Assignment 2

By Silje Anfindsen

Task 2: Beskrivelse av en micro desktop calculator

Denne består av tre deler:

1) Leximizer
   = konverterer én string av lexemes til flere enkelt-strenger av lexemes satt inn i en liste
   Eks på lexemes: lexemes er: '2', '4', '6', '-', 'f'

2) Tokenizer
   = konverterer en string av separate lexemes til tokens. Det vil si, den skiller mellom type lexemes. I dette tilfellet operator, tall og kommando.
   Eks på tokens: numeral('2'), numeral('4'), numeral('6'), operator('-'), command('f')

3) Interpreter
   = utfører stakkoperasjoner på en liste av tokens helt til det ikke er flere operatorerer igjen. Den traverserer gjennom tokens, hvis det er et tall pushes dette til en stakk, dersom det er en operator fjernes de to siste tallene i stakken, operasjonen utføres på disse (postfix) og resultatet pushes til stakken igjen. Funksjonen går rekursivt gjennom alle tokens til den er tom.

En mdc krever postfix uttrykk, dvs. operator kommer etter to operander/tall.
Gyldig input: 1 2 -

Task 3: Postfix til infix notasjon

Konverterer postfix notasjonen fra task 2 til infix-notasjon. Returnerer en expression stakk som er en string med infix-notasjon. Input er tokens i postfix notasjon.
Tar inn en input-stakk med tokens der følgende vurderes basert på type tokens:

- Hvis det er en ikke-operator, altså et tall i dette tilfellet, så legges dette til i en expression stakk.
- Hvis det er en operator så hentes ut de to siste tallene fra expression stakken, stringen som representerer uttrykket i infix notasjon legges så til i expression stakken på formen a+b.

Går så rekursivt gjennom input-stakken til den er tom.

Task 4: Teori

a) Formally describe the regular grammar of the lexemes in task 2.

V = {1,2,3,4,5,6,7,8,9....} (variabler)
S = {+,-,*,/} (symboler)
R = {v1-v2, v1+v2, v1*v2, v1/v2 | v1 >= v2} (regler)

b) Describe the grammar of the infix notation in task 3 using (E)BNF.
Beware of operator precedence. Is the grammar ambiguous? Explain why it is or is not ambiguous?
<expression>::= <integer> | <expression> <operator> <expression>
<operator> ::= - | + | * | /

Ambigious grammar kan vises hvis en sentning fra definisjonen over kan gi to eller flere parse trees. Fra forelesning var en løsning på amigiousness å bruke paranteser rundt kompleks notasjon. Da får vi unambigous grammar. Dette er tilfellet her.

c) What is the difference between a context-sensitive and a context-free grammar?
Hovedforskjellen ligger i hvordan reglene for grammar virker. En grammar som inneholder non-terminals som er avhengig av context der den blir brukt kalles context-sensitive. Det kan f.eks. være at en variabel må deklareres før den kan kalles på. Context-free grammars vil virke slik at expansion av non-terminals er uavhengig av hvor det brukes. Dette gjelder for EBNF-notation.

d) You may have gotten float-int errors in task 2. If you haven't, try running 1+1.0. Why does this happen? Why is this a useful error?
I oz kan man ikke utføre en operator på en int og en float sammen. Da får man type error paa formen 'Expected Type: Integer'. Det er veldig fort gjort å feks sette 1+0.1 når man jobber med ulike datatyper, spesielt i oz hvor man ikke trenger å spesifisere hvilke datatyper man deklarerer (dynamisk språk). Det er derfor til god nytte å få en slik feilmelding.