KALKULATOR

Strokovno poročilo

Mentor: Darjan Toth, prof.

Avtor: Rok Strah, R 4. C

Ljubljana, 16. april 2018

Povzetek

Abstract

Kazalo

Slike

1. Uvod

2. Metodologija

2.1 Okolje, orodja, jezik in standardi

Za projekt sem uporabil programski jezik C++, razvojno orodje Qt Creator in nekaj knjižnic.

2.1.1 C++

C++ je Bjarne Stroustrup kot razširitev C-ju naredil v letu 1979. Želel je ustvariti učinkovit in prilagodljiv jezik podoben C-ju, ki pa bi tudi imel visoko-nivojske funkcije za organizacijo programa, kot so razredi. C++ je ISO/IEC JTC 1/SC 22/WG 21 standardizirala leta 1998, in od takrat so izdali 5 standardov:[3]

- ISO/IEC 14882:1998 znan kot C++98,
- ISO/IEC 14882:2003 znan kot C++03,
- ISO/IEC 14882:2011 znan kot C++11 ali C++0x,
- ISO/IEC 14882:2014 znan kot C++14 ali C++1y in
- ISO/IEC 14882:2017 znan kot C++17 ali C++1z,

sedaj pa delajo na naslednjem standardu, ki naj bi prišel v letu 2020 in je znan pod imenom C++20.

Bjarne je leta 1979 začel razvijati „C with Classes”, razširitev C-ju in predhodnik C++. Ko je delal na doktorski nalogi, je opazil, da ima jezik „Simula” nekaj funkcij, ki so zelo uporabne za velike projekte, vendar je prepočasen za praktično rabo, vendar pa jeziku „BCPL”, ki je hiter, manjkajo visoko-nivojske funkcije za večje projekte. Med delom v „AT&T Bell Labs” je bil navdihnjen izboljšati C s funkcijami podobnimi tistim v Simuli. Izbral si je C,

ker je fleksibilen, učinkovit, dostopen in prenosljiv. Poleg vplivov C in Simule, so na C++ vplivali tudi drugi jeziki, kot so ALGOL 68, Ada, CLU in ML. Leta 1983 se je „C with Classes" preimenoval v C++, leta 1989 je pa bil dokončan C++ 2.0.

Bjarne je medtem tudi pisal knjigo „The C++ Programming Language". Do sedaj je knjiga izšla v štirih izvodih v letih: [1]

- 1985,
- 1991,
- 1997 in
- 2013

ter v eni posebni izdaji leta 2000. Po C++ 2.0 se je C++ razvijal relativno počasi do 2011, ko je izšel standard C++11, ki je dodal veliko novih funkcij in je razširil „standard library" (okrajšano STL). Trenutno je C++, za Javo in C-jem, tretji najbolj znan programski jezik. [2]

Filozofija C++ [3, 4]

- Splošna pravila:
  - Evolucija c++ morejo voditi resnični problemi.
  - Ne ukvarjaj se z nesmiselnim iskanjem popolnosti.
  - C++ more biti uporaben zdaj.
  - Vsaka funkcija jezika mora imeti razumno očitno implementacijo.
  - Vedno zagotovi prehodno pot.
  - C++ je jezik, ne popoln sistem.
  - Zagotovi celovito podporo za vsak podprt slog.
  - Ne poskušaj siliti ljudi, naj uporabljajo specifičen programski slog.
- Pravila za podporo oblike:
  - Podpiraj smiselne oblikovne ideje.

  - Zagotovi metode za organizacijo programa.
  - Reči kar misliš.
  - Vse funkcije morejo biti cenovno dostopne.
  - Bolj je pomembno dovoliti uporabno funkcijo, kot pa preprečiti vsako zlorabo.
  - Podpiraj sestavljanje programske opreme iz ločeno razvitih delov.

- Jezikovno-tehnična pravila:
  - Nobenih implicitnih kršenj statičnega sistema tipov.
  - Zagotovi tako dobro podporo za uporabniško-definirane tipe kot za vgrajene tipe.
  - Izogibaj se zaporednih odvisnosti.
  - Če si v dvomu, izberi varianto funkcije, ki jo je najlažje naučiti.
  - Sintaksa je važna (ponavadi na čudne načine)
  - Uporaba predprocesorja naj bi bila odpravljena.

- Pravila za podporo nizko-nivojskega programiranja:
  - Uporabljaj tradicionalne (neumne) povezovalnike (angl. „linker").
  - Nobenih neupravičenih nezdružljivosti s C.
  - Ne pusti mesta za nizko-nivojski jezik pod C++ razen ASM.
  - Za kar ne uporabljaš, ne plačaš (angl. zero-overhead rule).
  - Če si v dvomu, zagotovi možnost za ročni nadzor.

2.1.2 Qt

Qt ([kjut] angl. „*cute*"[5]) je odprtokodno ogrodje za C++, ki ponuja razvojno okolje „Qt Creator" (okrajšano IDE iz angl. *Integrated Development Enviroment*), oblikovalec vmesnikov „Qt Designer", prevajalec „Qt Linguist" in „Qt Assistant", ki kaže dokumentacijo in primere uporabe Qt knjižnic. V „Qt Creator" imajo uporabniki na voljo tudi dostop do drugih treh programov. Qt knjižnice ločimo od drugih po njihovi predponi „*Q*"–. Qt ima veliko funkcij, vendar najpomembnejše so pred-procesor moc (iz angl. Meta-Object Compiler[6]), Qt knjižnica gradnikov (Qt Widgets) in Qt signali in reže (Signals & Slots), s katerimi lahko nadziramo grafični vmesnik (okrajšano GUI iz angl. *Graphical User Interface*).

Signali[7]

Qt uporabnikom zagotavlja funkcije, za povezovanje signalov na reže. To sta „`connect`" in „`disconnect`". Signali se ponavadi sprožijo, ko se objekt (angl. object) spremeni, pa bi to želeli sporočiti njegovemu staršu (angl. parent). Ko si signal sproži se ponavadi reže povezane nanj takoj izvedejo, kot navaden klic funkcije, [7] vendar pa če uporabljamo zaporedne povezave (angl. queued connections) koda normalno nadaljuje z izvajanjem in se reže začnejo izvajati šele kasneje. [7]

Reže[8]

Reže so navadne funkcije, na katere pa lahko povežemo signale.

2.1.3 Knjižnice

Exprtk[9, 10]

Exprtk je knjižnica za računanje za C++.

KLFBackend[11]

KLFBackend je C++ knjižnica za prikazovanje LaTeX kode.

MufExprtkBackend

MufExprtkBackend je vmesnik med Qt programom in knjižnico Exprtk. Ker se ta knjižnica ne spreminja pogosto, jo lahko prevedemo posebej od aplikacije, kar skrajša čas prevajanja aplikacije za 1 do 10 minut, odvisno od zmogljivosti procesorja.

