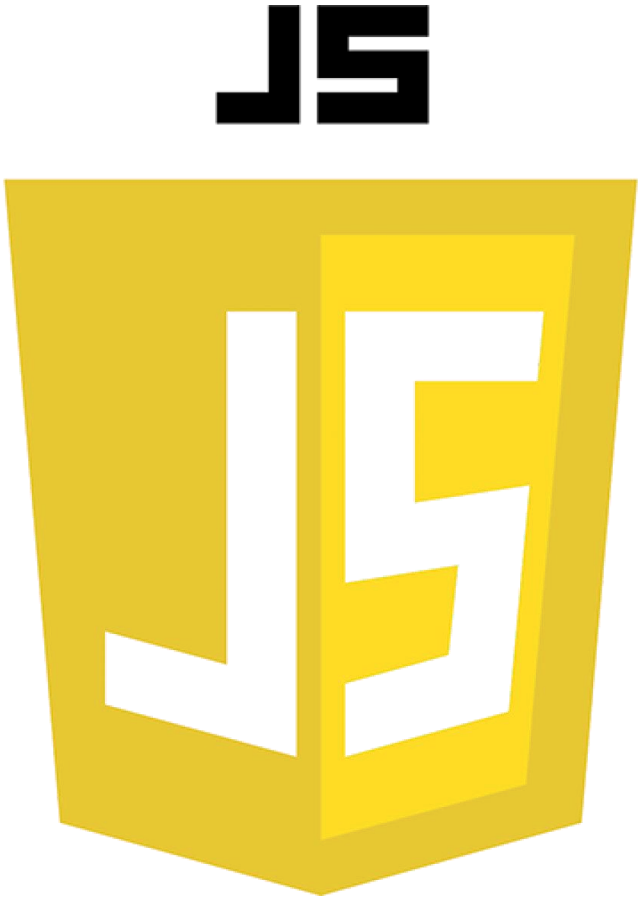

Рисунок 1.1 – Зображення логотипу JavaScript

Існує два інтерпретатора для JS:

V8 – це рушій JavaScript, що був розроблений Google і поширюється з відкритим кодом. З мінусів великий розмір мінімального інтерпретатора 160 MB на Windows. Висока складність інтеграції в існуючу програму.

WebKit – це веб рушій, що призначений для роботи в браузерах. Займає великий розмір та складно інтегрується.

ActionScript – це скриптова мова програмування, що дозволяє програмувати в середовищі Adobe Flash, легко в вивчені, швидкість виконання (див. рис. 1.2).

З найбільших недоліків це відсутність підтримки, останнє оновлення 2008 року і відсутність підтримки сучасними браузерами. Також потребує від користувача використання плеєра або інтеграції в рушій неофіційного інтерпретатора.

Рисунок 1.2 – Зображення логотипу ActionScript

Lua – відома скриптова мова програмування, що використовується в ігрових рушіях (див. рис. 1.3).

Рисунок 1.3 – Зображення логотипу Lua

Може бути використаний в великій кількості сфер, має не погану швидкодію та швидкий в вивчені, регулярно отримує оновлення.

З мінусів не зручна та інколи не зрозуміла інтеграція в рушій, наявність помилок повернення даних, відсутність підтримки опрацювання помилок, відсутність підтримки Unicode.

Python – це популярна мова програмування, що інтерпретується, використовується в більшості галузей розробки програмних додатків (див. рис. 1.4). Легкий в вивчені, але складний в вивчені до професійного рівня.

Рисунок 1.4 – Зображення логотипу Python

Зручний для виклику функцій написаних на C та C++, має не погану швидкодію. Але дуже важкий для інтеграції в існуючий проект.

Для проведення порівняльного аналізу мов програмування, що інтерпретується визначимо головні порівняльні характеристики: легка інтеграція, наявність підтримки, легкість модифікації, система опрацювання помилок, підтримка Unicode.

Проведено порівняння аналогів та занесено результати в таблицю 1.1.

Таблиця 1.1 – Результати порівняння мов програмування

	JavaScript	ActionScript	Lua	Python	Spigine
Можливість інтеграції	0	0	1	0	1
Можливість виконання в різних середовищах	1	0	1	1	1
Можливість модифікації	0	1	1	0	1
Система опрацювання помилок	1	0	0	1	0
Підтримка Unicode	1	1	0	1	1
Загалом	3	2	3	3	4

Згідно проведеного аналізу, результати якого показано в таблиці, доведено актуальність розробки, оскільки програмний додаток буде надавати більше можливостей ніж його аналоги.

1.3 Аналіз методів розв’язання поставленої задачі

Для створення інтерпретатора для рушія мови програмування сценаріїв можливо використання одиного з двох методів виконання коду.

Байт-код. Перетворення тексту програми в компактний набір інструкцій у форматі чисел, що представляють собою закодовані інструкції компілятора і виконаного семантичного аналізу [6].

В основному використовується з метою підвищення швидкості виконання коду програми, але потребує додаткового часу для компіляції коду, що може викликати додаткові помилки при виконанні та написанні коду, що модифікує сам себе.

Для виконання байт-коду потрібне використання віртуальної машини, або подальше перетворення на машинний код.

Проста інтерпретація, використовує код сценарію, що зрозумілий для людини, за допомогою використання інтерпретатора без потреби компіляції перед виконанням.

Потребує більше часу на виконання обчислень, але дозволяє більше можливостей при створенні програм та пришвидшує розробку. Зазвичай використовується в ситуаціях, що не потребують високої оптимізації, або за потреби модифікації без повторної компіляції проекту.

Для надання більших можливостей при написанні програмного забезпечення та уникненню потреби компіляції коду використано проста інтерпретація для реалізації рушія мови програмування “Spigine”.

Для використання інтерпретатором коду, було прийнято рішення провести аналіз методі кодування для написання коду користувачем.

ASCII – це американський стандарт для обміну інформації, використовує сім бітів для збереження закодованого символу англійської мови.

UTF-8 – це міжнародний стандарт, що надає можливість цифрового представлення символів любої мови світу. Використовує комбінацій блоків з 8 біт для кодування любого символу.

Тому для реалізації інтерпретаторі доцільно використати кодування символів Unicode.

Проведемо аналіз парадигм програмування.

Функціональне розглядає написання програм, як використання та комбінування окремих функцій для отримання результату обчислень. Надає можливість швидшого виконання операцій та зменшує використання пам’яті.

Об’єктно-орієнтоване програмування, що розглядає всю програму, як набір об’єктів, що взаємодіють між собою. Потребує додаткового часу для виконання коду та додаткове виділення пам’яті на збереження об’єктів.

Тому було прийнято рішення створити функціональну мову програмування.

1.4 Постановка задачі для розробки

У відповідності до технічного завдання та аналізу предметної області, розробка програмної системи управління ігровими рушіями на основі скриптової мови, необхідно створити динамічну, функціональну мову програмування сценаріїв, що виконується в реальному часі.

Потрібно провести аналіз існуючих скриптових мов програмування, що використовуються в ігрових рушіях, як метод створення модифікацій, а саме провести аналіз предметної області, проаналізувати мови програмування, що інтерпретується.

Необхідно розробити організацію даних та систему виконання функцій, що включає в себе надання типів даних для ініціалізації та створення методів, що дозволять виклик чистих функцій та функції користувача.

Для достатнього виконання скриптового коду користувача необхідні такі типи даних:

- цілочисельний;
- число з рухомою комою;
- логічний;
- функція;
- область видимості;
- виклик;
- символьна стрічка.

Потрібно розробити наступні чисті функції:

- додавання;
- віднімання;
- множення;
- ділення;
- присвоєння;
- порівняння;
- логічні оператори;
- друк.

Розробити область видимості об’єктів, їх ініціалізації та видалення з підтримкою рекурсивних функцій, що потребує розробити систему для збереження усіх об’єктів, що створюються під час виконання сценарію, що також буде надавати можливість звільнення пам’яті та виконання функцій з рекурсією.

Розробити інтерпретатор скриптового коду з підтримкою пріоритетності операторів, для реалізації необхідно розробити систему, що буде зчитувати сценарій та контролювати виконання коду, також виконувати операції у вірному порядку.

Затвердити роботу рушія мови програмування сценаріїв використовуючи різні методи тестування, а саме димове тестування, тестування сумісності, порівняння виводу та тестування продуктивності.

1.5 Висновки

В розділі описано результати проведеного аналізу предметної області, аналогових скриптових мов програмування, визначено головні завдання. В результаті було підтверджено доцільність розробки програмного продукту.

2 РОЗРОБКА МЕТОДУ УПРАВЛІННЯ ІГРОВИМИ РУШІЯМИ НА ОСНОВІ СКРИПТОВОЇ МОВИ

2.1 Вибір мови програмування

Для розробки системи управління ігровими рушіями необхідно обрати мову програмування на якій буде написано мову програмування сценарію, що дозволить інтеграцію в більшість ігрових рушіїв.

Провівши короткий, аналіз ігрових рушіїв, було встановлено, що в 8 з 10 ігрових рушіїв було використано мову програмування C++, що складає найбільший відсоток в усіх розглянутих ігрових рушіях, таблиця 2.1.

Таблиця 2.1 – Використанні мови програмування в ігрових рушіях

Ігровий рушій	Мови програмування
Unity	C++, C#
Unreal Engine	C++, C#, UnrealScript
Godot	C++
AppGameKit	C++, Delphi, AGK BASIC
CryEngine	C++, Lua
Amazon Lumberyard	C++, Lua
LibGDX	C++, C, Java
Urho3D	C++, AngelScript
GameMaker	C#, Delphi
RPG Maker	JavaScript, Ruby

Використання мови програмування C++ покращить швидкість виконання, надає можливість оперувати пам’яттю та дозволить виконання на усіх сучасних операційних системах, що підтримують компіляцію C++. Тому при написанні рушія мови програмування, необхідно уникнути прямого виклику системних функцій, використовуючи стандартну бібліотеку.

Для компіляції коду доцільно використати генератор сценаріїв складання CMake, що полегшить компіляцію з більшістю компіляторів та підтримується на усіх сучасних операційних системах.

Тому було обрано середовище розробки CLion, який використовує CMake для компіляції коду та надає можливість зручного налагодження програм.

Отже для написання рушія мови програмування сценаріїв буде використано мову програмування C++, генератор сценаріїв складання CMake та середовище розробки CLion.

2.2 Аналіз способів ініціалізації даних

В C++ наявно два способи ініціалізації даних: статична та динамічна типізація.

Статична типізація – автоматичне виділення пам’яті мовою програмування відповідно до коду програми. Контроль над пам’яттю виконується мовою програмування, що усуває проблему з звільненням виділеної пам’яті. Складність передачі даних між областями видимості та поверненням з функцій. Потребує вказати точний розмір пам’яті, що буде виділятись до виконання програми.

Динамічне виділення пам’яті – це запит операційної системи на надання програмі певної кількості пам’яті для збереження даних. Потребує системи контролю над виділеною пам’яттю. Дозволяє зручне використання об’єктів в любій частині програми. Потребує додаткові ресурси машини при виділенні пам’яті під час виконання програми. Можливі проблеми з звільненням пам’яті [7].

Для ініціалізації даних в рушії доцільно використати динамічну типізацію, що полегшить роботу з даними та дозволить контролювати звільнення пам’яті.

В стандартній бібліотеці C++ наявно багато типів контейнерів, що дозволять зберігати об’єкти, розглянемо декілька з них: vector, list, set, map, unordered_map.

vector – це динамічний масив, що дозволяє швидкі операції над кінцем списку, також наявний аналог deque, що надає доступ до обох кінців списку з однаковою швидкістю доступу.

list – це двобічний зв’язний список, його елементи зберігаються довільно в пам’яті, що зменшує швидкість пошуку до певного елементу, але надає високу швидкість додавання та видалення елементу.

set – це математична множина елементів, що дозволяє виконання більшості операцій над множинами. Додання або виділення елементів не впливає на роботу ітератора, що усуває необхідність повторного пошуку в контейнері.

map – це асоціативний масив даних, що дозволяє доступ до даних використовуючи ключ, зазвичай реалізований за допомого красно-чорного бінарного дерева, що дозволяє швидкий доступ до елементів, але потребує додаткового часу на додання елементів.

unordered_map – це асоціативний масив даних, що на відміну від звичайного map не сортується, потребує унікального ключа для кожного асоціативного об’єкту для визначення хеш значення, що дозволяє швидкий доступ до елементів, але потребує додаткового часу на додання нових елементів та використовує більшу кількість пам’яті, для пришвидшення додання елементів.

Отже, найкраще використовувати map або unordered map, що дозволить доступ до елементів по ключу та швидкий пошук елементів, для головного збереження даних. В випадку частого додання нових елементів доцільно використати map. В випадку частого доступу до об’єктів краще використати хеш таблицю.

Для збереження малої кількості елементів або для тих, що потребують ітерації кожного окремого елементу доцільно використати vector.

2.3 Організація даних

Для полегшення роботи з динамічною пам’яттю доцільно використати базовий абстрактний клас, що дозволить збереження усіх елементів в одному контейнері.

Базовий клас має забезпечувати:

– можливість визначення класу об’єкту, щоб уникнути додаткове визначення класу об’єкта та полегшить роботу системі перевірки типів;

- контроль проведення базових операцій над об’єктами, щоб уникнути проведення операцій між класами, що не передбачають дані операції, наприклад операція множення між об’єктами функції;
- надання віртуальної функції запису об’єкту, дозволить кожному класу записати свою функцію присвоєння.

Для забезпечення роботи користувача необхідно надати такі базові типи даних: цілочисельний, дробовий, логічний та стрічковий.

Для цілочисельного типу даних доцільно використати long long, оскільки це збільшує максимальне значення з яким можна оперувати і можливість використати, як вказівник.

Для дробового класу доцільно використати тип даних з подвійною точністю, що збільшить точність при обчислень не цілих чисел.

Для логічного типу даних використаємо bool, для полегшення роботи з логічними виразами.

Для стрічкового типу даних використаємо string з стандартної бібліотеки, що дозволить легкі операції з стрічками.

Для забезпечення роботи рушія мови програмування необхідно реалізувати такі класи: функції, виклику та області видимості.

Клас функції має забезпечувати:

- зберігання типів, над якими проводитиметься операції, для перевірки з типами об’єктів, що будуть використовуватись при виклику функції;
- зберігання назв об’єктів, що було вказано при ініціалізації функції, для надання доступу до об’єктів поданих при виконанні;
- надання інструкції виконання для функцій користувача або надання базової функції.

Клас виклик відповідає за:

- перевірку типів поданих для виконання функції;
- виклик функції;
- передача функції даних для виконання операцій;
- повернення результату;

– контроль пам’яті функції.

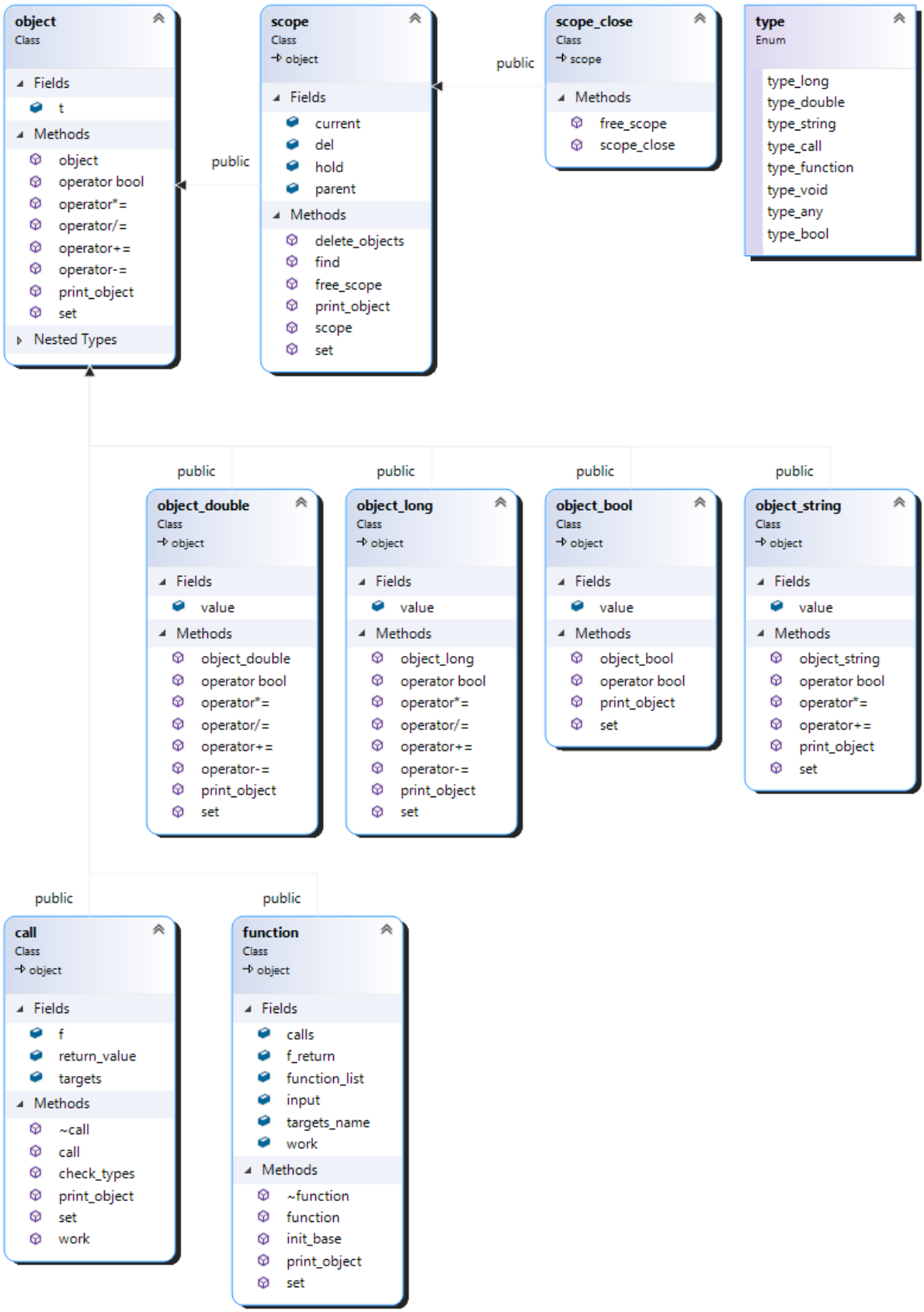

Рисунок 2.1 – Діаграма класів

Клас області видимості надає можливість:

– доступу до елементу по імені
– збереження елементів
– звільнення пам’яті
– відкриття нової області видимості

2.4 Розробка системи викликів

Для полегшення контролю над виконанням функцій, було створено клас викликів, що відповідає за контроль та виконання функції (див. рис. 2.2).

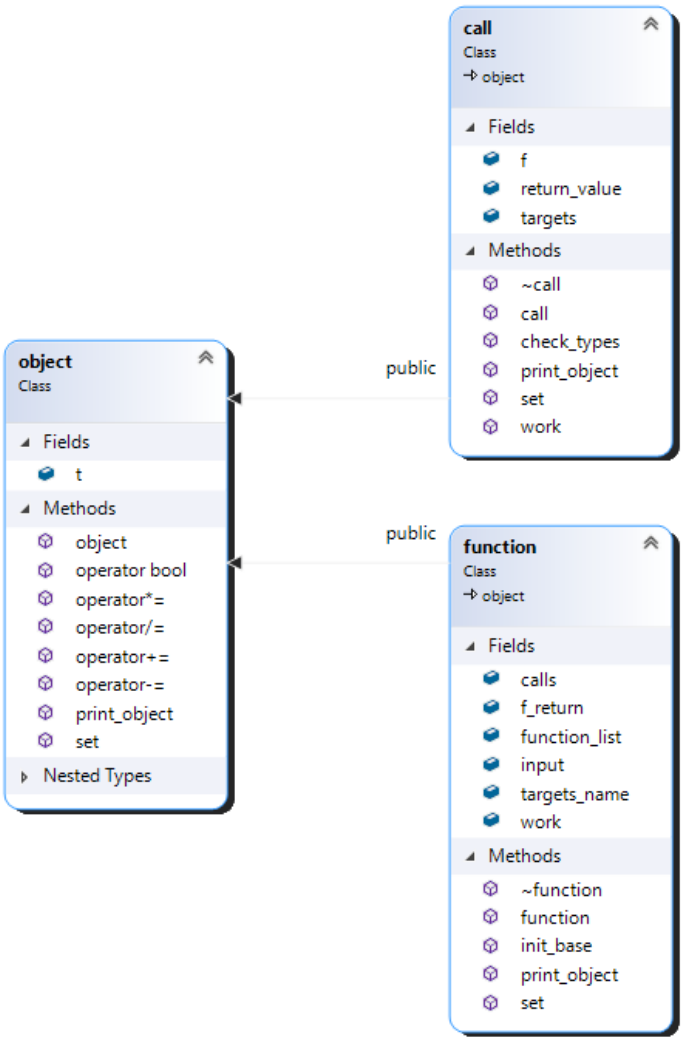

Рисунок 2.2 – Клас виклику та функції

Під час виклику функції (див. рис. 2.3), наявно декілька ключових стадій.

