ДОКУЕМНТАЦИЯ НА ПРОЕКТ

Име: Марк Огнянов Весков

Специалност: Информационни системи

Административна група: 1

Факултетен номер: 9MI0700339

Тема №5. Работа със CSV файлове

Съдържание

1. Въведение

Кратко описание на задачата

CSV Table Manager е приложение за управление на CSV файлове, разработено без използване на STL библиотеката. Проектът реализира функционалности за четене, обработка, модификация и запазване на табличните данни от CSV файлове с интерактивен потребителски интерфейс.

Основна идея на решението

Решението се базира на създаването на собствени имплементации на основните структури данни и алгоритми, които обикновено се предоставят от STL:

- Собствен String клас - заменя std::string с функционалност за работа със символни низове
- Собствен MyArray клас - заменя std::vector с динамично разширяване и основни операции
- Таблична структура - организира данните в колони и редове с автоматично разпознаване на типовете данни
- Командна система - интерактивен интерфейс за изпълнение на операции върху данните

Приложението поддържа различни типове данни и осигурява операции за сортиране, филтриране и обработка на данните.

2. Анализ на задачата и подход за решаване

Основни изисквания на задачата

- **Зареждане и запазване на CSV файлове** с поддръжка на различни разделители и опционални заглавни редове
- **Автоматично разпознаване на типове данни** (текст, числа, валута, ЕГН, факултетни номера)
- **Сортиране** на данни по произволна колона във възходящ или низходящ ред
- **Филтриране** на редове според различни критерии (равенство, неравенство, сравнения)
- **Управление на структурата** - добавяне/премахване на колони и редове

- **Работа с дублирани данни** - откриване и премахване на дублиращи се редове
- **Undo функционалност** за отмяна на последната операция
- **Интерактивен потребителски интерфейс** с команди

Разбивка на решението на отделни модули

Базови структури от данни

- **String клас** - заместител на std::string с динамично управление на паметта
- **MyArray шаблон** - заместител на std::vector с автоматично преоразмеряване

Типове данни и разпознаване

- **DataType енумерация** - дефиниране на поддържаните типове данни
- **Функции за валидация** - проверка на валидност на ЕГН, факултетни номера, числа и валута
- **Автоматичен type detection** - анализ на съдържанието за определяне на типа

Структура на таблицата

- **Cell клас** - представяне на една клетка с стойност и тип
- **Column клас** - колона с клетки и метаданни
- **Table клас** - главна структура, съдържаща колони и операции

Файлови операции

- **CSV парсър** - четене и анализ на CSV формат с поддръжка на кавички
- **CSV експорт** - запазване на данни в CSV формат
- **Конфигуриране** - различни разделители и настройки

Алгоритми и операции

- **QuickSort** за сортиране на данни

- **Филтриране** с различни условия
- **Статистически операции** - min, max, най-често срещана стойност

Потребителски интерфейс

- **Команден парсър** - анализ на потребителски команди
- **Интерактивен цикъл** - обработка на команди
- **Помощни функции** - показване на помощ и валидация на входни данни

Алгоритъм за сортиране

Избран е **QuickSort** поради:

- Средна сложност O(n log n)
- Ниска употреба на памет
- Добра производителност при реални данни

Управление на паметта

Използван е RAII принципа (Resource Acquisition Is Initialization):

- Автоматично освобождаване на памет в деструкторите
- Копиране с copy constructors и assignment operators
- Избягване на memory leaks чрез систематично управление

Алтернативни подходи и защо не са избрани

Използване на масиви вместо динамични структури

Разгледан подход: Фиксирани масиви за съхранение на данни

Защо отхвърлен:

- Ограничаване на размера на таблицата
- Неефективно използване на памет
- Невъзможност за динамично добавяне на колони

Merge Sort вместо Quick Sort

Разгледан подход: Сортиране с O(n log n) сложност

Защо отхвърлен:

- По-висока употреба на памет
- По-сложна имплементация без STL
- QuickSort е достатъчно ефективен

