# Ukeoppgaver i INF3110/4110

Uke 44 (29.-31.10.2003)

### Her er en grammatikk for eksempelspråket L

⟨*program*⟩ → ⟨*setningsListe*⟩
⟨*setningsListe*⟩ → ⟨*setning*⟩$^+$
⟨*setning*⟩ → ⟨*tilordning*⟩ | ⟨*inputSetning*⟩ | ⟨*outputSetning*⟩
⟨*tilordning*⟩ → ⟨*variabel*⟩ **=** ⟨*variabel*⟩ ⟨*operator*⟩ ⟨*operand*⟩
⟨*operator*⟩ → **+** | **-**
⟨*operand*⟩ → ⟨*variabel*⟩ | ⟨*tall*⟩
⟨*inputSetning*⟩ → **?** ⟨*variabel*⟩
⟨*outputSetning*⟩ → **!** ⟨*variabel*⟩
⟨*variabel*⟩ → **v** ⟨*siffer*⟩
⟨*siffer*⟩ → **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
⟨*tall*⟩ → ⟨*siffer*⟩$^+$

## Oppgave 1

Bli kjent med språket.

Skriv L-programmer som

- leser inn to tall fra brukeren og skriver ut summen
- skriver ut et tall $n$
- leser inn et tall $n$ og returnerer tallet $10 \times n$

## Oppgave 2

Du skal nå skrive ferdig en parser for L. Her er litt repetisjon fra forelesningene:

### En enkel *recursive descent*-parser i ML

Idé: La hvert terminalsymbol være en konstruktør av type `token`:

```
datatype token = Pluss | Gange | Navn;
exception Parserfeil of string; (* for å gi feilmeldinger *)
```

Vi forutsetter at en skanner gjør om strenger til lister av tokens. (Skanneren er i dette tilfellet en funksjon som tar "**navn * navn + navn**" som argument og som gir listen `[Navn,Gange,Navn,Pluss,Navn]` av tokens.)
For hvert metasymbol ønsker vi nå en funksjon fra `token list` til `token list`. Vi lar denne funksjonen ha samme navn som metasymbolet.

## En enkel *recursive descent*-parser i ML

⟨*uttrykk*⟩ → ⟨*term*⟩ ⟨*xterm*⟩

```
fun uttrykk tokens =
    let val tokens1 = term tokens
        val tokens2 = xterm tokens1
    in tokens2 end
```

⟨*xterm*⟩ → **+** ⟨*term*⟩ ⟨*xterm*⟩ | *ε*

```
and xterm tokens =
    case tokens of Pluss :: resttokens =>
                   (let val tokens1 = term resttokens
                        val tokens2 = xterm tokens1
                    in tokens2 end)
                 | _ => tokens
```

## En enkel *recursive descent*-parser i ML

⟨*term*⟩ → **navn** ⟨*xnavn*⟩

```
and term tokens =
    case tokens of Navn :: resttokens => xnavn resttokens
    | _ => raise Parserfeil("<term> begynner ikke med 'navn'")
```

⟨*xnavn*⟩ → ∗ **navn** ⟨*xnavn*⟩ | *ε*

```
and xnavn tokens =
    case tokens of Gange :: Navn :: resttokens => xnavn resttokens
                 | _ => tokens;
```

Og til slutt:

```
fun parse tokens =
    let val resultat = uttrykk tokens
    in if resultat = [] then resultat
       else raise Parserfeil("Noe igjen etter godkjent streng") end;
```

I filen `Lscanner.sml` finner du en ferdig scanner og en nesten ferdig parser for L. Legg merke til at det hvert metasymbol er en tilsvarede funksjon som tar en liste av tokens som argumenter.

Hjelpefunksjonene og skanneren trenger du ikke å skjønne fullstendig. Funksjonen `scanFil` tar et filnavn som argument og returnerer en liste av tokens som parseren kan overta.

- skriv ferdig funksjonen `setning`
- skriv ferdig funksjonen `inputSetning`

Sjekk at parseren virker ved å kjøre den på noen eksempler, f.eks. de tre programmene fra første oppgave.

## Oppgave 3

Vi endrer nå litt på grammatikken for ⟨*tilordning*⟩:

⟨*tilordning*⟩ → ⟨*variabel*⟩ = ⟨*operand*⟩ ⟨*operator*⟩ ⟨*operand*⟩ | ⟨*variabel*⟩ = ⟨*tall*⟩

- Skriv om parserern slik at denne endringene ivaretas.

Sjekk at alt fungerer ved å kjøre parseren på enkle eksempler. F.eks. skal nå følgende godtas som ikke ble godtatt tidligere:

```
- parse(scan "v1=123");
Parsing vellykket!

- parse(scan "v1=123 + v9");
Parsing vellykket!
```

## Oppgave 4

Her skal du skrive en enkel interpret for L ved å gjøre noen små endringer på parseren. Anta at følgende funksjoner er gitt:

- `setLager(`$i$`,`$x$`)` som legger tallet $x$ inn i lagerplass nummer $i$
- `getLager(`$i$`)` som tar et tall $i$ som argument og som gir det som ligger i lagerplass nummer $i$
- `lesTilLager(`$i$`)` som ber brukeren om en verdi for `v`$i$ og som legger det på lagerplass $i$
- `lesFraLager(`$i$`)` som skriver ut verdien i lagerplass $i$

Ta utgangspunkt i filen `Linterpret.sml`. Det eneste du trenger å fylle inn er funksjonene `tilordning`, `operator` og `operand`. Funksjonene `inputSetning` og `outputSetning` er allerede skrevet om. (Bortsett fra disse er alt helt identisk med parseren.)