KALKULATOR

Strokovno poročilo za 4. predmet splošne mature

Mentor: Darjan Toth, prof.

Avtor: Rok Strah, R 4. C

Ljubljana, april 2018

Povzetek

To poročilo predstavlja funkcionalnost in implementacijo programa StrahCalc. To je kalkulator, ki je poleg številskega računanja tudi zmožen simbolnega računanja in omejenega risanja grafov. Najprej poročilo opiše tehnologije in koncepte, ki so bili uporabljeni za izdelavo naloge. Opiše tudi nekaj specifičnih implementacij, nato pa še opiše sintakso jezika, ki ga kalkulator podpira.

Ključne besede

Kalkulator, Exprtk, Qt, AST, ARS, TRS, SRS, C++

Abstract

This report describes the features and implementation of a calculator StrahCalc. It also supports, along with numeric calculations, symbolic calculations and plotting graphs. The report first describes the technologies and concepts used for the project, then moves on to describe a few specific implementations. Lastly it describes the syntax of the language of the calculator.

Key words

Calculator, Exprtk, Qt, AST, ARS, TRS, SRS, C++

Kazalo

Slike

1 Uvod

Za izdelavo kalkulatorja sem se odločil, ker mi je zelo všeč „Wolfram Alpha”, vendar ni odprto-koden in se ga ne da razširjati. Zdelo se mi je, da bo to projekt, ki bo zahteval veliko načrtovanja, kar mi primanjkuje. Že na začetku, sem se odločil implementirati uporabniško definirane funkcije, ker je to res funkcionalnost, ki manjka v veliko drugih aplikacijah, ne le kalkulatorjih.

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:

- ISO/IEC 14882:1998[1] znan kot ISO C++ 1998 ali C++98 [2, 3, 4],
- ISO/IEC 14882:2003[5] znan kot ISO C++ 2003 ali C++03 [2, 3, 4],
- ISO/IEC 14882:2011[6] znan kot ISO C++ 2011, C++11 ali C++0x [2, 3, 4],
- ISO/IEC 14882:2014[7] znan kot ISO C++ 2014, C++14 ali C++1y [2, 3, 4] in
- ISO/IEC 14882:2017[8] znan kot ISO C++ 2017, C++17 ali C++1z [2, 3, 4],

sedaj pa delajo na naslednjem standardu, ki naj bi prišel v letu 2020 in je znan pod imenom C++20 [2, 3, 9].

Bjarne je leta 1979 začel razvijati „C with Classes”, razširitev C-ju in predhodnik

C++. [4] Ko je delal na doktorski nalogi, je opazil, da ima jezik „Simula” nekaj funkcij, ki so zelo uporabne za velike projekte, toda 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: [10]

- 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. [11]

Filozofija C++

- Splošna pravila: [4]
  - Evolucijo 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: [4]
  - 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: [4]
  - 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: [4]
  - 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"* [12]) 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* [13]), Qt knjižnica gradnikov (angl. *Qt Widgets*) in Qt signali in reže (angl. *Qt Signals & Slots*), s katerimi lahko nadziramo grafični vmesnik (okrajšano GUI iz angl. *Graphical User Interface*).

Signali

Qt uporabnikom zagotavlja funkcije, za povezovanje signalov na reže. [14] To sta „`connect`" in „`disconnect`". [14] Signali se ponavadi sprožijo, ko se objekt (angl. *object*) spremeni, pa bi to želeli sporočiti njegovemu staršu (angl. *parent*). [14] Ko si signal sproži se ponavadi reže povezane nanj takoj izvedejo, kot navaden klic funkcije, [14] vendar pa če uporabljamo zaporedne povezave (angl. *queued connections*) koda normalno nadaljuje z izvajanjem in se reže začnejo izvajati šele kasneje. [14]

Reže

Reže so navadne funkcije, na katere pa lahko povežemo signale. [15]

2.1.3 Knjižnice

Exprtk

Exprtk je knjižnica za računanje za C++. [16, 17]

KLFBackend

KLFBackend je C++ knjižnica, ki deluje kot vmesnik med aplikacijo in programom za izrisovanje LaTeX kode. [18]

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 Git

Git je sistem nadziranja verzij izvorne kode (angl. *version control system*), ki se od ostalih takih sistemov odlikuje s svojo hitrostjo. [19] Git je ustvaril Linus Torvalds. Beseda „git" v britansko-angleškem slengu pomeni „neprijetna oseba", Linus je pa ime razložil takole: [20]

„git" lahko pomeni karkoli, glede na tvoje razpoloženje.

- naključna kombinacija treh črk, ki jo je preprosto izgovoriti, vendar je ne dejansko uporablja noben pogost UNIX ukaz. Dejstvo, da je popačenka besede „get" je lahko, ali pa ne, povezano.
- neumen. zaničljiv in prezirajoč. preprost. Izberi si besedo iz slovarja slenga.
- „global information tracker": si dobre volje in dejansko deluje za tvoje potrebe. Angeli pojejo in svetloba nenadoma zapolni prostor.
- „goddamn idiotic truckload of sh*t": ko se pokvari

2.1.5 LaTeX

LaTeX ([lateh] iz grško $\tau\varepsilon\chi\nu\eta$) je izjemno močno programsko okolje za urejevanje (oblikovanje in izpis) besedil, ki ga je ustvaril znani ameriški matematik, računalnikar in programer Donald Knuth. Razširjen je na univerzah, še posebej se pa uporablja na področjih matematike, fizike in računalništva.

2.1.6 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.

2.1.7 Druga orodja in standardi

AStyle

AStyle je program, ki samodejno oblikuje izvorno kodo po definiciji, ki jo podaš. [21] Qt tudi omogoča povezavo z AStyle preko dodatkov (angl. *plugin*).

CMake

CMake je prosta in odprto-kodna rešitev za avtomatizacijo grajenja projektov. [22] V primerjavi s Qt-ovim qmake je močnejši, vendar zahtevnejši za uporabo.

