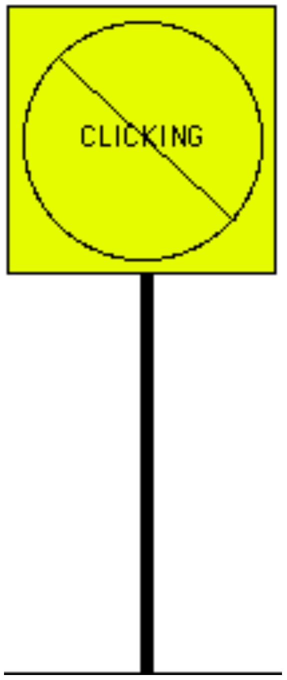
# CSCI 051G Practice Laboratory # 1
## An Introduction to Grace and the ObjectDraw Library

**Objective:** To demonstrate the use of the lab computers, Grace and graphics primitives

—

This lab will introduce you to the tools with which you will be working this semester. Your task is fairly simple. You will construct a Grace program that draws an image resembling a roadside warning sign, but with a message more appropriate for a computer screen, as shown below.



Then you will modify your Grace program so that it modifies the picture in simple ways in response to actions the user performs with the mouse.

## Part 1. Logging in & setting up your account

To get started, you must log in to one of the iMac computers in our lab.

Begin by entering the user name and password that you were provided for your computer science account in the login window and click "Log in". Once you log in, the screen should display a window containing icons for some of the files you can access on the machine.

While the computer is booting, you may see a window pop up asking you to log in to iCloud. Please just ignore this. You can click in the red dot in the upper left corner of the window to make it go away. Hopefully you will never see this again.

Before proceeding, you should change the password we assigned to your account to something that only you will know and that will be easier for you to remember. Don't, of course, make your new password too easy to guess. In particular, at least use something that is a mix of letters and numbers. However, do make sure you memorize it or write it down in some secure spot. You will need to use it every time you use the CS lab.

To change your password you should:

- Click on the fox-and-globe icon at the bottom of your screen. That will bring up the FireFox web browser.
- Go to http://www.dci.pomona.edu.
- On the left margin of the resulting page, select the item "change password".
- Type in your user name and current password, and then a new password (twice).
- Click "submit".

Now, log out and log in again with your new password. To log out select the last item, "Log Out" in the menu with the apple icon near the upper left corner of your screen. Click OK when it asks if you are sure.

The files you create for our course laboratory projects will not actually be stored in the computer on which you are working. Instead, they will be stored on our department's file server. By entering the account information for your account, you have instructed the computer you are using to access your files on that server. You can access your files from any computer in our lab by simply logging in to that computer.

Next observe that there is already a folder labeled "CS51Workspace" on the desktop for your cs51 work. This is where you will do all of your work for this class.

To work on the lab, you will need to move some files from the course folder (named "cs051", with a small red arrow in the upper corner) to your CS51Workspace folder.