MufTranslate

MufTranslate je knjižnica, ki sem jo uporabil za prevajanje vmesnika v različne jezike. Za razliko od Qt Linguist knjižnica podpira prevode v obliki JSON datoteke.

2.1.4 Linux

Linux je družina prostih in odprto-kodnih operacijskih sistemov, ki za jedro (angl. *kernel*) uporabljajo „Linux". Aplikacijo sem začel razvijati na Windows operacijskem sistemu, vendar sem moral zaradi omejitev sistema zamenjati na Linux.

2.1.5 Druga orodja in standardi

AStyle[12]

AStyle je program, ki samodejno oblikuje izvorno kodo po definiciji, ki jo podaš. Qt tudi omogoča povezavo preko dodatkov (angl. plugin);

CMake[13]

CMake je prosta in odprto-kodna rešitev za avtomatizacijo grajenja projektov. V primerjavi s Qt-ovim qmake je močnejši, vendar zahtevnejši za uporabo.

Ninja[14]

Ninja je prost odprto-kodni sistem za grajenje projektov. Za razliko od ostalih takih sistemov je bistveno hitrejši. Sicer ga je možno uporabljati samega, je mišljeno, da se ga uporablja v povezavi z programom, ki generira „`.ninja`" datoteke, kot je CMake.

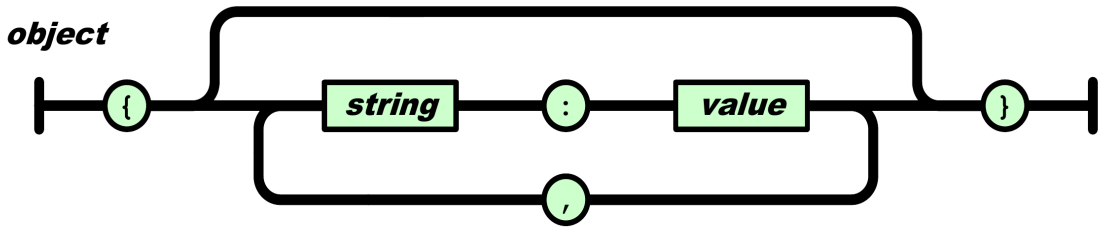
Clang[15]

Clang je prost odprto-kodni vmesnik za jezike družine C, ki se od GCC-ja razlikuje predvsem po boljših sporočilih in hitrosti.

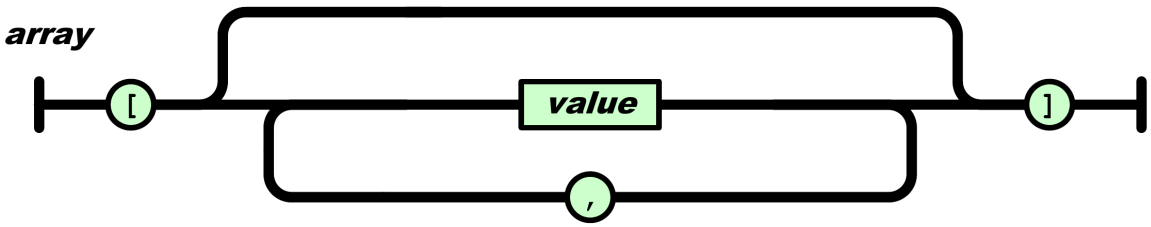
JSON[16]

JSON (iz angl. *JavaScript Object Notation*) je način zapisa tabel in objektov v tekstovno datoteko. Struktura JSON:[16]

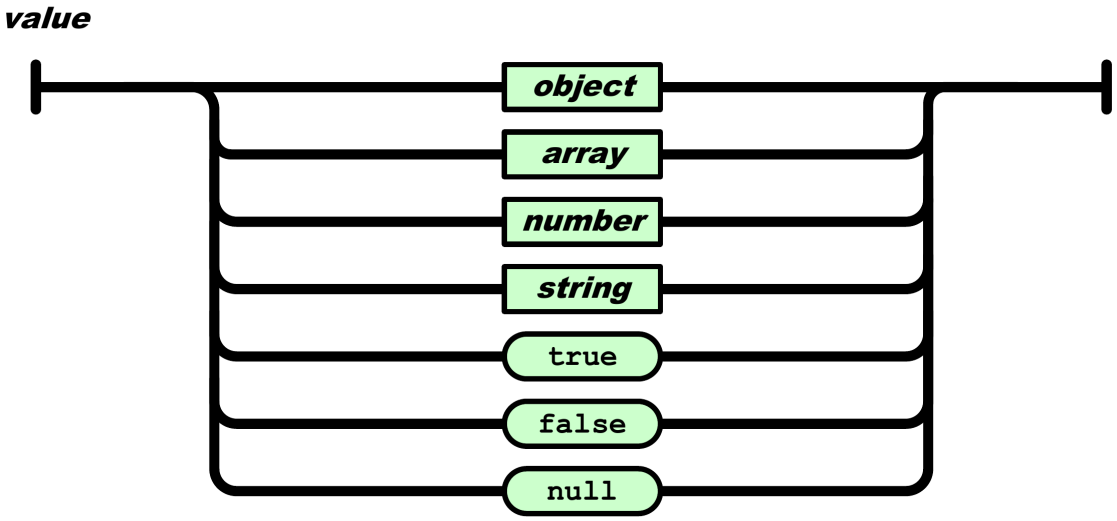
- JSON dokument je sestavljen iz objekta ali tabele.
- JSON objekt je sestavljen iz parov ključev in vrednosti. 2.1.
- JSON tabela vsebuje več vrednosti.2.2
- Ključ je niz, vrednost je pa lahko niz, število, tabela, objekt, dvojiška vrednost (true in false) ali null.2.3 2.4 2.5



