ETH zürich

FS 2020

Software Design Specifications

for

CAS

Version 0.2 approved

Prepared by L.Autunno, L.Knirsch, N.Baumann

Group K

01.04.2020

Table of Contents

Static Modeling **1**
Package CAS with Priority <3/3> 1
Class Algebra 1
Class Storage **2**
Class Constant **3**
Class Help **3**
Class Diagram of Package CAS 3
Package <Name> with Priority <x/3> 4
Composite Structure Diagram 4
Dynamic Modeling **5**
Sequence Diagrams 5
Storage of variables and multiplication 5
Nested expressions and exit 6
Trigonometric expressions 7
Error message and variable storage **8**

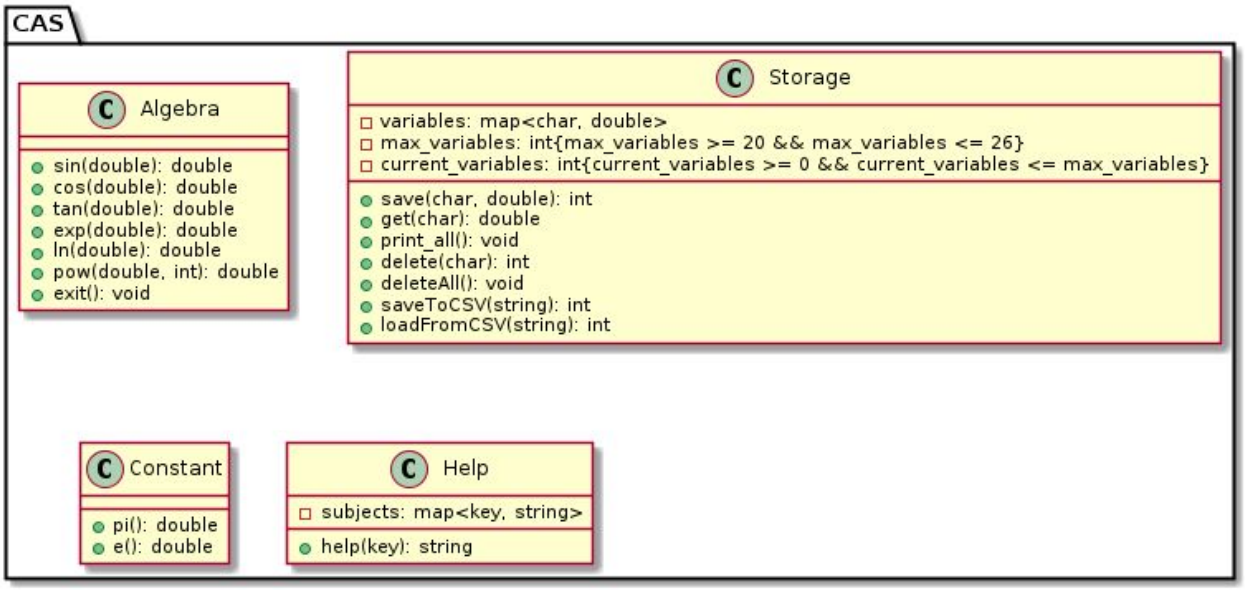
Revision History

Name	Date	Release Description	Version
Ioannis Athanasiadis	04/01/20	Adaptation of Karl E. Wiegers template for Software Engineering Course in ETHZ.	0.1
L.Autunno, L.Knirsch, N.Baumann	01/04/20	Writing first Version of SDS	0.2
L.Autunno, L.Knirsch, N.Baumann	28/04/20	Fixing from Feedback of GroupF	0.2.1

1. Static Modeling

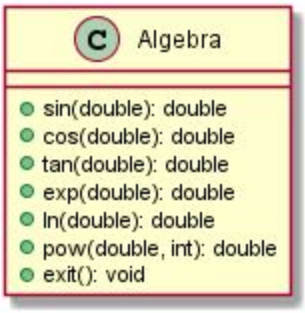
1.1 Package CAS with Priority <3/3>

This package represents the whole CAS set. There is only one package.



1.1.1 Class Algebra

Algebra is a class which only has methods. These methods are used after parsing, if a user calls them.



