Compilers Coursework 1 Feedback

A - Abstract syntax tree definition for `if`-command

The extended `if`-command can be written using a single constructor by utilising standard Haskell types. The `elsif` branches can be represented using a list type and the optional `else` branch with a *Maybe* type.

```
data Command
   = ..
   | CmdIf {
       ciCond     :: Expression,
       ciThen     :: Command,
       ciElsifs   :: [(Expression, Command)],
       ciMbElse   :: Maybe Command,
       cmdSrcPos :: SrcPos
       }
```

A2 - Productions for `if`-command

The `if`-command can be written as a single alternative in the command production, along with two additional productions for the `elsif` branches and the `else` branch.

```
command :: {Command}
command
   : ..
   | IF expression THEN command elsifs optelse
     {CmdIf {ciCond = $2, ciThen = $4, ciElseifs = $5,
                                 ciMbElse = $6, cmdSrcPos = $1}}
optelse :: {Maybe Command}
optelse :  {-epsilon -}
             {Nothing}
        |  ELSE command
             {Just $ 2}
elsifs  :: {[(Expression, Command)]}
elsifs  :  {-epsilon -}
             {[]}
        |  ELSIF expression THEN command elsifs
             {($2, $4) : $5}
```

B - Illegal character tokens in Character Literal

The extension for character literals should only accept the "non-control" characters as graphics but excluding $'$ and \\ . The non-control characters are given as the ASCII range from 32(space) to 126(tilde). The following function determines if a character is a graphic:

$$
\begin{aligned}
&\mathit{graphic} :: \mathit{Char} \rightarrow \mathit{Bool}\\
&\mathit{graphic}\ x = x \geqslant \texttt{' '}\\
&\qquad\qquad\quad \wedge\ x \leqslant \texttt{'\textasciitilde'}\\
&\qquad\qquad\quad \wedge\ x \not\equiv \texttt{'\textbackslash''}\\
&\qquad\qquad\quad \wedge\ x \not\equiv \texttt{'\textbackslash\textbackslash'}
\end{aligned}
$$

C - Bad error messages in Scanner

It is good practice to make sure Haskell definitions have complete definitions. By doing so we avoid unhelpful error messages such as the following:

```
*** Exception:  Scanner.hs:202:1-19:  Non-exhaustive patterns in
     function scan
```

Instead *emitErrD* should have been used for lexical errors to give a helpful error. Likewise care should be taken using partial functions as they also generate unhelpful error messages, for example you should only apply *tail* to a list that will always be non-empty.

D - Missing precedence in conditional expression

The conditional expression should be right associative and have the lowest precedence. Therefore `Parser.y` should contain the following declaration:

```
% right '?' ':'
% left '||'
..
```

It should come before other associativity declarations as it has the lowest precedence. An example of a program that is parsed incorrectly without the declaration is `e0 || e1 ? e2:e3`. It should be parsed to `(e0 || e1) ? e2:e3` but will instead be interpreted as `e0 || (e1?e2:e3)`.