Ninja

Ninja je prost odprto-kodni sistem za grajenja projektov. [23] Za razliko od ostalih takih sistemov je bistveno hitrejši. [23] Čeprav ga je možno uporabljati samega, je mišljeno, da se ga uporablja v povezavi z programom, ki generira „`.ninja`” datoteke, kot je CMake. [23]

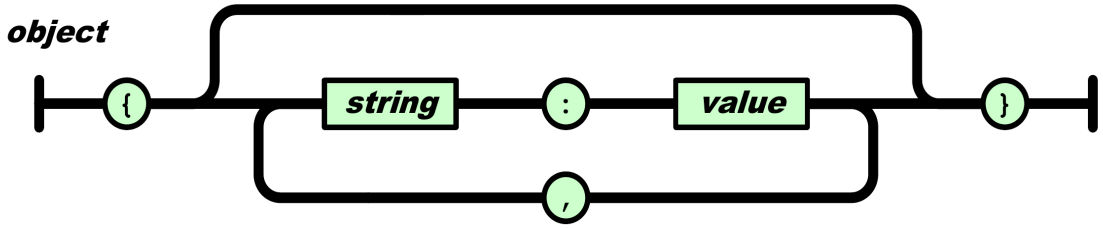
Clang

Clang je prost odprto-kodni vmesnik za jezike družine C, ki se od GCC-ja razlikuje predvsem po boljših sporočilih in hitrosti. [24]

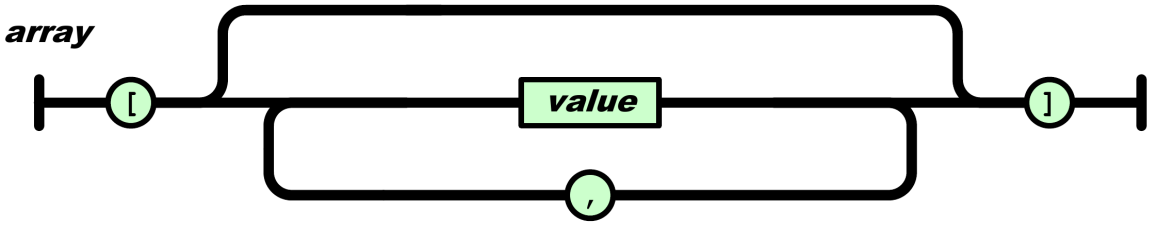
JSON

JSON (iz angl. *JavaScript Object Notation*) je način zapisa tabel in objektov v tekstovno datoteko. [25]
Struktura JSON: [25]

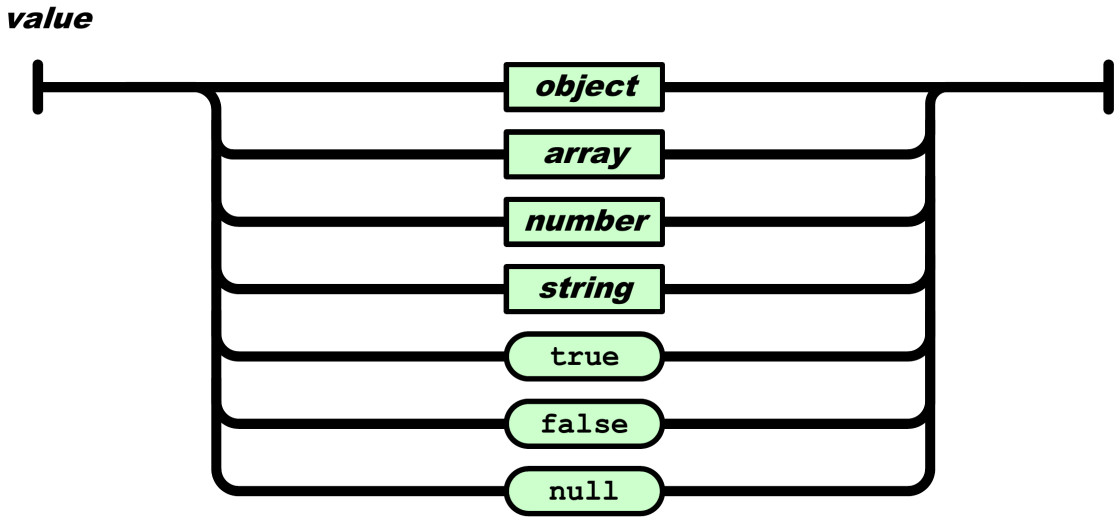
- JSON dokument je sestavljen iz objekta ali tabele.
- JSON objekt (slika 2.1) je sestavljen iz parov ključev in vrednosti.
- JSON tabela (slika 2.2) vsebuje več vrednosti.
- Ključ je niz(slika 2.4), vrednost (slika 2.3) je pa lahko niz , število (slika 2.5), tabela, objekt, dvojiška vrednost (true in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



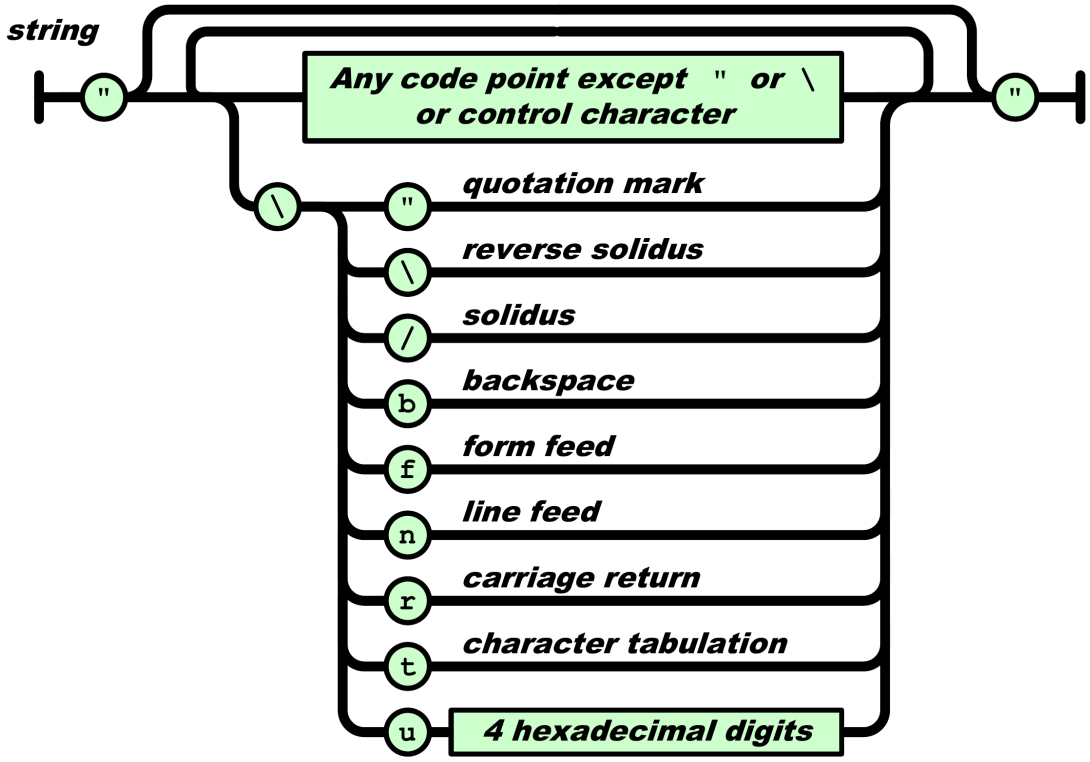
Slika 2.1: JSON objekt [26]



