Project 2: TreeBuilder Lexical Analysis
Due Friday, April 4th at 5:00 pm
100 points

Introduction

In this 2 part project we will be building a rudimentary compiler for a language that would allow us to work with Trees. Our language will be able to describe trees where each node contains a name and a weight and may have an unlimited number of children. After building trees, we should be able to print them off to the screen (as text, not a graphic representation). For the current stage of our project, we will only complete the lexical analysis portion of the project. We will complete Syntactic analysis of our language in project 3.

TreeBuilder Language

Our language has the following commands that need to be accepted:

1. buildnode{ name="value"; weight=7; isachildof ="other_node_name"; } ;
    a. isachildof is optional if the node is a root
2. for var in [1:10] { //one or more build node commands} ;
    a. var can be any valid variable name (starts with letter or number, can include letters, numbers and _)
    b. any integers can be used for low and high value, you may assume that they will be in order of low to high for the range
    c. loops through each value in the range, storing that value in var
    d. var is scoped to the loop, cannot be referenced outside of it, so no need for a symbol table for int variables
3. for var in ["s1", "s2", "s3" …] { //one or more build node commands };
    a. Any number of string literals separated by commas can be used
    b. var can be any valid variable name (starts with letter or number, can include letters, numbers and _)
    c. loops through string in the list, storing that value in var
    d. var is scoped to the loop, cannot be referenced outside of it, so no need for a symbol table for string variables
4. Additionally, the language can use integer expressions and string expressions using the + operator any place where an int or string literal is expected. Loop control variables can be used in these expressions
    a. integer expressions using + result in adding the values together, can be chained together for longer expressions (ex: weight= 5 + 4 + var + 3;)
    b. String expressions using + result in string concatenation. (ex name="node_"+var+"_example";)
    c. Any expression with the + operator with a mix of strings and integers evaluates as a string expression

Program Structure

For this stage of the project, we will only need to complete lexical analysis of the language using lex. You need to create the lex file to perform lexical analysis, and add a main function that will print the tokenized version of the input file to the screen.

In our previous project we used LEX, but for a different purpose. Some things may be different in this stage. For instance, we do not need tokens for individual symbols in our code (+, ;, {, }, etc). IN the next stage these tokens will be passed onto YACC, and it is much easier to resolve these symbols in YACC if we just pass the symbol along instead of a numeric token. We only tokenized everything in project 1 because we needed numbers for the winnowing algorithm.

EXAMPLE CODE

Here is an example program in the tree builder language:

```
buildnode{
name="root";
weight=10;
};
for i in [1:10] {
  buildnode {
  name = "A"+i;
  weight = 3+i+1;
  isachildof = "root";
  };
}
for i in [1:5] {
  buildnode {
  name = "B"+i;
  weight = 3;
  isachildof ="A"+i;
  };

}
for i in [1:5] {
  buildnode {
  name = "C"+i;
  weight = 3;
  isachildof = "B"+i;
  };
  buildnode {
  name = "D"+i;
  weight = 1;
  isachildof = "B"+i;
  };

};
```

Group Work Policy

You may work with one partner on this assignment, but you are not required to do so. Only one group member should submit, and they should include both partners names in all code files and as a comment in the blackboard submission. Since this is a two part project, if you work with a partner on project 2, you must either continue working with them on project 3 or continue working alone. You cannot change to a new partner. You do not need to work with the same partner you worked with on project 1.

Late Policy

Late projects are subject to a 20% penalty for up to 24 hours after the due date. After 24 hours, the assignment will no longer be accepted. You are responsible for submitting the correct file for your project submission and for submitting on time.