Házi feladat

Programozás alapjai 2

Terv

Mészáros Anna Veronika (I8SQUE)

Specifikáció

A feladat

A program egy számolótáblát (spreadsheet) fog megvalósítani. A számolótáblát egy N×M-es négyzetrács fogja reprezentálni, minden rácspontja pedig vagy egy számot, vagy egy, akár más cellákra való hivatkozást tartalmazó, aritmetikai műveletet fog tartalmazni. A cellákat egy konzolos felületen keresztül lehet majd módosítani, illetve tartalmukat a konzolra vagy egy fájlba kiírni.

Megvalósított műveletek

- egyszerű aritmetikai műveletek: összeadás, kivonás, szorzás, osztás (valós számokkal)
- más cellákra vett abszolút vagy relatív hivatkozás (és ezek felhasználása kifejezésekben)
  - a program a kifejezés kiértékelésekor figyelembe veszi, hogy ne legyen körkörös hivatkozás, illetve hogy a hivatkozások ne “lógjanak ki” a táblázatból
- tartományon végrehajtható függvények: pl. tartomány összegzése, átlaga
  - a tartományok mindenképp téglalap alakúak és két átellenes cellájukkal kell megadni őket
  - a program ebben az esetben is vizsgálja a körkörös hivatkozásokat

A konzolos felület és funkciók

A konzolos felületen alapvető parancsok kiadásával lehet majd manipulálni a számolótábla állapotát. Minden parancs egy egyszerű kulcsszóból, illetve pár paraméterből áll. A paramétereket szóközzel választjuk el, így a semelyik paraméter nem tartalmazhat szóközt. Cellák paraméterben átadásakor az oszlopot betűvel (a-tól), a sort pedig számmal (1-től) indexeljük (pl. A1, bc23). A kifejezéseket az ismert táblázatkezelők által használt formátumhoz hasonlóan kell megadni (pl. sum(b2:c3)+a1*2). Amennyiben a felhasználó valamilyen szempontból hibás parancsot ad ki, a program ezt egy megfelelő, a konzolra kiírt, hibaüzenettel jelzi.

A konzolos felületen az alábbi parancsok lesznek elérhetőek:

- new [egész] [egész] - létrehoz egy új N×M-es táblát és 0-val inicializálja (ha volt előző tábla, azt eldobja)
- resize [egész] [egész] - a jelenlegi táblát átméretezi (ha mérete csökken, az adatok elvesznek, ha nő, akkor az új cellák értéke 0 lesz)
- print - a számolótáblában található értékek listázása a konzolra
- set [cella] [kifejezés] - egy adott cellának az értékét beállítja a paraméterként kapott kifejezésre
- pull [cella] [cella] - az adott tartományban “lehúzza” az első paraméterként kapott cella tartalmát, azaz lemásolja, és a benne található relatív hivatkozásokat a cella kezdőcellától vett relatív pozíciójának megfelelően eltolja
- show [cella] - kiírja az adott cellában található kifejezést, és annak értékét is
- export [fájlnév] - a számolótáblában található értékeket kiírja egy fájlba
- save [fájlnév] - a számolótáblában található kifejezéseket kiírja egy fájlba
- load [fájlnév] - az adott fájlból beolvassa és létrehozza a számolótáblát (ha a fájlt az export utasítással írtuk ki a fájlba, akkor csak az értékeket tudja visszatölteni)

Terv

Kifejezések leírása - Expression osztály és leszármazottai

A programban minden kifejezés az Expression absztrakt alaposztályból származik. Így minden kifejezésnek megvannak (többek között) az alábbi alapvető metódusai:

- eval - kifejezés kiértékelése, const char* kivételt dob ha valami probléma lép fel
- checkCyclic - megnézi hogy a kifejezésben található hivatkozások között van-e körkörös, azaz hivatkozik-e a kapott listának bármely elemére
- safeEval - kiértékeli a kifejezést, úgy, hogy figyel a körkörös hivatkozásokra
- show - kiírja a kifejezést, mint képletet

Az Expression alaposztályból 4 további osztály származtatik: a számok, a cellahivatkozások tárolására, illetve a függvények, valamint az aritmetikai műveletek leírására szolgáló osztályok.