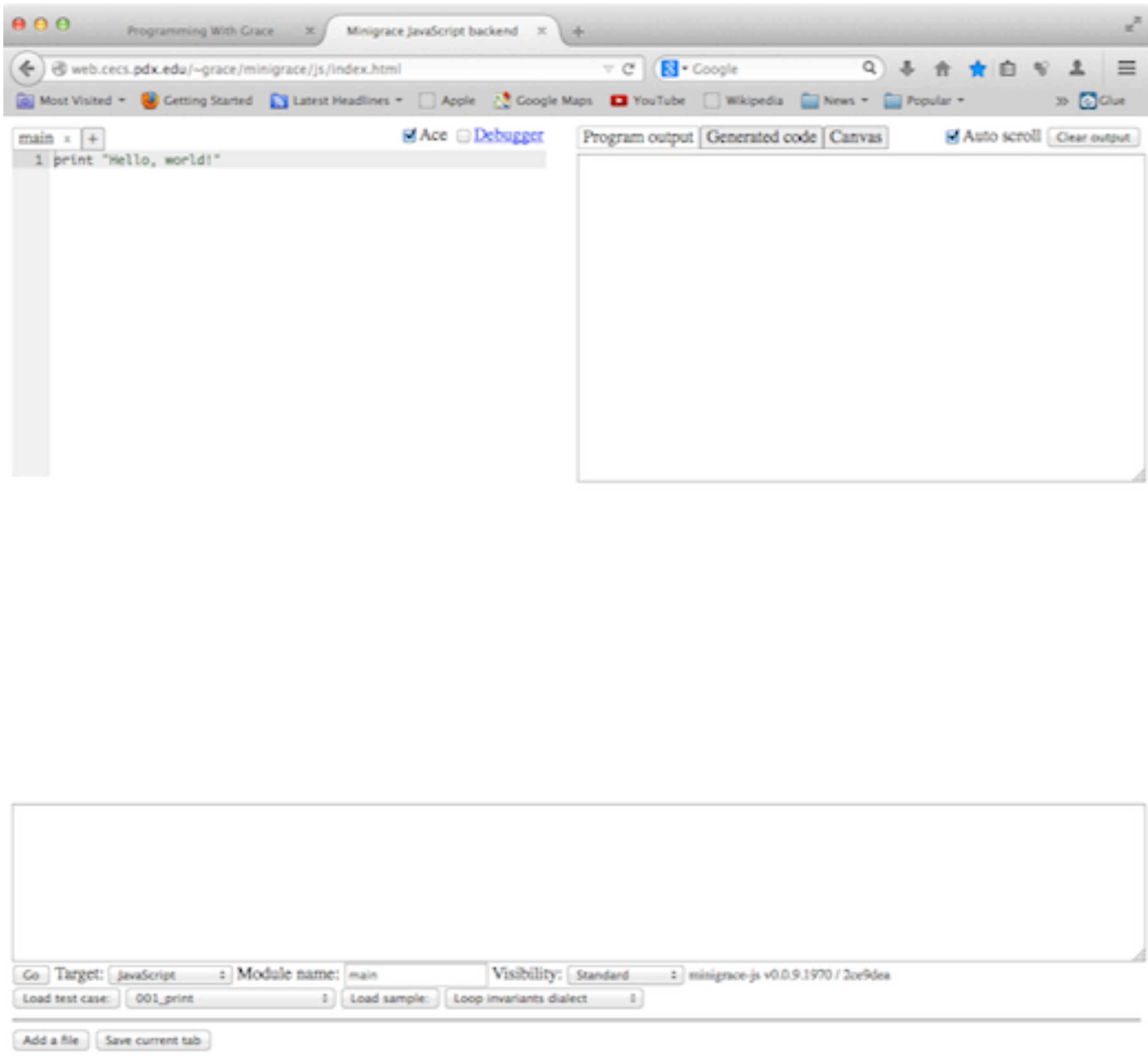
1. Double click on the cs051 folder to open it. Double click on the "labs" folder.
2. Drag the "Lab1-Sign" folder to the CS51Workspace folder on your desktop
3. Close the "labs" folder by clicking in the red circle in the upper left corner of that window.

## 1 Using the MiniGrace compiler

We are going to be using a new prototype compiler for the Grace language called minigrace (because it doesn't yet do all the things we want in Grace). A compiler is a piece of software that takes a program written by a programmer and converts it to a form that can be executed by the computer. In our case, the program is going to be executed in a web browser, so the compiler will convert your Grace program to a much more primitive program in the language Javascript, the language that web browsers understand.

To get started, to to the minigrace web page at http://web.cecs.pdx.edu/~grace/minigrace/js/.

The web browser will display a window on your screen that should look like this:



```
print "Hello, world!"
```

Ace: ✓ Debugger: ☐ Auto scroll: ✓

Target: JavaScript Module name: main Visibility: Standard minigrace-js v0.0.9.1970 / 2ce9dea

Load test case: 001_print Load sample: Loop invariants dialect

The components in this window are divided into four main units.

**The Program Inspector/Editor Window** The large rectangle appearing on the upper left part of the window is the area in which the your program will appear when you load it.

**Program Output** In the upper right portion of the window is an area where program output can occur.

**Compiler Control & Output** The wide box below the Editor and Output panels is where messages from the compiler will appear. They will let you know if your program has compiled correctly or if there are errors that must be corrected.

Just below that space there is a "Go" button that will start the compiler on whatever program is showing in the Editor window. If the compiler succeeds in translating your program then it will begin execution, possibly resulting in information being written in the Output panel or in a window popping up with graphics.

There are several other buttons there, but they are not important for your work.

