Compiler Construction

Lab 1

Sept 7, FSU/CS

Your language is defined as follows

$$
\begin{aligned}
\mathit{program} &\rightarrow (\texttt{define-fun}\ (\mathit{fun}\ (\mathit{type}\ \mathit{var})^*)\ \mathit{type}\ \mathit{expr})\ \mathit{program} \mid (\texttt{eval}\ \mathit{expr}) \\
\mathit{type} &\rightarrow \texttt{int} \mid \texttt{bool} \\
\mathit{expr} &\rightarrow \mathit{term} \mid \mathit{fla} \\
\mathit{term} &\rightarrow \mathit{const} \mid \mathit{var} \mid (\texttt{get-int}) \mid (\texttt{+}\ \mathit{term}\ \mathit{term}^+) \mid (\texttt{*}\ \mathit{term}\ \mathit{term}^+) \mid (\texttt{-}\ \mathit{term}\ \mathit{term}) \mid \\
&\quad\ (\texttt{div}\ \mathit{term}\ \mathit{term}) \mid (\texttt{mod}\ \mathit{term}\ \mathit{term}) \mid \\
&\quad\ (\texttt{if}\ \mathit{fla}\ \mathit{term}\ \mathit{term}) \mid (\mathit{fun}\ \mathit{expr}^*) \mid (\texttt{let}\ (\mathit{var}\ \mathit{expr})\ \mathit{term}) \\
\mathit{fla} &\rightarrow \texttt{true} \mid \texttt{false} \mid \mathit{var} \mid (\texttt{get-bool}) \mid \\
&\quad\ (\texttt{=}\ \mathit{term}\ \mathit{term}) \mid (\texttt{<}\ \mathit{term}\ \mathit{term}) \mid (\texttt{<=}\ \mathit{term}\ \mathit{term}) \mid (\texttt{>}\ \mathit{term}\ \mathit{term}) \mid (\texttt{>=}\ \mathit{term}\ \mathit{term}) \mid \\
&\quad\ (\texttt{not}\ \mathit{fla}) \mid (\texttt{and}\ \mathit{fla}\ \mathit{fla}^+) \mid (\texttt{or}\ \mathit{fla}\ \mathit{fla}^+) \mid (\texttt{if}\ \mathit{fla}\ \mathit{fla}\ \mathit{fla}) \mid \\
&\quad\ (\mathit{fun}\ \mathit{expr}^*) \mid (\texttt{let}\ (\mathit{var}\ \mathit{expr})\ \mathit{fla})
\end{aligned}
$$

In particular

- All monospace strings are the reserved words
- Comments start with ; and continue to the end of the line
- Functions can take any number of arguments
- `+, -, *, and, or` can take any number of arguments, but at least two
- Variable/function names can have letters or numbers but cannot start with a number
- The language is case-sensitive
- There are no rational numbers
- Hyphen/dash is allowed only in a few keywords and to represent the subtraction operator
- If you don’t understand some semantics, that is fine for now. It will be given in the next labs

Your task

- Write a scanner using `lex` to split the input file to tokens
- Write a parser using `yacc` that has ALL productions of your grammar. It should report syntax errors
  - But printing custom messages is optional
- Augment the `yacc` file with semantic actions to print the programs in infix notation:
  - Operators should be *between* operands
  - Parentheses should be *after* functions and `if`/`let`-instructions
  - First argument of `let` should have ”`=`“ between sub-arguments
  - Function arguments (and arguments of `if`/`let`-instructions) shoud be separated by commas
  - Function return type should be before function name
  - Function body should be separated from the declaration by “`:`”
  - `define-fun` should not be printed
  - Extra parentheses can be kept (but try to remove them)

Examples

Example 1

Input:
```
(eval 7)
```
Output
```
eval (7)
```

Example 2

Input:
```
(eval (+ var1 (* var2 5)))
```
Output
```
eval (var1 + (var2 * 5))
```

Example 3

Input:
```
(eval (let (a 1) (* a (+ a a))))
```
Output
```
eval (let (a = 1, a * (a + a)))
```

Example 4

Input:
```
(define-fun (foo (bool a) (bool b)) bool (or a b))
(eval (foo (< 1 5) (< 6 7)))
```

Output
```
bool foo (bool a, bool b) : (a or b)
eval (foo (1 < 5, 6 < 7))
```

Important

- Your code should be committed to your GitHub repository
- Invite `grigoryfedyukovich` as collaborator
- Commit your test cases (i.e., particular programs in the lab-language)
- Try to commit one feature/bugfix at a time and write meaningful commit messages
- Any general-purpose questions about the language/lab should be directed to the `#general` channel of Slack workspace
- If successful, you will get 10 points for this lab