Подход към предизвикателства

Четене на данни от файл

Проблем: Парсиране на CSV с различни разделители и escape characters

Решение:

- Четене символ по символ
- Поддръжка на кавички (единични и двойни) за escape-ване

Управление на памет

Проблем: Избягване на memory leaks без STL

Решение:

- Строго прилагане на RAII принципа
- Правилно имплементиране на copy constructors и assignment operators
- Систематично използване на delete[] за масиви

Сравнения между различни типове

Проблем: Как да сравняваме различни типове данни безопасно

Решение:

- Строга типова проверка преди сравнение
- Специализирани compare функции за всеки тип
- Exception handling за неподдържани сравнения

Undo функционалност

Проблем: Как да запазим предишното състояние без голяма употреба на памет

Решение:

- Shallow backup система - запазваме само една предишна версия
- Copy-on-write семантика за колоните

3. Архитектура и описание на програмата

3.1 Общ преглед на архитектурата

Основни модули/компоненти

Проектът е построен върху модулна архитектура без използване на STL библиотеката, като всички основни структури от данни са имплементирани от нулата. Основните компоненти са:

- String клас - Собствена имплементация на низ с динамично управление на паметта
- MyClass<T> клас - Шаблонен контейнер за динамични масиви с автоматично преоразмеряване

Бизнес логика за данни

- DataType енумерация - Дефинира поддържаните типове данни (TEXT, NUMBER, CURRENCY, EGN, FACULTY_NUMBER)

- Cell клас - Представя единична клетка от таблицата със стойност и тип
- Column клас - Управлява колона от клетки с операции за анализ (min, max, mode)

Таблично управление

- Table клас - Централният компонент за работа с CSV данни, включващ:
    - Зареждане и запазване на CSV файлове
    - Сортиране, филтриране и манипулация на данни
    - Undo функционалност
    - Управление на дублирани редове и колони

Потребителски интерфейс

- CSVManager клас - Командно-редов интерфейс за взаимодействие с потребителя
    - Парсер на команди и валидация на входни данни

Помощни функции

- Utils – файл с помощни функции, включващ:
    - Функции за конвертиране на типове данни
    - Валидатори за специални формати (ЕГН, факултетен номер)
    - Парсъри за CSV формат

3.2 Описание на класовете

3.2.1 Клас String

Предназначение

Клас **String** представлява собствена имплементация на символен низ, предназначена да замести стандартния std::string. Осигурява динамично управление на паметта за съхранение на текстови данни с автоматично преоразмеряване при нужда.

Основни член-данни и тяхното значение

```
private:
    char *data_;              // Указател към динамично заделената памет за символите
    size_t size_;             // Текущ брой символи в низа
    size_t capacity_;         // Общ капацитет на заделената памет
```

Основни член-функции и тяхната роля

```
// Конструктори и деструктор
String();                                   // Конструктор по подразбиране
String(const char *str);                    // Конструктор от C-string
String(const String &other);                // Копиращ конструктор
~String();                                  // Деструктор

// Оператори за присвояване
String &operator=(const String &other);
String &operator=(const char *str);

// Достъп до данните
const char *c_str() const;                  // Връща C-string представяне
size_t length() const;                      // Връща дължината
char &operator[](size_t index);             // Достъп до символ по индекс

// Модификации
void pushBack(char c);                      // Добавя символ в края
String substr(size_t pos, size_t len) const; // Извлича подниз
String operator+(const String &other) const; // Конкатенация

// Сравнения
```

bool operator==(const String &other) const;

bool operator<(const String &other) const;

Връзки между класовете

- **Композиция с класовете** Cell, Column, Table - използва се като основен тип за съхранение на текстови данни
- **Независим клас** - не наследява други класове и не се наследява

Прилагане на принципи на ООП

- **Капсулация**: Всички член-данни са private, достъпът се осъществява чрез public методи
- **RAII**: Автоматично управление на ресурсите чрез конструктори и деструктор
- **Претоварване на оператори**: Осигурява интуитивен синтаксис за работа с низове

3.2.2 Клас MyArray<T>

Предназначение

Динамичен масив с автоматично управление на размера, предназначен да замести std::vector. Осигурява типово-безопасно съхранение на елементи с възможност за динамично добавяне и премахване.

