НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра автоматизованих систем обробки інформації та управління

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з                     «Об’єктно орієнтоване програмування»

(назва дисципліни)

на тему:                 «Комп’ютерні шахмати»

Студентів IІ курсу ІП-52 групи

напряму підготовки 6.050103 «Програмна інженерія»

спеціальності «Програмне забезпечення систем»

Ващенка Ю. О.

(прізвище та ініціали)

Ільїної М. Д.

(прізвище та ініціали)

Керівник Муха І.П

Доцент кафедри АСОІУ

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ ‑ 2016 рік

Національний технічний університет України “КПІ”

(назва вищого навчального закладу)

Кафедра автоматизованих систем обробки інформації і управління

Дисципліна Основи програмування

Напрям "Програмна інженерія"

Курс ІІ Група ІП-52 Семестр 3

**ЗАВДАННЯ**

**на курсову роботу студентів**

|  |
| --- |
| Ващенка Юрія Олександровича |

(прізвище, ім’я, по батькові)

|  |
| --- |
| Ільїна Марія Дмитрівна |

(прізвище, ім’я, по батькові)

|  |  |
| --- | --- |
| 1. Тема роботи | Комп’ютерні шахмати |
|  | |
|  | |

|  |  |
| --- | --- |
| 2. Строк здачі студентом закінченої роботи |  |

|  |  |
| --- | --- |
| 3. Вихідні дані до роботи | Технічне завдання |
|  | |
|  | |
|  | |

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

|  |
| --- |
|  |
|  |
|  |
|  |

5. Перелік графічного матеріалу ( з точним зазначенням обов’язкових креслень )

|  |
| --- |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
| 6. Дата видачі завдання |  |

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи | 14.03.16-20.03.16 |  |
| 2. | Підготовка ТЗ | 21.03.16-03.04.16 |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи | 04.04.16-17.04.16 |  |
| 4. | Розробка алгоритму вирішення задачі | 18.04.16-24.04.16 |  |
| 6. | Узгодження алгоритму з керівником | 18.04.16-24.04.16 |  |
| 5. | Розробка сценарію роботи програми | 25.04.16-01.05.16 |  |
| 6. | Узгодження сценарію роботи програми з керівником | 25.04.16-01.05.16 |  |
| 7. | Узгодження з керівником інтерфейсу користувача | 25.04.16-01.05.16 |  |
| 8. | Розробка програмного забезпечення | 02.05.16-22.05.16 |  |
| 9. | Налагодження розрахункової частини програми | 02.05.16-22.05.16 |  |
| 10. | Розробка та налагодження інтерфейсної частини програми | 23.05.16-29.05.16 |  |
| 11. | Узгодження з керівником набору тестів для контрольного прикладу | 23.05.16-29.05.16 |  |
| 12. | Тестування програми | 30.05.16-05.06.16 |  |
| 13. | Підготовка пояснювальної записки | 06.06.16-12.06.16 |  |
| 14. | Здача курсової роботи на перевірку | 06.06.16-12.06.16 |  |
| 15. | Захист курсової роботи |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | |  |  | Ващенко Ю. О. |
|  | | (підпис) |  | (прізвище, ім’я, по батькові) |
| Студент | |  |  | Ільїна М. Д. |
|  | | (підпис) |  | (прізвище, ім’я, по батькові) |
|  |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник |  |  | Муха І. П. |
|  | (підпис) |  | (прізвище, ім’я, по батькові) |

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2016 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 80 сторінок, 22 рисунка,   
8 таблиць, 10 посилань.

Об’єкт дослідження: комп’ютерні шахи.

Мета роботи: створення програмного забезпечення для гри в шахи з можливістю гри в комп’ютерні режимі.

Вивчено методи реалізації гри в шахи, особливості реалізації динамічного інтерфейсу, робота з потоками та класами платформи Microsoft .NET. А також з технологією WCF. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв’язання кожної з них.

Виконана програмна реалізація гри в шахи з комп’ютерні режимом.

КОМП’ЮТЕРНІ ІГРИ, ШАХИ, 3D МОДЕЛЮВАННЯ

Зміст

[Вступ 6](#_Toc468214859)

[1 Постановка задачі 7](#_Toc468214860)

[2 Теоретичні відомості 8](#_Toc468214861)

[3 Опис алгоритмів 12](#_Toc468214862)

[3.1. Алгоритм синтаксичного аналізу математичних виразів: 12](#_Toc468214863)

[3.2. Алгоритм сортувальної станції: 13](#_Toc468214864)

[3.3. Алгоритм зміни графа таблиці: 15](#_Toc468214865)

[3.4. Алгоритм топологічного сортування 15](#_Toc468214866)

[3.5. Рекурсивний алгоритм обходу графу у глибину 16](#_Toc468214867)

[4 Опис програмного забезпечення 17](#_Toc468214868)

[4.1. Функціональна структура програмного забезпечення 17](#_Toc468214869)

[4.2. Опис функцій частин програмного забезпечення 22](#_Toc468214870)

[4.2.1. Користувацькі функції 23](#_Toc468214871)

[4.2.2. Стандартні функції 35](#_Toc468214872)

[5 Тестування програмного забезпечення 40](#_Toc468214873)

[5.1. Теорія 40](#_Toc468214874)

[5.2. План тестування 41](#_Toc468214875)

[5.3. Приклади тестування 41](#_Toc468214876)

[5.3.1. Тестування правильності введених значень 41](#_Toc468214877)

[5.3.1.1. Тестування при введенні некоректних посилань 41](#_Toc468214878)

[5.3.1.2. Тестування при введенні некоректних чисел 43](#_Toc468214879)

[5.3.1.3. Тестування при введенні неіснуючої функції 44](#_Toc468214880)

[5.3.2. Тестування коректності роботи 45](#_Toc468214881)

[5.3.2.1. Перевірка коректності збереження файлу 45](#_Toc468214882)

[5.3.2.2. Перевірка коректності відкриття файлу 46](#_Toc468214883)

[5.3.2.3. Перевірка коректності роботи при створенні таблиці малих розмірів 46](#_Toc468214884)

[5.3.2.4. Перевірка коректності роботи при створенні таблиці великих розмірів 47](#_Toc468214885)

[5.3.2.5. Перевірка поведінки програми при закриті файлу, що містить не збережені зміни 48](#_Toc468214886)

[5.3.3. Перевірка вірності роботи програми, коли користувачем вірно введені всі дані та для обраного методу рішення сходиться 48](#_Toc468214887)

[6 Інструкція користувача 51](#_Toc468214888)

[6.1. Робота з програмою 51](#_Toc468214889)

[6.2. Формат вхідних та вихідних даних 56](#_Toc468214890)

[6.3. Системні вимоги 56](#_Toc468214891)

[Висновки 57](#_Toc468214892)

[Перелік посилань 58](#_Toc468214893)

[Додаток А Тексти програмного коду 60](#_Toc468214894)

Вступ

Електронна таблиця являє собою інтерактивну комп'ютерну програму для організації, аналізу і зберігання даних в табличній формі. Електронні таблиці   
(англ. Spreadsheet) [1][2][3] розроблені як комп'ютеризованих моделювання бухгалтерського обліку паперу робочих листів. Програма працює на даних, введених в комірки таблиці [4]. Кожна клітинка може містити або числові або текстові дані або результати формул, які автоматично обчислювати і відображати значення, засноване на утриманні інших клітин. Електронна таблиця може також стосуватися одного такого електронного документа. Користувачі електронних таблиць може налаштувати будь-яку збережене значення і спостерігати ефекти на розрахункових значень [5] [6] [7]. Це робить таблицю корисною для "що-якщо" аналіз, так як багатьох випадках може бути швидко досліджена без ручного перерахунку.

Дана курсова робота призначена для використання в області бухгалтерського обліку та в усіх розділах аналізу інформації. Вона значно спростить процес створення власних таблиць, спростить користувачам процес громіздких обчислень та надають можливість представлення інформації у зручному для подальшого обробки та аналізу вигляді.

Для вирішення даної задачі необхідно розробити зручний для користувача інтерфейс, механізми обробки введених користувачем формул з можливістю посилатися на інші комірки таблиці. Передбачити динамічну змінну вмісту комірок якщо користувач змінив дані від яких залежать вміст цих комірок. Необхідно добавити можливість збереження та зчитування таблиці з файла.

Для реалізації обробки формул ми будем використовувати алгоритм сортувальної станції, який розроблений нідерландським науковцем у галузі комп'ютерних наук – Едгерсом Дейкстрою, який переводить вираз заданий в інфіксній нотації в обернену польську нотацію. В такому вигляді вираз можна обчислити за допомогою відомого алгоритма з використанням стеку.

Для реалізації механізму перерахунку формул ми будемо використовувати підхід перерахунку по графу, що попередньо топологічно відсортований.

# Постановка задачі

Розробити програмне забезпечення, що буде реалізовувати комп’ютерні шахи.

