Project 2: TreeBuilder Compiler
Due Friday, April 26th at 5:00 pm
200 points

Introduction

In this 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). In order to complete this project we will need to use our knowledge of LEX, YACC, and C++.

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 mist of strings and integers evaluates as a string expression

Program Structure

You will need several program files to complete this assignment

1. tree_builder.l : Your lex program file to tokenize your code. We do not need to include a main, or much auxiliary code, as this just needs to return tokens to YACC. Recall, not

everything NEEDS to be tokenized before going to YACC. Its fine to just pass individual symbols or operators on (such as =, +, etc)

2. tree_builder.y : Your YACC program to parse the grammar and perform syntactic analysis
3. tree_node.h : Code file containing a class or struct for one node in a tree
4. parse_tree.h : Code file containing class definitions and code to respond to the statements in the code.
5. A makefile to compile your compiler

Once completed, your compiler will be able to read in a Tree Builder code file (see examples) and build the tree in memory, then print the tree to the screen. See the examples folder for example input and output of the completed program.

Notes:

- Use the interpreter example from class as a guide. It is a very similar structure
- Write a simple C++ main to use your tree_node class to build and print a few trees to test before using that class with your compiler. You don't want to debug a pointer issue hidden behind a few layers of LEX and YACC

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;
    };

};
```

See the Examples folder for more examples and example output of the tree.

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.

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.

University rules do not allow me to have an assignment due after Friday unless it is a final project that replaces an exam. No extensions can be given on this assignment.