Spreadsheet App

Generated by Doxygen 1.9.8

1 Hierarchical Index 1

1.1 Class Hierarchy . . . 1

2 Class Index 3

2.1 Class List . . . 3

3 File Index 5

3.1 File List . . . 5

4 Class Documentation 7

4.1 AnsiTerminal Class Reference . . . 7

4.1.1 Detailed Description . . . 7

4.1.2 Member Function Documentation . . . 8

4.1.2.1 getKeystroke() . . . 8

4.1.2.2 getSpecialKey() . . . 8

4.1.2.3 isArrowKey() . . . 8

4.1.2.4 printAt() . . . 8

4.1.2.5 printInvertedAt() . . . 9

4.2 Cell Class Reference . . . 9

4.2.1 Detailed Description . . . 10

4.2.2 Constructor & Destructor Documentation . . . 10

4.2.2.1 Cell() . . . 10

4.2.2.2 ~Cell() . . . 10

4.2.3 Member Function Documentation . . . 10

4.2.3.1 getCellValueAsDouble() . . . 10

4.2.3.2 getCol() . . . 11

4.2.3.3 getLetterRepresentation() . . . 11

4.2.3.4 getRow() . . . 11

4.2.3.5 getValueAsString() . . . 11

4.2.3.6 setLetterRepresentation() . . . 11

4.3 DoubleValueCell Class Reference . . . 12

4.3.1 Detailed Description . . . 13

4.3.2 Constructor & Destructor Documentation . . . 13

4.3.2.1 DoubleValueCell() . . . 13

4.3.3 Member Function Documentation . . . 14

4.3.3.1 getValue() . . . 14

4.3.3.2 getValueAsString() . . . 14

4.3.3.3 setValue() . . . 14

4.4 FileHandler Class Reference . . . 14

4.4.1 Detailed Description . . . 15

4.4.2 Member Function Documentation . . . 15

4.4.2.1 loadFromFile() . . . 15

4.4.2.2 saveToFile() . . . 15

4.5 FormulaCell Class Reference . . . 15
4.5.1 Detailed Description . . . 17
4.5.2 Constructor & Destructor Documentation . . . 17
4.5.2.1 FormulaCell() . . . 17
4.5.3 Member Function Documentation . . . 17
4.5.3.1 addDependentCell() . . . 17
4.5.3.2 clearDependentCells() . . . 17
4.5.3.3 fetchDependentCells() . . . 17
4.5.3.4 getCalculatedValue() . . . 18
4.5.3.5 getFormula() . . . 18
4.5.3.6 getValueAsString() . . . 18
4.5.3.7 setCalculatedValue() . . . 18
4.6 FormulaParser Class Reference . . . 19
4.6.1 Detailed Description . . . 19
4.6.2 Constructor & Destructor Documentation . . . 19
4.6.2.1 FormulaParser() . . . 19
4.6.3 Member Function Documentation . . . 19
4.6.3.1 autoCalculate() . . . 19
4.6.3.2 parseAndEvaluate() . . . 20
4.7 IntValueCell Class Reference . . . 20
4.7.1 Detailed Description . . . 21
4.7.2 Constructor & Destructor Documentation . . . 22
4.7.2.1 IntValueCell() . . . 22
4.7.3 Member Function Documentation . . . 23
4.7.3.1 getValue() . . . 23
4.7.3.2 getValueAsString() . . . 23
4.7.3.3 setValue() . . . 23
4.8 spc::myset< T > Class Template Reference . . . 24
4.8.1 Detailed Description . . . 24
4.8.2 Member Function Documentation . . . 24
4.8.2.1 begin() [1/2] . . . 24
4.8.2.2 begin() [2/2] . . . 25
4.8.2.3 empty() . . . 25
4.8.2.4 end() [1/2] . . . 25
4.8.2.5 end() [2/2] . . . 25
4.8.2.6 find() . . . 25
4.8.2.7 insert() . . . 26
4.8.2.8 size() . . . 26
4.9 spc::myvec< T > Class Template Reference . . . 26
4.9.1 Detailed Description . . . 27
4.9.2 Constructor & Destructor Documentation . . . 28
4.9.2.1 myvec() [1/4] . . . 28

4.9.2.2 myvec() [2/4] . . . 28
4.9.2.3 myvec() [3/4] . . . 28
4.9.2.4 myvec() [4/4] . . . 29
4.9.3 Member Function Documentation . . . 29
4.9.3.1 begin() [1/2] . . . 29
4.9.3.2 begin() [2/2] . . . 29
4.9.3.3 empty() . . . 30
4.9.3.4 end() [1/2] . . . 30
4.9.3.5 end() [2/2] . . . 30
4.9.3.6 get_capacity() . . . 30
4.9.3.7 get_size() . . . 30
4.9.3.8 operator=() [1/2] . . . 30
4.9.3.9 operator=() [2/2] . . . 31
4.9.3.10 operator[]() [1/2] . . . 31
4.9.3.11 operator[]() [2/2] . . . 31
4.9.3.12 push_back() [1/2] . . . 32
4.9.3.13 push_back() [2/2] . . . 32
4.10 SheetHandler Class Reference . . . 32
4.10.1 Detailed Description . . . 33
4.10.2 Constructor & Destructor Documentation . . . 33
4.10.2.1 SheetHandler() . . . 33
4.10.3 Member Function Documentation . . . 33
4.10.3.1 add() . . . 33
4.10.3.2 getSheet() . . . 34
4.10.3.3 loadSheet() . . . 34
4.10.3.4 saveSheet() . . . 34
4.11 Spreadsheet Class Reference . . . 35
4.11.1 Detailed Description . . . 36
4.11.2 Constructor & Destructor Documentation . . . 36
4.11.2.1 Spreadsheet() . . . 36
4.11.3 Member Function Documentation . . . 36
4.11.3.1 displayScreen() . . . 36
4.11.3.2 enterData() [1/2] . . . 36
4.11.3.3 enterData() [2/2] . . . 37
4.11.3.4 getCell() . . . 37
4.11.3.5 getCellsInRange() . . . 37
4.11.3.6 getColCount() . . . 38
4.11.3.7 getRowCount() . . . 38
4.11.3.8 setCell() . . . 38
4.12 StringValueCell Class Reference . . . 39
4.12.1 Detailed Description . . . 40
4.12.2 Constructor & Destructor Documentation . . . 40

4.12.2.1 StringValueCell() . . . 40
4.12.3 Member Function Documentation . . . 40
4.12.3.1 getValue() . . . 40
4.12.3.2 getValueAsString() . . . 41
4.12.3.3 setValue() . . . 41
4.13 ValueCell Class Reference . . . 41
4.13.1 Detailed Description . . . 42
4.13.2 Constructor & Destructor Documentation . . . 42
4.13.2.1 ValueCell() . . . 42
4.13.2.2 ~ValueCell() . . . 43
4.13.3 Member Function Documentation . . . 43
4.13.3.1 setValue() . . . 43

5 File Documentation **45**
5.1 AnsiTerminal.h . . . 45
5.2 Cell.h . . . 45
5.3 FileHandler.h . . . 47
5.4 FormulaParser.h . . . 47
5.5 myset.h . . . 48
5.6 myvec.h . . . 49
5.7 SheetHandler.h . . . 51
5.8 Spreadsheet.h . . . 51

Index **53**

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AnsiTerminal 7
Cell 9
FormulaCell 15
ValueCell 41
DoubleValueCell 12
IntValueCell 20
StringValueCell 39
FileHandler 14
FormulaParser 19
spc::myset< T > 24
spc::myvec< T > 26
spc::myvec< spc::myvec< std::unique_ptr< Cell > > > 26
spc::myvec< std::pair< int, int > > 26
SheetHandler 32
Spreadsheet 35

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AnsiTerminal
: Provides an interface for ANSI terminal manipulation, including text printing, screen clearing, and key detection . . . 7

Cell . . . 9

DoubleValueCell . . . 12

FileHandler
: A utility class for handling file operations for a Spreadsheet . . . 14

FormulaCell . . . 15

FormulaParser
: Responsible for parsing and evaluating formulas in the spreadsheet . . . 19

IntValueCell . . . 20

spc::myset< T >
: A custom implementation of a set container that prevents duplicate elements . . . 24

spc::myvec< T >
: A custom dynamic array implementation similar to std::vector . . . 26

SheetHandler
: A class responsible for handling multiple spreadsheets, including saving, loading, and managing them . . . 32

Spreadsheet
: A class representing a spreadsheet consisting of cells arranged in rows and columns . . . 35

StringValueCell . . . 39

ValueCell . . . 41

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

AnsiTerminal.h 45
Cell.h 45
FileHandler.h 47
FormulaParser.h 47
myset.h 48
myvec.h 49
SheetHandler.h 51
Spreadsheet.h 51

Chapter 4

Class Documentation

4.1 AnsiTerminal Class Reference

Provides an interface for ANSI terminal manipulation, including text printing, screen clearing, and key detection.

