GOOGLE

by W. M. McKeeman and Niklaus Wirth

Abstract: A compiler designed for, and implemented under the Stanford PDP-1 time-sharing system is described.

The research reported here was supported in part by the National Science Foundation (GP-3207)
GOGOL

by W. M. McKeeman and Niklaus Wirth

Introduction:

Gogol is a simple, integer arithmetic language used under the PDP-1 time sharing system at Stanford. This memorandum includes the syntactical definition of the language and a number of sample programs as well as a brief description of the operational characteristics of the compiler. Gogol was designed to permit fast compilation of efficient machine code directly into memory. The speed of compilation together with the accessibility of the text editor make program debugging relatively rapid. The examples presented here plus the availability of the compiler should form an adequate basis for learning to use the language. More detailed information depends heavily on a knowledge of PDP-1 hardware.

Both this report and the compiler are preliminary in nature. The compiler will certainly undergo modification when the Philco displays and a CRT text editor with disk files become available. The authors solicit constructive criticism consistent with the objectives of the system.
Syntax and Semantics:

The following syntax defines the form of the language. For the most part, a deviation from the syntax will cause a compile time error.

\[
\text{<program> } ::= \text{<block>}
\]
\[
\text{<block> } ::= \text{begin } \text{<block list> end}
\]
\[
\text{<block list> } ::= \text{<block entry> } | \text{<block entry>,<block list>}
\]
\[
\text{<block entry> } ::= \text{<declaration> } | \text{<statement>}
\]
\[
\text{<declaration> } ::= \text{<integer declaration> } | \text{<index declaration> } | \text{<label declaration> } | \text{<procedure declaration>}
\]
\[
\text{<integer declaration> } ::= \text{integer <storage allocation list>}
\]
\[
\text{<storage allocation list> } ::= \text{<storage element> } | \text{<storage element>,<storage allocation list>}
\]
\[
\text{<storage element> } ::= \text{<identifier> } | \text{<identifier>[<bound pair>]}
\]
\[
\text{<bound pair> } ::= \text{<integer>:<integer>}
\]
\[
\text{<label declaration> } ::= \text{label <identifier list>}
\]
\[
\text{<index declaration> } ::= \text{index <identifier list>}
\]
\[
\text{<identifier list> } ::= \text{<identifier> } | \text{<identifier list>,<identifier>}
\]
\[
\text{<procedure declaration> } ::= \text{<procedure heading>;<statement>}
\]
\[
\text{<procedure heading> } ::= \text{procedure <identifier> } | \text{procedure <identifier> (<identifier list>)}
\]
(statement) ::= (empty) | (if statement) | (unconditional statement) |

(for statement) | (label): (statement)

(if statement) ::= (if clause) (unconditional statement) |

(if clause) (unconditional statement) else (statement)

(if clause) ::= if (Boolean expression) then

(Boolean expression) ::= (expression) |

(expression) (relational operator) (expression)

(relational operator) ::= < | ≤ | = | ≠ | ≥ | >

(unconditional statement) ::= (go to statement) | (assignment statement) |

(procedure statement) | (code statement) | (block)

(go to statement) ::= go to (label) | go to (variable) |

go to (procedure statement)

(assignment statement) ::= (label) (right part) | (expression) (right part) |

(procedure statement) (right part)

(right part) ::= -> (variable) | -> (variable) (right part)

(for statement) ::= for (lower bound) to (upper bound) -> (variable) do

(statement)

(lower bound) ::= (expression)

(upper bound) ::= (expression)

(procedure statement) ::= (procedure identifier) |

(procedure identifier) (parameter list)

(parameter list) ::= (actual parameter) | (actual parameter), (parameter list)

(actual parameter) ::= (expression) | (label) | (procedure statement)

(code statement) ::= code (code list)

(code list) ::= (instruction) | (instruction), (code list)

(instruction) ::= (constant) | (constant) (address)

