CSCI3180 – Principles of Programming Languages – Spring 2015

Course Project — A Simple LOGO Interpreter

Phase 3 deadline: April 12, 2015 (Sunday) 23:59

1 Introduction

LOGO is an educational programming language that allows programmers to move a virtual "turtle" with a pen attached to its tail around the screen to draw graphics. In this project, you will be asked to design and implement an interpreter for a subset of the LOGO language in Objective-C. Your LOGO interpreter has to be able to process simple LOGO programs and draw the resulting graphics.

You will do the project in three phases:

1. A short exercise to get familiar with Objective-C and create an object-oriented design for your LOGO interpreter. (4 weeks)
2. Implement your interpreter without support for graphics. (3 weeks)
3. Extend the LOGO language and add graphics support. (3 weeks)

2 Phase 3 - LOGO Enhancement

In this phase you are going to extend the LOGO interpreter you have implemented in the previous phases. This phase can be splitted into 2 parts. In this first part, you are required to extend your LOGO to support variables. In the second part, you have to build a LOGO program development environment with graphic user interface.

2.1 Introduction of variables

In this phase, the enhanced LOGO allows users to declare, initialize, set and use integer variables. It supports **integer** addition and substraction. In Phase 2, LOGO only supports REPEAT loops. In this phase, you need to enhance your LOGO interpreter to support also WHILE loops. The following 3 tables describe the enhanced features. In Table 2, $source_1$ and $source_2$ can be a constant integer or a variable while *destvar* must be a variable. In Table 3, i and j can be an constant integer or a variable.

Table 1: Commands for Initialization

MAKE *var*	Create a new variable called *var*.
SET var_1 value	Assign value to variable var_1.
SET var_1 var_2	Assign value of variable var_2 to variable var_1.

Table 2: Commands for Arithmetic

ADD *destvar* $source_1$ $source_2$	The value of $source_1$ is added to the value of $source_2$ ($source_1+source_2$) and the result is stored in *destvar*
SUB *destvar* $source_1$ $source_2$	The value of $source_1$ is substracted by the value of $source_2$ ($source_1 - source_2$) and the result is stored in *destvar*

Apart from the above tables, please note that commands FD, BK, LT, RT in the enhanced LOGO accept variables as their arguments too. The following examples have their corresponding output drawings in Figure 1.

Table 3: Commands for WHILE Loop

WHILE [EQ i j] [..]	While the value of i is equal to the value of j, execute the block
WHILE [GT i j] [..]	While the value of i is greater than the value of j, execute the block
WHILE [ST i j] [..]	While the value of i is smaller than the value of j, execute the block

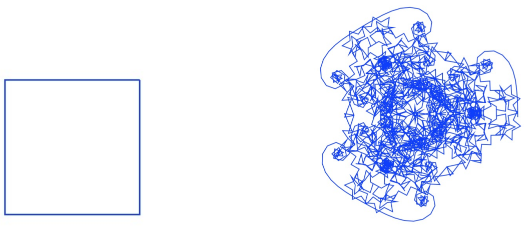

(a) Example 1 (b) Example 2

Figure 1: Outputs of the examples with variables

- Example 1: Drawing a square

```
MAKE INDEX
MAKE COUNT
CS
PD
SET INDEX 0
SET COUNT 4
ADD INDEX INDEX 90
WHILE [GT COUNT 0]
[ LT INDEX FD INDEX SUB COUNT COUNT 1 ]
```

- Example 2: Drawing a complex pattern

```
CS
PD
MAKE INDEX
MAKE COUNT
SET INDEX 1
SET COUNT 0
WHILE [ST COUNT 2000]
[ LT INDEX FD 12 ADD INDEX INDEX COUNT ADD COUNT COUNT 1 ]
```

You can make the following assumptions:

1. Variables must be declared and created before initialization.
2. Variables must be initialized before any use.
3. A declared variable must be available for storing the result of arithmetic operations.
4. Variable names must be words defined in Phase 1.
5. Our test cases for testing are free of errors.

2.2 MVC user interface

In this project, you have to use the Model-View-Controller (MVC) paradigm to implement the user interface. The essential idea of MVC is to divide a software application into three interconnected parts, so as to separate internal representation of information from the ways the information is presented to the user. The *model* is responsible for keeping an abstract representation of the state of the object. The *controller* is responsible for how your model is represented on screen. The *view* is the generic UI elements that *controller* could manipulate. MVC is central to a good design for a Cocoa application. A *controller* object acts as an intermediary between one or more of an applications *view* objects and one or more of its *model* objects. Your *view* objects and *model* objects should be independent to each other.

