Sissejuhatus

Tegemist on C-keele projekti LihtsamLatex dokumentatsiooniga. Dokumentatsioon on põhjalik, sest meie projekt on palju suurem, kui C-keele aine nõuab. See on programm, mis tõlgib kasutaja lihtsa ja kiirelt kirjutatud *lähtekoodifaili* keerulisemaks LaTeXi keeles kirjutatud *tõlkeks*. Iga koht, kus meie programm tajub lähtekoodis *käsu* kasutamist, asendatakse tõlkes selle käsu definitsiooniga. Käsud on defineeritud *config* failis, kus kasutaja saab nõutud formaati järgides käske juurde lisada, vähemaks võtta ja muuta. Programm jookseb pidevalt, aga tõlkeprotsess käivitub ainult siis, kui kasutaja salvestab lähtekoodifaili. Peale igat tõlkimist programm kompileerib tõlkeks saadud LaTeXi koodi, et saada `pdf` fail. Kui kasutajal on sobiv `pdf`-viewer, siis ta saab näha, kuidas pdf värksndab peale seda, kui on lähtekoodi salvestatud.

1 Programmi kasutamine

Programmikaustaks nimetame seda kausta, mille sees on kaustad `dok`, `obj`, `src` ja `templates`.

1.1 Enne käivitamist

Programmi töötamiseks peab kasutajal installitud olema mõni TeX keele kompilaator, näiteks MiKTeX või muu selline, mis annaks sisu käsule `pdflatex`. Seda käsku kasutab programm, et peale lähtekoodi tõlgendamist ja LaTeXi koodi genereerimist see ka kompileerida pdf-failiks.

Programmikaustas on meie arvutitel kompileerimise tulemusena genereeritud fail `a.exe`, mis ongi meie programm. Võib-olla see töötab teistel arvutitel, võib-olla ei tööta. Selleks, et kindlasti töötaks, tuleb programm kasutaja arvutil lähtekoodist kompileerida. Windowsi ja Linuxi jaoks mõeldud `makefile` failid on programmikaustas.

Kasutajal on soovitatav lisada selle `a.exe` faili asukoht süsteemi `PATH` muutujasse, et programmi saaks käivitada kergelt misiganes kaustast. Programm on mõeldud kiirelt kasutamiseks ja kui seda ei saaks kiirelt käivitada, oleks kasutamiskogemus palju kehvem.

Järgmised failid on programmil alguses kaasas. Nende sisu on mõeldud kasutajale muutmiseks, aga kasutaja ei tohi neid oma kohtadelt liigutada ega kustutada. Kui neid faile ei ole sobivas kohas, siis programm ei tööta.

1. `config.txt` fail peab olema `src` kaustas koos muu koodiga. Selles on defineeritud algsed käsud.
2. Mingi nimega tekstifail peab olema `templates` kaustas. Selles on defineeritud tõlke päis ja jalus.

1.2 Kasutamine

Ilma oskamata seda programmi pole võimalik õigesti kasutada, mistõttu tuleb lugeda ka edasist dokumentatsiooni. Kui programmi kasutaks keegi, kes juba selle süsteemi tajub, siis tema teguviis oleks järgnevaga sarnane.

1. Kasutaja valib mingi nö projektikausta.
2. Ta loob sellesse kausta lähtekoodifailiks mingi tekstifaili.
3. Paneb proge käima selles kaustas.
4. Avab lähtekoodi ja hakkab sinna kirjutama.
5. Salvestab, vaatab, kuidas tema ekraani teise poole peal pdf värskendab
6. Tuleb ette koht, kus tahaks mingit käsku kasutada
7. Avab sinnasamasse kausta tekkinud `config.txt` faili ja defineerib sinna vajaliku funktsionaalsusega käsu.
8. Kirjutab ja kasutab oma defineeritud käsku lähtekoodis.
9. Vaatab, kuidas selle käsu abil keeruline latexi konstruktsioon sai lähtekoodis loodud ainult 10 tähega.
10. Life has never been so good!

