CMSC 143: Introduction to Object-Oriented Programming with Robots

Lab 10: Postfix-Python
Due November 16, 2009

In this lab, we will create a new programming language for controlling our robots. The language will be similar to **Reverse Polish Notation (RPN)**[1]. In reverse polish notation, rather than putting the mathematical operators <u>between</u> the operands we put them <u>after</u> the operands. For example, 3 + 2 is represented by 3 2 + and (3 + 2) / 4 is represented by 3 2 + 4 /. Other than being a little strange, this notation makes writing computer programs to evaluate these expressions simpler, and we don't need parentheses. The programming languages Forth, Postscript, and Joy all use postfix notation.

Below is an example of a small postfix-mathematical interpreter (this particular program could benefit from some better error handling). We are using the pop() method for lists which grabs and deletes the last element of the list. In fact, we are using the list as a specific data structure known as a **stack**. When using a stack, we treat the list like a stack of plates or papers; we add items to the end of the list, and also remove items from the end. We are accessing the list in last-in-first-out order.

```
def display(x):
  print x.pop()

def add(x):
  v2 = int(x.pop())
  v1 = int(x.pop())
  x.append(v1 + v2)

def mult(x):
  v2 = int(x.pop())
  v1 = int(x.pop())
  x.append(v1 * v2)

def sub(x):
  v2 = int(x.pop())
  v1 = int(x.pop())
  x.append(v1 - v2)

def div(x):
  v2 = int(x.pop())
  v1 = int(x.pop())
  x.append(v1 / v2)

operators = {'display': display, '+': add, '-': sub, '*': mult, '/': div}

def interpret(expression):
    stack = []
    for token in expression.split():
        if token in operators:
            operators[token](stack)
        else:
            stack.append(token)

interpret('5 display')
interpret('3 2 + display')
interpret('3 2 + 4 * display')
```

---

[1] http://en.wikipedia.org/wiki/Reverse_Polish_notation

A Postfix Robot Programming Language

Your programming language should have the following functionality:

1. Read a list of commands from a file.

2. Lines starting with the # character should be treated as comments and ignored.

3. A display command for printing out values.

4. Simple forward, backward, turnRight, turnLeft, beep commands.

5. Commands for reading the sensors (e.g. getLeftLight, getRightIR getBattery).

6. Commands for taking and showing pictures.

7. The ability to perform simple mathematical expressions (+, -, /, *, **).

8. **EXTRA:** You might extend your language with `pop`, `dup`, `swap` commands that remove the top item on the stack, duplicate the top item on the stack, and swap the top two items on the stack, respectively.

9. **EXTRA:** The ability to store numbers in variables; for this a dictionary would be useful.

An Example Postfix-Python Program

```
# beep for 1 second at 440 Hz
1 440 beep

# beep both an A and an E for 0.5 seconds
0.5 440 2 * 650 beep2

# print out the current left light value
getLeftLight display

# take and show a picture
takePicture showPicture

# go forward for half a second and full power
1 0.5 forward

# create a variable named power with the value 0.5
0.5 power store

# go backward at power for 1 second
power 1 backward
power 2 / 0.5 turnLeft
```

Learning Objectives

○ Read from Files   ○ Use Dictionaries   ○ Create a Simple Interpreter

Deliverables

Submit two files:

1. `cmsc143_lab8_LASTNAME_FIRSTNAME.py` – Your interpreter

2. `cmsc143_lab8_LASTNAME_FIRSTNAME.yp` – An example program using all your language's features.