**Fractions v.1.0, aka Zlomky v.1.0**

**Matvei Slavenko**

Charles University in Prague

Faculty of Mathematics and Physics

Obecná matematika, the first semester

NMIN101 Programming I

Winter semester, school year 2016/2017

# General Information

This programme renders mathematical formulae using pseudo graphics. The input for the programme is a mathematical formula, and the output is the same formula presented in pseudo graphics.

The programme has a simple command-line user interface. The user can set the input and output streams at his/her convenience: the programme could both use files and standard input-output system. The list of commands and their detailed descriptions are available in the internal help module of the programme.

# Data Structures Used

Two data structures were used in this programme.

## Binary Tree

This data structure is used to store the abstract syntax tree of the processed expression. Apart from the operands and operations, each node contains all information necessary for calculating of its graphical parameters and its rendering. This information includes:

* The type of the node:
  + 'num' for numbers
  + '/', '\*', '+', '-' for operations
* The height and the width of the subexpression represented by this node. The values are measured in number of characters, that is needed to print the expression out;
* The initial coordinates, from which the expression represented by this node should be printed

Besides, each node contains information about its parent and children.

## Priority Queue

The main feature that allows printing of expressions on several levels with fraction lines is a printing buffer. Usually there is a sequential access to printing only. To avoid traversing the expression’s AST in the correct order, the programme uses the following scheme: before printing the expression out, the programme orders single parts of the expression in a buffer with a random writing access.

The priority queue is used as this buffer. The fact that the queue allows giving priorities to its items makes ordering of the single parts of the expression really easy and user-friendly. There is no need to order the parts manually. It is done automatically by the data structure.

# General Ideas of Algorithm

These are the main logical steps of the algorithm:

## Preparation of the external resources

The programme prepares the external resources: assigns the files on the disk to the proper variables and opens them.

The procedure prepareSources(inPath, outPath: String) is responsible for this.

## Parsing of the expression and building of its AST

The programme one by one reads tokens from the input stream and constructs the AST of the expression. The procedure parseFile() is responsible for it.

## Appending of the brackets

Using the information about AST structure and types of its nodes, the programme recursively calculates the number of brackets that should be added to each number or quotient in the AST.

The procedure insertBrackets(node: pTree; valL, valR: Integer) is responsible for it.

## Calculation of the auxiliary graphical parameters

On this stage, the programme recursively calculates the heights and the widths of the subexpressions represented by each node.

Procedures calculateHeights(node: pTree) and calculateWidths(node: pTree) are responsible for actions of this step.

## Calculation of the values that are to be printed

The programme recursively calculates the 'val' values of the expression - the values, which are to be printed. For example, the fraction lines and extra spaces are added to the expression on this stage.

The procedure fillVals(node: pTree) is responsible for it.

## Calculation of the main graphical parameters

The programme recursively calculates the starting coordinates for each node. The procedures prepareY(node: pTree; val: Integer) and prepareX(node: pTree; val: Integer) are responsible for this.

## Construction of the rendered formula in a buffer

All nodes are put to the printing queue on this stage. The internal mechanisms of the data structure sort the items in a right order. The procedure fillQ(node: pTree) is responsible for it. The order in the buffer is defined by the function compare(a,b: pQueue): Integer.

## Printing out the rendered formula from the buffer

The programme prints out the items in the order they are stored in the printing buffer. The procedure print(buffer: pQueue) is responsible for it. The procedures adjustX(item: pQueue) and adjustY(item: pQueue) help to adjust the position of the printing head, ensuring skipping of the empty character spaces.

## Closing of the external resources

All external resources are closed and freed on this staged. The procedure closeSources() ensures these actions.

See the documentation included to the source code for detailed information about each step, actual implementation and other details. Source code is distributed freely and could be downloaded, for example, from the [GitHub repository](https://github.com/slavenkof/zlomky) (<https://github.com/slavenkof/zlomky>).

# Input Data Format

* 1. The input expression should contain all parentheses;

Example:

* (1+2) is a valid expression;
* 1+2 is not a valid expression;

1. The input expression should not contain spaces or letters of the alphabet;

Example:

* (a+2) is not a valid expression;
* (2 + 3) is not a valid expression;

1. The input expression should contain integers only;

Example:

* (1.02+59) is not a valid expression;

1. The length of the expression does not matter and is limited only by the machine’s memory.

# Examples of Programme’s Work

Input:

((((1/2)+(2/6))/(25/(2654+8753)))+(258/96))

Output:

1 2

(- + -)

2 6 258

(--------------- + ---)

25 96

-------------

(2654 + 8753)

Input:

((((406\*580)+33045)-((75846/237)+(3/80000)))/((3625\*80)-243))

Output:

75846 3

(((406 \* 580) + 33045) - (----- + -----))

237 80000

---------------------------------------------

((3625 \* 80) - 243)

Input:

((((57\*162)/2052)+((12768\*5)/456))-((3468\*154)/(68\*357)))

Output:

(57 \* 162) (12768 \* 5) (3468 \* 154)

((---------- + -----------) - ------------)

2052 456 (68 \* 357)

Input:

((((4561+5739)\*12)/((701-501)/5))-(((445-22)/32)/((823+177)\*30)))

Output:

(445 - 22)

----------

((4561 + 5739) \* 12) 32

(-------------------- - ------------------)

(701 - 501) ((823 + 177) \* 30)

-----------

5

# Ideas for Programme’s Improvement

1. To implement the spaces tolerance;
2. To implement the support of decimal fractions and letters of the alphabet;
3. To implement the support of roots, powers, and integrals;
4. To implement the BASH-friendly working mode;
5. To deal with memory leaks.