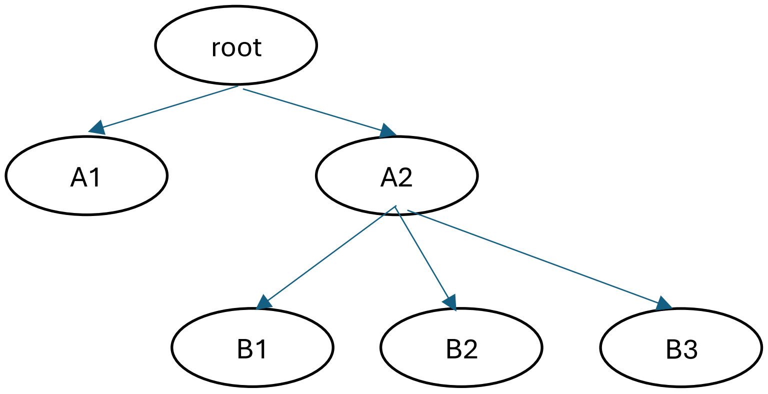
Project 3: TreeBuilder Compiler
Due Monday, April 21st at 8:00 pm
100 points

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++. This project will build off of the lexical analysis done in project 2, but bring in syntactic analysis with YACC and then generate runnable C++ code for our program in the example TreeBuilder language.

TreeBuilder Language

Please see the project 2 prompt for a description of the language. When printing a tree to the screen, please use the following format:
root_name [child1 [...], child2[...], ..., child_n[...]]
So the following tree:



Would be printed as"
root [A1, A2 [B1, B2, B3]]

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). This should largely be completed from project 2
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. I highly recommend using the parse_tree.h file from our interpreter example as an example structure for this file. It is fine to use code from the interpreter example given in class

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.

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

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. If you worked with a partner on project 2, you need to either work with that same partner or work alone. You cannot change to a different partner at this point. If you did not work with a partner on project 2, you may still partner up with someone else who worked alone for project 3.

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.