1 Administrative Topics

- Picture show from Proj 9

2 Project 10

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.

In lab, 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).

In class, we will accomplish a simpler task - we will add support for drawing text. The T symbol will indicate that text should be drawn. The text to draw should appear before the T within double quotes. e.g.

"Fly”T

will indicate that the word Fly should be written on the turtle window.
3 Interpreting Text Strings

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()
```

We need to augment the code within the loop. Why? Because not every symbol leads directly to a turtle command. In particular, what happens when the symbol is a double quote? We learn that we are about to start seeing characters that together make a text string. That means we need to store some information indicating that we are in “text-grabbing 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 second double quote symbol, we need to leave text-grabbing 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 double-quote and we are not
    # in text-grabbing mode
    # put us into text-grabbing mode
    # initialize a text-string variable to ’’
```
How do we

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

- Store the text string as we are building it up in text-grabbing mode?
  Use a variable named text_string. Set it to '' when we first enter text-grabbing mode. Add characters to it while we are in text-grabbing mode. We can use it when we encounter a T symbol.
This means we add these lines of code to the outline

```python
# preliminaries
stack = []

# set-up for parameter handling
text_string = ''
text_grab = False

# for each symbol in dstring
for c in dstring:
    # If the character is 
    if c == '"' and not text_grab:
        # put us into text-grabbing-mode
        text_grab = True
        text_string = ''
        continue
    elif c == '"':
        # put us out of text-grabbing-mode
        text_grab = False
        continue
    elif text_grab:
        # add this character to the number string
        text_string += c # assuming c is the variable name
        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)
    elif c == 'T':
        # write the text
        turtle.write( text_string )

# postliminaries
turtle.update()
```
We can test the new feature with this code:

```python
if __name__ == '__main__':
    # Test the Interpreter's ability to draw text strings
    terp = Interpreter()
    terp.color('blue')
    terp.drawString(dstring='F+F+F+F"Frogs are Fun"T{F−F−F−}', distance=100, angle=90)
    terp.hold()
```