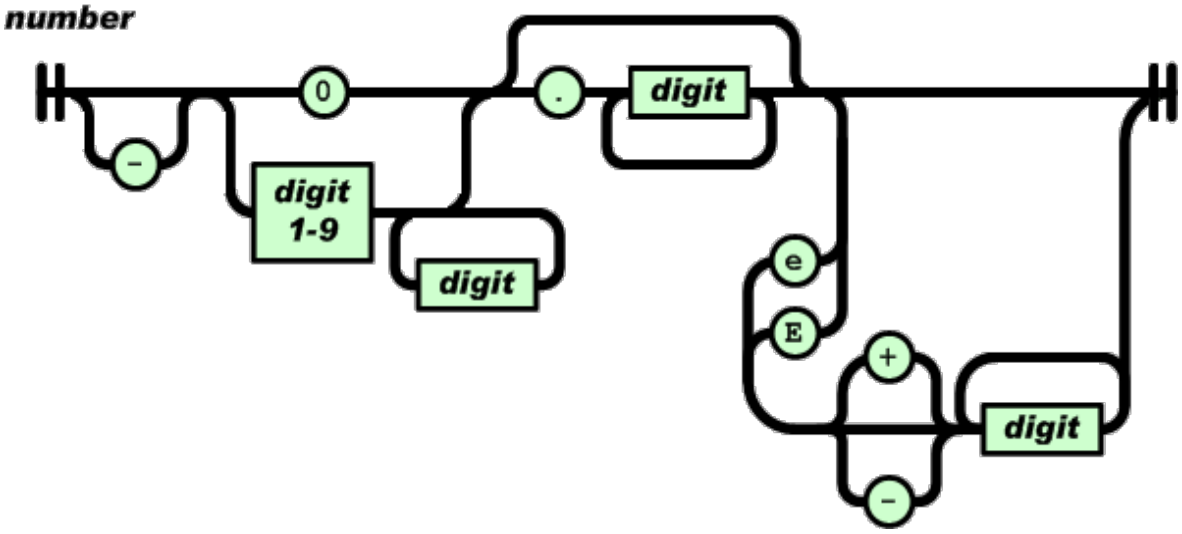
Slika 2.2: JSON tabela [26]



Slika 2.3: JSON vrednost [26]



Slika 2.4: Niz v JSON [26]



Slika 2.5: Število v JSON [26]

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” [27, 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.

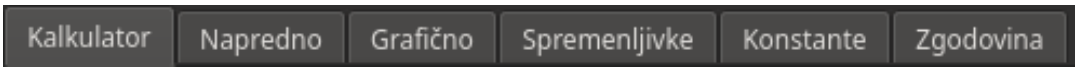
2.2.4 Abstraktno sintaktično drevo

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

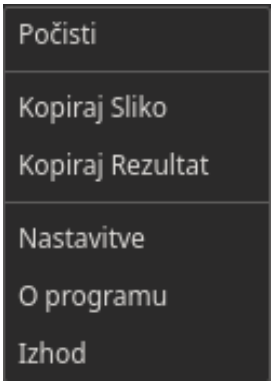
3.1 Glavno okno

Na glavnem oknu je več zavihkov (glej sliko 3.1) — „Kalkulator”, „Napredno”, „Grafično”, „Spremenljivke”, „Konstante” in „Zgodovina” — vsak s svojo funkcijo.



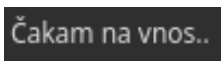
Slika 3.1: Zavihki na glavnem oknu

V zgornjem meniju najdemo nekaj orodji (glej sliko 3.2) , kot npr. „Nastavitve” ali „O programu”



Slika 3.2: Orodja v zgornjem meniju

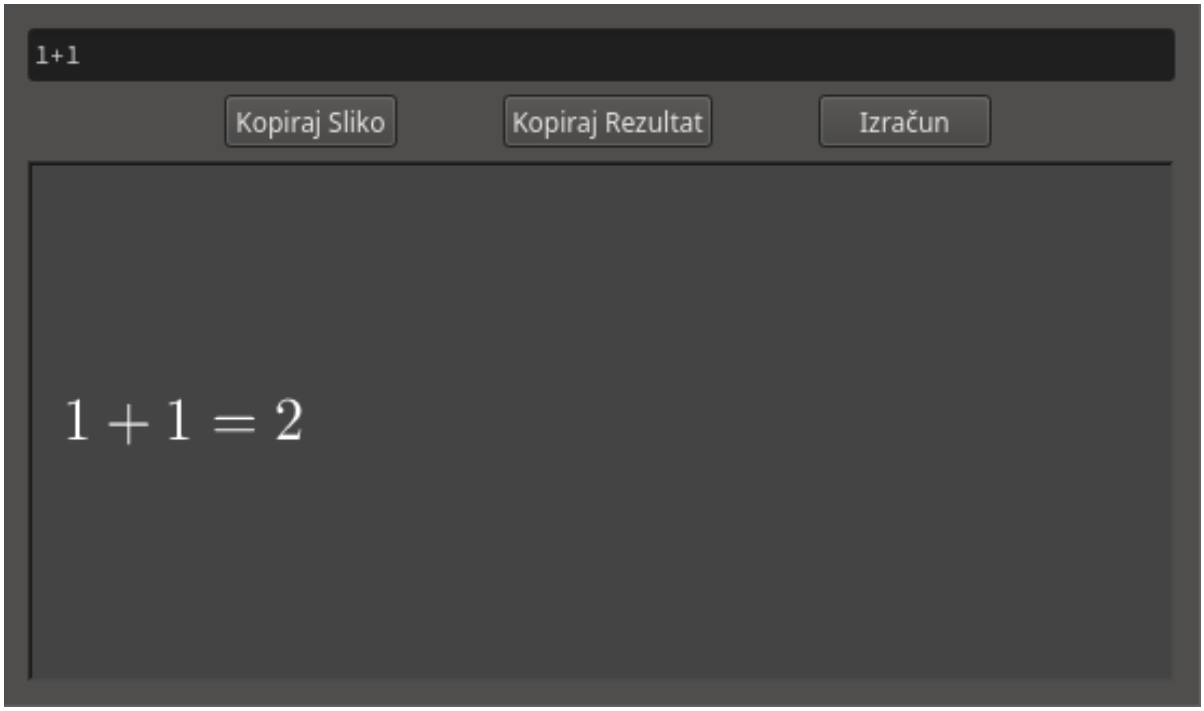
Na dnu je pa tudi viden status aplikacije (glej sliko 3.3), ki sporoča uporabniku, kaj aplikacija trenutno dela.