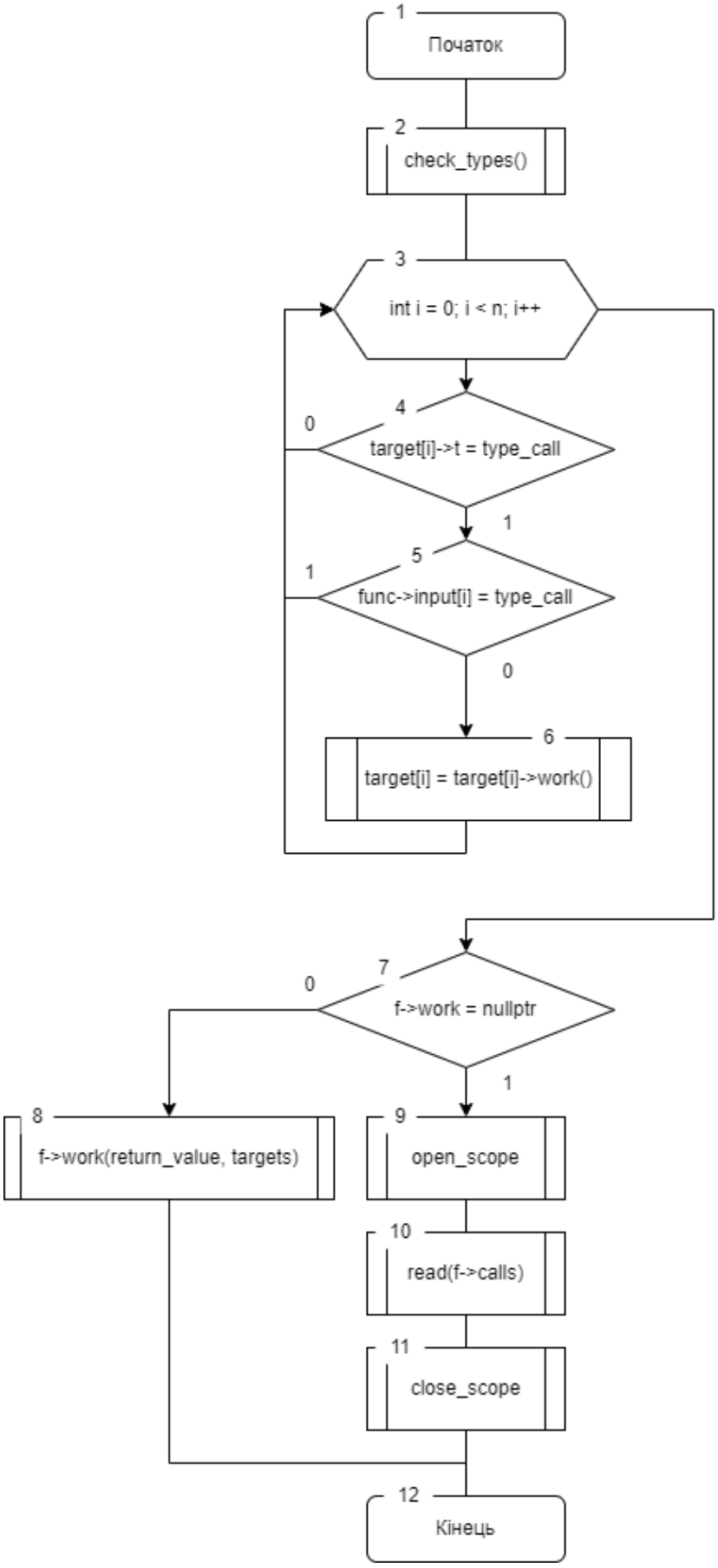

Рисунок 2.3 – Блок-схема алгоритму виклику функцій

Спершу, перевіряється відповідність типів, що надходять та типами, що було вказані при ініціалізації функції (див. рис. 2.4).

В наступному етапі, проходиться кожен елемент, що було надано та перевіряється чи він вже був обчислений, якщо ні то він обчислюється викликом цієї ж функції.

Далі йде перевірка чи це базова функція, перевіряється за допомогою перевірки елемента функції, що зберігає базову функцію.

Якщо функція базова вона виконується. Інакше відкривається нова область видимості, інтерпретатору надається код функції.

В завершені повертається обчислене значення.

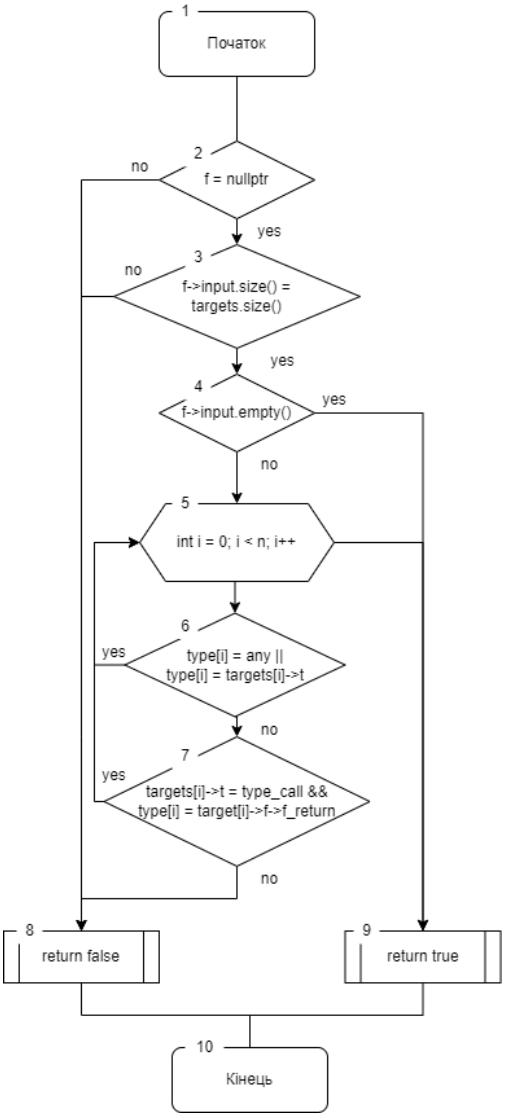

Рисунок 2.4 – Блок-схема алгоритму перевірка типів

Перевірка типів включає в себе перевірку на посилання. Порівняння кількості поданих елементів і потрібних. Потім обходиться кожен елемент і порівнюється з типом, що запитується функцією.

Якщо типи збігаються або тип на вході не має значення перевіряється наступний елемент. Інакше перевіряється чи це значення, що має бути обчислене і тип результату, якщо так то перевіряється наступний елемент інакше повертається з помилкою.

2.5 Розробка базових функцій

Для виконання коду необхідно надати базові функції, що будуть виконувати самі обчислення. Також потрібно створити можливість легкого додання нових базових функцій.

Необхідні базові функції:

- додавання;
- віднімання;
- множення;
- ділення;
- присвоєння;
- порівняння;
- логічні оператори;
- друк.

Такий набір дозволить виконувати базові операції та отримувати результат обчислень.

Для надання можливості додання нових функцій необхідно створити систему, що буде мінімізувати роботу з рушієм мови програмування, бажано надавати тільки необхідні дані для роботи функції, такі як: посилання на результат обчислення, дані над якими мають проходити обчислення та доступ до додання функції до списку базових (див. рис. 2.5).

```
void plus(object*& return_value, std::vector<object*> targets) {
    return_value = *targets[0] + *targets[1];
}
void minus(object*& return_value, std::vector<object*> targets) {
    return_value = *targets[0] - *targets[1];
}
void multiply(object*& return_value, std::vector<object*> targets) {
    return_value = *targets[0] * *targets[1];
}
void divide(object*& return_value, std::vector<object*> targets) {
    return_value = *targets[0] / *targets[1];
}
void print(object*& return_value, std::vector<object*> targets) {
    targets[0]->print_object();
}
void set_value(object*& return_value, std::vector<object*> targets) {
    targets[0]->set(targets[1]);
    return_value = targets[0];
}
```

Рисунок 2.5 – Базові функції

Створимо список усіх базових функції, що будуть використовуватись для їх пошуку і надання можливості додання нових (див. рис. 2.6).

```
function_list.emplace(std::pair<std::string, function*>{ "plus", new function({ type_any, type_any }, {}, plus) });
function_list.emplace(std::pair<std::string, function*>{ "minus", new function({ type_any, type_any }, {}, minus) });
function_list.emplace(std::pair<std::string, function*>{ "multiply", new function({ type_any, type_any }, {}, multiply) });
function_list.emplace(std::pair<std::string, function*>{ "divide", new function({ type_any, type_any }, {}, divide) });
function_list.emplace(std::pair<std::string, function*>{ "print", new function({ type_any }, {}, print) });
function_list.emplace(std::pair<std::string, function*>{ "set", new function({ type_any, type_any }, {}, set_value) });
function_list.emplace(std::pair<std::string, function*>{ "or", new function({ type_any, type_any }, {}, f_or) });
function_list.emplace(std::pair<std::string, function*>{ "reverse", new function({ type_any }, {}, f_reverse) });
function_list.emplace(std::pair<std::string, function*>{ "and", new function({ type_any, type_any }, {}, f_and) });
function_list.emplace(std::pair<std::string, function*>{ "less", new function({ type_any, type_any }, {}, less) });
function_list.emplace(std::pair<std::string, function*>{ "more", new function({ type_any, type_any }, {}, more) });
function_list.emplace(std::pair<std::string, function*>{ "less_eq", new function({ type_any, type_any }, {}, less_eq) });
function_list.emplace(std::pair<std::string, function*>{ "more_eq", new function({ type_any, type_any }, {}, more_eq) });
function_list.emplace(std::pair<std::string, function*>{ "equal", new function({ type_any, type_any }, {}, equal) });
function_list.emplace(std::pair<std::string, function*>{ "not_equal", new function({ type_any, type_any }, {}, not_equal) });
```

Рисунок 2.6 – Додання базових функції до списку

Такий спосіб додання функцій полегшує роботу з створенням викликів, оскільки при доданні до списку, надається усі необхідні дані.

2.6 Висновки

Проведено аналіз:

1) мов програмування, що використовується при написанні ігрових рушіїв та обрано найбільш популярну C++, з використанням генератора сценаріїв складання CMake та середовища розробки CLion;
2) способів ініціалізації даних та їх утримування в C++ та обрано зберігання використовуючи динамічну типізацію з використанням асоціативних масивів.

Розроблено:

1) структуру наслідування даних, що надає можливість виконання усіх базових операцій;
2) систему, що відповідає за перевірку типів та виклик функцій;
3) головні базові функцій та структуру, що надає можливість додання нових.

3 РОЗРОБКА ЗАСОБУ ДЛЯ ІНТЕРПРЕТАТАЦІЇ КОДУ

3.1 Опис синтаксису для інтерпретатора

Для роботи рушія мови програмування нам необхідно контролюючий блок, що буде зчитувати файл, аналізувати символи, що поступають і на основі них викликати ядро.

Нам необхідно, щоб інтепретатор надавав можливість розуміти декілька ситуацій.

Ініціалізація даних, потребує можливість вибору типу змінної, назви, що буде використовуватись для подальшого доступу до об’єкта та за потреби базове значення, що може бути обраховане (див. рис. 3.1).

```
long value=3
double pi = 3.0 + 0.14
string blank = ""
bool a
```

Рисунок 3.1 – Синтаксис ініціалізації

Для вибору типу будуть використовуватись чотири ключових слова: “long”, “double”, “string”, “bool”, кожен з яких буде ініціалізувати відповідний клас. Наступне слово буде використовуватись для доступу до об’єкта. За наявності знаку рівно, значення буде обраховане та виставлене.

Контроль області видимості, включає в себе відкриття нової та закриття старої.

Область видимості дозволяє керувати видаленням об’єктів, якщо вона закривається, уся пам’ять виділена об’єктам, що було створено в ній, буде звільнена.

Ініціалізація функцій, потребує вказати її назви, типи та назви об’єктів, що будуть надходити та код, що виконується при виклику (див. рис. 3.2).

```
function void(){
}

function fact(long a){
    if( a < 2 ){
        return 1;
    }
    return a * fact(a - 1);
}
```

Рисунок 3.2 – Синтаксис ініціалізації функції

Для визначення ситуації використаєм нове ключове слово “function”, після імені функції в круглих дужках надходять вхідні дані, в кінці в фігурних дужках, будуть надходити операції, що мають бути виконанні.

Обчислення, включає в себе виклик функцій, роботу з умовними переходами, повернення результату та виконання операцій (див. рис. 3.3).

```
{
    print(fact(3) + fact(4));
    int a;
    a = fact(3);
    a += fact(4);

    if(true){
        print( "work" );
    } else {
        print( "don't work" );
    }
    return 0;
}
```

Рисунок 3.3 – Синтаксис обчислень

Для показу завершення обчислення використовується крапка з комою. Інтерпретатор потребує додання нових ключових слів “if” та “return”, та умовно “else”, оскільки його пошук відбуватиметься тільки після “if”.

3.2 Розробка системи області видимості та ініціалізації об'єктів

Область видимості буде використовуватись для збереження усіх об’єктів, що будуть створюватись, включаючи об’єкти ініціалізовані користувачем, функції та тимчасові зміні.

Для доступу до об’єктів буде використовуватись посилання на саму нову область видимості. При відкриті нової посилання переміститься на неї, і в новій буде посилання на попередню, для отримання доступу до об’єктів збереженій в старій та надання можливості видалення (див. рис. 3.4).

При запускі автоматично буде створена нова область видимості, що буде використовуватись, як глобальна.

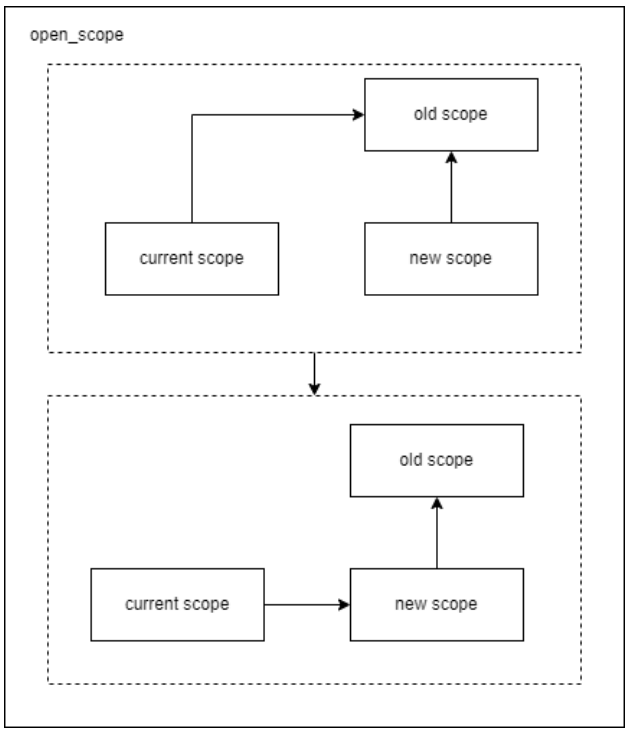

Рисунок 3.4 – Відкриття нової області видимості

Операція видалення буде виконувати усі дії в протилежному порядку, рисунок.

При видаленні області видимості, усі об’єкти збережені в ній звільняються і посилання переходить на попередню область.

Інтерпретатор контролює роботу з областю видимості та виконанням операцій (див. рис. 3.5)

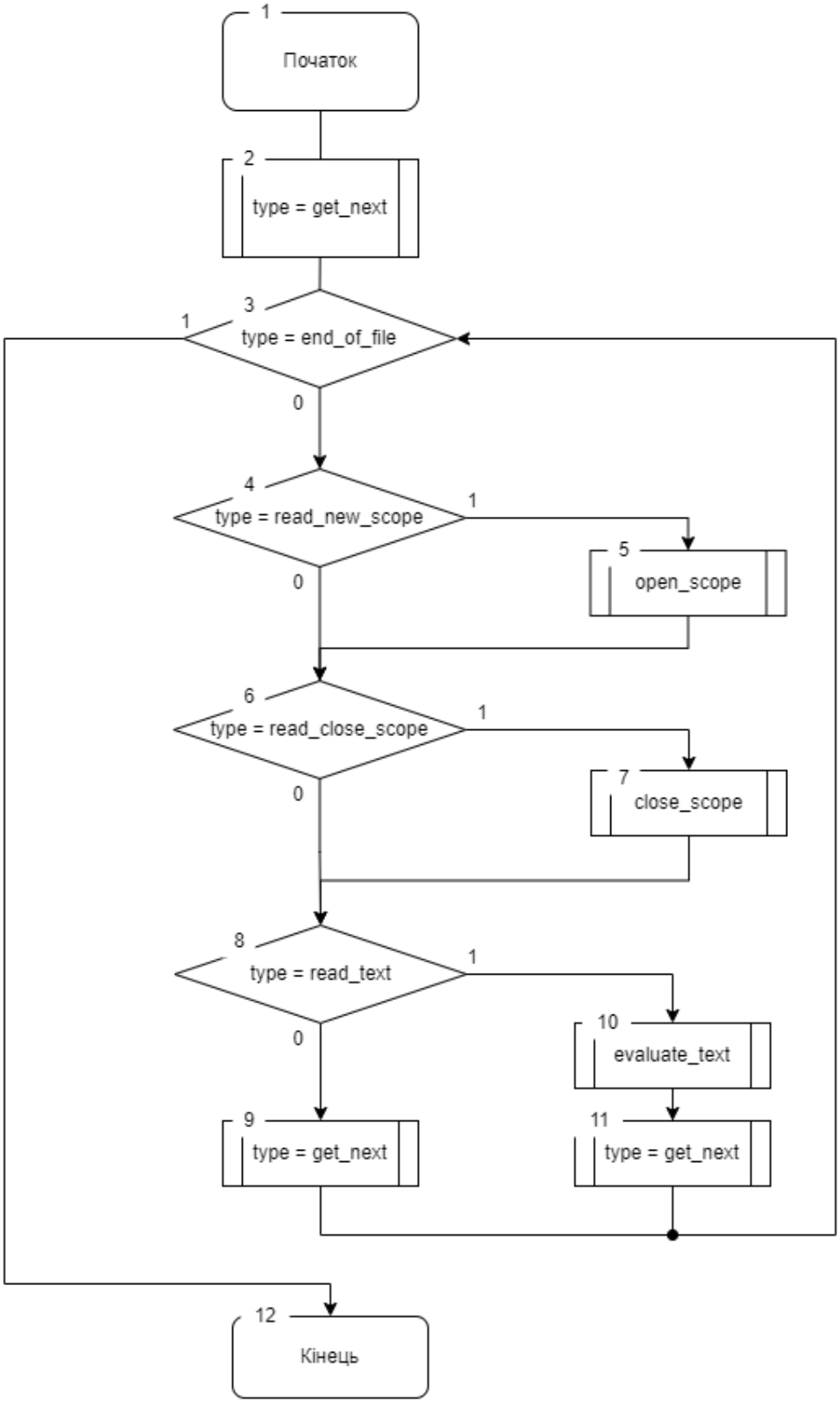

Рисунок 3.5 – Блок-схема вхідного блоку інтерпретатора

Така структура дозволяє розділити текст по блокам, зазвичай по стрічкам, у випадку ініціалізації функції та умовного переходу по фігурним дужкам.

В залежності від зчитаного символу інтерпретатор визначає дію, що має бути виконана (див. рис. 3.6).

Якщо зчитано символи пропуску або нової стрічки зчитується наступний символ. У випадку зчитування символу функція повертається з інструкцією, що необхідно визначити ключове слово. За надходження фігурних дужок запитується контроль над областю видимості.

Якщо символ не збігається з жодним вказаним, виводиться помилка, оскільки інтерпретатор не має ніяких інших методів опрацювання коду, що можуть його виконати, в виводі вказується символ, його цифрове значення та порядковий номер у тексті.

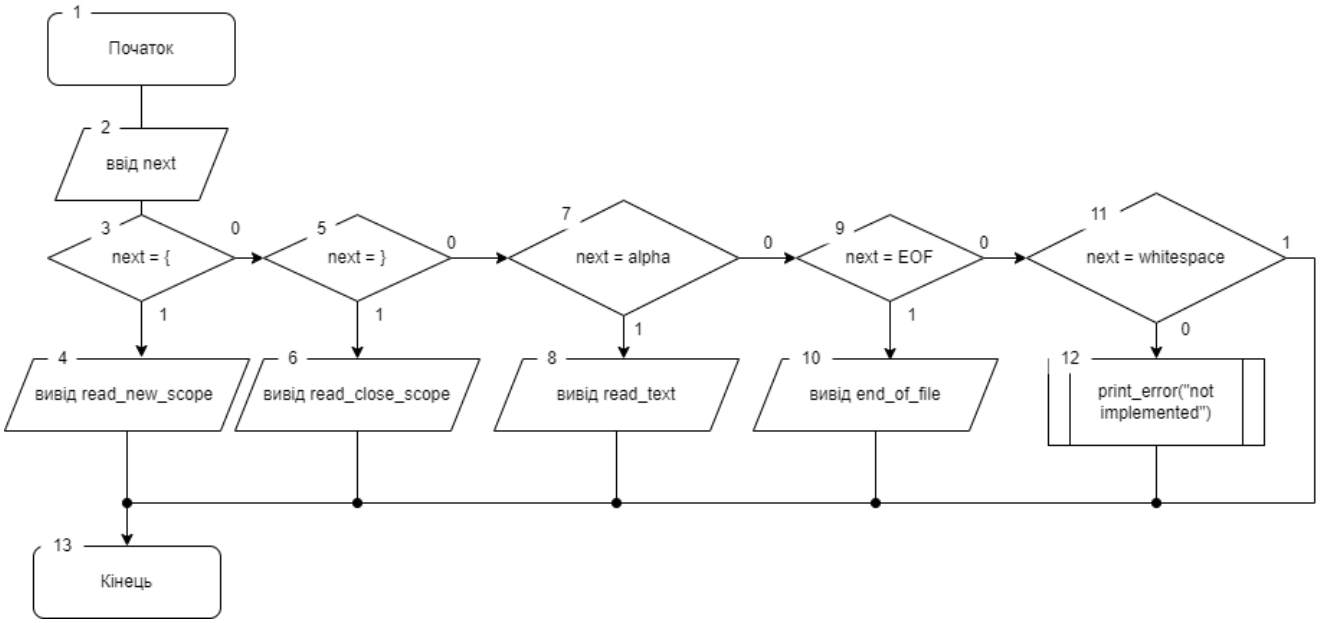

Рисунок 3.6 – Блок-схема алгоритму визначення типу

При перевірці ключового слова визначається одна з наступних дій: ініціалізація даних, ініціалізація функції, повернення даних, умовний перехід, у

випадку відсутності ключового слова інтерпретатор очікує на вираз, що потребує обчислення (див. рис. 3.7).

