CS421 Assignment 3 Tiger Parser Chen Gu CG736

Definition

Terminal Definition

First of all, We need to define our terminals.

```
%term
    EOF
  | ID of string
  | INT of int | STRING of string
  | COMMA | COLON | SEMICOLON | LPAREN | RPAREN | LBRACK | RBRACK
  | LBRACE | RBRACE | DOT
  | PLUS | MINUS | TIMES | DIVIDE | EQ | NEQ | LT | LE | GT | GE | UMINUS
  | AND | OR | ASSIGN
  | ARRAY | IF | THEN | ELSE | WHILE | FOR | TO | DO | LET | IN | END | OF

  | BREAK | NIL
  | FUNCTION | VAR | TYPE
```

Non Terminal Definition

Then we define our non-terminals.

```
%nonterm

     program
   | record_exp | record_aux
   | func_exp | func_aux
   | lvalue | lvalue_aux
   | expseq | expseq_aux
   | decs | dec | fundec | tydec| tyfields | tyfields_aux | vardec | ty
   | let_exp | assign_exp | array_exp | loop_exp | if_exp | ari_exp | bool
_exp | cmp_exp | uni_exp | exp
```

Precedence Definintion

Here we define the precedence relationship between terminals to avoid shift-reduce conflicts.

```
%nonassoc DO
%nonassoc THEN
%nonassoc ELSE
%nonassoc ASSIGN
%nonassoc OF
%left OR
%left AND
%nonassoc EQ NEQ LT GT LE GE
%nonassoc LBRACK
%left PLUS MINUS
%left TIMES DIVIDE
%left UMINUS
```

Problems Encountered during Implementation

After I wrote down the rules based on the appendix, I encounter multiple shift-reduce errors. In this section, I will describe how I solve these errors.

Operators and For (While)

The detailed conflicts is as follow (This conflicts occurred in multiple states and I select one of them):

```
error: state 89: shift/reduce conflict (shift OR, reduce by rule 36)
...
state 89:
    loop_exp: WHILE exp DO exp . (reduce by rule 36)
    ...
    ari_exp : exp OR exp
```

Here we can see that the conflict is generated because the program has ambiguity about whether to reduce while expression or shift the operators. I resolve this problem by giving

FOR (WHILE) loops a precedence, which is lower than operators.
Since the precedence of `DO` terminal represents the precedence of the whole expression, we can simply define the precedence of `Do` to be lower than normal operators.

```
%nonassoc DO
...
%left OR
%left AND
%nonassoc EQ NEQ LT GT LE GE
%nonassoc LBRACK
%left PLUS MINUS
%left TIMES DIVIDE
%left UMINUS
```

By doing this, the parser will shift those operators when in such situations.

Operators and Array Creation

The next conflicts I solve is as follow:

```
error:  state 126: shift/reduce conflict (shift PLUS, reduce by rule 25)
...
state 126:
    ari_exp : exp . PLUS exp
    ...
    exp : ID LBRACK exp RBRACK OF exp .  (reduce by rule 25)
```

Operators should have a high priority than array create statement (OF), we define the precedence of OF to be lower than operators.

```
%nonassoc OF
...
%left OR
%left AND
%nonassoc EQ NEQ LT GT LE GE
%nonassoc LBRACK
%left PLUS MINUS
%left TIMES DIVIDE
%left UMINUS
```

Handling Unary Minus

It is a common sense that unary minus should have a higher priority than regular expressions and all kinds of operations. Here we introduce a new TERMINAL `UMINUS`, which has the highest priority by putting `%left UMINUS` in the last. And we use the precedence of `UMINUS` to represent the priority of negative integer statement.

```
ari_exp : MINUS exp %prec UMINUS  ()
```

Handling if-then-else

In most mainstream programming languages, `else` statement is always grouped with previous nearest `if then` statement. There is one simple fix for this problem. We can make `else` prior to `if then` statement so that whenever the parser sees a `else` token, it will not reduce the `if then` statement, but rather shift the `else` token in the stack.

```
%nonassoc THEN
%nonassoc ELSE
```

Handling LBRACK conflicts

The final and the most difficult conflicts is how to handle LBRACK conflicts. One way is to define the precedence of LBRACK. But this approach is not very straightforward. I decided to rewrite the rules to eliminate left recursion to solve this conflicts.
Below is the rules I wrote to solve this problem.

```
lvalue  : ID lvalue_aux ()

lvalue_aux :                          ()
    | DOT ID lvalue_aux               ()
    | LBRACK exp RBRACK lvalue_aux    ()
```

Test

Here I select two representative test case to verify the correctness of this parser.