Range osztály

A függvények egy tartományon belül értékelődnek ki, ennek a tartománynak a reprezentációját a Range osztály valósítja meg. Az osztály legfontosabb tulajdonsága, hogy rendelkezik egy iterátorral, aminek segítségével a tartomány sorfolytonosan bejárható.

ExprPointer osztály

A kifejezéseket a gyakorlatban pointerként tároljuk el, ezen pointerek kezelésének megkönnyítésének érdekében létrehozunk egy ExprPointer osztály, ami dinamikus memóriaterületen található kifejezést kezel, destruktorában pedig felszabadítja őket.

A táblázat - Sheet osztály

A Sheet osztály egy N×M méretű dinamikus memóriaterületen sorfolytonosan tárolja el az adott cellában lévő kifejezés értékét az ExprPointer osztály pédényaiként. A publikus függvényei kezelik a betű+szám indexelési formáról a 0-tól kezdődő, kizárólag számokkal való indexelésre áttérést. Ezen kívül többek között képes a benne lévő kifejezések kiértékelésére és kiírására.

A felhasználói felület - Console osztály

A Console osztálynak megadhatunk egy input- és egy outputstream-et, ahonnan a parancsokat fogja beolvasni, illetve a parancsok eredményét ki fogja írni. Az osztály mindig egy darab táblázatot tartalmaz, a kapott utasításokat ezen hajtja végre. A parancsok kulcsszavait a readCommand tagfüggvény értelmezi, és ő hívja meg a megfelelő parancsot lekezelő tagfüggvényt. Az adott tagfüggvény az inputstreamről beolvassa a parancs paramétereit és végrehajtja a azt.

Kifejezések értelmezése - A Parser osztály

A parser osztályt egy std::string-el inicializáljuk, amit a Parser konstruktora az alábbi tokenekre bont: MINUS, PLUS, SLASH, STAR, LEFT_BR, RIGHT_BR, COLON, DOLLAR, NUMBER, STRING. Ezután a Parser a parse tagfüggvényének segítségével megpróbálja értelmezni az adott kifejezést, és amennyiben ez lehetséges (azaz a kifejezés szintaktikailag helyes), létre is hozza a kifejezést dinamikus memóriaterületen, különben const char* típusú kivételt dob. A kifejezés értelmezése az alábbi szabályok alapján zajlik:

```
expression →  factor  ( ( "-" | "+" ) factor )* ;
factor     →  unary  ( ( "/" | "*" ) unary )* ;
unary      →  "-" unary  |  function  |  primary;
function   →  STRING  "(" cell ":" cell ")" ;
cell       →  ('$')?  STRING  ('$' NUMBER)?;
primary    →  NUMBER  |  "(" expression ")"  |  cell;
```