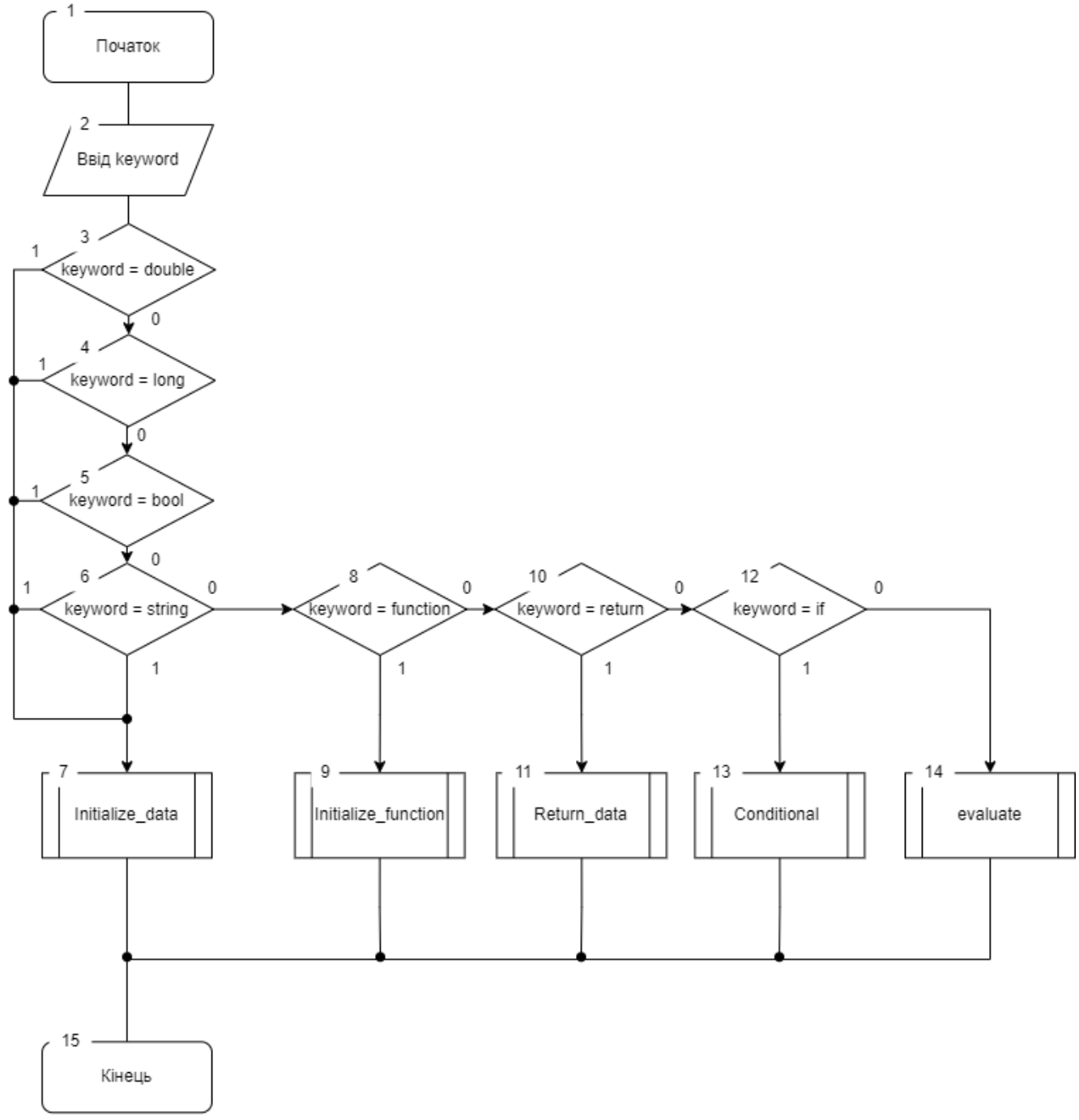

Рисунок 3.7 – Блок-схема обробки ключового слова

Після завершення операції, інтерпретатор переходить до наступного проходження циклу.

Для кожної ініціалізації змінної в залежності від типу наявна своя функція, що працює по однаковій послідовності (див. рис. 3.8).

Рисунок 3.8 – Блок-схема ініціалізації змінної

Зчитується назва змінної, далі зчитується символ, у випадку знаку рівності необхідно обчислити значення, якщо подається символ кінця стрічки використовується базове значення для об’єкта, потім об’єкт додається до області видимості з ім’ям, як ключ.

Якщо було зчитано інший символ, інтерпретатор повертає помилку.

Базові значення для типів даних:

- Цілочисельний 0
- Дробовий 0
- Стрічковий «»
- Логічний false

Така імплементація інтерпретатора дозволяє швидке додання нових ключових слів і методів їх обробки.

3.3 Розробка пріоритету операторів

Для дотримання до вірного порядку виконання операцій, необхідно виставити кожному оператору відповідний пріоритет виконання.

Під час розробки системи сортування операцій було взято пріоритети виконання операторів C++ і виставленно необхідним операторам значення (див. рис. 3.9) [8].

Для розробки рушія мови програмування необхідно оператори для таких базових операцій:

- логічне ні;
- множення та ділення;
- додавання та віднімання;
- більше, менше;
- рівно та не рівно;
- логічне І;
- логічне АБО;
- операція присвоєння;

Precedence	Operator	Description	Associativity
1	::	Scope resolution	Left-to-right
2	++ -- () [] . ->	Suffix/postfix increment and decrement Function call Array subscripting Element selection by reference Element selection through pointer	
3	++ -- + - ! ~ (*type*) * & sizeof new, new[] delete, delete[]	Prefix increment and decrement Unary plus and minus Logical NOT and bitwise NOT Type cast Indirection (dereference) Address-of Size-of Dynamic memory allocation Dynamic memory deallocation	Right-to-left
4	.* ->*	Pointer to member	Left-to-right
5	* / %	Multiplication, division, and remainder	
6	+ -	Addition and subtraction	
7	<< >>	Bitwise left shift and right shift	
8	< <= > >=	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
9	== !=	For relational = and ≠ respectively	
10	&	Bitwise AND	
11	^	Bitwise XOR (exclusive or)	
12	|	Bitwise OR (inclusive or)	
13	&&	Logical AND	
14	||	Logical OR	
15	?: = += -= *= /= %= <<= >>= &= ^= |=	Ternary conditional Direct assignment (provided by default for C++ classes) Assignment by sum and difference Assignment by product, quotient, and remainder Assignment by bitwise left shift and right shift Assignment by bitwise AND, XOR, and OR	Right-to-left
16	throw	Throw operator (for exceptions)	
17	,	Comma	Left-to-right

Рисунок 3.9 – Пріоритети операторів C++

Для можливості ідентифікації кожного оператора навіть з однаковою вагою помножим пріоритет на десять і додаймо унікальне число, що дозволить нам виконувати сортування без додаткових перестановок, відкинувши останній десяток та використовувати число для вказання операції, що має бути використана (див. рис. 3.10).

```
enum level{
    value = 00,
    level_function = 20,
    reverse = 30,
    multiply = 50,
    divide = 51,
    plus= 60,
    minus= 61,
    less = 90,
    less_eq = 91,
    more = 92,
    more_eq = 93,
    equal = 100,
    not_equal = 101,
    log_and = 140,
    log_or = 150,
    set = 200
};
```

Рисунок 3.10 – Ідентифікатори операторів

Далі нам необхідна функція, що буде перетворювати стрічку в набір об’єктів, що можуть бути прив’язані один до одного і відсортовані відповідно до пріоритету (див. рис. 3.11).

Спершу стрічка ділиться, по елементах, значеннях та операторах, та надходить до об’єктів, що використовуються для їх тримання.

Потім кожен об’єкт в масиві проходиться, в залежності від значення визначається операція. У випадку коли значення збігається з одним з операторів то в тип записується відповідний ідентифікатор. Якщо значення не збігається з жодним з вказаних операторів інтерпретатор перевіряє чи це виклик функції або виразу в дужках, якщо так то значення обраховується, інакше об’єкт приймається, як дані.

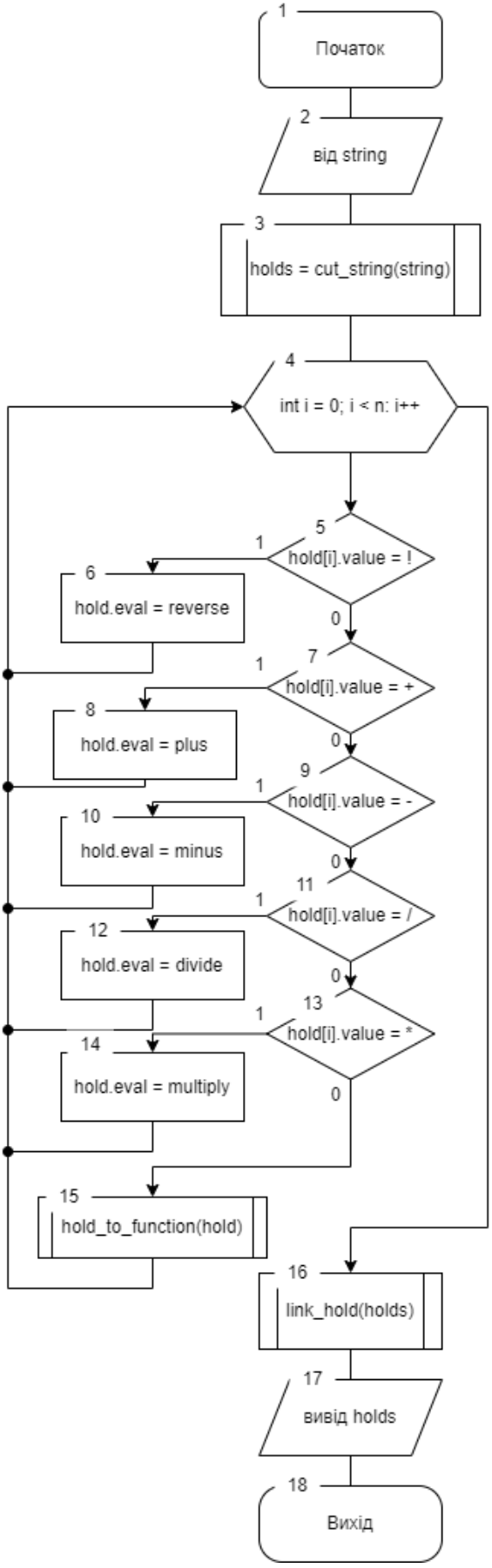

Рисунок 3.11 – Блок-схема перетворення стрічки на об’єкти

В завершенні необхідно пов’язати об’єкти один з одним, для подальшого сортування (див. рис. 3.12).

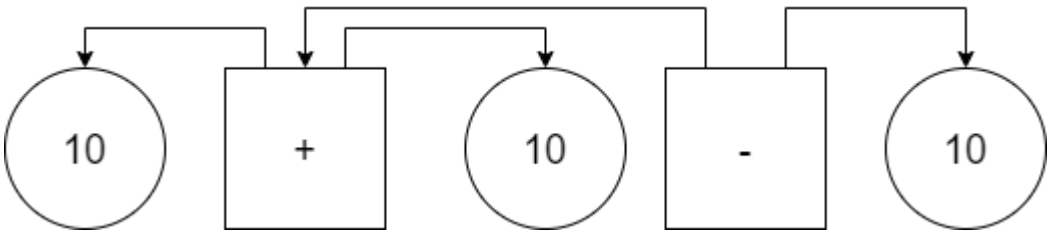

Рисунок 3.12 – Зв’язування об’єктів

В більшості випадків оператори виконують дію над двома або одним об’єктом, тому для сортування порядку виконання доцільно використати бінарне дерево.

Для зв’язування об’єктів береться перший оператор в масиві об’єктів та виставляється ліве та праве значення, як значення над якими проходитимуть операції. У випадку оператора, що використовує одне значення використовується ліве чи праве в залежності від конкретного оператора.

Береться наступний оператор, надається йому посилання на перші не використанні об’єкти зліва та справа.

Операції продовжується до проходження усіх операторів.

Далі необхідно провести сортування та обчислення значення операцій (див. рис. 3.13).

Для сортування потрібно визначити вершину дерева операторів, шукаючи об’єкт, що не посилається іншими об’єктами.

Сортування відбувається використовуючи стандартний алгоритм для впорядкування бінарного дерева, об’єкт з найбільшим пріоритетом на вершині.

Далів відбувається перетворення дерева на набір викликів функцій або на значення за відсутності операцій, що необхідно виконати.

В кінці виконується виклик функції і за потреби конвертується результат до необхідного типу, за можливості, наприклад ціле число до дробового.

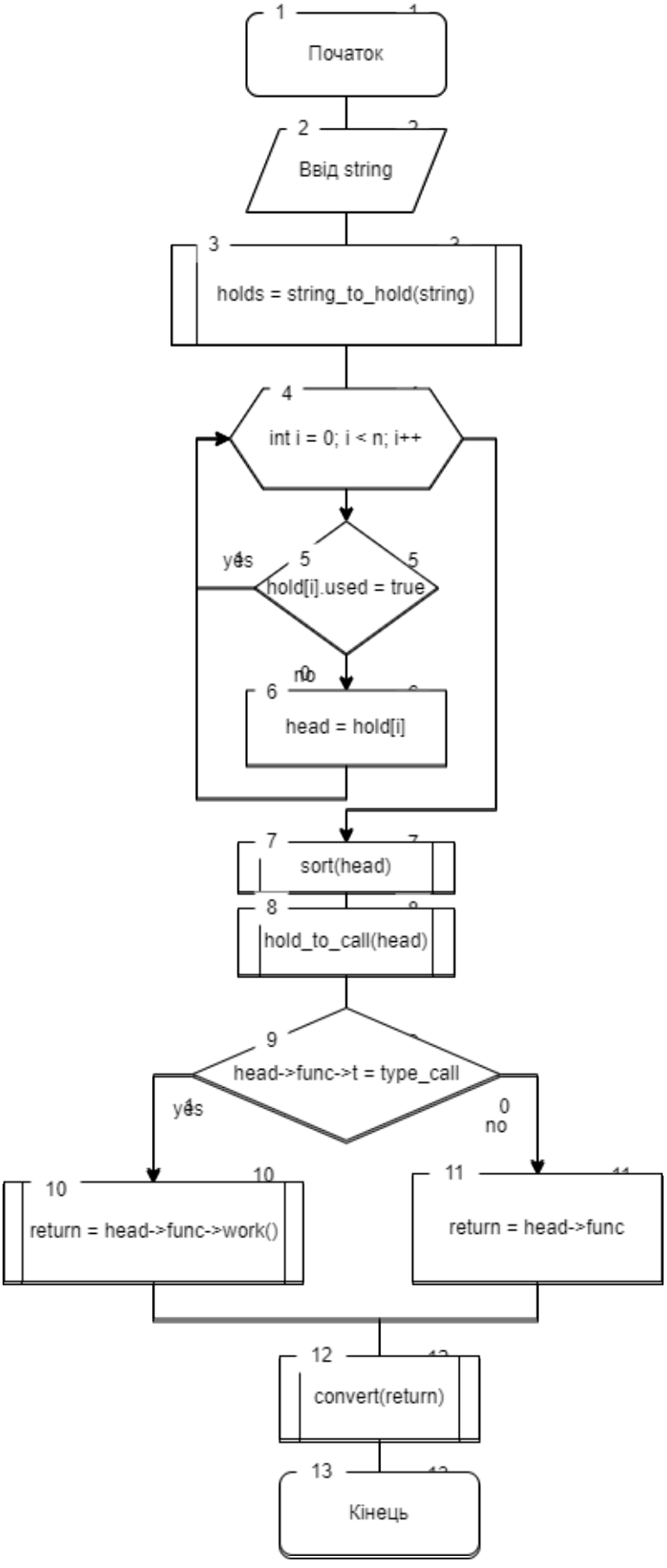

Рисунок 3.13 – Блок-схема обчислення виразу

Використовуючи таку структуру пріоритетів операторів і виконання дозволяє швидке додання нових операторів.

3.4 Розробка ініціалізації функцій користувача

Ініціалізація функції користувача потребує вказування назви функції, типів об’єктів та їх імені та набору операцій.

Інтерпретатор розпізнає п’ять типів, чотири стандартні та «будь-який», він передбачає можливість прийняття любого типу об’єкта (див. рис. 3.14).

```
function void(any test){
    print(test);
}
```

Рисунок 3.14 – Ініціалізація функції

Інтерпретатор дозволяє виконувати функції з наявністю рекурсії, оскільки обчислений результат копіюється в тимчасову зміну (див. рис. 3.15).

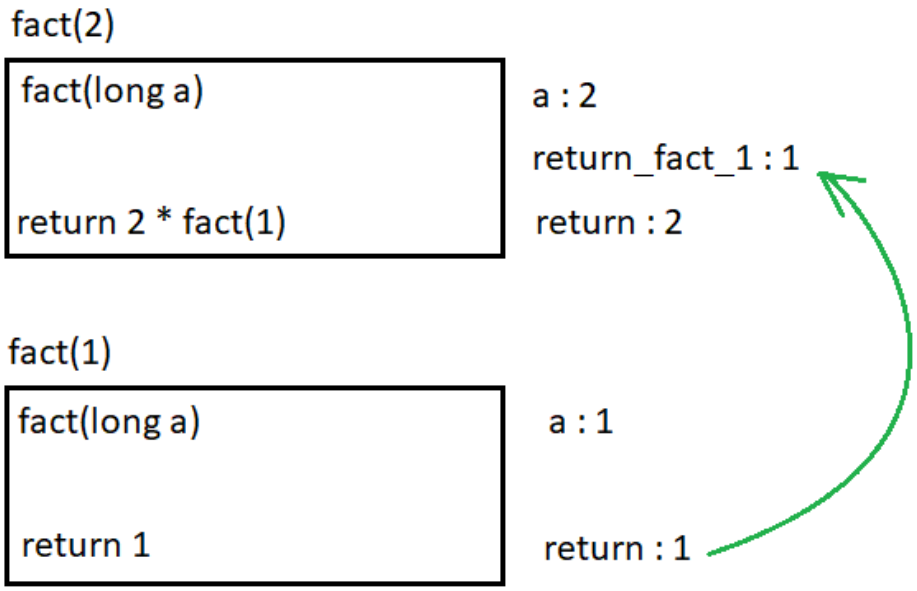

Рисунок 3.15 – Повернення з функції

Процес ініціалізації функції (див. рис. 3.16).

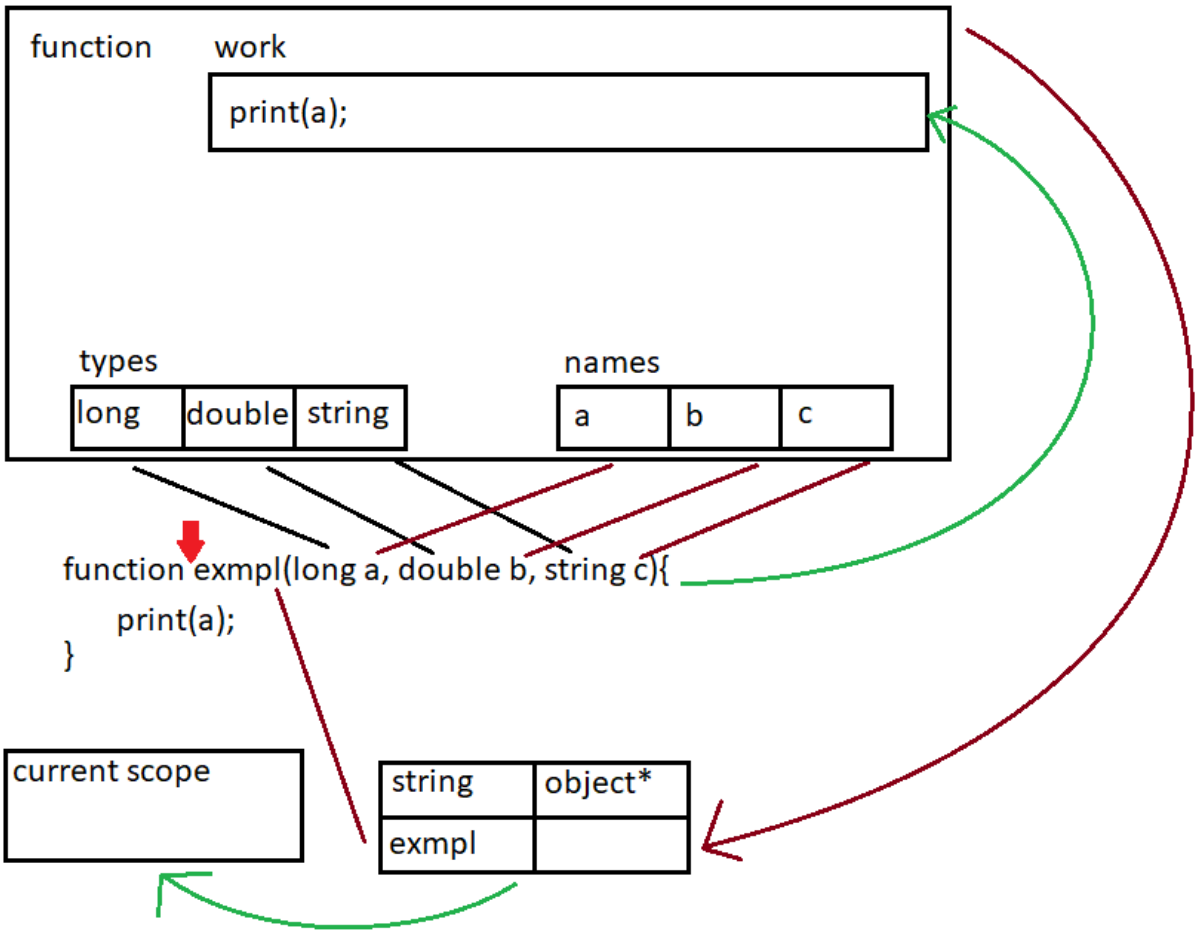

Рисунок 3.16 – Процес ініціалізація функції

Ця послідовність включає в себе:

- зчитування імені, що використається для доступу функції через область видимості;
- зчитування типу та імені змінної в відповідний масив елементів;
- копіювання частини стрічки, що зберігається в фігурних дужках для виконання при виклику;
- в кінці додається посилання на об’єкт функції та її ім’я до області видимості.

3.5 Висновки

Проведено розробку:

1) синтаксис для інтерпретатора та ситуації, що він може опрацювати в залежності від коду, що йому поступає;
2) системи відповідно якої працює область видимості та спосіб її керування за допомогою інтерпретатора;
3) зчитування ключових слів та виконання відповідних операцій. Розглянуто ініціалізація даних та функцій;
4) системи пріоритетів операторів та перетворення стрічки в вірну послідовність операцій для виконання.
5) Описано поведінку при ініціалізації та виконанні функції.

4 ТЕСТУВАННЯ РОБОТИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Засоби і методи для тестування

При розробці будь-якої програми відбувається тестування, що мають перевірити вірність роботи, відсутність помилок, час роботи, відповідність вимогам та інше.

Існує багато типів тестування програмного забезпечення, що можуть бути доцільними для тестування роботи рушія мови програмування.

Інсталяційне тестування перевіряє роботу установки програмного забезпечення.

Перевірка сумісності потребує тестування додатку з використанням різних операційних систем та їхніх версій, та інших програм, що використовуються при виконанні.

Деструктивне тестування підтверджує, що програма працює вірно навіть при отримані хибних вхідних даних, націлена на перевірку роботи системи обробки помилок.