Test Case 1

The *test11.tig* is the compilation of previous 10 test cases, so it can viewed as overall tests.

test11.tig

```
(  /* cascading them all together... */
/* test01.tig -- syntax ok */
let
type t
= {hd :


int, tl: t}

type
v
=
{}   /* empty record ok */

in
3 + - 5  /* unary minus */
end;
/* test02.tig -- syntax ok */
/* empty let expressions, nested */
let
in
  let in end;
  let in let in end end;
  let in let in let in end end end
end;
/* test03.tig -- syntax ok -- nested Lvalues */
a[1].
b[2].
c.
d.
e[3].
f.
g[7]
[8]
[9]
:=
42
;
/* test04.tig -- error -- unclosed let */
```

```
let in
let in
end;
/* test05.tig -- error line 4 */
(
x := 3 + - 5;
y := (3 +) - 5
);

/* test06.tig -- colons as terminators instead of separators */
/* this is correct..? */
(42; -6;);
/* test07.tig */
/* this is ok? */

function_call (with, extra, comma,);

/* test08.tig -- non-associative operators */
a = b < c;
/* test09.tig -- non-Lvalue used on left side of := */
(
x := 42;
f() := 42
);
/* A program to solve the 8-queens problem */

let
    var N := 8

    type intArray = array of int

    var row := intArray [ N ] of 0
    var col := intArray [ N ] of 0
    var diag1 := intArray [N+N-1] of 0
    var diag2 := intArray [N+N-1] of 0

    function printboard() =
       (for i := 0 to N-1
     do (for j := 0 to N-1
          do print(if col[i]=j then " O" else " .");
        print("\n"));
        print("\n"))

    function try(c:int) =
  (far i:= 0 to c do print("."); print("\n"); flush();  /* err */
     if c=N
     then printboard()
     else for r := 0 to N-1
       do if row[r]=0 & diag1[r+c]=0 & diag2[r+7-c]=0
```

```
                then (row[r]:=1; diag1[r+c]:=1; diag2[r+7-c]:=1;
                  col[c]:=r;
                      try(c+1);
              row[r]:=0; diag1[r+c]:=0; diag2[r+7-c]:=0)
  )

 in try(0)
end
)
```

Output

```
= tests/test11.tig:46.10:syntax error: inserting  LPAREN
tests/test11.tig:51.9:syntax error: inserting  LPAREN
tests/test11.tig:55.34:syntax error: replacing  COMMA with  RPAREN
tests/test11.tig:58.5:syntax error: inserting  LPAREN
tests/test11.tig:62.5:syntax error: replacing  ASSIGN with  SEMICOLON
tests/test11.tig:84.4:syntax error: replacing  ID with  FOR
:1.1679:syntax error found at EOF

uncaught exception Error
  raised at: parsetest.sml:18.47-18.61
```

Test Case 2

test12.tig

```
/* merge.tig -- syntax ok */
let

 type any = {any : int}
 var buffer := getchar()

function readint(any: any) : int =
 let var i := 0
     function isdigit(s : string) : int =
            ord(buffer)>=ord("0") & ord(buffer)<=ord("9")
     function skipto() =
       while buffer=" " | buffer="\n"
         do buffer := getchar()
  in skipto();
     any.any := isdigit(buffer);
     while isdigit(buffer)
```

```
        do (i := i*10+ord(buffer)-ord("0"); buffer := getchar());
      i
  end

  type list = {first: int, rest: list}

  function readlist() : list =
     let var any := any{any=0}
         var i := readint(any)
      in if any.any
          then list{first=i,rest=readlist()}
          else nil
     end

  function merge(a: list, b: list) : list =
    if a=nil then b
    else if b=nil then a
    else if a.first < b.first
       then list{first=a.first,rest=merge(a.rest,b)}
       else list{first=b.first,rest=merge(a,b.rest)}

  function printint(i: int) =
   let function f(i:int) = if i>0
         then (f(i/10); print(chr(i-i/10*10+ord("0"))))
    in if i<0 then (print("-"); f(-i))
       else if i>0 then f(i)
       else print("0")
   end

  function printlist(l: list) =
    if l=nil then print("\n")
    else (printint(l.first); print(" "); printlist(l.rest))

    var list1 := readlist()
    var list2 := (buffer:=getchar(); readlist())


   /* BODY OF MAIN PROGRAM */
  in printlist(merge(list1,list2))
end
```

Output

```
No error!
```

Extra Credit

See *extra* file for the first extra problem.
And I use my own tiger.lex.