(address) ::= (constant) | (variable)
\[\text{expression} ::= \text{intersection} \mid \text{expression}\text{or operator}\text{intersection}\]
\[\text{or operator} ::= V \mid \lor\]
\[\text{intersection} ::= \text{negation} \mid \text{intersection}\text{and}\text{negation}\]
\[\text{negation} ::= \text{sum} \mid \neg\text{sum} \mid \text{abs sum}\]
\[\text{sum} ::= \text{term} \mid \text{sum}\text{adding operator}\text{term}\]
\[\text{adding operator} ::= + \mid -\]
\[\text{term} ::= \text{factor} \mid \text{term}\text{multiplying operator}\text{factor}\]
\[\text{multiplying operator} ::= \times \mid / \mid \text{mod}\]
\[\text{factor} ::= \text{primary} \mid \text{factor}\text{unsigned integer}\]
\[\text{primary} ::= \text{variable} \mid \text{constant} \mid (\text{expression})\]
\[\text{variable} ::= \text{index} \mid \text{integer identifier}\]
\[\text{integer identifier}[\text{index expression}]\]
\[\text{index expression} ::= \text{integer} \mid \text{index} \mid \text{adding operator}\text{unsigned integer}\]
\[\text{constant} ::= \text{unsigned integer} \mid \text{string} \mid '\text{octal integer}'\]
\[\text{integer} ::= \text{unsigned integer} \mid -\text{unsigned integer}\]
\[\text{unsigned integer} ::= \text{digit} \mid \text{unsigned integer}\text{digit}\]
\[\text{string} ::= "\text{any string of 3 characters not including "}"\]
\[\text{octal integer} ::= \text{octal digit} \mid \text{octal integer}\text{octal digit}\]
\[\text{identifier} ::= \text{letter} \mid \text{identifier}\text{letter} \mid \text{identifier}\text{digit}\]
\[\text{letter} ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z\]
\[\text{octal digit} ::= 1|2|3|4|5|6|7|0\]
\[\text{digit} ::= \text{octal digit} | 8|9\]
All non-reserved identifiers must be declared prior to their use. An identifier is valid from the position of its declaration to the end of the closest enclosing, matching begin end pair. Upper case letters are distinguished from lower case -- only the first six characters of an identifier are significant.

A variable is declared either as an index or an integer, both types are own variables in the sense of Algol in that unique storage is assigned at each declaration. The two types differ only in that an integer variable may designate a one dimensional array while an index variable may appear in an index expression.

A subscript expression may take only the forms: $i \pm c$, $i$, or $i + c$ where $i$ is declared an index variable and $c$ is a constant. No checking on array limits is done.

Constants may have the forms:

```
13 = 13 decimal
'13 = 13 octal
"13" = 103 octal, characters 1 and 3
true = -0 octal
false = 0
```

The largest integer is

```
131072 decimal = 377777 octal = 2^{17}
```

The operators:

<table>
<thead>
<tr>
<th>operator</th>
<th>associated machine instructions</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>sar sal ral</td>
<td>shift right or left</td>
</tr>
<tr>
<td>{x}</td>
<td>mul</td>
<td>multiply</td>
</tr>
<tr>
<td>/</td>
<td>div</td>
<td>divide</td>
</tr>
<tr>
<td>mod</td>
<td>div</td>
<td>remainder</td>
</tr>
<tr>
<td>+</td>
<td>add</td>
<td>add</td>
</tr>
<tr>
<td>-</td>
<td>sub</td>
<td>subtract</td>
</tr>
<tr>
<td>abs</td>
<td>cma</td>
<td>complement</td>
</tr>
<tr>
<td>⊕</td>
<td>spa cma</td>
<td>absolute value</td>
</tr>
<tr>
<td>∧</td>
<td>and</td>
<td>logical and -(mask)</td>
</tr>
<tr>
<td>⊕</td>
<td>ior</td>
<td>logical or (inclusive)</td>
</tr>
<tr>
<td>▷</td>
<td>xor</td>
<td>exclusive or</td>
</tr>
</tbody>
</table>

are allowed in expressions with the hierarchy implicit in the list above.
Logical and arithmetic operators may be mixed in expressions and have to be understood in the sense of the PDP instructions.