Тестування продуктивності включає декілька підтипів, що націлені перевірити швидко дію, стабільність та можливість витримати навантаження.

Димове тестування – це мінімальний набір тестів для перевірки роботи на явні помилки, зазвичай виконується програмістом перед іншими типами тестування.

Альфа та бета тестування – це перевірка за допомогою симуляції або реальних користувачів.

Регресивне тестування відбувається після великої зміни коди або важливої її частини, націлене на зупинку роботи модулів та повернення помилок.

Тестування зручності націлене на перевірку інтерфейсу користувача, його зручності використання.

Тестування відповідності стандартам перевіряє чи відповідає програма певним міжнародним або державним стандартам.

Тестування захищеності, що націлене на перевірку супротиву системи до злому та втручання.

Інтернаціоналізація та локалізація, перевіряє прийнятність програмного забезпечення користувачами інших країн та вірності роботи на різних мовах.

А/Б тестування – метод, що дозволяє вибрати кращий варіант на основі порівняння ефективності нової та старої версії.

Тестування багато поточності перевіряє програму, що використовує більше ніж один процес або потік виконання.

Тестування порівняння виводу порівнює текст або зображення з очікуваним результатом та отриманим.

Тестування властивостей перевіряє не відповідність результату до очікуваного, а лише її частини, наприклад розмір масиву після сортування [9].

Для проведення тестування прийнято рішення виконати такі види тестування:

- димове тестування;
- перевірка сумісності;
- тестування продуктивності;
- тестування порівняння виводу.

4.2 Методика тестування

Для провелення тестування роботи рушія мови програмування сценаріїв, необхідно провести перевірку усіх його головних систем:

- ініціалізація даних;
- ініціалізація функцій;
- виклик функцій;
- порядку виконання операцій;
- умовних переходів;
- області видимості.

Для проведення димового тестування доцільне використання коду сценарію (див. рис. 4.1).

```
{
    double dus = (3.4 + 4.4) / 1;
    function void(){
    }
    function fact(long a){
        if( a < 2 ){
            return 1;
        }
        return a * fact(a - 1);
    }
    function a_lot_types(long a, string b, double d, any any_value){
    }

    if(1 + 1 == 2 ) {
        print( "work" );
    } else {
        print( "don't work" );
    }
    if( false ) {
        print( "don't work" );
    } else {
        print( "work" );
    }
    if(true){
        print( "work" );
    }
    if(false){
        print( "don't work" );
    }
    long dest;
    string help = "help";
    print(dus);
    double ger= 3.4 + 4.4 / multiply(2, 3);
    long gtd=3;
    void();
    print(ger);
    print(plus(ger, gtd));
    print(gtd + ger);
    print(help);

    {
        print(fact(3) + fact(4));
    }

    print(fact(3));
    print(fact(4));
    print(fact(15));
    print(fact(3));
}
```

Рисунок 4.1 – Сценарій для димового тестування

Для виконання сценарію слід використати операційну систему Windows, оболонку з інтерфейсом командного рядка PowerShell, компілятор MinGW-x86_64, щоб підтвердити правильність виконання результат обчислень має збігатись з результатом зображений (див. рис. 4.2).

```
PS E:\cpp\dyplom\clion_language_engine\cmake-build-debug> .\clion_language_engine.exe ..\script.txt
work
work
work
7.8
4.13333
7.13333
7
help
30
6
24
1307674368000
6
PS E:\cpp\dyplom\clion_language_engine\cmake-build-debug>
```

Рисунок 4.2 – Результат виконання димового тестування

Для проведення тестування сумісності з іншими операційними системами слід використовувати сценарій димового тестування. Та використати іншу операційну систему, наприклад Ubuntu 20.04 та скомпілювати використовуючи відповідний компілятор (див. рис. 4.3).

```
moon@Dragon:/mnt/e/cpp/dyplom/clion_language_engine/build$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.4 LTS
Release:        20.04
Codename:       focal
moon@Dragon:/mnt/e/cpp/dyplom/clion_language_engine/build$ ./clion_language_engine ../script.txt
work
work
work
7.8
4.13333
7.13333
7
help
30
6
24
1307674368000
6
```

Рисунок 4.3 – Результат виконання тестування на Ubuntu

При збігу результату з результатом виконання димового тестування можна затвердити роботу на відповідній версії операційній системі.

Також можливо провести перевірку роботи в Debian версії 11.3, не виконуючи компіляцію (див. рис. 4.4).

```
moon@Dragon:/mnt/e/cpp/dyplom/clion_language_engine/build$ ./clion_language_engine ../script.txt
work
work
work
7.8
4.13333
7.13333
7
help
30
6
24
1307674368000
6
moon@Dragon:/mnt/e/cpp/dyplom/clion_language_engine/build$ cat /etc/issue
Debian GNU/Linux 11 \n \l

moon@Dragon:/mnt/e/cpp/dyplom/clion_language_engine/build$ cat /etc/debian_version
11.3
moon@Dragon:/mnt/e/cpp/dyplom/clion_language_engine/build$
```

Рисунок 4.4 – Результат виконання тестування на Debian

Для проведення тестування порівняння виводу слід використати любий засіб для обчислення, наприклад стандартний калькулятор Windows.

Необхідно вивести результат обчислень та порівняти його з результатом. Для перевірки правильності виконання обчислень використаєм довільне обчислення (див. рис. 4.5).

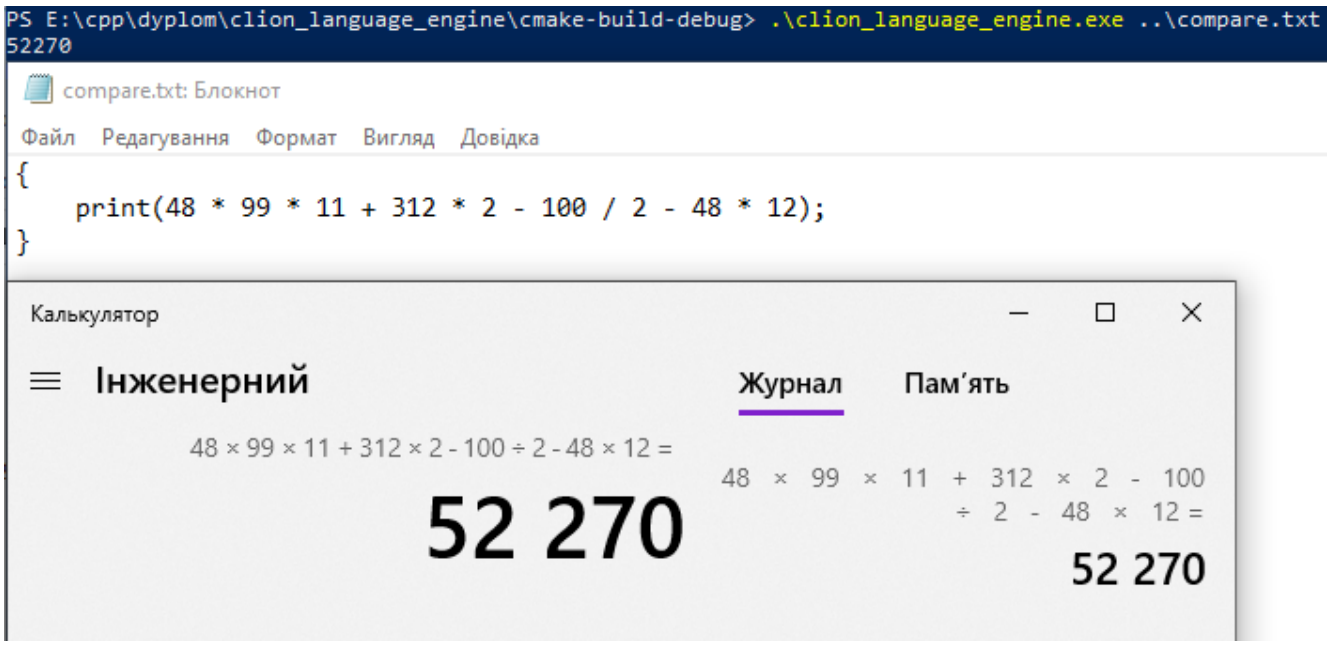

Рисунок 4.5 – Порівняння виконання порядку виконання операцій

При отримані однакового результату при обчислені можливо затвердити роботу системи сортування операторів.

Для тестування роботи виклику рекурсивних функцій слід використати довільну функцію, що включає виклик до самої себе, наприклад обчислення факторіалу (див. рис. 4.6).

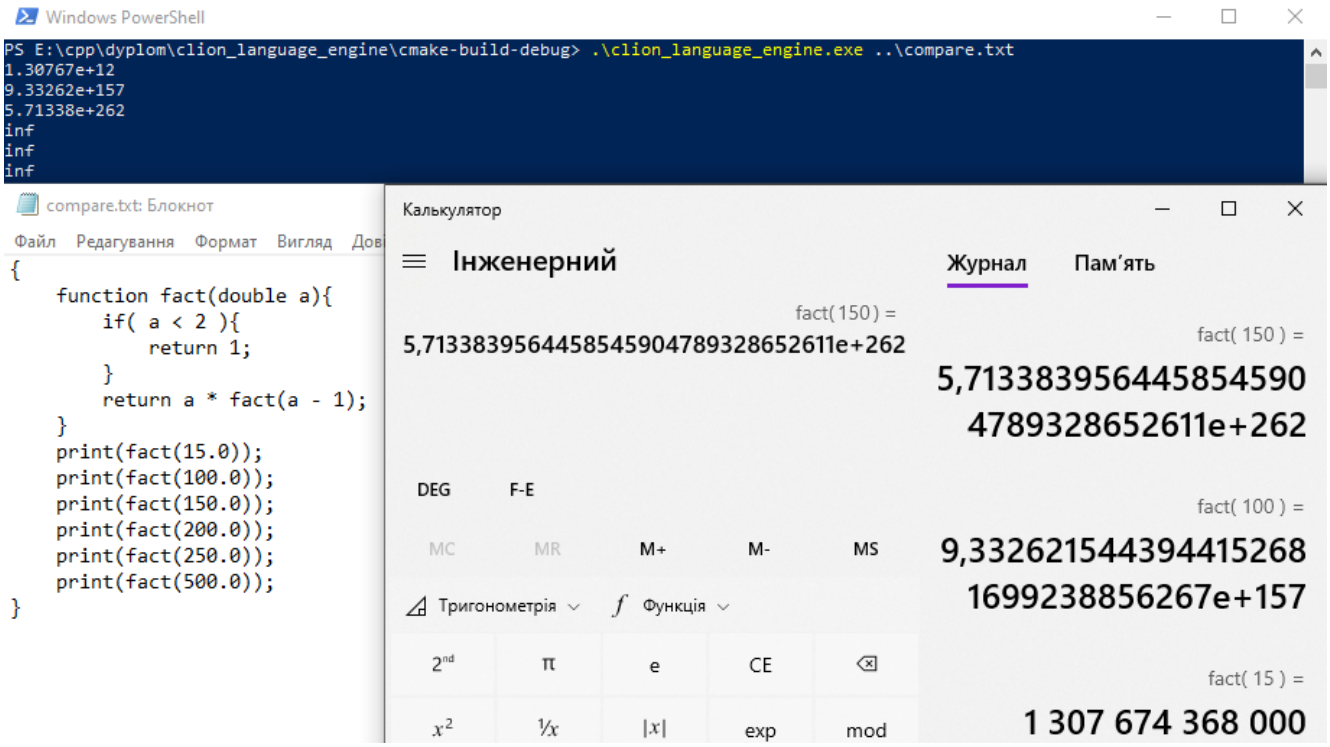

Рисунок 4.6 – Порівняння виконання рекурсивних функцій

Якщо виконання сценарію надало однаковий результат з засобом для обчислення, то рушій підтверджує вірність виконання рекурсивних функцій.

Потрібно провести тестування роботи чисел з плаваючою комою, слід виконати любі операції, що використовують не цілі числа (див. рис. 4.7).

За збігу результатів було підтвержено правильність виконання обчислень з використанням не цілих чисел.

Після проведення усіх тестувань порівняння можна стверджувати, що рушій мови програмування повертає очікуванні результати і працює відповідно виставленим вимогам.

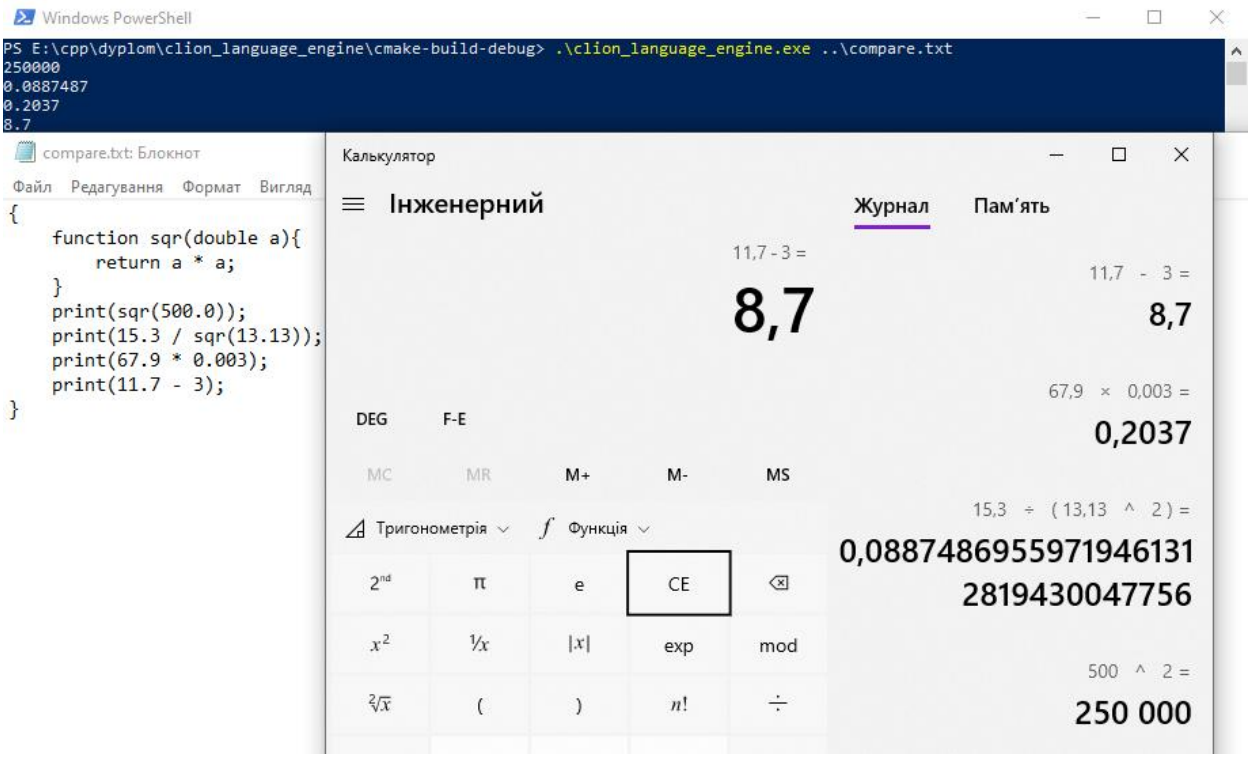

Рисунок 4.7 – Тестування дробових чисел

В завершенні доцільно провести тестування продуктивності, для перевірки роботи під навантаженням слід використати сценарій з 5040 стрічок обчислення факторіала 15 і перевірити на наявність помилок (див. рис. 4.7).

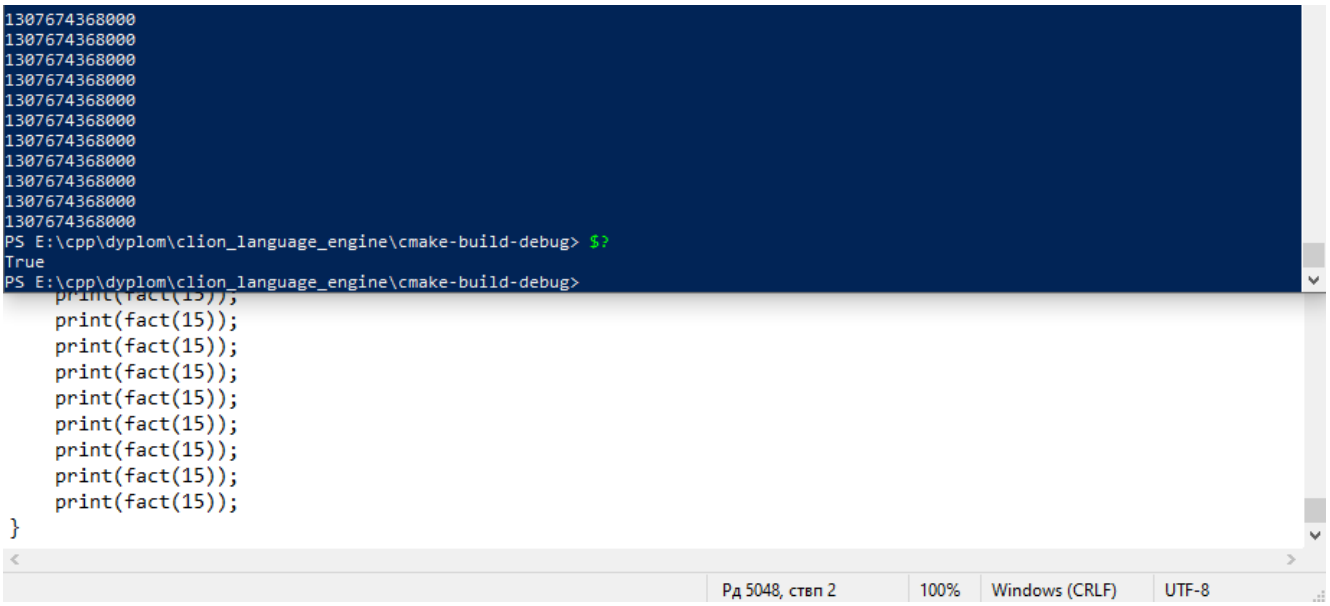

Рисунок 4.8 – Тестування роботи під навантаженням

Це тестування затверджує вірність роботи навіть під час довгого виконання операцій.

Для перевірки швидкості виконання задачі, слід зрівняти швидкість виконання задачі використовуючи мову програмування, що компілюється, наприклад C++, що буде виконувати задачу з попереднього тестування (див. рис. 4.9).

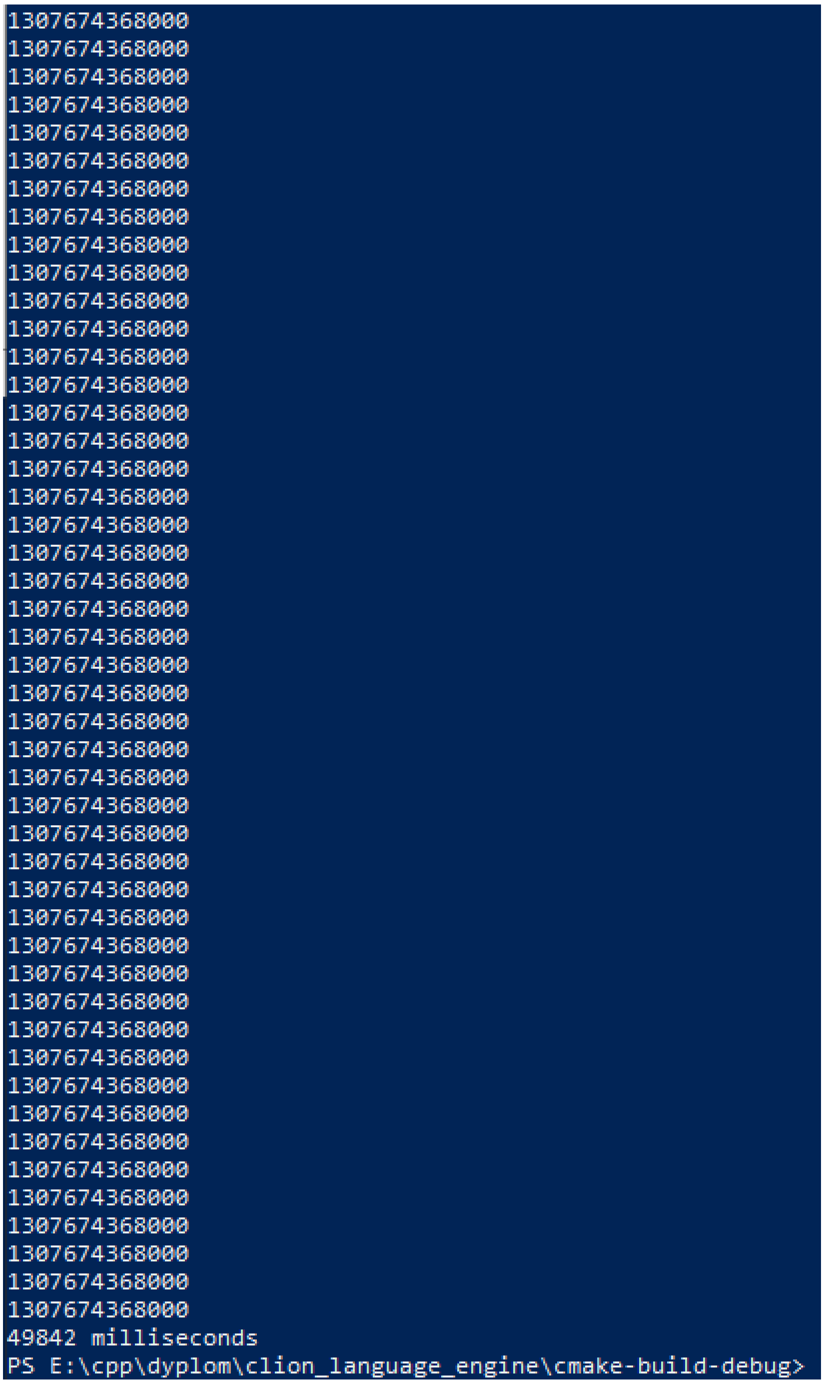

Рисунок 4.9 – Час виконання сценарію

Виконаємо аналоговий код C++ (див. рис. 4.10).

