Context Free Grammar for Baby – a Pascal-like language
{ } means "zero or more" [ ] means optional

program → **program** id**;**
variable_declarations
subprogram_declarations
compound_statement **.**

identifier_list → **id** {,**id**}

variable_declarations → **var** variable_declaration ; {variable_declaration;} | ε

variable_declaration → identifier_list : type

type → **integer**

subprogram_declarations → subprogram_declaration ; subprogram_declarations | ε

subprogram_declaration → subprogram_head declarations compound_statement

subprogram_head → **procedure id** arguments **;**

arguments → **(** parameter_list)

parameter_list → identifier_list **:** type { **;** identifier_list **:** type}

compound_statement → **begin** <statement_list> **end**

statement_list → statement {; statement }

statement → assignment_statement
| procedure_statement
| compound_statement
| if_statement
| while_statement
| read_statement
| write_statement
| ε

assignment_statement → **id assignop** expression

if_statement → **if** expression **then** statement [**else** statement]

while_statement → **while** expression **do** statement

procedure_statement → **call id (**expression_list**)**

expression_list → expression { , expression }

expression → simple_expression [**relop** simple_expression]

simple_expression → [**-**]term {**addop** term}

term → factor {**mulop** factor }

factor → **id** | **num** | **true** | **false** | **(**expression**)** | **not** factor

read_statement → **read (** input_list**)**

write_statement → **write(**output_item)

writeln_statement → **writeln(**output_item)

output_item → **string** | expression

input_list → **id** {,**id**}

Lexical conventions

- Comments begin with **!** and extend to the end of a line
- identifiers begin with a letter and consist of letters and digits
- the relational operators (relop) are = < <= > >= <>
  (<> is "not equals" and = compares two values. It is like == in Java)
- the addops are + - **or (or** is like **||** in Java**)**
- the mulops are * / % **and** -- division is integer division (**and** is like &&)
- the assignment operator is **:=**
- a string is enclosed in quotes -- '' '' denotes a quote in a string and \n is the newline character
- programs are case sensitive

Reserved words are:

program	while
var	do
integer	true
bool	false
procedure	and
begin	or
end	not

if then else	call read write

Some simplifying notes:

1. There are no arrays in the language

2. Procedures (i.e. methods) do not return values. In essence all procedures are void methods

3. Every procedure requires at least one parameter. If a procedure (method) does not require a parameter to complete its task, send 0 as a dummy parameter.

```
procedure printHi( dummyParameter: integer) ;
        begin
                write("Hi")
        end
end
```

In the main section of the program you would invoke this procedure as

```
call printHi(0)
```

the argument 0 is just a dummy argument and means nothing

The writeln statement appends a carriage return to the output item

Here is a recursive the tower of Hanoi procedure

```
program Hanoi;
var  height : integer;

procedure move( height, start, goal, extra : integer); ! a void method
begin
        if(height > 0)
        begin
                move (height-1, start, extra, goal);
                write(start);
                write('' to '');
                write(goal);
                write(''\n'');
                call move(height-1, extra, goal, start)
        end
end;

begin                                                   !this is like main(…)
        write(''How many disks? '');
        read(height);
        call move(height,1,3,2);
        write(''\n'')
end.
```

Here is a program that adds and prints the sum of first n positive integers

```
program Add;
 var  n : integer;

 procedure addN( n : integer);
 var count, sum: integer;                  ! local variables
 begin

        count  :=  1;
        sum := 0;

        while count <= n  do
        begin
           sum = sum + count;
           count = count + 1;
        end;

        write (''Sum is “);
        write(sum);
   end;

begin                                 !this is like main(…)
        write(''Enter n: '');
        read(n);
        call addN(n);
        write(''\n'')
end.
```