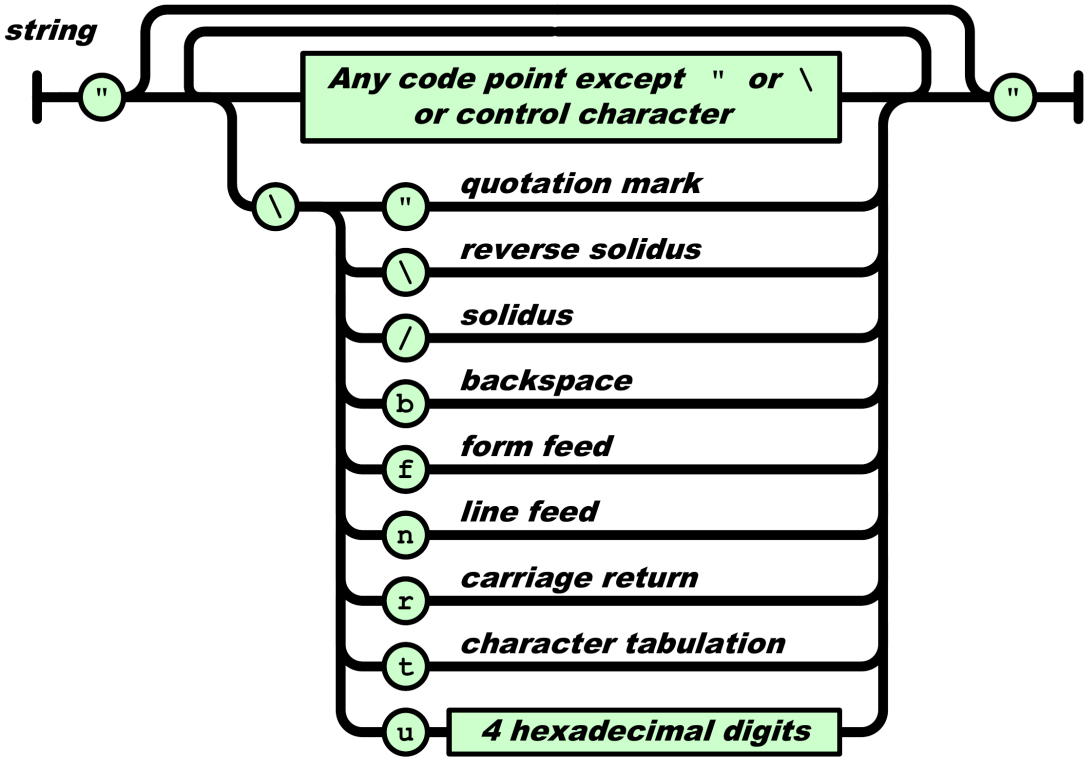
Slika 2.1: JSON objekt[17]



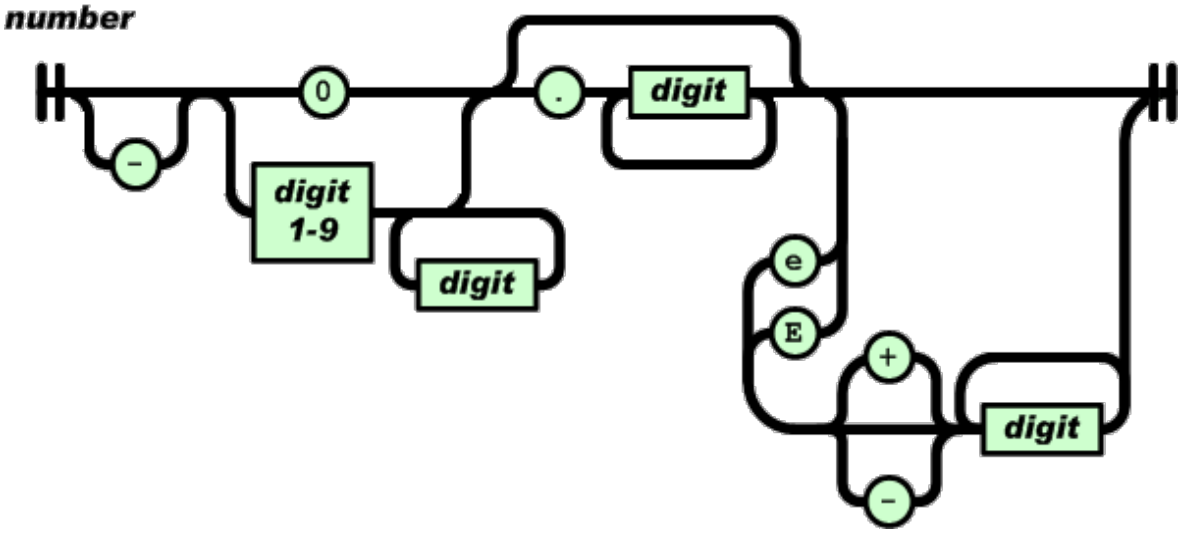
Slika 2.2: JSON tabela[17]



Slika 2.3: JSON vrednost[17]



Slika 2.4: Niz v JSON[17]



Slika 2.5: število v JSON[17]

2.2 Koncepti

2.2.1 Abstrakten sistem prepisovanja

Abstrakten sistem prepisovanja (okrajšano ARS iz angl. *abstract rewriting system*) je množica objektov A in binarna relacija, ki pove kako preoblikovati te objekte. To binarno relacijo imenujemo „prepisovalna relacija” [18, chapter, p. 7] in jo dobimo iz množice pravil. ARS se uporabljajo na več področjih, npr. v matematiki, računalništvu, logiki in jezikoslovju.

2.2.2 Sistem prepisovanja členov

Sistem prepisovanja členov (okrajšano TRS iz angl. *term rewriting system*) je ARS, ki ima za elemente člene matematičnega izraza.

2.2.3 Sistem prepisovanja nizov

Sistem prepisovanja nizov (okrajšano SRS iz angl. *string rewriting system*) ali semi-Thue sistem je ARS, ki ima za elemente nize ali podnize.

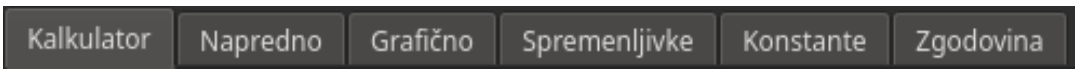
2.2.4 Abstraktno sintaktično drevo

Aplikacija iz vnosa zgradi abstraktno sintaktično drevo (okrajšano AST iz angl. *abstract syntax tree*). Ko je AST zgrajen, omogoča implementacijo TRS za poenostavljanje izrazov in SRS za pretvarjanje izraza v sintakso jezika LaTeX.

3. Funkcionalnost

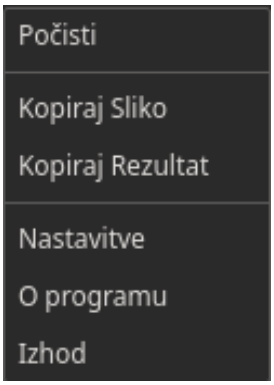
3.1 Glavno okno

Na glavnem oknu je več zavihkov — „Kalkulator", „Napredno", „Grafično", „Spremenljivke", „Konstante" in „Zgodovina" — vsak s svojo funkcijo.



