Софийски университет “Св. Климент Охридски”

Факултет по математика и информатика

Дисциплина: Структури от данни

Проект No 6 – “Електронна таблица”

(техническа документация)

Автор: Петя Личева, 3MI0700022

Проверяващ: доц. д-р Петър Армянов

София, януари 2026

СЪДЪРЖАНИЕ

Описание на заданието

Да се реализира система за електронна таблица, в която клетките могат да съдържат аритметични и логически изрази с абсолютни и релативни адреси към други клетки. Системата трябва да поддържа преизчисляване на стойности, функции върху области (sum, count, min, max, avg), работа с празни клетки, както и команди за задаване, отпечатване, запис и зареждане на таблицата във формат CSV, като обработва коректно всички възможни грешки и работи ефективно с големи таблици.

Архитектурна идея и основни структури

Задачата е разделена на няколко логически компонента, които отговарят пряко на изискванията за електронна таблица с изрази, абсолютни и релативни адреси, функции върху области и ефективна работа с големи, но рядко запълнени таблици.

Основната цел е **да не се заделя памет за празни клетки**, като същевременно се поддържа коректно преизчисляване на стойности при нужда.

Координати (Coordinates)

Coordinates представлява структура, която описва адреса на клетка в таблицата чрез двойка (ред, колона).

- Използват се 64-битови цели числа (int64_t), което позволява адресиране на клетки като R100000C250000, както е изискано в условието.

- Структурата е проектирана така, че да може да бъде използвана като **ключ в std::unordered_map**, чрез дефиниран оператор == и външен хеш.

Тази абстракция съответства директно на адресите от вида RxCy, използвани в изразите.

Хеш (Hash)

Hash е хеш-функтор за Coordinates, предназначен за използване със std::unordered_map.

- Използва се алгоритъмът **SplitMix64** на Sebastiano Vigna, който осигурява добра дисперсия и устойчивост срещу колизии.
- Добавя се фиксирана случайна стойност (FIXED_RANDOM), базирана на текущото време, за допълнителна защита срещу злонамерени входове.
- Редът и колоната се хешират поотделно и се комбинират.

Този подход е в синхрон с изискването таблицата да бъде **ефективна и сигурна** дори при големи и разредени данни.

Област (Area)

Area описва правоъгълна област в таблицата, дефинирана от два адреса:

- начален (from)
- краен (to)

Редът на координатите не е задължително нормализиран, затова структурата предоставя помощни методи:

- minRow(), maxRow()
- minCol(), maxCol()

Тази структура се използва при:

- командите PRINT VAL адрес:адрес
- PRINT EXPR адрес:адрес
- функциите sum, count, min, max, avg

Клетка (Cell)

Cell представя една клетка от електронната таблица.

В съответствие с условието, клетката съдържа:

- **израз** (аритметичен или логически, като низ);
- **кеширана числова стойност** (double), която се преизчислява само при нужда;
- **координати**, които указват позицията ѝ в таблицата.

Празна клетка е такава, която **не присъства** в таблицата (т.е. липсва в unordered_map).

Вид токен (TokenType)

TokenType е изброим тип, който описва всички възможни лексикални елементи в изразите:

- аритметични оператори (+, -, *, /, %);
- логически и сравнителни оператори (==, !=, <, >);
- скоби и разделители;
- числови константи;
- идентификатори (функции като if, sum, and, or, not);
- адреси на клетки (абсолютни и релативни).

Това покрива пълния синтаксис, описан в условието.

Токен (Token)

Token е структура, която представя един лексикален елемент от израза като:

- тип (TokenType);
- лексема (оригиналния текст от входния низ).

Тази абстракция се използва от токенайзера и парсъра.

Състояние на оценката (EvalState)

EvalState е струйтура, която съхранява данни за това дали даден аклетка се посещава в момента (и след малко ще бъде започнато нейното изчисляване) или вече стойността, записана в нейният израз е вече изчислена.

Резултат (Result)

Result е структура, която съхранява данни за това дали дадена клетка е била успешно парсната и в случай, че съхранява и нейната изчислена стойност, когато я получи.