`#include <AnsiTerminal.h>`

Public Member Functions

- **AnsiTerminal** ()

  Constructor: Sets up the terminal for capturing keystrokes.
- ~**AnsiTerminal** ()

  Destructor: Restores the terminal settings to the original state.
- void printAt (int row, int col, const std::string &text)

  Print text at a specified row and column.
- void printInvertedAt (int row, int col, const std::string &text)

  Print text with inverted background at a specified row and column.
- void **clearScreen** ()

  Clear the terminal screen.
- char getKeystroke ()

  Get a single keystroke from the terminal.
- char getSpecialKey ()

  Get the arrow key or special key input. Returns 'U', 'D', 'L', 'R' for Up, Down, Left, Right, respectively, or detects other key combinations such as Alt+Key, Ctrl+Key, etc.
- bool isArrowKey (const char ch)

  Check if a character corresponds to an arrow key.

4.1.1 Detailed Description

Provides an interface for ANSI terminal manipulation, including text printing, screen clearing, and key detection.

4.1.2 Member Function Documentation

4.1.2.1 getKeystroke()

```
char AnsiTerminal::getKeystroke ( )
```

Get a single keystroke from the terminal.

Returns

The character representing the keystroke.

4.1.2.2 getSpecialKey()

```
char AnsiTerminal::getSpecialKey ( )
```

Get the arrow key or special key input. Returns 'U', 'D', 'L', 'R' for Up, Down, Left, Right, respectively, or detects other key combinations such as Alt+Key, Ctrl+Key, etc.

Returns

A character representing the detected special key.

4.1.2.3 isArrowKey()

```
bool AnsiTerminal::isArrowKey (
            const char ch )
```

Check if a character corresponds to an arrow key.

Parameters

ch	The character to check.

Returns

True if the character represents an arrow key; false otherwise.

4.1.2.4 printAt()

```
void AnsiTerminal::printAt (
            int row,
            int col,
            const std::string & text )
```

Print text at a specified row and column.

Parameters

row	The row position (0-based index).
col	The column position (0-based index).
text	The text to be printed.

4.1.2.5 printInvertedAt()

```
void AnsiTerminal::printInvertedAt (
            int row,
            int col,
            const std::string & text )
```

Print text with inverted background at a specified row and column.

Parameters

row	The row position (0-based index).
col	The column position (0-based index).
text	The text to be printed with inverted colors.

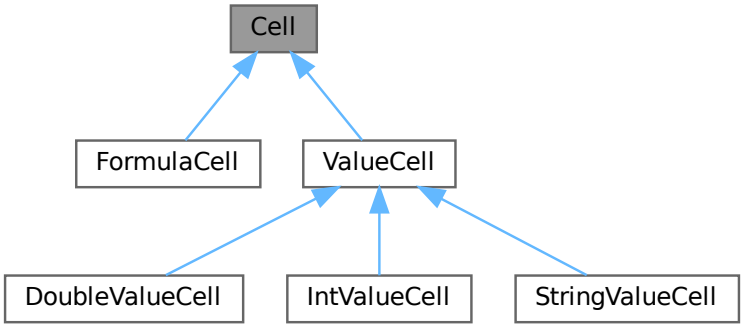
The documentation for this class was generated from the following files:

- AnsiTerminal.h
- AnsiTerminal.cpp

4.2 Cell Class Reference

```
#include <Cell.h>
```

Inheritance diagram for Cell:



Public Member Functions

- Cell (int r, int c)
- virtual ~Cell ()=default
- void setLetterRepresentation (int r, int c)
- std::string getLetterRepresentation () const
- int getRow () const
- int getCol () const
- double getCellValueAsDouble ()
- virtual std::string getValueAsString () const =0

4.2.1 Detailed Description

Abstract base class representing a generic spreadsheet cell. Provides common functionality for all cell types, including row and column management and letter representation for cell coordinates.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Cell()

```
Cell::Cell (
            int r,
            int c )  [inline]
```

Constructor for Cell.

Parameters

r	Row index (must be non-negative).
c	Column index (must be non-negative).

Exceptions

std::runtime_error	if row or column is negative.

4.2.2.2 ~Cell()

```
virtual Cell::~Cell ( )  [virtual], [default]
```

Virtual destructor for polymorphic behavior.

4.2.3 Member Function Documentation

4.2.3.1 getCellValueAsDouble()

```
double Cell::getCellValueAsDouble ( )
```

Retrieves the cell's value as a double. This method may throw if the value is non-numeric.

Returns

Cell value as double.

4.2.3.2 getCol()

```
int Cell::getCol ( ) const  [inline]
```

Retrieves the column index of the cell.

Returns

Column index.

4.2.3.3 getLetterRepresentation()

```
std::string Cell::getLetterRepresentation ( ) const  [inline]
```

Retrieves the letter representation of the cell.

Returns

String representing the cell's location (e.g., "A1").

4.2.3.4 getRow()

```
int Cell::getRow ( ) const  [inline]
```

Retrieves the row index of the cell.

Returns

Row index.

4.2.3.5 getValueAsString()

```
virtual std::string Cell::getValueAsString ( ) const  [pure virtual]
```

Pure virtual method to retrieve the cell's value as a string.

Returns

Cell value as a string.

Implemented in FormulaCell, IntValueCell, StringValueCell, and DoubleValueCell.

4.2.3.6 setLetterRepresentation()

```
void Cell::setLetterRepresentation (
            int r,
            int c )
```

Sets the letter representation for the cell based on its coordinates.

Parameters

r	Row index.
c	Column index.

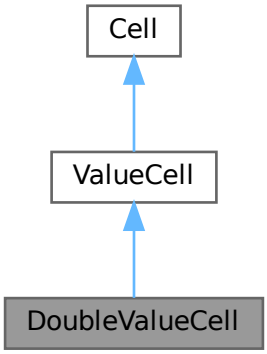
The documentation for this class was generated from the following files:

- Cell.h
- Cell.cpp

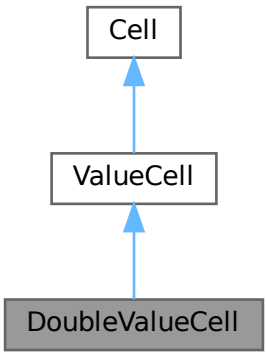
4.3 DoubleValueCell Class Reference

`#include <Cell.h>`

Inheritance diagram for DoubleValueCell:



Collaboration diagram for DoubleValueCell:



Public Member Functions

- DoubleValueCell (int r, int c, double value)
- std::string getValueAsString () const override
- double getValue () const
- void setValue (const std::string &v)

Public Member Functions inherited from ValueCell

- ValueCell (int r, int c)
- virtual ~ValueCell ()=default

Public Member Functions inherited from Cell

- Cell (int r, int c)
- virtual ~Cell ()=default
- void setLetterRepresentation (int r, int c)
- std::string getLetterRepresentation () const
- int getRow () const
- int getCol () const
- double getCellValueAsDouble ()

4.3.1 Detailed Description

Class representing a cell that stores a double value.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 DoubleValueCell()

```
DoubleValueCell::DoubleValueCell (
            int r,
            int c,
            double value )  [inline]
```

Constructor for DoubleValueCell.

Parameters

r	Row index.
c	Column index.
value	Initial double value.

4.3.3 Member Function Documentation

4.3.3.1 getValue()

```
double DoubleValueCell::getValue ( ) const  [inline]
```

Retrieves the double value of the cell.

Returns

Double value.

4.3.3.2 getValueAsString()

```
std::string DoubleValueCell::getValueAsString ( ) const  [inline], [override], [virtual]
```

Retrieves the cell's value as a string.

Returns

Double value as a string.

Implements Cell.

