

# CS 526 Project(s) Draft

## Fall 2001

### 1 Introduction

The projects described here are the default projects for the course. If you have an alternative project that you would like to pursue I will evaluate such proposals. There are three projects described here with varying levels of difficulty. The first is a basic imperative language. The second expands the capabilities of the imperative language. The third project takes a higher level non-imperative approach. All three projects implement languages that are primarily concerned with drawing images. The following descriptions are meant as a starting point for the language development. The language descriptions are incomplete, but should provide enough detail for starting. Either compilers or interpreters will be acceptable.

The intent of the project is to give practical experience with the concepts covered in class. The result will be a significant size program, so do not procrastinate.

### 2 Basic Project

The language has four system variables. The first one is the screen upon which the graphics are drawn. "screen" is a keyword that allows access to the variable. The other three system variables are implicit, input and output streams (for printing and reading) and a cursor object. The cursor object represents where you currently are. (The language is inspired by Logo.) Here are the basic operations:

**forward/backward X** move the cursor X units forward/backwards

**right/left X** change the angle that the cursor is facing

**pen down(up)** subsequent movement of the cursor will (not) draw a line

**pen COLOR** change the color of the pen to COLOR

**fill COLOR** fill in the last polygon traced by the cursor

**clear** remove everything from the screen

**read X** read the current input and put it into X

**print X** send X to the current output

**draw X** use the cursor to draw X

**repeat N { statement }** repeat the statement N times

**switch (N)** basic switch statement

**assignment, boolean operations, math** as in C

**function f (type arg, ...)** basic functions with a return type. Arguments are only input.

The base types for this language are int, real, color, bool, and string. Type-checking should be performed. The sequencing of the statements will be different than traditional C. ;X will indicate that X units of time should occur between the statements. A unit of time should be perceptible to the user and configurable. X will be allowed to be any non negative number. The default value can be set by assigning a value to ;. For example ; = 0 would have actions occur with no time in between. The value of ; (and other variables) will be scoped by the function level.

### 3 Medium Project

The medium project has the same base requirements as the basic project with the following additions.

**struct** structs will contain both base types and other previously defined structs. structs will be reference handles, not pointers. Structs can be dynamically created.

**garbage collection** A garbage collection algorithm will reclaim space used by unused variables.

**multiple files** a mechanism to allow the inclusion of other files.

**optimization** some non-trivial optimizations.

The details of these additions will depend on the specifics of what each team decides. In general the additions will use simple techniques, the point of them will be to provide an opportunity to better understand the issues involved in each.

## 4 Advanced Project

The structure of this language is very different than the other two. At the core it simply allows the definition of an object in terms of other objects. The base types for this language are: line, rectangle, triangle, circle, string, color, coord, int, real, bool, and list.

Graphical objects have a list of the object's properties. The properties can be initialized by the object creating it. A default initialization for the property can also be defined in terms of other properties already defined. The second part of the object description defines how it looks in terms of other graphical objects. An example object might look like:

```
stop_sign ==
properties
  location = [0, 0, 0]
  direction = 0
  size = 3
  top = location + [0, 0, size]
```

```
appearance
  octagon [
    color = red
    location = top
    direction = direction
  ]

  line [
    from = location
    to = top
    size = size / 2
  ]
```

Do not worry about the syntax too much, that is fairly flexible. Here the objects octagon and line are already defined (line is a base type).

In addition to simple values properties can have an existential or universal value. An existential value indicates that the property has a value that meets certain criteria, but as long as it meets the property it doesn't matter what it is. For example, the color property for the octagon could instead be `color = { dullred, brightred, purple }`. This would indicate that the system could choose any one of those colors for the octagon.

The universal value would essentially be a list of values, and an instance would be created for each of them. For instance, if we had wanted three stop signs `location = loc1, loc2, loc3 >` could be used. If we had assigned `direction = < 0, 180 >` then we would have had a total of 6 stop signs, one for each combination.

The existential and universal values are either hard coded in or created through generators of the form `< operator : range : boolean expression : value >`. Operator will be set constructor (for existential), list constructor (for universal), addition, and multiplication. Range will either be an integer range or set/list membership. An example, `< listcons : 0 < x < 359 : x mod 180 == 0 : x >` would create the list `< 0, 180 >`.

There are two system variables, input and output. read and print functions should also be supported. There is an implicit loop that keeps re-evaluating the top level graphical object named "screen". Between each evaluation of "screen" the graphics window is emptied.

Additional aspects of this language can be decided upon by the team's preferences.

## 5 Implementation

Teams of up to three people are allowed. You may work in any language that you want (so long as it does not trivialize the problem). The projects will be tested under Linux. Assume a configuration similar to the machines in the lab, I can install additional packages though. It should be possible to check out the team's sources, compile, and run.

## 6 Grading

You must use the CVS server provided for configuration management. We will cover CVS basics in class. The final product only will not be accepted, each team must check in their work every day or every four hours of work, whichever occurs first. To obtain the most partial credit (in the event that the project is not fully correct) different parts should be separately testable. It is recommended that you develop the tests at the same time that you are writing the code. I am flexible about the language the project implements so long as it does an adequate job of covering the topics covered.

The deadlines for the projects are hard. Whatever is in the CVS repository as of the deadline will be the team's submission. Once a team has chosen a project it may not choose a more advanced project, it may choose an easier one though.

## 7 What's next

The first step is to read the different project descriptions and ask questions in class. The next step is to choose a team and project. The team should then flesh out the language that will be developed. Try creating a grammar for the language. Identify what the run time system will need to do. Make sure the team agrees about the different aspects of the language. Email project proposals to me by Friday Sept. 7th. If the proposal is not based on one of these projects indicate what how advanced the project is. Include a grammar in the project description.