Програма повинна забезпечувати можливість виконання нижче наведених функцій:

1. Відображення шахової дошки в форматі 3D-rendered;
2. Виконання алгоритму гри в шахи для бота(автоматичної програми)
3. Реалізація підключення до віддаленого гравця напряму.
4. Реалізація підключення до віддаленого гравця через сайт(якщо доступний).
5. Робота з віддаленим сервером, зокрема зі списком гравців, та їх рейтингами.
6. Виявлення циклічних посилань у комірках та блокування можливості їх обчислення
7. Збереження та відкриття таблиці з файлу (\*.adc)

Вхідні дані до програми повинні бути організовані у вигляді файлів, відповідної специфікації та з розширенням (\*.adc).

Вихідними даними для програми організовані у вигляді файлів, відповідної специфікації та з розширенням (\*.adc).

# Теоретичні відомості

Для вирішення поставленої задачі необхідно розробити механізми обробки введених користувачем формул з можливістю посилатися на інші комірки таблиці. Передбачити динамічну змінну вмісту комірок якщо користувач змінив дані від яких залежать вміст цих комірок.

Для реалізації обробки формул ми будем використовувати алгоритм сортувальної станції. Алгоритм сортувальної станції — метод аналізу математичних виразів, які представлені в інфіксной нотації. Він може бути використаний для отримання математичного виразу у вигляді зворотної польської нотації або у вигляді абстрактного синтаксичного дерева. Алгоритм винайдений Едсгерсом Дейкстрою і названий ним «алгоритм сортувальної станції», оскільки нагадує дію залізничної сортувальної станції.

Так само, як і обчислення значень виразів в зворотної польської записи, алгоритм працює за допомогою стека. Інфіксний запис математичних виразів найчастіше використовується людьми, її приклади: 2 + 4 і 3 + 6 \* (3-2). Для перетворення в зворотний польський нотацію використовується 2 рядки: вхідний і вихідний, і стек для зберігання операторів, ще не доданих в вихідну чергу. При перетворенні алгоритм зчитує 1 символ і виробляє дії, що залежать від даного символу.

Кожен токен-число, функція або оператор виводиться тільки один раз, а також кожен токен-функція, оператор або кругла дужка буде додано і видалений з стека по одному разу. Постійна кількість операцій на токен дає в результаті лінійну складність алгоритму сортувальної станції O (n) [8].

На рисунку 2.1 зображено реалізацію алгоритму , що використовує тригілкове залізничний вузол. Початкові дані опрацьовуються по одному символу за раз, якщо отримана змінна або номер вона копіюється прямо на вихід b), d), f), h). Якщо ж це оператор, він заштовхнується в стек операторів c), e), однак, якщо старшинство менше ніж у оператора на верхівці стека або вони мають однакове старшинство й оператор ліво асоціативний, тоді той оператор виштовхується зі стека й записується на вихід g). Насамкінець оператор, що залишились у стеці, виштовхуються і додаються до виходу.

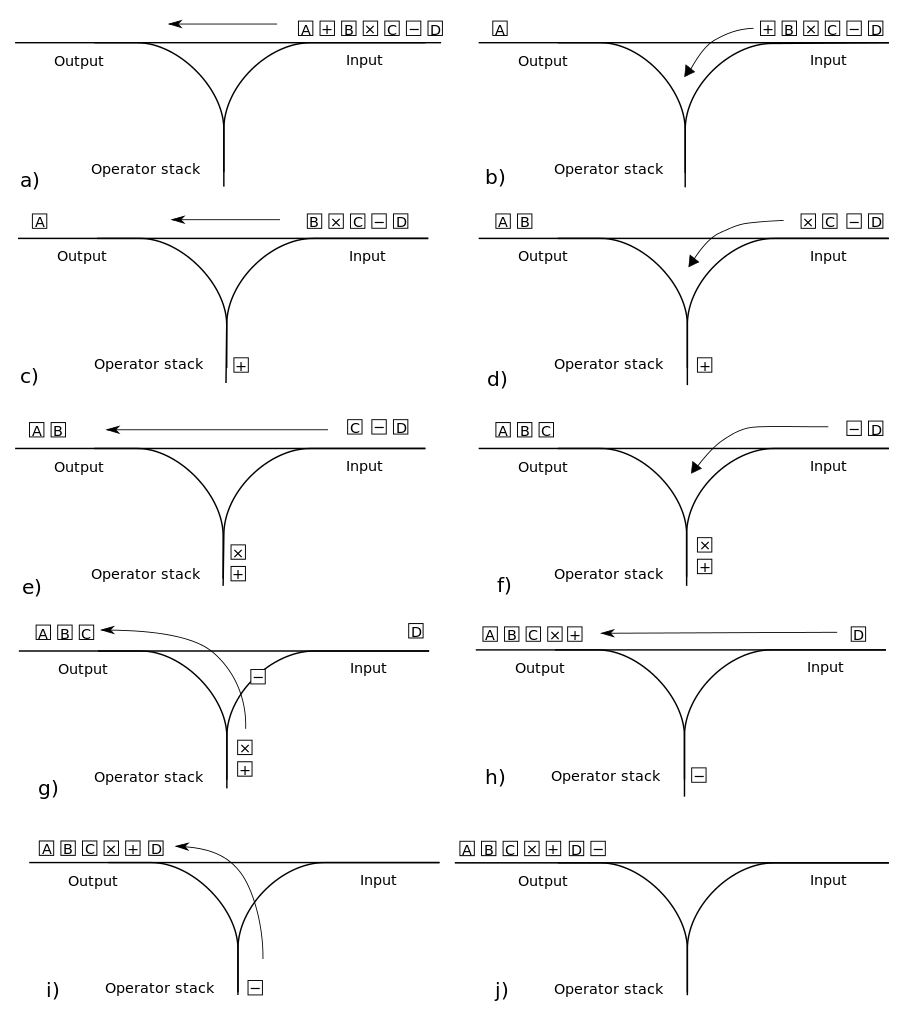


Рисунок 2.1 – Приклад роботи алгоритму сортувальної станції

Для реалізації механізму перерахунку формул ми будемо використовувати підхід перерахунку по графу, що попередньо топологічно відсортований.

Для реалізації алгоритму топологічного сортування необхідно представити таблицю у вигляді графу. Для цього представимо кожну комірку таблиці, що містить будь-яку інформацію як вершину графу та проведем ребра від даної вершини до тих вершин в яких вона використовується для обчислення формули. Приклад такого графу наведено на Рисунку 2.2.

C:\Users\Саша\Downloads\Untitled Diagram.png

Рисунок 2.2 – Приклад графу електронної таблиці з трьома комірками

Коли такий граф створений можна його топологічно відсортувати. Топологічне сортування — впорядковування вершин безконтурного орієнтованого графа згідно з частковим порядком, визначеним ребрами цього графу на множині його вершин.

Наприклад, для графу (рисунок 2.3)

,

існує декілька узгоджених послідовностей його вершин, які можуть бути отримані за допомогою топологічного сортування, наприклад:

* 7, 5, 11, 3, 8, 2, 9, 10
* 3, 7, 5, 8, 11, 10, 9, 2
* 7, 5, 11, 2, 3, 8, 9, 10

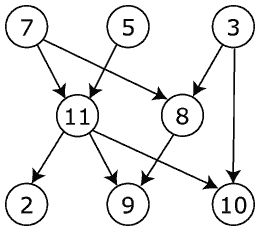


Рисунок 2.3 – Приклад граф, який можливо топологічно впорядкувати.

Час виконання для звичайного алгоритму топологічного сортування лінійний до кількості вершин плюс кількість ребер O(|V|+|E|) [9].

# Опис алгоритмів

## Алгоритм синтаксичного аналізу математичних виразів:

1. ПОЧАТОК
2. Перевести вхідну строку з інфіксною в постфіксну нотацію за допомогою алгоритму сортувальної станції (підрозділ 3.2).
3. ПОКИ не закінчилася вхідна строка:
   1. ЯКЩО поточний символ цифра АБО поточний символ знак мінус та наступний символ цифра, ТО починаючи з поточної позиції перевести символи в число та помістити його в стек
   2. ЯКЩО поточний символ «$» АБО поточний символ знак мінус та наступний символ «$», ТО:
      1. Знайти координати на які вказує посилання.
      2. ЯКЩО комірка з заданими координатами існує ТА даній комірці зберігається число, ТО помістити дане число в стек, ІНАКШЕ видати повідомлення про помилку.
   3. ЯКЩО поточний символ є елементарною операцією, ТО:
      1. ЯКЩО стек містить менше 2-х значень, ТО видати повідомлення про помилку.
      2. Витягнути зі стеку два числа та виконати над ними задану операцію. Результат операції помістити в стек.
   4. ЯКЩО поточний символ є літерою, ТО:
      1. ЯКЩО стек пустий, то видати повідомлення про помилку.
      2. ЯКЩО існує задана в формулі функція ТА верхній елемент стеку належить області допустимих значень даної функції, ТО виконати дану функці з цим значенням та результат помістити в стек. ІНАКШЕ видати повідомлення про помилку.
4. КІНЕЦЬ