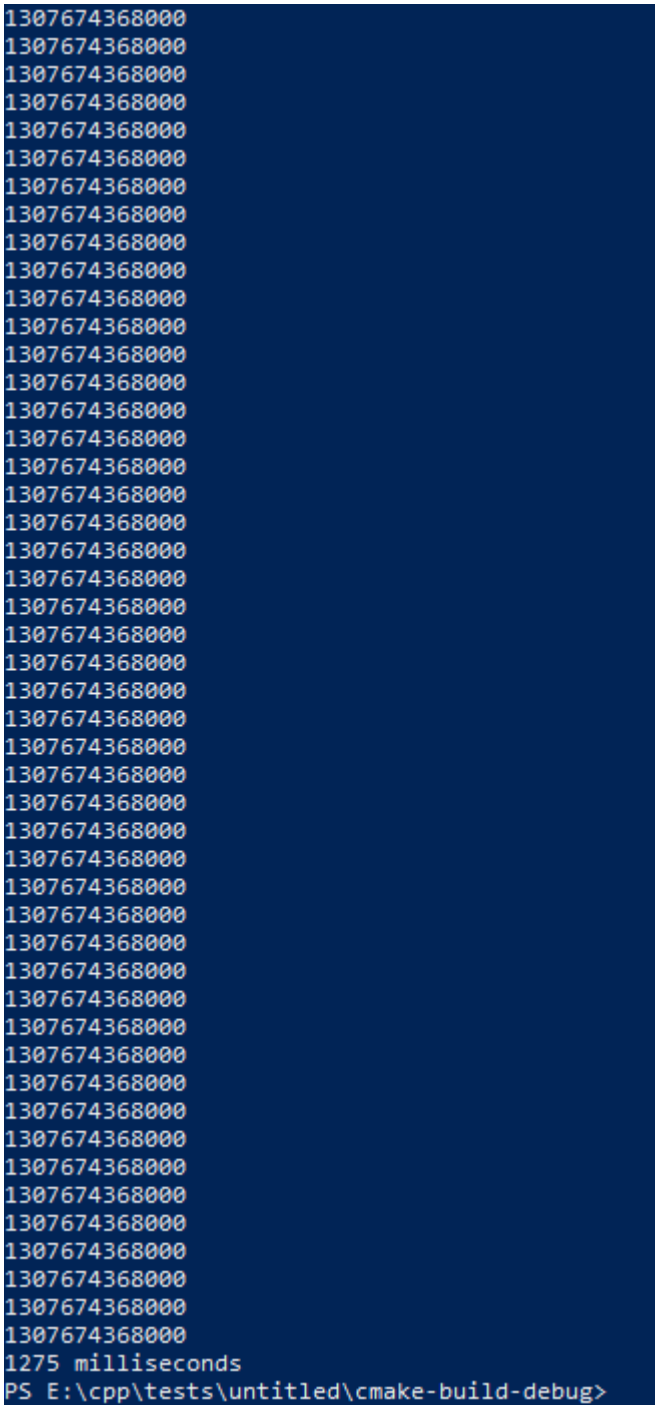

Рисунок 4.10 – Час виконання аналогового коду

Для визначення швидкості виконання слід порівняти отриманий час виконання, у випадку отримання результату у проміжку від 10 до 100 можна стверджувати, що швидкість виконання скрипту достатня.

В залежності від задачі швидкість виконання скомпільованого коду буде більшою ніж мов, що інтерпретують в проміжку від 10 до 1000 раз [10].

4.3 Розробка інструкції користувача

Розробимо інструкцію користувача для полегшення інтеграції рушія мови програмування в проект.

Для виконання сценарію необхідно ініціалізувати базові функції та викликати команду зчитування, з двома аргументами, ітератором на початок стрічки скрипту, та самої стрічки (див. рис. 4.11).

```
int main(int argc, char** argv)
{

    if(argc != 2)
        return 1;
    std::string scr = argv[1];

    function::init_base();
    std::ifstream script{scr};
    std::string str((std::istreambuf_iterator<char>(script)), std::istreambuf_iterator<char>());
    std::string::iterator it = str.begin();
    reader::read(it, str);

}
```

Рисунок 4.11 – Код для виконання сценарію

Для створення нової базової функції, необхідно написати функцію, що матиме два вхідних аргументи, перший посилання на об’єкт для повернення результату та список аргументів. Аргументи в масиві йдуть в порядку їх вказування в сценарію (див. рис. 4.12).

```
void pow2(object*& return_value, std::vector<object*> targets) {
    return_value = *targets[0] * *targets[0];
}
```

Рисунок 4.12 – Базова функція

Потім функцію треба додати до асоціативного масиву, ключ це стрічка по якій буде звірятися функція в сценарію та об’єкта функції в якому необхідно вказати вхідні типи даних та посилання на функцію (див. рис. 4.13).

```
function::function_list.emplace(
        std::pair<std::string, function*>
                { "pow2",
                  new function({ type_any }, {}, pow2)
                });
```

Рисунок 4.13 – Ініціалізація функції

Для ініціалізації об’єкта використовуючи мову програмування сценарію, необхідно використати другий аргумент для передачі коду та четвертий для назви вхідних елементів (див. рис. 4.14).

```
function::function_list.emplace(
        std::pair<std::string, function*>
                { "pow3",
                  new function({ type_any }, {"return a * a * a;"}, nullptr, {"a"})
                });
```

Рисунок 4.14 – Ініціалізація функції за допомогою мови сценаріїв

Після цих операцій можна почати виконання коду з зверненням до нових функцій (див. рис. 4.15).

```
PS E:\cpp\dyplom\clion_language_engine\cmake-build-debug> .\clion_language_engine.exe ..\user.txt
225
3375
```

user.txt: Блокнот

Файл Редагування Формат Вигляд Довідка

```
{
    print(pow2(15));
    print(pow3(15));
}
```

Рисунок 4.15 – Виклик нових базових функцій

4.4 Висновки

Розглянуто методи та методики тестування програмного забезпечення, обрано такі, що доцільно використати для тестування рушія мови програмування сценаріїв на сумісність з різними операційними системами, правильність виконання операцій та поведінку під навантаженням.

Проведене тестування підтвердило вірність виконання та роботу на різних дистрибутивах Linux та операційній системі Windows 10.

Розроблено інструкцію користувача для інтеграції рушія в проект для подальшого використання.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено скриптову мову програмування для підвищення рівня інтегрування авторського коду у системі керування ігрового рушія для подальшого модифікування та оновлення.

В першому розділі проведено аналіз предметної області, визначено задачі розробки, проведено аналіз аналогів інтерпритованих мов програмування та визначено основні критерії яким повинна відповідати розробка.

У другому розділі привидено аналіз методів і засобів розробки. Проведений аналіз показав пріоритетне використання мови програмування C++ у процесі розробки популярних ігрових рушіїв. Прийнято рішення використовувати C++ у процесі розробки, що дозволить виконувати код на усіх сучасних операційних системах при уникненні прямого виклику до системи. У розділі приведено результати аналізу способів збереження даних; розробку структури даних, що забезпечує збереження усіх необхідних типів даних для виконання сценарію; систему викликів функцій; базові функції.

У третьому розділ представлено розробку контролюючого блоку рушія мови програмування сценаріїв, що включає в себе: опис синтаксису інтерпретатора та варіантний аналіз станів; розробку системи доступу до даних та методи їх ініціалізації у сценарії; створення системи парсингу команд і даних для проведення обчислень; поведінку області видимості при виконанні функції та при її ініціалізації.

У четвертому розділі проведено аналіз способів тестування програмного забезпечення та вибір актуальних варіантів для перевірки роботи. Проведено тестування, що підтверджує вірність виконання обчислень, що проводить рушій виконання сценаріїв. Розроблено інструкцію користувача, що присвячена полегшенню інтеграції в існуючий проект.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Модифікація відеоігор [Електронний ресурс] Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Video_game_modding#Total_conversion

2. Статистика активних користувачів [Електронний ресурс] Режим доступу до ресурсу: https://steamdb.info/app/489830/graphs/

3. Статистика користувачів [Електронний ресурс] Режим доступу до ресурсу: https://store.steampowered.com/stats

4. Скриптові мови програмування [Електронний ресурс] Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Scripting_language#Extension/embeddable_languages

5. Закон Мура [Електронний ресурс] Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Moore%27s_law

6. Байт код [Електронний ресурс] Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Bytecode

7. С++ Мова програмування 4 видання \ Б'ярн Страуструп \\ 2013 р. – 1366 с.

8. Пріоритети С++ [Електронний ресурс] Режим доступу до ресурсу: https://en.cppreference.com/w/cpp/language/operator_precedence

9. Тестування програмного забезпечення [Електронний ресурс] Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Software_testing

10. Порівняння швидкості виконання мов програмування [Електронний ресурс] Режим доступу до ресурсу: https://attractivechaos.github.io/plb/

ДОДАТКИ

ДОДАТОК А.

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"____ "_____2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмної системи управління ігровими рушіями на основі скриптової мови» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доц. Рейда О. М.
"____"_____2022 р.
Виконав:
_____студент гр.2ПІ-18б Д. В. Доценко
"____"_____2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмної системи управління ігровими рушіями на основі скриптової мови».

Галузь застосування – ігрові рушії.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №66 від 24 березня 2022 року ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є створення скриптової мови програмування з можливістю легкого додання можливостей, що дозволить швидке створення системи управління компонентів ігрових рушіїв без потреби повторної компіляції проекту при зміні керуючого коду.

Призначення роботи – Розробка програмної системи управління ігровими рушіями на основі скриптової мови

4. Технічні вимоги

Створення, збереження та видалення об’єктів; система виклику функції та повернення результату з неї та перевіркою типів об’єктів; функції для виконання базових операцій; інтерпретатор скрипту; система області видимості об’єктів; пріоритетність операторів; ініціалізація функцій користувача з можливістю рекурсивного виклику; конвертування чисельних даних; вихідні дані для виконання скрипту – синтаксично вірний код; вихідні дані – результати виконання скрипта.

5. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БДР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз проблеми, обґрунтування актуальності розробки системи та постановка задачі	26.03.2022 – 10.04.2022
2	Розробка архітектури та алгоритмів роботи системи	11.04.2022 – 19.04.2022
3	Вибір середовища та мови розробки	20.04.2022 – 27.04.2022
4	Розробка програмного продукту	28.04.2022 – 10.05.2022
5	Тестування роботи системи	11.05.2022 – 19.05.2022
6	Оформлення матеріалів до захисту БДР	20.05.2022 – 10.06.2022

9. Порядок контролю та прийняття.

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

ДОДАТОК Б.

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмної системи управління ігровими рушіями на основі скриптової мови

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Рейда О. М.

Оригінальність	98,4%
Схожість	1,6%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Д. В. Доценко

Керівник роботи _____ Рейда О. М.

ДОДАТОК В. КОД РУШІЯ МОВИ ПРОГРАМУВАННЯ

В.1 Текст файлу «reader.h»

```
#ifndef CLION_LANGUAGE_ENGINE_READER_H
#define CLION_LANGUAGE_ENGINE_READER_H

#include "object.h"

class reader {
public:
    enum read_type{
        read_text,
        read_new_scope,
        read_close_scope,
    };
    struct next {
        read_type type;
        std::string::iterator it;
    };
    enum level{
        value = 00,
        level_function = 20,
        reverse = 30,
        multiply = 50,
        divide = 51,
        plus= 60,
        minus= 61,
        less = 90,
        less_eq = 91,
        more = 92,
        more_eq = 93,
        equal = 100,
        not_equal = 101,
        log_and = 140,
        log_or = 150,
        set = 200
    };
    struct hold{
        level eval;
        std::string val;
        hold* left = nullptr;
        hold* right = nullptr;
        object* func = nullptr;
        bool used = false;
        static void link_hold(std::vector<hold>& holds);
        static void sort(hold*& head){
            if(head){
                if(head->eval != value && head->eval != level_function && head->eval != reverse){
                    if(head->eval < head->left->eval){
                        {
                            auto old_head = head;
                            head = head->left;
                            auto head_old_right = head->right;
                            head->right = old_head;
                            old_head->left = head_old_right;
                        }
                        sort(head->right->left);
                        sort(head);
```

```
                } else {
                    sort(head->left);
                }
            }
        }
    }
};

    static void print_error(std::string::iterator it, const std::string& str, const std::string& error);
    static std::string get_word(std::string::iterator&, const std::string& str);
    static std::string get_any_word(std::string::iterator&, const std::string& str);
    static std::string get_line(std::string::iterator&, const std::string& str);
    static std::string get_evaluation(std::string::iterator&, const std::string& str);
    static std::string get_then(std::string::iterator&, const std::string& str);
    static std::string get_else(std::string::iterator&, const std::string& str);
    static char get_sign(std::string::iterator&, const std::string& str);
    static next get_next(std::string::iterator&, const std::string& str);
    static object* read(std::string::iterator&, const std::string& str);
    static std::string::iterator skip_whitespace(std::string::iterator, const std::string& str);
    static std::vector<hold> string_to_hold(std::string& str);
    static void hold_to_call(hold* head);
    static object_string* string_to_object(const std::string& str);
    static function* to_function(std::string::iterator&, const std::string& str);
    static void value_to_object(hold* head);
    static void op_to_object(hold* head);
    static std::vector<std::string> cut_input(const std::string& in);
    static object* evaluate(std::string str, type t);

};


#endif
```

