G++

G++ is a language being developed for teaching purposes at Gebze Technical University. This language has the following “vision”:

- Lisp like syntax
- Interpreted
- Imperative, non-object oriented
- Static scope, static binding, strongly typed, …
- A few built-in types to promote exact arithmetic for various domains such as computational geometry

G++ Interpreter

- Starting G++ without an input file…

  $ g++

  > _ \\READ-EVAL-PRINT loop starts here…

- Starting G++ with an input file…

  $ g++ myhelloword.g++

  \\READ-EVAL-PRINT everything in the file…

  > _ \\READ-EVAL-PRINT loop starts here…

G++ – Lexical Syntax

- Keywords: *and, or, not, eq, gt, nil, set, defvar, deffun, while, if, load, disp, true, false*
- Operators: *+ - / * ( ) ,*
- Comment: Line or part of the line starting with *;;*
- Terminals:
  - *Keywords*
  - *Operators*
  - *Literals: There is only predefined type in this language.*
    - *Unsigned fractions – two unsigned integers separated by the character "f". E.g., 123f12 is the fraction* $\frac{123}{12}$
  - *Identifier: Any combination of alphabetical characters, digits and "_" with only leading alphabetical characters.*

G++ Lexer Tokens

KW_NIL, DEFV, DEFF, KW_WHILE, KW_IF, KW_EXIT, KW_LOAD, KW_DISP, KW_TRUE, KW_FALSE

OP_PLUS, OP_MINUS, OP_DIV, OP_MULT, OP, CP, OP_SET OP_COMMA, OP_AND, OP_OR, OP_NOT, OP_EQ, OP_GT

COMMENT

VALUEF

ID

G++ – Concrete Syntax

- Non-terminals:
  - $START, $INPUT, $EXPLIST, $EXP, …

G++ – Concrete Syntax

- $START -> $INPUT
- $INPUT -> $FUNCTION | $EXP | $EXPLIST

G++ – Concrete Syntax

- An expression always returns a fraction
- An expression list returns the value of the last expression
- Expressions:
  - $EXP -> OP_OP OP_PLUS $EXP $EXP OP_CP |
    OP_OP OP_MINUS $EXP $EXP OP_CP |
    OP_OP OP_MULT $EXP $EXP OP_CP |
    OP_OP OP_DIV $EXP $EXP OP_CP |
    ID | VALUEF | $FCALL | $ASG
  - $EXPLIST -> OP_OP $EXPLIST $EXP OP_CP

G++ – Syntax

- Assignment:
  - $ASG -> OP OP_SET ID $EXP CP
  - Imperative, therefore $EXP will be evaluated first…

G++ – Syntax

- Functions:
  - Definition:

    $FUNCTION -> OP DEFF ID OP (| ID | ID ID | ID ID ID) CP
    OP $EXPLIST CP

    Extended syntax for four alternatives
  - Call:

    $FCALL -> OP ID (| $EXP| $EXP $EXP | $EXP $EXP $EXP) CP
  - Parameter passing by value (only up to 3 parameters allowed)
  - Returning the value of the last expression
  - *Note that function definition is an expression always returning 0f1*

G++ – Syntax

- Control Statements:
  - $EXP -> (if $EXPB $EXPLISTI $EXPLISTI)
  - $EXP -> (while $EXPB $EXPLISTI)
- Binary values and expressions
  - $EXPB -> (eq $EXP $EXP) : returns true if equal
  - $EXPB -> (gt $EXP $EXP) : returns true if greater
  - $EXPB -> KW_TRUE | KW_FALSE
  - $EXPB -> (and $EXPB $EXPB)
  - $EXPB -> (or $EXPB $EXPB)
  - $EXPB -> (not $EXPB)

For easy writing, '(', 'if', 'while' etc. are used instead of their corresponding tokens

G++ – Variables

- $EXP -> OP DEFV ID $EXP CP // delaring a variable
- $EXP -> OP KW_SET ID $EXP CP // setting a variable
  - Scope:
    - Static, lexical scope (shadowing)
  - Binding:
    - Static binding
  - Typing:
    - Strong typing…

Example Programming in G++

```
$ g++
> (load “helloworld.g++”)
> (sumup 4)
10
> (exit)
$ _
```

```
;; helloworld.g++
(deffun sumup (x)
  (if (eq x 1f1) (1)
    (+ x (sumup (- x 1f1)))
)
```