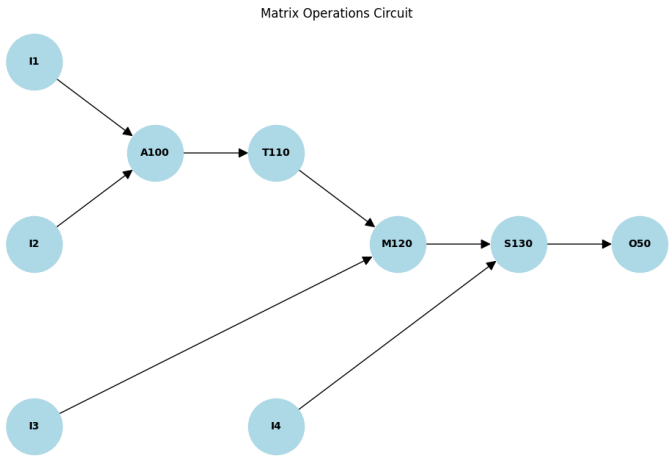
DSA 5005 – Computing Structures – Fall 2024 – Project Two
Due 11:59 PM, October 7, 2024

Imagine you have a set of specialized computing modules, each designed to perform a specific elementary matrix operation. These operations include matrix addition, subtraction, multiplication, and matrix transpose. Each computing module, except for the matrix transpose module, requires two input matrices and produces a single output matrix.

There are two unique types of computing modules: the **input module** and the **output module**. The output module has one input and no output, as its sole function is to display the matrix it receives through its input line. For example, if the output module receives a matrix, it displays the message: "I am the output module, and the matrix I received is: [matrix contents]."

In addition to the output module, an input module is designed to provide matrices to the computing system. This input module feeds values into the workflow, serving as the data source for the calculations performed by the other modules.

Now, consider a specific configuration where four matrix-processing modules are connected sequentially to perform a series of matrix operations.



These modules are configured as follows:

1. **A100 (Addition Module):** This module adds two input matrices. It is connected to two input modules labeled I1 and I2, which provide the matrices to be added.
2. **T110 (Matrix Transpose Module):** The output from the addition module A100 is sent to the matrix transpose module T110. This module takes the matrix output from A100 and computes its transpose, producing a new matrix where rows and columns are swapped.
3. **M120 (Multiplication Module):** This module requires two input matrices to perform matrix multiplication. One of its inputs is connected to the output of the matrix transpose

module T110, while the other input is connected to the output of a third input module labeled I3.

4. **S130 (Subtraction Module):** The output from the multiplication module M120 is fed into one of the inputs of the subtraction module S130. The other input to this module comes from the output of an additional input module labeled I4. The subtraction module computes the difference between the two input matrices and produces an output matrix.

Finally, the output from the subtraction module, S130, is connected to the input of the output module, labeled O50. This output module, O50, receives the final matrix computed from the sequence of operations. Upon receiving the matrix, O50 displays it. For example, if the final result is a matrix, it would display: "I am the output module O50, and the matrix I received is: [matrix contents]."

In summary, the configuration of these computing modules allows a series of matrix operations to be performed in sequence, where each module's output is passed as an input to the next module in the chain. The entire process culminates in the output module, which displays the final result of the computations.