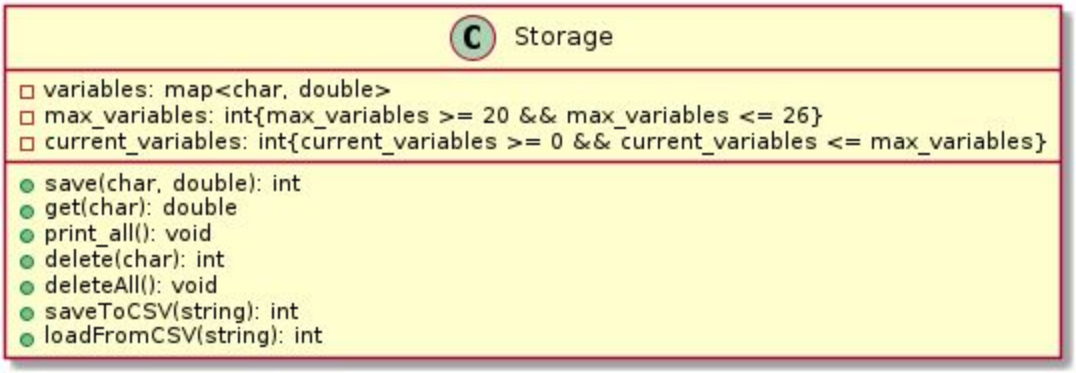
The class operations are:

- sin(double): double
  - This method returns the sine value of a number.
- cos(double): double
  - This method returns the cosine value of a number.
- tan(double): double
  - This method returns the tangent value of a number.
- exp(double): double
  - This method returns the exponential value of a number.
- ln(double): double

- This method returns the natural logarithm value of a number.
- pow(double, int): double
  - This method returns the value of a number to the power of another number.
- exit(): void
  - This method stops the program. If there are stored variables, the program asks if the user wishes to save the variables.

1.1.2 Class Storage

The class storage is used mainly for saving/loading data.



The class attributes are:

- variables: map<char, double>
  - This maps a char (key) to the corresponding double.
  - For chars please use small letters.
- max_variables: int{max_variables >= 20 && max_variables <= 26}
  - Maximal number of variables that can be stored at the same time.
  - max_variables has to be initialized in that range, because we want to accept at least 20 variables.
  - 20 is default.
  - If more wanted, use at most 26, because the alphabet goes up to 26.
- current_variables: int{current_variables >= 0 && current_variables <= max_variables}
  - Number of variables that are currently stored

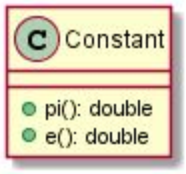
The class operations are:

- save(char, double): int
  - This method stores the value in char
  - Return 0 if success
  - Return 1 if the storage is full and the variable couldn't be stored
  - Return 2 if the char is already taken.
- get(char): double
  - This method returns the value stored at char
- print_all(): void
  - This method prints all variables currently stored
- delete(char): int
  - This method deletes the variable stored at char

    - Return 0 if success
    - Return 1 if the variable could not be found
    - We let the return type to be int, because of consistency & it's future proof.
- deleteAll(): void
    - This method deletes all variables.
- saveToCSV(string): int
    - This method saves the current variables to a file specified with string in csv format.
    - Return 0 if the variables were successfully stored
    - Return 1 if the location is write-protected
    - Return 2 for other errors
- loadFromCSV(string): int
    - This method loads the variables located in a CSV-file at string.
    - Return 0 opon success
    - Return 1 if the file could not be found
    - Return 2 if the file could not be opened or invalid format (not csv)
    - Return 3 if the file is too large (too many lines)

1.1.3 Class Constant

Constants are a special class with protected names. This class can easily be expanded for other purposes.



The class attributes are:

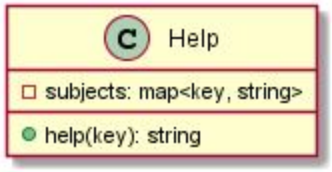
- No attributes are necessary

The class operations are:

- pi(): double
    - This returns the constant value of *Pi*.
- e(): double
    - This returns the constant value of the Euler number *e*.

1.1.4 Class Help

This class stores help subjects to different functions.