4.3.3.3 setValue()

```
void DoubleValueCell::setValue (
            const std::string & v )  [inline], [virtual]
```

Sets the double value of the cell.

Parameters

v	New value as a string.

Implements ValueCell.

The documentation for this class was generated from the following file:

- Cell.h

4.4 FileHandler Class Reference

A utility class for handling file operations for a Spreadsheet.

```
#include <FileHandler.h>
```

Public Member Functions

- void saveToFile (const std::string &filename, const Spreadsheet &spreadsheet)

  Saves the current state of the spreadsheet to a file.
- void loadFromFile (const std::string &filename, Spreadsheet &spreadsheet)

  Loads the state of the spreadsheet from a file.

4.4.1 Detailed Description

A utility class for handling file operations for a Spreadsheet.

4.4.2 Member Function Documentation

4.4.2.1 loadFromFile()

```
void FileHandler::loadFromFile (
            const std::string & filename,
            Spreadsheet & spreadsheet )
```

Loads the state of the spreadsheet from a file.

Parameters

filename	The name of the file to load from.
spreadsheet	The Spreadsheet object to populate.

4.4.2.2 saveToFile()

```
void FileHandler::saveToFile (
            const std::string & filename,
            const Spreadsheet & spreadsheet )
```

Saves the current state of the spreadsheet to a file.

Parameters

filename	The name of the file to save to.
spreadsheet	The Spreadsheet object to save.

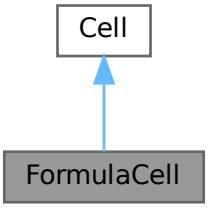
The documentation for this class was generated from the following files:

- FileHandler.h
- FileHandler.cpp

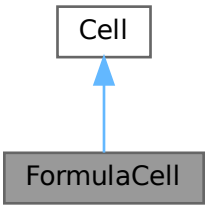
4.5 FormulaCell Class Reference

```
#include <Cell.h>
```

Inheritance diagram for FormulaCell:



Collaboration diagram for FormulaCell:



Public Member Functions

- FormulaCell (int r, int c, const std::string &f)
- void setCalculatedValue (double value)
- double getCalculatedValue () const
- const std::string & getFormula () const
- void addDependentCell (const std::pair< int, int > &coor)
- const spc::myvec< std::pair< int, int > > & fetchDependentCells () const
- void clearDependentCells ()
- std::string getValueAsString () const override

Public Member Functions inherited from Cell

- Cell (int r, int c)
- virtual ~Cell ()=default
- void setLetterRepresentation (int r, int c)
- std::string getLetterRepresentation () const
- int getRow () const
- int getCol () const
- double getCellValueAsDouble ()

4.5.1 Detailed Description

Class representing a formula-based cell. Stores a formula and its calculated value, and tracks dependent cells.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 FormulaCell()

```
FormulaCell::FormulaCell (
            int r,
            int c,
            const std::string & f )  [inline]
```

Constructor for FormulaCell.

Parameters

r	Row index.
c	Column index.
f	Formula string.

4.5.3 Member Function Documentation

4.5.3.1 addDependentCell()

```
void FormulaCell::addDependentCell (
            const std::pair< int, int > & coor )  [inline]
```

Adds a dependent cell to the list.

Parameters

coor	Pair representing the dependent cell's coordinates.

4.5.3.2 clearDependentCells()

```
void FormulaCell::clearDependentCells ( )  [inline]
```

Clears the list of dependent cells.

4.5.3.3 fetchDependentCells()

```
const spc::myvec< std::pair< int, int > > & FormulaCell::fetchDependentCells ( ) const  [inline]
```

Retrieves the list of dependent cells.

Returns

Vector of dependent cell coordinates.

4.5.3.4 getCalculatedValue()

```
double FormulaCell::getCalculatedValue ( ) const  [inline]
```

Retrieves the calculated value of the formula.

Returns

Calculated value as a double.

4.5.3.5 getFormula()

```
const std::string & FormulaCell::getFormula ( ) const  [inline]
```

Retrieves the formula string.

Returns

Formula string.

4.5.3.6 getValueAsString()

```
std::string FormulaCell::getValueAsString ( ) const  [inline], [override], [virtual]
```

Retrieves the cell's value as a string. Formats the value as an integer or double based on precision.

Returns

Formatted value string.

Implements Cell.

4.5.3.7 setCalculatedValue()

```
void FormulaCell::setCalculatedValue (
            double value )  [inline]
```

Sets the calculated value for the formula.

Parameters

value	The new calculated value.

The documentation for this class was generated from the following file:

- Cell.h

4.6 FormulaParser Class Reference

Responsible for parsing and evaluating formulas in the spreadsheet.

`#include <FormulaParser.h>`

Public Member Functions

- FormulaParser (Spreadsheet *sheet)

  Constructs a FormulaParser with a reference to the Spreadsheet.
- double parseAndEvaluate (std::string &formula, std::pair< int, int > coordinates, spc::myvec< std::pair< int, int > > &dependentCells)

  Parses and evaluates a formula, updating dependent cells as needed.
- void autoCalculate (std::pair< int, int > coordinate)

  Automatically recalculates cells dependent on a specified cell.

4.6.1 Detailed Description

Responsible for parsing and evaluating formulas in the spreadsheet.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 FormulaParser()

```
FormulaParser::FormulaParser (
            Spreadsheet * sheet )  [inline]
```

Constructs a FormulaParser with a reference to the Spreadsheet.

Parameters

sheet	Pointer to the Spreadsheet object.

4.6.3 Member Function Documentation

4.6.3.1 autoCalculate()

```
void FormulaParser::autoCalculate (
            std::pair< int, int > coordinate )
```

Automatically recalculates cells dependent on a specified cell.

Parameters

coordinate	The coordinates of the cell whose dependents need recalculating.

4.6.3.2 parseAndEvaluate()

```
double FormulaParser::parseAndEvaluate (
            std::string & formula,
            std::pair< int, int > coordinates,
            spc::myvec< std::pair< int, int > > & dependentCells )
```

Parses and evaluates a formula, updating dependent cells as needed.

Parameters

formula	The formula string to parse and evaluate.
coordinates	The coordinates of the cell containing the formula.
dependentCells	A vector to store dependent cell coordinates.

Returns

The calculated result of the formula.

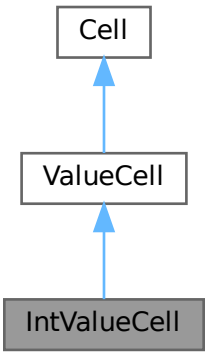
The documentation for this class was generated from the following files:

- FormulaParser.h
- FormulaParser.cpp

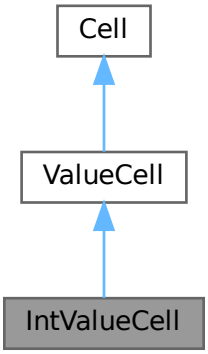
4.7 IntValueCell Class Reference

```
#include <Cell.h>
```

Inheritance diagram for IntValueCell:



Collaboration diagram for IntValueCell:



Public Member Functions

- IntValueCell (int r, int c, int value)
- std::string getValueAsString () const override
- int getValue () const
- void setValue (const std::string &v)

Public Member Functions inherited from ValueCell

- ValueCell (int r, int c)
- virtual ~ValueCell ()=default

Public Member Functions inherited from Cell

- Cell (int r, int c)
- virtual ~Cell ()=default
- void setLetterRepresentation (int r, int c)
- std::string getLetterRepresentation () const
- int getRow () const
- int getCol () const
- double getCellValueAsDouble ()

4.7.1 Detailed Description

Class representing a cell that stores an integer value.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 IntValueCell()

```
IntValueCell::IntValueCell (
            int r,
            int c,
            int value )  [inline]
```

Constructor for IntValueCell.

Parameters

r	Row index.
c	Column index.
value	Initial integer value.

4.7.3 Member Function Documentation

4.7.3.1 getValue()

```
int IntValueCell::getValue ( ) const  [inline]
```

Retrieves the integer value of the cell.

Returns

Integer value.

4.7.3.2 getValueAsString()

```
std::string IntValueCell::getValueAsString ( ) const  [inline], [override], [virtual]
```

Retrieves the cell's value as a string.

Returns

Integer value as a string.

Implements Cell.

4.7.3.3 setValue()

```
void IntValueCell::setValue (
            const std::string & v )  [inline], [virtual]
```

Sets the integer value of the cell.