To facilitate your implementation, we provide you with a user interface template. This template is based on the sample design we gave in Phase 2. Most of the UI elements are configured in the provided `MainMenu.xib`, and some are configured in the code. You are **not allowed** to modify those configurations. There are basically three view components in the window as shown in Figure 3. LOGO code is typed in the text sub-view, whose size is 470×100. When the "Execute" button is clicked, the pen will move and its path is shown in the 600×400 drawing sub-view. You do not need to animate the moving process of the pen. You should initialize the pen to be in the center of drawing sub-view, facing north and lower down. And the color of the pen (path) should be blue. The pen moves forward or backward 1 unit in the drawing sub-view's coordinate system at each step. For example, when the pen is at (10, 0) according to the drawing sub-view's coordinate system and facing north, after "FD 15", the pen's position should be (10,15).

In the template, some classes are already implemented for you. Some are interfaces that you need to implement. Since our template is based on the provided sample design, you are **allowed** to modify the types of properties, arguments and returned values of the methods in the template. But you are **not allowed** to modify the class structures, property names or method names. And you **should** implement the classes according to following instructions. We illustrate each component according to the provided sample design below.

Class **LogoModel**
This class is a subclass of **NSObject**. You should implement this class with the following components.

1. Properties

   (a) `sourceCode;`
   This property keeps the the LOGO code object. According to the sample design, it is a `SourceCode` object that knows how to tokenize itself.

2. Instance Methods

   (a) `setSourceCodeWithString`
   In this method, `sourceCode` is initialized with an argument that specifies the input LOGO code.

   (b) `executeLOGO`
   This method executes the interpreting process. According to the sample design, firstly, a message is sent to `sourceCode` so that `sourceCode` tokenizes itself and returns the result, which is a `TokenList` object. Then, a message is sent to the returned `TokenList` object so that it parses itself and return a `SyntaxTreeNode` object, which is the root node of the tree after parsing. In the last step, the root node evaluates itself and returns a path, which is a `NSBezierPath` object.

Class **LogoDrawingView**
This class is a subclass of **NSView**. It implements the drawing sub-view in the user interface. You should implement this class with the following components.

1. Properties

   (a) `path`
   This is the path that needs to be displayed in the view. It is a `NSBezierPath` object.

2. Instance Methods

   (a) `awakeFromNib`
   This method has already been implemented for you. Configurations for this view is put in this method.

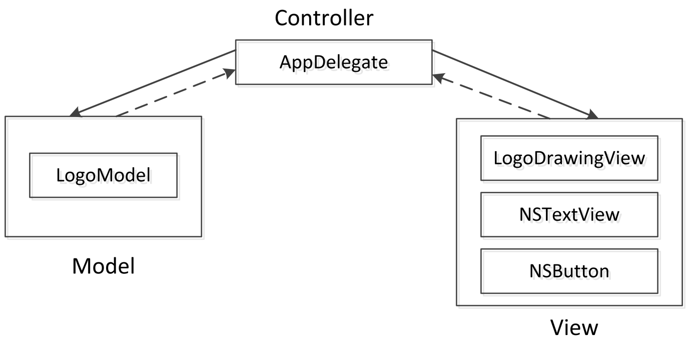


Figure 2: Model-View-Controller diagram

(b) `drawRect`
The `path` is drawn in this method.

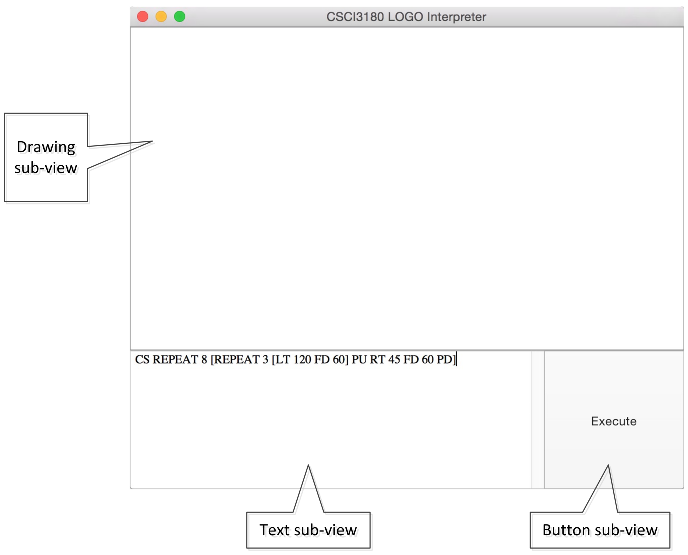
Class **AppDelegate**
This is the delegate of `NSApplication`, which is the controller of `MainMenu.xib`. And `MainMenu.xib` is the view component that is generated automatically by Xcode for you. You could take `AppDelegate` as the view controller in your MVC. You should implement this class with the following components.

