Obligatorisk innlevering 2 høsten 2014, INF3331

Nicolai Skogheim <nicolai.skogheim@gmail.com>

October 24, 2014

1 Oversikt

Oblig 2 er delt inn i fire deler: Meta, PreTeX, resources og tests. PreTeX er den viktigste og er der hoveddelen av oppgaven er løst. All preprosessering skjer det. Resrources inneholder alt som fulgte med oppgaven. Tests er mappen som inneholder alt av tester, inndelt i unit-tester og end-to-end-tester. Meta er bare en mappe hvor jeg samlet tankene mine planla programmet før og underveis, og kan trygt oversees.

Jeg har gjort alle oppgavene, men siden de henger veldig sammen med hverandre kommer jeg ikke til å ta for meg oppgavene hver for seg.

2 prepro.py

Denne filen består stort sett av tre ting: en front end, en scanner, og behandlere / "handlers". Brukerhjelp fåes ved å kjøre programmet med `-h`-flagget.

2.1 Frontend

Denne delen, som finnes nederst i filen, sørger for et ålreit brukergrenesnitt slik som det er bedt om i oppgave 11, med mulighet for verbose/quiet modus blant annet. Denne delen administrerer også kjøringen av preprosessoren, slik som man ville anta, i tillegg til å flytte kjøringen/bytte mappe til der hvor latex-filen finnes.

2.2 Scanner

Denne klassen tar en tekst, itererer over linjene og returnerer en ferdig preprosessert tekst. Prosessen (forhåpentligvis enkelt og) kort forklart er slik: Hvis en linje ikke begynner med tegnene "%@" legges linja til output. Hvis en linje begynner med disse tegnene starter en loop som itererer over over handlers. Inputlinjen blir testet på en handler ved å kalle handler.wants(input). Når en handler vil ha linjen, sjekkes handler.multiline for at scanneren skal vite om den skal plukke opp flere linjer. Hvis ja, så fanger scanneren opp linjer til den finner

en linje lik handler.endtoken og kaller så handler.handle med input. Hvis multiline er False kalles handler.handle bare med den ene linja. Resultatet fra en handler sendes rett inn til scanner for å lete etter prepro-tags inne i resultatet (typisk etter et treff på %@show), og resultatet av dette igjen legges til output.

2.3 Handlers

Hver handler har en wants-metode som returnerer True for input den "vil ha", altså input som matcher en regex. Videre har den en handle-metode som gjør prosesseringen. Resultatet fåes ved å kalle handler.output, alternativt med en funksjon som behandler output før det returneres (typisk en funksjon som pakker resultatet inn i latex.)

Alle handler'e arver fra klassen Handler, og får blant annet wants-metoden og output-metoden som er lik for alle handlers. Altså, funksjonaliteten er lik for hver handler, men hver handler har variabler for multiline, regex osv som gjør at wants- og output-metodene oppfører seg forskjellig.

Til sist (rettere sagt nesten først/øverst) er en klasse Handlers. Denne kan brukes i en loop for å iterere over handler'ene, og det er denne klassen scanneren kaller.

3 compile.py

Denne filen består stort sett av tre deler: frontend, kompilator, og en funksjon for å behandle output fra kompilatoren.

3.1 Frontend

Fontenden gjør akkurat det samme som frontenden til prepro i tillegg til å gi mulighet for å velge om man vil kjøre pdftex med eller uten –interaction=nonstopmode.

3.2 compile_latex

Kompilatoren er en enkel wrapper for pdftex, og returnerer kun output fra den kommandoen i tillegg til å printe ut eventuelle feilmeldinger.

3.3 parse_output

Denne tar tekst fra kompilatoren, hanter ut linjer som er interessante, og bytter ut linjenummere så de peker på den originale uprosesserte filen.

4 latex.py

Denne filen inneholder funksjoner for å pakke inn tekst i latex. Ved å bruke funksjonen `configure` kan man instille denne filen til å alltid returnere enkel

latex (typ verbatim). Dette er gjort ved at alle wrappere er dekorert med `mode`-funksjonen som rett og slett bare overstyrer alle wrapper'ene med verbatim-wrapperen hvis "simpleMode" er True.

