FMI Table

Generated by Doxygen 1.9.3

Chapter 1

FMI Tables

Author

Kamen Mladenov; FMI Computer Science, Year 1, Group 6; Project 2, Topic 7

Date

14 June 2022

Source Code https://github.com/Syndamia/FMI-OOP-P2_Tables

1.1 About

"FMI Tables" is a simple implementation of parsing and editing a CSV-style table file, that stores strings, formulas and numbers.

1.2 Structure overview

The project is roughly divided into 3 main components: User Interface, Models and Generic fig1.

- The User Interface is the messenger between a user and the underlying application. All input and output is handled here.
- Models are the general classes that are used in the user interface and that do the value parsing.
- Generic is a place for all code that is can be used independently from the project. Stuff like Pair, String or List.

1.3 Building the project

Instructions are all for `gcc`. All commands should be executed while in the `src` folder.

1.3.1 Linux/BSD/MacOS

```
g++ **/**.cpp -o TicketSystem.out && ./TicketSystem.out
```

1.3.2 Windows (PowerShell)

```
g++ (Get-ChildItem -Recurse *.cpp) -o TicketSystem.exe && ./TicketSystem.exe
```

1.4 Figures

Figure 1: Overview of the "FMI Ticket System" design

Figure 1.1 Not available in LaTeX!

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

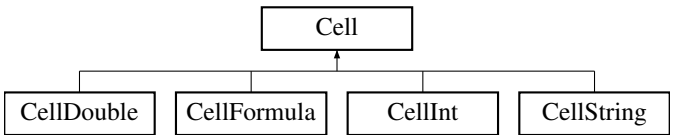
Class Documentation

5.1 Cell Class Reference

Fully abstract class that represents a cell in the table.

`#include <Cell.h>`

Inheritance diagram for Cell:



Public Member Functions

- virtual double getNumeralValue () const =0

  Returns the value of the cell as a double-precision floating point number.
- virtual String getValueForPrint () const =0

  Returns the value of the cell as a String.
- virtual void parseAndSetValue (const char *str)=0

  Parses given C-style string and stores value.
- virtual void writeToFile (std::ofstream &file)=0

  Writes value to a text file.

5.1.1 Detailed Description

Fully abstract class that represents a cell in the table.

5.1.2 Member Function Documentation

5.1.2.1 getNumeralValue()

```
virtual double Cell::getNumeralValue ( ) const  [pure virtual]
```

Returns the value of the cell as a double-precision floating point number.

Implemented in CellDouble, CellFormula, CellInt, and CellString.

5.1.2.2 getValueForPrint()

```
virtual String Cell::getValueForPrint ( ) const  [pure virtual]
```

Returns the value of the cell as a String.

Implemented in CellDouble, CellFormula, CellInt, and CellString.

5.1.2.3 parseAndSetValue()

```
virtual void Cell::parseAndSetValue (
            const char * str )  [pure virtual]
```

Parses given C-style string and stores value.

Implemented in CellDouble, CellFormula, CellInt, and CellString.

5.1.2.4 writeToFile()

```
virtual void Cell::writeToFile (
            std::ofstream & file )  [pure virtual]
```

Writes value to a text file.

Implemented in CellDouble, CellFormula, CellInt, and CellString.

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

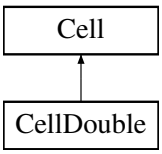
- src/Models/Cell.h

5.2 CellDouble Class Reference

A table cell which contains a double-precision floating point number.

`#include <CellDouble.h>`

Inheritance diagram for CellDouble:



Public Member Functions

- **CellDouble** (double value)
- CellDouble (const char *str)
- double getNumeralValue () const override

  Returns the value of the cell as a double-precision floating point number.
- String getValueForPrint () const override

  Returns the value of the cell as a String.
- void parseAndSetValue (const char *str) override

  Parses given C-style string and stores value.
- void writeToFile (std::ofstream &file) override

  Writes value to a text file.

5.2.1 Detailed Description

A table cell which contains a double-precision floating point number.

Implements the Cell class by implementing a value of type double.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 CellDouble()

```
CellDouble::CellDouble (
            const char * str )
```

Accepts a C-style string in the form "<+/-><digits>.<digits>"

5.2.3 Member Function Documentation

5.2.3.1 getNumeralValue()

```
double CellDouble::getNumeralValue ( ) const  [override], [virtual]
```

Returns the value of the cell as a double-precision floating point number.

Implements Cell.

5.2.3.2 getValueForPrint()

```
String CellDouble::getValueForPrint ( ) const  [override], [virtual]
```

Returns the value of the cell as a String.

Implements Cell.

5.2.3.3 parseAndSetValue()

```
void CellDouble::parseAndSetValue (
            const char * str )  [override], [virtual]
```

Parses given C-style string and stores value.

Implements Cell.

5.2.3.4 writeToFile()

```
void CellDouble::writeToFile (
            std::ofstream & file )  [override], [virtual]
```

Writes value to a text file.

Implements Cell.

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

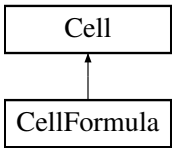
- src/Models/CellDouble.h
- src/Models/CellDouble.cpp

5.3 CellFormula Class Reference

A table cell which contains a formula.

`#include <CellFormula.h>`

Inheritance diagram for CellFormula:



Public Member Functions

- **CellFormula** (const char *str, const List< List< Cell * > > *tableCells)
- double getNumeralValue () const override
- String getValueForPrint () const override
- void parseAndSetValue (const char *str) override
- void writeToFile (std::ofstream &file) override

  Writes value to a text file.

5.3.1 Detailed Description

A table cell which contains a formula.