Основни член-данни и тяхното значение

private:

T *data_; // Указател към масива от елементи

size_t size_; // Брой валидни елементи

size_t capacity_; // Общ капацитет на масива

Основни член-функции и тяхната роля

// Конструктори и деструктор

MyArray(); // Конструктор по подразбиране

```
MyArray(const MyArray &other);          // Копиращ конструктор
~MyArray();                             // Деструктор

// Управление на елементи
void pushBack(const T &value);          // Добавя елемент в края
void erase(size_t index);               // Премахва елемент по индекс
void clear();                           // Изчиства всички елементи

// Достъп
T &operator[](size_t index);            // Достъп по индекс
size_t size() const;                    // Връща броя елементи
bool empty() const;                     // Проверява дали е празен
// Итератори
T *begin();                             // Указател към първия елемент
T *end();                               // Указател след последния елемент
```

Връзки между класовете

- **Композиция** в Column, Table, CSVManager - използва се за съхранение на колекции данни
- **Шаблонен клас** - може да работи с различни типове данни

Прилагане на принципи на ООП

- **Капсулация**: Скрива детайлите на имплементацията
- **Шаблони**: Осигурява типова гъвкавост
- **RAII**: Автоматично управление на паметта

3.2.3 Клас Cell

Предназначение

Представлява единична клетка в таблицата, съхраняваща стойност и нейния тип данни. Осигурява типово-специфични операции за сравнение и обработка.

Основни член-данни и тяхното значение

private:

```
String value_;          // Стойност на клетката като текст
DataType type_;         // Тип на данните в клетката
```

Основни член-функции и тяхната роля

```
// Конструктори
Cell();                                          // Празна клетка
Cell(const String &value, DataType type);    // Клетка със стойност и тип

// Достъп до данните
const String &getValue() const;                  // Връща стойността
DataType getType() const;                        // Връща типа
void setValue(const String &value);              // Задава стойност
void setType(DataType type);                     // Задава тип

// Оператори за сравнение
bool operator==(const Cell &other) const;
bool operator<(const Cell &other) const;
bool operator>(const Cell &other) const;
```

Връзки между класовете

- **Композиция** с String - използва String за съхранение на стойността
- **Агрегация** в Column - Column съдържа колекция от Cell обекти

- **Асоциация** с DataType enum - определя типа на данните

Прилагане на принципи на ООП

- **Капсулация**: Скрива вътрешното представяне на данните
- **Претоварване на оператори**: Осигурява естествен синтаксис за сравнения
- **Енкапсулация на бизнес логика**: Специализирани алгоритми за сравнение според типа данни

3.2.4 Клас Column

Предназначение

Представлява колона в таблицата, съдържаща име, тип данни и колекция от клетки. Осигурява операции за анализ и манипулация на данните в колоната.

Основни член-данни и тяхното значение

private:

```
String name_;                  // Име на колоната
DataType type_;                // Тип данни на колоната
MyArray<Cell> cells_;      // Колекция от клетки в колоната
```

Основни член-функции и тяхната роля

```
// Управление на колоната
const String &getName() const;                  // Връща името
void setName(const String &name);               // Задава име
DataType getType() const;                       // Връща типа
void setType(DataType type);                    // Задава тип

// Управление на клетки
const Cell &getCell(size_t index) const;        // Достъп до клетка
```

void setCell(size_t index, const Cell &cell); // Задава клетка

void addCell(const Cell &cell); // Добавя клетка

void removeCell(size_t index); // Премахва клетка

// Статистически операции

Cell getMinValue() const; // Минимална стойност

Cell getMaxValue() const; // Максимална стойност

Cell getMostFrequentValue() const; // Най-честа стойност

Връзки между класовете

- **Композиция** със String (за името)
- **Композиция** с MyArray<Cell> (за клетките)
- **Агрегация** в Table - Table съдържа множество Column обекти
- **Асоциация** с DataType enum

