Увод

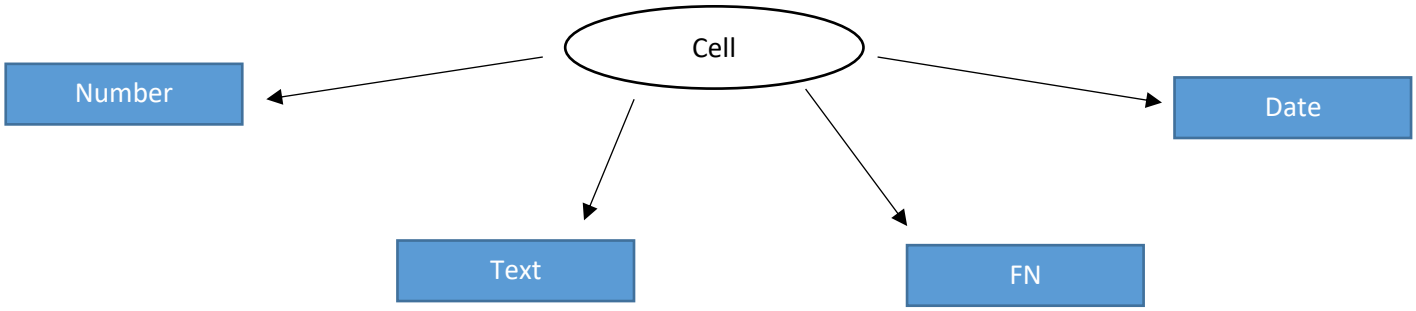
- Идеята на проекта е, разработената програма да взима избран от потребителя файл, от тип CSV, и да осъществяваме различни промени върху стойностите на файлът (добавяне на минимални/максимални стойности, изтриване на повтавящи редове, сортиране, филтриране и други) и при желание да се запишат отново във файл.

- Програмата работи с четири вида клетки: текст, число, факултетни номера и дата. Всяка една от тези клетки се определя от стрингът, който е записан в файла от който се чете. Всички клетки имат зададен тип след прочит от файл.

- Структурата на документацията главно се води по дадената примерна структура на документацията. Разликата е, че имам за някой от главите съм сметнал добре да бъдат разменени.

Проектиране

- В проекта същесвуват седем класа: Cell,CellFactory,Date,Number,FN,Text и CSV. Първо ще разгледам базовият абстрактен клас Cell. Той съдържа в себе си единствено типът на клетката, който ще бъде от полза, когато правя проверка за валидност на файла(дали всяка стойност от една колона от еднакиви типове). Той се наследява по следния начин:



<u>Всеки клас съдържа уникални за него харакеристики:</u>

- Number – съдържа в себе си число от тип double
  - `double number;`

- FN – съдържа в себе си факултетния номер (записан като стринг), специалността и дали факултетния номер е от новите или не.
  - `bool isNewFN;`
  - `Speciality spec;`
  - `std::string number;`

- Date – съдържа в себе си по едно число съответсващо на година, месец и ден на записанта дата.
    - `unsigned day;`
    - `unsigned month;`
    - `unsigned year;`

- Text– съдържа в себе си текст.
    - `std::string text;`

Останалите класове:

- CellFactory – съдържа в себе си функции, които опредeнят по подаден стринг, какъв тип е клетката и важната функция createNewCell, чрез която създавам клетките.
- CSV – съдържа в масив от структурата Row, която съответства на ред в таблицата.
    - Row – съдържа масив от динамично заделени обекти от тип Cell*, както и капацитет и размер.
        - `Cell** cells;`
        - `unsigned size;`
        - `unsigned capacity;`

  Освен Row, CSV класът съдържа: указател към предишното си състояние; размер и капацитет на редовете; името на файлът, от който е взета информацията, масив от числа, който определят дължината на една колона(ще послужи при принтиране) и помощни булеви променливи, които дават информация дали тази дължина е коректна, дали файлът е коректен, дали има хедър ред и дали досега е отварян CSV файл.

    - `mutable bool isMaxLenCorrect;`
    - `mutable bool isCorrect;`
    - `mutable bool isHeaderLine;`
    - `bool isOpenedAtleastOneTime;`
    - `unsigned rowsCount;`
    - `unsigned rowsCapacity;`
    - `std::string nameOfFile;`
    - `mutable std::vector<unsigned> lenght;`
    - `Row* rows;`
    - `CSV* previousState;`

Реализация

- Функцията до която външният свят има достъп е enterCommand(). Чрез нея се влиза в безкраен цикъл, в който се изисква от вас да напишете името на функцията, която желате да се изпълни и ако има съответствие в имената, то тя ще се изпълни.

- Всички функционалности, който са постигнати са направени, чрез „парсване“ на стрингове.

- Единственият алгоритъм, който съм използвал е „Метода на Мехурчето“,когато трябва да се изпълни някакъв вид сортиране. Останалите „алгоритми“ са цикли и функции измислени от мен, за да ми помогнат за успешното изпълнение на желата функционалност.

