2. obligatoriske innlevering, høsten 2014

{Jonathan Feinberg, Joakim Sundnes}
{jonathf,sundnes}@simula.no

October 21, 2014

Innleveringskrav

Denne skal følge malen gitt på emnesidene (Legges ut 2. september). Innlevering skjer ved opplasting til github. Mer informasjon om klasseromsløsningen vi bruker på github vil bli gitt i god tid før innleveringsfristen.

Oversikt

I denne obligen ønsker vi å automatisere preprosessering av LaTeX-dokumenter. For å beskrive hva preprosessoren skal gjøre, la oss begynne med et eksempel. Ta utgangspunk at mens man skriver et LaTeX-dokument `text.tex`. I dette dokumentet ønsker vi å inkludere en kode-snutt fra en kode-fil som er i samme mappe. Enkleste løsningen er å kopiere over innholdet fra filen manuelt, plassere snuttene `\begin{verbatim}` og `\end{verbatim}` på hver side av innlagt kode og si seg ferdig. Problemet med dette er at hvis man oppdaterer koden senere, må man samtidig gjøre ekstraarbeidet med å oppdatere LaTeX-dokumentet i samme slengen. I større prosjekter (som f.eks. masteroppgaver) kan dette fort bety veldig mye unødig ekstra-arbeid. Istedenfor å skrive inn ting manuelt, kand det være hensiktsmessig å importere ting direkte fra LaTeX. Det finnes noen løsninger internt i LaTeX, men de er klumsete og ikke spesielt kraftige til det behovet en informatikker skulle ha. Det er her konseptet preprosessor kommer inn. Målet med preprosessor er å gi LaTeX-dokumenteter ekstre funksjonalitet direkte nyttig i informatikksammenheng.

Deloppgavene under beskriver forskjellig funksjonalitet man kan legge inn. Det er 13 oppgaver hvorav 7 er obligatoriske. Av de resterende 6, skal man velge seg ut 3 oppgaver. De obligatoriske oppgavene er alle markert med '*'.

*Oppgave 1: Kodeimport

Istedenfor å importere kode manuelt, la oss istedenfor anta at vi kan skrive inn følgende liten snutt der vi ønsker å legge inn kode:

```
%@import script_example.py ( *out =(.|\n)*?return out)
```

Programmet dere skal skrive skal kunne finne snutter som dette inne i et LaTeX-dokument, åpne filen den referer til (`script_example.py`), finn kode fra regex (`( *out =(.|\n)*?return out)`) og lime dette tilbake inn i dokumentet hvor `%@import` ble kalt fra. Kode snutten vil dermed i sin enkleste form se ut som følger:

```
\begin{verbatim}
    out = 1.
    for o in xrange(2, order+1):
        out *= o
    return out
\end{verbatim}
```

Bruk gjerne flagget `re.MULTILINE` for å gjøre regular expression uttrykket enklere.

*Oppgave 2: Eksekvering av skript

I tillegg til viser biter av en kode er det av interesse å vise litt av hva et skript kan gjøre. Programmet dere skal lage må kunne kjøre kode til eksterne skript, returnere utskrift av kjøring og lime dette inn i LaTeX-dokumentet. I vårt eksempel kan vi forestille oss at snutten:

```
%@exec python script_example.py 4
```

gir oss utskriften:

```
\begin{verbatim}
$ python script_example.py 4
24.0
\end{verbatim}
```

Med andre ord, istedenfor å importere, skal koden kjøres og resultatet av kjøringen skal limes inn i dokumentet. Her kan det være nyttig å bruke `subprocess.Popen`-modulen. Ta en titt på oppgave 7 for et eksempel på bruk av denne modulen.

Oppgave 3: Kodeformatering

Koden over bruker `verbatim` som utgangspunkt for fremvisning av kode. Men siden dette skal automatiseres, kan vi legge inn formateringer som er mye penere. Det er ikke forventet at man skal kunne lage dette selv. Istedenfor kan man finne en mal i filen `format_template` for fin formatering av verbatim. Et eksempel på LaTeX-kode før og etter preprosessering kan du se i filene `tex_before.tex` og `tex_after.tex`.