Parameters

v	New value as a string.

Implements ValueCell.

The documentation for this class was generated from the following file:

- Cell.h

4.8 spc::myset< T > Class Template Reference

A custom implementation of a set container that prevents duplicate elements.

`#include <myset.h>`

Public Member Functions

- **myset** ()=default

  Default constructor for myset.
- void insert (const T &value)

  Inserts a value into the set if it is not already present.
- bool find (const T &value) const

  Checks if a value exists in the set.
- size_t size () const

  Gets the number of elements in the set.
- bool empty () const

  Checks if the set is empty.
- void **clear** ()

  Clears all elements from the set.
- T * begin ()

  Returns an iterator to the beginning of the set.
- T * end ()

  Returns an iterator to the end of the set.
- const T * begin () const

  Returns a constant iterator to the beginning of the set.
- const T * end () const

  Returns a constant iterator to the end of the set.

4.8.1 Detailed Description

template<typename T>
class spc::myset< T >

A custom implementation of a set container that prevents duplicate elements.

Template Parameters

T	The type of elements stored in the set.

4.8.2 Member Function Documentation

4.8.2.1 begin() [1/2]

```
template<typename T >
T * spc::myset< T >::begin ( )  [inline]
```

Returns an iterator to the beginning of the set.

Returns

Pointer to the first element in the set.

4.8.2.2 begin() [2/2]

```
template<typename T >
const T * spc::myset< T >::begin ( ) const  [inline]
```

Returns a constant iterator to the beginning of the set.

Returns

Constant pointer to the first element in the set.

4.8.2.3 empty()

```
template<typename T >
bool spc::myset< T >::empty ( ) const  [inline]
```

Checks if the set is empty.

Returns

True if the set is empty, false otherwise.

4.8.2.4 end() [1/2]

```
template<typename T >
T * spc::myset< T >::end ( )  [inline]
```

Returns an iterator to the end of the set.

Returns

Pointer to one past the last element in the set.

4.8.2.5 end() [2/2]

```
template<typename T >
const T * spc::myset< T >::end ( ) const  [inline]
```

Returns a constant iterator to the end of the set.

Returns

Constant pointer to one past the last element in the set.

4.8.2.6 find()

```
template<typename T >
bool spc::myset< T >::find (
            const T & value ) const  [inline]
```

Checks if a value exists in the set.

Parameters

value	The value to search for.

Returns

True if the value exists in the set, false otherwise.

4.8.2.7 insert()

```
template<typename T >
void spc::myset< T >::insert (
            const T & value )  [inline]
```

Inserts a value into the set if it is not already present.

Parameters

value	The value to insert.

4.8.2.8 size()

```
template<typename T >
size_t spc::myset< T >::size ( ) const  [inline]
```

Gets the number of elements in the set.

Returns

The size of the set.

The documentation for this class was generated from the following file:

- myset.h

4.9 spc::myvec< T > Class Template Reference

A custom dynamic array implementation similar to std::vector.

```
#include <myvec.h>
```

Public Member Functions

- myvec (int cap=10)

  Constructor to initialize the vector with a specified capacity.
- ~**myvec** ()

  Destructor to release allocated memory.
- myvec (const myvec &other)

  Copy constructor to create a new vector as a copy of another.
- myvec (myvec &&other) noexcept

  Move constructor to transfer ownership of resources.
- myvec & operator= (const myvec &other)

  Copy assignment operator to copy elements from another vector.
- myvec & operator= (myvec &&other) noexcept

  Move assignment operator to transfer resources from another vector.
- void push_back (T &&val)

  Adds an element to the end of the vector (move version).
- void push_back (const T &val)

  Adds an element to the end of the vector (copy version).
- T & operator[ ] (int index)

  Access operator to get or modify an element by index.
- const T & operator[ ] (int index) const

  Access operator to get an element by index (const version).
- int get_size () const

  Get the current size of the vector.
- int get_capacity () const

  Get the current capacity of the vector.
- template<typename InputIterator >
  myvec (InputIterator first, InputIterator last)

  Range-based constructor to initialize the vector from iterators.
- void **clear** ()

  Clear all elements from the vector.
- bool empty ()

  Check if the vector is empty.
- T * begin ()

  Get an iterator to the beginning of the vector.
- T * end ()

  Get an iterator to the end of the vector.
- const T * begin () const

  Get a const iterator to the beginning of the vector.
- const T * end () const

  Get a const iterator to the end of the vector.

4.9.1 Detailed Description

template<typename T>
class spc::myvec< T >

A custom dynamic array implementation similar to std::vector.

Template Parameters

T	The type of elements stored in the vector.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 myvec() [1/4]

```
template<typename T >
spc::myvec< T >::myvec (
            int cap = 10 )  [inline], [explicit]
```

Constructor to initialize the vector with a specified capacity.

Parameters

cap	The initial capacity of the vector (default is 10).

Exceptions

std::invalid_argument	if the capacity is not positive.

4.9.2.2 myvec() [2/4]

```
template<typename T >
spc::myvec< T >::myvec (
            const myvec< T > & other )  [inline]
```

Copy constructor to create a new vector as a copy of another.

Parameters

other	The vector to copy from.

4.9.2.3 myvec() [3/4]

```
template<typename T >
spc::myvec< T >::myvec (
            myvec< T > && other )  [inline], [noexcept]
```

Move constructor to transfer ownership of resources.

Parameters

other	The vector to move from.

4.9.2.4 myvec() [4/4]

```
template<typename T >
template<typename InputIterator >
spc::myvec< T >::myvec (
            InputIterator first,
            InputIterator last )  [inline]
```

Range-based constructor to initialize the vector from iterators.

Template Parameters

InputIterator	The type of the input iterator.

Parameters

first	The beginning of the range.
last	The end of the range.

4.9.3 Member Function Documentation

4.9.3.1 begin() [1/2]

```
template<typename T >
T * spc::myvec< T >::begin ( )  [inline]
```

Get an iterator to the beginning of the vector.

Returns

A pointer to the first element.

4.9.3.2 begin() [2/2]

```
template<typename T >
const T * spc::myvec< T >::begin ( ) const  [inline]
```

Get a const iterator to the beginning of the vector.

Returns

A const pointer to the first element.

4.9.3.3 empty()

```
template<typename T >
bool spc::myvec< T >::empty ( )  [inline]
```

Check if the vector is empty.

Returns

True if the vector is empty, false otherwise.

4.9.3.4 end() [1/2]

```
template<typename T >
T * spc::myvec< T >::end ( )  [inline]
```

Get an iterator to the end of the vector.

Returns

A pointer to one past the last element.

4.9.3.5 end() [2/2]

```
template<typename T >
const T * spc::myvec< T >::end ( ) const  [inline]
```

Get a const iterator to the end of the vector.

Returns

A const pointer to one past the last element.

4.9.3.6 get_capacity()

```
template<typename T >
int spc::myvec< T >::get_capacity ( ) const  [inline]
```

Get the current capacity of the vector.

Returns

The maximum number of elements the vector can hold.

4.9.3.7 get_size()

```
template<typename T >
int spc::myvec< T >::get_size ( ) const  [inline]
```

Get the current size of the vector.

Returns

The number of elements in the vector.

4.9.3.8 operator=() [1/2]

```
template<typename T >
myvec & spc::myvec< T >::operator= (
            const myvec< T > & other )  [inline]
```

Copy assignment operator to copy elements from another vector.

Parameters

other	The vector to copy from.

Returns

A reference to this vector.

4.9.3.9 operator=() [2/2]

```
template<typename T >
myvec & spc::myvec< T >::operator= (
            myvec< T > && other )  [inline], [noexcept]
```

Move assignment operator to transfer resources from another vector.

Parameters

other	The vector to move from.

Returns

A reference to this vector.

4.9.3.10 operator[]() [1/2]

```
template<typename T >
T & spc::myvec< T >::operator[] (
            int index )  [inline]
```

Access operator to get or modify an element by index.

Parameters

index	The index of the element to access.

Returns

A reference to the element.

4.9.3.11 operator[]() [2/2]

```
template<typename T >
const T & spc::myvec< T >::operator[] (
            int index ) const  [inline]
```

Access operator to get an element by index (const version).

Parameters

index	The index of the element to access.

Returns

A const reference to the element.