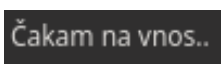
Slika 3.1: Zavihki na glavnem oknu

V zgornjem meniju najdemo nekaj orodji, kot npr. „Nastavitve" ali „O programu"



Slika 3.2: Orodja v zgornjem meniju

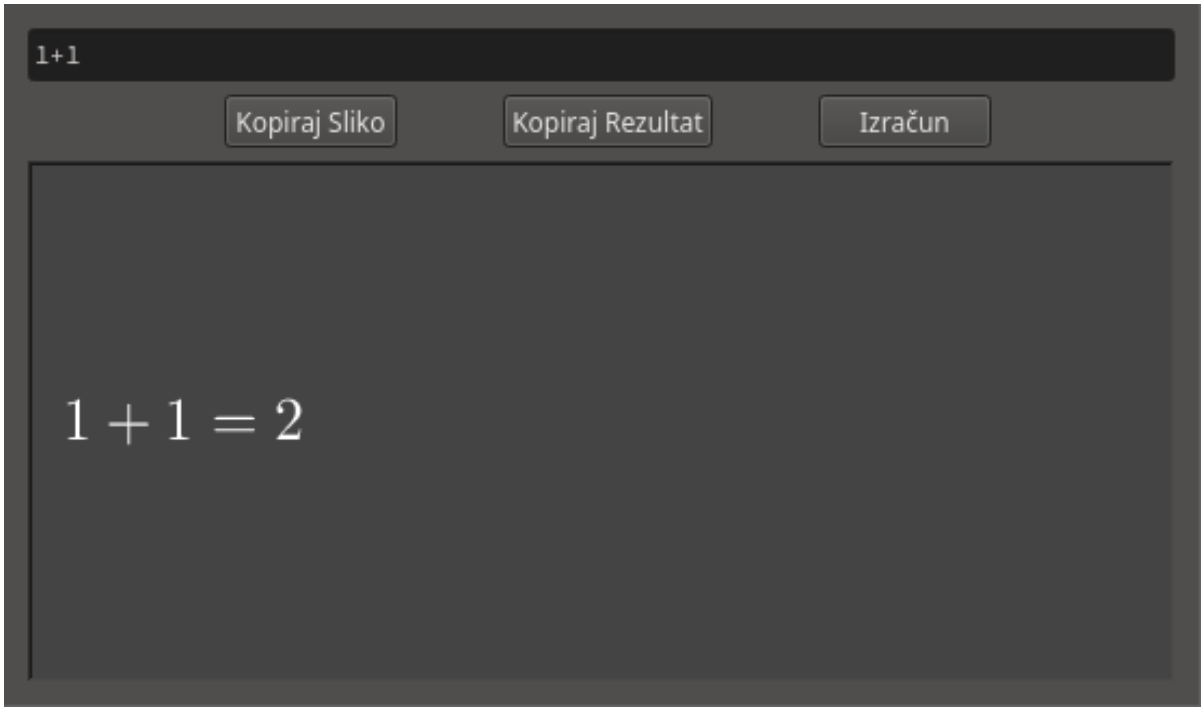
Na dnu je pa tudi viden status aplikacije, ki sporoča uporabniku, kaj aplikacija trenutno dela.



Slika 3.3: Trenutni status aplikacije

3.1.1 Kalkulator

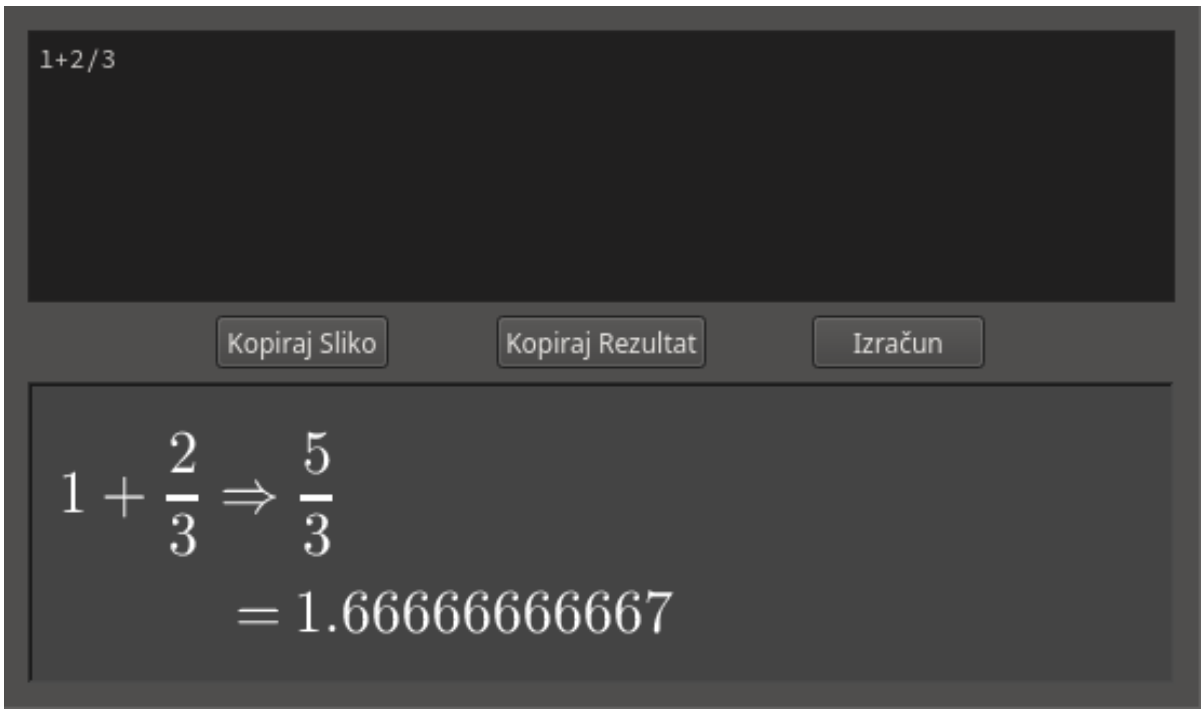
Prvi zavihek je „preprost” kalkulator. Zgoraj je vrstica za vnos računa, pod njo so pa trije gumbi, dva za kopiranje rezultata in en za začetek računanja. Računati začne tudi ob pritisku tipke „Enter”.