Прилагане на принципи на ООП

- **Капсулация**: Скрива сложността на статистическите изчисления
- **Единична отговорност**: Отговаря само за управлението на една колона
- **Енкапсулация на данни**: Осигурява контролиран достъп до клетките

3.2.5 Клас Table

Предназначение

Основен клас за представяне на цялата таблица. Управлява колекция от колони и осигурява функционалности за зареждане, запазване, сортиране, филтриране и други операции върху табличните данни.

Основни член-данни и тяхното значение

private:

```
MyArray<Column> columns_;                // Колекция от колони
String filename_;                        // Име на файла
bool hasChanges_;                        // Флаг за несъхранени промени
char delimiter_;                         // Разделител за CSV
// За undo функционалност
MyArray<Column> backupColumns_; // Backup копие на колоните
bool hasBackup_;                         // Флаг за наличие на backup
```

Основни член-функции и тяхната роля

```
// Файлови операции
bool loadFromCSV(const String &filename, bool hasHeaders = true, char delimiter = ',');
bool saveToCSV(const String &filename = String(""));


// Визуализация
void print() const;                                    // Принтира таблицата

// Операции за трансформация
void sort(const String &columnName, bool ascending = true);
void filter(const String &columnName, const String &op, const String &value);
void removeDuplicateRows();                            // Премахва дублиращи редове

// Управление на структурата
void removeColumn(const String &columnName);
void duplicateColumn(const String &columnName, const String &newName = String(""));
void setCellValue(const String &columnName, size_t row, const String &value);
```

```
void addRow(const String &type, int sourceRow = -1);


// Undo функционалност
void undo();                                    // Отменя последната операция
bool hasUnsavedChanges() const;                 // Проверява за несъхранени промени
```

Връзки между класовете

- **Композиция** с MyArray<Column> - съдържа колекция от колони
- **Композиция** с String - за името на файла
- **Асоциация** с Cell - работи с клетки чрез колоните
- **Използва** utility функции от Utils модула

Прилагане на принципи на ООП

- **Капсулация**: Скрива сложността на файловите операции и алгоритмите
- **Единична отговорност**: Отговаря за управлението на цялата таблица
- **Композиция**: Изгражда се от по-прости компоненти (Column, String)

3.2.6 Клас CSVManager

Предназначение

Главен клас за управление на приложението. Осигурява потребителски интерфейс и координира работата между различните компоненти на системата.

Основни член-данни и тяхното значение

```
private:
    Table table_;           // Таблицата, с която работи приложението
    bool running_;          // Флаг за състоянието на приложението
```

Основни член-функции и тяхната роля

```
public:
```

```
    CSVManager();                                   // Конструктор
    void run();                                     // Главният цикъл на приложението

private:
    // Парсване и обработка на команди
    MyArray<String> tokenize(const String &line);  // Разделя командата на токени
    void showHelp();                                // Показва помощна информация

    // Обработчици на команди
    void handleOpen(const MyArray<String> &tokens);
    void handleSave(const MyArray<String> &tokens);
    void handleSort(const MyArray<String> &tokens);
    void handleFilter(const MyArray<String> &tokens);
    void handleRemoveColumn(const MyArray<String> &tokens);
    void handleDuplicateColumn(const MyArray<String> &tokens);
    void handleSetCell(const MyArray<String> &tokens);
    void handleAddRow(const MyArray<String> &tokens);
    void handleExit();
```

Връзки между класовете

- **Композиция** с Table - съдържа и управлява таблица
- **Асоциация** с всички други класове - координира тяхната работа