Токенайзер (Tokenizer)

Tokenizer отговаря за лексикалния анализ на изразите.

- Разбива входния низ на последователност от токени;
- Разпознава числа, оператори, идентификатори и адреси на клетки;
- Игнорира празни символи;
- Поддържа синтаксиса за абсолютни и релативни адреси (R5C3, R[-1]C[0]).

При некоректен синтаксис се хвърля изключение, както е изискано в условието.

Таблица (Table)

Table е основната структура, която моделира електронната таблица.

- Представена е като std::unordered_map<Coordinates, Cell, Hash>;
- Съхранява **само непразните клетки**, което я прави подходяща за много големи и разредени таблици;
- Предоставя операции за:
    - задаване на израз (SET);
    - достъп до израз и кеширана стойност;
    - изчисляване на функции върху области (sum, count, min, max, avg);
    - намиране на реалните граници на таблицата при печат.

Всички числови стойности в таблицата са от тип double, което съответства на зададеното в условието.

Парсър на изрази (ExpressionParser)

ExpressionParser реализира рекурсивен десцендентен парсър за изразите в клетките.

Поддържа:

- приоритет и асоциативност на операторите (като в C++);
- логически операции (and, or, not);
- условния оператор if, с изчисляване **само на необходимите два аргумента**;
- функции върху области (sum, count, min, max, avg);
- абсолютни и релативни адреси спрямо текущата клетка.

Парсърът работи в контекста на конкретна таблица и клетка, което позволява коректно разрешаване на референциите.

Команден интерпретатор (CmdInterpreter)

CmdInterpreter осъществява връзката между потребителя и таблицата.

Той реализира логиката на всички команди от условието:

- SET
- PRINT VAL
- PRINT EXPR
- PRINT VAL ALL
- PRINT EXPR ALL
- SAVE
- LOAD

Класът:

- валидира входа;
- извиква съответните методи на Table и ExpressionParser;
- осигурява коректна и разбираема обратна връзка към потребителя при грешки.

Описание на използваните алгоритми

При реализацията на електронната таблица са използвани няколко основни алгоритмични подхода, които осигуряват коректност, ефективност и съответствие с изискванията на задачата.

1. Разредено съхранение на данни (Sparse representation)

Таблицата е реализирана като std::unordered_map<Coordinates, Cell, Hash>, в която се съхраняват **само непразните клетки**.

Алгоритмична идея:

- При достъп до клетка първо се проверява дали съществува ключ в хеш-таблицата.
- Ако клетката е празна, тя не се създава и не заема памет.
- По този начин операциите SET, GET и изчисленията върху области работят ефективно дори при много големи индекси на редове и колони.

Предимства:

- Времева сложност: средно **O(1)** за достъп до клетка.

- Паметна ефективност при редки таблици, каквито са изискани в условието.

2. Хеширане на координати

За адресиране на клетките се използва хеш-функция, базирана на алгоритъма **SplitMix64**.

Алгоритмична идея:

- Редът и колоната се хешират независимо.
- Получените хешове се комбинират чрез побитови операции.
- Използва се фиксирана случайна стойност, инициализирана по време на изпълнение, за защита срещу колизионни атаки.

Предимства:

- Добро разпределение на ключовете.
- Минимизиране на вероятността от колизии при голям брой клетки.
- Стабилна работа при екстремни координати.

3. Лексикален анализ (Tokenization)

Изразите в клетките се обработват първо чрез **лексикален анализ**, реализиран в класа Tokenizer.

Алгоритмична идея:

- Входният низ се обхожда линейно отляво надясно.
- Символите се групират в токени: числа, оператори, идентификатори, адреси на клетки.
- Празните символи се игнорират.

Сложност:

- Времева сложност: **O(n)**, където n е дължината на израза.

4. Синтактичен анализ и оценка на изрази (Recursive descent parsing)

Оценяването на изразите се извършва чрез **рекурсивен десцендентен парсър**.

Алгоритмична идея:

- Граматиката е разделена на нива според приоритета на операторите.
- Всеки нетерминал се реализира като отделен метод (parseExpression, parseTerm, parseFactor и др.).
- Оценяването се извършва едновременно с парсването (on-the-fly evaluation).