## Алгоритм сортувальної станції:

1. ПОЧАТОК
2. ПОКИ не кінець вхідної строки:
   1. ПОКИ поточний символ є пробілом, ТО перейти до наступного символа.
   2. ЯКЩО досягнути кінець вхідної строки, ТО перейти до пункту 3.
   3. ЯКЩО поточний символ цифра, ТО:
      1. ПОКИ поточний символ є цифрою, ТО вставити поточний символ в вихідну строку та перейти до наступного символу.
      2. ЯКЩО поточний символ є точкою, ТО помістити її вихідну строку та перейти до пункту 2.3.3. ІНАКШЕ перейти до пункту 2.3.4.
      3. ПОКИ поточний символ є цифрою, ТО вставити поточний символ в вихідну строку та перейти до наступного символу.
      4. ЯКЩО поточний символ не пробілом АБО кінцем строки, ТО видати повідомлення про помилку.
      5. Вставити пробільний символ в вихідну строку.
      6. Перейти до пункту 2.
   4. ЯКЩО поточний символ відкриваюча дужка, ТО помістити її в стек та перейти до пункту 2.
   5. ЯКЩО поточний символ ТО:
      1. ПОКИ стек не пустий ТА поточний елемент стеку не відкриваюча дужка, ТО помістити верхній елемент стеку в вихідну строку та видалити даний елемент зі стеку.
      2. Вставити пробільний символ в вихідну строку.
      3. Перейти до пункту 2.
   6. ЯКЩО поточний символ є символом елементарної операції, ТО:
      1. ЯКЩО поточний символ є знаком мінус ТА вихідна строка пуста АБО верхній елемент стеку є відкриваючою дужкою, ТО помістити знак мінус як унарний операнд в вихідну строку ТА перейти до пункту 2.
      2. ПОКИ стек не пустий ТА на вершині стеку знаходиться операція з пріоритетом не нижчим, ніж дана операції, ТО помістити операцію зі стеку в вихідну строку та видалити її зі стеку.
      3. Помістити поточну операцію у стек.
      4. Вставити пробільний символ в вихідну строку.
      5. Перейти до пункту 2.
   7. ЯКЩО поточний символ «$», ТО:
      1. ПОКИ поточний символ літера, ТО помістити поточну літеру в вихідну строку та перейди до наступного символу.
      2. ЯКЩО поточний символ не дорівнює «$», ТО видати повідомлення про помилку.
      3. ПОКИ поточний символ цифра, ТО помістити цю цифру в вихідну строку та перейти до наступного символу.
      4. ЯКЩО поточний символом не є пробілом АБО кінцем строки, ТО видати повідомлення про помилку.
      5. Помістити пробіл в вихідну строку.
      6. Перейти до пункту 2.
   8. ЯКЩО поточний символ є літерою, ТО:
      1. ПОКИ поточний символ є літерою, ТО помістити її вихідну строку та перейти до наступного символу.
      2. ПОКИ поточний символ є пробілом, ТО перейти до наступного символу.
      3. ЯКЩО досягнуто кінець строки АБО поточний символ не є відкриваючою дужкою, ТО видати повідомлення про помилку.
      4. Добавити назву функції в стек.
      5. Добавити відкриваючу дужку в стек.
      6. Добавити пробільний символ в вихідну строку.
      7. Перейти до пункту 2.
   9. Вивести повідомлення про помилку.
3. ПОКИ стек не пустий, ТО помісти операцію з вершини стеку в вихідну строку та видалити її зі стеку.
4. КІНЕЦЬ

## Алгоритм зміни графа таблиці:

1. ПОЧАТОК
2. ЯКЩО змінена комірка містила вірну формулу, ТО:
   1. Знайти список всіх комірок на які посилається стара формула зміненої комірки.
   2. Видалити з графу відповідні ребра між зміненою коміркою та всіма комірками, що знайдені у пункті 2.1.
3. ЯКЩО змінена комірка у комірку введено вірно формулу, ТО:
   1. Знайти список всіх комірок на які посилається нова формула зміненої комірки.
   2. Добавити у граф відповідні ребра між зміненою коміркою та всіма комірками, що знайдені у пункті 3.1.
4. Виконати топологічне сортування графу, починаючи зі зміненої комірки (підрозділ 3.4) та помістити результат сортування у список.
5. Видалити змінену комірку зі списку знайденому у пункті 4.
6. ПОКИ список, що знайдений у пункті 5, містить елементи, ТО:
   1. Обчислити значення комірки, що містить у хвості списку.
   2. Видалити комірку зі хвосту списку.
7. КІНЕЦЬ

## Алгоритм топологічного сортування

1. ПОЧАТОК
2. Стоворити масив відвіданих вершин.
3. Виконати обхід в глибину графа (підрозділ 3.5) починаючи з комірки, що була змінена.
4. КІНЕЦЬ

## Рекурсивний алгоритм обходу графу у глибину

1. ПОЧАТОК
2. Помітити поточну вершину, як ту, що знаходиться в процесі обробки.
3. ЦИКЛ по всім ребрам, що виходять з поточної вершини:
   1. ЯКЩО вершина в яку веде поточне ребро ще не відвідана, ТО рекурсивний виклик алгоритму обходу графу у глибину (підрозділ 3.5).
   2. ЯКЩО поточно вершина знаходить в процесі обробки, ТО помітити поточну вершину як з’єднувачем графа.
4. Помітити поточну вершину як оброблену.
5. Помістити поточну вершину у чергу, що є топологічно відсортованою.
6. КІНЕЦЬ

# Опис програмного забезпечення

## Функціональна структура програмного забезпечення

На рисунку 4.1 Ви можете бачити загальну структуру додатку, де наявні усі класи програми та стрілками позначено наслідування класів (стрілка напрямлена до Базового класу).

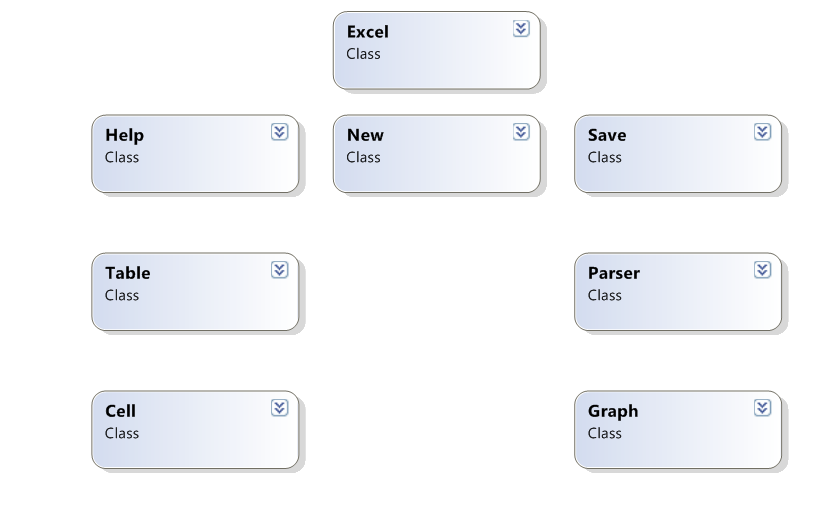


Рисунок 4.1 – Загальна структура класів проекту

На рисунку 4.2 зображено загальну структуру основного класу додатку – Excel, який безпосередньо пов'язує всі компоненти програми між собою та надає інтерфейс користувачеві.

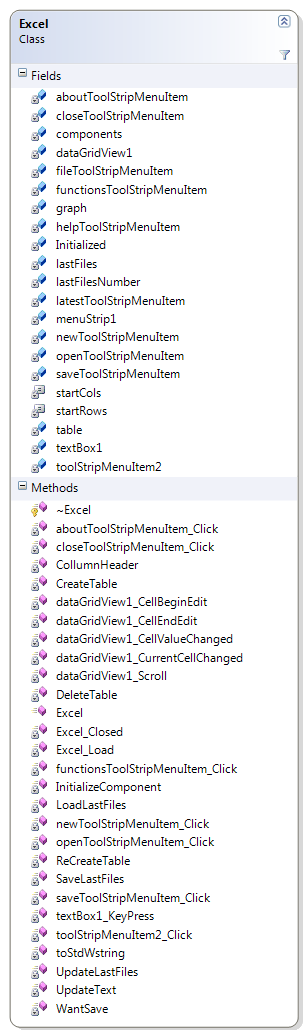


Рисунок 4.2 – Загальна структура класу Excel

На рисунку 4.3 зображено загальну структуру форми збереження таблиці у файл – Save.

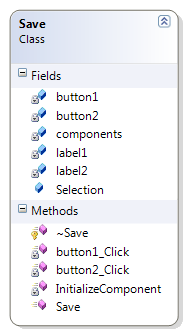


Рисунок 4.3 – Загальна структура класу Save

На рисунку 4.4 зображено загальну структуру форми створення нової   
таблиці – New.

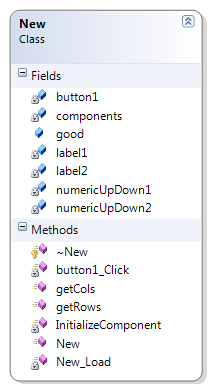


Рисунок 4.4 – Загальна структура класу New

На рисунку 4.5 зображено загальну структуру форми спраки – Hepl.