The class attributes are:

- subjects: map<key, string>
    - This map stores help subjects. key is a std::string

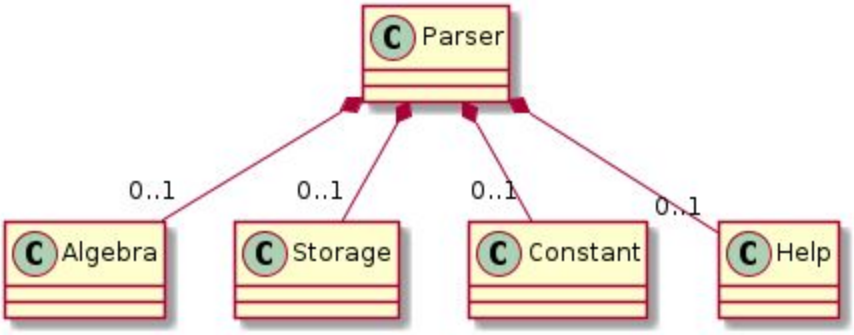
The class operations are:

- help(string): key

- Returns the string specified at the key or an error message if the key could not be found.

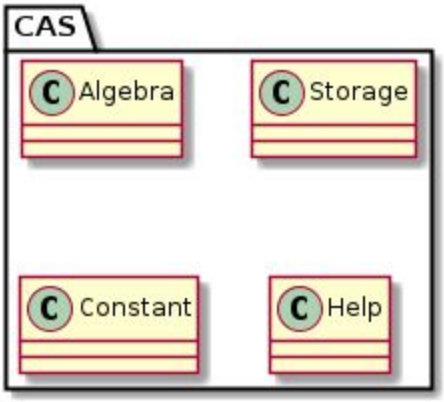
1.1.5 Class Diagram of Package CAS

The Parser will be given. There is a composition between the parser and all the other classes, which we described before. The parser can call 0 or 1 of our given classes.



1.2 Composite Structure Diagram

Every Class is only called once at most and are independent of each other.



2. Dynamic Modeling

2.1 Sequence Diagrams

2.1.1 Storage of variables and multiplication

This sequence describes an example, where the user wants to store value in Storage and multiply them.

The functional requirements related to this sequence are:

- FREQ-1: Parsing expressions

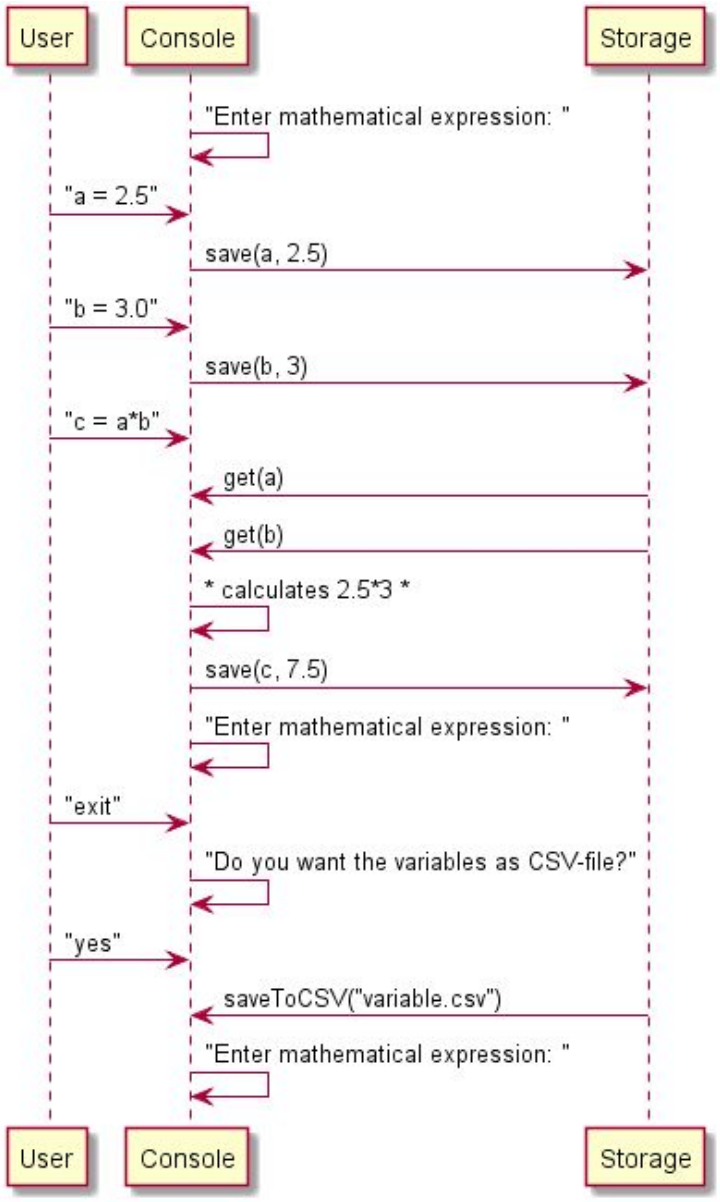
- FREQ-2: Basic arithmetic operations
- FREQ-3: Store variables
- FREQ-9: Saving and loading data
- FREQ-10: Exiting the program

