Programspråk

Laboration 2

Datum:
Namn:
Kurs:

Innehållsförteckning

Parsing av Pascalprogram

Lexer

ranslateToToken

Parsern

Parsning av ett program

nd

Bilaga – Programkod

Parser.pro

```
/* TESTER ATT KORA */

runProgram(_):-
parseFiles(['Test/testok1.pas','Test/testok2.pas','Test/testok3.pas','Test/testok4.pa
s','Test/testok5.pas','Test/testok6.pas','Test/testok7.pas','Test/fun1.pas',
'Test/fun2.pas', 'Test/fun3.pas', 'Test/fun4.pas', 'Test/fun5.pas', 'Test/testa.pas',
'Test/testb.pas',      'Test/testc.pas',      'Test/testd.pas',      'Test/teste.pas',
'Test/testf.pas',      'Test/testg.pas',      'Test/testh.pas',      'Test/testi.pas',
'Test/testj.pas',      'Test/testk.pas',      'Test/testl.pas',      'Test/testm.pas',
'Test/testn.pas',      'Test/testo.pas',      'Test/testp.pas',      'Test/testq.pas',
'Test/testr.pas',      'Test/tests.pas',      'Test/testt.pas',      'Test/testu.pas',
'Test/testv.pas',      'Test/testw.pas',      'Test/testx.pas',      'Test/testy.pas',
'Test/testz.pas', 'Test/sem1.pas', 'Test/sem2.pas', 'Test/sem3.pas', 'Test/sem4.pas',
'Test/sem5.pas']).

parseFiles([]).
parseFiles([H|T]):-parse(H), write('\n\n'), parseFiles(T).


/* -- LEXER -- */
singleCharacter(44). /*,*/
singleCharacter(59). /*;*/
singleCharacter(58). /*:*/
singleCharacter(63). /*?*/
singleCharacter(33). /*!*/
singleCharacter(46). /*.*/
singleCharacter(40). /*(*/
singleCharacter(41). /*)*/
singleCharacter(42). /***/
singleCharacter(43). /*+*/
singleCharacter(61). /*=*/
singleCharacter(-1). /*EOF*/

keyWord(program).
keyWord(input).
keyWord(output).
keyWord(var).
keyWord(begin).
keyWord(end).
keyWord(integer).
keyWord(real).

translateToToken([:,=],assign):-!.
translateToToken(.,.):-!.
translateToToken((,),,):-!.
translateToToken('(','('):-!.
translateToToken(')',')'):-!.
translateToToken(:,:):-!.
translateToToken(;,;):-!.
translateToToken(*,*):-!.
translateToToken(+,+):-!.
translateToToken(-1,eof):-!.
translateToToken(S,number):-S\=[_,_], atom_chars(S, [H|_]), isNum(H).
translateToToken(L,id):-L\=[_,_],!.

inWord(C,C):-C>96, C<123. /* a - z */
inWord(C,L):-C>64, C<91, L is C+32. /*A - Z*/
inWord(C,C):-C>47, C<58. /*1-9*/
inWord(39,39). /*' (')*/
inWord(45,45). /*-*/

lastWord('.').
lastWord(-1).

isNum(0).
isNum('0').
isNum(1).
```

```
isNum('1').
isNum(2).
isNum('2').
isNum(3).
isNum('3').
isNum(4).
isNum('4').
isNum(5).
isNum('5').
isNum(6).
isNum('6').
isNum(7).
isNum('7').
isNum(8).
isNum('8').
isNum(9).
isNum('9').

openFile(File,S):-see(File), readIn(S), seen.

readIn([W|Ws]):-get0(C), readWord(C,W,C1), restSent(W,C1,Ws).

restSent(W,_,[]):- lastWord(W), !.
restSent(_,C,[W1|Ws]):-readWord(C,W1,C1), restSent(W1,C1,Ws).

readWord(C,W,C1):-C\=(-1), singleCharacter(C), !, name(W,[C]), get0(C1).
readWord(C,W,C2):-C\=(-1),   inWord(C,NewC),   !,   get0(C1),   restWord(C1,Cs,C2),
name(W,[NewC|Cs]).
readWord(C,W,C2):-C\=(-1), get0(C1), readWord(C1,W,C2).
readWord(_,-1,-1):-!.

restWord(C,[NewC|Cs],C2):-inWord(C,NewC), !, get0(C1), restWord(C1,Cs,C2).
restWord(C,[],C).

makeTokenList([E],E):-keyWord(E), !.
makeTokenList([E],[B]):-translateToToken(E,B), !.
makeTokenList([H|T],[H|B]):-keyWord(H), !, makeTokenList(T,B).
makeTokenList([H,H2|T],[F|B]):-translateToToken([H,H2],F), !, makeTokenList(T,B).
makeTokenList([H|T],[F|B]):-translateToToken(H,F), makeTokenList(T,B).

getTokensFromFile(File, B):-openFile(File, S), makeTokenList(S,B).


/* -- PARSER -- */
match(To, [To|Tail], Tail):-!.

parse(File):-getTokensFromFile(File,B), write('-- Program: '), write(File), write(' -
-\n'), prog(B).

prog(P):-progHeader(P,T),   varPart(T,Y),   statPart(Y),   !,   write('**   Parse
successful.\n').
prog(_):-write('** Parse failed.\n').

progHeader(In,Ut):- match(program,In,Sv1),
                    match(id,Sv1,Sv2),
                    match('(',Sv2,Sv3),
                    match(input,Sv3,Sv4),
                    match(,,Sv4,Sv5),
                    match(output,Sv5,Sv6),
                    match(')',Sv6,Sv7),
                    match(;,Sv7,Ut).

varPart(In,Ut):-   match(var,In,Sv1),
                                                    varDecList(Sv1,Ut).

varDecList(In, Ut):-varDec(In, [H|T]), H\=begin, !, varDecList([H|T], Ut).
varDecList(In, Ut):-varDec(In, Ut).

varDec(In,   Ut):-         idList(In,   Sv1),   match(:,Sv1,Sv2),   varType(Sv2,Sv3),
match(;,Sv3,Ut).

idList(In, Ut):-   match(id, In, [H|T]), H=,, !, match(,,[H|T], Sv1), idList(Sv1,Ut).
idList(In, Ut):-   match(id, In, Ut).

varType(In, Ut):-  match(integer,In, Ut);match(real, In, Ut).
```

```
statPart(In):-       match(begin,In,Sv1), statList(Sv1, Sv2), match(end, Sv2, Sv3),
match(.,Sv3,Rest), Rest=[].

statList(In, Ut):- stat(In, [H|T]), H=;, match(;,[H|T],Sv1), statList(Sv1, Ut).
statList(In, Ut):- stat(In, Ut).

stat(In, Ut):-               assignStat(In, Ut).

assignStat(In, Ut):-match(id,In,Sv1), match(assign, Sv1, Sv2), expr(Sv2, Ut).

expr(In, Ut):-               operand(In,  [H|T]),  (H='*';  H='+'),  operator([H|T],
Sv2), expr(Sv2, Ut).
expr(In, Ut):-               operand(In, Ut).

operand(In, Ut):-  match(id, In, Ut); match(number, In, Ut).

operator(In, Ut):- match(+, In, Ut); match(*, In, Ut).
```