**The file loading controls** Below the horizontal line (which is itself below the "Go" button) there are a series of buttons that you can use to load your program for execution. The two buttons showing initially are labelled "Add a file" and "Save current tab". You will use the "Add a file" button to initiate loading programs into the browser.

Meanwhile, the first line of the Program window should be showing the one line program:

```
print "Hello, world!"
```

Press the "Go" button toward the bottom of the window and you will see the output `Hello, world!` appear in the Program output window. Congratulations! You have just run your first Grace program.

To make it even more exciting, go to the Program window and change "world" to your name. Make sure the double quotes stay around the whole phrase to be printed.

## Part 2: Creating Graphical Programs

Now let's begin to create your program for today. Almost every program you write will use the objectdrawDialect. This is a special dialect of Grace that provides you with easy-to-use graphics primitives and allows your program to recognize mouse actions.

Click on "Add a file". A button "Browse" will show up right next to where the "Add a file" button was. Click on "Browse" and navigate to the file objectdrawDialect.grace in your "CS51 Workspace" folder. Select it and click on "open".

The filename "objectdrawDialect" should now show up right next to the "Browse: objectdrawDialect" button. Click on the "Go" button above it to compile the file. This will likely take a minute or more depending on your computer. When it is done there will be a series of messages that start with "minigrace" that appear in the Compiler output panel. Luckily you only have to do this once for each lab session.

Click "Add a file" one more time. Then "Browse" and select "NoClicking.grace" from your "Lab1-Sign" folder. Make sure this is the file showing in the Program window (click on a tab on top that says "NoClicking" to select it if its code is not showing), and then select "Go" again. After a much shorter pause than last time, the messages will appear again in the Compiler output panel, but this time a 500 by 400 pixel window will also appear. Right now nothing will be in the window that popped up, but you will fix that over the course of this lab. Close this window by clicking in the red box in the upper left hand corner of the window. You are now ready to start working on your program.

The text of the program as it should initially appear in your Program window is shown in the figure on the next page. The text we have placed in "NoClicking.grace" is the skeleton of a complete Grace program. It includes an object expression whose first line is `inherits graphicApplication.size(500,400)`. The combination of this line and the last one: `startGraphics` ensures that when the program is executed

```
dialect "objectdrawDialect"

//   CS 51 Laboratory 1
//   Enter your name, lab section and the date here.

object {
   inherits graphicApplication.size(500,400)

   method onMousePress(mousePoint) {
     // commands placed here are executed after the user clicks the mouse
   }

   method onMouseEnter(mousePoint) {
     // commands placed here are executed when the mouse enters the program window
   }

   method onMouseExit(mousePoint) {
     // commands placed here are executed when the mouse leaves the program window
   }

   method onMouseRelease(mousePoint) {
     // commands placed here are executed when the mouse button is released
   }

   // pop up window and wait for mouse actions
   startGraphics
}
```

Figure 1: The contents of file NoClicking.grace

it will pop up a window that is 500 by 400 pixels and that when there are mouse actions, the system will respond by executing the appropriate mouse event handling methods in the program.

Inside the object expression are headers for the event handling methods you will use. We have not, however, included any Grace commands within the bodies of these methods, only a Grace comment that reminds you when the Grace system will follow any instructions you might add to the method body.

You should follow our lead and begin writing your program by typing comments rather than actual Grace commands. Near the top of the file we have included a temporary comment telling you to enter your name, lab section and the current date. Such identifying comments are absolutely necessary so the the grader can give you the appropriate grade. When they print out your program to grade it, they do not see the name of the file or any folder it might be in. Hence it is crucial that you place your name as a comment in every file you turn in.

Rather than editing your program in the web browser you will use an application Aquamacs, which you can find in the dock at the bottom of the screen. Select the application and then when it opens, click on the "Open" icon at the top of the window, or select "Open File..." from the File menu in the top left corner of the screen. Navigate again to the file "NoClicking.grace" and open the file in Aquamacs.

While you can edit your program in the browser, you will not be able to easily save it where you want it. As a result you will have a much better experience if you do all the editing in Aquamacs. Your first task will be to add Grace instructions to the object expression to draw a "no clicking" warning sign. The code for this can go immediately after the **inherits** line.

