Universitetet i Oslo
Institutt for Informatikk

Reliable Systems
Martin Steffen, Gianluca Turin

INF 5110: Compiler construction

Spring 2021 — **Series 2** — 17. 1. 2021

Topic: Context free grammars

Issued: 17. 1. 2021

This exercise set covers more than one lecture. It's about grammars, and partly for the lectures about *parsing*. We might not be able to cover it within 2 hours.

Exercise 1 (First- and follow sets) Compute the *First* and *Follow*-sets for the grammar Figure 1.

$$
\begin{array}{rcl}
\mathit{exp} & \to & \mathit{term}\ \mathit{exp}' \\
\mathit{exp}' & \to & \mathit{addop}\ \mathit{term}\ \mathit{exp}' \mid \boldsymbol{\epsilon} \\
\mathit{addop} & \to & + \mid - \\
\mathit{term} & \to & \mathit{factor}\ \mathit{term}' \\
\mathit{term}' & \to & \mathit{mulop}\ \mathit{factor}\ \mathit{term}' \mid \boldsymbol{\epsilon} \\
\mathit{mulop} & \to & * \\
\mathit{factor} & \to & (\ \mathit{exp}\ ) \mid \mathbf{n}
\end{array}
$$

Figure 1: Expression grammar (left-recursion removed)

Exercise 2 (Nullable) Describe an algorithm that finds all nullable non-terminals without first finding the first-sets.

Exercise 3 (Associativity and precedence) Take the binary ops $+$, $-$, $*$, $/$ and $\uparrow$. Let's agree also on the following precedences and associativity

op	precedence	associativity
$+, -$	low	left assoc.
$*, /$	higher	left. assoc.
$\uparrow$	highest	right. assoc

Write an *unambiguous* grammar that captures the given precedences and associativies (of course, directly with a BNF grammar, without allowing yourself specifying those requirements as extra side-conditions).

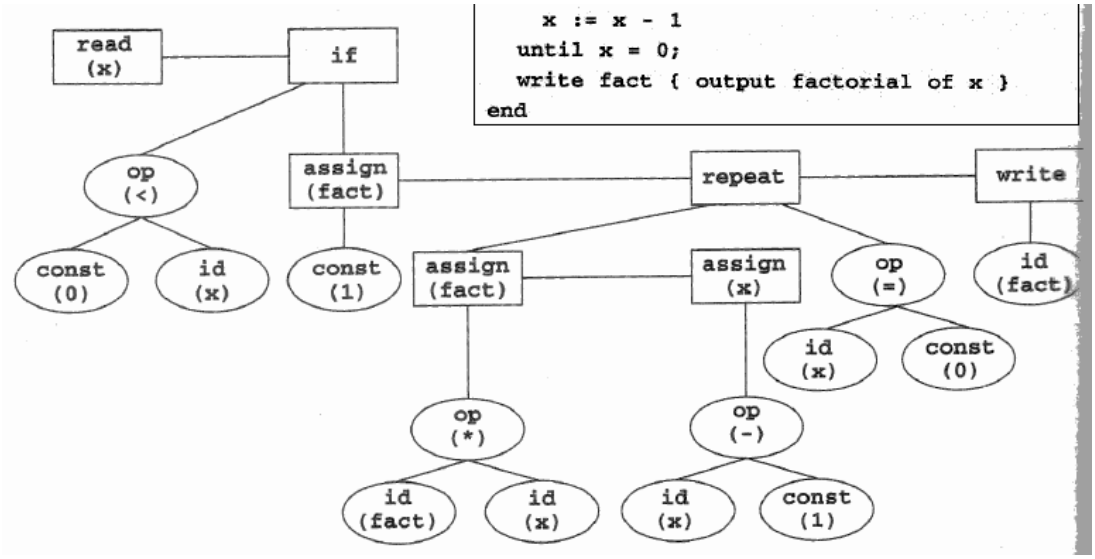
Exercise 4 (Tiny grammar) For the grammar given answer the following questions:

- Is the grammar *unambiguious?*

- How can we change the grammar, so that TINY allows empty statements?
- How can we arrange it that semicolons are required *in between* statements, not *after* statements?
- What's the precedence and associativity of the different operators?

$$
\begin{array}{rcl}
\textit{program} & \rightarrow & \textit{stmt-seq} \\
\textit{stmt-seq} & \rightarrow & \textit{stmt-seq}\ \textbf{;}\ \textit{stmt} \mid \textit{stmt} \\
\textit{stmt} & \rightarrow & \textit{if-stmt} \mid \textit{repeat-stmt} \mid \textit{assign-stmt} \\
 & \mid & \textit{read-stmt} \mid \textit{write-stmt} \\
\textit{if-stmt} & \rightarrow & \textbf{if}\ \textit{expr}\ \textbf{then}\ \textit{stmt}\ \textbf{end} \\
 & \mid & \textbf{if}\ \textit{expr}\ \textbf{then}\ \textit{stmt}\ \textbf{else}\ \textit{stmt}\ \textbf{end} \\
\textit{repeat-stmt} & \rightarrow & \textbf{repeat}\ \textit{stmt-seq}\ \textbf{until}\ \textit{expr} \\
\textit{assign-stmt} & \rightarrow & \textbf{identifier}\ \textbf{:=}\ \textit{expr} \\
\textit{read-stmt} & \rightarrow & \textbf{read identifier} \\
\textit{write-stmt} & \rightarrow & \textbf{write}\ \ \textit{expr} \\
\textit{expr} & \rightarrow & \textit{simple-expr}\ \textit{comparison-op}\ \textit{simple-expr} \mid \textit{simple-expr} \\
\textit{comparison-op} & \rightarrow & < \mid = \\
\textit{simple-expr} & \rightarrow & \textit{simple-expr}\ \textit{addop}\ \textit{term} \mid \textit{term} \\
\textit{addop} & \rightarrow & + \mid - \\
\textit{term} & \rightarrow & \textit{term}\ \textit{mulop}\ \textit{factor} \mid \textit{factor} \\
\textit{mulop} & \rightarrow & * \mid / \\
\textit{factor} & \rightarrow & (\ \textit{expr}\ ) \mid \textbf{number} \mid \textbf{identifier}
\end{array}
$$

Exercise 5 (AST) The book [1] give some illustration and proposal for an AST data structure for TINY:



The tree representation corresponds to the following piece of source code.

Listing 1: Sample TINY program

```
read x; { input as integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x -1
  until x = 0;
  write fact   { output factorial of x }
end
```

Design an appropriate AST data structure, using object-oriented structuring. In particular, make use if an appropriately define class *hierarchy* (i.e., use inheritance). This should give a "better-structured" AST data structure compared to [1], where all the nodes of the AST tree are ultimately just "nodes".

References

[1] K. Louden. *Compiler Construction, Principles and Practice.* PWS Publishing, 1997.