Slika 3.4: Preprost kalkulator

3.1.2 Napredno

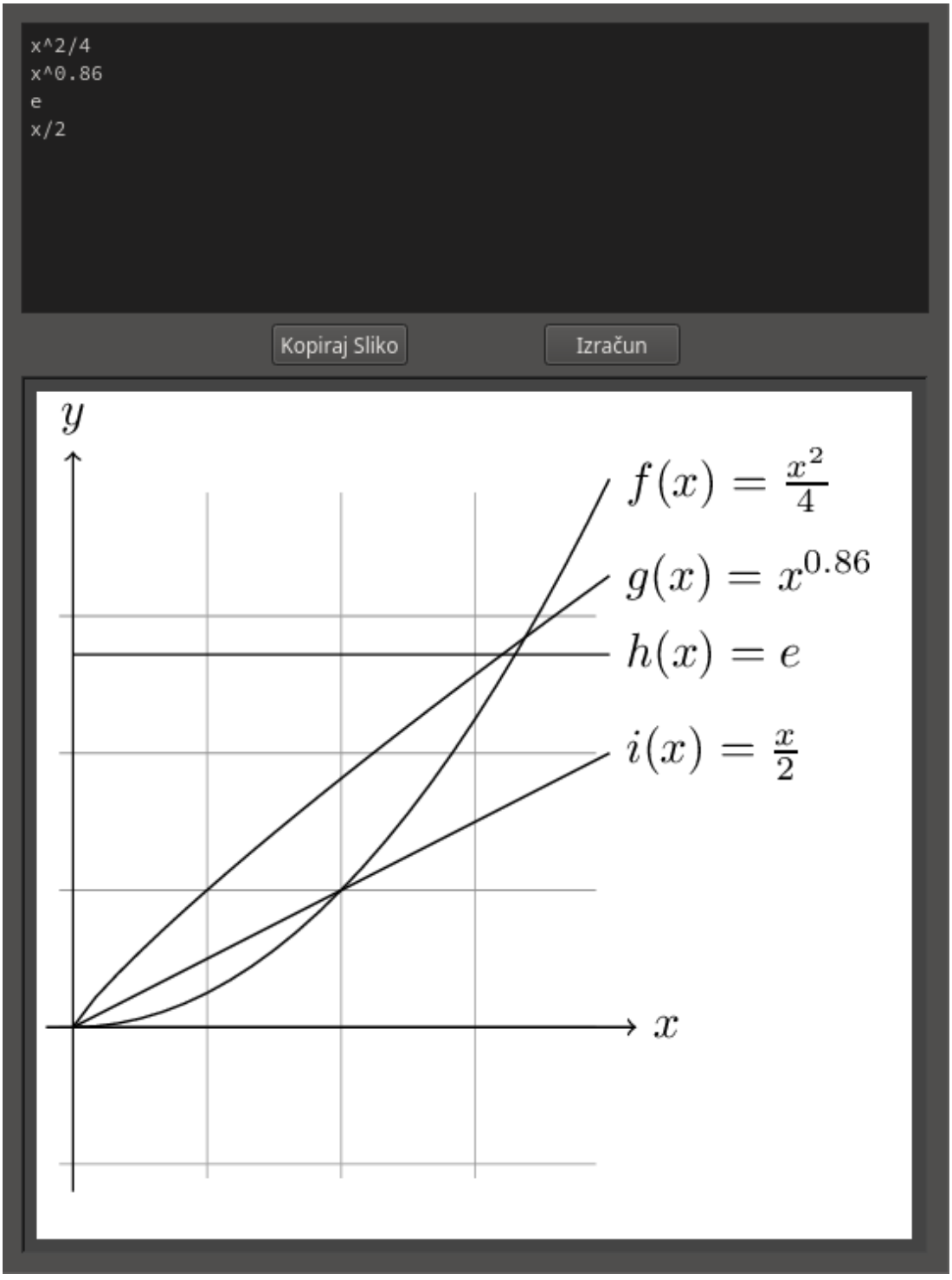
Napredni kalkulator je razširitev preprostega kalkulatorja. Vnosno polje za račun dovoljuje vnašanje večih vrstic, kar pomeni, da je možno uporabiti tudi kompleksnejše strukture kot so zanke ter pogojni stavki. Poleg rezultata se pa tudi izriše poenostavljen izraz.



Slika 3.5: Napredni kalkulator

3.1.3 Grafično

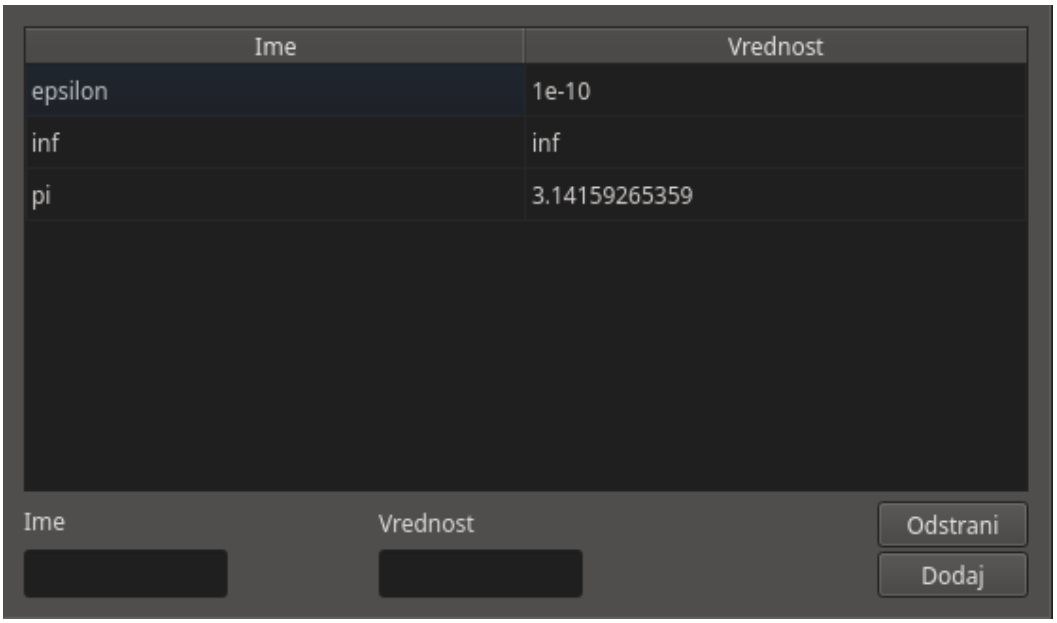
Vnosno polje je podobno kot v naprednem kalkulatorju, manjka pa gumb „Kopiraj rezultat”, saj aplikacija v tem načinu nič ne računa, ampak riše grafe funkcij, ki jih pišemo v vnosno polje.



