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 coffee 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, equal, less, nil, list, append, concat, set, deffun, for, if, exit, load, disp, true, false*
- Operators: *+ - / * ( ) ** “ “ ,*
- Comment: Line or part of the line starting with *;;*
- Terminals:
  - *Keywords*
  - *Operators*
  - *Literals: There are two types of values in this language.*
    - *Unsigned integers – any combination of digits with no leading zeros. 0 is considered an instance.*
    - *Unsigned real numbers – two unsigned integers separated by a ‘.’. There could be no leading zeros on the left-hand side. Right hand side has no such limitations.*
  - *Identifier: Any combination of alphabetical characters and digits with no leading digit.*

G++ Lexer Tokens

KW_AND, KW_OR, KW_ NOT, KW_EQUAL, KW_LESS, KW_NIL, KW_LIST, KW_APPEND, KW_CONCAT, KW_SET, KW_DEFFUN, KW_FOR, KW_IF, KW_EXIT, KW_LOAD, KW_DISP, KW_TRUE, KW_FALSE

OP_PLUS, OP_MINUS, OP_DIV, OP_DIV2, OP_MULT, OP_OP, OP_CP, OP_DBLMULT, OP_OC, OP_CC, OP_COMMA

COMMENT

VALUE

IDENTIFIER

G++ – Concrete Syntax

- Non-terminals:
  - START, INPUT, EXPLISTI, EXPI, EXPB, …

G++ – Concrete Syntax

- START -> INPUT
- INPUT -> EXPI | EXPLISTI

G++ – Concrete Syntax

- Lists
  - LISTVALUE -> ‘(VALUES) | ‘() | null
- VALUES -> VALUES IntegerValue | IntegerValue

G++ – Concrete Syntax

- An expression returns either a binary, integer or integer list (prints the corresponding value, e.g. “true”, “123”, “(12,13,14)”)
- Expressions:
  - EXPI -> (+ EXPI EXPI) |
    (- EXPI EXPI) | (* EXPI EXPI) |
    (/ EXPI EXPI) | Id | IntegerValue | (Id EXPLISTI)
  - EXPB -> (and EXPB EXPB) |
    (or EXPB EXPB) | (not EXPB) |
    (equal EXPB EXPB) | (equal EXPI EXPI) | BinaryValue
  - EXPLISTI -> (concat EXPLISTI EXPLISTI) | (append EXPI EXPLISTI) | LISTVALUE | null

G++ – Syntax

- Assignment:
  - EXPI -> (set Id EXPI)
  - Imperative, therefore EXPI will be evaluated first…

G++ – Syntax

- Functions:
  - Definition:
    - EXPI -> (deffun Id IDLIST EXPLISTI)
  - Call:
    - EXPI -> (Id EXPLISTI)
  - Parameter passing by value
  - Returning the value of the last expression
  - *Note that function definition is an expression always returning 0*

G++ – Syntax

- Control Statements:
  - EXPI -> (if EXPB EXPLISTI)
  - EXPI -> (if EXPB EXPLISTI EXPLISTI)
  - EXPI -> (while (EXPB) EXPLISTI)
  - EXPI -> (for (Id EXPI EXPI) EXPLISTI)

G++ – Variables

- EXPI -> (defvar Id EXPI) // defining a variable
- EXPI -> (set Id EXPI) // 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 (equal x 0)
      1
      (+ x (sumup (- x 1)))
  )
)
```