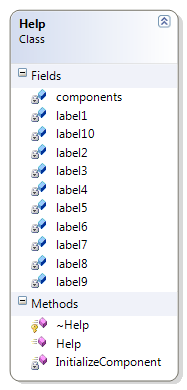


Рисунок 4.5 – Загальна структура класу Help

На рисунку 4.6 зображено загальну структуру класа синтаксичного аналізатора – Parser.

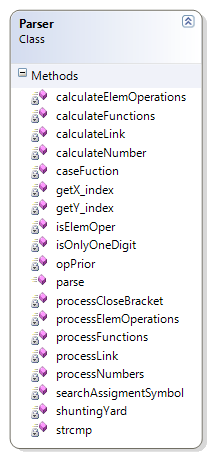


Рисунок 4.6 – Загальна структура класу Parser

На рисунку 4.7 зображено загальну структуру класа графу – Graph.

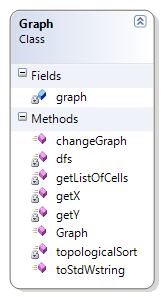


Рисунок 4.7 – Загальна структура класу Graph

На рисунку 4.8 зображено загальну структуру класа представлення комірки таблиці – Cell.

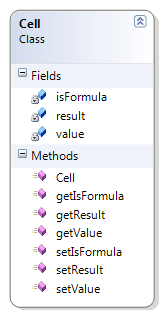


Рисунок 4.8 – Загальна структура класу Cell

На рисунку 4.9 зображено загальну структуру класа представлення   
таблиці – Table.

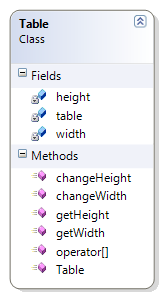


Рисунок 4.9 – Загальна структура класу Table

## Опис функцій частин програмного забезпечення

В ході виконання поставленого завдання було створено наступні модулі та бібліотеки:

а) Cell.h – реалізує комірку таблиці;

б) Graph.h – реалізує побудову та проведення дій з графом таблиці;

в) Number.h – реалізує тип число, що є основним контейнером результатів;

г) Parser.h – реалізує механізм синтаксичного розбору математичних виразів;

д) Table.h – реалізує таблицю;

е) resource.h – містить унікальні ідентифікатори ресурсів проекту;

є) Excel.h - реалізує головне вікно програми;

ж) New.h - реалізує вікно створення нової таблиці;

з) About.h - реалізує вікно About;

и) Help.h - реалізує вікно зі справою;

і) Save.h - реалізує вікно запитання про збереження незбереженого файлу;

### Користувацькі функції

Користувацькі функції, які використані в даній курсовій роботі, описані в таблиці 4.1.

Таблиця 4.1 – Опис користувацьких функцій

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 1 | Cell | Cell | Конструктор класу | - | - |
| 2 | Cell | setValue | Присвоєння нового значення поля value | Нове значення поля value | - |
| 3 | Cell | getValue | Отримання поточного значення поля value | - | Поточне значення поля value |
| 4 | Cell | getResult | Отримання поточного значення поля result | - | Поточне значення поля result |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 5 | Cell | setResult | Присвоєння нового значення поля result | Нове значення поля result | - |
| 6 | Cell | getIsFormula | Отримання поточного значення поля isFormula | - | Поточне значення поля isFormula |
| 7 | Cell | setIsFormula | Присвоєння нового значення поля isFormula | Нове значення поля isFormula | - |
| 10 | Graph | topologicalSort | Виконує топологічне сортування графу, починаючи з поточної комірки | Координати поточної комірки, посилання на таблицю, посилання на масив результату сортування та посилання на масив  циклів | - |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 11 | Graph | dfs | Виконує обхід графу у глибину | Номер поточної вершини, посилання на масив відвіданих вершин, посилання на масив результату обходу та посилання на масив циклів | - |
| 12 | Graph | getY | Визначає Y-координату комірки у таблиці | Строка, що містить лінк (посилання), індекс з якої починати розбір Y-координату, ширина таблиці | Число, Y-координата лінку |
| 13 | Graph | getX | Визначає Y-координату комірки у таблиці | Строка, що містить лінк (посилання), індекс з якої починати розбір X-координату, висота таблиці | Число, X-координата лінку |
|  |  |  |  |  |  |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 14 | Graph | changeGraph | Зміна графа при зміні вмісту комірки таблиці | Посилання на таблицю, посилання на поточне представлення таблиці, координати зміненої комірки та чи містила раніше поточна комірка формулу | - |
| 15 | Graph | toStdWstring | Перетворює строку С# в строку С++ | Строка С# | Строка С++ |
| 16 | Parser | Parser | Розрахунок значення заданої формули | Формула, яку необхідно підрахувати та посилання на таблицю | Значення формули |
| 17 | Parser | shuntingYard | Перетворення інфіксного формату запису в постфіксний | Формула в інфіксному форматі | Формула в постфіксному форматі |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 18 | Parser | opPrior | Визначення пріоритету елементарних операцій | Символ операції | Пріоритет операції |
| 20 | Parser | getX\_index | Повертає Х-координату заданого лінку (посилання) | Строка, що містить лінк, індекс звідки починається розбір, висота таблиці | Число, X-координата лінку |
| 21 | Parser | getY\_index | Повертає Y-координату заданого лінку (посилання) | Строка, що містить лінк, індекс звідки починається розбір, ширина таблиці | Число, Y-координата лінку |
| 22 | Parser | strcmp | Порівняння назви поточної функції з назвою функції з бібліотеки доступних | Строка, що містить назву функції та індекс її першого символу назви, індекс назви функції з бібліотеки | True, якщо назви співпадають, інашкше False |
| 23 | Parser | processNumbers | Обробка чисел | Вхідна строка, вихідна строка, індекс початку розбору в вхідній строці | - |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 24 | Parser | processCloseBracket | Обробка закриваючих дужок | Вхідна строка, вихідна строка, індекс початку розбору в вхідній строці, стек операцій | - |
| 25 | Parser | processElemOperations | Обробка елементарних операцій | Вхідна строка, вихідна строка, індекс початку розбору в вхідній строці, стек операцій | - |
| 26 | Parser | processLink | Обробка посилань | Вхідна строка, вихідна строка, індекс початку розбору в вхідній строці | - |
| 27 | Parser | processFunctions | Обробка функцій | Вхідна строка, вихідна строка, індекс початку розбору в вхідній строці, стек операцій | - |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 28 | Parser | calculateNumber | Обчислення значення числа | Вхідна строка, стек чисел, індекс значення в вхідній строці | - |
| 29 | Parser | calculateLink | Обчислення значення посилання | Посилання на таблицю, вхідна строка, стек чисел, індекс значення в вхідній строці | - |
| 30 | Parser | calculateElemOperations | Обчислення елементарних операцій | Вхідна строка, стек чисел, індекс значення в вхідній строці | - |
| 31 | Parser | calculateFunctions | Обчислення функції | Вхідна строка, стек чисел, індекс значення в вхідній строці | - |
| 32 | Parser | searchAssigmentSymbol | Пошук символа присвоєння | Вхідна строка | Індекс символу присвоєння |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 33 | Parser | caseFuction | Обчислення значення функцій | Індекс функції у бібліотеці та її аргумент | Значення функції |
| 34 | Parser | isOnlyOneDigit | Обробка строки, що складається з одного числа | Вхідна строка та індекс з якого починається пошук | Значення єдиного числа |
| 35 | Table | Table | Конструктор класу | Розміри таблиці | - |
| 36 | Table | getHeight | Отримання поточного значення поля height | - | Поточне значення поля height |
| 37 | Table | getWidth | Отримання поточного значення поля width | - | Поточне значення поля width |
| 38 | Table | changeHeight | Зміна значення поля height на певна значення | Значення на яке треба змінити поле height | - |
| 39 | Table | changeWidth | Зміна значення поля width на певна значення | Значення на яке треба змінити поле width | - |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 40 | Excel | WantSave | Виводить повідомлення про те що файл не збережено | - | - |
| 41 | Excel | SaveLastFiles | Зберігає список останніх файлів | - | - |
| 42 | Excel | LoadLastFiles | Завантажує список останніх файлів | - | - |
| 43 | Excel | UpdateLastFiles | Додає файл до списку останніх використаних | Шлях до файлу | - |
| 44 | Excel | CollumnHeader | Підраховує заголовок i-го стовпця | Номер стовпця | Строка - заголовок |
| 45 | Excel | toStdWstring | Перетворює .NET рядок у с++ рядок | Рядок | Рядок |
| 46 | Excel | DeleteTable | Очищує таблицю | - | - |
| 47 | Excel | CreateTable | Створює таблицю | Розмір нової таблиці | - |
| 48 | Excel | ReCreateTable | Заново створює таблицю | Розмір нової таблиці | - |
| 49 | Excel | UpdateText | Поновлює текст комірки таблиці | Рядок з новим текстом, координати комірки | - |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 50 | Excel | Excel\_Closed | Обробляє закриття вікна | Об’єкт що надсилає подію, аргументи події | - |
| 51 | Excel | Excel\_Load | Обробляє відкриття вікна | Об’єкт що надсилає подію, аргументи події | - |
| 52 | Excel | dataGridView1\_CurrentCellChanged | Обробляє зміну поточної комірки | Об’єкт що надсилає подію, аргументи події | - |
| 53 | Excel | dataGridView1\_CellValueChanged | Обробляє зміну данних у комірці | Об’єкт що надсилає подію, аргументи події | - |
| 54 | Excel | textBox1\_KeyPress | Обробляє нажаті клавіші у строці вводу | Об’єкт що надсилає подію, аргументи події | - |
| 55 | Excel | dataGridView1\_CellEndEdit | Обробляє завершення вводу данних до комірки користувачем | Об’єкт що надсилає подію, аргументи події | - |
| 56 | Excel | saveToolStripMenuItem\_Click | Обробляє нажимання на кнопку Save | Об’єкт що надсилає подію, аргументи події | - |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 58 | Excel | dataGridView1\_CellBeginEdit | Обробляє початок вводу данних до комірки користувачем | Об’єкт що надсилає подію, аргументи події | - |
| 59 | Excel | newToolStripMenuItem\_Click | Обробляє нажимання на кнопку New | Об’єкт що надсилає подію, аргументи події | - |
| 60 | Excel | aboutToolStripMenuItem\_Click | Обробляє нажимання на кнопку About | Об’єкт що надсилає подію, аргументи події | - |
| 61 | Excel | toolStripMenuItem2\_Click | Обробляє нажимання на будь який з останніх файлів | Об’єкт що надсилає подію, аргументи події | - |
| 62 | Excel | closeToolStripMenuItem\_Click | Обробляє нажимання на кнопку Close | Об’єкт що надсилає подію, аргументи події | - |
| 63 | Excel | functionsToolStripMenuItem\_Click | Обробляє нажимання на кнопку Docs | Об’єкт що надсилає подію, аргументи події | - |
| 64 | Excel | dataGridView2\_CurrentCellChanged | Обробляє зміну вибраної комірки в 2й таблиці | Об’єкт що надсилає подію, аргументи події | - |

