Description of the “Expressive” language

Erik Stenman

1 The language

This paper describes the language “Expressive”, which is a small language of arithmetic expressions used in the course Compilation A.

1.1 Grammar

```
Prog → Statements
Statements → Statement '.' Statements | ε
Statement → PrintStatement | ReadStatement | LetStatement
          | CondStatement | WhileStatement
PrintStatement → 'print' Expr
               |  'print' String
               |  'printnl'
ReadStatement → 'readint' Identifier | 'readbool' Identifier
LetStatement → Identifier ':=' Expr
CondStatement → 'if' Expr  '{' Statements '}'
              | 'if' Expr  '{' Statements '}' 'else'  '{' Statements '}'
WhileStatement → 'while' Expr '{'  Statements '}'
Expr → Comparison | Arithmetic | BoolExpr
    | '(' Expr ')'
    |  Boolean | Integer | Identifier
Comparison → Expr CompOp Expr
Arithmetic → Expr Op Expr | UnOp Expr
BoolExpr → Expr BoolOp Expr

CompOp → '<' | '<=' | '=' | '>=' | '>' | '!='
Op → '+' | '-' | '*' | '/'  | '%' | '^'
UnOp → '!' | '-'
BoolOp → '|' | '&'
Boolean → 'true' | 'false'
Identifier → Letter (Letter | Digit)*
Integer → Digit Digit*
Letter → [a-zA-Z_]
Digit → [0-9]
String → '"' StringChars '"'
StringChars → [^"]*
```

1.2 White space and comments

The language also allows white space (tab, space, newline) between any token. Comments are like in Java, one-liners starting with "//", or multi line comments from "/*" to "*/".

1.3 Precedence order

The precedence order for expressions is as usual. Be careful with the precedence of the power operator "^", the expression `5 + 2 * 10 ^ 3 - 1` should be interpreted as `(5+(2*(10^3)))-1`, and the expression `-2^5` as `-(2^5)`.

Note that "**and**" has higher precedence than "**or**".

2 Semantics

The language has two types: integers and booleans. Each variable (identifier) will have the type of the right hand side of its first assignment. Any subsequent assignments to the variable must have the same type. (One could argue that the language also has a third type: string, but this type can only be used in print statements.)

Arithmetic expressions correspond to their usual integer arithmetic counterparts, note that "/" is integer division, "reminder, and "a ^ b" is raise "a" to the power of "b". The arguments in an arithmetic expressions has to be integers and the result will always be an integer.

The comparative expressions are also as usual, with "a != b" meaning "a" not equal to "b" and "!a" meaning not "a". The arguments to a binary comparative expressions must be of the same type and the result will always be a boolean.

Boolean expressions can be combined with "|" (or) and "&" (and) to form new boolean expressions, and negated with "!" (not).

The statement `while E { Ss}` will evaluate E and if the result is `true` then the statements Ss will be executed, this will be repeated until the evaluation of E returns `false`.

The stament `if E { Ss1 } else { Ss2 }` will evaluate E and iff the result is `true` then the statements Ss1 will be executed otherwise the statments Ss2 will be executed. The statement `if E { Ss }` will evaluate E and iff the result is `true` then the statements Ss will be executed.

The statement `I := E` will evaluate E and assign the resulting value to the variable I.

The statementss `readint I` and `readbool Id` will prompt the user for an integer or a boolean respectively, and then assign the integer or boolean supplied by the user to the variable I. (If the user types something of the wrong type the system will keep promting the user until he types a type correct constant.)

The stament `print E` will evaluate E and then print the result. The stament `print S` will print the string S. The stament `printnl` will print a newline character.

Sequences of statements (`S1 . S2 . S3 .`) will be executed in order, i.e. first S1 then S2 then S3.

3 Scope

All statements in the program are considered to be in a *block* of code. There is one block of code that covers the whole program from start to finish, which we will call the *global block*. Furthermore, the `if` and the `while` statements introduces new blocks from one '{' to the matching '}'.

A variable `x` is *defined* by the first assignment to that variable, either by `x :=` *Expr*, `readint x`, or `readbool x`. The scope of a variable is from its first definition till the end of the block in which is was defined. That is, a variable which is defined in the global block will be defined till the end of the program, and a variable that is defined inside a if- or while-statement will only be visible within that statement.

4 Types

All expressions in Expressive has a type: it is either an *integer* or a *boolean*. The constants `true` and `false` both have the type boolean. The type of an integer constants is integer. The type of a variable is given by its definition: `readint x` gives x the type integer, `readbool x` gives x the type boolean, and `x :=` *Expr* gives x the type of *Expr*. The type of a *Comparison* is always a boolean, the type of an *Arithmetic* expression is always an integer, and the type of a *BoolExpr* is always a boolean.

The `if` and `while` statements require the contained expressions to be booleans:
`if` *Expr*:**boolean** **{*Statements*}**
`if` ***Expr*:boolean {*Statements*} else {*Statements*}**
`while` ***Expr*:boolean {*Statements*}**

The types of expressions are as follows:

(*Expr*:integer < *Expr*:integer):boolean
(*Expr*:integer <= *Expr*:integer):boolean
(*Expr*:integer > *Expr*:integer):boolean
(*Expr*:integer >= *Expr*:integer):boolean

(*Expr*:integer = *Expr*:integer):boolean
(*Expr*:boolean = *Expr*:boolean):boolean
(*Expr*:integer != *Expr*:integer):boolean
(*Expr*:boolean != *Expr*:boolean):boolean

(*Expr*:boolean | *Expr*:boolean):boolean
(*Expr*:boolean & *Expr*:boolean):boolean
(! *Expr*:boolean):boolean

(*Expr*:integer + *Expr*:integer):integer
(*Expr*:integer - *Expr*:integer):integer
(*Expr*:integer * *Expr*:integer):integer
(*Expr*:integer / *Expr*:integer):integer
(*Expr*:integer % *Expr*:integer):integer
(*Expr*:integer ^ *Expr*:integer):integer
(- *Expr*:integer):integer