Slika 3.3: Trenutni status aplikacije

3.1.1 Kalkulator

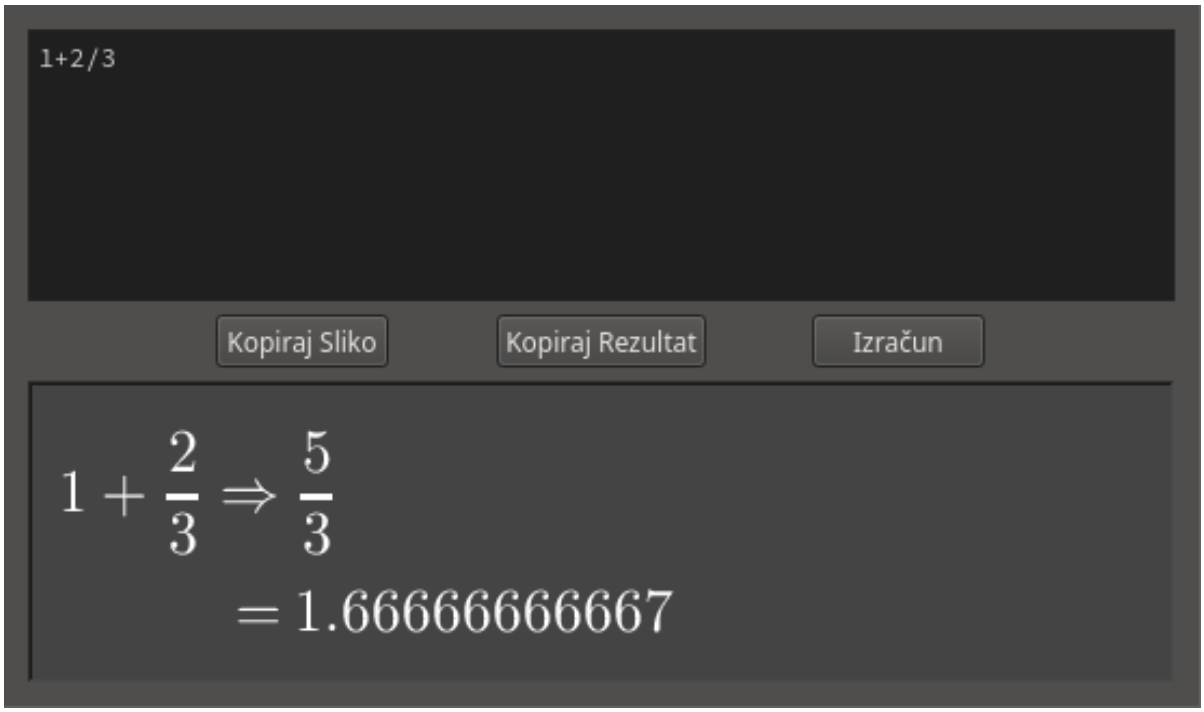
Prvi zavihek je „preprost” kalkulator. Zgoraj je vrstica za vnos računa, pod njo so pa trije gumbi, dva za kopiranje rezultata in en za začetek računanja. Računati začne tudi ob pritisku tipke „Enter”. Celoten zavihek s primerom zgleda takole:



Slika 3.4: Preprost kalkulator

3.1.2 Napredno

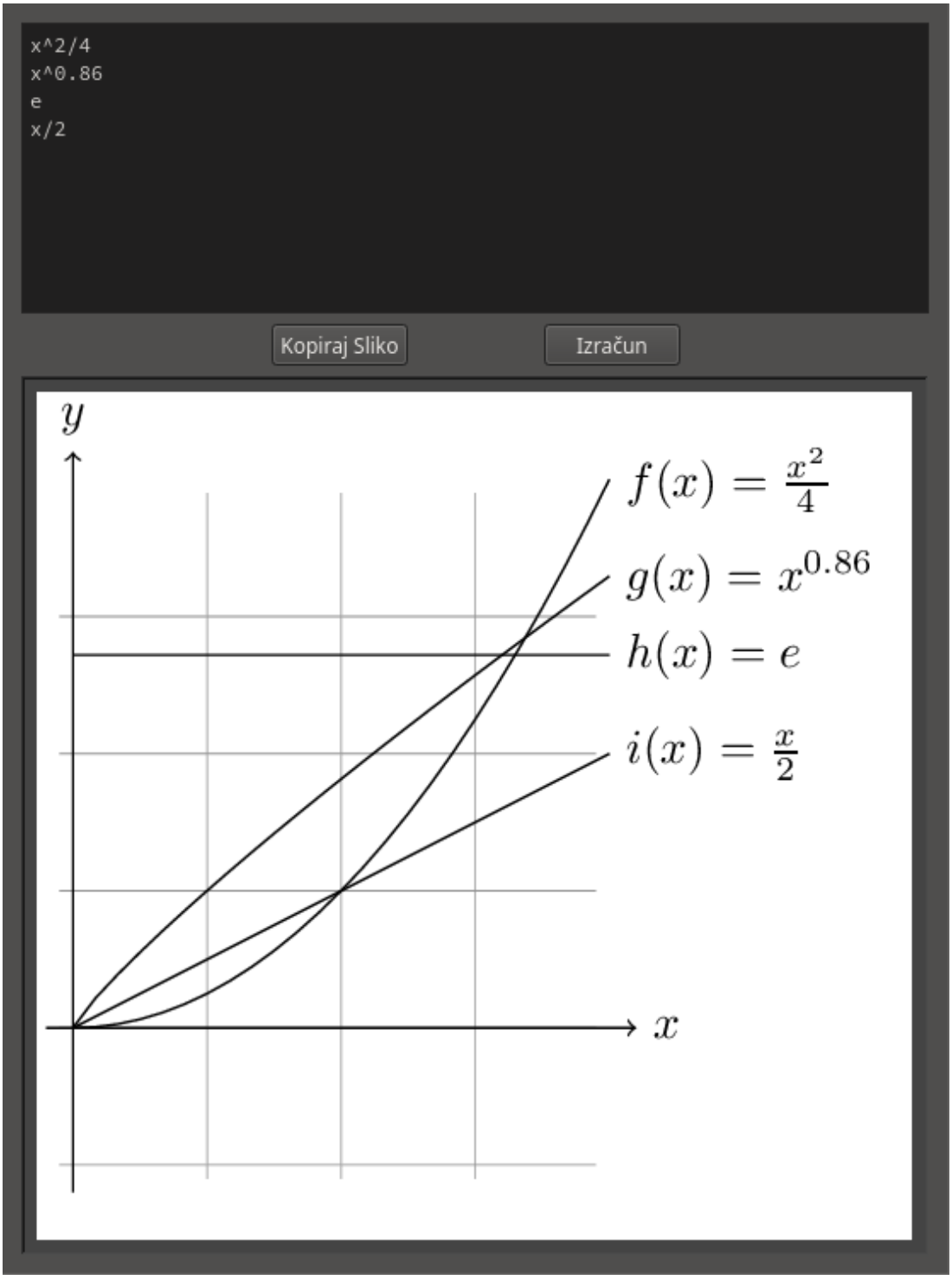
Napredni kalkulator je razširitev preprostega kalkulatorja. Vnosno polje za račun dovoljuje vnašanje večih vrstic, kar pomeni, da je možno uporabiti tudi kompleksnejše strukture kot so zanke ter pogojni stavki. Poleg rezultata se pa tudi izriše poenostavljen izraz. Celoten zavihek s primerom zgleda takole:



Slika 3.5: Napredni kalkulator

3.1.3 Grafično

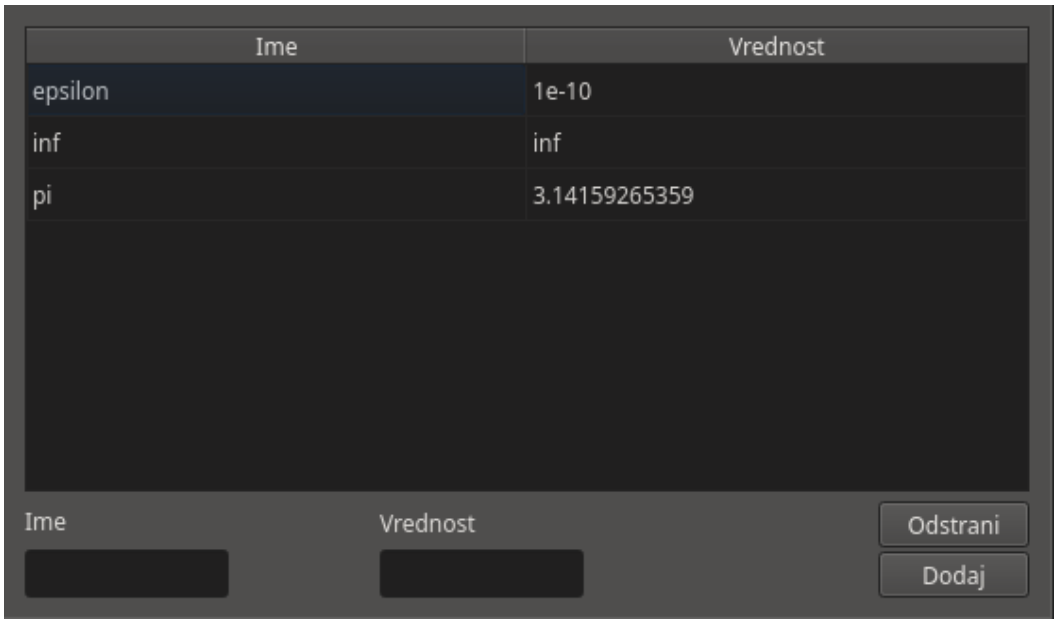
Vnosno polje je podobno kot v naprednem kalkulatorju, manjka pa gumb „Kopiraj rezultat”, saj aplikacija v tem načinu nič ne računa, ampak riše grafe funkcij, ki jih pišemo v vnosno polje. Celoten zavihek s primerom zgleda takole:



Slika 3.6: Grafični kalkulator

3.1.4 Spremenljivke in Konstante

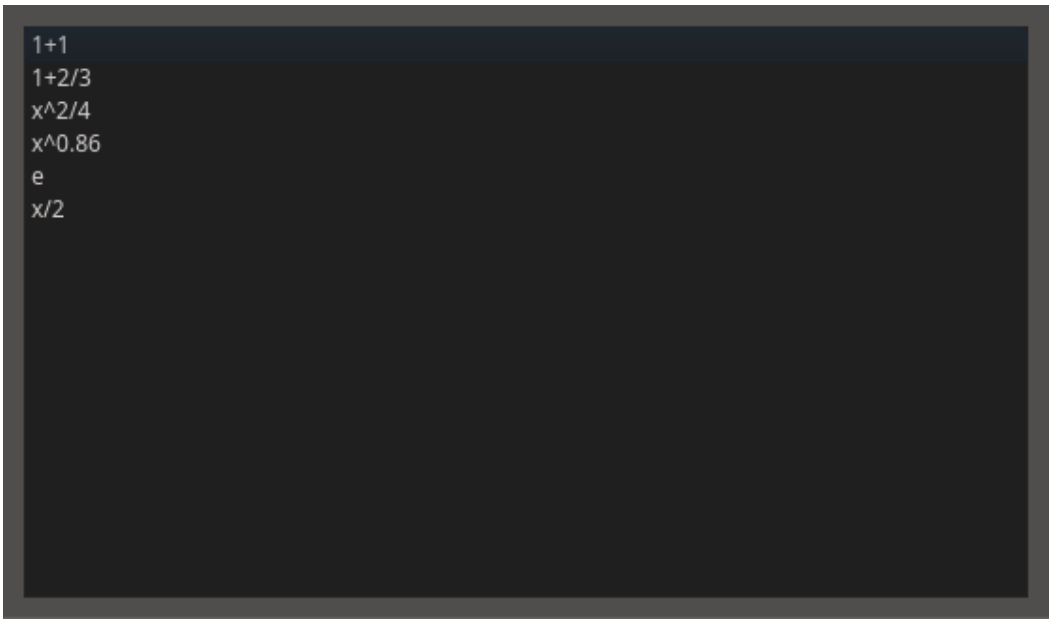
Ta dva zavihka v tabeli prikazujeta vse spremenljivke in konstante, ki so vnesene v programu. Celoten zavihek s primerom zgleda takole:



Slika 3.7: Konstante

3.1.5 Zgodovina

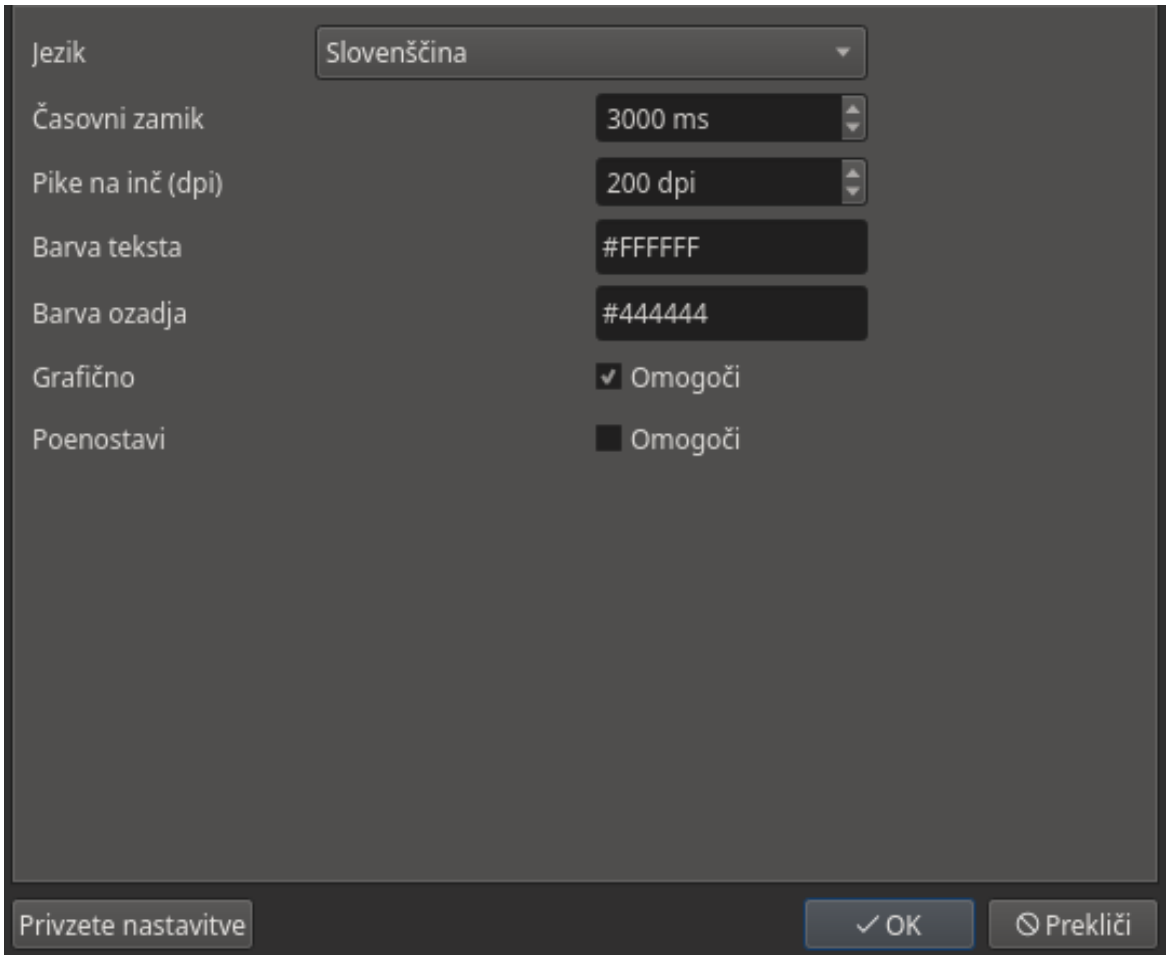
Aplikacija beleži zgodovino vnesenih računov. Celoten zavihek s primerom zgleda takole:



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. Celotno okno s privzetimi nastavitvami zgleda takole:



Slika 3.9: Nastavitve

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 Drevesna struktura programa

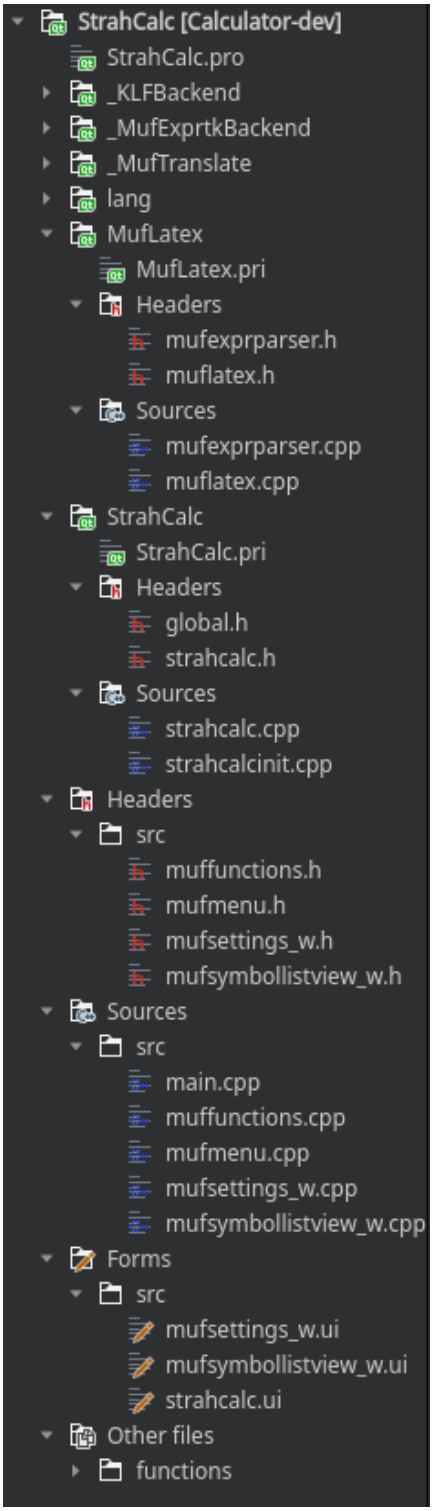
Kot vidimo na sliki 4.1 je projekt razdeljen na dva podprojekta:

1. StrahCalc in
2. MufLatex,

ter dve skupini:

1. StrahCalc in
2. MufLatex,

v drevesu je pa vključena še izvorna koda knjižnic, ki sem jih uporabil. Kot podprojekt „lang" sem dodal tudi jezikovne datoteke (za „`MufTranslate`"), kar je pospešilo prevajanje uporabniškega vmesnika.



Slika 4.1: Drevesna struktura programa

4.1.1 StrahCalc

V podprojektu StrahCalc se nahajajo datoteke:

1. **global.h**,
2. **strahcalc.h**,
3. **strahcalc.cpp** in
4. **strahcalcinit.cpp**.

V **strahcalc.h** je deklaracija razreda „`StrahCalc`", ki deduje „`QMainWindow`" in nadzoruje glavno okno aplikacije, v **strahcalc.cpp** se pa nahaja definicija tega razreda, v **strahcalcinit.cpp** pa definicije funkcij za inicializacijo opisanih v razdelku 4.2.1. V ta podprojekt sem dodal **global.h**, v katerem se nahajajo funkcije, definicije in objekti, ki jih uporabljam v več datotekah.

