Jeremiah Howden
Elias Moreno

Project 2 Report

Introduction

We started the project with the Project 1 Scanner codebase with some minor fixes to return both the tokens and their values from our test cases. We stuck to recursive descent systematically to ensure an all encompassing top-down approach. The heaviest data structures we used were arrays because the recursion was the bulk of the project. When we needed to look forward we did so with fail states to ensure that nothing gets printed or modified that shouldn't be there, and if it did it errored out the entire program.

Pseudo Code

Parser:

```
constructor(self):
    tokens[]
    tokenValues[]

parse(self):
    called in from main.py
    program(0)

program(self, tabs):
    print('<program>', tabs)
    stmt_list(tabs + 1)
    print('</program>', tabs)

stmt_list(self, tabs):
    print('<stmt_list>', tabs)
    if there is still something left to read:
        stmt(tabs + 1)
        stmt_list(tabs + 1)
    print('</stmt_list>', tabs)

stmt(self, tabs):
    print('<stmt>', tabs)
    if current token is 'id'
```

```
            print id with tags
            increment token index

            if current token is ':='
                print assign with tags

            else:
                throw exception

            expr(level)

        elif current token is 'read'
            print read with tags
            increment token index

            if current token is an id
                print id with tags
                increment token position
            else:
                throw exception

        elif current token is 'write'
            print 'write' with tags
            increment token position

            else:
                throw exception

        print('</stmt>', tabs)


expr(self, tabs):
        print('<expr>', tabs)
        term(level + 1)
        term_tail(level + 1)
        print('</expr>', tabs>

term(self, tabs):
        print('<term>', tabs)
```

```
     factor(tabs + 1)
     fact_tail(tabs + 1)
     print('</term>', tabs)
     return True

term_tail(self, tabs):
     print('<term_tail>', tabs)
     if add op is successful
          and if termis successful
               term_tail(tabs + 1)
          else
               throw exception

     print('</term_tail>', tabs>

factor(self, tabs):
     if current token is 'lparen'
          print lparen with tags
          increment token position
          expr(tabs + 1)
          ensure token after is 'rparen'

     elif current token is 'id'
          print id with tags
          increment token position

     elif current token is 'number'
          print number with tags
          increment token position


fact_tail(self,tabs):
     print('<fact_tail>', tabs)
     if mult op is successful
          and if factor is successful
               recur
          else
               throw exception
     print('</fact_tail>', tabs>
```

```
add_op(self, tabs):
     if current token is ‘plus’
          print plus with tags
          increment token position

     elif current token is ‘minus’
          print minus with tags
          increment token position

mult_op(self, tabs):
     if current token is ‘mult’
          print mult with tags
          increment token position

     elif current token is ‘div’
          print div with tags
          increment token position

match(self, value):
     return current token == value
```

Test Cases

Test Case file	**Test Case**
BadAssignment.txt	y = 7 /* this should throw an error */
GoodAssignment.txt	x := 7
Read.txt	Read A
WithComments.txt	/** * With some kind of comment */ Write x + 5 + y
Write.txt	Write 1 + 2

Jeremiah Howden
Elias Moreno

Acknowledgment

- Ynigo Reyes helped us understand Recursive Descent