Implements the Cell class by storing a formula as a String and in a dynamic List

5.3.2 Member Function Documentation

5.3.2.1 getNumeralValue()

`double CellFormula::getNumeralValue ( ) const [override], [virtual]`

Value is recalculated every time this function is called (in case a referenced cell is changed).

Returns

0

Implements Cell.

5.3.2.2 getValueForPrint()

```
String CellFormula::getValueForPrint ( ) const  [override], [virtual]
```

Value is recalculated every time this function is called (in case a referenced cell is changed).

Returns

"Error"

Implements Cell.

5.3.2.3 parseAndSetValue()

```
void CellFormula::parseAndSetValue (
            const char * str )  [override], [virtual]
```

Each math operation is represented with a Pair that holds indecies of cell reference and a character for the operation.

Negatve value for the operation character means it's with priority and the result get's calculated immediately (rather than recursively, taking into account other values).

Exceptions

std::logic_error("Error	Invalid column character!") Thrown when there is something between Rx and C in cell references \exception std::logic_error("Error: Invalid value!") Thrown when met value isn't a cell reference or number \exception throw std::logic_error("Error: Could not find operand!") Thrown when there isn't an operator character between two values

Implements Cell.

5.3.2.4 writeToFile()

```
void CellFormula::writeToFile (
            std::ofstream & file )  [override], [virtual]
```

Writes value to a text file.

Implements Cell.

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

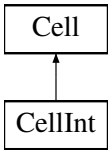
- src/Models/CellFormula.h
- src/Models/CellFormula.cpp

5.4 CellInt Class Reference

A table cell which contains an integer.

`#include <CellInt.h>`

Inheritance diagram for CellInt:



Public Member Functions

- **CellInt** (int value)
- **CellInt** (const char *str)
- double getNumeralValue () const override

  Returns the value of the cell as a double-precision floating point number.
- String getValueForPrint () const override

  Returns the value of the cell as a String.
- void parseAndSetValue (const char *str) override

  Parses given C-style string and stores value.
- void writeToFile (std::ofstream &file) override

  Writes value to a text file.

5.4.1 Detailed Description

A table cell which contains an integer.

Impelements the Cell class by implementing a value of type int

5.4.2 Member Function Documentation

5.4.2.1 getNumeralValue()

`double CellInt::getNumeralValue ( ) const [override], [virtual]`

Returns the value of the cell as a double-precision floating point number.

Implements Cell.

5.4.2.2 getValueForPrint()

```
String CellInt::getValueForPrint ( ) const  [override], [virtual]
```

Returns the value of the cell as a String.

Implements Cell.

5.4.2.3 parseAndSetValue()

```
void CellInt::parseAndSetValue (
            const char * str )  [override], [virtual]
```

Parses given C-style string and stores value.

Implements Cell.

5.4.2.4 writeToFile()

```
void CellInt::writeToFile (
            std::ofstream & file )  [override], [virtual]
```

Writes value to a text file.

Implements Cell.

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

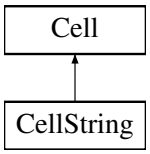
- src/Models/CellInt.h
- src/Models/CellInt.cpp

5.5 CellString Class Reference

A table cell which contains a String value.

```
#include <CellString.h>
```

Inheritance diagram for CellString:



Public Member Functions

- **CellString** (const char *str)
- double getNumeralValue () const override

  Returns the value of the cell as a double-precision floating point number.
- String getValueForPrint () const override

  Returns the value of the cell as a String.
- void parseAndSetValue (const char *str) override

  Parses given C-style string and stores value.
- void writeToFile (std::ofstream &file) override

  Writes value to a text file.

5.5.1 Detailed Description

A table cell which contains a String value.

5.5.2 Member Function Documentation

5.5.2.1 getNumeralValue()

```
double CellString::getNumeralValue ( ) const  [override], [virtual]
```

Returns the value of the cell as a double-precision floating point number.

Implements Cell.

5.5.2.2 getValueForPrint()

```
String CellString::getValueForPrint ( ) const  [override], [virtual]
```

Returns the value of the cell as a String.

Implements Cell.

5.5.2.3 parseAndSetValue()

```
void CellString::parseAndSetValue (
            const char * str )  [override], [virtual]
```

Parses given C-style string and stores value.

Implements Cell.

5.5.2.4 writeToFile()

```
void CellString::writeToFile (
            std::ofstream & file )  [override], [virtual]
```

Writes value to a text file.

Implements Cell.

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

- src/Models/CellString.h
- src/Models/CellString.cpp

5.6 Command Class Reference

Stores a 256 character name and a function pointer to be executed when calling run()

```
#include <Command.h>
```

Public Member Functions

- **Command** ()
  Leaves name empty and exec function pointer to nullptr.
- Command (const char *name, void(*exec)(const char *params))
  Copies contents of nameInMenu and stores exec.
- void run (const char *params) const
  Executes the stored function pointer.
- const char * **get_name** () const

5.6.1 Detailed Description

Stores a 256 character name and a function pointer to be executed when calling run()

Menu uses this class for a more generic implementation of it's navigate function.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Command()

```
Command::Command (
            const char * name,
            void(*)(const char *params) exec )
```

Copies contents of nameInMenu and stores exec.

Parameters

name	C-style string, class stores at most 255 characters (last character is always terminating zero)
exec	Function pointer that will be executed when run() is called. Can be nullptr.

5.6.3 Member Function Documentation

5.6.3.1 run()

```
void Command::run (
            const char * params ) const
```

Executes the stored function pointer.

Executes the stored function pointer, when it's not nullptr. Otherwise does nothing.