Kode-import er som brukt gjennom dokumentet i lyseblå og ser slik ut etter kompilering av det endelige LaTeX-dokumentet:

```
from __future__ import braces
```

filen inneholder også et tilsvarende snutt for kode-eksekvering. Den vil se slik ut etter kompilering:

Terminal

```
$ echo "help I'm stuck in a small shell-script"
help I'm stuck in a small shell-script
```

Oppgave 4: Fake kode-import/-eksekvering

Istedenfor å importere eller eksekvere fra ekstern fil, kan det hende at det er mer hendig å skrive koden verbatim med ny fin formatert kode. Ved å ikke legge ved et filnavn etter `%@import` og `%@exec`forventer koden at alt etter blir sett på som fake kode frem til snutten `%@` er funnet. Her `%@` på egen linje indikerer at verbatim kode er slutt.

For eksempel:

```
%@import
Dette er et fake kode-import.
%@
```

skal være nok for å plassere setningen inne i den formaterte lyse-blå verbatim-klassen. Tilsvarende vil:

```
%@exec
$ echo "Just another Perl hacker,"
%@
```

resultere i en utskrift:

Terminal

```
$ echo "Just another Perl hacker,"
```

Merk at å velge oppgave 4, uten oppgave 3 i praksis ikke er spesielt nyttig. Men det valget er fortsatt gyldig.

Oppgave 5: Kode-eksekvering

I denne del-oppgaven skal dere lese eksekverbar Python og Bash kode fra LaTeX-dokumentet, kjøre koden og lime utskrift fra kjøringen tilbake i dokumentet. Med andre ord, dere skal gjøre det samme som i oppgave 2, men istedenfor å hente kode fra eksternt skript, skal koden hentes direkte fra LaTeX-dokumentet. For eksempel vil

```
%@python fake_name.py fake_arg
print 2+2
%@
```

gi resultatet

Terminal

```
$ python fake_name.py fake_arg
4
```

Tilsvarende vil

```
%@bash fake_name.sh fake_arg
echo "2+2" | bc
%@
```

gi resultatet

Terminal

```
$ bash fake_name.sh fake_arg
4
```

I dette eksempelet brukes `%@` både som start og slutt på blokk.

Oppgave 6: Skjult tekst

I denne oppgaven ønsker vi å kunne gjemme eller trekke frem tekst i LaTeX-dokumentet ved behov. For å definere dette må vi først introdusere variabler. Disse defineres ved hjelp av `%@var`:

```
%@var lang python
```

Her variablen `lang` får verdien python. Denne snutten er tenkt til å være plassert i begynnelsen av dokumentet slik at brukeren kan sette den derfra, men den kan i utgangspunktet plasseres hvor som helst.

Med denne variablen definert kan man bruke `show` til å inkludere kode etter ønske. For eksempel:

```
%@show lang matlab
%@verb
for i=1:10
    i
end
%@
%@end
%@show lang python
%@verb
for i in xrange(10):
    print i+1
%@
%@end
```

I dette eksempelet vises den første kodesnutten hvis `lang` er satt til `matlab`, mens i den andre vil vises hvis tilsvarende `lang` er satt til `python`.

Det skal være mulig å legge andre `%@`-komandoer inne i blokker, så `%@show` skal eksplisitt avsluttes med `%@end`.

I tillegg til `show` skal man også lage en `hide`-funksjon som gjør det motsatte og gjemmer tekst når en variabel er satt.

`%@verb` og `%@` skal her tolkes som `\begin{verbatim}` og `\end{verbatim}`.

*Oppgave 7: Kompilering

Bruk Python-modulen `subprocess.Popen` for kompilere LaTeX-dokumentet:

```
>>> import subprocess
>>> proc = subprocess.Popen(
"pdflatex -file-line-error -interaction=nonstopmode path/to/file",
shell=True, stdout=subprocess.PIPE)
>>> out, err = proc.communicate()
```

De to variablene `out` og `err` inneholder standard-utskrift og feilmeldinger fra kjøring.