4.9.3.12 push_back() [1/2]

```
template<typename T >
void spc::myvec< T >::push_back (
            const T & val )  [inline]
```

Adds an element to the end of the vector (copy version).

Parameters

val	The value to copy into the vector.

4.9.3.13 push_back() [2/2]

```
template<typename T >
void spc::myvec< T >::push_back (
            T && val )  [inline]
```

Adds an element to the end of the vector (move version).

Parameters

val	The value to move into the vector.

The documentation for this class was generated from the following file:

- myvec.h

4.10 SheetHandler Class Reference

A class responsible for handling multiple spreadsheets, including saving, loading, and managing them.

```
#include <SheetHandler.h>
```

Public Member Functions

- SheetHandler (const std::string &dir_path="sheets")

  Constructs a SheetHandler object with an optional directory path.
- void add (const std::string &filename, Spreadsheet *newSheet)

  Adds a new spreadsheet to the handler.
- void saveSheet (const std::string &filename)

  Saves a spreadsheet to a file.
- void loadSheet (const std::string &filename)

  Loads a spreadsheet from a file.
- Spreadsheet * getSheet (const std::string &filename) const

  Retrieves a pointer to a spreadsheet by its filename.
- void **viewSavedSheets** () const

  Displays the list of saved spreadsheets.
- void **runMenu** ()

  Runs the menu interface for the user to interact with the spreadsheets.
- ~**SheetHandler** ()

  Destructor that cleans up any dynamically allocated memory.

4.10.1 Detailed Description

A class responsible for handling multiple spreadsheets, including saving, loading, and managing them.

The SheetHandler class allows the user to add, save, load, and view spreadsheets. It also manages a menu for creating and running spreadsheets through user interaction.

Note

The spreadsheets are stored in a specified directory (default is "sheets").

4.10.2 Constructor & Destructor Documentation

4.10.2.1 SheetHandler()

```
SheetHandler::SheetHandler (
            const std::string & dir_path = "sheets" )
```

Constructs a SheetHandler object with an optional directory path.

Parameters

dir_path	The path to the directory where sheets will be stored. Defaults to "sheets".

4.10.3 Member Function Documentation

4.10.3.1 add()

```
void SheetHandler::add (
```

```
            const std::string & filename,
            Spreadsheet * newSheet )
```

Adds a new spreadsheet to the handler.

Parameters

filename	The name of the spreadsheet to add.
newSheet	A pointer to the new Spreadsheet object.

4.10.3.2 getSheet()

```
Spreadsheet * SheetHandler::getSheet (
            const std::string & filename ) const
```

Retrieves a pointer to a spreadsheet by its filename.

Parameters

filename	The name of the spreadsheet to retrieve.

Returns

A pointer to the Spreadsheet object, or nullptr if not found.

4.10.3.3 loadSheet()

```
void SheetHandler::loadSheet (
            const std::string & filename )
```

Loads a spreadsheet from a file.

Parameters

filename	The name of the file to load the spreadsheet from.

4.10.3.4 saveSheet()

```
void SheetHandler::saveSheet (
            const std::string & filename )
```

Saves a spreadsheet to a file.

Parameters

filename	The name of the file to save the spreadsheet as.

The documentation for this class was generated from the following files:

- SheetHandler.h
- SheetHandler.cpp

4.11 Spreadsheet Class Reference

A class representing a spreadsheet consisting of cells arranged in rows and columns.

`#include <Spreadsheet.h>`

Public Member Functions

- Spreadsheet (int rows, int cols)

  Constructs a Spreadsheet object with a specified number of rows and columns.
- **Spreadsheet** ()

  Default constructor that creates a 3x3 spreadsheet.
- Cell * getCell (int r, int c) const

  Retrieves a pointer to the cell at the specified row and column.
- void setCell (int r, int c, std::unique_ptr< Cell > cell)

  Sets the cell at the specified row and column with a given cell object.
- void enterData (int r, int c, std::string &input)

  Enters data into a specified cell in the spreadsheet by taking a reference to the input string.
- void enterData (int r, int c, std::string &&input)

  Enters data into a specified cell in the spreadsheet by taking an r-value reference to the input string.
- int getRowCount () const

  Returns the total number of rows in the spreadsheet.
- int getColCount () const

  Returns the total number of columns in the spreadsheet.
- spc::myvec< Cell * > getCellsInRange (std::pair< int, int > startPos, std::pair< int, int > endPos)

  Retrieves a list of cells within a specified range.
- void displayScreen (int currentRow, int currentCol, AnsiTerminal &terminal, std::string inputLine="")

  Displays the contents of the spreadsheet on the terminal.
- void **run** ()

  Runs the spreadsheet, initiating the user interface and interactive features.

Static Public Attributes

- static const int **MAX_ROWS** = 100

  Maximum number of rows in the spreadsheet.
- static const int **MAX_COLS** = 50

  Maximum number of columns in the spreadsheet.

Friends

- class FileHandler

  Friend class FileHandler, allowing it to access private members of Spreadsheet.

4.11.1 Detailed Description

A class representing a spreadsheet consisting of cells arranged in rows and columns.

The Spreadsheet class allows you to manage a grid of cells, input data into individual cells, display the content of cells, and perform operations like navigating and expanding the grid. It supports formulas and provides an interface for interacting with the content of the spreadsheet.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Spreadsheet()

```
Spreadsheet::Spreadsheet (
            int rows,
            int cols )
```

Constructs a Spreadsheet object with a specified number of rows and columns.

Parameters

rows	The number of rows in the spreadsheet.
cols	The number of columns in the spreadsheet.

4.11.3 Member Function Documentation

4.11.3.1 displayScreen()

```
void Spreadsheet::displayScreen (
            int currentRow,
            int currentCol,
            AnsiTerminal & terminal,
            std::string inputLine = "" )
```

Displays the contents of the spreadsheet on the terminal.

Parameters

currentRow	The current row index to highlight.
currentCol	The current column index to highlight.
terminal	A reference to the AnsiTerminal object used for display.
inputLine	The input line to be displayed, if any.

4.11.3.2 enterData() [1/2]

```
void Spreadsheet::enterData (
            int r,
```

```
            int c,
            std::string && input )
```

Enters data into a specified cell in the spreadsheet by taking an r-value reference to the input string.

Parameters

r	The row index of the cell.
c	The column index of the cell.
input	The input string containing the data to be entered into the cell.

4.11.3.3 enterData() [2/2]

```
void Spreadsheet::enterData (
            int r,
            int c,
            std::string & input )
```

Enters data into a specified cell in the spreadsheet by taking a reference to the input string.

Parameters

r	The row index of the cell.
c	The column index of the cell.
input	The input string containing the data to be entered into the cell.

4.11.3.4 getCell()

```
Cell * Spreadsheet::getCell (
            int r,
            int c ) const
```

Retrieves a pointer to the cell at the specified row and column.

Parameters

r	The row index of the cell.
c	The column index of the cell.

Returns

A pointer to the Cell object at the specified position.

4.11.3.5 getCellsInRange()

```
spc::myvec< Cell * > Spreadsheet::getCellsInRange (
            std::pair< int, int > startPos,
            std::pair< int, int > endPos )
```

Retrieves a list of cells within a specified range.

Parameters

startPos	A pair representing the starting row and column.
endPos	A pair representing the ending row and column.

Returns

A vector of pointers to the cells within the specified range.

4.11.3.6 getColCount()

```
int Spreadsheet::getColCount ( ) const  [inline]
```

Returns the total number of columns in the spreadsheet.

Returns

The number of columns in the spreadsheet.

4.11.3.7 getRowCount()

```
int Spreadsheet::getRowCount ( ) const  [inline]
```

Returns the total number of rows in the spreadsheet.

Returns

The number of rows in the spreadsheet.

4.11.3.8 setCell()

```
void Spreadsheet::setCell (
            int r,
            int c,
            std::unique_ptr< Cell > cell )
```

Sets the cell at the specified row and column with a given cell object.

Parameters

r	The row index of the cell.
c	The column index of the cell.
cell	A unique pointer to the Cell object to set at the specified position.

