# Problem Description

Name: Algol60 Interpreter

Description: The following program, written in Snobol4, will accept Algol60 programs (procedures) and output a response that will indicate whether the program is legally formatted or not.

Input: An Algol60 procedure, correctly formatted or not. The program treats everything like a procedure and will act as such.

Output: ‘Yes’ if the given input is correct, ‘No’ otherwise.

# Pattern Descriptions

COMMENTS

Description: Each of the following three patterns serves to match one form of the comments; one for ';', 'begin', and 'end'.

IDENTIFIER/NORMAL VARIABLE

Description: Represents an identifier. To enforce the rule of identifiers where they must all begin with a letter, the position of the cursor is attained first. This is followed by checking the character against the list of letters. If that succeeds, then the entire pattern will succeed, since nothing needs to come after.

;\*NUMBER

;\* Description: Matches both signed and unsigned numbers. The '+' and '-' that makes a number signed is optional, as well as a decimal point. If a decimal is found, however, it must be followed by at least one digit.

;\*ARRAY

;\* Description: Matches subscripted variables. If an identifier is followed by a '[', then it is checked for a list of array items. These items can be either another subscripted variable or an arithmetic expression.

;\*FUNCTIONCALL

;\* Description: Matches function and procedure calls. An identifier followed by a '(' will then be checked for a list of parameters until a final ')' is found. If the identifier was not followed by a '(' then it the next thing to check for is a ';' or an 'end' to signify that this is a function. Otherwise, it will be recognized as an identifier by something else.

;\*BOOLEANEXP

;\* Description: Matches boolean expressions. These include if clauses that are followed by simplified boolean expressions (those that are composed of nothing but primaries and operators) or just a simplified boolean expression. Arithmetic expressions are used here, so the operation has to be delayed.

;\*ARITHMETICEXP

;\* Matches arithmetic expressions. These include if clause versions and simple arithmetic expressions composed of only functions, variables, and numbers. Grouping is allowed through parenthesis.

;\*DESIGNATIONEXP

;\* Description: Matches designation expressions. These are either subscripted variables, labels, or if clauses followed by a designation expression.

;\*ASSIGNMENT STATEMENT

;\* Description: Matches an assignment statement. This can be any number of left part expressions but must be ended with an arithmetic or boolean expression.

;\*GO TO STATEMENT

;\* Description: Matches a go to statement.

;\*DUMMY STATEMENT

;\* Description: Matches a dummy statement.

;\*COMPOUND STATEMENTS AND BLOCKS

;\* Description: One matches a compound statement and another matches a block statement.

;\*CONDITIONAL STATEMENT

;\* Description: Matches a conditional statement. This is just an if clause followed by an unconditional statement (compound statement, a block, a for statement, an assign statement, a go to statement, a function call, or a dummy statement) that can also be followed by an else with a statement.

;\*FOR STATEMENT

;\* Description: Matches a for statement. This includes any variations for statements are capable of taking.

;\*TYPE DECLARATIONS

;\* Description: Matches type declarations. These take a type and, after ensuring there is a space, get however many identifiers may follow.

;\*ARRAY DECLARATIONS

;\* Description: Matches array declarations. Does not accept actual arrays, but those with bounds, etc.

;\*SWITCH DECLARATIONS

;\* Description: Matches switch declarations.

;\*PROCEDURE DECLARATIONS

;\* Description: Matches procedure declarations. Since the code following a procedure must be a block, it is used to substitute the document's <code> tag.

# Tests

### Test Plan

To test this, I used several different procedures that were either legal or not. Only procedures were tested.

### Settings

The program was written and tested in tkSlide version 0.35. The OS used was a 64-bit Windows 7. The parameters used were –b –P 9999999.

**Note**: In tkSlide, at least during my testing, pasting input from a text file that contained tabs broke the program. I’m not certain why this is the case, as it only happens when tabs are used, yet not spaces. Due to this, the input that is used must contain no tabs, only direct spaces.

**Note:** This program is setup to read procedures and *only* procedures. While the patterns work for everything, the interpreter was designed to read procedures and work down.