The six relational operators

$<$, $\leq$, $=$, $\neq$, $\geq$, $>$

are allowed in if clauses only. If the Boolean expression is 0 it is false, otherwise it is true. All procedures are functions but may never be used in expressions. Their value must be assigned to a variable or used as an actual parameter to another procedure. The value of a procedure may be a label as well as an integer or a Boolean value.

The bounds in a for loop may be any expressions, the step is always +1. A loop is executed zero or more times.

The operator $\uparrow$ has the meaning shift left or right if it is followed with a + or -, otherwise it has the meaning rotate left. A shift is the same as a multiplication by a power of 2.
Library Procedures

read → a;
write (a);
tyo (a);
tyi → a;
message ("hi");
data → a[l]
overflow → a;

are examples of calls on each of the 7 gogol library procedures.

"read" expects a decimal number from the keyboard. The statement above will cause the printout

a =

and then wait for digits. Any illegal character will terminate the read-in and the execution will continue. Leading spaces, tabs and carriage returns are ignored.

"write" simply types the decimal value of its parameter. It does not do any carriage control.

"tyo" types the character in the right-most 6 bits of its parameter. For example,

    tyo("c");
    tyo('36);
    tyo('77);

will produce the letter c, a tab and then a carriage return.

"tyi" accomplished the inverse operation.

"message" is a routine to produce titles and the like in the output. The parameter of "message" must be a string and can be of any length.
"data" is a multiple read that gets its data from the source tape following the end of the program. Successive data values must be separated by tabs, spaces, stop codes or carriage returns. Any illegal character will terminate the data read in. The first value will be stored in the location of the variable in the assignment. Successive values are assigned sequentially in memory after the first value. Hence in the example above, if ten values are on the tape after the end, we would find data initialization in variables a[1] through a[10]. "data" may be called as often as desired.

"overflow" is true (-0) if the overflow light is on and false (0) if the overflow light is off. The overflow light is turned off by the call on "overflow".
Error Messages:

Error halts may occur during compilation. The compiler will print a message, for example:

```
error number 3
last identifier integer
```

to help the user identify his error. A current list of error messages is kept on the PDP-1 console and is also available at the Computation Center. The next four lines of code following the error will also be typed out.

A divide by zero causes an error message but does not stop execution.

A reference to an undefined label will cause a halt within the bounds of the program with no error message.

Various other error conditions can occur. A compile-time stack overflow will generally result in illegal instruction execution in lower memory. It is normally caused by an expression with too many unmatched left parentheses.

Array overruns can be the source of trouble when a compiled program behaves erratically during execution.

During the next few months it is inevitable that errors will be found in the compiler. If you suspect that the compiler has misbehaved, please:

1. Get a listing of a compilation,
2. l,ddt
3. Use ddt to type out the relevant code emitted by the compiler.
4. Bring all of the output to W. M. McKeeman, room 117, Polya.
Operational Characteristics:

The 4K user core on the PDP is allocated thusly:

\[
\begin{array}{c}
0 \\
1 \\
II \\
III \\
7777_8 \\
3233_8 \\
4177_8 \\
\end{array}
\]

.locations subject to change

Section I contains the part of the compiler that is left in this place during execution for the user's convenience. This includes the library subroutines.

Section II is the region into which the compiled code is deposited. The user's constants also build down from the top of Section II.

Section III contains the compiler. Just before execution the compiler is cleared and the compiled program uses Section III for variable storage. For small programs the top of Section III is generally clear and can be used for DDT or machine language subroutines.

Under ODIN, a useful set of system commands are:

c,edit,(i,text
o,text
l,et)
c,et,(l,et
e,s1)
c,s1,(t,440
e,s1)
c,exec,(i,text
l,gogol)
c,list,(i,text
l,gogol)

These commands allow rapid switching from editing to compilation and execution. One page is sufficient for a gogol program, thus the input and output pointers may indicate the same area on the drum.
Once a text is prepared the command

\texttt{e,exec}

will cause a compilation and execution in \textit{approximately} 6 seconds elapsed time.