Прилагане на принципи на ООП

- **Капсулация**: Скрива детайлите на потребителския интерфейс

- **Единична отговорност**: Отговаря само за управлението на приложението
- **Командно управление**: Обработва команди чрез специализирани методи

3.2.7 Енумерация DataType

Предназначение

Дефинира възможните типове данни, които могат да се съхраняват в клетките на таблицата.

```
enum class DataType {
    TEXT,                   // Обикновен текст
    NUMBER,                 // Числова стойност
    CURRENCY,               // Парична сума
    EGN,                    // Единен граждански номер
    FACULTY_NUMBER          // Факултетен номер
};
```

Връзки с класовете

- Използва се в Cell за определяне на типа данни
- Използва се в Column за определяне на типа на цялата колона
- Използва се от utility функциите за автоматично разпознаване на типове

Архитектурата следва принципите на обектно-ориентиран дизайн с разделение на отговорностите, капсулация и гъвкавост чрез използването на композиция вместо наследяване.

4. Управление на паметта и грешки

Проектът използва динамично заделяне на памет в няколко ключови компонента:

String клас

- **Динамично заделяне**: Използва char* масив за съхранение на символите
- **Автоматично преоразмеряване**: При необходимост капацитетът се удвоява
- **RAII принцип**: Деструкторът автоматично освобождава паметта с delete[]
- **Copy constructor** и **assignment operator**: Осигуряват правилно копиране без споделяне на памет

MyArray клас

- **Темплейтен дизайн**: Работи с произволни типове данни
- **Експоненциален растеж**: Капацитетът се удвоява при нужда
- **Защита от изтичане**: Деструкторът винаги освобождава заделената памет
- **Exception safety**: При грешка в конструктора паметта се освобождава правилно

Table клас

- **Композитна структура**: Съдържа MyArray от Column обекти
- **Backup механизъм**: Запазва копие на данните за undo функционалност
- **Автоматично управление**: Всички обекти се управляват автоматично чрез RAII

Валидация на данни

- **Предварителна проверка**: Всички функции проверяват входните параметри преди обработка
- **Безопасни стойности по подразбиране**: При грешка се използват разумни default стойности
- **Graceful degradation**: При проблем с част от данните, останалите остават достъпни

Този подход осигурява стабилност на програмата дори при неочаквани ситуации, като позволява на потребителя да продължи работа след грешка.

5. Идеи за бъдещи подобрения

5.1 Функционални подобрения

Разширяване на възможностите за филтриране:

- Сложни филтри: Възможност за комбиниране на множество условия с логически оператори (AND, OR, NOT)
- Запазване на филтри: Възможност за запазване и преизползване на често използвани филтри

5.2 Потребителски интерфейс

- **Desktop приложение с GUI**
- **Web интерфейс**: HTML/CSS/JavaScript frontend с REST API backend
- **Real-time визуализация**: Графики и диаграми за данните

Подобрен CLI

- **История на команди**: Запазване и преглед на изпълнени команди

5.3 Качество и надеждност

Тестване

- **Unit тестове**: Пълно покритие на всички класове и функции
- **Integration тестове**: Тестване на взаимодействията между компонентите
- **Performance тестове**: Бенчмаркове за оптимизация

Обработка на грешки

- **Logging система**: Детайлно логване за debugging
- **Recovery механизми**: Възстановяване от частични грешки

5.4 Интеграция и разширяемост

- **Multi-language поддръжка**: Поддръжка на различни езици за интерфейса

Тези подобрения биха направили проекта значително по-мощен, гъвкав и пригоден за производствена среда.

6. Използване на чужд код

В проекта е използвана адаптирана версия на QuickSort алгоритъм от следния източник:

StackOverflow: C++ Quick Sort Algorithm

Кодът е модифициран, за да работи със структури от тип MyArray<size_t>, и е допълнен с логика за сравнение по стойности в конкретна колона от таблица.

Всички използвани фрагменти от външния източник са адаптирани спрямо нуждите на проекта, като е запазена основната идея на алгоритъма.