4.1.2 MufLatex

V podprojektu MufLatex se nahajajo datoteke:

1. **mufexprparser.h**,
2. **mufexprparser.cpp**,
3. **muflatex.h** in
4. **muflatex.cpp**.

Datoteka **mufexprparser.h** vsebuje deklaracijo razreda „`MufExprParser`”, ki je definiran v **mufexprparser.cpp**. V datoteki **muflatex.h** je deklaracija razreda „`MufLatex`”, ta ima pa definicijo v **muflatex.cpp**.

4.1.3 Headers

V skupini Headers se nahajajo datoteke:

1. **muffunctions.h**,
2. **mufmenu.h**,
3. **mufsettings_w.h** in
4. **mufsymbollistview_w.h**,

ki deklarirajo manjše razrede:

1. „`MufFuncions`”, ki skrbi za dodajanje uporabniških funkcij iz datotek,
2. „`MufMenu`”, ki skrbi za gornji meni,
3. „`MufSettings_w`”, ki skrbi za okno z nastavitvami in
4. „`MufSymbolListView_w`”, ki skrbi za prikaz spremenljivk in konstant v glavnem oknu.

4.1.4 Sources

V skupini Sources se pa nahajajo datoteke z definicijami razredov opisanih v razdelku 4.1.3:

1. **muffunctions.cpp**,
2. **mufmenu.cpp**,
3. **mufsettings_w.cpp** in
4. **mufsymbollistview_w.cpp**.

4.2 Glavno okno

4.2.1 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 prevedene ključe in pokliče podprograme štirih drugih objektov: spremenljivke, konstante, nastavitve in gornji meni, ki pa naredijo podobno. Funkcija se kliče tudi, ko spremenimo jezik.

4.2.2 Potek računanja

Za računanje so zadolžene tri skupine funkcij, vsaka za svoj način računanja. Vsaka skupina funkcij je označena s svojo končnico, -{∅} 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 4.2: 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 4.3: input.latex

4.3 MufExprParser

Razred „`MufExprParser`" v sebi definira več drugih razredov:

- `enum class Assoc` (Koda 4.1), ki oštevilči asociativnost operatorja,
- `enum class TokenType` (Koda 4.2), ki oštevilči tip tokena,
- `struct str_tok_t` (Koda 4.3), ki definira token,
- `struct pat_t` (Koda 4.4), ki definira „regex" vzorec za določen token in
- `class ExprTree` (Koda 4.5), ki je implementacija AST.

Koda 4.1: Assoc enum

```
enum class Assoc {
  NONE,
  LEFT,
  RIGHT,
  PREFIX,
  POSTFIX
};
```

Koda 4.2: TokenType enum

```
enum class TokenType {
  B, // binary op
  b, // brace
  U, // unary
  v, // value
  x, // variable
  NONE
};
```

Koda 4.3: Razred za hranjenje tokenov

```
struct str_tok_t {
  QString          s;
  Assoc            assoc;
  int              prec;
  int              rightPrec;
  int              nextPrec;
  TokenType        type;
  friend bool operator>(const str_tok_t& lhs, const
     str_tok_t& rhs);
  friend bool operator<(const str_tok_t& lhs, const
     str_tok_t& rhs);
```

```
  friend bool operator>=(const str_tok_t& lhs, const
     str_tok_t& rhs);
  friend bool operator<=(const str_tok_t& lhs, const
     str_tok_t& rhs);
  friend bool operator==(const str_tok_t& lhs, const
     str_tok_t& rhs);
  friend bool operator!=(const str_tok_t& lhs, const
     str_tok_t& rhs);
};
```

Koda 4.4: Razred za hranjenje „regex” vzorea tokena

```
struct pat_t {
  QString          str;
  str_tok_t*       op;
  QRegularExpression re;
};
```

Koda 4.5: Deklaracija AST

```
class ExprTree
{
public:
  ExprTree();
  ExprTree(const ExprTree& rhs);
  ExprTree(str_tok_t v); // set value as operator
  ExprTree(str_tok_t _op, ExprTree _operand);
  ExprTree(str_tok_t _op, ExprTree* _operand);
  ExprTree(str_tok_t _op, ExprTree _left, ExprTree
     _right);
  ExprTree(str_tok_t _op, ExprTree* _left, ExprTree*
     _right);
  ~ExprTree();

  // TODO: add int check
  // check if it's worth to convert
  int              negative();
  // check if it can be reduced to a value
  bool             isValue();
  bool             isFrac();
  bool             hasFrac();
  void             toFrac();
  double           eval();
  bool             isInt();
  bool             isOdd();
  bool             isEven();
  QString          print();
  QString          toLatex();
  void             reduce();
  void             negate();
```

```
    void            multiply(const int& factor);
    void            multiply(const QString& var);
    void            multiply(ExprTree* t);
    QStringList     var();
    QString         expr();

    str_tok_t       op;     // can be any operator or value
    QList<ExprTree*> operands;

  private:
    void            prefixUnary();
    void            setValue(const double& v);
    void            setValue(const QString& v);
    void            setChild(const int& i);

  public:
    bool            numeric;
};
```

Razred vsebuje tudi veliko metod, vendar bom napisal le tiste, ki se dejansko uporabljajo. Odstranil bom tudi definicije zgoraj navedenih razredov (Kode: 4.1, 4.2, 4.3, 4.4 in 4.5).

Koda 4.6: Deklaracija razreda „MufExprParser"

```
class MufExprParser : public QObject
{
  Q_OBJECT
public:
  enum class Assoc;
  Q_ENUM(Assoc)
  enum class TokenType;
  Q_ENUM(TokenType)
  struct str_tok_t;
  struct pat_t;

  class ExprTree;

public:
  explicit MufExprParser(QObject* parent = nullptr);
  ~MufExprParser()
  {
    if (tree != nullptr) {
      delete tree;
      tree = nullptr;
    }
  }
```

```
private:
  int               init();

  bool              expect(const QString& tok_s); // true
     if next.s == tok_s
  bool              expect(const str_tok_t& tok); // true
     if next == tok
  bool              expect(const TokenType& tok_t); //
     true if next.type == tok_t
  void              consume(); // removes next token
  str_tok_t         next();    // returns next token

  bool              tokenize(QString input);
  pat_t*            match(QString s,
                          QList<pat_t>& p,
                          str_tok_t* t,
                          QString* e);

  ExprTree*         exprTD(int p);
  ExprTree*         pTD();

public slots:
  QString           exprParseTD(QString input);
public:
  QString           operator()(QString input);

private:
  // predefined tokens
  static str_tok_t tok_end; // end token
  static str_tok_t op_sent; // sentinel
  static str_tok_t op_rbr;
  static str_tok_t op_lbr;
  static str_tok_t op_exp;
  static str_tok_t op_fac;
  static str_tok_t op_mul;
  static str_tok_t op_div;
  static str_tok_t op_mod;
  static str_tok_t op_add;
  static str_tok_t op_sub;
  static str_tok_t op_equ;
  static str_tok_t op_and;
  static str_tok_t op_or;
  static str_tok_t op_neg;
  static str_tok_t arg_pr;
  static str_tok_t arg_num;
  static str_tok_t arg_var;
  pat_t             pat_eos; // end of string pattern
  QList<pat_t>      pat_ops; // list of operator patterns
```

```
    QList<pat_t>     pat_arg; // list of arg? patterns

    QQueue<str_tok_t> tokens; // tokens to parse
    QStack<str_tok_t> mOperators;
    QStack<ExprTree> mOperands;

  public:
    ExprTree*        tree;

    bool             reduce;

    friend bool operator>(const str_tok_t& lhs, const
       str_tok_t& rhs);
  };
```