Siden argumentet `-file-line-error` er inkludert inneholder utskriften alle feilmeldinger på formatet:

```
filnavn:linjenummer:feilmelding
```

Hent ut alle feilmeldingene samt de to siste linjene i loggen og skriv dem ut til skjerm istedenfor den vanlige LaTeX-utskriften.

For de som er kjent med LaTeXog bryr som som slikt: Hvis man ønsker å bruke sin egen variant av LaTeX, som `latex`, `xetex` eller `luatex` er dette også lov, men da må man være tydlig i rapporten på hva som gjør hva. Det skal være enkelt for en utenforstående å se hva man har gjort.

*Oppgave 8: Filnavn

Istedenfor å redigere på brukerens egen LaTeX-fil direkte er det kanskje hensiktsmessig å lage en kopi av LaTeX-filen som man redigerer på. All redigeringen gjøres da i kopien. Alternativt gjør alle enringene internt i programmet og lagres til en annen fil. Implementer denne funksjonaliteten.

I tillegg, LaTeXhar funksjonenen `\input` som lar deg importere andre LaTeXinnhold fra andre filer. Den vil ha formen:

```
\include{path/to/file}
```

For at underfilene skal kunne proseseres ordentlig, må man først importere innholdet i alle filene referert til med `\include` før andre prosesser blir gjort.

Oppgave 9: Filtre

Istedenfor å samle alt i en fil, kan det være hensiktsmessig å å beholde filene separate. Lag en mappe (eller mappetre) og kopier alle refererte filer inn i denne mappen. Fiks dokumentet ditt slik at filene blir referert til riktig. Pas på at like navn på filer som er i forskjellige mapper ikke er et problem. Husk å preprosessere alle LaTeX-dokumentene i mappen før man kompilerer.

Oppgave 10: Linjenummerering

Linjenummereringen referert til i oppgave 7 tar utgangspunkt i det manipulerte dokumentet. Fra brukerens pespektiv er ikke dette veldig hensiktsmessig, siden nummeret ikke samsvarer med dokumentet brukeren redigerer. Så i denne oppgaven skal man fikse dette problemet ved å erstatte linjenummerene i utskriften, men tilsvarende i det orginale dokumentet.

Hint: Før du redigerer på det nye dokumentet kan det være hensiktsmessig å legge til en kommentar på slutten av hver linje:

```
\documentclass{article}%1:/path/to/file
%2:/path/to/file
\usepackage[T1]{fontenc}%3:/path/to/file
\usepackage[utf8]{inputenc}%4:/path/to/file
%5:/path/to/file
```

Disse lineje kan dermed bli funnet igjen etter preprosesering. Evt. kan man legge inn filnavn

Merk at hvis man har laget et filtre, må man også passe på at navnene refererer til riktig fil.

*Oppgave 11: Frontend

Fra et brukerperspektiv kan det være greit å ha et vellfungerende Bash-brukergrensesnitt. Implementer dette ved hjelp av `argparse`-modulen. Følgende funksjoner forventes å være inkludert:

- Mulighet for velge navn på preprosessert fil. (mappe)
- Verbose-mode skriver ut på skjerm alle operasjoner som blir gjort.
- Mulighet for å bytte mellom enkel og fancy verbatim utskrift.
- Skru av og på `-interaction=nonstopmode` i pdflatex (eller tilsvarende).

*Oppgave 12: Testing og dokumentasjon

Alle interne funksjoner skal dokumenteres med doc-testes. I tillegg skal man implementere en test suit som tester de følgende kriteriene:

- Kodeimport
- Eksekvering av skript

- Kode-eksekvering
- Skjult tekst
- Kompilering
- Linjenummerering

Testing av kode som ikke er implementert er selvsagt untatt.

*Oppgave 13: Rapport

En rapport av hva du har gjort skal inn i en LaTeX-rapport. Denne rapporten skal inkludere relevante kodesnutter og kjøringer som både viser til og bruker i praksis programmet du nettopp har laget. Det er for eksempel lov til å lage sikkelig dokumentasjon i et program, og bruke `%@import` til å inkludere den i rapporten.