2 Template failid

Esmalt tegeleme template failidega. Template folder on mõeldud kõikide erinevate template failide hoidmiseks. Hetkel on seal ainult `defaultTemplate.txt`, mille sisu võiks olla järgmine

```
\documentclass{article}
\usepackage{amsfonts}
\usepackage{amsmath}
\usepackage{amssymb}
\usepackage[estonian]{babel}

\usepackage{parskip}

\begin{document}
{{content}}
\end{document}
```

Siia saab oma valikul juurde lisada nii palju pakete, kui neid vaja võiks minna. Samuti võib kausta juurde lisada veel template faile, kusjuures nende nimed võivad olla suvalised. Igas template failis peab aga olema märksõna

```
{{content}}
```

mis enamasti käib koostatava dokumendi alustava ja lõpetava definitsiooni vahele. See koht, kus template failis on kirjas `{{content}}`, asendatakse kasutaja kirjutatud lähtekoodifaili tõlkega. Tähendab, lõpuks kompileeritava TEX-faili päis ja jalus on pärit template failist, aga sisuks on kood, mille genereerib programm tõlkides kasutaja kirjutatud lähtekoodi. Seda, kuidas kasutaja kirjutatud lähtekoodifaili tõlgitakse, kontrollib `config.txt` fail.

3 Config.txt

Nüüd saame liikuda järgmise faili juurde. Peamised käskude ja keskkondade definitsioonid, sealjuures ka muud lipud, lähevad kõik `config.txt` faili. Näiteks üks `config` fail võiks olla järgmine:

```
template = KAAREL

TEXTMODE KÄSUD
//(arg1)// -> \emph{arg1}
pealk (arg1) -> \secton{arg1}

MATHMODE KÄSUD
sum(al)(ül) -> \sum_{al}^{ül}
to -> \to
inf -> \infty
lim(arg1) -> \lim_{arg1}

KESKKONNAD
enum [multiline, end:{--}] -> \begin{enumerate} #content \end{enumerate} | (item(arg1) -> \item arg1)
```

Märgime, et see on lihtsalt näide ning et programmiga tuleb kaasa põhjalikum fail.

Välimuseltki on selge, et selle faili sisu on neljas eri osas Mida eraldavad erilised tekstid `TEXTMODE KÄSUD`, `MATHMODE KÄSUD` ja `KESKKONNAD`. Iga pealkira järel tuleb sellele vastav osa. Iga osa kirjeldatakse allpool eraldi.

3.1 Template faili nimi

Reaga `template = KAAREL` määratakse template faili nimi, mida kompileerimisel kasutatakse. Selle faili nimi peab olema kirjutatud ilma `.txt` laienduseta. Samanimeline fail peab leiduma kaustas nimega `templates`. Näite puhul peab leiduma selles kaustas fail `KAAREL.txt`.

3.2 Mathmode käsud

Peamine asi, mis meie programmiga teeb LATEXi kirjutamist kiiremaks on käskude defineerimine. Nendega on võimalik suhteliselt lihtsal moel kirjutatud tekst asendada keerulise LATEXi koodiga. Kasutaja saab defineerida kaks komplekti käske. Esimene neist on mõeldud kasutamiseks olles mathmode sees. Teine on kasutamiseks textmode sees. Esmalt kirjeldame mathmode käske.

Lähtekoodis mathmode'i avamiseks ja lõpetamiseks on ainuke valik kasutada tähejärjendit `mm`. See tähendab ka seda, et nii tekstisisese matemaatika (LATEXis dollarid) kui displaymath'i (LATEXis kaldkriips ja kandilised sulud) tegemiseks peab kasutama sama alustavat ja lõpetavat sümbolite järjendit, kusjuures displaymath'i jaoks peab alustav `mm` olema uuel real. Lähtekoodis näeks seega matemaatika kirjutamine välja selline:

```
mm Siin on matemaatiline tekst, mis võibolla kasutab ka sümboleid ning kasutaja defineeritud käske. mm
```

Edasi selgitame käske. Käsu definitsiooni saame jaotada kaheks pooleks

```
Tekst, mida lähtekoodis otsitakse. -> Tekst, mis sellega asendatakse.
```

Niimoodi saame defineerida lihtsad tekstasenduse käsud, näiteks

```
alfa -> \alpha
kord -> \cdot
RR -> \mathbb{R}
```

Vahel sellest aga ei piisa. Peame suutma ka defineerida integraale, summasid ning muid abikäske, mis võtavad endale teatud arv argumente. Selleks saab defineeritavas käsus anda sisse n argumenti, kujul

```
käsuNimi(arg1)[arg2]... -> ℱ(arg1, arg2,...)
```

kus

```
ℱ(arg1, arg2,...)
```

on vastava käsu LaTeXi definitsioon, mis sisaldab endas vajalikke argumente (kusjuures need peavad olema sama nimega, mis käsu definitsiooni vasakul pool). Samuti tuleb eristada käsu defineerimisel argumentide tüüpe. Nendeks on