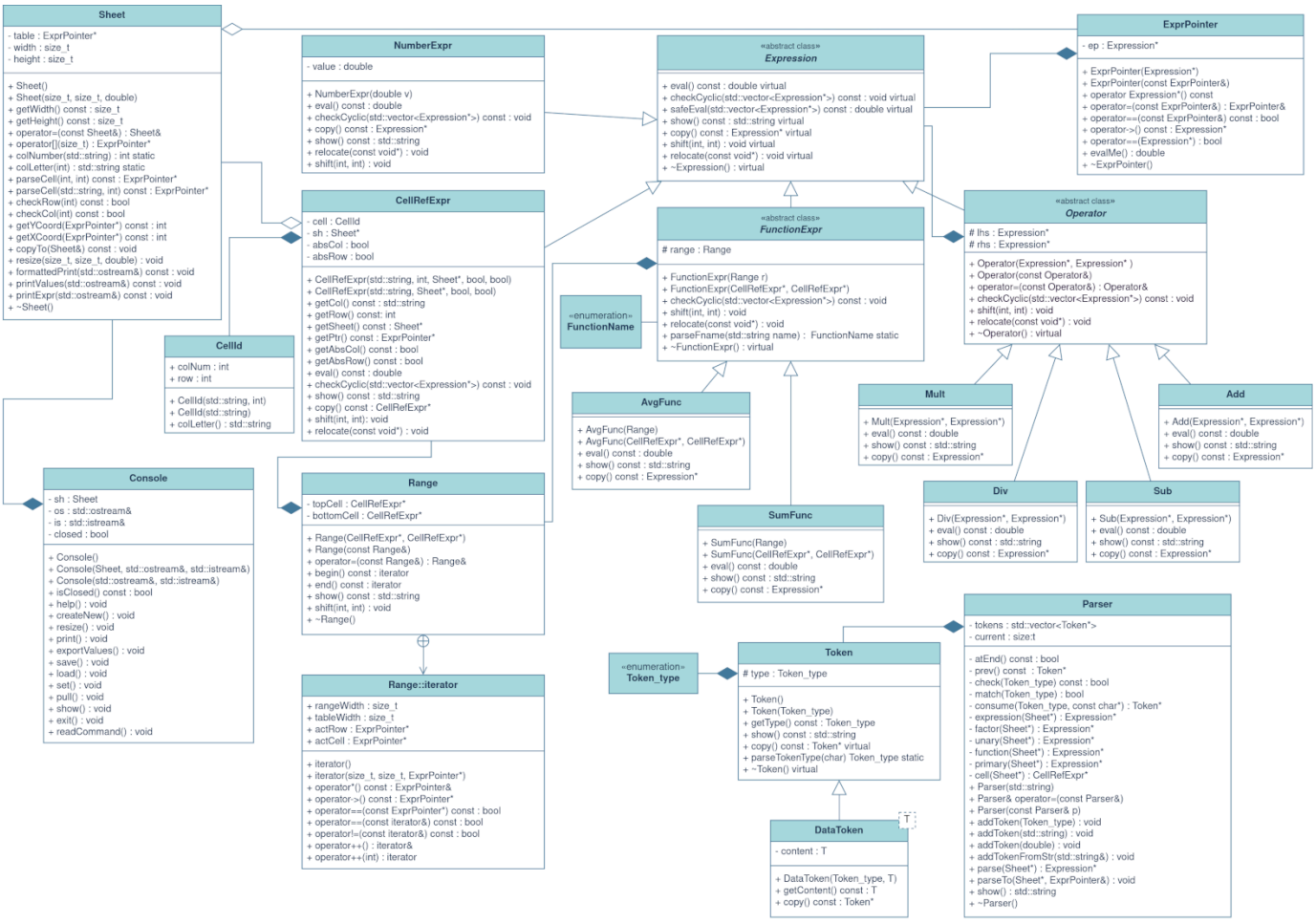
Minden ilyen fent leírt szabályhoz tartozik egy-egy tagfüggvény, amelyeknek feladata, hogy a tokenlistának éppen aktív (current) tokenjétől kezdve megpróbáljon értelmezni egy megfelelő típusú kifejezést, ha ez lehetséges, akkor az ehhez szükséges tokeneket “elfogyasztja”, így a következő ilyen tagfüggvény azokat a tokeneket már nem fogja tudni felhasználni. Így láthatjuk, hogy a parse függvény igazából csak annyit tesz, hogy az első tokent állítja be aktív tokennek (current=0) és meghívja az expression-t értelmező függvényt.
A kifejezések értelmezésének pontos menetének megvalósításához a program Robert Nystrom: Crafting Interpreters c. könyvéből merít ihletet, amely az alábbi linken elérhető: https://craftinginterpreters.com.

Tokenek

A tokeneket egy külön osztály reprezentálja, a Token alaposztály példányai azok a tokenek, amelyeknek a típusán kívül nincs más jelentéstartalmuk (pl. szorzás, $). Az olyan tokeneknek, melyeknek lehet a típusukon kívül más tartalma a DataToken osztály példányai - ebben a programban ilyen tokenek a STRING és a NUMBER tokenek, ezeket DataToken<std::string> és DataToken<double> osztályok reprezentálják.