As a first step, let's add the single instruction needed to draw the rectangle that frames the contents of the sign. The form of the command you will need to enter is:

```
framedRect.at(pt) size (w,h) on (canvas)
```

where pt should be replaced by an argument of the form `x@y`, where `x` and `y` are numbers and (x,y) is the location of the upper left corner of the rectangle. Similarly `w` should be replaced by the width of the rectangle, and `h` by the height. (`canvas` stays as it is.)

We suggest that you place the rectangle so it will be centered in the upper portion of the window (see the picture at the beginning of this document). A width and height of about 100 each should look good.

When you are done adding this line to your program in Aquamacs, save your program by clicking on the "Save" icon at the top of the window. Now go back to FireFox and click on the "reload" button next to the file name `NoClicking.grace`. Then click on the "Go" button. This time when the window comes up, it should show a framed rectangle where you told it to create it.

Of course you might have made a mistake when you typed the program and you might have gotten an error message in the compiler output window. If so, you should read the error message and correct the problem. The error message should give you a line number and explain what the problem was. Hopefully it will point out exactly the error you made, but sometimes it gets confused and will point at somewhere different than where the error actually occurred. Try correcting your error and then "Reload" and "Go" again. If you can't see what your error is, call over a TA or the instructor for help.

Of course you may not be happy with where your framed rectangle appears on the screen. If so, modify the parameters to the construction and "reload" and "Go" until you are happy with it.

Each time you make a change to your program you must be sure to:

1. Save the program in Aquamacs
2. Reload the program in the web browser
3. Click on "Go" to run the program

If you forget any one of these steps, your newly revised program will not be executed, making you very frustrated until you figure out that you forgot one or more of the steps.

Also, don't forget to close the popup window showing the graphics from your program before you run it again. Otherwise, the new version of the graphic output will be added below the previous output, but you will not be able to see it without scrolling down.

Now that it does something, it would be a good idea to place a comment on the line before the construction to say what it does so far. A comment starts with //. Anything after those two characters to the end of the line will be ignored by the Grace compiler (but not human readers!), so write whatever is helpful to a reader to understand your code. You should get in the habit of adding and updating comments to keep them as accurate as possible as you add instructions to your programs. Not only does this mean you will not have to go back and add comments when you're done, it will help you to remember what everything does as you are writing your programs.

If you are working on the iMacs in our lab and have not used Macs before, there really is not much difference between it and Windows (and Linux) any more. Cutting and pasting on the Mac is similar to that with Windows except you use the command or "fan" key, to the left of the spacebar, in combination with "x" (for cut), "c" (for copy), and "v" (for paste). You can erase text by selecting it and then hitting the backspace key.

Once your rectangle program is working, you should add more instructions to it to turn it into a complete warning sign drawing program. Back in Aquamacs, immediately beneath the line you added to draw the rectangle, add additional lines to create `text.at(pt) with (w,h) on (canvas)`, a `framedOval.at (pt) size (w,h.) on (canvas)`, `line.from(start)to(end) on (canvas)` and all the other components. You will need to experiment to find reasonable coordinates and dimensions for the various objects. By the way, in our drawing, the post is a rectangle, while the base is a line.

In the picture, we show a yellow background for the sign. Leave that part out of the version your program draws for now; we will add it later. As you work, it is a good idea to "Run" your program every time you add a line or two to ensure that you catch mistakes early. Also, make sure your comment lines get updated by the time you are finished. When you are done, your program should draw a sign when it is executed.

## Making Your Program Responsive

Now, to explore event handling methods a bit more, revise your program so that it reacts to the mouse in more interesting ways. In the revised version, the sign will change as the program runs. In particular, when the user moves the mouse into the program window or presses the mouse, your revised code will alter the appearance of the sign.

Making the sign change in response to the mouse is a bit tricky. Suppose, for example, that you want to emphasize the sign's warning by changing the color of the word "CLICKING" from black to red when the user moves the mouse into the program's window. There is a method `color:=` that you can use to change the color of the text. There is also an obvious place to tell Grace to make this change. The `onMouseEnter` method is executed whenever the mouse moves into your program window. Placing an appropriate `color:=` in that method would do the trick.

The problem is that you can't simply say `color:=` in the `onMouseEnter` method. If that was all you said, Grace would have no way of knowing which of the several objects' color to change. It could change the rectangle, the oval, the text, all of them, etc. Your code has to be more specific and identify the object that should change.

To be able to specify the text object as the object whose color we wish to set, we will have to give it a name. We will use the name `message`, but we could use any name that seems appropriate.