Поддържани конструкции:

- Аритметични операции
- Логически операции (and, or, not)
- Сравнения (==, !=, <, >)
- Условен израз if
- Функции върху области

Сложност:

- Времева сложност: **O(n)** за един израз при липса на циклични зависимости.

5. Разрешаване на адреси (Absolute и Relative addressing)

Адресите на клетки могат да бъдат абсолютни или релативни.

Алгоритмична идея:

- При парсване се разпознава дали координатата е в квадратни скоби.
- Абсолютните адреси се използват директно.
- Релативните адреси се изчисляват като отместване спрямо текущата клетка.

Това поведение напълно съответства на условието за адресиране в задачата.

6. Изчисляване на функции върху области

Функциите sum, count, min, max и avg работят върху правоъгълни области от клетки.

Алгоритмична идея:

- Областта се нормализира чрез минимални и максимални координати.
- Обхождат се само клетките, които реално съществуват в таблицата.
- Празните клетки не участват в изчисленията.

Сложност:

- В най-лошия случай: **O(k)**, където k е броят на непразните клетки в областта.

- Не зависи от абсолютния размер на областта, а от реално съхранените данни.

7. Кеширане на изчислени стойности

Всяка клетка съдържа кеширана стойност на израза си.

Алгоритмична идея:

- При първо изчисляване стойността се записва в клетката.
- При следващи заявки стойността се използва директно.
- Преизчисляване се извършва само при промяна на израза.

Предимства:

- Намалява броя на повторните изчисления.
- Подобрява производителността при чести PRINT операции.

8. Входно-изходни операции (CSV сериализация)

Записът и зареждането на таблицата се извършват във формат .csv.

Алгоритмична идея:

- Таблицата се обхожда по редове и колони.
- На всеки ред от файла се записват изразите, разделени със ;.
- При зареждане всеки ред се парсва и съответните клетки се създават.

Описание на структурата на проекта

Проектът е организиран като C++ приложение с разделна компилация и се управлява чрез конфигурационен файл **CMakeLists.txt**, който описва зависимостите, целите за компилация и настройките на проекта.

Кодът и ресурсите са разпределени в следните основни директории:

- **src/**
  Съдържа изходния код на проекта – както header (.h) файловете, така и съответните имплементации (.cpp).
  Тук са реализирани основните логически компоненти на системата: електронната таблица, парсването и оценката на изрази, както и интерпретацията на потребителските команди.

- **public/**
  Съдържа външни ресурси, използвани за тестване и демонстрация на функционалността на проекта.
  В тази директория са разположени:
  - демонстрационни скриптове, които илюстрират типични сценарии на работа със системата и могат да бъдат използвани при защита на проекта.
- **tests/**
  Съдържа тестови сценарии и помощни файлове за проверка на коректността на реализацията.
  Тестовете покриват основните функционалности на системата – работа с клетки, изчисляване на изрази, функции върху области, обработка на грешки и коректна работа с файлове.

Тази структура осигурява ясна разделеност между логиката на приложението, тестовите данни и сценарии, както и улеснява поддръжката и разширяването на проекта.

Описание на тестовите данни

Тестовите данни в проекта са подбрани така, че да покриват максимално пълно функционалните изисквания, описани в условието на задачата, както и гранични и нетипични случаи.

1. Данни за клетки и изрази

Използват се тестови данни, които включват:

- клетки с константни числови стойности;
- клетки с аритметични изрази с различна сложност;
- изрази с вложени скоби и комбинирани оператори;
- логически изрази и сравнения;
- условни изрази с функцията if.

2. Адресиране на клетки

Тестовите сценарии покриват:

- абсолютни адреси (R5C3);
- релативни адреси (R[-1]C[0]);

- комбинации от абсолютни и релативни координати;
- адресиране на клетки с много големи индекси за ред и колона.

3. Функции върху области

В тестовите данни са включени сценарии за всички поддържани функции:

- sum, count, min, max, avg;
- области с една клетка;
- празни области;
- области, съдържащи само празни клетки;
- области с частично запълнени клетки.

