Programming Languages and Translators
COMS W4115

Prof. Stephen Edwards
Summer 2008

VINL
VINL Is Not Logo

Language Proposal

Yang Cao
June 3, 2008

Introduction

VINL is designed as an educational programming language especially targeting younger children. VINL has its root in Logo programming language and is largely a subset of Logo's turtle graph drawing ability with refined syntax.

Why VINL while we already have Logo?

While Logo is feature complete in terms of turtle graphic ability, its syntax is showing age -- it's just not fun to write “repeat / end repeat” all the time. Some dialect of Logo replaces the “repeat / end repeat” sequence with brackets [] but the syntax is not standardized. Furthermore, Logo's syntax is very different from popular programming languages widely in use today. On the other hand, VINL is designed based on popular programming languages such as C/C++/Java syntactically, and we believe it's easier for the young users of VINL to make a leap to C/C++/Java in the future.

Lastly, Logo was well designed and many quality interpreters have been implemented already. We missed out the fun of creating a powerful visual language so let's do it again!

Language Overview

Turtle

The turtle starts in the middle of the screen, and leave a trail when it has been moved. The turtle knows where it is at (the X and Y coordinates) and which direction it is facing (the polar angle in polar coordinate system). It can only move straight forward, but it moves as fast as you program it to!

Data Types

We have the following data types in VINL:

- int
- degree which can be a decimal
- boolean
- void

int type can be used in places where a degree type is required. In such cases, an implicit cast while be done by VINL. Unlike C, boolean and int are fundamental different data types and one can not use those two data types interchangeably.

Functions

Function declaration follows Java syntax closely where and return type is defined first, followed by function name, and then followed by a list of parameters enclosed in {}.

Flow control statements and their syntax

- if -- if (boolean) {...statements...}

the statements in {} will be executed once if and only if the boolean value in () followed by if keyword is evaluated to true. In the case an expression is used inside (), the expression will be evaluated to be a boolean value first then passed to if.

- while – while (boolean) {...statements...}

while is similar to if. The only diffference is the statements in {} will be executed as long as the boolean value maintains to be true.

- for – for(initialization;condition;post-action) {...statements...}

for can be rewrite as a while loop:

```
initialization
while(condition) {
        ...statements...
        post-action
}
```

Operators

VINL supports addition, subtraction, multiplication and division with +.-.*,/. The assignment operator is denoted with ←. Comparison between two expressions are supported by the following operators: <, >, =, and !=. VINL also support logical operator && and ||.

Comments

Comments are denoted with //. Everything follows // on the same line is considered comments. If there are multiple // on the same line, everything follows the first // is considered comments. The subsequent // has no meaning.

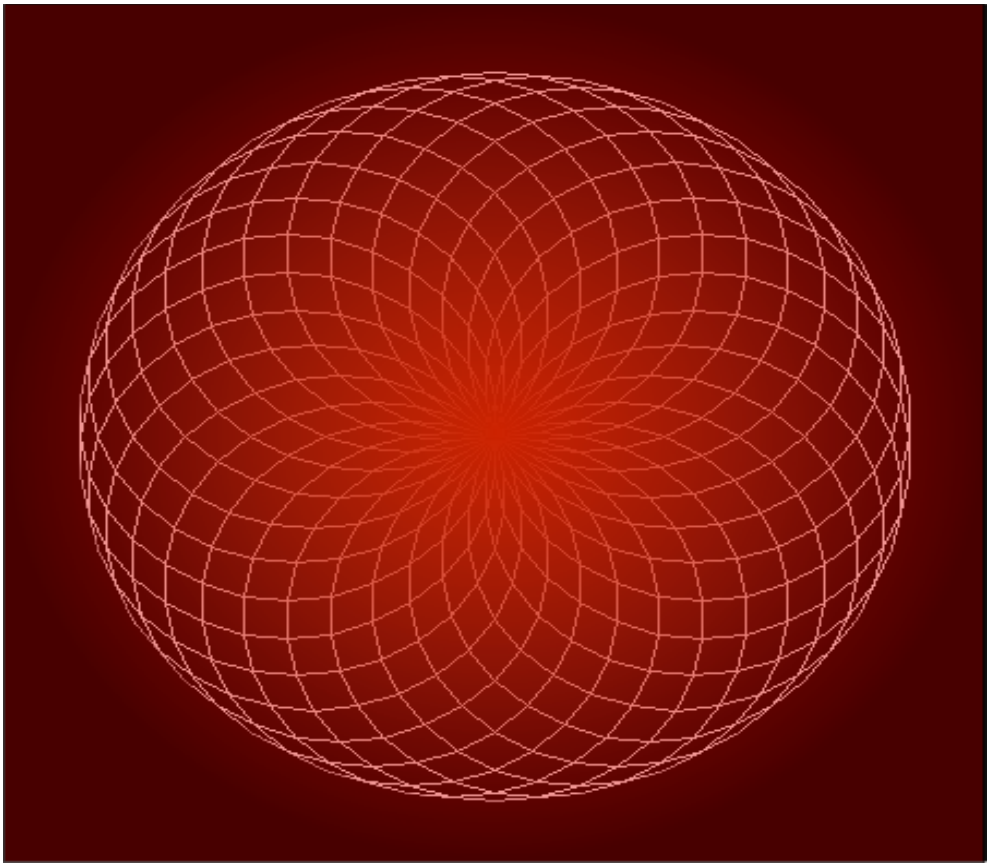
Core library

void advance(int stepsize);
moves the turtle forward with stepsize steps

void turnright(degree angle);
change the turtle's polar angle by certain degree clockwise.

void move(int stepsize);
moves the turtle forward with stepsize steps but does not leave behind a trail.

Code Example



Picture obtained from Wikipedia: http://en.wikipedia.org/wiki/Image:Remi_turtlegrafik.png

```
void drawCircles(int numberOfCircles) {  // function declaration
        int circlesLeft ← numberOfCircles;
        while (circlesLeft > 0) {
                for(int n ← 0; n < 360; n=n+1) {
                        advance(1);
                        turnright(1);
                }
                turnright(360/numberOfCircles);  //implicit cast between int / degree
                circlesLeft ← circlesLeft-1;
        }
}
```

type drawCircle(36); in the interpreter will produce a similar graph on screen.