- `(argumendiNimi)` – Ümarad sulud tähendavad pikemat argumenti. See tähendab, et lähtekoodis argumendi lõppu tähistab ainult tühik või rea lõpp. Sellele vastandub lühem argumenditüüp.
- `[argumendiNimi]` – Kandilised sulud tähendavad lühemat argumenti. See tähendab, et lähtekoodis tähistavad argumendi lõppu peale tühiku ka tähemärgid `+`, `-`, `*`, `=` ja `,`. Tihtipeale ei ole vahet, kas definitsioonis on argumenditüüp märgitud pikaks või lühikeseks. Vahe tuleb aga sisse siis kui tahetakse näiteks kirjutada polünoome. Definitsiooni `^(arg1) -> ^{arg1}` puhul tõlgitakse lähtekoodi tekst `a^n+b^m` koodiks

  ```
  a^{n+b^{m}}
  ```

  Polünoomi kirjutamise soovi jaoks see on vale. Õige tõlge oleks `a^{n}+b^{m}`, mis saavutatakse, kui definitsioonis oleks kasutatud kandilisi sulgusid järgmiselt `^[arg1] -> ^{arg1}`.

Niisiis toome mõned näited

```
sum(al)(ül) -> \sum_{al}^{ül}
   sin(uuga) -> \sin{uuga}
 ^[mingiAsi] -> ^{mingiAsi}
```

On käks käsku, mida defineerima ei pea. Esimene on jagamine:

```
a/b -> \frac{a}{b}
```

ning teine on muutujate järgi tuletise võtmine `tul`:

```
f tulaabc... -> f''''_{a^{2}bc}
```

3.3 Textmode käsud

Mathmode sees kasutamiseks mõeldud mathmode käskudele lisaks saab kasutaja defineerida texmode käskude komplekti, mille käske otsitakse tavalise teksti seest. Näiteks on textmode käske ideaalne kasutada pealkirjade kiireks vormistamiseks, kui on vaja korduvalt panna mingit teksti LaTeXi `center` keskkonna sisse.

Kuna textmode käsud on mõneti nüansikamad, kui mathmodekäsud, siis on parim neid selgitada näitega. Allpool on esitatud üks keskmise käsu definitsioon. Numbritega on tähistatud selle definitsiooni osad ja seejärel selgitatakse, millega on tegu.

```
      1       2   3    2 4 5  6  5  8 _________9_________________   10   _9_  11 ______9_________
edevpealkiri (pealkiri)   (alune) -> \begin{center}\n\section*{pealkiri}\nalune\n\end{center}\n
                                  7                12                   12     12           12
```

1. Tegu on käsku alustava tekstiga. Kui see tekst lähtekoodist leitakse, ss sellele järgnevat teksti tõlgitakse erilisel moel nagu edasises kirjeldatud on. Panna tähele, et näidatud juhul on käsku alustavaks tekstiks `edevpealkiri ` – sellel on viimane täht tühik.
2. Tegu on esimest argumenti avava ja sulgeva suluga. Argumente eraldatakse muust käsudefinitsiooni tekstist neid ümbritsevate ümarsulgude järgi. Seega ei või käsu nimes ümarsulgi kasutada, sest asja võidaks tõlgendada valesti.
3. See on esimese argumendi nimi. See tähistab teksti, mida kasutaja kirjutab lähtekoodis käsku alustava teksti järele kuni selle tekstini, mis on argumenti sulgeva sulu järel (4). Argumente saab käsul olla praktiliselt lõputu kogus.
4. Tegu on esimest argumenti lõpetava tekstiga. Lähtekoodis käsku alustava teksti ja esimet argumenti lõpetava teksti vahele jääv tekst loetakse esimeseks argumendiks ja see asendatakse käsu definitsiooni paremas pooles kohale, kus on esimese argumendi nimi (10). Selles näites on mõeldud, et esimene argument läheb latexi sectioni loogeliste sulgude vahele.
5. Need sulud eraldavad muust definitsioonist teise argumendi nime (6). Järelikult kasutaja tahab peale esimese argumendi lõppu ka teise argumendi anda.
6. See on teise argumendi nimi. See tähistab teksti, mis jääb lähtekoodis esimest argumenti lõpetava teksti (4) ja teist argumenti lõpetava teksti (7) vahele.
7. See on teist argumenti lõpetav tekst. Näites on teist argumenti lõpetav tekst tühi sõne. Kui kumbagi argumenti lõpetav tekst on tühi sõne, siis loetakse vastava argumendi lõpuks realõpp e uuereamärk.
8. Tegu on tähistusega, mis eraldab definitsiooni vasakut poolt paremast poolest. Selleks on tekst `  ->  ` – selles on tühikud kummalgi pool. Oluline on meeles pidada, et tühikud on selle tähistuse osa. Vasakul on kirjeldatud, kuidas käsk algab, kuidas argumendid eraldatud on ja kui palju neid on. Parem pool aga kirjeldab seda, milliseks tekstiks vasaku poole järgi vormistatud tekst lähtekoodis tõlgendada.
9. Kõik see tekst on see, mis lähtekoodis kirjutatud käsualguse teksti asemel pannakse LaTeXi faili. Kohtades, kus käsu paremal poolel on kirjas argumentide nimed, saadetakse LaTeXi faili tekstid, mis loeti argumentideks.
10. Esimene argument pannakse kohale (10) ja teine kohale (11).
12. Textmodekäsu paremale poolele saab kirjutada teksti `\n`, mis on eriline tekstilõik, mille asemel pannakse LaTeXi faili reavahetusmärk, mitte tekst `\n`. Kui tahta tõlkesse panna teksti `\n`, siis tuleb kirjutada `\\n`