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

- src/Generic/ConsoleInterface/Command.h
- src/Generic/ConsoleInterface/Command.cpp

5.7 List< T > Class Template Reference

Templated class that stores an array of elements in dynamic memory.

```
#include <List.hpp>
```

Public Member Functions

- **List** (const T *elements, unsigned elementsCount)
  Copies all elements in given array.
- **List** (unsigned length)
  Allocates memory for the given length.
- void add (const T &element)
  Adds an element.
- void insertAt (const T &element, unsigned index)
  Inserts element at given index.
- bool removeAt (unsigned index)
  Removes element at index.
- unsigned findIndex (const T &element) const
  Finds the index of element.
- bool contain (const T &element) const
- T & **operator[ ]** (unsigned index)
  Returns reference to element at index.

- const T & **operator[ ]** (unsigned index) const
  Returns constant reference to element at index.
- const T ∗ **raw_data** () const
  Returns raw array of data.
- void **clear** ()
  Removes all data.
- List< T > & **operator+=** (const List< T > other)
  Appends elements from other list.
- std::istream & read (std::istream &istr)
  Reads from stream.
- std::ostream & write (std::ostream &ostr) const
  Writes to stream.
- unsigned get_length () const
  Returns the length.
- unsigned get_count () const
  Returns the count.
- List & **operator=** (const List &other)
- **List** (const List &other)
- **List** (List &&other)
- List & **operator=** (List &&other)

Protected Member Functions

- void **resize** ()
- void **free** ()
- void **copyFrom** (const List &other)

Protected Attributes

- T ∗ **elements**
- unsigned **length**
- unsigned **count**

5.7.1 Detailed Description

template<typename T>
class List< T >

Templated class that stores an array of elements in dynamic memory.

Warning

findIndex(), contain(), read(), write(), operator<<() and operator>>() require the type to have defined a couple of functions.

See also

findIndex()

contain()

read()

write()

operator<<()

operator>>()

5.7.2 Member Function Documentation

5.7.2.1 add()

```
template<typename T >
void List< T >::add (
            const T & element )
```

Adds an element.

Resizes internal array if there is no space for additional elements.

5.7.2.2 contain()

```
template<typename T >
bool List< T >::contain (
            const T & element ) const
```

Returns

Whether the element is contained in the current List

Warning

Function depends on findIndex(), which means the same "compare" function must be defined in the type

See also

findIndex()

5.7.2.3 findIndex()

```
template<typename T >
unsigned List< T >::findIndex (
            const T & element ) const
```

Finds the index of element.

Returns

Index of element. If element isn't found, returns the count of element.

Warning

The function depends on the type having a function "compare" defined, which takes two elements and returns a number <0 if elem1 < elem2, >0 if elem1 > elem2, 0 if elem1 == elem2

Note

Searching is done linearly

5.7.2.4 get_count()

```
template<typename T >
unsigned List< T >::get_count
```

Returns the count.

Count is the amount of elements that are stored.

5.7.2.5 get_length()

```
template<typename T >
unsigned List< T >::get_length
```

Returns the length.

Length is the size of the underlying array (allocated memory).

5.7.2.6 insertAt()

```
template<typename T >
void List< T >::insertAt (
            const T & element,
            unsigned index )
```

Inserts element at given index.

If index is after the last element, the element is just added. Otherwise all elements after the index are shifted right and element is put in place.

Resizes internal array if there is no space for additional elements.

5.7.2.7 read()

```
template<typename T >
std::istream & List< T >::read (
            std::istream & istr )
```

Reads from stream.

Parameters

istr	An input stream

Directly reads bytes from stream (calls read() function). Any stored values are deleted and replaced with those from the stream.

Warning

The function depends on the type having a function "read" defined, which takes an std::istream& and writes it's data to it. Return type doesn't matter.

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ifstream

5.7.2.8 removeAt()

```
template<typename T >
bool List< T >::removeAt (
            unsigned index )
```

Removes element at index.

Returns

Wether element could be removed

If index is after that of the last element, nothing is done and false is returned. Otherwise elements after index are shifted right and count is reduced.

5.7.2.9 write()

```
template<typename T >
std::ostream & List< T >::write (
            std::ostream & ostr ) const
```

Writes to stream.

Parameters

ostr	An output stream

Directly writes bytes to stream (calls write() function).

Warning

The function depends on the type having a function "write" defined, which takes an std::ostream& and writes it's data to it. Return type doesn't matter.

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ofstream

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

- src/Generic/List/List.hpp

5.8 Menu Class Reference

Handles navigation between multiple commands.

`#include <Menu.h>`

Public Member Functions

- **Menu** ()

  Sets name as "Menu", leaves all flags to false and creates an empty Command list.
- **Menu** (const Command *commands, unsigned commandCount)

  Copies commands.
- void **addCommand** (const Command &command)

  Adds a command to the internal Command list.
- void navigate () const

  Lists commands and after user input executes an appropriate command run() function.

5.8.1 Detailed Description

Handles navigation between multiple commands.

Shows the title, then below it an Error/Warning/Success message if set, lists all command options as an enumerated list, takes command index and calls the chosen command's run() function.

5.8.2 Member Function Documentation

5.8.2.1 navigate()

```
void Menu::navigate ( ) const
```

Lists commands and after user input executes an appropriate command run() function.

Prints the title, then an Error/Warning/Sucess message (if set), prints all command names as an ordered list (starting from 1), waits for user input to select one of those commands (by list number) and finally executes the appropriate command's run() function.

After the run() function exists, everything is reprinted. The 0 list index is always "Go Back"/"Exit" and it always stops the reprinting loop.

If there are no commands, prints "Menu is empty!". If user input doesn't correspond to any command, registers an error message and reprints.