The scenarios which are related to this sequence are:

- SCN-1: The user wants to store variables and multiply them.

Scenario Narration:

The user starts the software, which then displays “Enter mathematical expression:”. The user writes “a = 2.5” in the command line, then “b = 3”, and finally “c = a * b”. After each step, the system stores the value of the equation in the corresponding variable. At the end, the user writes “exit” and the system asks if he/she wants to save the variables. The user writes “yes” and the software creates a “variables.csv” file where the variables are stored.



2.1.2 Nested expressions and exit

This sequence describes an example, where the user wants to calculate the results of nested expressions with parentheses and then exits the software.

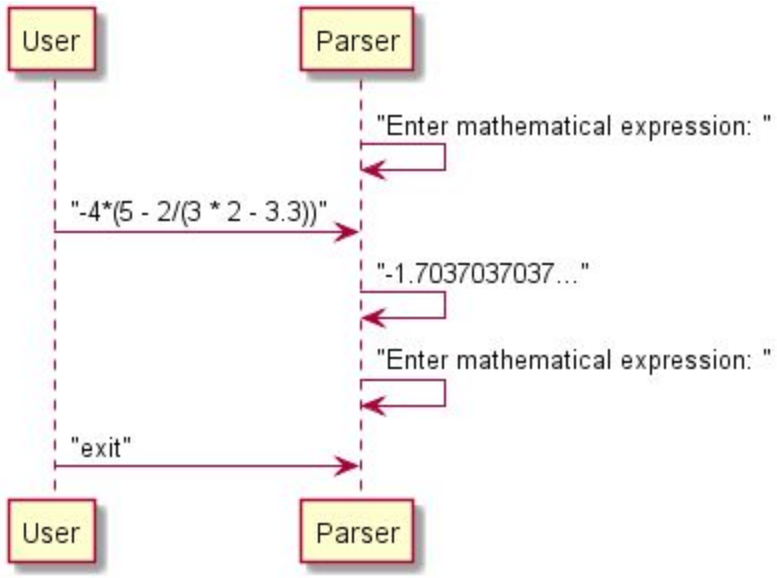
The functional requirements related to this sequence are:

- FREQ-1: Parsing expressions
- FREQ-2: Basic arithmetic operations
- FREQ-4: Delete variables
- FREQ-5: Nested expression using parentheses
- FREQ-10: Exiting the program

The scenarios which are related to this sequence are:

- SCN-2: The user wants to know the results of nested expressions with parentheses and exit the software.

Scenario Narration:
The user starts the software, which then displays “Enter mathematical expression:”. The user writes “-4*(5 - 2/(3 * 2 - 3.3))” in the command line and the system displays the result “-1.7037037037…” directly over the console because no “=” was detected. At the end, the user writes “exit” and the program shuts down because there was no variable that could have been stored.



2.1.3 Trigonometric expressions

This sequence describes an example, where the user wants to calculate the result of a trigonometric expression.