B.2 Текст файлу «reader.cpp»

```
#include "reader.h"

void reader::print_error(std::string::iterator it, const std::string& str, const std::string& error) {
    std::cout << *it << " - " << (int)*it << " place: " << it - str.begin() << ": " + error + "\n";
}

std::string::iterator reader::skip_whitespace(std::string::iterator it, const std::string& str) {
    while(it != str.end()) {
        switch (*it) {
            case ' ':
            case '\n':
            case '\r': {
                break;
            }
            default:{
                return it;
            }
        }
        it++;
    }
    return it;
}

reader::next reader::get_next(std::string::iterator& it, const std::string& str) {
    while(it != str.end()){
        switch(*it){
            case ' ':
```

```
            case '\n':
            case '\r':{
                break;
            }
            case '{':{
                return {read_new_scope, it};
            }
            case '}':{
                return {read_close_scope, it};
            }
            case 'A'...'Z':
            case 'a'...'z':{
                return {read_text, it};
            }
            default:{
                print_error(it, str, "not implemented");
            }
        }
        it++;
    }
}

char reader::get_sign(std::string::iterator &it, const std::string &str) {
    it = skip_whitespace(it, str);
    while(it != str.end()){
        switch(*it){
            case ';':
            case '=':{
                return (*it++);
            }
            default:{
                print_error(it, str, "sign ; or = expected");
            }
        }
    }
    return ' ';
}

std::string reader::get_any_word(std::string::iterator &it, const std::string &str) {
    it = skip_whitespace(it, str);
    std::string string;
    if(*it == ""){
        string += *it;
        it++;
        while(it != str.end()){
            string += *it;
            if(*it == ""){
                it++;
                return string;
            }
            it++;
        }
    }
    if(*it == '\'){
        string += *it;
        it++;
        while(it != str.end()){
            string += *it;
            if(*it == '\'){
                it++;
                return string;
            }
```

```
            it++;
        }

    }
    while(it != str.end()){
        if(*it == '!'){
            it++;
            return string + "!";
        }
        switch(*it){
            default:{
                string += *it;
                break;
            }
            case '(':{

                int level = 0;
                char quotes = ' ';
                while(it != str.end()){
                    if(*it == '(' || *it == '{'){
                        level++;
                    } else if(*it == '\'' || *it == '"'){
                        if(quotes == *it){
                            quotes = ' ';
                        } else if (quotes == ' ') {
                            quotes = *it;
                        }
                    }

                    string += *it;
                    if(quotes == ' '){
                        if(level > 1 ){
                            if(*it == ')' || *it == '}'){
                                level--;
                            }
                        } else {
                            if(*it == ')'){
                                it++;
                                break;
                            }
                        }
                    }
                    it++;
                }
                return string;

            }
            case' ':
            case';':{
                return string;
            }
        }
        it++;
    }
    return string;
}
std::string reader::get_word(std::string::iterator &it, const std::string &str) {
    it = skip_whitespace(it, str);
    std::string string;
    while(it != str.end()){
        switch(*it){
            case '_':
```

```
            case 'a' ... 'z':
            case 'A' ... 'Z':{
                string += *it;
                break;
            }
            default:{
                return string;
            }
        }
        it++;
    }
    return string;
}



std::string reader::get_evaluation(std::string::iterator &it, const std::string &str) {
    it = skip_whitespace(it, str);
    std::string string;
    if(*it == '('){
        it++;
        int level = 0;
        char quotes = ' ';
        while(it != str.end()){
            if(*it == '(' || *it == '{'){
                level++;
            } else if(*it == '\'' || *it == '"'){
                if(quotes == *it){
                    quotes = ' ';
                } else if (quotes == ' ') {
                    quotes = *it;
                }
            }

            if(level > 0 || quotes != ' '){
                string += *it;
                if(*it == ')' || *it == '}'){
                    level--;
                }
            } else {
                if(*it != ')'){
                    string += *it;
                } else {
                    it++;
                    break;
                }
            }
            it++;

        }
        return string;
    }
    return string;
}


std::string reader::get_line(std::string::iterator &it, const std::string &str) {
    it = skip_whitespace(it, str);
    std::string string;
    while(it != str.end()){
        if(*it != ';'){
            string += *it;
```

```
        } else {
            it++;
            break;
        }
        it++;
    }
    return string;
}

std::string reader::get_else(std::string::iterator &it, const std::string &str) {
    std::string string;
    it = skip_whitespace(it, str);
    auto else_str = str.substr(it - str.begin(), 4);
    if(else_str  == "else"){
        it = skip_whitespace(it + 4, str);
        if(*it == '{'){
            int level = 0;
            char quotes = ' ';
            while(it != str.end()){
                if(*it == '{'){
                    level++;
                } else if(*it == '\'' || *it == '"'){
                    if(quotes == *it){
                        quotes = ' ';
                    } else if (quotes == ' ') {
                        quotes = *it;
                    }
                }


                string += *it;
                if(quotes == ' '){
                    if(level > 1 ){
                        if(*it == '}'){
                            level--;
                        }
                    } else {
                        if(*it == '}'){
                            it++;
                            break;
                        }
                    }
                }
                it++;

            }
            return string;
        }
    }
    return "";
}

std::string reader::get_then(std::string::iterator &it, const std::string &str) {
    std::string string;
    it = skip_whitespace(it, str);
    if(*it == '{'){
        int level = 0;
        char quotes = ' ';
        while(it != str.end()){
            if(*it == '{'){
                level++;
            } else if(*it == '\'' || *it == '"'){
```

```
            if(quotes == *it){
                quotes = ' ';
            } else if (quotes == ' ') {
                quotes = *it;
            }
        }

        string += *it;
        if(quotes == ' '){
            if(level > 1 ){
                if(*it == '}'){
                    level--;
                }
            } else {
                if(*it == '}'){
                    it++;
                    break;
                }
            }
        }
        it++;
    }
    return string;
  }
  if(*it == ';'){
    it++;
    return "";
  }
}

std::vector<std::string> reader::cut_input(const std::string &in) {
  int level = 0;
  char in_string = ' ';
  std::string str;
  std::vector<std::string> out;
  for(char c : in){
    switch (c) {
      case ',':{
        if (level == 0 && in_string == ' '){
          out.push_back(std::move(str));
          str = "";
        } else {
          str += c;
        }
        break;
      }
      case '"':
      case '\'':{
        if(in_string == c){
          in_string = ' ';
        }
        else if(in_string == ' '){
          in_string = c;
        }
        str += c;
        break;
      }
      case '{':
      case '(':{
        if(in_string == ' '){
          level++;
          str += c;
```

```
            }
            break;
        }
        case '}':
        case ')':{
            if(in_string == ' '){
                level--;
                if(level < 0){
                    std::cout << str << " to much for function\n";
                }
            }
            str += c;
            break;
        }
        default:{
            str += c;
        }
    }
  }
  out.push_back(std::move(str));
  return std::move(out);
}

std::vector<reader::hold> reader::string_to_hold(std::string &str) {
  std::string::iterator it = str.begin();
  std::vector<hold> holds;
  it = skip_whitespace(it, str);
  while(it != str.end() && *it != ';'){
    std::string val = get_any_word(it, str);
    holds.push_back({value, val});
    it = skip_whitespace(it, str);
  }
  for(auto& h : holds){
    if(h.val == "!"){
      h.eval = reverse;
    } else if (h.val == "+"){
      h.eval = plus;
    } else if (h.val == "-"){
      h.eval = minus;
    } else if (h.val == "/"){
      h.eval = divide;
    } else if (h.val == "*"){
      h.eval = multiply;
    } else if (h.val == "&&"){
      h.eval = log_and;
    } else if (h.val == "||"){
      h.eval = log_or;
    } else if (h.val == "<"){
      h.eval = less;
    } else if (h.val == ">"){
      h.eval = more;
    } else if (h.val == "<="){
      h.eval = less_eq;
    } else if (h.val == ">="){
      h.eval = more_eq;
    } else if (h.val == "=="){
      h.eval = equal;
    } else if (h.val == "!="){
      h.eval = not_equal;
    } else if (h.val == "="){
      h.eval = set;
    } else {
```

```
auto open = h.val.find('(');
auto close = h.val.find(')');
if (open < close && close != std::string::npos){
   std::string name = h.val.substr(0, open);

   std::string input;
   {
      std::string::iterator it = h.val.begin() + open + 1;
      int level = 0;
      char quotes = ' ';
      while(it != str.end()){
         if(*it == '(' || *it == '{'){
            level++;
         } else if(*it == '\'' || *it == '"'){
            if(quotes == *it){
               quotes = ' ';
            } else if (quotes == ' ') {
               quotes = *it;
            }
         }

         input += *it;
         if(quotes == ' '){
            if(level > 0 ){
               if(*it == ')' || *it == '}'){
                  level--;
               }
            } else {
               if(*it == ')'){
                  it++;
                  break;
               }
            }
         }
         it++;
      }
   }
   input = input.substr(0, input.size() - 1);
   h.eval = level_function;
   if(name.empty()){
      h.func = evaluate(input,  type_any);
      continue;
   }
   auto func = function::function_list[name];
   if(func == nullptr){
      auto obj = scope::current->find(name);
      if(obj->t == type_function){
         func = (function*)obj;
      } else {
         print_error(it, str, "function not found\n");
         throw;
      }
   }
   auto input_cut = cut_input(input);
   if(func->input.size() != input_cut.size()){
      if( !( input.empty() && func->input.empty() ) ){
         print_error(it, str, "arguments count mismatch\n");
         throw;
      }
   }
   std::vector<object*> inputs;
   inputs.reserve(func->input.size());
```

```
            for(int i = 0; i < func->input.size(); i++){
                inputs.push_back( evaluate( input_cut[i],  func->input[i]) );
            }
            h.func = new call(func, inputs);
            scope::current->hold.insert(std::pair<std::string, object*>{"function_call" + std::to_string((long long)h.func),
h.func});
        }
    }
  }
  hold::link_hold(holds);
  return std::move(holds);
}

object_string* reader::string_to_object(const std::string &str) {
  auto it = str.end();
  --it;
  while (*it == ' '){
    --it;
  }
  if(*it == "" && str.begin() != it){
    auto obj = new object_string(std::string(str.begin() + 1, it));
    scope::current->hold.insert( std::pair<std::string, object*>{ "temp" + std::to_string((long long)obj), obj} );
    return obj;
  }
  return new object_string("");
}

void reader::value_to_object(hold *head) {
  if(head->eval == value){
    if(head->val.empty()){
      head->func = new object_long(0);
      return;
    }
    skip_whitespace(head->val.begin(), head->val);
    if(head->val[0] == ""){
      head->func = string_to_object(head->val);
    } else if(head->val[0] >= '0' && head->val[0] <= '9'){
      if(head->val.find('.') != std::string::npos){
        auto obj = new object_double(std::stod(head->val));
        scope::current->hold.insert( std::pair<std::string, object*>{ "temp" + std::to_string((long long)obj), obj} );
        head->func = obj;
      } else {
        auto obj = new object_long(std::stoll(head->val));
        scope::current->hold.insert( std::pair<std::string, object*>{ "temp" + std::to_string((long long)obj), obj} );
        head->func = obj;
      }
    } else if(head->val == "true"){
      auto obj = new object_bool(true);
      scope::current->hold.insert( std::pair<std::string, object*>{ "temp" + std::to_string((long long)obj), obj} );
      head->func = obj;
    } else if(head->val == "false"){
      auto obj = new object_bool(false);
      scope::current->hold.insert( std::pair<std::string, object*>{ "temp" + std::to_string((long long)obj), obj} );
      head->func = obj;
    } else {
      auto obj = scope::current->find(head->val);
      if(obj == nullptr){
        std::cout << head->val << " object not found\n";
        throw;
      }
      head->func = obj;
    }
```

```
    }
    if(head->left){
        value_to_object(head->left);
    }
    if(head->right){
        value_to_object(head->right);
    }
}

void reader::op_to_object(hold *head) {
    if(head->left){
        op_to_object(head->left);
    }
    if(head->right){
        op_to_object(head->right);
    }
    call* obj = nullptr;
    switch (head->eval) {
        case plus:{
            obj = new call( function::function_list["plus"], {head->left->func, head->right->func} );
            break;
        }
        case minus:{
            obj = new call( function::function_list["minus"], {head->left->func, head->right->func} );
            break;
        }
        case multiply:{
            obj = new call( function::function_list["multiply"], {head->left->func, head->right->func} );
            break;
        }
        case divide:{
            obj = new call( function::function_list["divide"], {head->left->func, head->right->func} );
            break;
        }
        case log_or:{
            obj = new call( function::function_list["or"], {head->left->func, head->right->func} );
            break;
        }
        case log_and:{
            obj = new call( function::function_list["and"], {head->left->func, head->right->func} );
            break;
        }
        case reverse:{
            obj = new call( function::function_list["reverse"], {head->left->func} );
            break;
        }
        case less:{
            obj = new call( function::function_list["less"], {head->left->func, head->right->func} );
            break;
        }
        case more:{
            obj = new call( function::function_list["more"], {head->left->func, head->right->func} );
            break;
        }
        case less_eq:{
            obj = new call( function::function_list["less_eq"], {head->left->func, head->right->func} );
            break;
        }
        case more_eq:{
            obj = new call( function::function_list["more_eq"], {head->left->func, head->right->func} );
            break;
        }
```

```
        case equal:{
            obj = new call( function::function_list["equal"], {head->left->func, head->right->func} );
            break;
        }
        case not_equal:{
            obj = new call( function::function_list["not_equal"], {head->left->func, head->right->func} );
            break;
        }
        case set:{
            obj = new call( function::function_list["set"], {head->left->func, head->right->func} );
            break;
        }
    }
    if(obj != nullptr){
        scope::current->hold.insert( std::pair<std::string, object*>{ "call" + std::to_string((long long)obj), obj} );
        head->func = obj;
    }
}

void reader::hold_to_call(hold *head) {
    value_to_object(head);
    op_to_object(head);
}

void reader::hold::link_hold(std::vector<hold> &holds) {
    for(auto it = holds.begin() ; it < holds.end(); ++it){
        switch(it->eval){
            case level_function:
            case value:
                break;
            case reverse:{
                it->left = (it + 1).base();
                it->left->used = true;
                break;
            }
            case less_eq:
            case less:
            case more:
            case more_eq:
            case equal:
            case not_equal:
            case set:
            case plus:
            case minus:
            case multiply:
            case divide:
            case log_or:
            case log_and:{
                auto it_copy = it - 1;
                while (it_copy->used){
                    if(it_copy == holds.begin()){
                        throw;
                    }
                    it_copy--;
                }
                it->left = (it_copy).base();
                it->left->used = true;
                it->right = (it + 1).base();
                it->right->used = true;
            }
        }
    }
```

```
}

object *reader::evaluate(std::string str, type t) {

    auto holds = string_to_hold(str);
    hold* head;
    for(auto& h : holds){
        if(!h.used){
            head = &h;
        }
    }
    hold::sort(head);
    reader::hold_to_call(head);
    object* ret;
    if(head->func->t == type_call){
        ret = ((call*)head->func)->work();
    }
    else {
        ret = head->func;
    }
    if(t == type_any && ret == nullptr){
        return ret;
    }
    if(ret->t != t && t != type_any){
        if(t == type_double){
            if(ret->t == type_long){
                auto obj = object_double( ((object_long*)ret)->value );
                ret->set( &obj );
                return ret;
            }
        }
        if(t == type_long){
            if(ret->t == type_double){
                auto obj = object_long( ((object_double*)ret)->value );
                ret->set( &obj );
                return ret;
            }
        }
    } else {
        return ret;
    }
    std::cout << "type mismatch\n";
    throw;
}

object* reader::read(std::string::iterator& it, const std::string &str) {
    while (it != str.end()){
        auto it_copy = it;
        auto ret = get_next(it, str);
        switch(ret.type){
            case read_new_scope:{
                auto new_scope = new scope(scope::current);
                scope::current = new_scope;
                it++;
                break;
            }
            case read_close_scope:{
                auto old = scope::current;
                scope::current = scope::current->parent;
                old->free_scope();
                delete old;
                scope::delete_objects();
```

```
    it++;
    break;
}
case read_text:{
    std::string word = get_word(it, str);
    if(word == "long"){
        std::string name = get_word(it, str);
        object_long* obj;
        switch(get_sign(it, str)){
            case '=': {
                obj = (object_long*)evaluate(get_line(it, str), type_long);
                break;
            }
            case ';':{
                obj = new object_long(0);
                break;
            }
            default:{
                print_error(it, str, "unexpected sign");
            }
        }
        scope::current->hold.insert(std::pair<std::string, object*>{name, obj});
    } else if(word == "double"){
        std::string name = get_word(it, str);
        object_double* obj;
        switch(get_sign(it, str)){
            case '=': {
                obj = (object_double*)evaluate(get_line(it, str), type_double);
                break;
            }
            case ';':{
                obj = new object_double(0);
                break;
            }
            default:{
                print_error(it, str, "unexpected sign");
            }
        }
        scope::current->hold.insert(std::pair<std::string, object*>{name, obj});
    } else if(word == "string"){
        std::string name = get_word(it, str);
        object_string* obj;
        switch(get_sign(it, str)){
            case '=': {
                obj = (object_string*)evaluate(get_line(it, str), type_string);
                break;
            }
            case ';':{
                obj = new object_string("");
                break;
            }
            default:{
                print_error(it, str, "unexpected sign");
            }
        }
        scope::current->hold.insert(std::pair<std::string, object*>{name, obj});
    } else if(word == "bool"){
        std::string name = get_word(it, str);
        object_bool* obj;
        switch(get_sign(it, str)){
            case '=': {
                obj = (object_bool*)evaluate(get_line(it, str), type_bool);
```

```
                    break;
                }
                case ';':{
                    obj = new object_bool(false);
                    break;
                }
                default:{
                    print_error(it, str, "unexpected sign");
                }
            }
            scope::current->hold.insert(std::pair<std::string, object*>{name, obj});
        } else if(word == "function"){
            std::string name = get_word(it, str);
            auto obj = to_function(it, str);
            scope::current->hold.insert(std::pair<std::string, object*>{name, obj});
        } else if(word == "return"){
            auto value = evaluate(get_line(it, str), type_any);
            return value;
        } else if(word == "if"){
            auto value = evaluate(get_evaluation(it, str), type_any);
            auto then = get_then(it, str);
            if(!then.empty()){
                auto else_ = get_else(it, str);
                if(*value){
                    auto then_it = then.begin();
                    auto obj = read(then_it, then);
                    if(obj != nullptr){
                        return obj;
                    }
                } else {
                    if(!else_.empty()){
                        auto else_it = else_.begin();
                        auto obj = read(else_it, else_);
                        if(obj != nullptr){
                            return obj;
                        }
                    }
                }
            } else {
                evaluate(get_line(it_copy, str), type_any);
                it = it_copy;
            }
        }
    }
  }
  return nullptr;
}

function *reader::to_function(std::string::iterator &it, const std::string &str) {
  it = skip_whitespace(it, str);
  std::vector<type> types;
  std::vector<std::string> names;
  if(*it == '('){
    ++it;
    while (*it != ')'){
      auto string = get_word(it, str);
      if(!string.empty()){
        if(string == "long"){
          types.push_back(type_long);
        } else if(string == "double"){
          types.push_back(type_double);
```

```
            } else if(string == "bool"){
                types.push_back(type_bool);
            } else if(string == "any"){
                types.push_back(type_any);
            } else if(string == "string"){
                types.push_back(type_string);
            }
            names.push_back( get_word(it, str) );
            it = skip_whitespace(it, str);
            if(*it == ',')
                it++;
        }
    }
  }
  it++;
  std::string function_body = get_then(it, str);
  function_body = function_body.substr(1, function_body.size() - 2);
  auto obj = new function( types, function_body);
  obj->targets_name = std::move(names);
  return obj;
}
```

B.3 Текст файлу «object.h»

```
#ifndef object_h
#define object_h

#include <utility>
#include <vector>
#include <string>
#include <map>
#include <iostream>
#include <set>
#include <cstring>

enum type
{
        type_long,
        type_double,
        type_string,
        type_call,
        type_function,
        type_void,
        type_any,
  type_bool
};


class function;

class object
{
public:
        typedef void (*function_)(object*& return_value, std::vector<object*> targets);
        explicit object(type t) : t(t) {};
        virtual void print_object() = 0;
        virtual void set(object* o) = 0;
        type t;
#pragma region operators

#pragma region line_operators
        friend object* operator+(object& l, const object& r);
```

```
        friend object* operator-(object& l, const object& r);
        friend object* operator*(object& l, const object& r);
        friend object* operator/(object& l, const object& r);
#pragma endregion
#pragma region assigmeent_operators

   object& operator+=(const object& r);

        object& operator-=(const object& r);

        object& operator*=(const object& r);

        object& operator/=(const object& r);

#pragma endregion
#pragma region comparison_operators
        friend bool operator==(const object& l, const object& r);
        friend bool operator!=(const object& l, const object& r);
        friend bool operator<(const object& l, const object& r);
        friend bool operator<=(const object& l, const object& r);
        friend bool operator>(const object& l, const object& r);
        friend bool operator>=(const object& l, const object& r);
#pragma endregion

#pragma region conversion_operators
   explicit operator bool();
#pragma endregion

#pragma endregion

};

class scope : public object {
public:
   explicit scope(scope* p) : object(type_void), parent(p) {};
   scope* parent;
   static scope* current;
   static std::set<object*> del;
   std::map<std::string, object*> hold;
   void print_object() override {};
   void set(object* o) override {};

   virtual object* find(const std::string& str){
      auto obj = hold[str];
      if(obj == nullptr){
         obj = parent->find(str);
      }
      return obj;
   }
   static void delete_objects(){
      for(auto p : del){
         if(p)
            delete p;
      }
      del.clear();
   }
   virtual void free_scope(){
      current = parent;
      for(const auto& pair : hold){
         del.insert(pair.second);
      }
      hold.clear();
```

```
    }
};

class scope_close : public scope {
public:
    void free_scope() override{}
    scope_close() : scope(nullptr){}
};

class call : public object {
public:
    ~call(){
    }
        call(function* func, std::vector<object*> t) : object(type_call), f(func), targets(std::move(t)) {};
        bool check_types();
        object* work();
        void print_object() override;
        void set(object* o) override {
            if (t == o->t) {
                *this = *(call*)o;
            }
        }

        std::vector<object*> targets;
        object* return_value = nullptr;
        function *f;
};

class function : public object {
public:
    ~function();
        function(std::vector<type> in, std::string c, function_ w = nullptr, std::vector<std::string> names={}) :
object(type_function), input(std::move(in)), work(w), calls(std::move(c)), targets_name(names) {};
        static void init_base();
        static std::map<std::string, function*> function_list;
        void print_object() override {};
        void set(object* o) override {
            if (t == o->t) {
                *this = *(function*)o;
            }
        }
    std::vector<std::string> targets_name;
    std::string calls;

        function_ work = nullptr;
        std::vector<type> input;
        type f_return = type_any;
};

class object_bool : public object {
public:
    explicit object_bool(bool v) : object(type_bool), value(v) {};
    void print_object() override {
        std::cout << (value ? "true" : "false") << "\n";
    };
    void set(object* o) override {
        if (t == o->t) {
            *this = *(object_bool*)o;
        }
    }
    bool value;
#pragma region operators
```

```
    explicit operator bool() const{
        return value;
    }
#pragma endregion
};

class object_long : public object {
public:
        explicit object_long(long long v) : object(type_long), value(v) {};
        void print_object() override {
                std::cout << value << "\n";
        };
        void set(object* o) override {
                if (t == o->t) {
                        *this = *(object_long*)o;
                }
        }
        long long value;
#pragma region operators

#pragma region line_operators

        friend object_long operator+(object_long l, const object& r);
        friend object_long operator-(object_long l, const object& r);
        friend object_long operator*(object_long l, const object& r);
        friend object_long operator/(object_long l, const object& r);
#pragma endregion
#pragma region assigmeent_operators
        object_long& operator+=(const object& r);

        object_long& operator-=(const object& r);

        object_long& operator*=(const object& r);

        object_long& operator/=(const object& r);

#pragma endregion
#pragma region comparison_operators
        friend bool operator==(const object_long& l, const object& r);
        friend bool operator!=(const object_long& l, const object& r);
        friend bool operator<(const object_long& l, const object& r);
        friend bool operator<=(const object_long& l, const object& r);
        friend bool operator>(const object_long& l, const object& r);
        friend bool operator>=(const object_long& l, const object& r);
#pragma endregion
#pragma region conversion_operators
    explicit operator bool() const{
        return (bool)value;
    }
#pragma endregion
#pragma endregion
};


class object_double : public object {
public:
        explicit object_double(double v) : object(type_double), value(v) {
        };
        void print_object() override {
                std::cout << value << "\n";
        };
        void set(object* o) override {
```

```
                if (t == o->t) {
                        *this = *(object_double*)o;
                }
        }
        double value;
#pragma region operators

#pragma region line_operators
        friend object_double operator+(object_double l, const object& r);

        friend object_double operator-(object_double l, const object& r);

        friend object_double operator*(object_double l, const object& r);
        friend object_double operator/(object_double l, const object& r);
#pragma endregion
#pragma region assigmeent_operators
        object_double& operator+=(const object& r);
        object_double& operator-=(const object& r);
        object_double& operator*=(const object& r);
        object_double& operator/=(const object& r);
#pragma endregion
#pragma region comparison_operators
        friend bool operator==(const object_double& l, const object& r);
        friend bool operator!=(const object_double& l, const object& r);
        friend bool operator<(const object_double& l, const object& r);
        friend bool operator>(const object_double& l, const object& r);
        friend bool operator<=(const object_double& l, const object& r);
        friend bool operator>=(const object_double& l, const object& r);
#pragma endregion
#pragma region conversion_operators
   explicit operator bool() const{
      return (bool)value;
   }
#pragma endregion

#pragma endregion
};


class object_string : public object {
public:
        explicit object_string(std::string v) : object(type_string), value(std::move(v)) {};
        void print_object() override {
      std::cout << value << "\n";
   };
        void set(object* o) override {
                if (t == o->t) {
                        *this = *(object_string*)o;
                }
        }
        std::string value;
#pragma region operators

#pragma region line_operators
        friend object_string operator+(object_string l, const object& r);

        friend object_string operator*(object_string l, const object& r);
#pragma endregion
#pragma region assigmeent_operators
        object_string& operator+=(const object& r);
        object_string& operator*=(const object& r);
#pragma endregion
```

```
#pragma region comparison_operators
	friend bool operator==(const object_string& l, const object& r);
	friend bool operator!=(const object_string& l, const object& r);
#pragma endregion
#pragma region conversion_operators
   explicit operator bool() const{
      return !value.empty();
   }
#pragma endregion
#pragma endregion
};


#endif
```