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

- src/Generic/ConsoleInterface/Menu.h
- src/Generic/ConsoleInterface/Menu.cpp

5.9 Pair< T, U > Struct Template Reference

Stores two values of any types.

`#include <Pair.hpp>`

Public Member Functions

- **Pair** (const T &left, const U &right)

Public Attributes

- T **left**
- U **right**

5.9.1 Detailed Description

template<typename T, typename U>
struct Pair< T, U >

Stores two values of any types.

Values are public, so no explicit getters or setters are made.

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

- src/Generic/Pair/Pair.hpp

5.10 String Class Reference

A dynamically-allocated C-style string with extra features.

`#include <String.h>`

Public Member Functions

- String (unsigned length)

  The underlying array is allocated with length (+ 1) size.
- String (const char ∗str)

  Copies str.
- const char ∗ **get_cstr** () const

  Returns internal C-style string.
- unsigned **get_length** () const

  Returns number of characters in string.
- char & **operator[ ]** (unsigned index)

  Returns a character reference (from the underlying C-style string) at the given index.
- String & operator+= (const char ∗str)

  Appends a C-style string.
- String & operator+= (int number)

  Appends a number.
- String & operator+= (unsigned number)

  Appends a number.
- String & operator+= (double number)

  Appends a number.
- **String** (const String &other)
- String & **operator=** (const String &other)
- **String** (String &&other)
- String & **operator=** (String &&other)
- void read (std::istream &istr)

  Reads String from stream.
- void write (std::ostream &ostr) const

  Writes String to stream.
- int compare (const String &other) const

  Compares two strings.

Friends

- std::istream & operator>> (std::istream &istr, String &str)

  Reads String from stream with >> operator.
- std::ostream & operator<< (std::ostream &ostr, const String &str)

  Writes String to stream with << operator.

5.10.1 Detailed Description

A dynamically-allocated C-style string with extra features.

Remarks

There is no buffer allocation, meaning each String takes only the amount of memory it needs and every append operation juggles around memory.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 String() [1/2]

```
String::String (
            unsigned length )
```

The underlying array is allocated with length (+ 1) size.

Allocated with length + 1 size, so there is a terminating zero at the end.

5.10.2.2 String() [2/2]

```
String::String (
            const char * str )
```

Copies str.

\params str C-style string

5.10.3 Member Function Documentation

5.10.3.1 compare()

```
int String::compare (
            const String & other ) const
```

Compares two strings.

\params other C-style string

Returns

strcmp between underlying C-style string and "other"

5.10.3.2 operator+=() [1/4]

```
String & String::operator+= (
            const char * str )
```

Appends a C-style string.

\params str C-style string

5.10.3.3 operator+=() [2/4]

```
String & String::operator+= (
            double number )
```

Appends a number.

Converts the number to a C-style string and then uses += to append it.

5.10.3.4 operator+=() [3/4]

```
String & String::operator+= (
            int number )
```

Appends a number.

Converts the number to a C-style string and then uses += to append it.

5.10.3.5 operator+=() [4/4]

```
String & String::operator+= (
            unsigned number )
```

Appends a number.

Converts the number to a C-style string and then uses += to append it.

5.10.3.6 read()

```
void String::read (
            std::istream & istr )
```

Reads String from stream.

Parameters

istr	An input stream

Directly reads bytes from stream (calls read() function). First reads the string length, then the underlying C-style string (including terminating zero).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ifstream

5.10.3.7 write()

```
void String::write (
            std::ostream & ostr ) const
```

Writes String to stream.

Parameters

ostr	An output stream

Directly writes bytes to stream (calls write() function). First writes the string length, then the underlying C-style string (including terminating zero).

Remarks

Doesn't alter the stream in any other way.

Note

Best used with binary ofstream

5.10.4 Friends And Related Function Documentation

5.10.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & ostr,
            const String & str )  [friend]
```

Writes String to stream with << operator.

Uses the stream's << operator to write the underlying C-style string

Note

Best used with std::cout or text std::ofstream

5.10.4.2 operator>>

```
std::istream & operator>> (
            std::istream & istr,
            String & str )  [friend]
```

Reads String from stream with >> operator.

Uses the stream's getline function to read the data.

Warning

It takes at most 1024 characters from the stream!

Note

Best used with std::cin or text std::ifstream

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

- src/Generic/String/String.h
- src/Generic/String/String.cpp

5.11 Table Class Reference

A 2D collection of Cell instances.

`#include <Table.h>`

Public Member Functions

- **Table** (const char *filePath)
- unsigned **get_rows** () const

  Returns count of rows.
- unsigned **get_cols** () const

  Returns largest count of columns in all rows.
- void **putCell** (unsigned row, unsigned col, const char *rawValue)

  Parses rawValue and replaces the value at row and col with it.
- List< String > **getAllCells** () const

  Returns a List of String with String representation of every cell in order.
- void **saveToFile** (const char *filePath) const

  Saves all cells to a text file.

5.11.1 Detailed Description

A 2D collection of Cell instances.

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

- src/Models/Table.h
- src/Models/Table.cpp

5.12 UserInterface Class Reference

The class that contains and via which the user controls the application's user interface.

`#include <UserInterface.h>`

Public Member Functions

- void **run** ()

  Starts the user interface.

5.12.1 Detailed Description

The class that contains and via which the user controls the application's user interface.

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

- src/UserInterface/UserInterface.h
- src/UserInterface/UserInterface.cpp

Chapter 6

File Documentation

6.1 src/Generic/ConsoleInterface/Command.h File Reference

Stores the declaration of class Command.

Classes

- class Command

  Stores a 256 character name and a function pointer to be executed when calling run()

6.1.1 Detailed Description

Stores the declaration of class Command.

6.2 Command.h

Go to the documentation of this file.