\texttt{e,list}

will cause the compiler to list the source code as it compiles --
with the obvious penalty in time.
Linkage to Machine Language:

The assembler for the PDP produces binary tape that will load into core locations fixed by the programmer during assembly. For example,

```
5000/
  dsp, 0
dsp dsr
law 6000
dac dl
d2, lac i dl
sad dsp
jmp d3
idx dl
sas s7
jmp d2
cla+cma-opr
dsrm jmp .
s7, 7001
dl, 0
d3, lac dl
jmp dsr
```

is a Macro program to search the cells 6000\_8 to 7000\_8 inclusive for the quantity in the ac upon the execution of a

```
  jda dsp.
```

To facilitate loading the binary tape on top of the compiled program, the gogol statement "execute halt",

```
  code '760400;
```

should be included in the source program. After the halt has been executed, the user may read in the binary tape in the usual manner and then continue execution from the time sharing control language. Linkage to the subroutine to find an occurrence of the number -17 then would take the form

```
  code '200000 -17,
  '175000;
  '240000 location;
```
APPENDIX I

List of Reserved Identifiers

overflow
tyo
tyi
read
write
message
data
core

for
do
go
to
if
then
else
procedure
integer
index
begin
end
label
leq
geq
neg
mod
abs
APPENDIX II

e, list

begin comment test uniqueness of identifiers;

    integer a, A, aA, Aa, aa, AA, aa, aAa, aAa, AAA, AAA, aaa;

1->a;
2->A;
3->aA;
4->Aa;
5->aa;
6->AA;
7->aAa;
8->AAA;
9->aaa;
write(a);write(A);write(aa);
write(Aa);write(aa);write(AA);
write(aAa);write(AAA);write(aaa);
end

program  0424   0470
constants  3213   3227
variables  3230   3266
halt      at  40470

II-1
begin comment GOGOL prime generator;
  integer p[1:10], a;
  index i, k;
  procedure tab; tyo('36);
  procedure cr; tyo('77);
  procedure divisor(a);
  begin
    index k;
    label 1;
    for 1 to 1 + k do
      if a mod p[k] = 0 then
        begin
          comment locate a divisor;
          true => divisor;
          go to 1
        end;
    false => divisor;
    1:
  end;

  procedure enter(k);
  begin
    i+1=>1;
    k => p[i];
    write(i);
    tab;
    write(p[i]);
    cr;
  end;

  0=>1;
  for 2 to 30 => k do
  begin
    divisor(k) => a;
    if ~a then enter(k);
  end

end

program 0424  0562
constants 3217  3227
variables 3230  3276

1  2  3  4  5  6  7  8  9  10
2  3  5  7 11 13 17 19 23 29

halt. at 40562

II-2
begin comment factor a small integer into prime factors;
integer a[1:10], x; index i;
label 10,11,12,13,14;
procedure out(x); comment print out one character in upper case;
begin tyo('74); tyo(x); tyo('72) end;
data->a[1]; comment store ten values in a[1] through a[10]
The values to be stored are on the tape after the end of the program;
10: read->x; if x > 1 then
begin
11: for 1 to 10 + 1 do
begin
if x mod a[i] = 0 then
begin write(a[i]); out("x"); x/a[i]->x;
if x=1 then go to 14 else go to 11
end;
end;
end;
out("?"); tyo('77); go to 10;
14: write(1); tyo('77); go to 10;
end
begin comment test for loops;
integer a[1..10], b;
index k;

for 1 to 10 \rightarrow k do k \rightarrow a[k];
for 1 to 10 \rightarrow k do write(a[k]);
tyo('77);
for 1 to 4 \rightarrow b do
  for 2 to b \rightarrow k do
    for 1 to a[k] - 1 \rightarrow a[1] do write(b);

end

begin comment GOGOL TIME FUNCTION test;
  procedure time;
    code '720032, '320000 time;
    integer-k, m;
    label l;
    message(""
    clock| time
    reading| diff.
  ");
  l:
    time -> k;
    write(k);
    message(" ");
    write(k-m);
    k->m;
    message(" ms.
  ");
    go to l;
end

program      0443 0507
constants    3222 3227
variables    3230 3253

clock| time
reading| diff.
18010| 18010 ms.
21826| 3816 ms.
25744| 3918 ms.
30740| 4996 ms.
34626| 3886 ms.
38515| 3889 ms.
424

