Теорія мов програмування

Лабораторна робота 6. Компіляція Standard ML

6.1 Враховуючи специфікацію ML-lex на наступних рисунках:

```
(* mlcomp.lex -- lexer spec *)
type pos = int
type svalue = Tokens.svalue
type ('a, 'b) token = ('a, 'b) Tokens.token
type lexresult = (svalue, pos) token
val pos = ref 1
val error = fn x => TextIO.output(TextIO.stdErr, x ^ "\n")
val eof = fn () => Tokens.EOF(!pos, !pos)
fun countnewlines s =
    let val lst = explode s
        fun count (c:char) nil = 0
          | count c (h::t) =
            let val tcount = count c t
            in
              if c = h then 1+tcount else tcount
            end
    in
      pos:= (!pos) + (count #"\n" lst)
    end
%%
%header (functor mlcompLexFun(structure Tokens : mlcomp_TOKENS));
alpha=[A-Za-z];
alphanumeric=[A-Za-z0-9_\.];
digit=[0-9];
ws=[\ \t];
dquote=[\"];
squote=[\'];
anycharbutquote=[^"];
anychar=[.];
pound=[\#];
tilde=[\~];
period=[\.];
```

```
%%
\(\*([^*]|[\r\n]|(\*+([^*\)]|[\r\n])))*\*+\) => (countnewlines yytext; lex());
\n  => (pos := (!pos) + 1; lex());
{ws}+  => (lex());
"+"  => (Tokens.Plus(!pos,!pos));
"*"  => (Tokens.Times(!pos,!pos));
"-"  => (Tokens.Minus(!pos,!pos));
"@"  => (Tokens.Append(!pos,!pos));
"=" => (Tokens.Equals(!pos,!pos));
"("  => (Tokens.LParen(!pos,!pos));
")"  => (Tokens.RParen(!pos,!pos));
"[" => (Tokens.LBracket(!pos,!pos));
"]" => (Tokens.RBracket(!pos,!pos));
"::" => (Tokens.ListCons(!pos,!pos));
"," => (Tokens.Comma(!pos,!pos));
";" => (Tokens.Semicolon(!pos,!pos));
"_" => (Tokens.Underscore(!pos,!pos));
"=>" => (Tokens.Arrow(!pos,!pos));
"|" => (Tokens.VerticalBar(!pos,!pos));
">" => (Tokens.Greater(!pos,!pos));
(* a few token are omitted here *)
{tilde}?{digit}+  => (Tokens.Int(yytext,!pos,!pos));
{pound}{dquote}{anychar}{dquote} => (Tokens.Char(yytext,!pos,!pos));
{dquote}{anycharbutquote}*{dquote} => (Tokens.String(yytext,!pos,!pos));
{alpha}{alphanumeric}*=>
   (let val tok = String.implode (List.map (Char.toLower)
              (String.explode yytext))
    in
      if      tok="let" then Tokens.Let(!pos,!pos)
      else if tok="val" then Tokens.Val(!pos,!pos)
      else if tok="in" then Tokens.In(!pos,!pos)
      else if tok="end" then Tokens.End(!pos,!pos)
      else if tok="if" then Tokens.If(!pos,!pos)
      else if tok="then" then Tokens.Then(!pos,!pos)
      else if tok="else" then Tokens.Else(!pos,!pos)
      else if tok="div" then Tokens.Div(!pos,!pos)
      else if tok="mod" then Tokens.Mod(!pos,!pos)
      else if tok="fn" then Tokens.Fn(!pos,!pos)
      else if tok="while" then Tokens.While(!pos,!pos)
      else if tok="do" then Tokens.Do(!pos,!pos)
      else if tok="and" then Tokens.And(!pos,!pos)
      else if tok="rec" then Tokens.Rec(!pos,!pos)
      else if tok="fun" then Tokens.Fun(!pos,!pos)
      else if tok="as" then Tokens.As(!pos,!pos)
      else if tok="handle" then Tokens.Handle(!pos,!pos)
      else if tok="raise" then Tokens.Raise(!pos,!pos)
      else if tok="true" then Tokens.True(!pos,!pos)
      else if tok="false" then Tokens.False(!pos,!pos)
      else Tokens.Id(yytext,!pos,!pos)
    end);
.  => (error ("error: bad token "^yytext); lex())
```

що ще потрібно було б додати, щоб дозволити правильно виразити подібні вирази сканером? Які нові маркери потрібно було б розпізнати? Як би ви змінили специфікацію, щоб прийняти ці маркери?

6.2 Як би ви модифікували абстрактний синтаксис, щоб подібні вирази могли бути представлені?

```
case x of
    1 => "hello"
  | 2 => "how"
  | 3 => "are"
  | 4 => "you"
```

6.3 Які модифікації потрібні в специфікації mlcomp.grm для аналізу подібних виразів?

```
case x of
    1 => "hello"
  | 2 => "how"
  | 3 => "are"
  | 4 => "you"
```

Контрольні запитання

1. Мова регулярних виразів може використовуватися для визначення лексем мови. Наведіть приклад регулярного виразу з глави та вкажіть, який вид лексем він представляє.
2. Що робить **ML-lex**? Який вхід потрібен? Що це виробляє?
3. Чому ключові слова повинні розпізнаватися за допомогою оператора **if-else-if** у визначенні **ML-lex**? Чому кожне ключове слово не можна було просто розпізнати, як інші фіксовані маркери в мові?
4. Як оголошується абстрактне дерево синтаксису в ML?
5. Використовуючи специфікацію граматики для Small, що таке AST наступного виразу?

```
fun abs(x) = if x > 0 then x else ~1*x
```

6. Як функція навантаження генератора коду вирішує, яку команду навантаження генерувати?
7. При створенні коду для викликів функцій на рис. 13.15 (див. лекцію 13), яка мета двох рекурсивних викликів кодегену?
8. Яка функція генератора коду відповідає за повернення нових прив'язок, створених виразом **let**?
9. Що означає для мови Small підтримку логіки короткого замикання? Що відбувається при генерації коду?
10. На рис. 13.33 (див. лекцію 13), до чого відноситься **nameList** та налаштовані прив'язки для програми, поданої на рис. 13.32 (див. лекцію 13)? Наведіть фактичний зміст трьох списків? Чому три списки?

Виконані завдання та відповіді на контрольні запитання надішліть на пошту викладачеві.