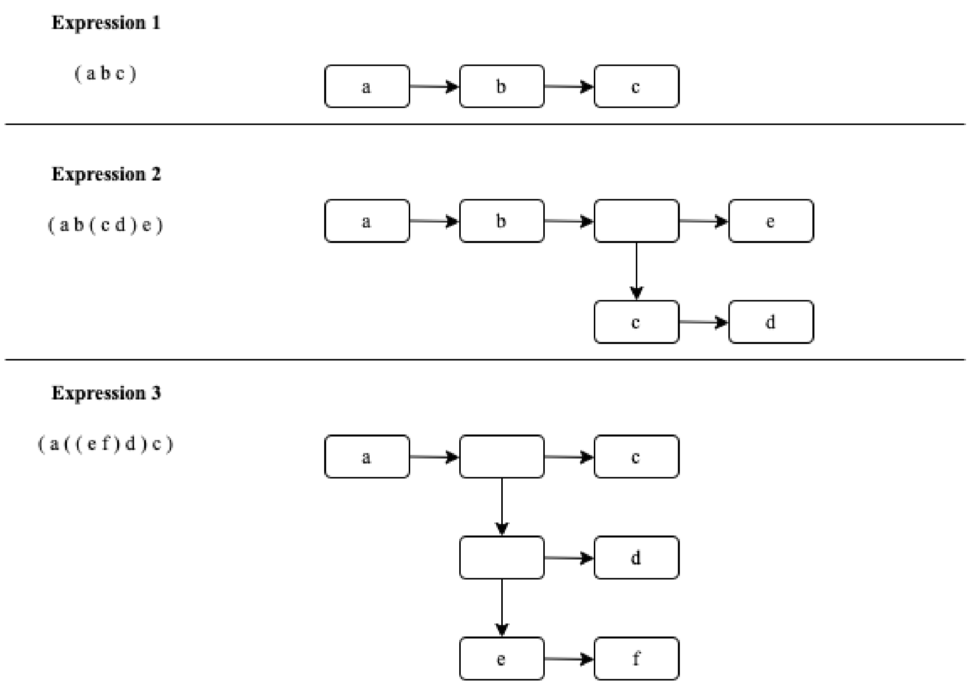
Computing Structures – Fall 2022
Project 4
Due: November 7th 2022 at 11:59 PM

Objective:
The goal of this project is to create and learn about Generalized Lists data structure that is a type of linked list structure we are going to use to store parenthesized expressions.

Description:
A Generalized list is a type of linked list structure that has a data field, a next pointer and a down pointer as well. This down pointer can point to another node that can be a linked list by itself. So a node can be pointing to two linked lists, one to the next of it and another to the bottom (down).

Let us take a picture to explain this:



Given are three expressions in the parenthesis form. The variables *a*, *b*, *c*, *etc*. are the data that are present in the nodes of the linked list. In expression 3, we say that *a* and *c* are in the first level, *d* is in the second level and variables *e* and *f* are in the third level.

The parenthesized expression gives the levels with the number of parenthesis present. A node that has a not NULL down pointer will not have a char variable stored in it. We use this data structure to store the expression. You can also use a generalized list to store a tree structure that we will study later.

For this project, you are given the parenthesized expressions as an input and you are first expected to build the generalized list (GL) data structure for these expressions given one by one. Once the GL is built, you are going to perform some queries on this structure.

The boilerplate with the class definition is given to you. You are not allowed to change the class fields. All the queries given are supposed to be performed in the GL object structure that is given to you. All traversals for the queries must be performed in the GL structure itself.

Query options are explained below. Example for each is given in the next section.

- D – display the expressions one by one.
- F – find the char variable in the given expression. Return true or false.
- L – Least Common Ancestor (LCA) is the lowest node that has both given char variables as descendants. Here we find the longest path to the least common ancestor from either of the char variable nodes. For example: In expression 0 the LCA of *a* and *c* is 1. In expression 2, the LCA of *f* and *d* is 2.
- H – find the height of the given expression.
- U – find and display the different duplicates in the given expression. If no duplicates, print “No duplicates”.

Input explanation:

The input file is different from the csv document that is given. The input file is read in via redirected input (like project 1). The input file contains in this specific order, the number of rows/records, number of columns, if the csv file contains headers or not, the csv file’s name and followed by a series of set of character options/command and a corresponding information. Here is a sample input file:

```
3               ← number of expressions
(abc)           ← expression 0
(ab(cd)e)       ← expression 1
(a((ef)a)c)     ← expression 2

D               ← call display method to display all expressions
F 0 b           ← find the charVariable b in expression 0
F 0 c
L 1 a d         ← find the least common ancestor distance between a and d chars in expression 1
H 1             ← find the height of expression 1
H 2
U 2             ← display all the duplicates in expression 2
```

Class definition:

A boiler plate source file has been provided along with this project specification. The class definition and a main function have also been given with comments to get you started with the project.

A sample output file will be provided (in the next few days) for the given corresponding input file. Your output is supposed to exactly be the same as this output. You need to essentially follow this format of the output. This is necessary for your program to pass the autograder.

Redirected input:

Redirected input provides you a way to send a file to the standard input of a program without typing it using the keyboard. To use redirected input in Visual Studio environment (on windows), follow these steps: After you have opened or created a new project, on the menu go to project, project properties, expand configuration properties until you see Debugging, on the right you will see a set of options, and in the command arguments type <"input filename". The < sign is for redirected input and the input filename is the name of the input file (including the path if not in the working directory).

If you use macOS or linux (or windows using powershell), you may use the terminal to compile and run your program using g++. This is the compiler that we use in the autograder on GradeScope (you may need to install g++ on your computer). Once you installed g++, you may compile your program using the following command:

```
g++ project4.cpp -o p4
```

You may then run the program with the redirected input using the following command:

```
./p4 < input1.txt
```

Make sure your program and your input file are in the same folder.

A simple program that reads the input file and displays everything that was read is given below.

```
#include <iostream>

using namespace std;

int main ()
{
	int numExpressions;

	cin >> numExpressions;
	cout << numExpressions;

	return 0;
}
```

Submission:

Submission will be through GradeScope. Your program will be autograded with test cases and then hand graded to check for documentation and if you have followed the specifications given. You may submit how many ever times to check if your program passes the test cases provided. Test cases will be released at the beginning and later other test cases will be released while grading. For the autograder to work, the program you upload must be named as **project4.cpp**.

Your final submission must contain your source file named **project4.cpp**. You access GradeScope using the tab on the left in our course page on canvas.

Sample output file for corresponding input files will be released. The input1.txt file given is a simple input file that will help you understand the project, more complicated ones will be released later and used for grading.

Constraints:

1. In this project, the only header you will use the header files that are given to you in the boiler plate code. Using other or excess header files will be subject to a heavy grade penalty for the project.
2. None of the projects is a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.
    - Please review academic integrity policies in the modules.

How to ask for help:

1. Check FAQ discussion board of the project first.
2. Email the TA with your precise questions.
3. Upload your well commented program to CodePost.
    a. This is a website which is used to share code.
    b. You will upload your program and I can view it and comment on it.
    c. Here is the invite link to our class for the summer.
    d. Once you join the class on CodePost, you can upload your program to the Project 1 assignment.

        Note: Your program will not be auto graded at CodePost, this is just for when you want to ask a question and a place where I can look at your program and comment on it. CodePost is great for live feedback. GradeScope is the place where you should submit and where your program will be autograded.