1. Properties
   (a) `logoDrawingView`
   This is the view in which pathes are drawn. It is a `LogoDrawingView` object.
   (b) `programTextView`
   This property is a UI component corresponding to the text sub-view in the user interface. LOGO codes are input in this sub-view.
   (c) `logoModel`
   This property holds an object of `LOGOModel`.
2. Instance Methods
   (a) `executeButtonClicked`
   This method is created by control-drag the “Execute” button to the implementation area in ”AppDelegate.m“. This method handles the click event that acts on the “Execute” button. First, it fetches the input code from `programTextView`. Then, a message is sent to `logoModel` so that `executeLOGO` is called in the model. A `NSBezierPath` is returned and is used to update `path` in `logoDrawingView`. This method has already been implemented for you.
   (b) `logoModel`
   This is the default `get` method for the property `logoModel`. You should initialize the property `logoModel` in this method. This is called *lazy initialization*, which means that you initialize something just before you really need to use it.

The MVC diagram of our design is shown in Figure 2. You should develop an MVC user interface as shown in Figure 3. By clicking the “Execute” button, the finished MVC user interface should be able to execute the LOGO program in the text sub-view and draw the proper graphics in the drawing sub-view. You can use the sample code provided by us to check the correctness of your program.

3 Submission Guidelines

Please read the guidelines CAREFULLY. If you fail to meet the deadline because of submission problem on your side, marks will still be deducted. So please start your work early!



CS REPEAT 8 [REPEAT 3 [LT 120 FD 60] PU RT 45 FD 60 PD]

Figure 3: Sample screen output

1. In the following, **SUPPOSE**

   your group ID is *01*,
   for group member 1: your name is *Chan Tai Man*, student ID is *1155234567*, username is *tmchan*, and email address is *tmchan@cse.cuhk.edu.hk*,
   for group member 2: your name is *Lee Man*, student ID is *117654321*, username is *mlee*, and email address is *mlee@cse.cuhk.edu.hk*.

2. In your source files, insert the following header. REMEMBER to insert the header according to the comment syntaxes of Objective-C.

```
/*
 * CSCI3180 Principles of Programming Languages
 *
 * --- Declaration ---
 *
 * I declare that the assignment here submitted is original except for source
 * material explicitly acknowledged.  I also acknowledge that I am aware of
 * University policy and regulations on honesty in academic work, and of the
 * disciplinary guidelines and procedures applicable to breaches of such policy
 * and regulations, as contained in the website
 * http://www.cuhk.edu.hk/policy/academichonesty/
 *
 * Phase 3
 * Name : Chan Tai Man
 * Student ID : 1155234567
 * Email Addr : tmchan@cse.cuhk.edu.hk
 *
 * --- Declaration ---
 *
 * I declare that the assignment here submitted is original except for source
 * material explicitly acknowledged.  I also acknowledge that I am aware of
 * University policy and regulations on honesty in academic work, and of the
 * disciplinary guidelines and procedures applicable to breaches of such policy
 * and regulations, as contained in the website
 * http://www.cuhk.edu.hk/policy/academichonesty/
 *
 * Phase 3
 * Name : Lee Man
 * Student ID : 117654321
* Email Addr : mlee@cse.cuhk.edu.hk
*/
```

The sample file header is available at

```
http://www.cse.cuhk.edu.hk/~csci3180/resource/header.txt
```

3. In Phase 3, you are required to submit your codes and a report. In the report, you should specify whether you are using the sample design provided by us or sticking to the one you submitted in Phase 1 or Phase 2. In addition, if you use your design with any modifications, you are required to submit your modified design also.

4. Your codes should be included in a Xcode project. The project name should be "LOGOInterpreter". You should submit the whole project folder. The report should have the filename "report.pdf". All file naming should be followed strictly and without the quotes.

5. `Tar` your source files to `groupID.tar` by

   ```
   tar cvf group01.tar report.pdf LOGOInterpreter
   ```

6. `Gzip` the `tarred` file to `groupID.tar.gz` by

   ```
   gzip group01.tar
   ```

7. `Uuencode` the `gzipped` file and send it to the course account with the email title "HW1 *studentID yourName*" by

   ```
   uuencode group01.tar.gz group01.tar.gz \
   | mailx -s "Project Phase 3 Group01" csci3180@cse.cuhk.edu.hk
   ```

8. Please submit your assignment using your Unix accounts.

9. An acknowledgement email will be sent to you if your assignment is received. **DO NOT** delete or modify the acknowledgement email. You should contact your TAs for help if you do not receive the acknowledgement email within 5 minutes after your submission. **DO NOT** re-submit just because you do not receive the acknowledgement email.

10. You can check your submission status at

    `http://www.cse.cuhk.edu.hk/~csci3180/submit/hw1.html`.

11. You can re-submit your assignment, but we will only grade the latest submission.

12. Enjoy your work :>