```
1 #ifndef HEADER_CONSOLEINTERFACE_COMMAND
2 #define HEADER_CONSOLEINTERFACE_COMMAND
3
12 class Command {
13     char name[256];
14     void (*exec)(const char* params);
15
16 public:
18     Command();
20     Command(const char* name, void (*exec)(const char* params));
22     void run(const char* params) const;
23
24     const char* get_name() const;
25 };
26
27 #endif
```

6.3 src/Generic/ConsoleInterface/Menu.h File Reference

Stores the declaration of class Menu.

```
#include "Command.h"
#include "../List/List.hpp"
```

Classes

- class Menu

  Handles navigation between multiple commands.

6.3.1 Detailed Description

Stores the declaration of class Menu.

6.4 Menu.h

Go to the documentation of this file.

```
1 #ifndef HEADER_CONSOLEINTERFACE_MENU
2 #define HEADER_CONSOLEINTERFACE_MENU
3
8 #include "Command.h"
9 #include "../List/List.hpp"
10
17 class Menu {
18     List<Command> menuOptions;
19
20 public:
22     Menu();
24     Menu(const Command* commands, unsigned commandCount);
25
27     void addCommand(const Command& command);
28
30     void navigate() const;
31 };
32
33 #endif
```

6.5 src/Generic/ConsoleInterface/Toolbox.hpp File Reference

Stores a wide range of functions for simpler/more automated printing.

```
#include <iostream>
#include "../List/List.hpp"
#include "../String/String.h"
```

Macros

- #define **MAX_LINE_WIDTH** 1024

Functions

- void inputLineBox (char *output, unsigned maxWidth, bool ignore=true)

  Prints label, gets a whole line of input and stores it to output.
- void printTable (const List< String > &items, unsigned columns)

  Prints a string list as a table.
- template<typename T >
  void **read** (T *storage)

  Reads user input and stores it.
- template<typename T >
  void **read** (T &storage)

  Reads user input and stores it.
- template<typename T >
  void **print** (const T *item)

  Prints given item.
- template<typename T >
  void **print** (const T &item)

  Prints given item.
- template<typename T >
  void **printLine** (const T *item)

  Prints given item and an endline character.
- template<typename T >
  void **printLine** (const T &item)

  Prints given item and an endline character.

6.5.1 Detailed Description

Stores a wide range of functions for simpler/more automated printing.

Adds a lot of functions for printing.

Remarks

iostream is included by necessity (templated functions), you should only use the provided functions, if you can.

6.5.2 Function Documentation

6.5.2.1 inputLineBox()

```
void inputLineBox (
            char * output,
            unsigned maxWidth,
            bool ignore )
```

Prints label, gets a whole line of input and stores it to output.

Parameters

label	C-style string, there are no size check so it could wrap
output	Pointer to a char array
maxWidth	Maximum count of characters to read from user input. output MUST be able to hold that many characters!
ignore	Whether or not to ignore the first new-line delimiter. true by default, should be set to false only when an inputLineBox/SubBox has been issued directly prior or when it's the very first issued box command.

Prints "(+)" before the label.

6.5.2.2 printTable()

```
void printTable (
            const List< String > & items,
            unsigned columns )
```

Prints a string list as a table.

Parameters

startNumber	The number by which column and row enumeration begins
columns	How many columns the table should have
items	C-style string, each cell is a single character from the string

Prints a string as a grid/table of characters from top to bottom, left to right. The first character is on the top left, the last one is on the bottom right.

Table is printed until a terminating zero is encountered. The rows are "calculated" from the columns count and the items length.

6.6 Toolbox.hpp

Go to the documentation of this file.

```
1 #ifndef HEADER_CONSOLEINTERFACE_TOOLBOX
2 #define HEADER_CONSOLEINTERFACE_TOOLBOX
3
4 #include <iostream>
5 #include "../List/List.hpp"
6 #include "../String/String.h"
7
16 #define MAX_LINE_WIDTH 1024
17
19 void inputLineBox(char* output, unsigned maxWidth, bool ignore = true);
20
22 void printTable(const List<String>& items, unsigned columns);
23
25 template <typename T>
26 void read(T* storage) {
27     std::cin » storage;
28 }
29
31 template <typename T>
32 void read(T& storage) {
33     std::cin » storage;
34 }
35
```

```
37 template <typename T>
38 void print(const T* item) {
39     std::cout « item;
40 }
41
43 template <typename T>
44 void print(const T& item) {
45     std::cout « item;
46 }
47
49 template <typename T>
50 void printLine(const T* item) {
51     std::cout « item « std::endl;
52 }
53
55 template <typename T>
56 void printLine(const T& item) {
57     std::cout « item « std::endl;
58 }
59
60 #endif
```

6.7 src/Generic/List/List.hpp File Reference

Stores declaration and definition of templated class List.

```
#include <istream>
#include <ostream>
```

Classes

- class List< T >

  Templated class that stores an array of elements in dynamic memory.

Functions

- template<typename T >
  std::istream & operator>> (std::istream &istr, List< T > &obj)

  Reads List from stream with >> operator.
- template<typename T >
  std::ostream & operator<< (std::ostream &ostr, const List< T > &obj)

  Writes List to stream with << operator.

6.7.1 Detailed Description

Stores declaration and definition of templated class List.

6.7.2 Function Documentation

6.7.2.1 operator<<()

```
template<typename T >
std::ostream & operator<< (
            std::ostream & ostr,
            const List< T > & obj )
```

Writes List to stream with << operator.

Uses the stream's << operator to write the count and then all objects.

Warning

The function depends on the type having the operator << defined, which takes an std::ostream& and writes it's data to it. Return type doesn't matter.

Note

Best used with std::cout or text std::ofstream

6.7.2.2 operator>>()

```
template<typename T >
std::istream & operator>> (
            std::istream & istr,
            List< T > & obj )
```

Reads List from stream with >> operator.

Uses the stream's >> operator to read and parse the elements. The first item in the stream should be the count.

Warning

The function depends on the type having the operator >> defined, which takes an std::istream& and writes it's data to it. Return type doesn't matter.

Note

Best used with std::cin or text std::ifstream

6.8 List.hpp

Go to the documentation of this file.