4. Таблици с големи размери

За да се провери ефективността на реализацията, са включени тестове с:

- много големи координати на клетки;
- малък брой реално съществуващи клетки (разредена таблица);
- чести операции за достъп и печат.

5. Входно-изходни тестове

Използват се тестови CSV файлове за проверка на:

- коректен запис на таблицата във файл;
- коректно зареждане на таблицата от файл;
- запазване на формулите, а не само на изчислените стойности.

6. Демонстрационни скриптове

Демонстрационните скриптове съдържат последователности от команди:

- SET
- PRINT VAL
- PRINT EXPR
- SAVE
- LOAD

Те са предназначени да покажат типичен работен поток на системата и да демонстрират коректното ѝ поведение при реална употреба, включително при възникване на грешки.

Описание на тестовете към проекта

За валидиране на коректността и стабилността на реализацията са разработени автоматизирани тестове с помощта на библиотеката **Catch2**. Тестовете са организирани по функционални модули и покриват основните компоненти на системата: токенизация, парсване и оценка на изрази, работа с таблицата и входно-изходни операции.

1. Тестове за токенайзер (Tokenizer)

Тестовете за токенайзера проверяват коректното разпознаване и класифициране на лексикалните елементи в изразите.

Покриват се следните случаи:

- разпознаване на числови литерали;
- разпознаване на абсолютни и релативни адреси на клетки (R5C3, R[-1]C[0]);
- разпознаване на идентификатори, включително имена на функции и логически оператори (sum, avg, if, and, or, not);
- коректна токенизация на аритметични, логически и сравнителни оператори;
- правилно детектиране на края на входния низ.

Целта на тези тестове е да гарантират, че входните изрази се разбиват коректно на токени, което е критично за правилната работа на парсъра.

2. Тестове за парсър и оценка на изрази (ExpressionParser)

Тестовете за парсъра проверяват правилното изчисляване на стойности на изрази в контекста на таблица.

Те включват:

- проверка на приоритет и асоциативност на аритметичните оператори;
- коректна обработка на скоби;
- логически операции (and, or, not);
- условни изрази с функцията if;
- достъп до стойности на други клетки чрез абсолютни и релативни адреси.

Тези тестове удостоверяват, че семантиката на изразите съвпада с описаната в условието и с тази на езика C++.

3. Тестове за адресиране на клетки

Част от тестовете са фокусирани върху работата с клетъчни адреси:

- достъп до клетки чрез абсолютни адреси (R0C0);
- използване на релативни адреси спрямо текущата клетка (R[0]C[1]);
- комбиниране на няколко адреса в един израз.

Тези тестове гарантират коректното изчисляване на координатите и правилното разрешаване на референциите.

4. Тестове за таблицата (Table)

Тестовете за класа Table проверяват основната функционалност на електронната таблица.

Покрити са следните сценарии:

- създаване на празна таблица;
- добавяне и извличане на клетки чрез командата set;
- работа с клетки на произволни координати;
- базово поведение при обновяване на съдържанието на клетка.

Допълнително се проверяват и агрегационните функции:

- sum
- count
- min
- max
- avg

като се използват предварително зададени кеширани стойности, за да се изолира логиката на самите функции от парсването на изрази.

5. Тестове за агрегиращи функции върху области

Агрегиращите функции се тестват върху правоъгълни области от клетки.

Проверките включват:

- коректно изчисляване на сума, минимум, максимум и средна стойност;
- броене на непразни клетки;
- работа с частично запълнени области.

Тези тестове потвърждават, че функциите работят само върху реално съществуващите клетки, в съответствие със спецификацията на задачата.

6. Тестове за запис и зареждане от файл (SAVE / LOAD)

Реализирани са тестове за входно-изходните операции върху таблицата:

- запис на таблица във CSV файл чрез командата SAVE;
- зареждане на таблица от CSV файл чрез командата LOAD;
- проверка за запазване на формулите в клетките, а не само на изчислените стойности.

След приключване на тестовете временните файлове се изтриват, за да не остават странични ефекти.