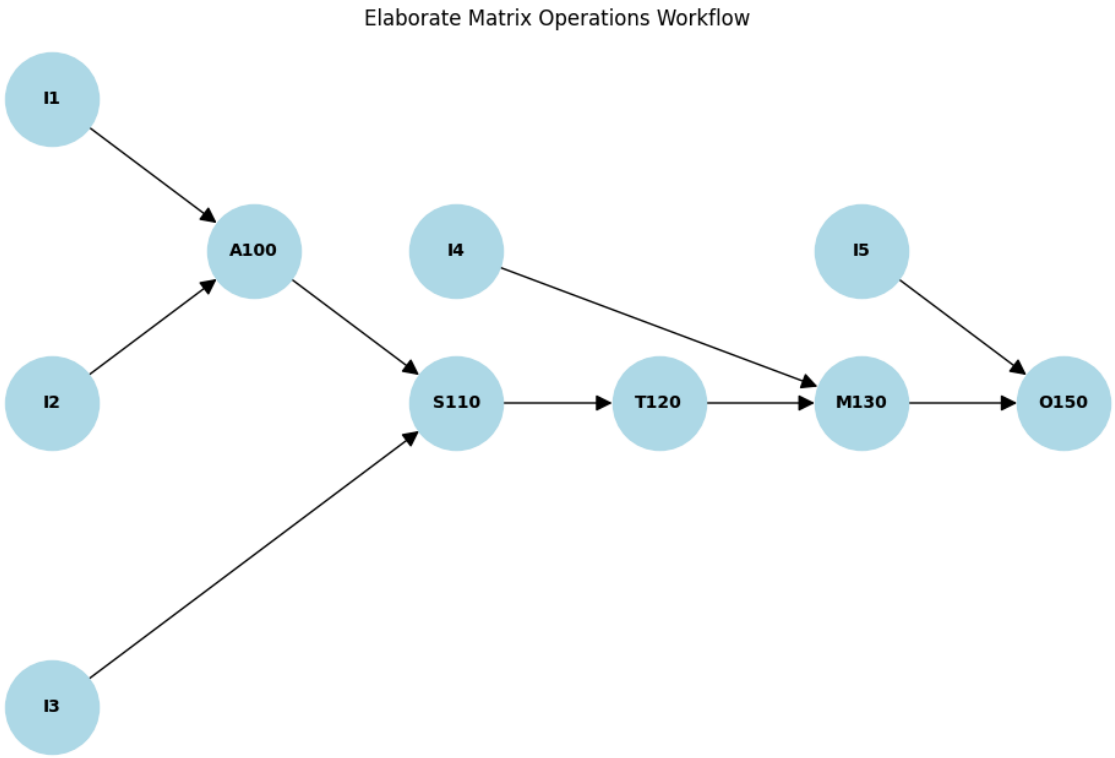
The input to your program will have the following format for the above. First, we will add the modules, and then we will connect them. For example, the input for the above workflow is given below:

```
9       <- The number of modules
I1
I2
I3
I4
A100
M120
T110
S130
O50
8       <- Number of commands to establish the links in the workflow
A I1 A100
A I2 A100
A A100 T110
A T110 M120
A I3 M120
A M120 S130
A I4 S130
A S130 O50
```

After the connections are made. We will supply the inputs for the program as follows. Note that the output computing module is O50. The first line of all these are followed by the name of the input module followed by the number of rows and columns of the input matrix. The next lines will contain the matrix.

```
9          <- The number of modules
I1
I2
I3
I4
A100
M120
T110
S130
O50
13         <- Number of commands to establish the links in the workflow and inputs and output.
A I1 A100
A I2 A100
A A100 T110
A T110 M120
A I3 M120
A M120 S130
A I4 S130
A S130 O50
I I1 5 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
I I2 5 5
25 24 23 22 21
20 19 18 17 16
15 14 13 12 11
10 9 8 7 6
5 4 3 2 1
I I3 5 4
2 3 4 5
6 7 8 9
10 11 12 13
14 15 16 17
18 19 20 21
I I4 5 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
O O50
```

Now consider a much more elaborate workflow, as shown pictorially below.



The input for building the above workflow will be as follows, I have not mentioned the Input matrices and they will have same format like the ones above.

10 <- The number of modules
I1
I2
I3
I4
I5
A100
S110
T120
M130
O150
14 <- Number of commands to establish the connections and inputs and output
A I1 A100
A I2 A100
A A100 S110
A I3 S110
A S110 T120
A T120 M130

A I4 M130
A M130 O150
A I5 O150

Your Project Implementation: As part of this project, you will create the Module class as described below.

Programming Objectives:

1. All code must be in C++. You will create a class CM (compute Module), as given below, with appropriate fields.
2. The class will have several methods whose prototypes are provided to you.
3. All input will be read via redirected input. That is, you will not open a file inside the program.
4. The class structure is shown below (you are responsible for fixing any syntax errors).
5. The structure for your main program is also provided. You will use that for your project.

Program Structure:

You will have the following class structure. You have to ensure that all methods and appropriate additional fields are added.

```
#include <string>
using namespace std;

class CM {
private:
    char CMType;    // Type of the CM (A: Addition, S: Subtraction, etc.)
    string id;      // Unique identifier for the CM
    CM* input1;     // Pointer to the first input CM
    CM* input2;     // Pointer to the second input CM (can be NULL)
    CM* output;     // Ptr to the output CM (is NULL for output CMs)
    double computedOutput;  // The result/output of the CM's computation
    bool outputComputed;  // Flag to indicate if the output has been computed

public:
    // Constructor
    CM(char type, const string& id);

    // Method prototypes
    void setInput1(CM* inputCM);  // Sets the first input CM
    void setInput2(CM* inputCM);  // second input CM (can be NULL)
    void setOutput(CM* outputCM); // Sets the output CM (can be NULL)
    void compute();    // Performs the operation based on the CM type
    bool isReady() const;    // Checks if the CM is ready to execute
    void reset(); // Resets the CM to its initial state; Keep the
                  //connections reset; outputComputed to false.
    bool hasOutput() const; //Checks if CM has already computed its output
    void displayInfo() const; // Displays the CM's information

    double getOutput() const; // Returns the computed output value
    string getId() const;    // Returns the CM ID
```

```
};
```

The compute() method is the key to your program. It is first executed on the output CM, that is, the object corresponding to CM, say O50.