Продовження таблиці 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів |
| 66 | New | getRows | Повертає введене число рядків | - | Число рядків |
| 67 | New | getCols | Повертає введене число стовпців | - | Число стовпців |
| 68 | New | button1\_Click | Обробляє нажимання на кнопку Create | Об’єкт що надсилає подію, аргументи події | - |
| 69 | New | New\_Load | Обробляє відкриття вікна | Об’єкт що надсилає подію, аргументи події | - |
| 70 | Save | button1\_Click | Обробляє нажимання на кнопку Yes | Об’єкт що надсилає подію, аргументи події | - |
| 71 | Save | button2\_Click | Обробляє нажимання на кнопку No | Об’єкт що надсилає подію, аргументи події | - |

### Стандартні функції

Стандартні функції, які використані в даній курсовій роботі, описані в   
таблиці 4.2.

Таблиця 4.2 – Опис стандартних функцій

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 1 | round | Округлення числа до найближчого цілого | Число, що необхідно округлити | Округлене число | сmath |
| 2 | abs | Знаходження абсолютного значення числа | Число для якого необхідно знайти абсолютне його значення | Абсолютне значення числа | cmath |
| 3 | min | Знаходження мінімуму двох чисел | Два числа з яких необхідно обрати мінімум | Мінімум з двох чисел | Algorithm |
| 4 | ToDouble | Приводить до типу double певну змінну | Змінна простого типу | Число з плаваючою крапкою | - |
| 5 | sin | Обчислення функції sin | Параметр функції | Значення функції для заданого параметру | сmath |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 6 | cos | Обчислення функції cos | Параметр функції | Значення функції для заданого параметру | сmath |
| 7 | tan | Обчислення функції tan | Параметр функції | Значення функції для заданого параметру | сmath |
| 8 | ctg | Обчислення функції ctg | Параметр функції | Значення функції для заданого параметру | сmath |
| 9 | ln | Обчислення функції ln | Параметр функції | Значення функції для заданого параметру | сmath |
| 10 | exp | Обчислення функції exp | Параметр функції | Значення функції для заданого параметру | сmath |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 11 | asin | Обчислення функції asin | Параметр функції | Значення функції для заданого параметру | сmath |
| 12 | acos | Обчислення функції acos | Параметр функції | Значення функції для заданого параметру | сmath |
| 13 | atan | Обчислення функції atan | Параметр функції | Значення функції для заданого параметру | Сmath |
| 14 | ShowDialog | Показує діалогове вікно | - | - | - |
| 15 | strcpy | Копіює рядок | 2 рядки | - | string.h |
| 16 | strcat | Зклеює рядки | 2 рядки | - | String.h |
| 17 | mkdir | Зтворює каталог | Рядок з адресою нового каталогу | Код помилки | direct.h |
| 18 | GetBytes | Перетворює вхідні данні на масив байтів | Данні | Масив байтів | - |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 19 | Write | Записує масив байтів у файл | Масив байтів, зміщення, розмір | - | - |
| 20 | WriteByte | Записує байт у файл | Байт данних | - | - |
| 21 | getenv | Повертає змінну системи | Ім’я змінної | Рядок зі змінною | stdlib.h |
| 22 | Read | Зчитує масив байт з файлу | зміщення, розмір | Масив байтів | - |
| 23 | ToInt32 | Перетворює масив байтів у int | Масив байтів | Число int | - |
| 24 | Resize | Змінює розмір массиву | Масив, новий розмів | - | - |
| 25 | ReadByte | Зчитує байт з файлу | - | Байт данних | - |
| 26 | Add | Додає елемент до коллекції | Елемент | - | - |
| 27 | Equals | Порівнює .net рядки | 2 рядки | Індикатор рівності рядків | - |

Продовження таблиці 4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| 28 | ToCharArray | Переводить рядок .net в масив символів | Рядок | Масив символів | - |
| 29 | Remove | Видаляє єлемент з коллекції | Елемент | - | - |
| 30 | Add | Додає стовпець | Ім’я стопвця, заголовок стовпця | - | - |
| 31 | Add | Додає рядки | Кількість рядків | - | - |
| 32 | clock | Повертає час | - | Час | time.h |
| 33 | OpenFile | Відкриває файл вибраний в діалозі вибору файла | - | - | - |
| 34 | ToString | Перетворює об’єкт у рядок | Об’єкт | Рядок | - |
| 35 | Close | Закриває вікно | - | - | - |

# Тестування програмного забезпечення

## Теорія

Тестування програмного забезпечення — це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки. Може оцінюватись:

* відповідність вимогам, якими керувалися проектувальники та розробники
* правильна відповідь для усіх можливих вхідних даних
* виконання функцій за прийнятний час
* практичність
* сумісність з програмним забезпеченням та операційними системами
* відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення багів (помилок або інших дефектів).

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершено). Процес розробки зазвичай передбачає коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

## План тестування

Усі можливі випадки виникнення помилок у програмі залежать від вхідних даних, тобто тої інформації, що отримується від користувача. Тому тестування програми полягає у виявленні правильності та коректності обробки програмою різних вхідних даних.

Задля виявлення усіх помилок у роботі програми потрібно запустити її на виконання в таких умовах:

1. Тестування правильності введених значень.
   1. Тестування при введенні некоректних посилань
   2. Тестування при введенні некоректних чисел
   3. Тестування при введенні неіснуючої функції
2. Тестування коректності роботи
   1. Перевірка коректності збереження файлу
   2. Перевірка коректності загрузки зі збереженого файлу
   3. Перевірка коректності роботи при створенні таблиці малих розмірів
   4. Перевірка коректності роботи при створенні таблиці великих розмірів
   5. Перевірка поведінки програми при закриті файлу, що містить не збережені зміни

Для вирішення відповідних помилкових ситуацій слід вдосконалити алгоритми роботи програми та обробити всі виключні ситуації.

Далі слід упевнитись, що усі методи програми коректно працюють на усіх наборах вхідних значень.

## Приклади тестування

### Тестування правильності введених значень

#### Тестування при введенні некоректних посилань

Якщо під час введення інформації в таблицю користувач введе посилання на комірку, що не існує (рисунок 5.1, рисунок 5.2, рисунок 5.3) або введе посилання у невірному форматі (рисунок 5.4), то в відповідну комірку програма видасть попередження.

