1 Introduction

This document describes a simple imperative language, called LOOP, that we will use as a running example in class.

2 Lexical issues

We use “Identifier” to designate the lexical class of identifiers. Identifiers in LOOP follow the C convention: they are formed from sequences of letters, digits, and underscores, and must start with either a letter or underscore. LOOP has several reserved identifiers (or keywords), which are the following:

   begin bool do else end false if int not print then true while

We use “Number” to designate the lexical class of integer literals, which are written in decimal notation without a leading sign. LOOP also has a collection of operators and punctuation characters:

   = ; + - < <= == * / && || ( )

Comments in LOOP are C style; they begin with a “/∗” and end with a “∗/.” Any character not covered by the above rules is an error.

3 LOOP syntax

The syntax of LOOP programs is given by the following context-free grammar.

\[
\text{Program} ::= \text{Block} \\
\text{Block} ::= (\text{Declaration } ;)^{\star} (\text{Statement } ;)^{+} \\
\text{Declaration} ::= \text{Type Identifier } = \text{Expression}
\]
Type
::= bool | int

Expression
::= Expression || Expression
| Expression && Expression
| Expression == Expression
| Expression < Expression
| Expression <= Expression
| Expression + Expression
| Expression - Expression
| Expression * Expression
| Expression / Expression
| - Expression
| not Expression
| ( Expression )
| Identifier
| Number

Statement
::= skip
| Identifier = Expression
| if Expression then Statement else Statement
| while Expression do Statement
| print Expression
| begin Block end

The syntax of expressions is disambiguated using precedence and associativity. All binary operators are left associative and have the following precedence (from lowest to highest):

\[
\begin{array}{c}
\| \quad \text{Lowest precedence} \\
&& \\
== \\
< \quad <= \\
+ \quad - \\
* \quad / \quad \text{Highest precedence}
\end{array}
\]

The unary operators (− and not) have higher precedence than the binary operators.

4 Static semantics

The typing rules of LOOP are simple. Variables must be declared before use; their scope runs from their declaration to the end of the block in which they are declared. A variable may only be declared once in a given block, but a variable with the same name may be redeclared in a nested block. Variables and expressions have either integer or boolean type. The equality operator accepts either boolean or integer arguments and returns a boolean result. The types of the operators are as follows:
The typing rules for statements are straightforward too. The conditionals in if and while statements must have boolean type and the two sides of an assignment must have the same type. The argument to a print statement may be either a boolean or integer expression.

5 Dynamic semantics

Execution of LOOP programs follows the obvious imperative semantics. Integer arithmetic is arbitrary precision. The only form of runtime error is a division by zero, which causes the program to terminate with an error message.

6 Example

Here is a LOOP program for computing 5!.

```plaintext
int n = 5;
int i = 0;
int f = 1;
while i < n do
  begin
    i = i + 1;
    f = i * f;
  end;
print f;
```