The documentation for this class was generated from the following files:

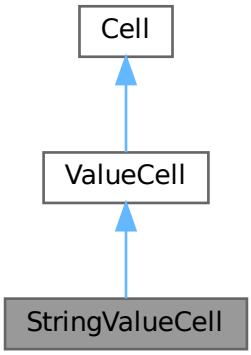
- Spreadsheet.h

- Spreadsheet.cpp

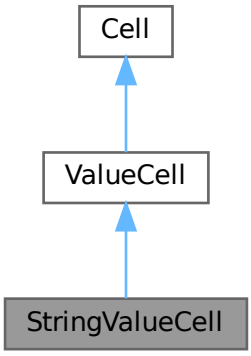
4.12 StringValueCell Class Reference

`#include <Cell.h>`

Inheritance diagram for StringValueCell:



Collaboration diagram for StringValueCell:



Public Member Functions

- StringValueCell (int r, int c, const std::string &value)
- std::string getValueAsString () const override
- std::string getValue () const
- void setValue (const std::string &v)

Public Member Functions inherited from ValueCell

- ValueCell (int r, int c)
- virtual ~ValueCell ()=default

Public Member Functions inherited from Cell

- Cell (int r, int c)
- virtual ~Cell ()=default
- void setLetterRepresentation (int r, int c)
- std::string getLetterRepresentation () const
- int getRow () const
- int getCol () const
- double getCellValueAsDouble ()

4.12.1 Detailed Description

Class representing a cell that stores a string value.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 StringValueCell()

```
StringValueCell::StringValueCell (
            int r,
            int c,
            const std::string & value )  [inline]
```

Constructor for StringValueCell.

Parameters

r	Row index.
c	Column index.
value	Initial string value.

4.12.3 Member Function Documentation

4.12.3.1 getValue()

```
std::string StringValueCell::getValue ( ) const  [inline]
```

Retrieves the string value of the cell.

Returns

String value.

4.12.3.2 getValueAsString()

```
std::string StringValueCell::getValueAsString ( ) const  [inline], [override], [virtual]
```

Retrieves the cell's value as a string.

Returns

String value.

Implements Cell.

4.12.3.3 setValue()

```
void StringValueCell::setValue (
            const std::string & v )  [inline], [virtual]
```

Sets the string value of the cell.

Parameters

v	New value as a string.

Implements ValueCell.

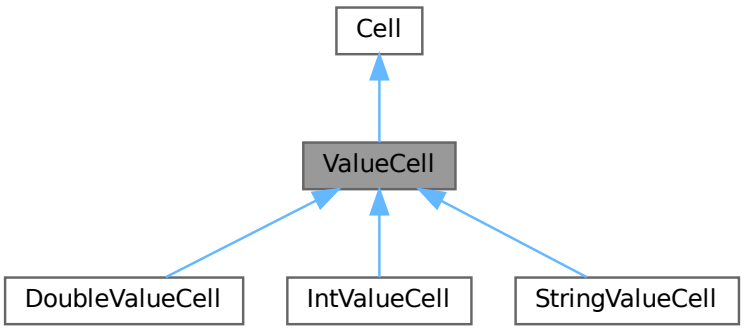
The documentation for this class was generated from the following file:

- Cell.h

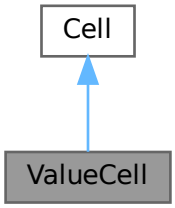
4.13 ValueCell Class Reference

```
#include <Cell.h>
```

Inheritance diagram for ValueCell:



Collaboration diagram for ValueCell:



Public Member Functions

- ValueCell (int r, int c)
- virtual ~ValueCell ()=default
- virtual void setValue (const std::string &v)=0

Public Member Functions inherited from Cell

- Cell (int r, int c)
- virtual ~Cell ()=default
- void setLetterRepresentation (int r, int c)
- std::string getLetterRepresentation () const
- int getRow () const
- int getCol () const
- double getCellValueAsDouble ()
- virtual std::string getValueAsString () const =0

4.13.1 Detailed Description

Abstract base class for value-based cells. Provides common functionality for cells storing direct values.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 ValueCell()

```
ValueCell::ValueCell (
            int r,
            int c )  [inline]
```

Constructor for ValueCell.

Parameters

r	Row index.
c	Column index.

4.13.2.2 ~ValueCell()

```
virtual ValueCell::~ValueCell ( )  [virtual], [default]
```

Virtual destructor for polymorphic behavior.

4.13.3 Member Function Documentation

4.13.3.1 setValue()

```
virtual void ValueCell::setValue (
            const std::string & v )  [pure virtual]
```

Pure virtual method to set the cell's value.

Parameters

v	New value as a string.

Implemented in IntValueCell, StringValueCell, and DoubleValueCell.

The documentation for this class was generated from the following file:

- Cell.h

Chapter 5

File Documentation

5.1 AnsiTerminal.h

```
00001 #ifndef ANSI_TERMINAL_H
00002 #define ANSI_TERMINAL_H
00003
00004 #include <string>
00005 #include <termios.h>
00006
00011 class AnsiTerminal {
00012 public:
00016     AnsiTerminal();
00017
00021     ~AnsiTerminal();
00022
00029     void printAt(int row, int col, const std::string &text);
00030
00037     void printInvertedAt(int row, int col, const std::string &text);
00038
00042     void clearScreen();
00043
00048     char getKeystroke();
00049
00056     char getSpecialKey();
00057
00063     bool isArrowKey(const char ch);
00064
00065 private:
00066     struct termios original_tio;
00067 };
00068
00069 #endif // ANSI_TERMINAL_H
```

5.2 Cell.h

```
00001 #ifndef CELL_H
00002 #define CELL_H
00003
00004 #include <string>
00005 #include <stdexcept>
00006 #include <sstream>
00007 #include <iomanip>
00008 #include "myvec.h"
00009
00015 class Cell
00016 {
00017 public:
00024     Cell(int r, int c) : row(r), col(c)
00025     {
00026         if (row < 0 || col < 0)
00027             throw std::runtime_error("Invalid initializer for Cell instance.");
00028         setLetterRepresentation(r, c);
00029     }
00030
00034     virtual ~Cell() = default;
00035
00041     void setLetterRepresentation(int r, int c);
```

```
00042
00047     std::string getLetterRepresentation() const { return letter_rep; }
00048
00053     int getRow() const { return row; }
00054
00059     int getCol() const { return col; }
00060
00066     double getCellValueAsDouble();
00067
00072     virtual std::string getValueAsString() const = 0;
00073
00074 private:
00075     std::string letter_rep;
00076     int row, col;
00077 };
00078
00083 class FormulaCell : public Cell
00084 {
00085 public:
00092     FormulaCell(int r, int c, const std::string &f)
00093         : Cell(r, c), formula(f), calculatedValue(0) {}
00094
00099     void setCalculatedValue(double value) { calculatedValue = value; }
00100
00105     double getCalculatedValue() const { return calculatedValue; }
00106
00111     const std::string &getFormula() const { return formula; }
00112
00117     void addDependentCell(const std::pair<int, int> &coor) { dependentCells.push_back(coor); }
00118
00123     const spc::myvec<std::pair<int, int>> &fetchDependentCells() const { return dependentCells; }
00124
00128     void clearDependentCells() { dependentCells.clear(); }
00129
00135     std::string getValueAsString() const override
00136     {
00137         std::ostringstream oss;
00138
00139         if (isInteger(calculatedValue))
00140         {
00141             oss « std::fixed « std::setprecision(0) « calculatedValue;
00142         }
00143         else
00144         {
00145             oss « std::fixed « std::setprecision(2) « calculatedValue;
00146         }
00147
00148         return oss.str();
00149     }
00150
00151 private:
00157     bool isInteger(double calculatedValue) const
00158     {
00159         return calculatedValue == static_cast<int>(calculatedValue);
00160     }
00161
00162     std::string formula;
00163     double calculatedValue;
00164     spc::myvec<std::pair<int, int>> dependentCells;
00165 };
00166
00171 class ValueCell : public Cell
00172 {
00173 public:
00179     ValueCell(int r, int c) : Cell(r, c) {}
00180
00184     virtual ~ValueCell() = default;
00185
00190     virtual void setValue(const std::string &v) = 0;
00191 };
00192
00196 class IntValueCell : public ValueCell
00197 {
00198 public:
00205     IntValueCell(int r, int c, int value)
00206         : ValueCell(r, c), val(value) {}
00207
00212     std::string getValueAsString() const override
00213     {
00214         return std::to_string(val);
00215     }
00216
00221     int getValue() const { return val; }
00222
00227     void setValue(const std::string &v) { val = std::stoi(v); }
00228
00229 private:
```

```
00230     int val;
00231 };
00232
00236 class StringValueCell : public ValueCell
00237 {
00238 public:
00245     StringValueCell(int r, int c, const std::string &value)
00246         : ValueCell(r, c), val(value) {}
00247
00252     std::string getValueAsString() const override
00253     {
00254         return val;
00255     }
00256
00261     std::string getValue() const { return val; }
00262
00267     void setValue(const std::string &v) { val = v; }
00268
00269 private:
00270     std::string val;
00271 };
00272
00276 class DoubleValueCell : public ValueCell
00277 {
00278 public:
00285     DoubleValueCell(int r, int c, double value)
00286         : ValueCell(r, c), val(value) {}
00287
00292     std::string getValueAsString() const override
00293     {
00294         std::ostringstream oss;
00295         oss « val;
00296         return oss.str();
00297     }
00298
00303     double getValue() const { return val; }
00304
00309     void setValue(const std::string &v) { val = std::stod(v); }
00310
00311 private:
00312     double val;
00313 };
00314
00315 #endif
```