```
1 #ifndef HEADER_LIST
2 #define HEADER_LIST
3
8 #include <istream>
9 #include <ostream>
10
21 template <typename T>
22 class List {
23 protected:
24     T* elements;
25     unsigned length;
26     unsigned count;
27
28     void resize();
29     void free();
30     void copyFrom(const List& other);
31
32 public:
34     List(const T* elements, unsigned elementsCount);
36     List(unsigned length);
38     void add(const T& element);
40     void insertAt(const T& element, unsigned index);
42     bool removeAt(unsigned index);
44     unsigned findIndex(const T& element) const;
45     bool contain(const T& element) const;
47     T& operator[](unsigned index);
49     const T& operator[](unsigned index) const;
51     const T* raw_data() const;
53     void clear();
54
56     List<T>& operator+=(const List<T> other);
57
59     std::istream& read(std::istream& istr);
61     std::ostream& write(std::ostream& ostr) const;
62
64     unsigned get_length() const;
66     unsigned get_count() const;
67
68     List();
69     List& operator=(const List& other);
70     List(const List& other);
71     ~List();
72
73     List(List&& other);
74     List& operator=(List&& other);
75 };
76
78 template <typename T>
79 std::istream& operator»(std::istream& istr, List<T>& obj);
81 template <typename T>
82 std::ostream& operator«(std::ostream& ostr, const List<T>& obj);
83
84 /* Private */
85
86 template <typename T>
87 void List<T>::resize() {
88     length = (length == 0) ? 8 : length « 1;
89     T* temp = new T[length];
90     for (int i = 0; i < count; i++)
91         temp[i] = elements[i];
92     delete[] elements;
93     elements = temp;
94 }
95
96 template <typename T>
97 void List<T>::free() {
98     delete[] elements;
99 }
100
101 template <typename T>
102 void List<T>::copyFrom(const List& other) {
103     elements = new T[other.length];
104     for (int i = 0; i < other.count; i++)
105         elements[i] = other.elements[i];
106     length = other.length;
107     count = other.count;
108 }
109
110 /* Public */
111
112 template <typename T>
113 List<T>::List(const T* elements, unsigned elementsCount) {
```

```
114     length = 8;
115     count = elementsCount;
116     this->elements = new T[length];
117     for (unsigned i = 0; i < count; i++)
118         this->elements[i] = elements[i];
119 }
120
121 template <typename T>
122 List<T>::List(unsigned length) {
123     this->length = count = length;
124     this->elements = new T[length];
125     for (unsigned i = 0; i < count; i++)
126         this->elements[i] = elements[i];
127 }
128
132 template <typename T>
133 void List<T>::add(const T& element) {
134     if (length == count) resize();
135
136     elements[count++] = element;
137 }
138
145 template <typename T>
146 void List<T>::insertAt(const T& element, unsigned index) {
147     if (index >= count) {
148         add(element);
149         return;
150     }
151     if (length == count) resize();
152
153     for (unsigned i = count; i > index; i--)
154         elements[i] = elements[i - 1];
155
156     elements[index] = element;
157     count++;
158 }
159
165 template <typename T>
166 bool List<T>::removeAt(unsigned index) {
167     if (index >= count) return false;
168
169     for (int i = index; i < count; i++)
170         elements[i] = elements[i + 1];
171     count--;
172     return true;
173 }
174
180 template <typename T>
181 unsigned List<T>::findIndex(const T& element) const {
182     unsigned ind = 0;
183     while (ind < count && elements[ind].compare(element) != 0)
184         ind++;
185     return ind;
186 }
187
193 template <typename T>
194 bool List<T>::contain(const T& element) const {
195     return findIndex(element) < count;
196 }
197
198 template <typename T>
199 T& List<T>::operator[](unsigned index) {
200     return elements[index];
201 }
202
203 template <typename T>
204 const T& List<T>::operator[](unsigned index) const {
205     return elements[index];
206 }
207
208 template <typename T>
209 const T* List<T>::raw_data() const {
210     return elements;
211 }
212
213 template <typename T>
214 void List<T>::clear() {
215     free();
216     length = 8;
217     count = 0;
218     this->elements = new T[length];
219 }
220
221 template <typename T>
222 List<T>& List<T>::operator+=(const List<T> other) {
223     for (unsigned i = 0; i < other.length; i++)
224         add(other[i]);
```

