CQUPT – University at Albany
Computer Science – International College

ICSI 403 --- Design and Analysis of Algorithms
Programming Assignment 3 --- Spring 2025

Assigned: Tuesday, April 16th, 2025.
Due: Your co-instructor will inform you the date this exercise is to be submitted.

Purpose of the Project

The primary goal of this exercise is to develop a simple tool to provide a linear execution precedence order for a set of algebraic expressions. A secondary goal of the exercise is to develop skills for working with the fundamental graph algorithms such as DFS and Topological Sort.

Background

A cycle in directed graph G is a set of edges, $\{(v_1, v_2), (v_2, v_3), ..., (v_{r-1}, v_r)\}$ where $v_1 = v_r$. A diagraph is acyclic if it has no cycles. Such a graph is often referred to as a directed acyclic graph, or DAG, for short. DAGs are used in many applications to indicate precedence among events. For example, applications of DAGs include the following:

- Inheritance between C++ classes or Java interfaces.
- Prerequisites between courses of a degree program.
- Scheduling constraints between the tasks of a projects.

A DAG can be used for modeling processes and structures that have a **partial order**: $a > b$ and $b > c$ imply $a > c$. But may have a and b such that neither $a > b$ nor $b > a$. One can always make a **total order** (either $a > b$ or $b > a$ for all $a \neq b$) from a partial order. In fact, that's what a **topological sort** will do. In essence, a topological sort is a linear ordering of all its vertices such that if DAG G contains an edge (v_i, v_j), then v_i appears before v_j in the ordering.

A **precedence graph** is a DAG whose nodes correspond to individual statements or program fragments. The following characteristics apply to such graphs:

1) One node for each activity
2) A directed edge *(u, v)* means activity *v* depends on activity *u*. The syntax *u -> v* will be used to represent this.

To Do

You are to provide a linear ordering for a set of expressions such that the expected result is produced. The data set may be available from either a text file or a spreadsheet. You are to use the attached files *project3-data.xlsx* and *project3-data.txt* for the development and testing of your initial prototype.

Consider the following set of expressions obtained from a spreadsheet:

	A	B	C
1	C4-7	4	C4+6
2	A3+A1-C4	1+B1	B1-A1
3	7	A3*C2-B2	B3+A3
4	A1*B1*A2	C2-A4	9

The above document uses syntax A1, A2, A3, and C4 to collect the results of the evaluation of expressions on column A, row 1; column A, row 2; column A, row 3, and column C, row 4 respectively. The expressions associated with the referred locations are illustrated in the text box below.

1) A1 = C4-7
2) A2 = A3+A1-C4
3) A3 = 7
4) C4 = 9

The above four expressions will provide the correct result when processed according to the following evaluation sequence: 3, 4, 1, and 2.

Your solution must:

1) Construct a **precedence graph** to collect the data to be processed. For instance, the precedence graph representing the above spreadsheet data will have 12 nodes. Nodes n_1 to n_4 will be associated with expressions A1 to A4, nodes n_5 to n_8 will represent expressions B1 to B4 and nodes n_9 to n_{12} will be assigned to expressions C1 to C4. In essence, a one-to-one mapping between node and activity is to be provided and a precedence graph is to be constructed.

2) Obtain a linear ordering of your DAG by means of a **topological sort** of your precedence graph. Use **Depth First Search** (DFS) algorithm as the basis for the implementation of the topological sort.

You are to use the C++ programming language for this exercise.

Testing

For testing your prototype, you are to read the set of expressions from a text file. Input files are to follow the spreadsheet format discussed in this document.

Documentation

Your program must be developed using the C++ programing language. Your program should be layered, modularized, and well commented on. The following is a tentative marking scheme and what it is expected to be submitted.

1. *External Documentation* including as many pages as necessary to fulfill the requirements listed below.
    a. Title page
    b. A table of contents
    c. [20%] System documentation
        i. A high-level data flow diagram for the system
        ii. A list of routines and their brief descriptions
        iii. Implementation details including information such as data structures and algorithms used.
    d. [5%] Test documentation
        i. How you tested your program
        ii. Testing outputs
    e. [5%] User documentation
        i. How to run your program
        ii. Describe parameter (if any)

2. Source Code
    a. [65%] Correctness
    b. [5%] Programming style
        i. Layering
        ii. Readability
        iii. Comments
        iv. Efficiency

How to Submit your Work

- Your solutions must be submitted to the address provided by your co-instructor.
- Your files for Project *3* must include the source code for all modules used for the exercise as well as an executable file to allow your solution to be tested. You must include all your source files in a Compressed (zipped) folder (.zip). Your (.zip) folder as well as the subject of your message must follow the format: *Project 3 Your Name*. Marks will be deducted if you do not follow this requirement.