5.3 FileHandler.h

```
00001 #ifndef HANDLE_EM
00002 #define HANDLE_EM
00003
00004 #include <string>
00005 #include "Spreadsheet.h"
00006
00007 class Spreadsheet;
00008
00013 class FileHandler
00014 {
00015 public:
00021     void saveToFile(const std::string &filename, const Spreadsheet &spreadsheet);
00022
00028     void loadFromFile(const std::string &filename, Spreadsheet &spreadsheet);
00029
00030 private:
00036     bool isInteger(const std::string& str);
00037
00043     bool isDouble(const std::string& str);
00044 };
00045
00046 #endif
```

5.4 FormulaParser.h

```
00001 #ifndef PARSE_EM
00002 #define PARSE_EM
00003
00004 #include "myvec.h"
00005 #include "myset.h"
00006 #include <string>
00007 #include <vector>
```

```
00008 #include <set>
00009 #include <iostream>
00010
00015 enum class FunctionType
00016 {
00017     SUM,
00018     AVER,
00019     STDDEV,
00020     MAX,
00021     MIN,
00022     INVALID
00023 };
00024
00025 class Spreadsheet;
00026
00031 class FormulaParser
00032 {
00033 public:
00038     FormulaParser(Spreadsheet *sheet) : spreadsheet(sheet) {}
00039
00047     double parseAndEvaluate(std::string &formula, std::pair<int, int> coordinates,
      spc::myvec<std::pair<int, int> &dependentCells);
00048
00053     void autoCalculate(std::pair<int, int> coordinate);
00054
00055 private:
00056     Spreadsheet *spreadsheet;
00057
00063     spc::myvec<std::string> parsePlusAndMinus(const std::string &formula) const;
00064
00070     spc::myvec<std::string> parseMultpAndDiv(const std::string &token) const;
00071
00078     double evaluateMultpAndDivToken(std::string &token, spc::myset<std::pair<int, int>
      &uniqueDependents) const;
00079
00085     bool isValidRange(const std::string &range) const;
00086
00092     std::pair<int, int> getCellReference(const std::string &token) const;
00093
00099     FunctionType getFunctionType(std::string &token) const;
00100
00106     bool isValue(const std::string &token) const;
00107
00114     double evaluateSingleToken(std::string &singleToken, spc::myset<std::pair<int, int>
      &uniqueDependents) const;
00115
00123     double SUM(std::pair<int, int> startPos, std::pair<int, int> endPos, spc::myset<std::pair<int,
      int>> &uniqueDependents) const;
00124
00132     double AVER(std::pair<int, int> startPos, std::pair<int, int> endPos, spc::myset<std::pair<int,
      int>> &uniqueDependents) const;
00133
00141     double STDDEV(std::pair<int, int> startPos, std::pair<int, int> endPos, spc::myset<std::pair<int,
      int>> &uniqueDependents) const;
00142
00150     double MAX(std::pair<int, int> startPos, std::pair<int, int> endPos, spc::myset<std::pair<int,
      int>> &uniqueDependents) const;
00151
00159     double MIN(std::pair<int, int> startPos, std::pair<int, int> endPos, spc::myset<std::pair<int,
      int>> &uniqueDependents) const;
00160 };
00161
00162 #endif
```

5.5 myset.h

```
00001 #ifndef SPC_MYSET_H
00002 #define SPC_MYSET_H
00003
00004 #include <iostream>
00005 #include "myvec.h"
00006 #include <algorithm>
00007
00008 namespace spc {
00009
00015 template <typename T>
00016 class myset {
00017 private:
00018     spc::myvec<T> elements;
00019
00020 public:
00024     myset() = default;
00025
```

```
00030     void insert(const T& value) {
00031         if (std::find(elements.begin(), elements.end(), value) == elements.end()) {
00032             elements.push_back(value);
00033         }
00034     }
00035
00041     bool find(const T& value) const {
00042         return std::find(elements.begin(), elements.end(), value) != elements.end();
00043     }
00044
00049     size_t size() const {
00050         return elements.size();
00051     }
00052
00057     bool empty() const {
00058         return elements.empty();
00059     }
00060
00064     void clear() {
00065         elements.clear();
00066     }
00067
00072     T* begin() { return elements.begin(); }
00073
00078     T* end() { return elements.end(); }
00079
00084     const T* begin() const { return elements.begin(); }
00085
00090     const T* end() const { return elements.end(); }
00091
00092 };
00093
00094 } // namespace spc
00095
00096 #endif // SPC_MYSET_H
```

5.6 myvec.h

```
00001 #ifndef MYVEC_H
00002 #define MYVEC_H
00003
00004 #include <stdexcept>
00005 #include <iterator>
00006
00007 #include <iostream>
00008
00009 namespace spc
00010 {
00011
00017     template <typename T>
00018     class myvec
00019     {
00020     public:
00026         explicit myvec(int cap = 10)
00027             : size(0), capacity(cap)
00028         {
00029             if (capacity <= 0)
00030                 throw std::invalid_argument("Capacity must be positive");
00031             data = new T[capacity];
00032         }
00033
00037         ~myvec()
00038         {
00039             delete[] data;
00040         }
00041
00046         myvec(const myvec &other)
00047             : size(other.size), capacity(other.capacity)
00048         {
00049             data = new T[capacity];
00050             for (int i = 0; i < size; ++i)
00051             {
00052                 data[i] = other.data[i];
00053             }
00054         }
00055
00060         myvec(myvec &&other) noexcept
00061             : data(other.data), size(other.size), capacity(other.capacity)
00062         {
00063             other.data = nullptr;
00064             other.size = 0;
00065             other.capacity = 0;
00066         }
```

```
00067
00073         myvec &operator=(const myvec &other)
00074         {
00075             if (this != &other)
00076             {
00077                 delete[] data;
00078                 size = other.size;
00079                 capacity = other.capacity;
00080                 data = new T[capacity];
00081                 for (int i = 0; i < size; ++i)
00082                 {
00083                     data[i] = other.data[i];
00084                 }
00085             }
00086             return *this;
00087         }
00088
00094         myvec &operator=(myvec &&other) noexcept
00095         {
00096             if (this != &other)
00097             {
00098                 delete[] data;
00099                 data = other.data;
00100                 size = other.size;
00101                 capacity = other.capacity;
00102                 other.data = nullptr;
00103                 other.size = 0;
00104                 other.capacity = 0;
00105             }
00106             return *this;
00107         }
00108
00113         void push_back(T &&val)
00114         {
00115             if (size == capacity)
00116             {
00117                 resize();
00118             }
00119             data[size++] = std::move(val);
00120         }
00121
00126         void push_back(const T &val)
00127         {
00128             if (size == capacity)
00129             {
00130                 resize();
00131             }
00132             data[size++] = val;
00133         }
00134
00140         T &operator[](int index)
00141         {
00142             return data[index];
00143         }
00144
00150         const T &operator[](int index) const
00151         {
00152             return data[index];
00153         }
00154
00159         int get_size() const { return size; }
00160
00165         int get_capacity() const { return capacity; }
00166
00173         template <typename InputIterator>
00174         myvec(InputIterator first, InputIterator last)
00175         {
00176             size = std::distance(first, last);
00177             capacity = size;
00178             data = new T[capacity];
00179             int index = 0;
00180             for (auto it = first; it != last; ++it)
00181             {
00182                 data[index++] = *it;
00183             }
00184         }
00185
00189         void clear()
00190         {
00191             size = 0;
00192         }
00193
00198         bool empty()
00199         {
00200             return size == 0;
00201         }
00202
```

```
00207         T *begin() { return data; }
00208
00213         T *end() { return data + size; }
00214
00219         const T *begin() const { return data; }
00220
00225         const T *end() const { return data + size; }
00226
00227     private:
00228         T *data;
00229         int size;
00230         int capacity;
00235         void resize()
00236         {
00237             capacity *= 2;
00238             T *new_data = new T[capacity];
00239             for (int i = 0; i < size; ++i)
00240             {
00241                 new_data[i] = std::move(data[i]);
00242             }
00243             delete[] data;
00244             data = new_data;
00245         }
00246     };
00247
00248 } // namespace spc
00249
00250 #endif // MYVEC_H
```