7. Обобщение на тестовото покритие

Разработените тестове:

- покриват всички ключови компоненти на системата;
- проверяват както стандартни, така и гранични случаи;
- осигуряват увереност в коректността на реализацията и устойчивостта ѝ при различни входни данни.

Тестовата инфраструктура позволява лесно добавяне на нови тестове при разширяване на функционалността на проекта.

Използвани библиотеки, технологии и външни зависимости

При реализацията на проекта са използвани стандартни и външни инструменти и библиотеки, които осигуряват ефективна разработка, лесна поддръжка и автоматизирано тестване.

1. C++ (ISO C++17)

Проектът е реализиран на езика **C++**, като се използват възможностите на стандарта **C++17**.

Основни характеристики:

- използване на стойностна семантика;
- автоматично управление на паметта чрез обхват (RAII);
- липса на ръчно заделяне и освобождаване на динамична памет.

Изборът на C++17 осигурява добра производителност и гъвкавост при работа с големи и разредени таблици.

2. Стандартна библиотека на C++ (STL)

Проектът разчита изцяло на стандартната библиотека на C++ за реализацията на основната функционалност.

Използвани са:

- std::unordered_map – за съхранение на клетките на таблицата, което позволява ефективна работа с големи и рядко запълнени структури;
- std::string – за представяне на изрази, команди и файлови имена;
- std::vector – за съхранение на токени и аргументи при парсването на изрази;
- стандартни потоци за вход и изход (<fstream>, <iostream>).

В проекта **не се използват умни указатели** (std::unique_ptr, std::shared_ptr). Управлението на паметта е реализирано чрез стандартни контейнери и стойностна семантика, което опростява кода и намалява риска от грешки.

3. CMake

CMake се използва като система за изграждане и управление на проекта.

Чрез CMake са реализирани:

- конфигурация на компилационния процес;
- дефиниране на изпълними файлове и тестови цели;
- лесна интеграция на външни библиотеки (Catch2);
- преносимост между различни среди и компилатори.

4. Catch2

За автоматизирано тестване е използвана библиотеката **Catch2**.

Catch2 предоставя:

- декларативен и четим синтаксис за unit тестове;

- възможност за групиране и маркиране на тестове чрез тагове;
- автоматично откриване и изпълнение на тестовете.

С помощта на Catch2 са реализирани тестове за:

- токенизация на входните изрази;
- парсване и изчисляване на аритметични и логически изрази;
- работа с клетъчни адреси (абсолютни и релативни);
- коректност на операциите върху таблицата;
- запис и зареждане на данни от файлове.

5. Файлови формати

Проектът използва текстови файлови формати за вход и изход:

- **CSV (Comma-Separated Values)**
  Използва се за запис и зареждане на съдържанието на таблицата. Във файла всяки ред отговаря на ред от таблицата, а клетките са разделени със символ ;

6. Инструменти за разработка

Проектът може да бъде компилиран с всеки стандартен компилатор, поддържащ C++17, като например:

- g++
- clang++

Използваните технологии и библиотеки са широко разпространени и не изискват специализирана среда за разработка.

Ресурси

При разработката на проекта са използвани външни ресурси с цел по-добро разбиране и коректна реализация на използваните алгоритми и структури от данни.

Статия за хеширането

Използвана е статията „*Custom Hash Functions*“ от платформата Codeforces, която разглежда проблеми, свързани с хеш колизии, както и техники за създаване на по-надеждни хеш функции при работа с хеш таблици. Материалът е използван като теоретична основа за избора и разбирането на подходящи стратегии за хеширане при работа с асоциативни контейнери.

Източник:
https://codeforces.com/blog/entry/62393

Лиценз и алгоритъм на Себастиано Вигна

Използван е алгоритъмът **SplitMix64**, разработен от Себастиано Вигна, който представлява бърз и качествен генератор на псевдослучайни числа, подходящ за хеширане. Алгоритъмът е използван съгласно условията на предоставения лиценз, публикуван от автора.

Източник (оригинален код и лиценз):
https://xoshiro.di.unimi.it/splitmix64.c