Osztálydiagram

Az eddig leírt terv UML diagramja:



Terv későbbi módosításai

Hibakezelés

A const char* típusú hibák helyett két egyedi hibaosztály van származtatva az std::runtime_error osztályból. Ezek az alábbiak:

- eval_error - kifejezések kiértékelésekor léphet fel pl. körkörös hivatkozás vagy indexelési hibák miatt
- syntax_error - szintaktikailag helytelen kifejezések értelmezésekor léphet fel pl. nics elég operandusa egy műveletnek vagy érvénytelen függvénynevet adott meg a felhasználó

Teljesen globális függvények kiiktatása

A globális scope-ban található függvényeket a megfelelő osztályba vannak áthelyezve, mint static függvények.

Minden adattag privát

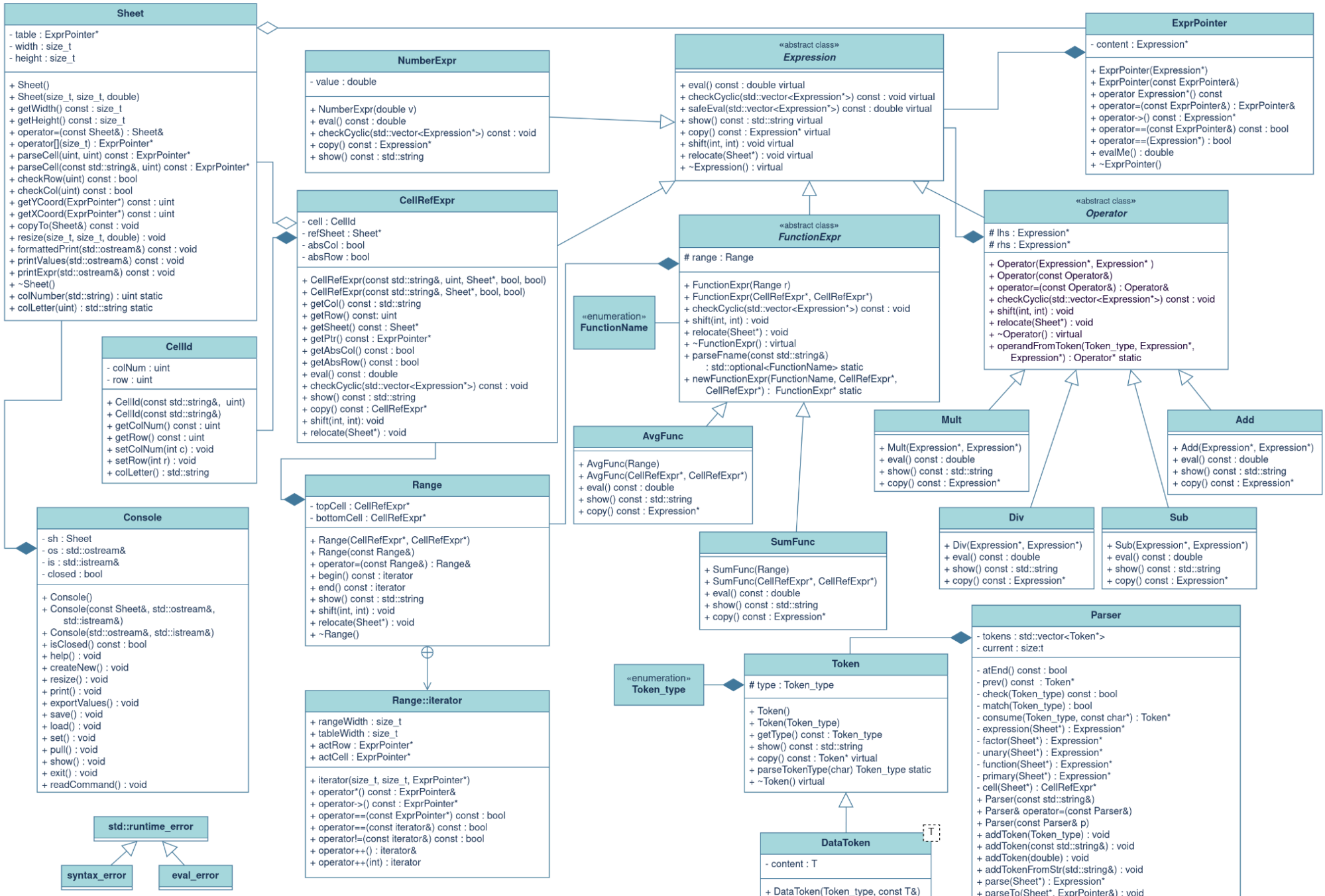
Korábban publikus adattagok helyett privát adattagok, getter és setter függvények lettek bevezetve.