```
225     return *this;
226 }
227
237 template <typename T>
238 std::istream& List<T>::read(std::istream& istr) {
239     istr.read((char*)&length, sizeof(length));
240     istr.read((char*)&count, sizeof(count));
241
242     delete[] elements;
243     elements = new T[length];
244
245     for (int i = 0; i < count; i++)
246         elements[i].read(istr);
247
248     return istr;
249 }
250
259 template <typename T>
260 std::ostream& List<T>::write(std::ostream& ostr) const {
261     ostr.write((const char*)&length, sizeof(length));
262     ostr.write((const char*)&count, sizeof(count));
263
264     for (int i = 0; i < count; i++)
265         elements[i].write(ostr);
266
267     return ostr;
268 }
269
273 template <typename T>
274 unsigned List<T>::get_length() const {
275     return length;
276 }
277
281 template <typename T>
282 unsigned List<T>::get_count() const {
283     return count;
284 }
285
286 // Rule of 4
287
288 template <typename T>
289 List<T>::List() : List(nullptr, 0) {}
290
291 template <typename T>
292 List<T>& List<T>::operator=(const List& other) {
293     if (this != &other) {
294         free();
295         copyFrom(other);
296     }
297     return *this;
298 }
299
300 template <typename T>
301 List<T>::List(const List& other) {
302     copyFrom(other);
303 }
304
305 template <typename T>
306 List<T>::~List() {
307     free();
308 }
309
310 // Move semantics
311
312 template <typename T>
313 List<T>::List(List&& other) {
314     length = other.length;
315     count = other.count;
316     elements = other.elements;
317     other.elements = nullptr;
318 }
319
320 template <typename T>
321 List<T>& List<T>::operator=(List&& other) {
322     if (this != &other) {
323         free();
324         length = other.length;
325         count = other.count;
326         elements = other.elements;
327         other.elements = nullptr;
328     }
329     return *this;
330 }
331
332 /* Outside of class */
333
341 template <typename T>
```

```
342 std::istream& operator»(std::istream& istr, List<T>& obj) {
343     List<T> newObj;
344     unsigned count;
345     istr » count;
346 
347     T temp;
348     for (int i = 0; i < count; i++) {
349         istr » temp;
350         obj.add(temp);
351     }
352 
353     return istr;
354 }
355 
362 template <typename T>
363 std::ostream& operator«(std::ostream& ostr, const List<T>& obj) {
364     ostr « obj.get_count() « std::endl;
365     for (int i = 0; i < obj.get_count(); i++)
366         ostr « obj[i];
367 
368     return ostr;
369 }
370 
371 
372 #endif
```

6.9 src/Generic/Pair/Pair.hpp File Reference

Stores the declaration of struct Pair.

Classes

- struct Pair< T, U >

  Stores two values of any types.

6.9.1 Detailed Description

Stores the declaration of struct Pair.

6.10 Pair.hpp

Go to the documentation of this file.

```
1 #ifndef PAIR
2 #define PAIR
3 
12 template <typename T, typename U>
13 struct Pair {
14     T left;
15     U right;
16 
17     Pair();
18     Pair(const T& left, const U& right);
19 };
20 
21 template <typename T, typename U>
22 Pair<T, U>::Pair() : left(), right() {}
23 
24 
25 template <typename T, typename U>
26 Pair<T, U>::Pair(const T& left, const U& right) : left(left), right(right) {}
27 
28 #endif
```

6.11 src/Generic/String/String.h File Reference

Stores declaration of class String.

```
#include <istream>
#include <ostream>
```

Classes

- class String

  A dynamically-allocated C-style string with extra features.

Macros

- #define **DOUBLE_PRECISION** 100

6.11.1 Detailed Description

Stores declaration of class String.

6.12 String.h

Go to the documentation of this file.

```
1 #ifndef HEADER_STRING
2 #define HEADER_STRING
3
8 #include <istream>
9 #include <ostream>
10
11 #define DOUBLE_PRECISION 100
12
17 class String {
18     char* str;
19     unsigned length;
20
21     void free();
22     void copyFrom(const String& other);
23
24 public:
26     String(unsigned length);
28     String(const char* str);
30     const char* get_cstr() const;
32     unsigned get_length() const;
34     char& operator[](unsigned index);
35
37     String& operator+=(const char* str);
39     String& operator+=(int number);
41     String& operator+=(unsigned number);
43     String& operator+=(double number);
44
45     String();
46     String(const String& other);
47     String& operator=(const String& other);
48     ~String();
49
50     String(String&& other);
51     String& operator=(String&& other);
52
54     void read(std::istream& istr);
56     void write(std::ostream& ostr) const;
58     int compare(const String& other) const;
59
61     friend std::istream& operator»(std::istream& istr, String& str);
63     friend std::ostream& operator«(std::ostream& ostr, const String& str);
64 };
65
66 #endif
```

6.13 src/main.cpp File Reference

Creates a defualt instance of UserInterface and runs it.

```
#include "UserInterface/UserInterface.h"
```

Functions

- int **main** ()

6.13.1 Detailed Description

Creates a defualt instance of UserInterface and runs it.

6.14 src/Models/Cell.h File Reference

Stores the declaration of class Cell.

```
#include <fstream>
#include "../Generic/String/String.h"
```

Classes

- class Cell

  Fully abstract class that represents a cell in the table.

6.14.1 Detailed Description

Stores the declaration of class Cell.

6.15 Cell.h

Go to the documentation of this file.

```
1 #ifndef HEADER_CELL
2 #define HEADER_CELL
3
8 #include <fstream>
9 #include "../Generic/String/String.h"
10
13 class Cell {
14 public:
16     virtual double getNumeralValue() const = 0;
18     virtual String getValueForPrint() const = 0;
20     virtual void parseAndSetValue(const char* str) = 0;
22     virtual void writeToFile(std::ofstream& file) = 0;
23
24     virtual ~Cell() = default;
25 };
26
27 #endif
```

6.16 src/Models/CellDouble.h File Reference

Stores the declaration of class CellDouble.

```
#include "Cell.h"
```

Classes

- class CellDouble

  A table cell which contains a double-precision floating point number.

6.16.1 Detailed Description

Stores the declaration of class CellDouble.

6.17 CellDouble.h

Go to the documentation of this file.