Bilaga – Körning

```
-- Program: Test/testok1.pas --
** Parse successful.


-- Program: Test/testok2.pas --
** Parse successful.


-- Program: Test/testok3.pas --
** Parse successful.


-- Program: Test/testok4.pas --
** Parse successful.


-- Program: Test/testok5.pas --
** Parse successful.


-- Program: Test/testok6.pas --
** Parse successful.


-- Program: Test/testok7.pas --
** Parse successful.


-- Program: Test/fun1.pas --
** Parse successful.


-- Program: Test/fun2.pas --
** Parse failed.


-- Program: Test/fun3.pas --
** Parse successful.


-- Program: Test/fun4.pas --
** Parse successful.


-- Program: Test/fun5.pas --
** Parse successful.


-- Program: Test/testa.pas --
** Parse failed.
```

```
-- Program: Test/testb.pas --
** Parse failed.


-- Program: Test/testc.pas --
** Parse failed.


-- Program: Test/testd.pas --
** Parse failed.


-- Program: Test/teste.pas --
** Parse failed.


-- Program: Test/testf.pas --
** Parse failed.


-- Program: Test/testg.pas --
** Parse failed.


-- Program: Test/testh.pas --
** Parse failed.


-- Program: Test/testi.pas --
** Parse failed.


-- Program: Test/testj.pas --
** Parse failed.


-- Program: Test/testk.pas --
** Parse failed.


-- Program: Test/testl.pas --
** Parse failed.


-- Program: Test/testm.pas --
** Parse failed.


-- Program: Test/testn.pas --
** Parse failed.


-- Program: Test/testo.pas --
** Parse failed.


-- Program: Test/testp.pas --
** Parse failed.


-- Program: Test/testq.pas --
** Parse failed.


-- Program: Test/testr.pas --
** Parse failed.


-- Program: Test/tests.pas --
** Parse failed.


-- Program: Test/testt.pas --
** Parse failed.
```

```
-- Program: Test/testu.pas --
** Parse failed.


-- Program: Test/testv.pas --
** Parse failed.


-- Program: Test/testw.pas --
** Parse failed.


-- Program: Test/testx.pas --
** Parse failed.


-- Program: Test/testy.pas --
** Parse failed.


-- Program: Test/testz.pas --
** Parse failed.


-- Program: Test/sem1.pas --
** Parse successful.


-- Program: Test/sem2.pas --
** Parse successful.


-- Program: Test/sem3.pas --
** Parse failed.


-- Program: Test/sem4.pas --
** Parse successful.


-- Program: Test/sem5.pas --
** Parse successful.
```