Selle käsu kasutamine lähtekoodifailis näeks muu teksti seas välja järgmine.

```
Tere mina olen Kaarel ja see on minu fancy ass pealkiri.
edevpealkiri MIMMA 6. kodune töö   Kaarel Parve, 3. variant, 15.12.2024
Uuga buuga veel matemaatilist juttu mitme muutuja funktsionaalreak koondumine blaa.
```

Ja see tõlgitakse järgmiseks LaTeXi koodiks.

```
Tere mina olen Kaarel ja see on minu fancy ass pealkiri.
\begin{center}
\section*{MIMMA 6. kodune töö}
Kaarel Parve, 3. variant, 15.12.2024
\end{center}
Uuga buuga veel matemaatilist juttu mitme muutuja funktsionaalreak koondumine blaa.
```

3.4 Keskkonnad

Keskkonnad nagu keerulisema süntaksiga textmodekäsud, aga keerulisema süntaksi arvelt on nendega võimalik teha rohkem kui textmodekäskudega. Keskkonnad on mõeldud LaTeXi selliste keskkondade jaoks nagu `itemize`, `enumerate` või `matrix`, milles on selline olukord, kus iga rea puhul on vaja teha midagi ühesugust. `itemize` ja `enumerate` keskkondades on iga rea ette vaja panna `\item`, `matrix` keskkonna puhul on iga rea lõppu vaja panna `\\`. Selleks, et selline igarealine käitumine saavutada, kutsub keskkond kas automaatselt iga rea peal või kasutaja valitud ridade peal välja kasutaja defineeritud käsu. Selle käsu abil on võimalik iga rea peal nimetatud juhtude katmiseks vajalikku käitumist saada.

Kuigi keskkonnakäske rakendatakse teksti peal, on need töötamise poolest lähemal mathmodekäskudele oma lihtsakoelisuse poolest. Näiteks ei saa valida keskkonnakäsu jaoks argumendilõppe nagu textmodekäskudele,

vaid iga argumendi lõpuks on fikseeritult kolm tühikut: `   `. Esitame keskkonna defineerimise süntaksi ning seejärel selgitame igat osa eraldi:

```
keskkonnaNimi [lipp1,...,lippN, end:{lõpetav sümbol}] ->
                        -> \begin{keskkonna algus} #content \end{keskkonna lõpp} | (käsk1) ... (käskM)
```

Definitsioonis on järgmised osad:

1. `keskkonnaNimi` – see on tekst, mida otsitakse lähtefailis ning mille nägemisel on selge, et siit algab uus keskkond. Kõik sellele järgnev tekst loetakse keskkonna osaks.
2. `[lipud]` – pärast keskkonna nime antakse kandiliste sulgude vahel kõik vajalikud lipud, mis keskkonna kohta käivad, kusjuures nende järjekord sulgude vahel ei ole oluline. Lippudeks on:
   (a) `end:{lõpetav sümbol}` – see on ainus **kohustuslik** lipp. Sellega antakse tähemärkide järjend, mille abil on teada, et keskkond on lõppenud.
   (b) `multiline` – selle lipu olemasolu tähendab, et mitmerealine tekst saab minna keskkonna käsu argumendiks. See tähendab, et näiteks `itemize` keskkonna sisse saab kirjutada displaymathi, mis meie süsteemi järgi peab algama uuelt realt. Seega multiline lipu olemasolul ei kutsuta keskkonnakäske välja automaatselt ja igal real, vaid kasutajalt oodatakse, et tema ise kutsub keskkonnakäsku välja seal, kus ta vajalikuks peab. Kui rea ees tuntud käsku ei ole, siis eeldatakse, et see rida on jätk viimasena välja kutsutud käsule. Kui multiline lipp puudub, siis on kaks võimalust: kui keskkonnas on defineeritud kuni üks käsk, siis ei ole tarvis iga rea ette käsu nime kirjutada, see pannakse sinna automaatselt. Kui on aga rohkem kui üks käsk defineeritud, siis peab **igal real** välja kutsuma mõne käsu, sest multiline lipp puudub.

