

Софийски университет “Св. Климент Охридски”

ФМИ

Курсов проект

Тема:

Приложение за работа с електронни таблици

Изготвил:

Георги Костадинов Манчев

София

2023

Глава 1

Увод

1.1 Описание и идея на проекта

Проектът Приложение за работа с електронни таблици представлява програма, която работи с електронни таблици, поддържа коректни данни на всяка клетка и записва данните в йерархията. То има за цел да помага на потребителя да управлява данните си лесно и ефективно.

1.2 Цел и задачи за разработка

Целта на проекта е да се работи с електронни таблици. При подаване на файл, програмата проверява дали файла е валиден. Ако не съобщава за грешките. Ако проверките преминат успешно, програмата прочита файла отново и го съхранява в подходящите структури.

Задачите на проекта бяха разделени на 4 части. Първо разучаване на работата с електронни таблици и разписване на задание при разработка на програмния код. Втората задача беше валидация на данните в клетките, като тя беше разделена на подзадачи, които отговаряха на типовете данни. Третата задача беше да запазя данните в подходящата структура, а последната да имплементирам командите за манипулация на въпросната структура.

1.3 Структура на документацията

В документацията е разгледана идеята на проекта, предметната област, проектирането му и реализацията.

Глава 2

Преглед на предметната област

2.1 Основни използвани дефиниции и концепции

Използвани са основните концепции на обектното ориентирано програмиране – енкапсулация, полиморфизъм, наследяване и абстракция.

2.2 Дефиниране на проблеми и сложност на поставената задача

Първия проблем е как да бъде валидиран един текстов файл. Втория проблем е създаването на обектите от файла и разпознаването на командите.

2.3 Подходи и методи за разрешаване на поставените проблеми

Подходът към първият проблем е следния – една електронна таблица може да се състои от обикновен тип стойности.

Стойностите са няколко типа – символен низ, цяло число, дробно число и формула. Във файла всеки елемент има специфичен начин на записване. При валидиране се следят кавичките на стринговете, цифрите на целите числа(също така и минус), цифрите и точките на дробните числа(също така и минус) и първият елемент да бъде '=' за формулата.

Подходът към втория проблем е подобен на този за валидацията, но се изпускат проверките, тъй като знаем че всички данни във файла са коректни. Използван е обектно ориентирания похват „Фабрика“ за разпознаване и създаване на обект от конкретния тип. Също така този похват е използван, за създаване на команда,

Глава 3.

Проектиране

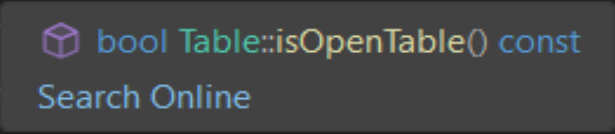
[illegible]

Линк за диаграмата: <https://ibb.co/Ltzrczn>

А другия е за типовете команди. Той се наследява от класовете, които представляват тези команди.

3.2 Най-важни извадки от кода

```
void OpenFile::execute(Table& table, const char* buffer)
{
    std::ifstream file(buffer);
    if (!file.is_open()) {
        std::ofstream newFile(buffer);
        newFile.close();
        throw std::invalid_argument("File is not open.");
    }
    unsigned rows = getRowCount(file) + 1;
    if (!isValidFileData(file, rows)) {
        return;
    }
    if (table.isOpenTable()) {
        throw std::invalid_argument("Table is already open.");
    }
    readTable(table, file);
    table.setOpen();
    file.close();
    std::cout << "Successfully opened " << buffer << std::endl;
}
```



Фигура 2 – Команда “open”

На **Фигура 2** е изобразен кода, който изпълнява функционалността “open”. Първо отваря файла за четене, след това проверява дали файла има валидни данни, връща се в началото, като преди това възстановява доброто му съдържание и прочита файла.

```

static Command* commandFactory(const char* cmd, char filePath[1024]) {
    if (strcmp(cmd, "open") == 0) {
        std::cout << "Enter file path: ";
        std::cin.getline(filePath, 1024);
        return new OpenFile();
    }
    else if (strcmp(cmd, "save") == 0) {
        return new Save();
    }
    else if (strcmp(cmd, "save as") == 0) {
        return new SaveAs();
    }
    else if (strcmp(cmd, "print") == 0) {
        return new Print();
    }
    else if (strcmp(cmd, "edit") == 0) {
        return new EditCell();
    }
    else if (strcmp(cmd, "close") == 0) {
        return new Close();
    }
    else {
        throw std::invalid_argument("Invalid command. ");
    }
}

```

Фигура 3 – Фабрика, която създава командите

Фабриката разпознава дадена команда, спрямо подадения масив от символи, дали съвпада, с някоя от тях и така определя какъв тип команда да създаде.

Глава 4

Реализация, тестване

4.1 Реализация на класовете

Базовият клас за типовете – Cell

Наследници:

- IntCell
- DoubleCell
- StringCell
- FormulaCell

Базовият клас за команди – Command

Наследници:

- OpenFile
- Save
- Save as
- Edit
- Print
- Close

Помощни класове:

- Table
- Commands
- CellFactory
- CommandFactory
- SomeFunc

4.2 Управление на паметта и алгоритми. Оптимизации

Оптимизирано е при добавяне на клетка да не се копира, а направо да се добавя по указател, защото обекта създаден от фабриката е указател и няма смисъл да бъде копиран по стойност.

4.3 Планиране, описание и създаване на тестови сценарии

Тестовите сценарии са следните – създаваме файл, в който запазваме Електронна таблица и след подаване на файла на команда open, могат да се изпълняват всички други команди.

Команди, които обекта поддържа:

- open
<file name> - проверява дали съдържанието на файла е коректно и зарежда съдържанието му в класа Table

- save – запазва данните на класа Table, във файла от който са прочетени първоначално
- save as
<file name> - запазва съдържанието на класа Table в указан от потребителя файл
- print – извежда в терминала цялото съдържанието на класа Table
- edit
<row>
<column>
<new data> - запазва <new data> на клетката с указан ред и колона, като преди това проверява дали <new data> е валиден тип, проверява дали реда или колона не излиза от таблица и изтрива старото съдържание на тази клетка
- close – изтрива съществуващата таблица, но програмата може да продължи работа
- exit – приключва програмата

Може да се тества програмата чрез подаване на грешна таблица, за да се провери коректността на валидирането. Например:

```
"Georgi",=2+a,,123
"G T",43,23.2,123
=R1C1*R1C2,,,
,,,
,,,

```

Фигура 4 – валидна таблица

```
"string", 12.5, 17
Invalid, 123, =R1C1+23

```

Фигура 5 – невалидна таблица

```
"string", 12.5, 17
Invalid cell, 123, =R1C1+23

```

Фигура 6 – невалидна таблица

Глава 5

Заклучение

5.1 Обобщение на изпълнението на началните цели

Успях да изпълня проекта по начина, по който исках първоначално.

5.2 Насоки за бъдещо развитие и усъвършенстване

За бъдещо развитие бих добавил командите да могат да се изпълняват от текстови файл, освен от терминала.

Използвана литература

<https://refactoring.guru/design-patterns/factory-comparison> - информация за дизайн pattern тип "Фабрика"

<https://cplusplus.com/> - документация на C++

https://github.com/Justsvetoslavov/Object-oriented_programming_FMI/tree/master/Seminars - семинарни упражнения по ООП

Презентации и лекции от курса по ООП