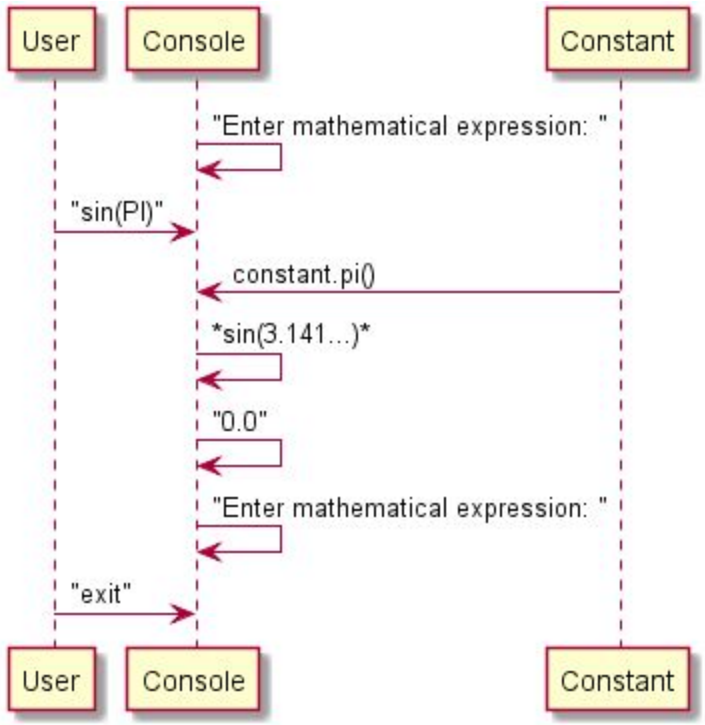
The functional requirements related to this sequence are:

- FREQ-1: Parsing expressions
- FREQ-6: Must known basic functions & constants

The scenarios which are related to this sequence are:

- SCN-3: The user wants to know the result of a trigonometric expression.

Scenario Narration:
The user starts the software and writes: “sin(constant.pi())”. Device gets constant.pi() and returns 3.141... . Now the device calculates sin(3.141...) and returns 0.0. Device responds with: “Enter mathematical expression: “. User writes “exit” and Device shuts down.



2.1.4 Error message and variable storage

This sequence describes an example, where the user enters an illegal expression which leads to an error, and then stores a value in a variable.

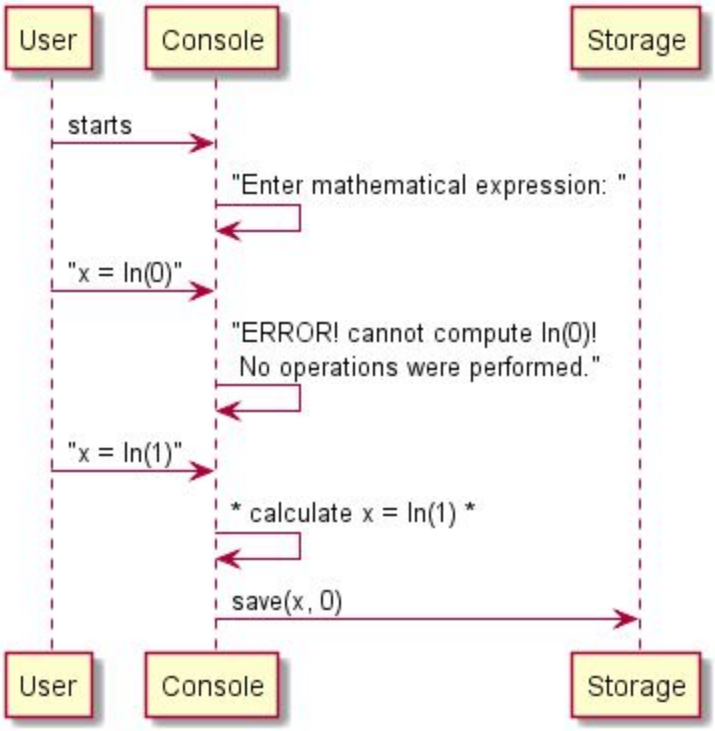
The functional requirements related to this sequence are:

- FREQ-1: Parsing expressions
- FREQ-3: Store variables
- FREQ-7: Error feedback

The scenarios which are related to this sequence are:

- SCN-4: The user enters an illegal expression which leads to an error.

Scenario Narration:
The user starts the software, which then displays “Enter mathematical expression:”. The user writes “x = ln(0)” in the command line and the system displays “ERROR: cannot compute ln(0)! No operations were performed. ”, and then again “Enter mathematical expression:”. This time, the user writes “x = ln(1)” and the software stores the result in the variable “x”, i.e. 0.


User
Console
Storage
starts
"Enter mathematical expression: "
"x = ln(0)"
"ERROR! cannot compute ln(0)!
No operations were performed."
"x = ln(1)"
* calculate x = ln(1) *
save(x, 0)
User
Console
Storage