

MM 319 Lab Two: Control Structures and Basic I-O

Introduction

This activity will help you understand simple selection (if) and iteration (repeat) using Lingo. You will also create a simple program that collects user input and outputs a result.

Syntax of the IF Structure

The if structure gives us the ability to choose between alternate courses of action based on some condition or conditions. The fundamental syntax is:

```
if condition then  
    statement(s)  
else  
    statement(s)  
endif
```

The else clause is optional, as it is in most programming languages. However, the keywords **then** and **endif** are mandatory, as is the indentation. The *condition* is any expression that evaluates to a Boolean value, and the *statements* are any legal Lingo statements. You will notice that with the exception of the words **then** and **endif**, this is just like any other language you've dealt with.

Let's try a few if structures in the message window. Launch Director® and create a new file. Open the message window. **Important:** to test a multiple-line statement in the message window, you must shift-return at the end of each line until you have entered to entire structure. Director attempts to evaluate statements in the message window when the return (enter) key is pressed alone. For the following group of statements, you need to use shift-return after the word "then."

```
x = 20  
if x < 30 then  
    put "yes"
```

Now try these:

```
y = 9  
if y > 5 then put "y is more"  
if y < 5 then put "less" else put "more"
```

As you can see, there are some apparent inconsistencies in what Lingo allows. The best advice is avoid adopting irregular structures.

By now you are probably getting annoyed with the message window as a testing tool. It's time to move script to a friendlier environment.

Creating a Message Handler in a Script

We can write scripts for many of the objects in Director: sprites, cast members, individual frames, and even the movie itself. Examining where to place instructions for different types of actions will be the topic of later work, as will discussion of the different types of events that scripts can handle. For this activity, we are going to deal with a single event being handled in the script for a single sprite.

Insert a button cast member to your movie. Use the **Insert** menu to do this, because the button tool on the tool bar will insert a Flash button that may not behave as we need it to.

MM 319 Lab Two

Add two text members, side by side. We are going to use one as an input field and the other as a label for it. In the “label” field, type “Enter the name of a month.” Name the other field “monthIn,” using the property inspector or by naming it in the cast window **and** in the text property inspector click the “editable” box. We want the user to be able to type in this field.

You should now have three objects on the stage: a button, a label, and an empty field for input. (You may want to modify the stage color in the movie properties so the input field is easy to see). Arrange the three objects in any way that makes sense, then save your movie (you know how cranky this program can be!)

We’re going to make this a program that reports how many days are in a given month. Not very exciting, but good enough to make a point or two.

Edit the script for the button’s sprite by selecting the sprite either on the stage or in the score, and Control-clicking it, and then choosing **Script** from the menu that pops up. The script window will open, and you’ll see the lines

```
on mouseUp me

end
```

The cursor will be blinking between the two lines. This is where you’ll put the instructions for what to do when the button receives the mouseUp message—in other words, when someone clicks the button. Add two lines so that the handler looks like this:

```
on mouseUp me
    monthName = member("monthIn").text

    put monthName
end
```

This first assigns the contents of the text member named **monthIn** to the variable **monthName**, and then puts the value of the variable in the message window.

Near the top of the script window is a tool bar. Near the right end of the tool bar is an icon shaped like a lightning bolt. This recompiles all the scripts—click it to register the changes you made, and then return to the stage, save, rewind, and run your movie. Type anything in the entry field and click the button. Now go to the message window, and you should see the value you typed in the field.

Now edit the script again so it looks like this: (Sorry it’s so tiny, and note that there should not be a line break after July!)

```
on mouseUp me
    monthName = member("monthIn").text

    dayCount = 0

    if monthName = "January" or monthName = "March" or monthName = "May" or monthName = "July"
        or monthName = "August" or monthName = "October" or monthName = "December" then
        dayCount = 31
    else
        if monthName = "April" or monthName = "June" or monthName = "September" or monthName = "November" then
            dayCount = 30
```

MM 319 Lab Two

```
else
  dayCount = 28
end if
end if

put dayCount
end
```

Recompile the script. Don't be surprised if you have to fix a few typos before it's happy. Then save, and run again to test. Put a month name in the input field, then click the button and see if the last value in the message window is the right number. If you are seeing a word instead of a number, you overlooked the last line in the revised script above.

To finish this program we need the output to show up somewhere the user can see. Add another text member, and name it "Result." Change the last line in the handler from "put dayCount" to:

```
member("Result").text = string(dayCount)
```

Recompile the script and test your results.

Exploration

You may have wondered about some of the details in the statements. Was it really necessary to write out all those pieces of the **if** structure? Couldn't we just say **if monthName = "April" or "June" or...**and so on? Try it and see what happens—and then try to find an explanation.

Another item you may have wondered about was the final change. Why use the word **string** in this way? Once again, try removing it to see what happens.

The REPEAT Structure

Lingo uses the keyword **repeat** to control iteration. Repeat can be used for counter-controlled iteration (what some call *definite loops*) and also for sentinel-controlled iteration (*indefinite loops*). Director Help will show you the syntax for the two forms. If they don't make sense to you, raise a question in class!