Razred definira tipične metode razredov v C++. To so konstruktor, destruktor, ki sprosti drevo iz spomina, ter metoda „`init`". Funkcija se kliče v konstruktorju in inicializira vrednosti spremenljivk. Ker so operatorji deklarirani kot statične spremenljivke, jih „`init`" skripta ne inicializira, nastavi pa začetne vrednosti članom (angl. *member*) „`pat_eos`", „`pat_ops`" in „`pat_arg`".

Pretvorbo v AST lahko kličemo na dva načina, prvi je metoda „`exprParseTD`", drugi je pa s pomočjo preobloženega operatorja „`()`", kot parameter, jima pa podamo niz z izrazom, ki ga želimo pretvoriti.

Razred si definira tudi privatne funkcije, ki olajšajo implementacijo. Metoda „`consume`" vzame token iz sklada, metoda „`next`" pa vrne token na vrhu sklada, če je pa ta prazen pa vrne „`tok_end`".

„`expect`" metode vzamejo neko obliko tokena kot parameter, in če je enak naslednjemu tokenu, le tega vzamejo iz sklada in vrnejo `true`, drugače pa vrnejo `false`.

Metoda „`match`" glede na „regex" vzorce izve kakšne vrste token je naslednji v nizu, metoda „`tokenize`" pa zaporedno kliče metodo „`match`" z različnimi vhodnimi podatki. Metodi „`exprTD`" in „`pTD`" pomagata metodi „`exprParseTD`" pri pretvorbi v AST.

5 Navodila za uporabo

5.1 Jezik kalkulatorja

Jezik kalkulatorja je enak jeziku knjižnice Exprtk. Te podatke sem preveril na 17. april 2018, za spremembe pa se nanašajte na uradno dokumentacijo knjižnice. [16, 17]

- 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`” (ali doda z uporabniškim vmesnikom) 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 Zaključek

Med izdelavo aplikacije sem se naučil veliko o orodjih, ki sem jih uporabljal, pridobil sem znanje LaTeX-a, poglobil sem znanje o avtomatskih sistemih za gradnjo projektov kot so „qmake” in „CMake”, pridobil sem pa tudi veliko pomembnih izkušenj, na

primer z uporabo „GitHub”-a za nadziranje verzij programa in z uporabo LaTeX-a v programu za izrisovanje računov, pa tudi za pisanje tega poročila.

Iz izobraževalnega vidika, sem verjetno največ pridobil z implementacijo TRS in SRS ter z implementacijo razčlenjevanja (angl. *parsing*), saj so to dokaj napredne teme, in jih je težko implementirati v tako nizko-nivojskem jeziku kot je C++.

Dobil sem malo izkušenj tudi s pisanjem knjižnic, ko sem pisal alternativo Qt razredu „`QTranslate`”.

Programu trenutno manjka stabilnost, saj ni čudno, če se sesuje ko mu podamo čuden vhod, in samostojnost, saj se zanaša na zunanje programe za prevajanje LaTeX kode. Izboljšati bi se tudi dalo grafični način, saj trenutno izrisuje samo na definicijskem območju $x \in (0,4]$ in ne bo izrisal ničesar, če katera od funkcij tam ni definirana, tako da je bolj „proof-of-concept” kot pa dejansko uporabna funkcionalnost aplikacije.

Čeprav sem nameraval narediti bolj zmogljiv kalkulator, ki bi reševal enačbe, se je izkazalo, da nimam več let in skupine doktorjev znanosti, tako da sem kar zadovoljen s funkcijami, ki mi jih je uspelo implementirati.

7 Viri

[1] *ISO/IEC 14882:1998*. URL: https://www.iso.org/standard/25845.html (pridobljeno 15. 4. 2018).

[2] *Current Status: Standard C++*. URL: https://isocpp.org/std/status (pridobljeno 15. 4. 2018).

[3] *C++ reference*. URL: https://en.cppreference.com/w/ (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] *ISO/IEC 14882:2003*. URL: https://www.iso.org/standard/38110.html (pridobljeno 15. 4. 2018).

[6] *ISO/IEC 14882:2011*. URL: https://www.iso.org/standard/50372.html (pridobljeno 15. 4. 2018).

[7] *ISO/IEC 14882:2014*. URL: https://www.iso.org/standard/64029.html (pridobljeno 15. 4. 2018).

[8] *ISO/IEC 14882:2017*. URL: https://www.iso.org/standard/68564.html (pridobljeno 15. 4. 2018).

[9] Herb Sutter. *Trip report: Summer ISO C++ standards meeting (Oulu)*. URL: https://herbsutter.com/2016/06/30/trip-report-summer-iso-c-standards-meeting-oulu/ (pridobljeno 15. 4. 2018).

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

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

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

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

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

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

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

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

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

[19] *About - git*. URL: https://git-scm.com/about (pridobljeno 16. 4. 2018).

[20] Linus Torvalds. *git/README*. URL: https://github.com/git/git/blob/e83c5163316f89bfbde7d9ab23ca2e25604af290/README (pridobljeno 16. 4. 2018).

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

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

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

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

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

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

[27] Terese. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, apr. 2003.

https://stackoverflow.com/

https://tex.stackexchange.com/

7.1 Viri slik

http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Izjava o avtorstvu

Izjavljam, da je strokovno poročilo Kalkulator v celoti moje avtorsko delo, ki sem ga izdelal samostojno s pomočjo navedene literature in pod vodstvom mentorja.

Ljubljana, 17. april 2018 Rok Strah