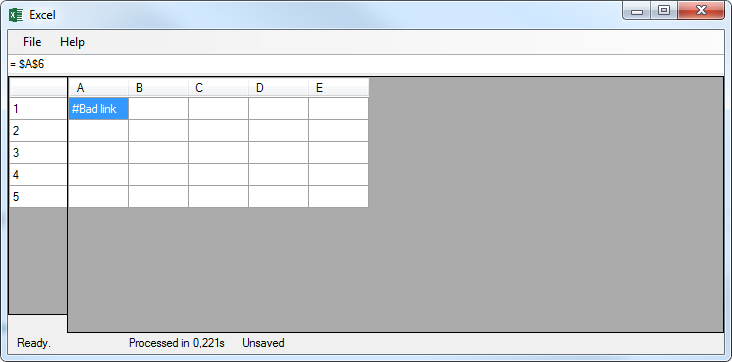


Рисунок 5.1 – Вікно програми, після введення посилання на неіснуючий рядок

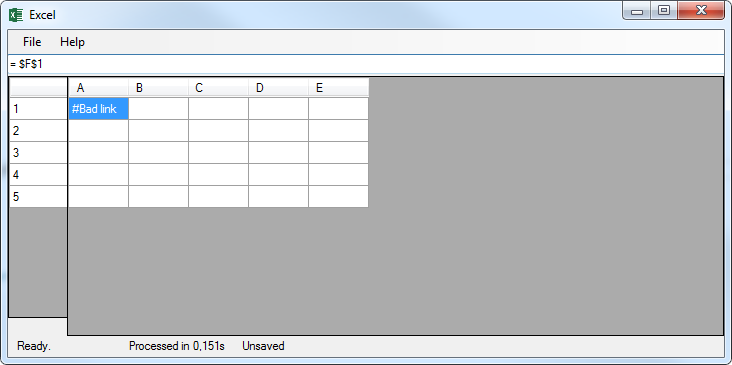


Рисунок 5.2 – Вікно програми, після введення посилання на неіснуючю колонку

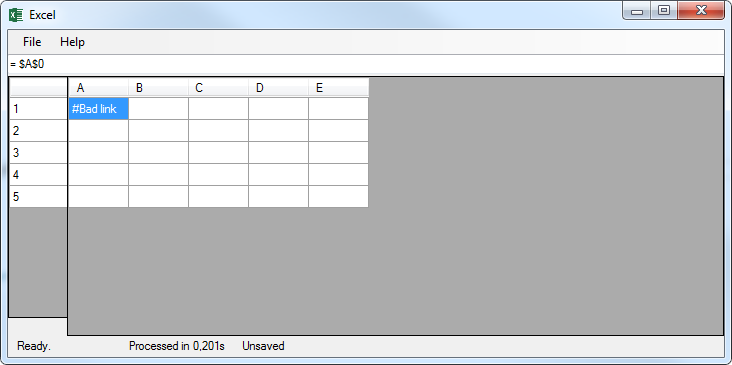


Рисунок 5.3 – Вікно програми, після введення посилання на неіснуючий рядок

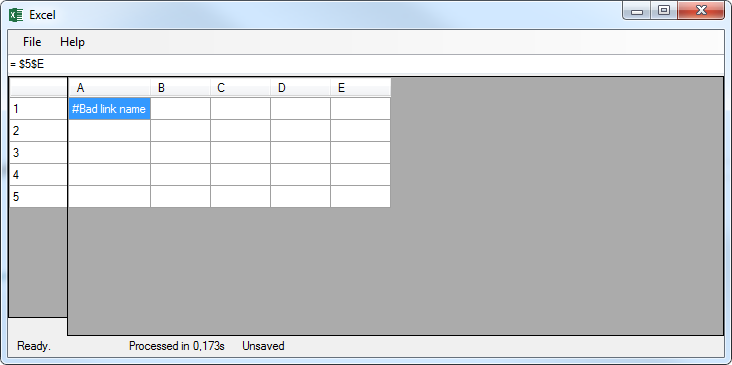


Рисунок 5.4 – Вікно програми, після введення посилання у некоректному форматі

#### Тестування при введенні некоректних чисел

Якщо під час введення інформації в таблицю користувач введе число у некоректному форматі (рисунок 5.5), то в відповідну комірку програма видасть попередження.

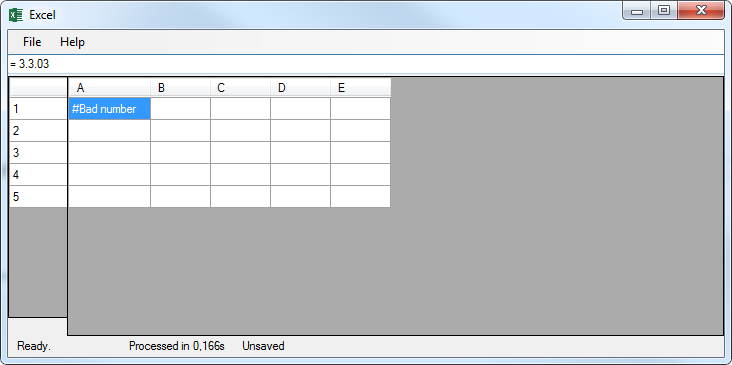


Рисунок 5.5 – Вікно програми, після введення числа у некоректному форматі

#### Тестування при введенні неіснуючої функції

Якщо під час введення інформації в таблицю користувач введе назву неісуючої функції (рисунок 5.6), то в відповідну комірку програма видасть попередження.

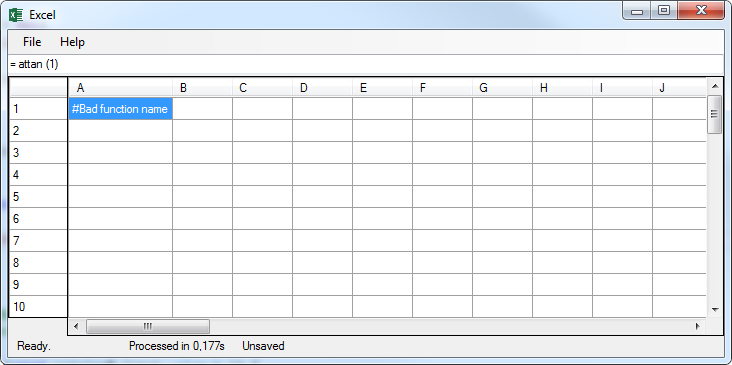


Рисунок 5.6 – Вікно програми, після введення неіснуючої функції

### Тестування коректності роботи

#### Перевірка коректності збереження файлу

Якщо під час користування програмним продуктом буде винесено якісь зміни у файл, то програма виведе відповідне повідомлення в строці статусу (рисунок 5.7). Відповідні зміни можна зберегти у новий файл або поточний. Тоді відповідно дане повідомлення зникне (рисунок 5.8).

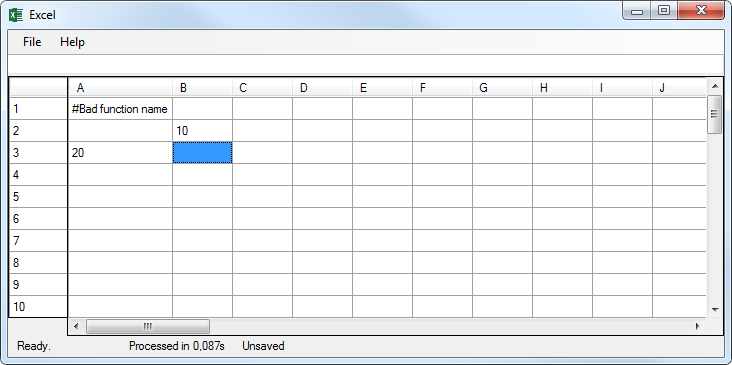


Рисунок 5.7 – Вікно програми, після внесення змін у файл

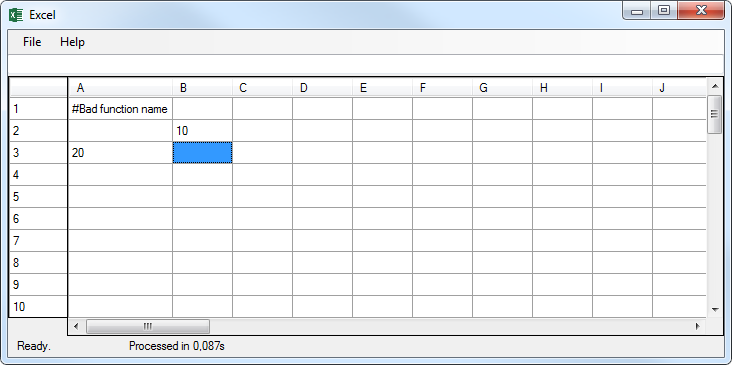


Рисунок 5.8 – Вікно програми, після збереження у файл

#### Перевірка коректності відкриття файлу

Відкриття збереженого файла в пункті 5.3.2.1 зображено на рисунок 5.9.

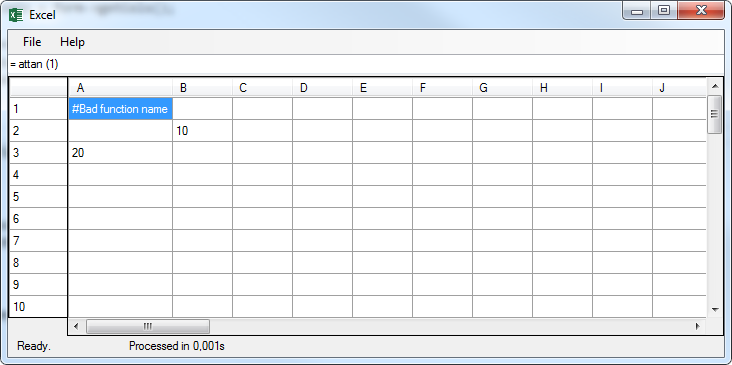


Рисунок 5.9 – Вікно програми, після відкриття таблиці з файлу

#### Перевірка коректності роботи при створенні таблиці малих розмірів

Якщо користувач створить таблицю мінімального можливого розміру (1х1) та введе в неї які-небудь дані, то програма спрацює коректно (рисунок 5.10).