Terv véglegesítése

- egyértelműbb változó- és adattag nevek
- konstans referenciák használata paraméterbeli átadás helyett
- nemnegatív számok reprezentálására unsigned int használata int helyett
- felesleges iterator konstruktor eltávolítása
- std::optional használata INVALID függvénynév helyett

Frissített osztálydiagram

A következő oldalon található a végleges osztálydiagram:


Sheet
- table : ExprPointer*
- width : size_t
- height : size_t
+ Sheet()
+ Sheet(size_t, size_t, double)
+ getWidth() const : size_t
+ getHeight() const : size_t
+ operator=(const Sheet&) : Sheet&
+ operator[](size_t) : ExprPointer*
+ parseCell(uint, uint) const : ExprPointer*
+ parseCell(const std::string&, uint) const : ExprPointer*
+ checkRow(uint) const : bool
+ checkCol(uint) const : bool
+ getYCoord(ExprPointer*) const : uint
+ getXCoord(ExprPointer*) const : uint
+ copyTo(Sheet&) const : void
+ resize(size_t, size_t, double) : void
+ formattedPrint(std::ostream&) const : void
+ printValues(std::ostream&) const : void
+ printExpr(std::ostream&) const : void
+ ~Sheet()
+ colNumber(std::string) : uint static
+ colLetter(uint) : std::string static
CellId
- colNum : uint
- row : uint
+ CellId(const std::string&, uint)
+ CellId(const std::string&)
+ getColNum() const : uint
+ getRow() const : uint
+ setColNum(int c) : void
+ setRow(int r) : void
+ colLetter() : std::string
Console
- sh : Sheet
- os : std::ostream&
- is : std::istream&
- closed : bool
+ Console()
+ Console(const Sheet&, std::ostream&, std::istream&)
+ Console(std::ostream&, std::istream&)
+ isClosed() const : bool
+ help() : void
+ createNew() : void
+ resize() : void
+ print() : void
+ exportValues() : void
+ save() : void
+ load() : void
+ set() : void
+ pull() : void
+ show() : void
+ exit() : void
+ readCommand() : void
std::runtime_error
syntax_error
eval_error
NumberExpr
- value : double
+ NumberExpr(double v)
+ eval() const : double
+ checkCyclic(std::vector<Expression*>) const : void
+ copy() const : Expression*
+ show() const : std::string
CellRefExpr
- cell : CellId
- refSheet : Sheet*
- absCol : bool
- absRow : bool
+ CellRefExpr(const std::string&, uint, Sheet*, bool, bool)
+ CellRefExpr(const std::string&, Sheet*, bool, bool)
+ getCol() const : std::string
+ getRow() const: uint
+ getSheet() const : Sheet*
+ getPtr() const : ExprPointer*
+ getAbsCol() const : bool
+ getAbsRow() const : bool
+ eval() const : double
+ checkCyclic(std::vector<Expression*>) const : void
+ show() const : std::string
+ copy() const : CellRefExpr*
+ shift(int, int): void
+ relocate(Sheet*) : void
Range
- topCell : CellRefExpr*
- bottomCell : CellRefExpr*
+ Range(CellRefExpr*, CellRefExpr*)
+ Range(const Range&)
+ operator=(const Range&) : Range&
+ begin() const : iterator
+ end() const : iterator
+ show() const : std::string
+ shift(int, int) : void
+ relocate(Sheet*) : void
+ ~Range()
Range::iterator
+ rangeWidth : size_t
+ tableWidth : size_t
+ actRow : ExprPointer*
+ actCell : ExprPointer*
+ iterator(size_t, size_t, ExprPointer*)
+ operator*() const : ExprPointer&
+ operator->() const : ExprPointer*
+ operator==(const ExprPointer*) const : bool
+ operator==(const iterator&) const : bool
+ operator!=(const iterator&) const : bool
+ operator++() : iterator&
+ operator++(int) : iterator