Slika 3.6: Grafični kalkulator

3.1.4 Spremenljivke in Konstante

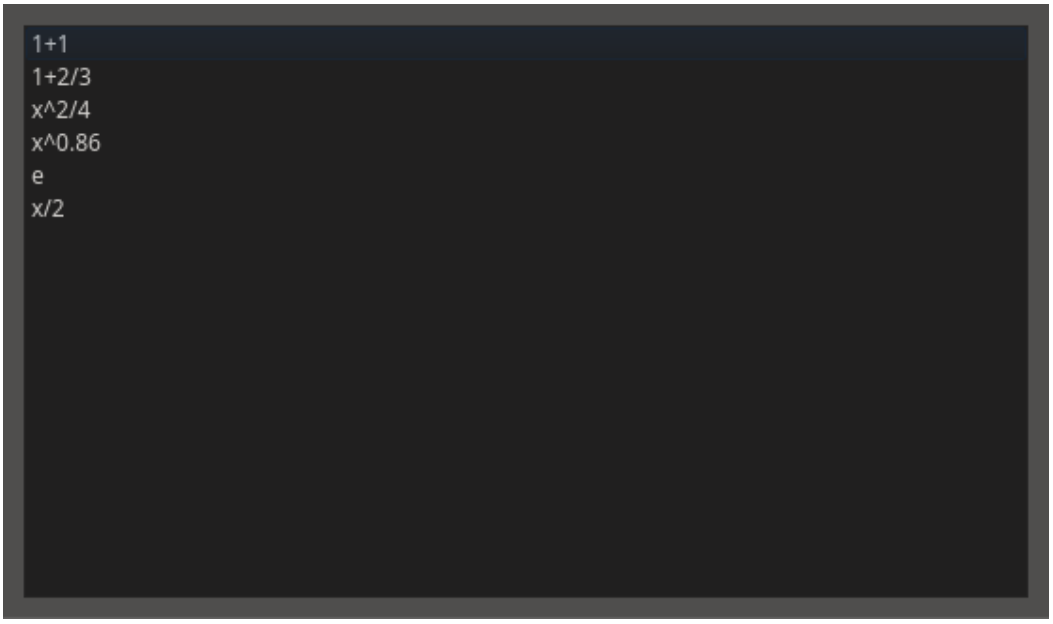
Ta dva zavihka v tabeli prikazujeta vse spremenljivke in konstante, ki so vnesene v programu.



Slika 3.7: Konstante

3.1.5 Zgodovina

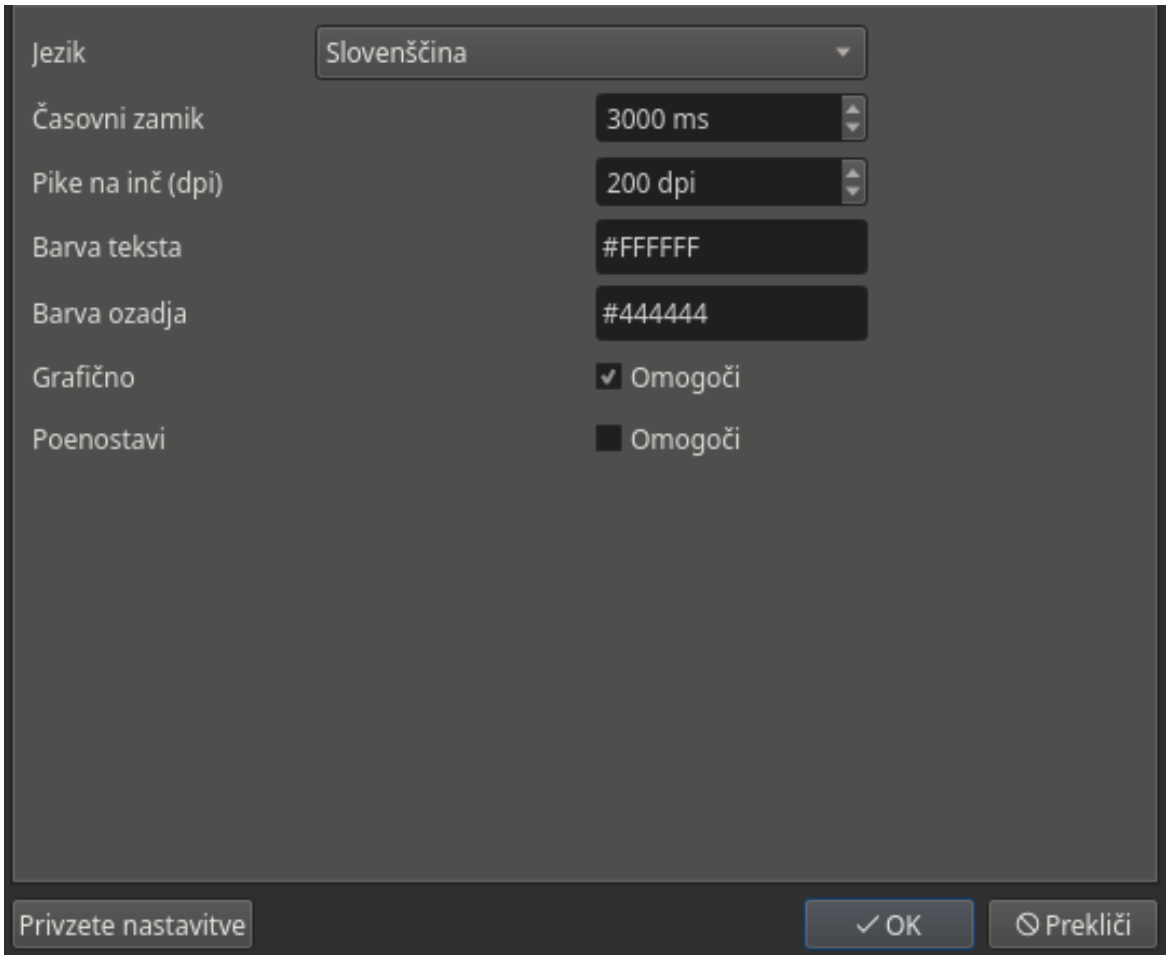
Aplikacija beleži zgodovino vnesenih računov.



Slika 3.8: Zgodovina

3.2 Nastavitve

V nastavitvah lahko nastavljamo jezik, čas prikaza statusa, velikost izrisane slike in barvo slike, lahko pa tudi vklopimo in izklopimo razne funkcije.



Slika 3.9: Zgodovina

Na dnu okna najdemo gumba „OK” in „Prekliči” ter gumb za nastavitev privzetih vrednosti.

3.3 Simboli

Aplikacija iz podanega izraza zgradi AST, kar pa reši dva od problemov naloge.

3.3.1 Poenostavljanje izrazov

Poenostavljanje izrazov poteka v treh delih:

- ponovno poenostavljanje po preprostih pravilih.
- pretvorbe vseh elementov drevesa v ulomke in
- ponovno poenostavljanje po preprostih pravilih.

Prvi del poenostavi izraze kot so $x+0$ in x^1, drugi del pretvori izraze kot so $\frac{a}{b}+\frac{c}{d}$ v $\frac{ad+bc}{bd}$ ali $1+\frac{2}{3}$ v $\frac{5}{3}$, tretji del pa poenostavi izraze oblike $\frac{x}{1}$

3.3.2 Pretvorba v LaTeX

Za pretvorbo v LaTeX se program preprosto spusti po drevesu in na primer za ulomke izpiše `{\frac{števec}{imenovalec}}`. Z le par preprostimi pravili dokaj zanesljivo prepiše cel izraz.

4. Implementacija

4.1 Glavno okno

4.1.1 Inicializacija

Aplikacija najprej naloži nastavitve in postavi par spremenljivk na začetne vrednosti, nato pa začne klicati podprograme za inicializacijo posameznih logičnih enot. Potem poveže signal, ki se sproži, ko je rezultat obdelan, na režo, ki rezultat posreduje izrisovalcu. Na koncu še vpiše ves tekst v aplikacijo.