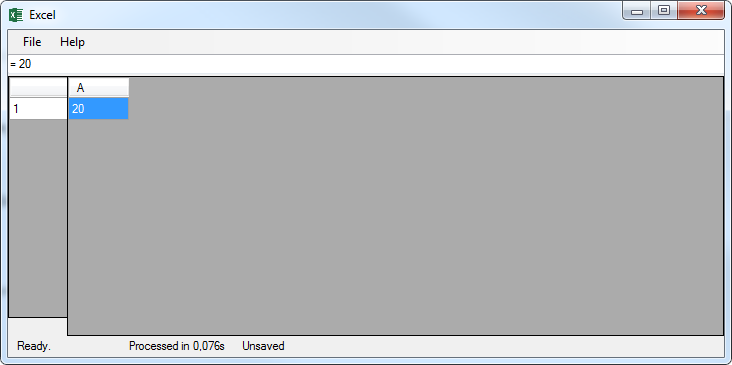


Рисунок 5.10 – Вікно програми, після створення таблиці мінімального розміру

#### Перевірка коректності роботи при створенні таблиці великих розмірів

Якщо користувач створить таблицю максимально можливого розміру (500х500) та введе в неї які-небудь дані, то програма спрацює коректно   
(рисунок 5.1).

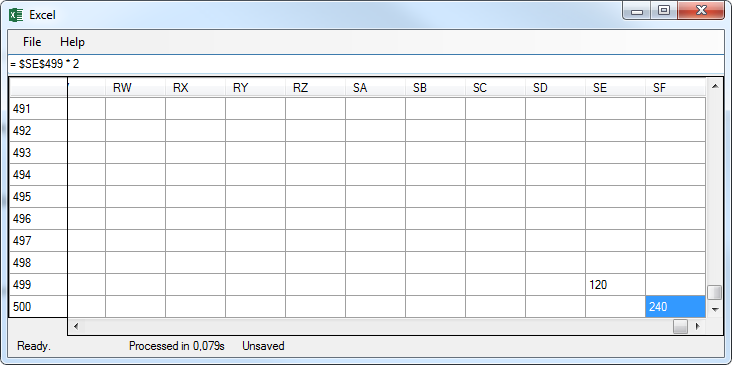


Рисунок 5.11 – Вікно програми, після створення таблиці максимального розміру

#### Перевірка поведінки програми при закриті файлу, що містить не збережені зміни

Якщо користувач внесе певні зміни у файл та спробує його зберегти то він отримає повідомлення про те, що файл містить не збережені зміни (рисунок 5.12).

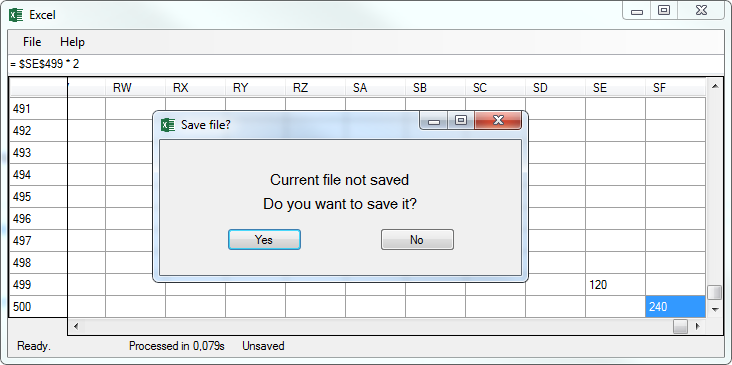


Рисунок 5.12 – Вікно програми, після спроби закрити файл, що містить незбережені зміни

### Перевірка вірності роботи програми, коли користувачем вірно введені всі дані та для обраного методу рішення сходиться

Результати тестування програми на коретних даних наведена у   
таблицях 5.1 - 5.3.

Таблиця 5.1 – Тестування обробки посилань

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректну роботу програми при використані посилань у формулі |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Розмір таблиці: 50 на 50 |
| Схема проведення тесту | Введення даних та натискання кнопки «Enter» |
| Очікуваний результат | Виведення рішення формули |
| Стан програми після проведення випробувань | Виведено рішення формули |

Таблиця 5.2 – Тестування обробки функцій

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректну роботу програми при використані функцій у формулі |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Розмір таблиці: 50 на 50 |
| Схема проведення тесту | Введення даних та натискання кнопки «Enter» |
| Очікуваний результат | Виведення рішення формули |
| Стан програми після проведення випробувань | Виведено рішення формули |

Таблиця 5.3 – Тестування обробки складних формул з багатьма посиланнями та функціями

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректну роботу програми при використані обробки складних формул з багатьма посиланнями та функціями |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Розмір таблиці: 50 на 50 |
| Схема проведення тесту | Введення даних та натискання кнопки «Enter» |
| Очікуваний результат | Виведення рішення формули |
| Стан програми після проведення випробувань | Виведено рішення системи |

Критичні ситуації у роботі програми виявлені не були. Під час тестування було виявлено, що більшість помилок виникало тоді, коли користувачем вводилися не коректні вхідні дані. Тому всі дані, які вводить користувач, ретельно провіряються на валідність і лише потім подаються на обробку програмі.

# Інструкція користувача

## Робота з програмою

Після запуску виконавчого файлу з розширенням \*.exe, відкривається головне вікно програми (Рисунок 6.1).

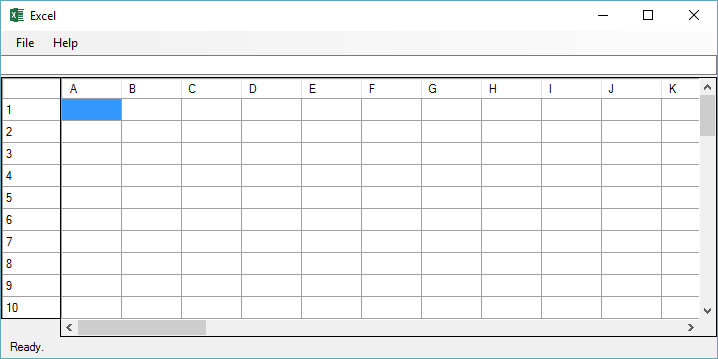


Рисунок 6.1 – Головне вікно програми

Далі за допомогою меню File -> New можна створити нову таблицю з заданою висотою та шириною (рисунок 6.2):

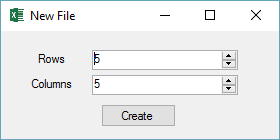


Рисунок 6.2 – Вибір необхідного розміру таблиці

Далі можливо редагувати данні в таблиці (Рисунок 6.3):

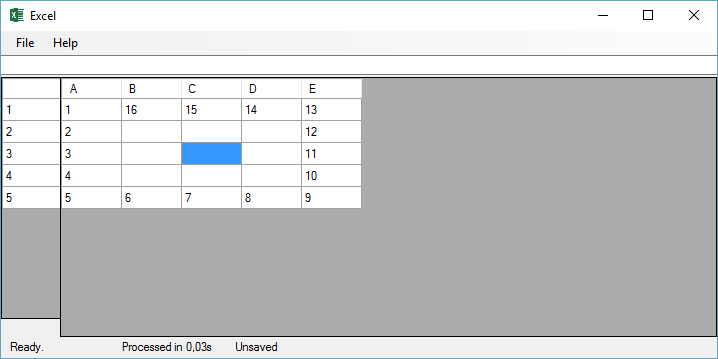


Рисунок 6.3 – Введення вхідних даних

Якщо якісь вхідні дані є формулами вони автоматично будуть підраховані(Рисунок 6.4).

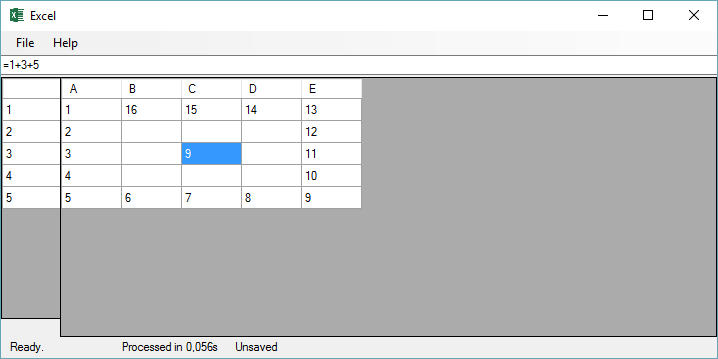


Рисунок 6.4 – Приклад формули

Якщо буде занесена неправильна формула програма сповістить про помилку (Рисунок 6.5):