Your main program will have the following structure: You will need to write plenty of code and make changes.

```
#include <iostream>
#inlcude <string>
using namespace std;

//Define your CM class and all its methods.

int main () {

      int numCMs;
      CMs** allCMs;

      cin >> numCMs;
      //create an array CM objects pointers
      allCMs = new CM*[numCMs];
      //each array location is a pointer to a CM Object

      for (int i=0; I < numCMs; i++) {
         //read the CM ID based on the first letter to determine its type
         //create the CM object and initialize it appropriately
         //store the CM object in the allCMs array
      }

      // read from input the links and make the appropriate connections.
      //You may need to search the array allCMs to find the CM that is
      //referred and connect.

      // If the first letter is an O then execute the compute method on the
      //object referred.
      // if the first letter is an R, the perform reset on all the objects.

      return 0;
}
```

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, 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).

Constraints

1. In this project, the only header you will use is #include <iostream> and #include <string> and using namespace std.
2. None of the projects is a group project. Consulting with other members of this class our seeking coding solutions from other sources including the web on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.

Project Submission Requirements:

1. **Code Development (75%):** Implement the provided class structure for the CM by writing the necessary methods to manipulate the CM circuit. Your implementation must be fully compatible with the main program provided, which is designed to create and manipulate CM objects. Your code must run successfully with the main program and the corresponding input, demonstrating the correct functionality of the CM class and its methods.

    - **LLM/AI Tool Usage:** You can use Large Language Models (LLMs) or AI tools, such as GitHub Copilot, to assist in writing and refining your classes and their methods. If you did not use LLM or AI tools to write your project, you still have to show for 2. below how you would have used it to find a solution to the project. **You need to make sure that you use the class structure provided to you as the basis, and failure to do that will result in zero points for this project.**

    - **Specification and Documentation:** The remaining 25% of the credits are assigned to how well the program adheres to the structure provided in the project description. You are free to use LLMs, but the program should work in a way that it is intended.
        - **Specification**: Specification includes the following rubrics:
            - **Readability –** The program must be readable. It should declare all variables inside a function at a single place, followed by the I/O, calculations, etc.
            - **Modularity –** The program must use functions to perform tasks. Division of to main task into smaller sub-tasks is up to you.
            - **Correctness –** The program must adhere to the project description. There are multiple ways to solve a problem, but grading will be done on the basis of how well your code implements the description provided in this document.
        - **Documentation:** The code needs to be well documented. Each function should have comments specifying what it is doing. Every variable and container also need to be commented to specify it's purpose. Since everyone is allowed to use LLMs to create their project, it is imperative that you use it for the right purpose and not abuse it. Thus, you have to include comments in your program everywhere you used code suggested from an LLM like Github CoPilot to document your query. The queries have to be specific, for e.g. how do I iterate

through a vector to find a certain element, etc. There will be some rubric items Gradescope related to that.

- **Deadlines:** October 7 11:59 pm is the deadline for this project. A lateness penalty of 10% will be enforced for every day the submission is delayed for.

Programming Advice:

1. Please don't start the project at the last moment. You should have sufficient time to ask about doubts prior to the deadline.
2. Try to divide the program into smaller parts. For example, make sure to check if your program is reading the storing the data correctly before beginning with the calculations part. To that end, you can also create your own input files with limited information to check if your program is working on those first.
3. Know how to debug your programs. If you are using VS Code, you will have to create your own configuration files (tasks.json & launch.json inside the .vscode folder). VS Code sometimes does create them correctly by itself but in my experience that is a hit or a miss. You will have to take help of LLMs for that. If you are using Visual Studio, that comes with a fully functioning debugger out of the box.
4. I hope all of you are comfortable with redirected input by now. If you are using VS Code, debugging with redirected input is not yet supported. You will have to change your program to accept file input and then debug the program. Then just before submission you change it to accept redirected input.