5 line_number_map.py

Jeg ville ikke søple til den preprosesserte fila med kommentarer, så jeg valgte heller å ha en map i slutten av fila, med informasjon om hvor hver linje kom fra. Denne funksjonaliteten er litt komplisert så derfor er den i en egen fil, slik at man skal slippe å tenke så mye på hvordan dette fungerer når man leser prepro.py og compile.py. Løsningen endte opp med å bli litt for magisk for min smak, og derfor vil jeg heller ikke gå inn på hvordan det fungerer.

6 helper.py

Denne filen inneholder funksjonalitet som er ofte brukt rundt om kring. Innlasting og skriving av filer, samt kjøring av kode, og til slutt en funksjon som tar tekst og en regex og returnerer hele treffet.

Når det er gjort på denne måten har man også full kontroll, på et sted, over hvordan disse tingene skjer ettersom all skriving og lesing av filer etc. skjer gjennom disse hjelperne. Skulle man senere bestemme seg for kun å tillate å kjøre bash-kode, eller nekte å kjøre `rm -rf`, kan man enkelt kontrollere dette her.

7 Testing

Testene er delt opp i unit-tester og end-to-end-tester. Jeg har prøvd å holde programkoden testbar og fleksibel, men jeg er ikke fornøyd med hvordan testene ser ut nå.

For å kjøre testene trenger py.test og coverage å være installert som kan installeres med pip, eller for eksempel brew hvis du er på mac. Til selve konverteringen bruker jeg pdftex.

Testene og coverage rapport kan kjøres med følgende kommandoer

Terminal

```
cd oblig02
coverage run -a --branch --source=PreTeX `which py.test` -vv
coverage run -a --branch PreTeX/prepro.py tests/e2e/testfiles/tex_before.xtex /dev/null -q
coverage run -a --branch PreTeX/compile.py tests/e2e/testfiles/tex_after.tex -q
coverage report -m --fail-under=85 --omit="*__init__.py","*argparse*"
```

`-q`-flagget stopper output, og kan fjernes for å se hvordan det ser ut når programmene printer feilmeldinger. Om det legges til et `-v`-flagg aktivieres verbose modus, og man får mye informasjon om hva som skjer.

Om du ikke innstallerer coverage kan du fortsatt kjøre testene vanlig med pytest. cd oblig02 py.test For morroskyld har jeg brukt travis-ci for å kjøre testene mine. Hva som kjøres på travis kan man se i konfigurasjonsfilen .travis.yml i rota av prosjektet. Hvis du ikke har hørt om travis før anbefaler jeg å gå til repoet mitt på github, se på README'en, og trykke på knappen hvor det står "build passing". Du taes da til travis, hvor du må logge inn med github-kontoen din. Trykk på knappen i README'en en gang til når du er logget inn for å komme til teste-siden min. Trykker du da på en av jobbene får du se testresultater og coverage.

Hva gjelder doctesting har vurdert det dithen at det ikke er noe av funksjonaliteten i programmet som er hensiktsmessig å teste på denne måten.

8 Avvik fra oppgaven

Jeg gjør oppmerksom på at det er tatt enkelte valg som viker far oppgaveteksten.
En av de viktigere er at syntaksen for importering av .tex-filer er `%@include{sti/til/fil.tex}` i stedet for `\include{sti/til/fil.tex}`

9 Epilog

Jeg kunne skrevet mye mer om programmet og hvordan oppgavene er løst, men jeg tror leser vil ha mer utbytte av å bare lese koden. Jeg håper og tror at koden tydelig kommuniserer hva den gjør, men setter stor pris på tilbakemeldinger rundt dette. Ved å titte på tex_before.xtex og tex_after.tex i tests/e2e/testfiles kan man se hvordan output fra prepro ser ut, og etter å ha kjørt påfølgende kommando kan man se pdf'en som blir generert.

Terminal

```
python PreTeX/prepro.py tests/e2e/testfiles/tex_before.xtex tex_after.tex -q
```