B.4 Текст файлу «object.cpp»

```
#include "object.h"
#include "reader.h"
#include "functions.h"

std::map<std::string, function*> function::function_list;
scope* scope::current = new scope_close();
std::set<object*> scope::del{};


void function::init_base()
{
   function_list.emplace(std::pair<std::string, function*>{ "plus", new function({ type_any, type_any }, {}, plus) });
   function_list.emplace(std::pair<std::string, function*>{ "minus", new function({ type_any, type_any }, {}, minus) });
   function_list.emplace(std::pair<std::string, function*>{ "multiply", new function({ type_any, type_any }, {}, multiply)
});
   function_list.emplace(std::pair<std::string, function*>{ "divide", new function({ type_any, type_any }, {}, divide) });
   function_list.emplace(std::pair<std::string, function*>{ "print", new function({ type_any }, {}, print) });
   function_list.emplace(std::pair<std::string, function*>{ "set", new function({ type_any, type_any }, {}, set_value) });
   function_list.emplace(std::pair<std::string, function*>{ "or", new function({ type_any, type_any }, {}, f_or) });
   function_list.emplace(std::pair<std::string, function*>{ "reverse", new function({ type_any }, {}, f_reverse) });
   function_list.emplace(std::pair<std::string, function*>{ "and", new function({ type_any, type_any }, {}, f_and) });
   function_list.emplace(std::pair<std::string, function*>{ "less", new function({ type_any, type_any }, {}, less) });
   function_list.emplace(std::pair<std::string, function*>{ "more", new function({ type_any, type_any }, {}, more) });
   function_list.emplace(std::pair<std::string, function*>{ "less_eq", new function({ type_any, type_any }, {}, less_eq) });
   function_list.emplace(std::pair<std::string, function*>{ "more_eq", new function({ type_any, type_any }, {}, more_eq)
});
   function_list.emplace(std::pair<std::string, function*>{ "equal", new function({ type_any, type_any }, {}, equal) });
   function_list.emplace(std::pair<std::string, function*>{ "not_equal", new function({ type_any, type_any }, {},
not_equal) });
}

bool call::check_types() {
	if (f) {
		if (f->input.size() == targets.size()) {
			if (!f->input.empty()) {
				int i = 0;
				for (const auto& type : f->input) {
					if ( !(type == type_any || type == targets[i]->t) ) {
//check for nullptr
						if(! (targets[i]->t == type_call && type == ( (call*)targets[i] )->f-
>f_return) )
		std::cout << "type mismatch\n";
							return false;
					}
```

```
                                        i++;
                                }
                        }
                        return true;
                }
        }
        return false;
}

void call::print_object() {
        std::cout << "call:" << targets.size() << ":" << f->work << "\n";
}

object* call::work() {
        if (check_types()) {
                {       int i = 0;
                        for (auto& target : targets) {
                                if (target->t == type_call && f->input[i] != type_call) {
                                        target = ((call*)target)->work();
                                }
                                i++;
                        }
                }
                if (f->work != nullptr) {
                        f->work(return_value, targets);
                }
                else {
        auto new_scope = new scope(scope::current);
        scope::current = new_scope;
        for(int i = 0; i < f->targets_name.size(); ++i){
           scope::current->hold.insert(std::pair<std::string, object*>{f->targets_name[i], targets[i]});
        }
        auto it = f->calls.begin();
        auto ret = reader::read(it, f->calls);
        std::string name;
        for(const auto& p : scope::current->hold){
           if(p.second == ret){
              scope::current->parent->hold.insert(std::pair<std::string, object*>{p.first, p.second});
              name = p.first;
              break;
           }
        }
        if(!name.empty()){
           scope::current->hold.erase(name);
        }

        auto old = scope::current;
        scope::current = scope::current->parent;
        old->free_scope();
        delete old;

        return ret;
                }
                return return_value;
         }
         return nullptr;
}


#pragma region object_operators
#pragma region line_operators
object* operator+(object& l, const object& r) {
```

```
            if (l.t == type_long) {
        auto obj = new object_long( *(object_long*)&l + r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
                    return obj;
            }
            else if (l.t == type_double) {
        auto obj = new object_double( *(object_double*)&l + r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
        return obj;
            }
            else if (l.t == type_string) {
        auto obj = new object_string( *(object_string*)&l + r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
        return obj;
            }
            throw;
}
object* operator-(object& l, const object& r) {
    if (l.t == type_long) {
        auto obj = new object_long( *(object_long*)&l - r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
        return obj;
    }
    else if (l.t == type_double) {
        auto obj = new object_double( *(object_double*)&l - r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
        return obj;
    }
            throw;
}
object* operator/(object& l, const object& r) {
    if (l.t == type_long) {
        auto obj = new object_long( *(object_long*)&l / r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
        return obj;
    }
    else if (l.t == type_double) {
        auto obj = new object_double( *(object_double*)&l / r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
        return obj;
    }
            throw;
}
object* operator*(object& l, const object& r) {
    if (l.t == type_long) {
        auto obj = new object_long( *(object_long*)&l * r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
        return obj;
    }
    else if (l.t == type_double) {
        auto obj = new object_double( *(object_double*)&l * r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
        return obj;
    }
    else if (l.t == type_string) {
        auto obj = new object_string( *(object_string*)&l * r );
        scope::current->hold.insert(std::pair<std::string, object*>{"temp" + std::to_string((long long)obj), obj});
        return obj;
    }
            throw;
}
#pragma endregion
```

```
#pragma region assigment_operators

object& object::operator+=(const object& r) {
	if (t == type_long) {
		return *(object_long*)this += r;
	}
	else if (t == type_double) {
		return *(object_double*)this += r;
	}
	else if (t == type_string) {
		return *(object_string*)this += r;
	}
	throw;
}
object& object::operator-=(const object& r) {
	if (t == type_long) {
		return *(object_long*)this -= r;
	}
	else if (t == type_double) {
		return *(object_double*)this -= r;
	}
	throw;
}
object& object::operator/=(const object& r) {
	if (t == type_long) {
		return *(object_long*)this /= r;
	}
	else if (t == type_double) {
		return *(object_double*)this /= r;
	}
	throw;
}
object& object::operator*=(const object& r) {
	if (t == type_long) {
		return *(object_long*)this *= r;
	}
	else if (t == type_double) {
		return *(object_double*)this *= r;
	}
	else if (t == type_string) {
		return *(object_string*)this *= r;
	}
	throw;
}
#pragma endregion
#pragma region comparison_operators

bool operator==(const object& l, const object& r) {
	if (l.t == type_long) {
		return (object_long&)l == r;
	}
	else if (l.t == type_double) {
		return (object_double&)l == r;
	}
	else if (l.t == type_string) {
		return (object_string&)l == r;
	}
	throw;
}
bool operator!=(const object& l, const object& r) {
	if (l.t == type_long) {
		return (object_long&)l != r;
```

```
		}
		else if (l.t == type_double) {
			return (object_double&)l != r;
		}
		else if (l.t == type_string) {
			return (object_string&)l != r;
		}
		throw;
}
bool operator>(const object& l, const object& r) {

		if (l.t == type_long) {
			return (object_long&)l > r;
		}
		else if (l.t == type_double) {
			return (object_double&)l > r;
		}
		throw;
}
bool operator<(const object& l, const object& r) {

		if (l.t == type_long) {
			return (object_long&)l < r;
		}
		else if (l.t == type_double) {
			return (object_double&)l < r;
		}
		throw;
}
bool operator>=(const object& l, const object& r) {

		if (l.t == type_long) {
			return (object_long&)l >= r;
		}
		else if (l.t == type_double) {
			return (object_double&)l >= r;
		}
		throw;
}
bool operator<=(const object& l, const object& r) {

		if (l.t == type_long) {
			return (object_long&)l <= r;
		}
		else if (l.t == type_double) {
			return (object_double&)l <= r;
		}
		throw;
}
#pragma endregion
#pragma region conversion_operators
object::operator bool(){
   if(t == type_long){
      return (bool)*((object_long*)this);
   } else if(t == type_double){
      return (bool)*((object_double*)this);
   } else if(t == type_bool){
      return (bool)*((object_bool*)this);
   } else if(t == type_string){
      return (bool)*((object_string*)this);
   }
   return false;
```

```
}
#pragma endregion
#pragma endregion
#pragma region long_operators

#pragma region line_operators

object_long operator+(object_long l, const object& r) {
	if (r.t == type_long) {
		l.value += ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		l.value += ((object_double*)&r)->value;
	}
	return l;
}

object_long operator-(object_long l, const object& r) {
	if (r.t == type_long) {
		l.value -= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		l.value -= ((object_double*)&r)->value;
	}
	return l;
}

object_long operator*(object_long l, const object& r) {
	if (r.t == type_long) {
		l.value *= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		l.value *= ((object_double*)&r)->value;
	}
	return l;
}

object_long operator/(object_long l, const object& r) {
	if (r.t == type_long) {
		l.value /= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		l.value /= ((object_double*)&r)->value;
	}
	return l;
}

#pragma endregion
#pragma region assigment_operators

object_long& object_long::operator+=(const object& r) {
	if (r.t == type_long) {
		value += ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		value += ((object_double*)&r)->value;
	}
	return *this;
}

object_long& object_long::operator-=(const object& r) {
	if (r.t == type_long) {
```

```
                value -= ((object_long*)&r)->value;
        }
        else if (r.t == type_double) {
                value -= ((object_double*)&r)->value;
        }
        return *this;
}

object_long& object_long::operator*=(const object& r) {
        if (r.t == type_long) {
                value *= ((object_long*)&r)->value;
        }
        else if (r.t == type_double) {
                value *= ((object_double*)&r)->value;
        }
        return *this;
}

object_long& object_long::operator/=(const object& r) {
        if (r.t == type_long) {
                value /= ((object_long*)&r)->value;
        }
        else if (r.t == type_double) {
                value /= ((object_double*)&r)->value;
        }
        return *this;
}

#pragma endregion
#pragma region comparison_operators

bool operator==(const object_long& l, const object& r) {
        if (r.t == type_long) {
                return l.value == ((object_long*)&r)->value;
        }
        else if (r.t == type_double) {
                return l.value == ((object_double*)&r)->value;
        }
        throw;
}
bool operator!=(const object_long& l, const object& r) {
        if (r.t == type_long) {
                return l.value != ((object_long*)&r)->value;
        }
        else if (r.t == type_double) {
                return l.value != ((object_double*)&r)->value;
        }
        throw;
}
bool operator>(const object_long& l, const object& r) {
        if (r.t == type_long) {
                return l.value > ((object_long*)&r)->value;
        }
        else if (r.t == type_double) {
                return l.value > ((object_double*)&r)->value;
        }
        throw;
}
bool operator<(const object_long& l, const object& r) {
        if (r.t == type_long) {
                return l.value < ((object_long*)&r)->value;
        }
```

```
	else if (r.t == type_double) {
		return l.value < ((object_double*)&r)->value;
	}
	throw;
}
bool operator>=(const object_long& l, const object& r) {
	if (r.t == type_long) {
		return l.value >= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		return l.value >= ((object_double*)&r)->value;
	}
	throw;
}
bool operator<=(const object_long& l, const object& r) {
	if (r.t == type_long) {
		return l.value <= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		return l.value <= ((object_double*)&r)->value;
	}
	throw;
}

#pragma endregion

#pragma endregion
#pragma region double_operators

#pragma region line_operators

object_double operator+(object_double l, const object& r) {
	if (r.t == type_long) {
		l.value += ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		l.value += ((object_double*)&r)->value;
	}
	return l;
}

object_double operator-(object_double l, const object& r) {
	if (r.t == type_long) {
		l.value -= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		l.value -= ((object_double*)&r)->value;
	}
	return l;
}

object_double operator*(object_double l, const object& r) {
	if (r.t == type_long) {
		l.value *= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		l.value *= ((object_double*)&r)->value;
	}
	return l;
}

object_double operator/(object_double l, const object& r) {
```

```
	if (r.t == type_long) {
		l.value /= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		l.value /= ((object_double*)&r)->value;
	}
	return l;
}

#pragma endregion
#pragma region assigment_operators

object_double& object_double::operator+=(const object& r) {
	if (r.t == type_long) {
		value += ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		value += ((object_double*)&r)->value;
	}
	return *this;
}

object_double& object_double::operator-=(const object& r) {
	if (r.t == type_long) {
		value -= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		value -= ((object_double*)&r)->value;
	}
	return *this;
}

object_double& object_double::operator*=(const object& r) {
	if (r.t == type_long) {
		value *= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		value *= ((object_double*)&r)->value;
	}
	return *this;
}

object_double& object_double::operator/=(const object& r) {
	if (r.t == type_long) {
		value /= ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		value /= ((object_double*)&r)->value;
	}
	return *this;
}

#pragma endregion
#pragma region comparison_operators

bool operator==(const object_double& l, const object& r) {
	if (r.t == type_long) {
		return l.value == ((object_long*)&r)->value;
	}
	else if (r.t == type_double) {
		return l.value == ((object_double*)&r)->value;
	}
```

```
		throw;
}
bool operator!=(const object_double& l, const object& r) {
		if (r.t == type_long) {
			return l.value != ((object_long*)&r)->value;
		}
		else if (r.t == type_double) {
			return l.value != ((object_double*)&r)->value;
		}
		throw;
}
bool operator>(const object_double& l, const object& r) {
		if (r.t == type_long) {
			return l.value > ((object_long*)&r)->value;
		}
		else if (r.t == type_double) {
			return l.value > ((object_double*)&r)->value;
		}
		throw;
}
bool operator<(const object_double& l, const object& r) {
		if (r.t == type_long) {
			return l.value < ((object_long*)&r)->value;
		}
		else if (r.t == type_double) {
			return l.value < ((object_double*)&r)->value;
		}
		throw;
}
bool operator>=(const object_double& l, const object& r) {
		if (r.t == type_long) {
			return l.value >= ((object_long*)&r)->value;
		}
		else if (r.t == type_double) {
			return l.value >= ((object_double*)&r)->value;
		}
		throw;
}
bool operator<=(const object_double& l, const object& r) {
		if (r.t == type_long) {
			return l.value <= ((object_long*)&r)->value;
		}
		else if (r.t == type_double) {
			return l.value <= ((object_double*)&r)->value;
		}
		throw;
}

#pragma endregion

#pragma endregion
#pragma region string_operators

#pragma region line_operators

object_string operator+(object_string l, const object& r) {
		if (r.t == type_long) {
			l.value += std::to_string( ( (object_long*)&r )->value );
		}
		else if (r.t == type_double) {
			l.value += std::to_string( ( (object_double*)&r)->value );
		}
```

```
	else if (r.t == type_string) {
		l.value += ((object_string*)&r)->value;
	}
	return l;
}

object_string operator*(object_string l, const object& r) {
	if (r.t == type_long) {
		long long val = ((object_long*)&r)->value;
		if (l.value.length() > 0 && val > 1) {
			std::string copy = l.value;
			for (int i = 1; i < val; ++i) {
				l.value += copy;
			}
		}
	}
	return l;
}


#pragma endregion
#pragma region assigment_operators


object_string& object_string::operator+=(const object& r) {
	if (r.t == type_long) {
		value += std::to_string( ( (object_long*)&r )->value );
	}
	else if (r.t == type_double) {
		value += std::to_string( ( (object_double*)&r)->value );
	}
	else if (r.t == type_string) {
		value += ((object_string*)&r)->value;
	}
	return *this;
}

object_string& object_string::operator*=(const object& r) {
	if (r.t == type_long) {
      long long val = ((object_long*)&r)->value;
		if (value.length() > 0 && val > 1) {
			std::string copy = value;
			for (int i = 1; i < val; ++i) {
				value += copy;
			}
		}
	}
	return *this;
}


#pragma endregion
#pragma region comparison_operators

bool operator==(const object_string& l, const object& r) {
	if (r.t == type_string) {
		return l.value == ((object_string*)&r)->value;
	}
	throw;
}
bool operator!=(const object_string& l, const object& r) {
	if (r.t == type_string) {
```

```
                return l.value != ((object_string*)&r)->value;
        }
        throw;
}

#pragma endregion

#pragma endregion
```

B.5 Текст файлу «functions.h»

```
#include"object.h"

void plus(object*& return_value, std::vector<object*> targets);
void minus(object*& return_value, std::vector<object*> targets);
void multiply(object*& return_value, std::vector<object*> targets);
void divide(object*& return_value, std::vector<object*> targets);
void f_and(object*& return_value, std::vector<object*> targets);
void f_or(object*& return_value, std::vector<object*> targets);
void f_reverse(object*& return_value, std::vector<object*> targets);

void print(object*& return_value, std::vector<object*> targets);
void set_value(object*& return_value, std::vector<object*> targets);

void less(object*& return_value, std::vector<object*> targets);
void more(object*& return_value, std::vector<object*> targets);
void less_eq(object*& return_value, std::vector<object*> targets);
void more_eq(object*& return_value, std::vector<object*> targets);
void not_equal(object*& return_value, std::vector<object*> targets);
void equal(object*& return_value, std::vector<object*> targets);
```

B.6 Текст файлу «functions.cpp»

```
#include "functions.h"

void plus(object*& return_value, std::vector<object*> targets) {
        return_value = *targets[0] + *targets[1];
}
void minus(object*& return_value, std::vector<object*> targets) {
        return_value = *targets[0] - *targets[1];
}
void multiply(object*& return_value, std::vector<object*> targets) {
        return_value = *targets[0] * *targets[1];
}
void divide(object*& return_value, std::vector<object*> targets) {
        return_value = *targets[0] / *targets[1];
}
void print(object*& return_value, std::vector<object*> targets) {
        targets[0]->print_object();
}
void set_value(object*& return_value, std::vector<object*> targets) {
        targets[0]->set(targets[1]);
        return_value = targets[0];
}

void less(object*& return_value, std::vector<object*> targets){
   auto obj = new object_bool(*targets[0] < *targets[1]);
   scope::current->hold.insert(std::pair<std::string, object*>{"temporal" + std::to_string((long long)obj), obj});
   return_value = obj;
```

```
}
void more(object*& return_value, std::vector<object*> targets){
    auto obj = new object_bool(*targets[0] > *targets[1]);
    scope::current->hold.insert(std::pair<std::string, object*>{"temporal" + std::to_string((long long)obj), obj});
    return_value = obj;
}
void less_eq(object*& return_value, std::vector<object*> targets){
    auto obj = new object_bool(*targets[0] <= *targets[1]);
    scope::current->hold.insert(std::pair<std::string, object*>{"temporal" + std::to_string((long long)obj), obj});
    return_value = obj;
}
void more_eq(object*& return_value, std::vector<object*> targets){
    auto obj = new object_bool(*targets[0] >= *targets[1]);
    scope::current->hold.insert(std::pair<std::string, object*>{"temporal" + std::to_string((long long)obj), obj});
    return_value = obj;
}
void not_equal(object*& return_value, std::vector<object*> targets){
    auto obj = new object_bool(*targets[0] != *targets[1]);
    scope::current->hold.insert(std::pair<std::string, object*>{"temporal" + std::to_string((long long)obj), obj});
    return_value = obj;
}
void equal(object*& return_value, std::vector<object*> targets){
    auto obj = new object_bool(*targets[0] == *targets[1]);
    scope::current->hold.insert(std::pair<std::string, object*>{"temporal" + std::to_string((long long)obj), obj});
    return_value = obj;
}

void f_and(object*& return_value, std::vector<object*> targets){
    auto obj = new object_bool(*targets[0] && *targets[1]);
    scope::current->hold.insert(std::pair<std::string, object*>{"temporal" + std::to_string((long long)obj), obj});
    return_value = obj;
}
void f_or(object*& return_value, std::vector<object*> targets){
    auto obj = new object_bool(*targets[0] || *targets[1]);
    scope::current->hold.insert(std::pair<std::string, object*>{"temporal" + std::to_string((long long)obj), obj});
    return_value = obj;
}
void f_reverse(object*& return_value, std::vector<object*> targets){
    auto obj = new object_bool(!*targets[0]);
    scope::current->hold.insert(std::pair<std::string, object*>{"temporal" + std::to_string((long long)obj), obj});
    return_value = obj;
}
```

B.7 Текст файлу «main.cpp»

```
#include <iostream>
#include <fstream>
#include "reader.h"


void pow2(object*& return_value, std::vector<object*> targets) {
    return_value = *targets[0] * *targets[0];
}

int main(int argc, char** argv)
{

    if(argc != 2)
        return 1;
    std::string scr = argv[1];
```

```
    function::init_base();

    function::function_list.emplace(
        std::pair<std::string, function*>
            { "pow2",
              new function({ type_any }, {}, pow2)
            });

    function::function_list.emplace(
        std::pair<std::string, function*>
            { "pow3",
              new function({ type_any }, {"return a * a * a;"}, nullptr, {"a"})
            });

    std::ifstream script{scr};
    std::string str((std::istreambuf_iterator<char>(script)), std::istreambuf_iterator<char>());
    std::string::iterator it = str.begin();
    reader::read(it, str);

}
```

Додаток Г.

ГРАФІЧНА ЧАСТИНА
РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ УПРАВЛІННЯ ІГРОВИМИ РУШІЯМИ НА ОСНОВІ СКРИПТОВОЇ МОВИ