```
1 #ifndef HEADER_CELLDOUBLE
2 #define HEADER_CELLDOUBLE
3
8 #include "Cell.h"
9
14 class CellDouble : public Cell {
15     double value;
16
17 public:
18     CellDouble() = default;
19     CellDouble(double value);
20     CellDouble(const char* str);
21
22     double getNumeralValue() const override;
23     String getValueForPrint() const override;
24     void parseAndSetValue(const char* str) override;
25     void writeToFile(std::ofstream& file) override;
26 };
27
28 #endif
```

6.18 src/Models/CellFormula.h File Reference

Stores the declaration of class CellFormula.

```
#include "Cell.h"
#include "CellDouble.h"
#include "../Generic/List/List.hpp"
#include "../Generic/Pair/Pair.hpp"
```

Classes

- class CellFormula

  A table cell which contains a formula.

6.18.1 Detailed Description

Stores the declaration of class CellFormula.

6.19 CellFormula.h

Go to the documentation of this file.

```
1 #ifndef HEADER_CELLFORMULA
2 #define HEADER_CELLFORMULA
3
8 #include "Cell.h"
9 #include "CellDouble.h"
10 #include "../Generic/List/List.hpp"
11 #include "../Generic/Pair/Pair.hpp"
12
17 class CellFormula : public Cell {
18     const List<List<Cell*>>* tableCells;
19     List<CellDouble> localCells;
20
21     List<Pair<Pair<int, int>, char» formula;
22     String rawFormula;
23
24     const Cell* ptrByInd(Pair<int, int> loc) const;
25     double calculate(unsigned index = 0) const;
26
27 public:
28     CellFormula() = default;
29     CellFormula(const char* str, const List<List<Cell*>>* tableCells);
30
31     double getNumeralValue() const override;
32     String getValueForPrint() const override;
33     void parseAndSetValue(const char* str) override;
34     void writeToFile(std::ofstream& file) override;
35 };
36
37 #endif
```

6.20 src/Models/CellInt.h File Reference

Stores the declaration of class Cell.

```
#include "Cell.h"
```

Classes

- class CellInt

  A table cell which contains an integer.

6.20.1 Detailed Description

Stores the declaration of class Cell.

6.21 CellInt.h

Go to the documentation of this file.

```
1 #ifndef HEADER_CELLINT
2 #define HEADER_CELLINT
3
8 #include "Cell.h"
9
14 class CellInt : public Cell {
15     int value;
16
17 public:
18     CellInt() = default;
19     CellInt(int value);
20     CellInt(const char* str);
21
22     double getNumeralValue() const override;
23     String getValueForPrint() const override;
24     void parseAndSetValue(const char* str) override;
25     void writeToFile(std::ofstream& file) override;
26 };
27
28 #endif
```

6.22 src/Models/CellString.h File Reference

Stores the declaration of class CellString.

```
#include "Cell.h"
```

Classes

- class CellString

  A table cell which contains a String value.

6.22.1 Detailed Description

Stores the declaration of class CellString.

6.23 CellString.h

Go to the documentation of this file.

```
1 #ifndef HEADER_CELLSTRING
2 #define HEADER_CELLSTRING
3
8 #include "Cell.h"
9
12 class CellString : public Cell {
13     String value;
14
15 public:
16     CellString() = default;
17     CellString(const char* str);
18
19     double getNumeralValue() const override;
20     String getValueForPrint() const override;
21     void parseAndSetValue(const char* str) override;
22     void writeToFile(std::ofstream& file) override;
23 };
24
25 #endif
```

6.24 src/Models/Table.h File Reference

Stores the declaration of class Table.

```
#include "Cell.h"
#include "../Generic/List/List.hpp"
#include <fstream>
```

Classes

- class Table

  A 2D collection of Cell instances.

6.24.1 Detailed Description

Stores the declaration of class Table.

6.25 Table.h

Go to the documentation of this file.

```
1 #ifndef MODELS_TABLE
2 #define MODELS_TABLE
3
8 #include "Cell.h"
9 #include "../Generic/List/List.hpp"
10 #include <fstream>
11
14 class Table {
15     List<List<Cell*>> cells;
16     unsigned longestRow;
17     void readFromFile(std::ifstream& inFile);
18
19 public:
20     Table(const char* filePath);
21     ~Table();
22
24     unsigned get_rows() const;
26     unsigned get_cols() const;
27
29     void putCell(unsigned row, unsigned col, const char* rawValue);
30
32     List<String> getAllCells() const;
34     void saveToFile(const char* filePath) const;
35 };
36
37 #endif
```

6.26 src/UserInterface/UserInterface.h File Reference

Stores the declaration of class UserInterface.

```
#include "../Generic/ConsoleInterface/Menu.h"
#include "../Generic/String/String.h"
#include "../Models/Table.h"
```

Classes

- class UserInterface

  The class that contains and via which the user controls the application's user interface.

6.26.1 Detailed Description

Stores the declaration of class UserInterface.

6.27 UserInterface.h

Go to the documentation of this file.

```
1 #ifndef HEADER_USERINTERFACE
2 #define HEADER_USERINTERFACE
3
8 #include "../Generic/ConsoleInterface/Menu.h"
9 #include "../Generic/String/String.h"
10 #include "../Models/Table.h"
11
14 class UserInterface {
15     Menu menu;
16
17     static Table* table;
18     static String fileName;
19     static bool saved;
20
22     static void com_open(const char* params);
24     static void com_close(const char* params);
26     static void com_save(const char* params);
28     static void com_saveas(const char* params);
30     static void com_help(const char* params);
32     static void com_exit(const char* params);
33
35     static void com_print(const char* params);
37     static void com_edit(const char* params);
38
39 public:
40     UserInterface();
42     void run();
43 };
44
45 #endif
```