KLFBackend

Podprogram najde nastavitve LaTeX ozadja in z njimi naredi nov „`KLFPreviewBuilderThread`”.

Exprtk

Podprogram naredi nov „`MufExprtkBackend`” in poveže signal za konec računanja in signal za napake na njihove reže.

Glavno okno

Podprogram najprej kliče podprogram za inicializacijo gornjega menija in poskrbi, da ko se spremeni jezik v knjižnici za prevajanje, se tudi posodobi tekst.

Meni

Podprogram naredi nov „`MufMenu`” in poveže elemente menija na reže, ki bojo izvedle želeno akcijo.

Spremenljivke in konstante

Podprogram naredi nove „`MufSymbolListView_w`”-je, jih doda v njihove zavihke, jih posodobi s podatki iz „`Exprtk`”-ja in poveže gumbe za dodajanje in odstranjevanje spremenljivk in konstant.

Funkcije

Podprogram najde pot do definicij funkcij in s to potjo naredi nov „`MufFunctions`”.

Kalkulator

Vsi trije podprogrami za inicializacijo zavihkov s kalkulatorji povežejo signale, ki sprožijo računanje na reže, ki začnejo računanje in povežejo gumbe za kopiranje rezultata. Nato nastavijo še kazalec na vnosno polje.

Nastavitve

Podprogram naredi nov „`MufSettings_w`". Nato najde pot do jezikovnih datotek in jezike doda v spisek jezikov.

Posodobitev teksta

Podprogram v program vpiše na novo prevedene ključe in pokliče podprograme štirih drugih objektov: spremenljivke, konstante, nastavitve in gornji meni, ki pa naredijo podobno.

4.1.2 Potek računanja

Za računanje so zadolžene tri skupine funkcij, vsaka za svoj način računanja. Vsaka skupina funkcij je označena s svojo končnico, `-{0}` za navadni način, $-\_adv$ za napredni način in $-\_plot$ za grafični način. Ker so si te trije načini računanja zelo podobni, se je veliko kode ponavljalo, kar lahko povzroča napake. To sem rešil tako, da sem združil podobne funkcije v eno, pustil sem pa nekaj ključnih funkcij, ki skrbijo za celoten potek izračuna.

Najprej bom opisal celoten potek, potem pa bom še povedal kaj se spremeni v različnih skupinah funkcij. Računanje se začne v funkciji „`compute`", ki s pomočjo Qt-ovih funkcij „`connect`" in „`disconnect`" nadzira, katera skupina funkcij naj se uporablja, kliče pa tudi funkciji „`updateHistory`" in „`updateExprtkInput`", ki posodobita zgodovino in račun v knjižnici „Exprtk". Ko „Exprtk" obdela račun in vrne rezultat, je ta posredovan funkciji, ki poskusi odpraviti napake, ki nastanejo zaradi omejitev jezika. Ko je rezultat obdelan se začne izvajati funkcija „`updatePreviewBuilderThreadInput`", ki nastavi nastavitve KLF-ja, pretvori izraz v LaTeX in začne risanje. Ko „KLFBackend" izriše izraz, se

slika pošlje funkciji, ki to sliko postavi v aplikacijo.

Med funkcijami iz različnih skupin ni veliko razlik, spremenijo se le imena spremenljivk, kar je težko spreminjati, glede na trenuten potek izvajanja programa, lahko pa vidimo bistvene razlike v funkciji „`updatePreviewBuilderThreadInput`”.

Navadni način

V navadnem načinu funkcija nastavi „KLFBackend” nastavitve:

- „`input.preamble`” nastavi na „`\usepackage{amssymb,mathtools,mathrsfs}`”
- „`input.mathmode`” nastavi na „`\[ ... \]`”
- „`input.bypasstemplate`” nastavi na „`false`”
- „`input.latex`” nastavi na „`vhod = vrednost`”

Napredni način

V naprednem načinu funkcija nastavi „KLFBackend” nastavitve:

- „`input.preamble`” ne nastavi
- „`input.mathmode`” ne nastavi
- „`input.bypasstemplate`” nastavi na „`true`”
- „`input.latex`” nastavi na:

```
"\\documentclass{article}"
"\\usepackage{amssymb,amsmath,mathtools,mathrsfs}"
"\\usepackage[dvipsnames]{xcolor}"
"\\definecolor{fg}{HTML}{"·+·_color.name().mid(1)·+·"}"
"\\definecolor{bg}{HTML}{"·+·_bg_color.name().mid(1)·+·"}"
"\\begin{document}"
"\\color{fg}"
"\\pagecolor{bg}"
"\\begin{align*}"·+
Muf::toLatex(ui->eqnInput_adv->toPlainText(),·_reduce)·+
+·"=&\\:·"·+·roundValue·+
"\\end{align*}"
"\\pagenumbering{gobble}"
"\\end{document}";
```

Slika 4.1: input.latex

Grafični način

V grafičnem načinu funkcija nastavi „KLFBackend” nastavitve:

- „`input.preamble`” ne nastavi
- „`input.mathmode`” ne nastavi
- „`input.bypasstemplate`” nastavi na „`true`”
- „`input.latex`” nastavi tako:

```
input.latex = "\\documentclass{article}"
    →    ......"\\usepackage{tikz}"
    →    ......"\\begin{document}"
    →    ......"\\textcolor{white}{.}\\\\"
    →    ......"\\begin{tikzpicture}[domain=0.00000001:4]" //support 1/x
    →    ......"\\draw[very thin,color=gray] (-0.1,-1.1) grid (3.9,3.9);"
    →    ......"\\draw[->] (-0.2,0) --- (4.2,0) node[right] {$x$};"
    →    ......"\\draw[->] (0,-1.2) --- (0,4.2) node[above] {$y$};";
char func_name = 'f';
for (QString el : ui->eqnInput_plot->toPlainText().split("\n")) {
    →    Muf::toLatex.mPar(el);
    →    el.replace("x", "\\x");
    →    input.latex += "\\draw[color=black] plot (\\x," // domain here
    →        →    .......+ el +
    →        →    .......") node[right] {$" + func_name + "(x) = "
    →        →    .......+ Muf::toLatex.mPar.tree->toLatex() +
    →        →    ......."$};";
    →    if (func_name == 'z') {
    →        →    func_name = 'f'; // start over from f
    →    } else {
    →        →    ++func_name; // next function
    →    }
}
input.latex += "\\end{tikzpicture}"
    →    .......\\textcolor{white}{.}\\\\\\textcolor{white}{.}"
    →    .......\\pagenumbering{gobble}"
    →    .......\\end{document}";
```