5.7 SheetHandler.h

```
00001 #ifndef SHEETHANDLER_H
00002 #define SHEETHANDLER_H
00003
00004 #include "Spreadsheet.h"
00005 #include "FileHandler.h"
00006 #include <string>
00007 #include <unordered_map>
00008 #include <filesystem>
00009
00010 namespace fs = std::filesystem;
00011
00024 class SheetHandler
00025 {
00026 public:
00033     SheetHandler(const std::string& dir_path = "sheets");
00034
00041     void add(const std::string& filename, Spreadsheet* newSheet);
00042
00048     void saveSheet(const std::string& filename);
00049
00055     void loadSheet(const std::string& filename);
00056
00064     Spreadsheet* getSheet(const std::string& filename) const;
00065
00069     void viewSavedSheets() const;
00070
00074     void runMenu();
00075
00079     ~SheetHandler();
00080
00081 private:
00083     std::unordered_map<std::string, Spreadsheet*> sheets;
00084
00086     FileHandler handler;
00087
00089     const std::string directory_path;
00090
00094     void displayMenu() const;
00095
00099     void handleCreate();
00100
00104     void handleRun();
00105 };
00106
00107 #endif
```

5.8 Spreadsheet.h

```
00001 #ifndef SPREADSHEET_H
```

```
00002 #define SPREADSHEET_H
00003
00004 #include "Cell.h"
00005 #include "AnsiTerminal.h"
00006 #include "myvec.h"
00007 #include "FormulaParser.h"
00008 #include <string>
00009 #include <stdexcept>
00010 #include <memory>
00011 #include <iostream>
00012
00024 class Spreadsheet
00025 {
00026 public:
00028     static const int MAX_ROWS = 100;
00029
00031     static const int MAX_COLS = 50;
00032
00039     Spreadsheet(int rows, int cols);
00040
00044     Spreadsheet() : Spreadsheet(3, 3) {}
00045
00054     Cell *getCell(int r, int c) const;
00055
00063     void setCell(int r, int c, std::unique_ptr<Cell> cell);
00064
00072     void enterData(int r, int c, std::string& input);
00073
00081     void enterData(int r, int c, std::string&& input);
00082
00088     int getRowCount() const { return cells.get_size(); }
00089
00095     int getColCount() const { return cells[0].get_size(); }
00096
00105     spc::myvec<Cell *> getCellsInRange(std::pair<int, int> startPos, std::pair<int, int> endPos);
00106
00115     void displayScreen(int currentRow, int currentCol, AnsiTerminal& terminal, std::string inputLine =
    "");
00116
00120     void run();
00121
00125     friend class FileHandler;
00126
00127 private:
00129     spc::myvec<spc::myvec<std::unique_ptr<Cell>» cells;
00130
00132     std::shared_ptr<FormulaParser> parser;
00133
00140     void expand(int newRowCount, int newColCount);
00141
00149     std::string getColumnLabel(int columnIndex) const;
00150
00159     std::string getCellLabel(int r, int c) const;
00160
00169     std::string formatCellText(const std::string &cellText, int width);
00170
00178     void moveCell(int &currentRow, int &currentCol, const char dir);
00179 };
00180
00181 #endif
```

Index

~Cell
  Cell, 10
~ValueCell
  ValueCell, 43

add
  SheetHandler, 33
addDependentCell
  FormulaCell, 17
AnsiTerminal, 7
  getKeystroke, 8
  getSpecialKey, 8
  isArrowKey, 8
  printAt, 8
  printInvertedAt, 9
autoCalculate
  FormulaParser, 19

begin
  spc::myset< T >, 24, 25
  spc::myvec< T >, 29

Cell, 9
  ~Cell, 10
  Cell, 10
  getCellValueAsDouble, 10
  getCol, 11
  getLetterRepresentation, 11
  getRow, 11
  getValueAsString, 11
  setLetterRepresentation, 11
clearDependentCells
  FormulaCell, 17

displayScreen
  Spreadsheet, 36
DoubleValueCell, 12
  DoubleValueCell, 13
  getValue, 14
  getValueAsString, 14
  setValue, 14

empty
  spc::myset< T >, 25
  spc::myvec< T >, 29
end
  spc::myset< T >, 25
  spc::myvec< T >, 30
enterData
  Spreadsheet, 36, 37

fetchDependentCells
  FormulaCell, 17
FileHandler, 14
  loadFromFile, 15
  saveToFile, 15
find
  spc::myset< T >, 25
FormulaCell, 15
  addDependentCell, 17
  clearDependentCells, 17
  fetchDependentCells, 17
  FormulaCell, 17
  getCalculatedValue, 17
  getFormula, 18
  getValueAsString, 18
  setCalculatedValue, 18
FormulaParser, 19
  autoCalculate, 19
  FormulaParser, 19
  parseAndEvaluate, 20

get_capacity
  spc::myvec< T >, 30
get_size
  spc::myvec< T >, 30
getCalculatedValue
  FormulaCell, 17
getCell
  Spreadsheet, 37
getCellsInRange
  Spreadsheet, 37
getCellValueAsDouble
  Cell, 10
getCol
  Cell, 11
getColCount
  Spreadsheet, 38
getFormula
  FormulaCell, 18
getKeystroke
  AnsiTerminal, 8
getLetterRepresentation
  Cell, 11
getRow
  Cell, 11
getRowCount
  Spreadsheet, 38
getSheet
  SheetHandler, 34
getSpecialKey

AnsiTerminal, 8
getValue
DoubleValueCell, 14
IntValueCell, 23
StringValueCell, 40
getValueAsString
Cell, 11
DoubleValueCell, 14
FormulaCell, 18
IntValueCell, 23
StringValueCell, 40

insert
spc::myset< T >, 26
IntValueCell, 20
getValue, 23
getValueAsString, 23
IntValueCell, 22
setValue, 23
isArrowKey
AnsiTerminal, 8

loadFromFile
FileHandler, 15
loadSheet
SheetHandler, 34

myvec
spc::myvec< T >, 28

operator=
spc::myvec< T >, 30, 31
operator[]
spc::myvec< T >, 31

parseAndEvaluate
FormulaParser, 20
printAt
AnsiTerminal, 8
printInvertedAt
AnsiTerminal, 9
push_back
spc::myvec< T >, 32

saveSheet
SheetHandler, 34
saveToFile
FileHandler, 15
setCalculatedValue
FormulaCell, 18
setCell
Spreadsheet, 38
setLetterRepresentation
Cell, 11
setValue
DoubleValueCell, 14
IntValueCell, 23
StringValueCell, 41
ValueCell, 43
SheetHandler, 32
add, 33
getSheet, 34
loadSheet, 34
saveSheet, 34
SheetHandler, 33
size
spc::myset< T >, 26
spc::myset< T >, 24
begin, 24, 25
empty, 25
end, 25
find, 25
insert, 26
size, 26
spc::myvec< T >, 26
begin, 29
empty, 29
end, 30
get_capacity, 30
get_size, 30
myvec, 28
operator=, 30, 31
operator[], 31
push_back, 32
Spreadsheet, 35
displayScreen, 36
enterData, 36, 37
getCell, 37
getCellsInRange, 37
getColCount, 38
getRowCount, 38
setCell, 38
Spreadsheet, 36
StringValueCell, 39
getValue, 40
getValueAsString, 40
setValue, 41
StringValueCell, 40

ValueCell, 41
~ValueCell, 43
setValue, 43
ValueCell, 42