### Input and Results

procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);

value n, m; array a; integer n, m, i, k; real y;

comment The absolute greatest element of the matrix a, of size n by m

is transferred to y, and the subscripts of this element to i and k;

begin integer p, q;

y := 0; i := k := 1;

for p:=1 step 1 until n do

for q:=1 step 1 until m do

if abs(a[p, q]) > y then

begin y := abs(a[p, q]);

i := p; k := q

end

end Absmax

End Result: Yes

----

procedure Spur(a)Order:(N)Result:(S) ; value n;

array a; integer n; real s;

begin integer k;

s := 0;

for k:=1 step 1 until n do s := s + a[k,k]

end

End Result: Yes

----

integer procedure Step (u) ; real u;

Step := if 0<-u and u<=1 then 1 else 0

End Result: Yes

---

procedure Innerproduct(a,b)Order:(k,p)Result:(y) ;

value k; integer k,p; real y,a,b ;

begin real s;

s := 0;

for p:=1 step 1 until k do s:=s+a\*b; y:=s

end Innerproduct

End Result: Yes

---

procedure euler (fct, sum, eps, tim) ; value eps, tim;

integer tim; real procedure fct; real sum, eps;

comment euler computes the sum of fct(i) for i from zero up to infinity by means of a suitably refined euler ;

begin integer i,k,n,t; array m[0:15] ; real mn, mp, ds ;

i:=n := t := 0 ; m[0] := fct(0) ; sum := m[0]/2;

nextterm: i := i+1 ; mn:= fct(i) ;

for k:=0 step 1 until n do begin mp := (mn+m[n])/2 ; m[k] := mn ; mn := mp end means ;

if (abs(mn) < abs(m[n])) and (n<15) then begin ds := mn/2; n:=n+1; m[n] := mn end accept else ds := mn;

sum := sum + ds ;

if abs(ds)<eps then t:= t+1 else t:=0;

if t<tim then go to nextterm

end euler

End Result: Yes

---

procedure Innerproduct(a,b)Order:(k,p)Result:(y) ;

value ; integer ; real ,a,b ; <= **Error**

begin real s;

s := 0;

for p:=1 step 1 until k do s:=s+a\*b; y:=s

end Innerproduct

End Result: No

---

procedure euler (fct, sum, eps, tim) ; value eps, tim;

integer tim; real procedure fct; real sum, eps;

comment euler computes the sum of fct(i) for i from zero up to infinity by means of a suitably refined euler ;

begin integer i,k,n,t; array m[0:15] ; real mn, mp, ds ;

i:=n := t := 0 ; m[0] := fct(0) ; sum := m[0]/2;

nextterm: i := i+1 ; mn:= fct(i) ;

for k:=0 step 1 until n do begin mp := (mn+m[n])/2 ; m[k] := mn ; mn := mp end means <= **No semicolon**

if (abs(mn) < abs(m[n])) and (n<15) then begin ds := mn/2; n:=n+1; m[n] := mn end accept else ds := mn;

sum := sum + ds ;

if abs(ds)<eps then t:= t+1 else t:=0;

if t<tim then go to nextterm

end euler

End Result: No

---

integer procedure Step (u) ; real u;

Step := if 0<-u and u<=1 then 1 else 0 end <= **end marker where there shouldn’t be**

INTEGER procedure Step (u) ; real u; <= **integer tag in all caps**

Step := if 0<-u and u<=1 then 1 else 0

End Result for both: No

---

procedure Spur(aOrder:(N)Result:(S) ; value n; <= **incorrect formatting of procedure declaration**

array a; integer n; real s;

begin integer k;

s := 0;

for k:=1 step 1 until n do s := s + a[k,k]

end

End Result: No

---

procedure Innerproduct(a,b)Order:(k,p)Result:(y) ; <! **Test of nested procedure declarations.**

value k; integer k,p; real y,a,b ; <! **This was supposed to work.**

procedure FKT;

comment : This program is a load of horse bullwinky. You won't ever need to see this part. Ya probably stopped reading the moment you saw the word 'comment' anyway ;

begin array z,y1,y2,y3[1:n] ; real x1, x2, x3, H; Boolean out;

integer k,j ; own real s,Hs ;

procedure RK1ST(x,y,h,xe,ye) ; real x,h,xe ; array y, ye;

comment : Dear god, what the hell?! It looks so strange! ;

begin

array w[1:n],a[1:5]; integer k,j ;

a[1]:=a[2]:=a[5]:= h/2 ; a[3] := a[4] := h ;

xe :=x;

for k := 1 step 1 until n do ye[k] := w[k]:= y[k];

for j:=1 step 1 until 4 do

begin FKT(xe,w,n,z) ; xe:= x+a[j] ;

for k:= 1 step 1 until n do

begin

w[k]:=y[k]+a[j]\*z[k] ; ye[k]:= ye[k]+a[j+1]\*z[k]/3

end k

end j

end RK1ST ;

begin real s;

s := 0;

for p:=1 step 1 until k do s:=s+a\*b; y:=s

end Innerproduct

End Result: Yes