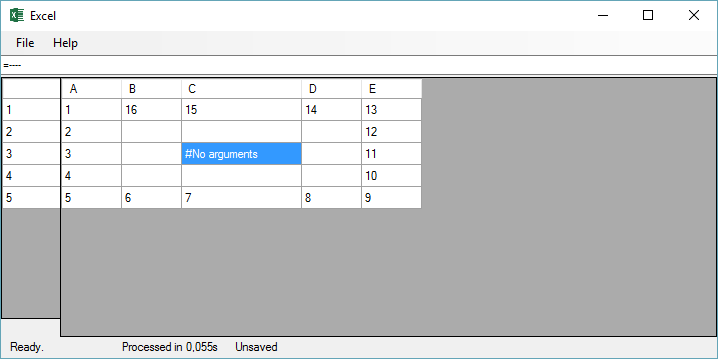


Рисунок 6.5 – Приклад повідомлення про помилку

В формулах можна посилатись на інші комірки (Рисунок 6.6):

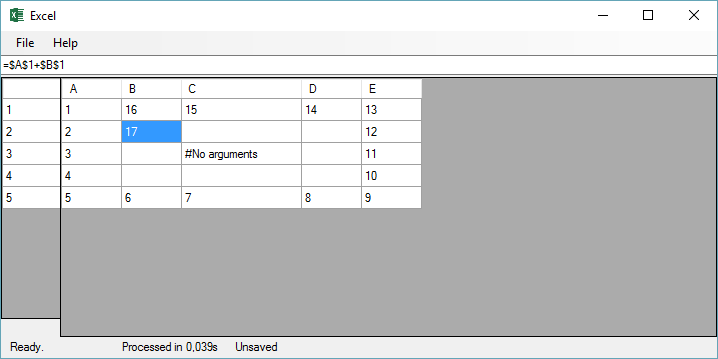


Рисунок 6.6 – Приклад складної формули

За допомогою меню File -> Save можна зберегти таблицю (Рисунок 6.7):

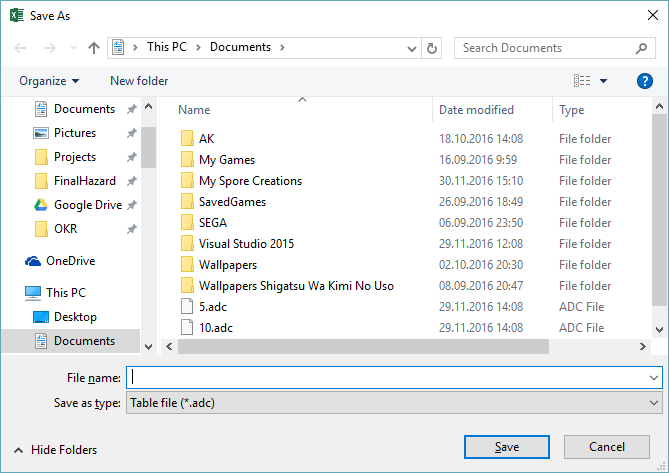


Рисунок 6.7 – Збереження таблиці

За допомогою меню File -> Open можна відкрити таблицю (Рисунок 6.8):

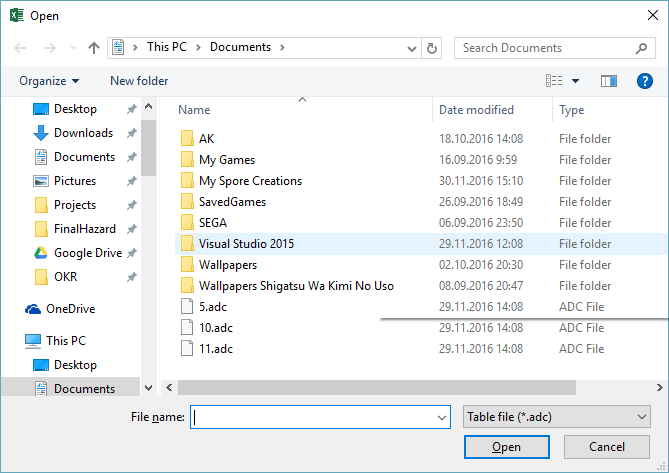


Рисунок 6.8 – Приклад відкриття таблиці

За допомогою меню Latest можна відкрити останні використані файли   
(Рисунок 6.9):

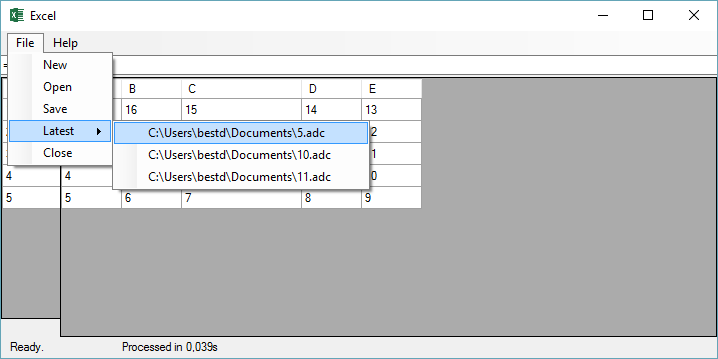


Рисунок 6.9 – Приклад останніх файлів

Також можливо закрити програму за допомогою меню File -> Close.

Якщо користувач спробує закрити змінений файл програма запропонує зберегти поточний файл.

## Формат вхідних та вихідних даних

Користувачем на вхід програми подається таблиця з даними та формулами.

Результатом є таблиця з підрахованими формулами.

## Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows 7/ Windows 8/Windows 10 | |
| Процесор | Intel® Pentium® IV  3.0 GHz або  AMD Athlon™ 64 3000+ 2.0 GHz | Intel® Pentium® D або AMD Athlon™ 64 X2 |
| Оперативна пам'ять | 1 GB RAM | 2 GB RAM |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Жорсткий диск | 512Мб | 1Гб |
| Дисплей | 800х600 | 1024х768 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Microsoft .Net Framework 4.5.2 або вище | |

Висновки

На етапі ознайомлення з теоретичними аспектами завдання даної курсової роботи було виявлено необхідні механізми для вирішення поставленої задачі та алгоритми їх реалізації, проаналізовано їх складність виконання, особливості та доцільність використання в межах реальних проектів, що було доведено реалізацією програмного забезпечення, яке розглядається в данні курсовій роботі.

На етапі проектування програмного забезпечення було реалізовано наступний функціонал: синтаксичний аналізатор, механізм перерахунку залежних формул за допомогою використання топологічного сортування, функції збереження та відкриття таблиці з провідника операційної системи Windows. Результатом розробки програми є набуття досвіду роботи з електронними таблицями, графами та структурами даних STL, закріплення, узагальнення та поглиблення знань і навичок програмування на мовах С\С++, набуття практичних навичок алгоритмізації прикладних задач.

Для спрощення процесу взаємодії з користувачем та розширення кола можливих користувачів програмного забезпечення був розроблений графічний інтерфейс. Також була розроблена інструкція користувача, що дозволяє ознайомитися зі всіма можливостями та ключовим моментами використання програмного забезпечення.

На етапі тестування було визначено, що програма вірно обробляє всі вхідні дані та видає очікуваний результат.

Результатом курсової роботи є створена програма, яку можна використовувати у навчально-інженерних цілях та/або бухгалтерському обліку з метою спрощення важких та об’ємних обчислень.

Перелік посилань

1. SPREADSHEET Defined for English Language Learners [Електронний ресурс] // Merriam-webster.com – Режим доступу:

<http://www.merriam-webster.com/dictionary/spreadsheet>

1. American Heritage Dictionary of the English Language (5th ed.). Houghton Mifflin Harcourt Publishing Company. 2011. “A software interface consisting of an interactive grid made up of cells in which data or formulas are entered for analysis or presentation”.
2. Collins English Dictionary – Complete and Unabridged (12th ed.). HarperCollins Publishers. 2014. “(Computer Science) a computer program that allows easy entry and manipulation of figures, equations, and text, used esp for financial planning and budgeting”.
3. DEFINITION spreadsheet [Електронний ресурс] // Wbatls.com – Режим доступу:

<http://whatis.techtarget.com/definition/spreadsheet>

1. Spreadsheet [Електронний ресурс] // Dictionary.com – Режим доступу:

<http://www.dictionary.com/browse/spreadsheet>

1. Spreadsheet [Електронний ресурс] // Webopedia.com – Режим доступу:

<http://www.webopedia.com/TERM/S/spreadsheet.html>

1. Spreadsheet [Електронний ресурс] // Computerhope.com – Режим доступу:

<http://www.computerhope.com/jargon/s/spreadsh.htm>

1. Hannan B. Higgins The Grid Book. – Cambridge, Massachusetts, U.S., The MIT Press, 2009. – 141 c.
2. Алгоритм сортувальної станції [Електронний ресурс] // Wikipedia.org – Режим доступу:

<https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%97_%D1%81%D1%82%D0%B0%D0%BD%D1%86%D1%96%D1%97>

1. Топологічне сортування [Електронний ресурс] // Wikipedia.org – Режим доступу:

<https://uk.wikipedia.org/wiki/%D0%A2%D0%BE%D0%BF%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F>

Додаток А Тексти програмного коду

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення*

Електронні таблиці

(Вид носія даних)

*CD-RW*

(Обсяг програми (документа), арк., Кб)

*арк, Кб*

*студентів групи ІП-51 ІІ курсу*

Булатова Дмитра Єгоровича

Зарічкового Олександра Анатолійовича

Худа Анна Олександрівна