Compilers Principals - Chapter4

Zhixin Zhang, 3210106357

Problems：4.2.

4.2

Implement Program 4.4 as a recursive-descent parser, with the semantic actions embedded in the parsing functions.

```
// Program 4.4
%{
typedef struct table *Table_;
Table_ {string id; int value; Table_ tail};
Table_ Table(string id, int value, struct table *tail); (see page 13)
Table_ table=NULL;
int lookup(Table_ table, string id) {
assert(table!=NULL);
if (id==table.id) return table.value;
else return lookup(table.tail, id);
}
void update(Table_ *tabptr, string id, int value) {
*tabptr = Table(id, value, *tabptr);
}
%}
%union {int num; string id;}
%token <num> INT
%token <id> ID
%token ASSIGN PRINT LPAREN RPAREN
%type <num> exp
%right SEMICOLON
%left PLUS MINUS
%left TIMES DIV
%start prog
%%
prog: stm
stm : stm SEMICOLON stm
stm : ID ASSIGN exp {update(&table,ID,$3);}
stm : PRINT LPAREN exps RPAREN {printf("\n");}
exps: exp {printf("%d ", $1);}
exps: exps COMMA exp {printf("%d ", $3);}
exp : INT {$$=$1;}
exp : ID {$$=lookup(table,$1);}
exp : exp PLUS exp {$$=$1+$3;}
exp : exp MINUS exp {$$=$1-$3;}
exp : exp TIMES exp {$$=$1*$3;}
exp : exp DIV exp {$$=$1/$3;}
exp : stm COMMA exp {$$=$3;}
exp : LPAREN exp RPAREN {$$=$2;}
```

Solution:

```
enum token {
  ID, INT, PLUS, MINUS, TIMES, DIV, COMMA,
  LPAREN, RPAREN, PRINT, SEMICOLON, ASSIGN
};
union tokenval { string id; int num; };
enum token tok;
union tokenval tokval;

typedef struct table *Table_;
Table_ {string id; int value; Table_ tail};
Table_ Table(string id, int value, struct table *tail);
Table_ table = NULL;
int lookup(Table_ table, string id)
{
  assert(table != NULL);
  if(id == table.id) return table.value;
  else return lookup(table.tail, id);
}
void update(Table_ *tabptr, string id, int value)
{
  *tabptr = Table(id, value, *tabptr);
}
void eatOrSkipTo(int expected, int *stop)
{
  if(tok == expected) eat(expected);
  else skipto(stop);
}

int SMT_follow[] = { SEMICOLON, COMMA };
void SMT(void)
{
  switch(tok)
  {
    case ID:
      string id = tokval.id;
      if (lookahead() == ASSIGN)
      {
        advance();
        update(table, id, E());
      }
      skipto(SMT_follow);
      break;
    case PRINT:
      advance();
      if(lookahead() == LPAREN)
      {
        L();
        eatOrSkipTo(RPAREN, SMT_FOLLOW);
      }
      break;
    default:
      printf("expected ID or PRINT\n");
      skipto(SMT_FOLLOW);
    }
}

int L_follow[] = { COMMA, SEMICOLON, RPAREN };
void L(void)
{
  switch(tok)
  {
    case ID:
    case INT:
      printf("%d", E());
      break;
    default:
      printf("expected ID or INT\n");
      skipto(L_follow);
  }
}

int EXP_follow[] = { SEMICOLON, PLUS, MINUS, TIMES, DIV, RPAREN };
int EXP()
{
  switch(tok)
  {
    case ID:
      int i = loopup(table, tokval.id);
      token pre = lookahead();
      if(pre == PLUS || pre == MINUS || pre == TIMES || pre == DIV)
      {
        advance();
        return Binary(i);
      }
      advance();
      return i;
    case INT:
      int i = tokval.num;
      token pre = lookahead();
      if(pre == PLUS || pre == MINUS || pre == TIMES || pre == DIV)
      {
        advance();
        return Binary(i);
      }
      advance();
      return i;
    case LPAREN:
      int i = E();
      token pre = lookahead();
      if(pre == RPAREN) return i;
      else { skipto(EXP_follow); return 0; }
    default:
      print("expected ID, INT, LPAREN\n");
      skipto(EXP_follow);
      return 0;
  }
```

```
}

int Binary_follow[] = { ID, INT, LPAREN };
int Binary (int v)
{
  switch(tok)
  {
    case PLUS:
      advance(); return v + EXP();
    case MINUS:
      advance(); return v - EXP();
    case TIMES:
      advance(); return v * EXP();
    case DIV:
      advance(); return v / EXP();
    default:
      printf("expected some Binary Operator\n");
      skipto(Binary_follow);
      return 0;
  }
}
```