      Vaatame mõnda näidet. Olgu `config` failis defineeritud keskkond:

      ```
      enum [multiline, end:{--}] -> \begin{enumerate} #content \end{enumerate} | (item(arg1) -> \item arg1)
      ```

      Et multiline on olemas, siis eeldatakse, et käsk kutsutakse ainult siis välja, kui soovitakse seda kasutada. Niisiis võime lähtekoodis kirjutada:

      ```
      enum
      item Tekst!
      Siin on veel teksti. Kusjuures see lisatakse eelmise käsu otsa.
      Siin on veel teksti. Ikkagi lisatakse see viimati kutsutud käsule otsa.
      item Siit algab alles uus käsk!
      --
      ```

      mille tulemus on:

      ```
      \begin{enumerate}
      \item Tekst! Siin on veel teksti. Kusjuures see lisatakse eelmise käsu otsa. Siin on veel teksti.
       Ikkagi lisatakse see viimati kutsutud käsule otsa.
      \item Siit algab alles uus käsk!
      \end{enumerate}
      ```

      Olgu nüüd defineeritud keskkond:

      ```
      mat2 [end:{--}] -> \begin{pmatrix} #content \end{pmatrix} | (r(el1)(el2) -> el1 & el2 \\)
      ```

      Et multiline lippu pole ja et on defineeritud ainult üks käsk `r`, siis ei pea me ühelgil real käsku välja kutsuma, seda tehakse meie eest. Meeldetuletus, et keskkonnakäskudel mitmeargumendilise käsu iga argumendilõpp on kolm tühikut. Kirjutades lähtekoodis:

      ```
      mat2
      2   5
      98   4
      --
      ```

      saame tulemuseks

```
\begin{pmatrix}
2 & 5 \\
98 & 4 \\
\end{pmatrix}
```

Märgime ka, et isegi kui on defineeritud ainult üks käsk ja multiline lippu pole, võib ikkagi igal real kutsuda välja keskkonna ainukest käsku, see ei muuda tulemust.

3. `\begin{keskkonnanimi} #content \end{keskkonnanimi}` – käskude `begin` ja `end` sisse tuleb panna keskkonna nimi nii, nagu see on LaTeXis
4. `(käsk1) ... (käskM)` – Püstkriipsu taga saab defineerida kõik käsud, mida keskkonna sees kasutada saab, iga käsk eraldi sulupaari `()` vahele.

4 Mis tuleks juurde lisada

Kuigi see programm on täiuslikkusele lähemal kui ei midagi muud, saaks seda siiski paremaks teha mitmel moel.

1. Ei saa lihtsal moel kirjutada käsunimesid sisaldavat teksti ilma, et neid käske tõlgitaks. Näiteks selline olukord, kus tahaks meie programmi kasutada selleks, et kirjutada dokumenti, milles oleks näide mingist võimalikust meie programmi tõlgitavast lähtekoodist. Lahendamiseks on võimalus lisada textmodekäsu definitsiooni selline süntaks, kus argumenti avava sulu ette saab panna hüüumärki näitamaks, et sellele argumendile lähtekoodist omistatavat väärtust ei tohi rekursiivselt tõkida ja et see peab minema täht tähelt tõlkesse.
2. Mathmodes sulgemata sulg crashib programmi. Tuleb teha midagi selleks, et programm vähemalt ei paneks end kinni.
3. Keskkondasid ei saa üksteise sisse panna.
4. Juhul, kui lähtekood on täiesti thi, võiks latexi dokumenti ikkagi mingi täht minna, et pdf kompileeriks edukalt. See ei eruta, kui peale tühja faili loomist proget käima pannes loodud pdf-i ei saa avada.
5. Programm käima pannes loetakse template failiks see, mis programmikaustas default configis on ja seda ei saa üldse muuta programmi jooksu käigus.
6. Sulgusid saab panna lähtekoodis argumentide ümber, et vältida nende liiga vara lõppemist juhul, kui see argument sisaldab tühikut. Sel juhul, kui sulud on lisatud selleks, et vältida argumendi varem lõppemist, ei tohi need sulud tõlkesse minna. Praegu ikkagi lähevad lühikeste argumentide puhul. Pikkade puhul on fine.s