ODIN
begin comment McCarthy's FJCC contest problem
find the smallest integer that is the sum of two cubes for two different pairs of cubes;

integer i, j, a[1:500], isave[1:500], jsave[1:500], t;
index k, nc;
0->nc;
for 1 to 30 -> i do
  for 1 to 1 -> j do
    begin
      1*i*i + j*j*j -> t;
      for 1 to nc -> k do
        if t = a[k] then
          begin
            write(t);
tyo('36);
write(i);
write(j);
tyo('36);
write(isave[k]);
write(jsave[k]);
tyo('77);
          end;
nc+1->nc;
t->a[n];
i->isave[n];
j->jsave[n];
    end
end

program
0424 0575
constants
3220 3227
variables
3230 6221
1729 12 1 10 9
4104 16 2 15 9
13832 24 2 20 18
20683 27 10 24 19
halt. at 40575

II-6
begin comment GOGOL sqrt routine;

procedure sqrt(x);
begin
  label 1; integer t;
  if x < 0 then code '760400;
  if x < 2 then x -> t else
  begin
    integer del;
    x**-1 -> t;
    l:
    t -> sqrt;
    (sqrt + x/sqrt)**-1 -> t,
    abs(t-sqrt) -> del;
    if del > 1 then go to l
    end;
  t->sqrt
end;

integer k, t;
for 1 to 100 -> k do
begin
  write(k);
  sqrt(k) -> t;
  write(txt);
  write(t);
  tyo('77)
end
end

program 0420 0515
constants 3220 3227
variables 3230 3255

1  1
2  1
3  4
4  4
5  4
6  4
7  4
8  9
9  9
10 9
11 9
12 9
13 9
14 9
15 9
16 16
17 16

ODIN
begin  comment  Test the gogol read procedure
1. read in two numbers, m and n
2. compute and write the GCD of m and n
3. go back for more;

integer m, n, q, r;  -label A, B;
A: read -> m;  read -> n;
B: m/n -> q;  m - nxq -> r;
   if r neq 0 then
      begin n->m;  r->n;  go to B end
   else write(n);

end

program
constants
variables
m = 33  n = 55  11
m = 729  n = 51  3
m = -303  n = -99  - 3
m = 0  n = 0
divide by zero
0
m = 7659  n = 3541  9
m = 7254  n = 873  1
error number
halt. at 40148

number too large
begin comment integer conversion routines;

procedure ininteger(base);
begin
  integer i;
  label l, k, m;
  0→ininteger;
  l:
  tyi→i;
  if i = 0 then go to l;
  m:
  if i = 0 then go to k;
  if i = "0" then 0 → i;
  ininteger×base + 1 → ininteger;
  tyi→1;
  go to m;
  k:
end;

procedure outinteger(base, nr);
begin
  integer f[0:17], k, d, i;
  index t, bk;
  label l;
  i→ f[0];
  if f[1] neq base then
    for 0 to 16 + t do
      begin
        base×f[t] + f[t+1];
        if f[t+1]/base neq f[t] then go to 1
      end;
  l:
    for 0 to t + 1 do
      begin
        t-1 → bk;
        nr/f[bk] → d;
        if d = 0 then tyo("0") else tyo(d);
        nr mod f[bk] → nr;
      end;
end;

integer bin, bout;
lable l;
read→bin;
read→bout;
l: outinteger(bout, ininteger(bin));
tyo('77);
go to 1
end

program 0450 0661
constants 3221 3227
variables 3230 3307
bin= 8 bout = 2
377777 11111111111111111
70707 00111000111000111
252525 10101010101010101
    1 00000000000000001
ODIN
492 000754
100 000144