Associating a name with an object requires that we add a line that introduces the variable and that provides an initial value. We plan to associate the name message with the object created with text, so we have to tell Grace that the name will be associated with that object. The form of a Grace declaration is simply the to write the keyword **def** followed by the name being declared followed with = and the value it will be associated with. So, the form for our declaration is:

```
    def message = text (...)  size  (...,...)  on (canvas)
```

where the code to the right of the = is exactly what you had written originally to create the text. As a result, all you must do now is to add the **def** message = before the existing code.

Now that the text has a name, we can use the name to change its color. Within the body of the onMouseEnter method add the line:

```
  message.color := red
```

Then, run your program (correcting any errors as needed).

The program isn't quite complete. It draws the sign immediately and makes the text turn red when the mouse enters the window, but it doesn't make the text black again when the mouse is moved back out of the window. You should be able to figure out what to add to make it black when the mouse exits. Give it a try.

To get more practice using names and other event handling methods, we would like you to modify your program a bit more. First, change the program so that while the user is depressing the mouse button, the circle with the diagonal line through the text disappears. (Of course, it should reappear when the mouse is released.) This will require that you declare names for the circle and the line and associate them with the correct objects. This can be accomplished by adding **def** ... = to the beginning of the statements creating those objects. Your code to make the objects disappear and reappear goes in the onMousePress and onMouseRelease methods. You can use the visible := methods with right hand side either false or true to handle the disappearing and reappearing.

Finally, add the yellow background to the sign. We didn't have you do this earlier because you had not yet seen how to associate names with objects. Associate a name with the background rectangle when you create it and then use the name to set its color to yellow. Think carefully about where to put this code so that the shapes are stacked in the right order. If you put it in the wrong place then the circle, line and text will be hidden behind the yellow filled rectangle!

## Grading Guidelines

In general, about half the points in each lab will be for correct functionality and the other half for the quality of the code you write in the program. This lab is worth 10 points, which are distributed as follows:

Table 1: Grading Guidelines

| Value | Feature |
|---|---|
| 2 pts. | Drawing the sign correctly when the program starts |
| 2 pts. | Changing the color of the text correctly |
| 2 pts. | Making the slash and circle disappear and reappear correctly |
| 1 pt. | Meaningful names used in declarations |
| 1 pt. | Informative comments |
| 1 pt. | Good and consistent formatting |
| 1 pt. | Good choice of Grace commands |

The last point is worth additional discussion. Often there are several Grace commands that can be used to accomplish the same result. It is important to pick the most appropriate commands. For example, in this lab you could make your circle and slash disappear by sending them behind the background, but this would not be the best choice as it is not obvious what your intent is and there is a more direct way of accomplishing the same result.

## Submitting Your Work

Congratulations, you have written a Grace program! You are encouraged to continue experimenting with it. Before you do, however, you should submit it as a completed assignment. The submission procedure is electronic and will be basically the same every week.

- First, return to Aquamacs and make sure you included your name and course number in a comment at the start of the program. If you modified the program in the web program editor (we strongly recommend that you DO NOT!), make sure the file in Aquamacs is updated to be the same as the one that was run.
- On the desktop, navigate to your folder called "Lab1-Sign" in your CS51Workspace folder. You should now rename that folder by clicking on it and pressing return, and typing in a more descriptive name (one that identifies you and the lab you are working on, such as "Bruce-Lab1" or "Bruce-NoClicking" – except replace "Bruce" by your name!). Note that dashes in names are OK, but don't include spaces or periods.
- Now open the "cs051G" folder icon on the desktop by double-clicking on it. Within the "cs051G" folder you should see a "dropbox" folder.
- Drag the folder you just created into the dropbox folder. When you do this, the computer may warn you that you will not be able to look at this folder. That is fine. Just click "OK".

You can submit your work up to 11 p.m. on Monday evening. If you submit it and later discover that your submission was flawed, you can submit again. We will grade the latest submission made before the 11 pm deadline. The computer may not let you submit again unless you change the name of your folder slightly. It does this to prevent another student from accidentally overwriting one of your submissions. Just add something to the folder name (like the word "revised") and the re-submission will work fine.

When you are done with your session, be sure to log out of the computer by selecting "Log Out" from the "Apple" menu as you did earlier. Please log out each time you leave as otherwise anyone who sits down at the computer can get full access to all of your files.