Разяснение на функциите:

- `void copyRow(const std::string& text);` – Копиране на ред по индекс (в края на файла)
    - По подаден стринг, аз създавам индекс, който съответства на даден ред. Запазвам предишното състояние. Ако се наложи отделям място за нови редове и чрез оператор=, създавам нов ред, копие на друг.
- `void changeColumns(const std::string& permutation);` – Размяна на колони по дадена пермутация.
    - По подаден стринг, аз прочитам числата и ги записвам с вектор.Запазвам предишното състояние. Създавам нов масив от редове, в който взимам всяка клетка с индекса от пермутацията. На всяка клетка от старите редове и давам стойност nullptr, защото при delete[] rows, ще изтрия и самите клетки.
- `void removeIdenticalRows();` – Премахва еднакви редове.
    - Запазва предишното състояние. Чрез оператор== разбирам кои редове съвпадат и ги изтривам.
- `void removeColumn(const std::string& text);` – Премахва дадена колона по име на колоната/ индекс.
    - По подаден стринг, аз създавам индекс, който съответства на даден колона. Запазва предишното състояние. За всеки ред изтривам клетката с този индекс и премествам всички клетки след нея с една позиция назад. (За да запазя същата пермутация)
- `void addMinValues();` – Добавя нов ред с минималните стойности, за всяка колона
    - Запазва предишното състояние. Отделям памет за нов ред, при необходимост. Обхождам всяка колона, като сравнявам всяка клетка, чрез оператор>, който е чисто виртуален и е разписан за всеки клас. Намирам минималният елемент и го добавям в новия ред.
- `void addMaxValues();` – Добавя нов ред с максималните стойности, за всяка колона
    - Запазва предишното състояние. Отделям памет за нов ред, при необходимост. Обхождам всяка колона, като сравнявам всяка клетка, чрез

оператор<, който е чисто виртуален и е разписан за всеки клас. Намирам максималният елемент и го добавям в новия ред.

- `void mostCommon();` – Добавя нов ред с най-често срещаните стойности, за всяка колона

    - Запазва предишното състояние. Отделям памет за нов ред, при необходимост. Обхождам всяка колона, като сравнявам всяка клетка и запазвам в вектор колко пъти е срещан елемента в дадената колона. Следя кой е най-големият елемент и неговият индекс. За най-често срещан елемент взимам елемент с индекси за ред и колона – индексът който съм запазил и обхожданата колона. Обхождам още веднъж колоната като първо сравнявам дали елемента, който съответства на някакво число в векторът е толкова, колкото най-големият елемент. Ако да се прави проверка дали елементът е по-малък от текущия най-често срещан. Ако да то той се запазва като най-често срещан.

- `void sortByNumberOrName(const std::string& text);- Сортира таблицата, по име на колона/индекс и някакво условие`

    - Остановявам коя е желаната колона за сортировка, както и операцията която е подадена. Запазвам предишното състояние. Използвам Метода на Мехурчето и сравнявам елементите от посочената колона и ако изпълняват разменям редовете.

- `void sortByNumberOrName(const std::string& text);- Сортира редовете, по име на колона/индекс и някакво условие`

    - Остановявам коя е желаната колона за сортировка, както и операцията която е подадена. Запазвам предишното състояние. Използвам Метода на Мехурчето и сравнявам елементите от посочената колона и ако изпълняват разменям редовете.

- `void filterByValue(const std::string& text);` – Филтрира (премахва ред ако не е изпълнено условие) редовете, по име на колона/индекс, някакво условие и стойност, с която се сравнява

    - Остановявам коя е желаната колона за филтриране, както и операцията която е подадена. Създавам нов обект, който да сравнявам. Запазвам предишното състояние. Обхождам по колони, и ако условието не е изпълнено за дадена клетка от колоната, изтривам редът на който се намира.

- `void undo();` – Връща една стъпка назад.

    - По време на изпънение на гореспоменатите функции, съм запазил предишното състояние на таблицата и чрез оператор= на *this задавам предишните стойности. (Възможно най-не оптималното, но е това за което сам съм успял да измисля)

Тестване

- За тестване съм подговтвил файл TEST.csv в в който има таблица, която съдържа всякакъв тип стойности. По време на защитата мога да представа функционалностите, както и да отговоря на въпроси свързани с кода.

Заключение

- Началните цели да се да се изпълнят всички функционалности са постигнати. Старал съм се за спазвам всички възможни принципи на ООП. Оптималността е сравнително добра според мен, освен за undo функцията. За нея препоръчвам на себе се да измисля друг начин.

- Източник на една от функциите (sprintf):

  https://www.geeksforgeeks.org/cpp-program-for-double-to-string-conversion/

Благодаря за вниманието и съжалявам ако има правописни и пунктуационни грешки!