1 Administrative Topics

- Picture show from Proj 9
- Quiz on Fri will be based on today’s lecture, so I will also give you a hw to help you to study.

2 Adding Parameters to Interpreter Symbols

Did anyone try to draw a parallelogram or trapezoid for Proj 9? It wasn’t straight-forward because the current design allows for only one angle. Likewise, you had to get rather clever to create a rectangle. What if we could specify the angle for every turn and the distance for every forward move?

Using some additional code, we can do that. This code will “parse” the string of characters to make sense out of it. Parsing is a common idea in computer science – we analyze a string of characters by looking at them one at a time, and build up some values that are related to the string’s content.

We will add support for numeric parameters preceding the F, +, and - symbols. E.g. (5)F means “forward(5*distance)” (i.e. go 5 times as far as you would with a regular old F).

Let’s start with a simple example to illustrate the idea. Here is the string for a rectangle using the new scheme:
(5)F(90)+(2)F(90)+(5)F(90)+(2)F(90)+

We need to update Interpreter.drawString to handle this new option.

The current basic structure of drawString is this:

```python
# preliminaries
stack = []

# for each symbol in dstring
for c in dstring:
    # interpret the symbol as a turtle command
    if c == 'F':
        turtle.forward(distance)
    elif c == '+':
        turtle.left(angle)
    elif c == '-':
        turtle.right(angle)

# postliminaries
turtle.update()
```

I demo the code using the example F+F. It draws the lower right hand corner of a square.

This needs to change. Why? Because not every symbol leads directly to a turtle command. In particular, what happens when the symbol is "("? We learn that we are about to start seeing characters that together make a number. That means we need to store some information indicating that we are in “number-parsing mode”. This is information about the context or current “state” of the function – sort of like the stack of positions and headings. Then, when we encounter the ")" symbol, we need to leave number-parsing mode and return to turtle-drawing mode.

This means our new structure will look more like this

```python
# preliminaries
stack = []
# set-up for parameter handling

# for each symbol in dstring
for c in dstring:
    # If the character is (
        # put us into number-parsing-mode
```
How do we

- Indicate that we are in (or not in) number-parsing mode?
  Use a variable (like stack) named pargrab. Set it to True if we are in number-parsing mode and False if we are not.

- Indicate whether or not there is a “fresh” parameter value to be used?
  Use a variable named parval. Set it to None if there is no “fresh” parameter value. If there is a “fresh” parameter value, then set it to that.

- Store the number string as we are building it up in number-parsing mode?
Use a variable named parstring. Set it to ‘ ’ when we first enter number-parsing mode. Add characters to it while we are in number-parsing mode, and then use its value to set parval’s value when we leave number-parsing mode.

This means we add these lines of code to the outline

```python
# preliminaries
stack = []
# set-up for parameter handling
parstring = ‘ ’
parval = None
pargrab = False

# for each symbol in dstring
for c in dstring:
    # If the character is (:
    if c == ‘(‘:
        # put us into number-parsing-mode
        pargrab = True
        # initial a number-string variable to ‘ ’
        parstring = ‘ ’
        # continue to the next iteration of the loop
        continue
    # elif the character is ):
    elif c == ‘)’:
        # put us out of number-parsing-mode
        pargrab = False
        # convert the number string to a float
        parval = float(parstring)
        # continue to the next iteration of the loop
        continue
    # elif we are in number-parsing-mode
    elif pargrab:
        # add this character to the number string
        parstring += c # assuming c is the variable name
        # continue to the next iteration of the loop
        continue
    # if we get this far, then the symbol was not part of
    # a parameter, so
    # interpret the symbol as a turtle command
    if c == ‘F’:
        # but we may need to use a parameter here
        turtle.forward(distance)
```
elif c == '+':
    # but we may need to use a parameter here
    turtle.left(angle)
elif c == '-':
    # but we may need to use a parameter here
    turtle.right(angle)

    # indicate that a parameter has been used, so we don’t
    # mistakenly use it for the next symbol
    parval = None

# postliminaries
turtle.update()

This code allows us to parse parameters and throw them away. Let’s play
with this code for a bit. I will add a bunch of print statements and run it
with the example (5)F(30)+(2)F.

Now, let’s add code to use the parameters.

# preliminaries
stack = []
# set-up for parameter handling
parstring = ''
parval = None
pargrab = False

# for each symbol in dstring
for c in dstring:
    # If the character is (  
    if c == '(':  
        # put us into number-parsing-mode
        pargrab = True  
        # initial a number-string variable to ''
        parstring = ''  
        # continue to the next iteration of the loop
        continue
    # elif the character is )
    elif c == ')':  
        # put us out of number-parsing-mode
        pargrab = False  
        # convert the number string to a float
        parval = float(parstring)  
        # continue to the next iteration of the loop
        continue
# elif we are in number-parsing-mode
elif pargrab:
    # add this character to the number string
    parstring += c  # assuming c is the variable name
    # continue to the next iteration of the loop
    continue

# if we get this far, then the symbol was not part of
# a parameter, so
# interpret the symbol as a turtle command
if c == 'F':
    if parval == None:
        turtle.forward(distance)
    else:
        turtle.forward(distance*parval)
elif c == '+':
    if parval == None:
        turtle.left(angle)
    else:
        turtle.left(parval)
elif c == '-':
    if parval == None:
        turtle.right(angle)
    else:
        turtle.right(parval)

    # indicate that a parameter has been used, so we don’t
    # mistakenly use it for the next symbol
    parval = None

# postliminaries
turtle.update()

I rerun it for the same example and the picture looks like I wanted it to.