«abstract class»
Expression
+ eval() const : double virtual
+ checkCyclic(std::vector<Expression*>) const : void virtual
+ safeEval(std::vector<Expression*>) const : double virtual
+ show() const : std::string virtual
+ copy() const : Expression* virtual
+ shift(int, int) : void virtual
+ relocate(Sheet*) : void virtual
+ ~Expression() : virtual
«abstract class»
FunctionExpr
range : Range
+ FunctionExpr(Range r)
+ FunctionExpr(CellRefExpr*, CellRefExpr*)
+ checkCyclic(std::vector<Expression*>) const : void
+ shift(int, int) : void
+ relocate(Sheet*) : void
+ ~FunctionExpr() : virtual
+ parseFname(const std::string&) : std::optional<FunctionName> static
+ newFunctionExpr(FunctionName, CellRefExpr*, CellRefExpr*) : FunctionExpr* static
«enumeration»
FunctionName
AvgFunc
+ AvgFunc(Range)
+ AvgFunc(CellRefExpr*, CellRefExpr*)
+ eval() const : double
+ show() const : std::string
+ copy() const : Expression*
SumFunc
+ SumFunc(Range)
+ SumFunc(CellRefExpr*, CellRefExpr*)
+ eval() const : double
+ show() const : std::string
+ copy() const : Expression*
ExprPointer
- content : Expression*
+ ExprPointer(Expression*)
+ ExprPointer(const ExprPointer&)
+ operator Expression*() const
+ operator=(const ExprPointer&) : ExprPointer&
+ operator->() const : Expression*
+ operator==(const ExprPointer&) const : bool
+ operator==(Expression*) : bool
+ evalMe() : double
+ ~ExprPointer()
«abstract class»
Operator
lhs : Expression*
rhs : Expression*
+ Operator(Expression*, Expression*)
+ Operator(const Operator&)
+ operator=(const Operator&) : Operator&
+ checkCyclic(std::vector<Expression*>) const : void
+ shift(int, int) : void
+ relocate(Sheet*) : void
+ ~Operator() : virtual
+ operandFromToken(Token_type, Expression*, Expression*) : Operator* static
Mult
+ Mult(Expression*, Expression*)
+ eval() const : double
+ show() const : std::string
+ copy() const : Expression*
Add
+ Add(Expression*, Expression*)
+ eval() const : double
+ show() const : std::string
+ copy() const : Expression*
Div
+ Div(Expression*, Expression*)
+ eval() const : double
+ show() const : std::string
+ copy() const : Expression*
Sub
+ Sub(Expression*, Expression*)
+ eval() const : double
+ show() const : std::string
+ copy() const : Expression*
«enumeration»
Token_type
Token
type : Token_type
+ Token()
+ Token(Token_type)
+ getType() const : Token_type
+ show() const : std::string
+ copy() const : Token* virtual
+ parseTokenType(char) Token_type static
+ ~Token() virtual
DataToken
T
- content : T
+ DataToken(Token_type, const T&)
+ getContent() const : T
+ copy() const : Token*
Parser
- tokens : std::vector<Token*>
- current : size:t
- atEnd() const : bool
- prev() const : Token*
- check(Token_type) const : bool
- match(Token_type) : bool
- consume(Token_type, const char*) : Token*
- expression(Sheet*) : Expression*
- factor(Sheet*) : Expression*
- unary(Sheet*) : Expression*
- function(Sheet*) : Expression*
- primary(Sheet*) : Expression*
- cell(Sheet*) : CellRefExpr*
+ Parser(const std::string&)
+ Parser& operator=(const Parser&)
+ Parser(const Parser& p)
+ addToken(Token_type) : void
+ addToken(const std::string&) : void
+ addToken(double) : void
+ addTokenFromStr(std::string&) : void
+ parse(Sheet*) : Expression*
+ parseTo(Sheet*, ExprPointer&) : void
+ show() : std::string
+ ~Parser()