Slika 4.2: input.latex

5. Navodila za uporabo

5.1 Jezik kalkulatorja

• Osnovni operatorji:	`+, -, *, /, %, ^`
• Prireditev:	`:=, +=, -=, *=, /=, %=`
• Enakosti in neenakosti:	`=, ==, <>, !=, <, <=, >, >=`
• Logični operatorji:	`and, mand, mor, nand, nor, not, or, shl, shr, xnor, xor, true, false`
• Funkcije:	`abs, avg, ceil, clamp, equal, erf, erfc, exp, expm1, floor, frac, log, log10, log1p, log2, logn, max, min, mul, ncdf, nequal, root, round, roundn, sgn, sqrt, sum, swap, trunc`
• Trigonometrija:	`acos, acosh, asin, asinh, atan, atanh, atan2, cos, cosh, cot, csc, sec, sin, sinc, sinh, tan, tanh, hypot, rad2deg, deg2grad, deg2rad, grad2deg`
• Nadzor pretoka:	`if-then-else, trojiški pogojni operator (?:), switch-case, return`
• Zanke:	`while, for, repeat-until, break, continue`

Različni stavki so ločeni s podpičjem (;), kalkulator bo pa izpisal vrednost zadnjega izraza.

Aplikacija podpira tudi definicijo funkcij, ki se naložijo ob zagonu programa. Definicije funkcij se nahajajo v mapi „function”. V tej mapi so funkcije razdeljene na „skupine” predstavljene s podmapami. Na primer funkcija „`add`” se nahaja v datoteki „math” v mapi „stl”, torej lahko rečemo da pripada knjižnici „math” iz skupine „stl”. Imena funkcij se ne smejo ponavljati.

Aplikacija v večini primerov spremenljivke obravnava kot globalne spremenljivke. Uporabnik jih preprosto inicializira „`x := 3`” in potem jih lahko prosto uporablja. Na voljo so pa tudi lokalne spremenljivke, ki se uporabljajo, ko v izrazu ni nobenih zunanjih (globalnih) spremenljivk, vendar so vse uporabljene spremenljivke definirane v izrazu. V tem primeru *moramo* uporabiti rezervirano besedo „`var`”. Nekaj primerov preprostih izrazov, ki uporabljajo le lokalne spremenljivke so:

- `1 + 2`
- `var x := 3; 2 * x - 3`

- `var x := 3; var y := abs(x - 8); x - y / 7`

Sintaksa kontrolnih struktur je zelo podobna C, C++ ipd.:

- ```
  if(cond) {
       expr1;
       expr2;
  };
  ```

- ```
  if(cond) {
       expr1;
       expr2;
  } else {
       expr3;
       expr4;
  };
  ```

- ```
  switch {
       case cond:  expr1;
       case cond:  expr2;
  };
  ```

- ```
  while(cond) {
       expr1;
       expr2;
  };
  ```

- ```
  repeat
       expr1;
       expr2;
  untill(cond);
  ```

- ```
  for(init;cond;iter) {
       expr1;
       expr2;
  };
  ```

- `break` in `break[]` zapusti zanko in vrne vrednost znotraj `[]`.

- `continue` spusti iteracijo in nadaljuje z izvajanjem zanke

- Trojiški operator: cond ? exprT : exprF

- `(expr1, expr2) == expr2`:

  Izračuna vse pod-izraze potem pa vrne zadnji pod-izraz.

- ```
  [*] {
       case cond:  expr1;
       case cond:  expr2;
  };
  ```

  Za razliko od `switch-case` bo `[*]` izračunal vse izraze za katere je `cond` enak `true`
  ```

5.1.1 Vhodno/izhodne operacije

Exprtk pozna dve vrsti I/O operacij (vhodno/izhodnih, iz angl. Input/Output):

- Standardne vhodno/izhodne operacije:
  1. `print`
  2. `println`
- Datotečne vhodno/izhodne operacije:
  1. `open`
  2. `close`
  3. `write`
  4. `read`
  5. `getline`
  6. `eof`

5.1.2 Vektorske operacije

Na voljo so tudi dodatne vektorske funkcije:

1. `all_true`
2. `all_false`
3. `any_true`
4. `any_false`
5. `count`
6. `copy`
7. `rotate-left`
8. `rotate-right`
9. `shift-left`
10. `shift-right`
11. `sort`
12. `nth_element`
13. `iota`
14. `sumk`
15. `axpy`
16. `axpby`
17. `axpyz`
18. `axpbyz`
19. `axpbyz`
20. `dot`
21. `dotk`

6. Rezultati in ugotovitve

Literatura

[1] *The C++ Programming Language*. URL: https://en.wikipedia.org/wiki/The_C%2B%2B_Programming_Language (pridobljeno 15. 4. 2018).

[2] TIOBE - The Software Quality Company. *TIOBE Index*. URL: https://www.tiobe.com/tiobe-index/ (pridobljeno 15. 4. 2018).

[3] *C++*. URL: https://en.wikipedia.org/wiki/C%2B%2B (pridobljeno 15. 4. 2018).

[4] *Evolving a language in and for the real world: C++ 1991-2006*. URL: http://stroustrup.com/hopl-almost-final.pdf (pridobljeno 15. 4. 2018).

[5] *About Us*. URL: https://www.qt.io/company/ (pridobljeno 15. 4. 2018).

[6] *Using the Meta-Object Compiler (moc)*. URL: https:/http://doc.qt.io/qt-5/moc.html (pridobljeno 15. 4. 2018).

[7] *Signals*. URL: https://doc.qt.io/qt-5/signalsandslots.html#signals (pridobljeno 15. 4. 2018).

[8] *Slots*. URL: https://doc.qt.io/qt-5/signalsandslots.html#slots (pridobljeno 15. 4. 2018).

[9] *C++ Mathematical Expression Parsing And Evaluation Library*. URL: http://www.partow.net/programming/exprtk/index.html (pridobljeno 15. 4. 2018).

[10] *C++ Mathematical Expression Parsing And Evaluation Library*. URL: https://github.com/ArashPartow/exprtk (pridobljeno 15. 4. 2018).

[11] *Qt and LaTeX via KLFBackend*. URL: https://wiki.qt.io/Qt_and_LaTeX_via_KLFBackend (pridobljeno 15. 4. 2018).

[12] *Artistic Style 1.24*. URL: http://www.cs.utsa.edu/~rmurphy/astyle.html (pridobljeno 15. 4. 2018).

[13] *CMake Wiki*. URL: https://cmake.org/Wiki/CMake (pridobljeno 15. 4. 2018).

[14] *The Ninja build system*. URL: https://ninja-build.org/manual.html (pridobljeno 15. 4. 2018).

[15] *clang: a C language family frontend for LLVM*. URL: https://clang.llvm.org/ (pridobljeno 15. 4. 2018).

[16] *JSON*. URL: https://json.org/ (pridobljeno 15. 4. 2018).

[17] *The JSON Data Interchange Syntax*. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (pridobljeno 15. 4. 2018).

[18] Terese.