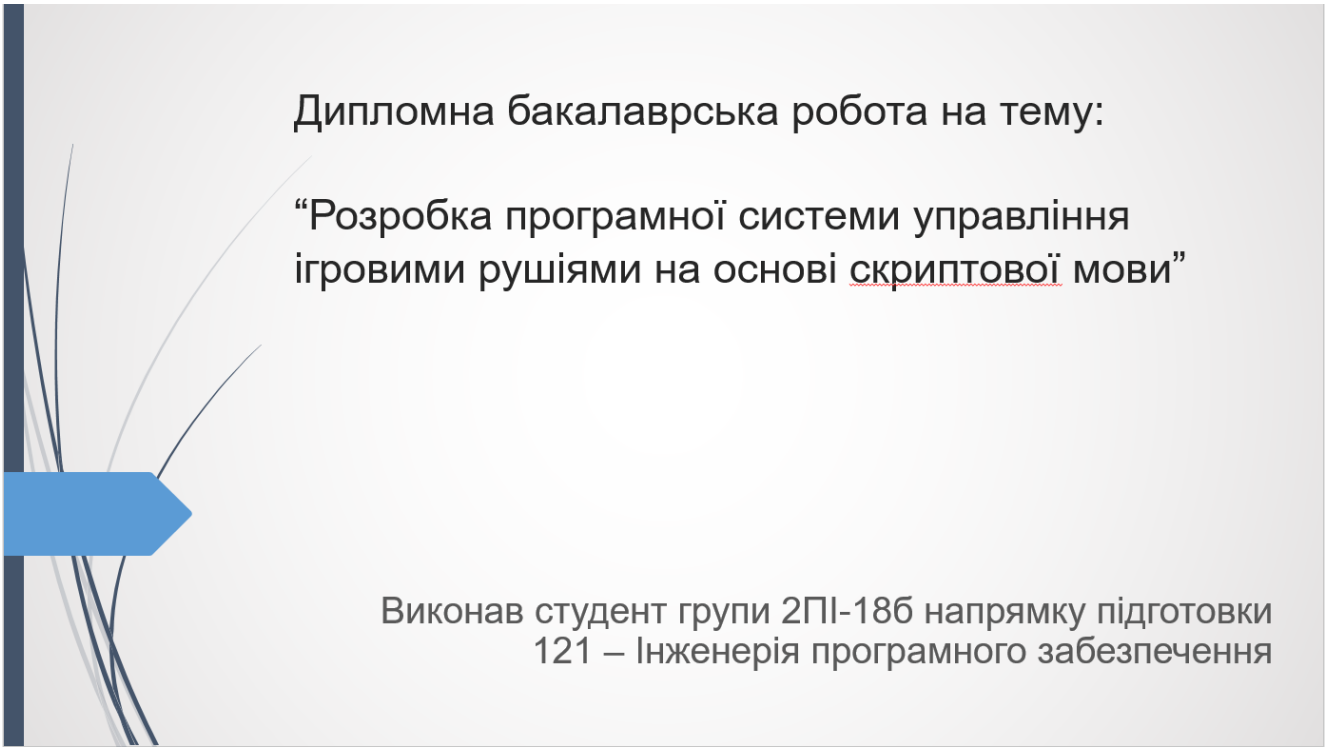

Рисунок Г.1 – Заголовок роботи

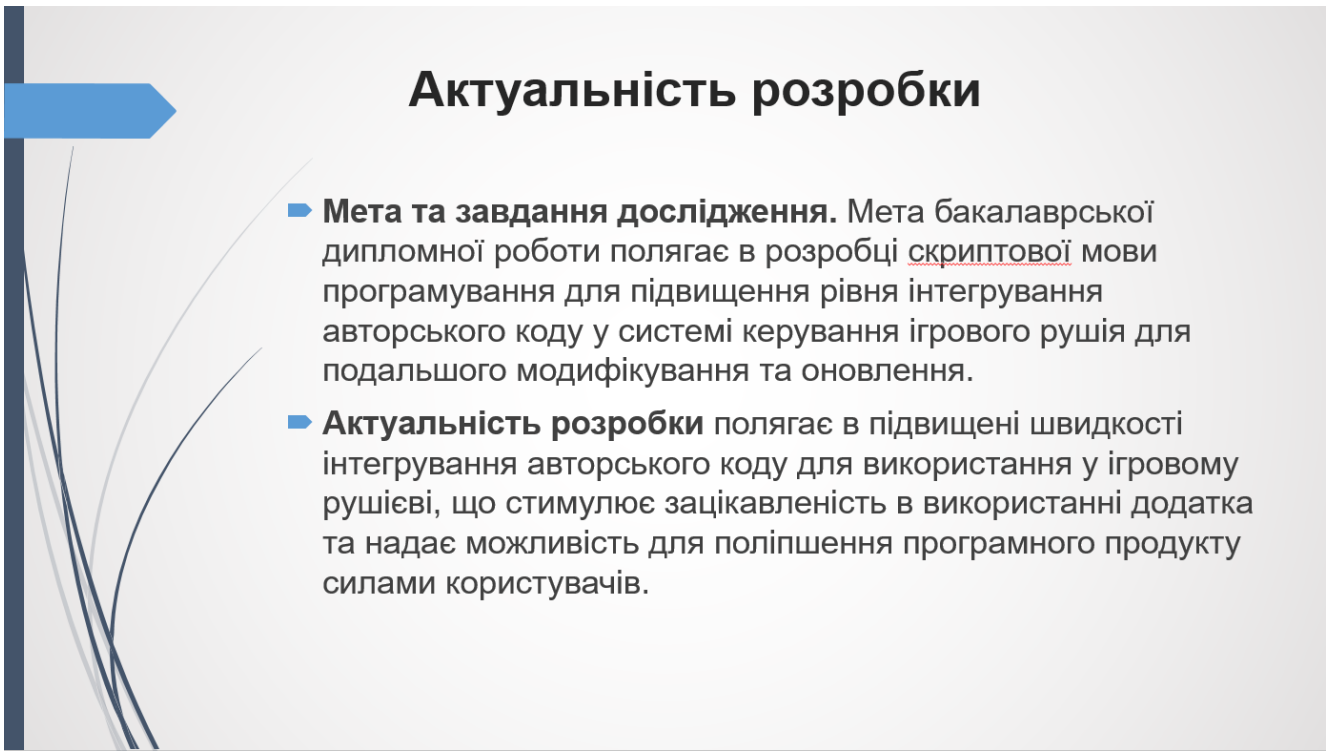

Рисунок Г.2 – Актуальність розробки

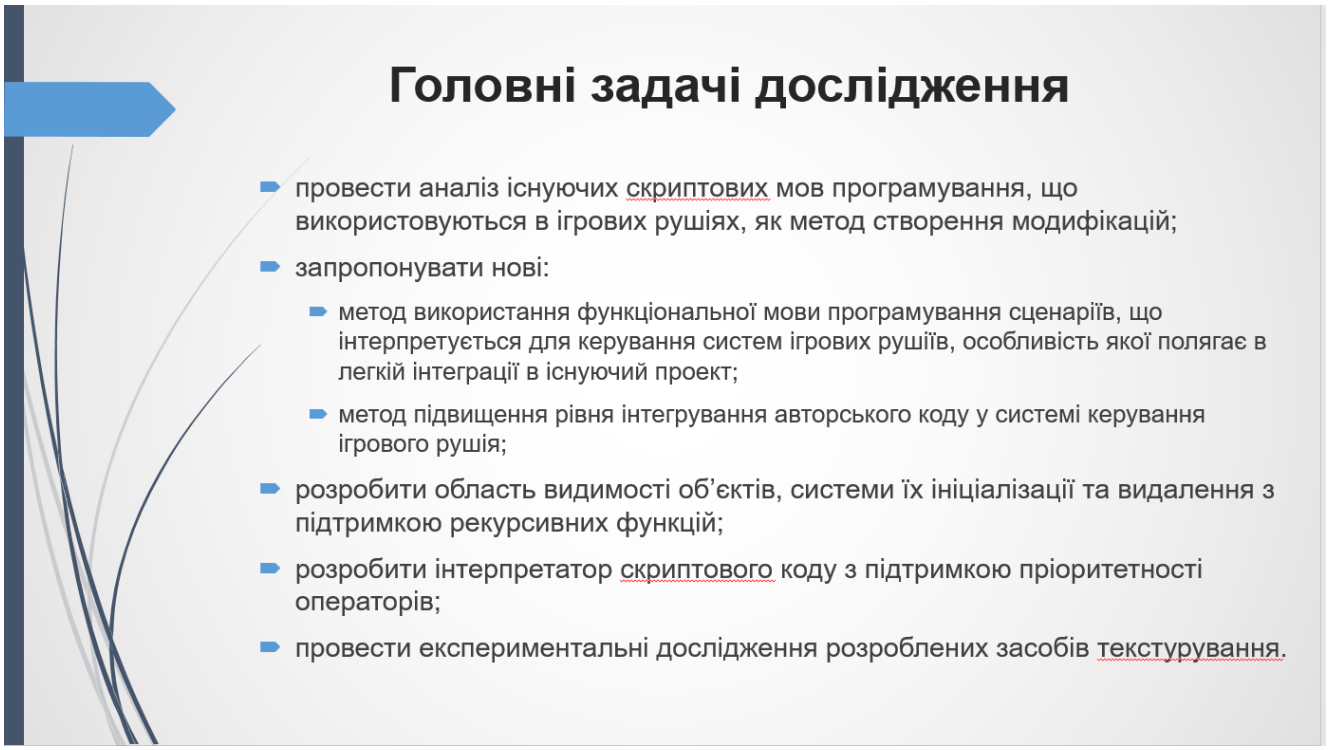

Рисунок Г.3 – Головні задачі дослідження

Рисунок Г.4 – Новизна отриманих результатів

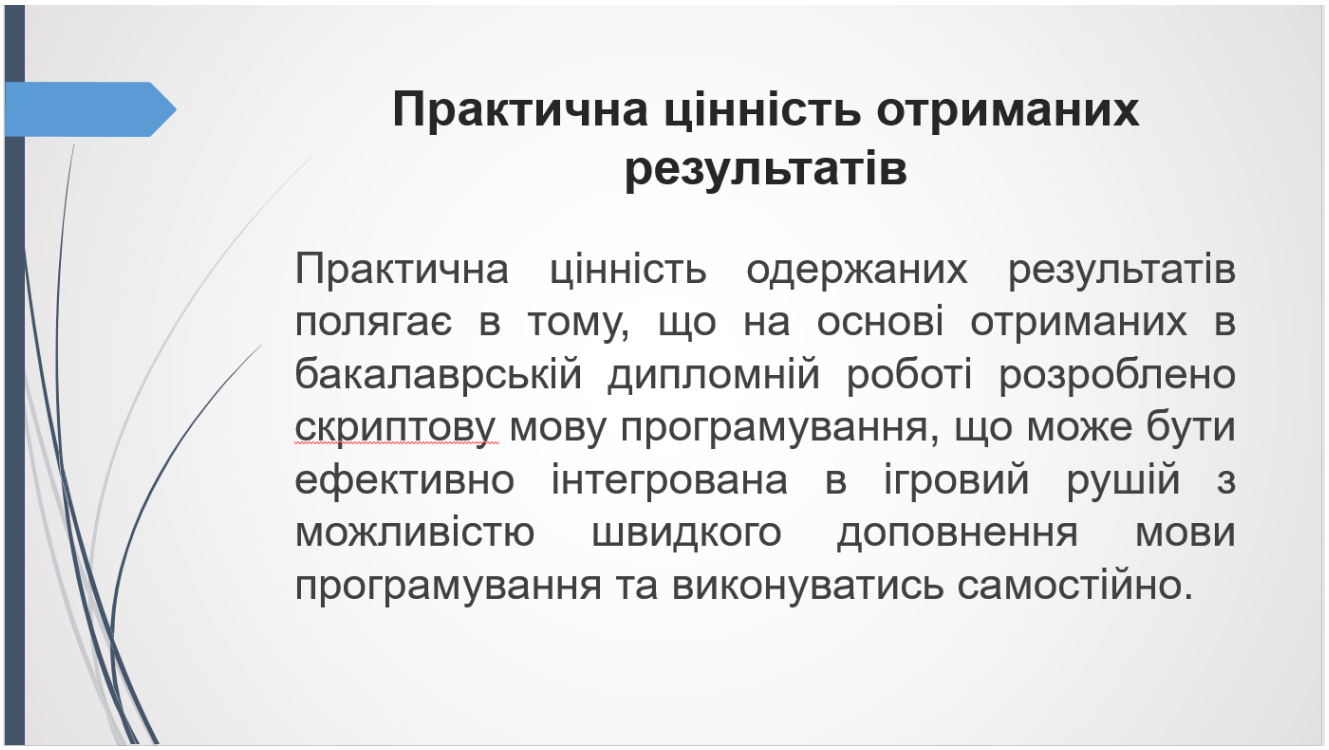

Рисунок Г.5 – Практична цінність отриманих результатів

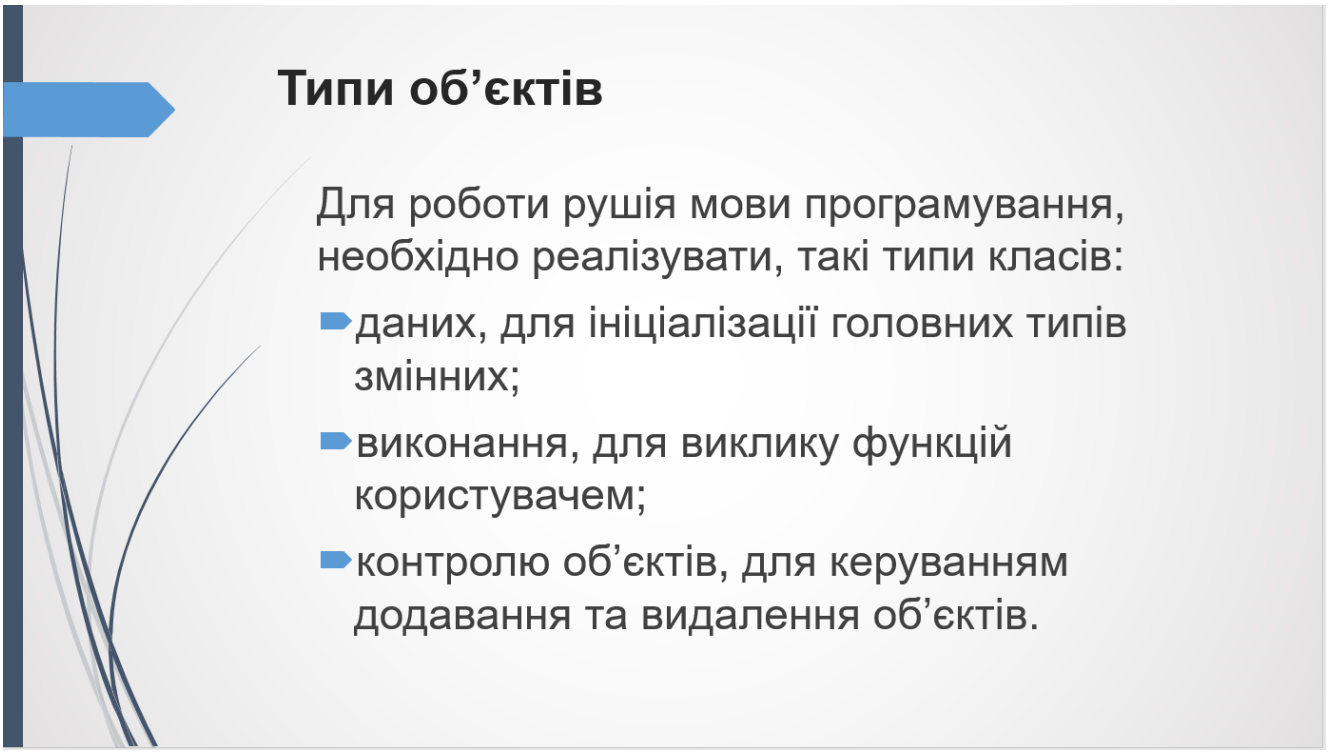

Рисунок Г.6 – Типи об’єктів

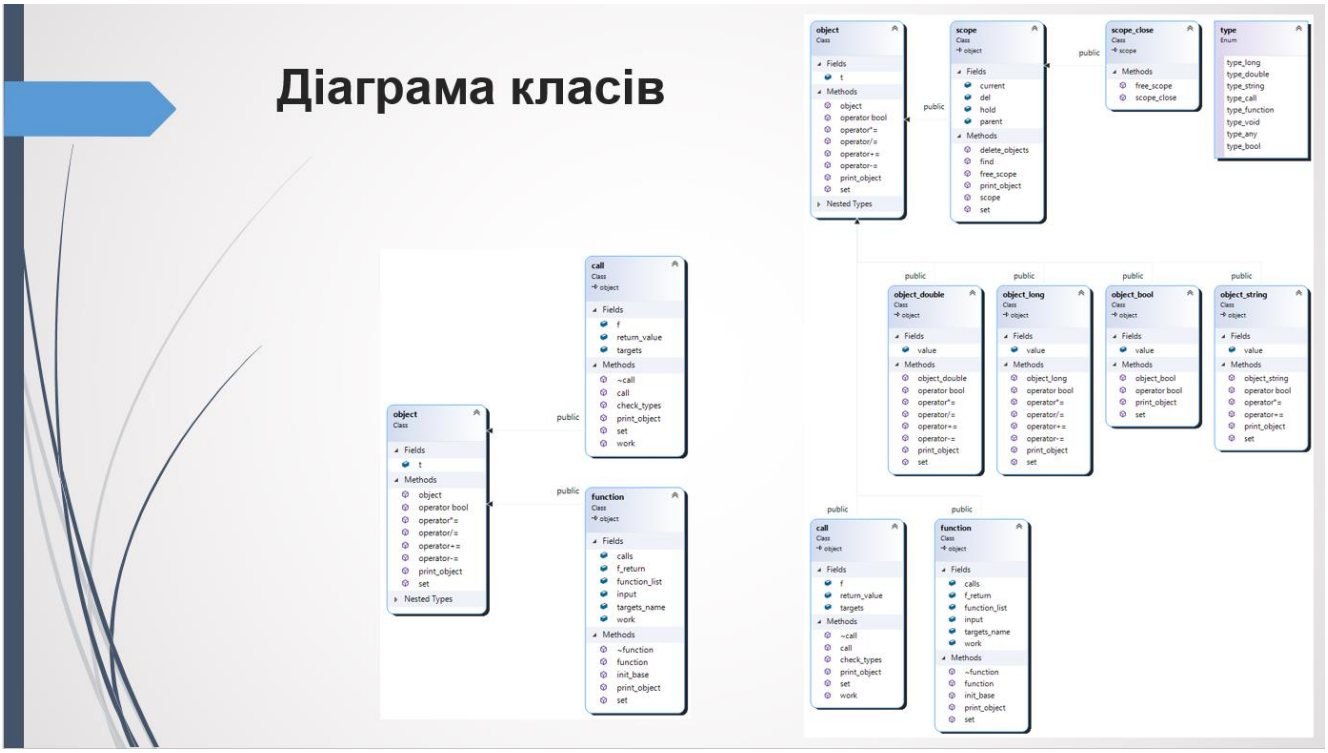

Рисунок Г.7 – Діаграма класів

Рисунок Г.8 – Виклик функцій

Рисунок Г.9 – Обробка ключових слів

Рисунок Г.10 – Ініціалізація даних

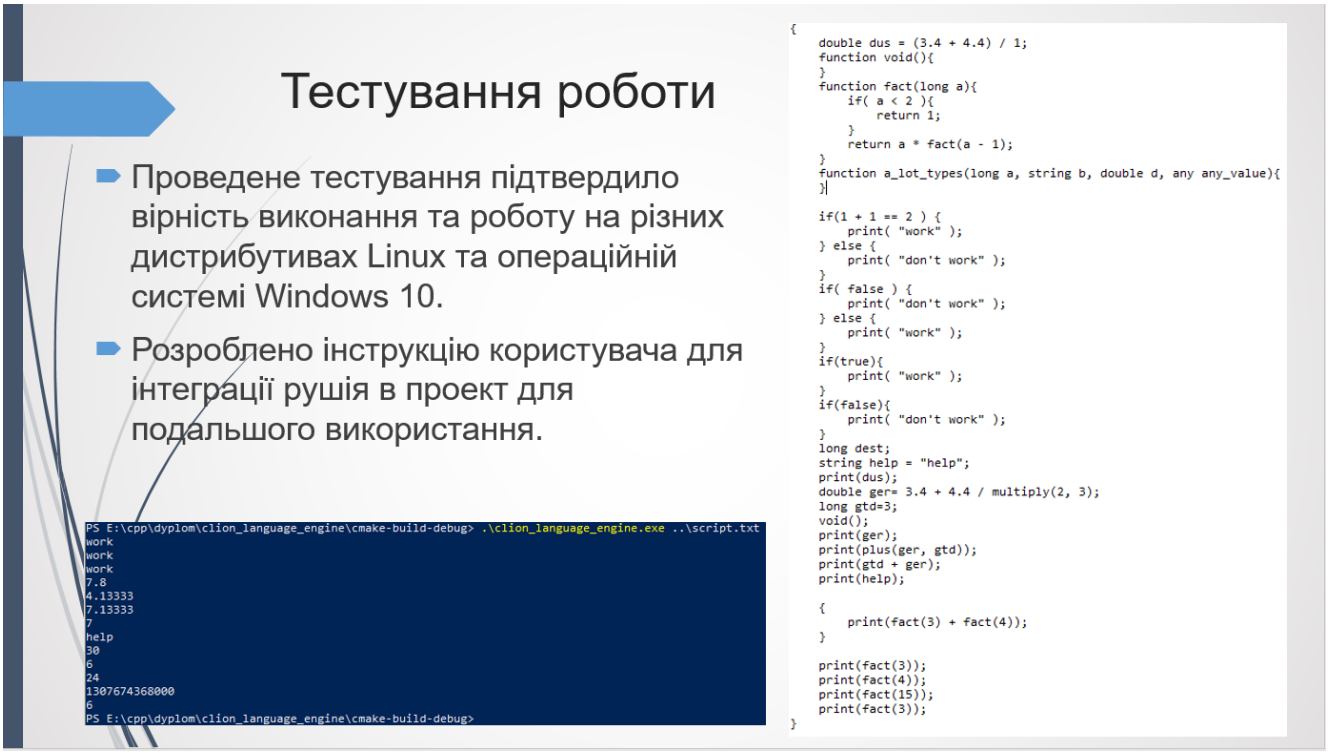

Рисунок Г.11 – Тестування роботи

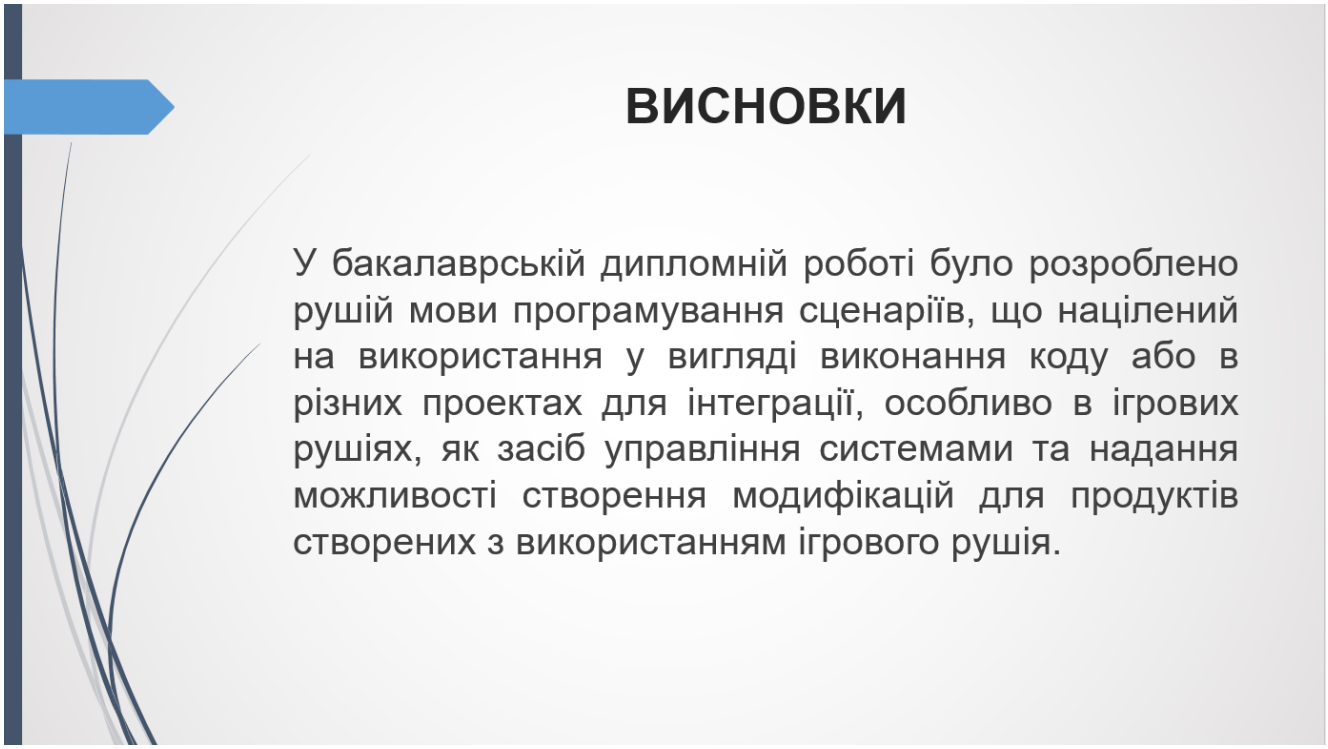

Рисунок Г.12 – Висновки

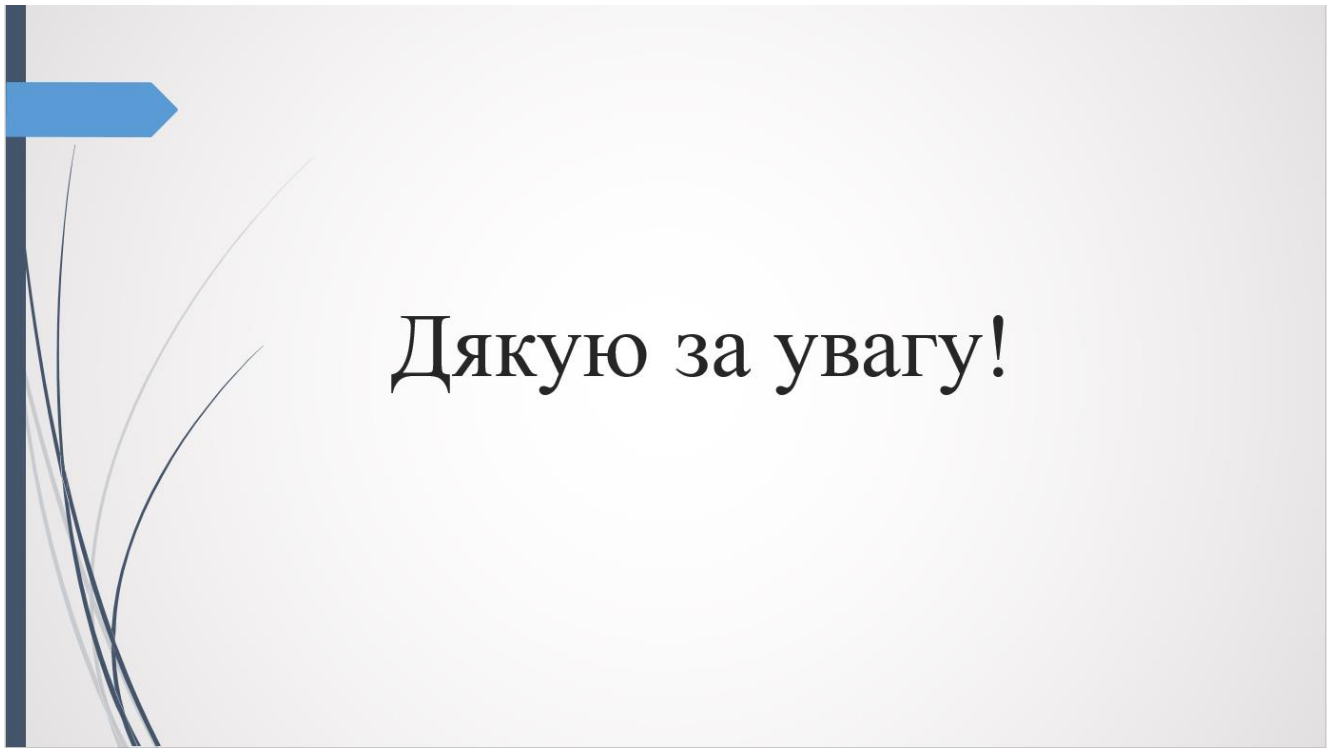

Рисунок Г.13 – Слайд 13