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

1

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…

2

G++ – Lexical Syntax

- Keywords: *and, or, not, equal, less, nil, list, append, concat, set, deffun, for, if, exit, load, print, true, false*
- Operators: *+ - / * ( ) ,*
- Comment: Line or part of the line starting with *;;*
- Terminals:
  - *Keywords*
  - *Operators*
  - *Literals: The following are the only predefined types in this language.*
    - *Unsigned integers.*
    - *Unsigned fractions – two unsigned integers separated by the character ":". E.g., 123:12 is the fraction* $\frac{123}{12}$
  - *Identifier: Any combination of alphabetical characters, digits and "_" with only leading alphabetical characters.*

3

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_MULT, OP_OP, OP_CP, OP_COMMA

COMMENT

VALUEF

IDENTIFIER

4

G++ – Concrete Syntax

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

5

G++ – Concrete Syntax

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

6

G++ – Concrete Syntax

- Lists
  - $LIST -> ‘($VALUES) | ‘() | nil
- $VALUES -> $VALUES OP_COMMA VALUEF | VALUEF

7

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 |
    IDENTIFIER | VALUEF | $FCALL
  - $EXPLIST -> OP_OP $EXPLIST $EXP OP_CP

8

G++ – Syntax

- Assignment:
  - $EXP -> (set IDENTIFIER $EXPLIST)
  - Imperative, therefore, $EXPLIST will be evaluated first…

9

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*

10

G++ – Syntax

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

11

G++ – Variables

- EXPI -> (defvar Id EXPI) // delaring a variable
- EXPI -> (set Id EXPI) // setting a variable
  - Scope:
    - Static, lexical scope (shadowing)
  - Binding:
    - Static binding
  - Typing:
    - Strong typing…

12

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)))
  )
)
```

13