Cairo University
Faculty of Engineering
CMP N103

Spring 2022

بسم الله الرحمن الرحيم

"نرفع درجاتٍ من نشاء وفوق كل ذي علم عليم"

Programming Techniques

Flowchart designer/simulator

Project Requirements

Introduction

A flowchart is a type of diagrams that represents an algorithm or a process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation can give a step-by-step solution to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process of a program in various fields.

Flowcharts are used in designing and documenting complex processes or programs. Like other types of diagrams, they help visualize what is going on and thereby help the viewer understand a process, and perhaps also find flaws, bottlenecks, and other less-obvious features within it. There are many different types of flowcharts, and each type has its own repertoire of boxes and notational conventions. The most common types of boxes in a flowchart are:

- Processing step; usually called activity, and denoted as a rectangular box
- Decision; usually denoted as a diamond.
- Inputs and outputs; usually denoted as parallelograms.

Your Task

You are required to implement a flowchart designer/simulator. Your implementation should be in C++ code. You must use ***object oriented programming*** to implement this application. Your application should help a user draw a flowchart using different statements, connect them with connectors. It should also allow the user to execute the chart show its output and convert the flowchart to a C++ code file.

NOTE: The application should be designed so that the types of flowchart's building statements and operations can be easily extended.

The rest of this document describes the details of the application you are required to build.

Project Schedule

Project Phase	***Deliverables***	***Due date***
Phase 1 Input/Output classes	Phase 1 Delivery	22/4/2022
Phase 2 Project Delivery	Media Delivery And Project Discussion	17/5/2022

Note: At any delivery:

One day late makes you lose 1/2 of the grade.
Two days late makes you lose 3/4 of the grade.

Main Operations

The application should support two modes of operation; design mode and simulation mode. (The default is the design mode)

[I] Design Mode:

In this mode a user is allowed to design a flowchart graphically by providing the following operations (actions):

1- **Add** a new statement block to the chart.
The application should support, at least, the following statements:
   i. Start and End statements
   ii. Simple Value Assignment statement. Its right hand side is a value (e.g. Age = 12.5)
   iii. Variable assignment statement (e.g. X = Y)
   iv. Single Operator Assignment Statement: It has only one operator and its right hand side may contain variables. Supported operators are: (+, -, * and /).
   (e.g. M = 3* 120, X = 2 / Y, A = A - 1, X = Y + Z)
   v. Conditional statement. (For if and loop structures). User can compare a variable with a value or compare two variables. Supported operators are (==, !=, >, <, >= and <=)
   vi. I/O Statements: Read and Write statements
      - Read statement: Reads a variable value form the input device (default is the keyboard, bonus is a file)
      - Write statement: Write a variable value or a message to the output device (default is the screen, bonus is a file)

2- **Connect** two statements: this means to connect two statements with an arrow (a connector) to specify the direction of control flow in the chart. There are no free connections in the flowchart. When a connection is created it must connect between existing statements.

3- **Edit a statement:** To edit a statement text

4- **Edit a connection**: user can edit a certain connection to change its source or destination statements.

5- **Comment** on a statement: user can write a comment for any of the statements or can edit or delete an existing comment. The comment is not shown as a part of the chart; rather it is shown on the status bar when the statement is selected by the user.

6- **Delete** an existing statement or a connector: when deleting a statement all its connections should be automatically deleted. Also, a separate connector can be deleted.

7- **Select/Unselect** a statement or a connector. When the user clicks on one of the statements, it should be highlighted and all information about it should be printed on the status bar. For example, when a statement is selected, the statement comment should be printed on the status bar.

8- **Move** a statement: All its connectors should be moved with it.

9- **Copy-Cut-Paste** a statement.
   i. Copying a statement is to copy only the statement without its connections.
   ii. Cutting a statement is to erase the original statement with its connections then paste the statement with no connections.

10- **Multiple-Selection:** User can select multiple statements and/or connectors to perform an operation on all of them (e.g. move or delete). All selected statements/connectors must be highlighted.

11- **Save** a flowchart to a file (see file format section).

12- **Load** a flowchart from a file (see file format section).

13- **Switch to simulation mode.**

14- **Exit** the application: application should perform necessary cleanup before exiting.

[II] Simulation Mode:

In this mode the system allows the user to run an algorithm defined by a flowchart. Operations supported by this mode are:

1- **Chart Validation**: For the simulation to start, The following should be valid
   a. Chart have exactly one Start and on End statements
   b. Chart is fully connected

2- **Run** the flowchart and display the final results of each variable.

3- **Step-by-step** run (similar to debug mode) and **display** values of the variables after each step.

   Note: If the flowchart has Read statements, user can interact with it to supply inputs. This should be supported by the two types of run mentioned above.

4- **Generate** a C++ code file equivalent to the flowchart. C++ ***goto*** statement better be avoided as possible.

5- **Switch back to design mode and/or rerun**.

Important Notes:

- The above operations are the minimum requirements to accept the project.
- Each operation should have a ***corresponding action class***. (see "Main Classes" section)
- The code of any operation does NOT compensate for the absence of any other operation; for example, if you make all the above operations except the delete operation, you will lose the grades of the delete operation no matter how good the other operations are.

Bonus Operations:

The following operations are bonus and you can get the full mark without supporting them

1- **Zoom in/out** for the whole graph.
2- **Complex expression** assignment statement. (e.g. X = 2 * Y + 5* Z / (M + 3/K))
3- **Subprogram** modules support **& Function call** statements
4- **Detecting Flowchart Errors**: To detect errors like:
   - Syntax errors
   - A loop with no exit condition
   - A call to a function that does not exist
5- **Undoing, Redoing** actions.
6- **Preventing C++ goto statements:** If the user creates a flowchart with some connectors that ***must*** be translated to **goto** statements, the application signals an error and highlights such connectors and asks the user to replace them.

Main Classes

The following classes are the main classes for your application. Because this is your first object oriented application, we have ***partially*** written the code of some of these classes. You are given a ***code framework***. To make it easy to draw flowchart blocks, the code framework makes use of a ***graphics library*** that has classes with member function related to graphics user interface (GUI) part of the application.

Your task is to complete the given framework by either: extending some classes, inheriting from some classes or creating new base classes.

Below is the class diagram then a description is given for the classes.

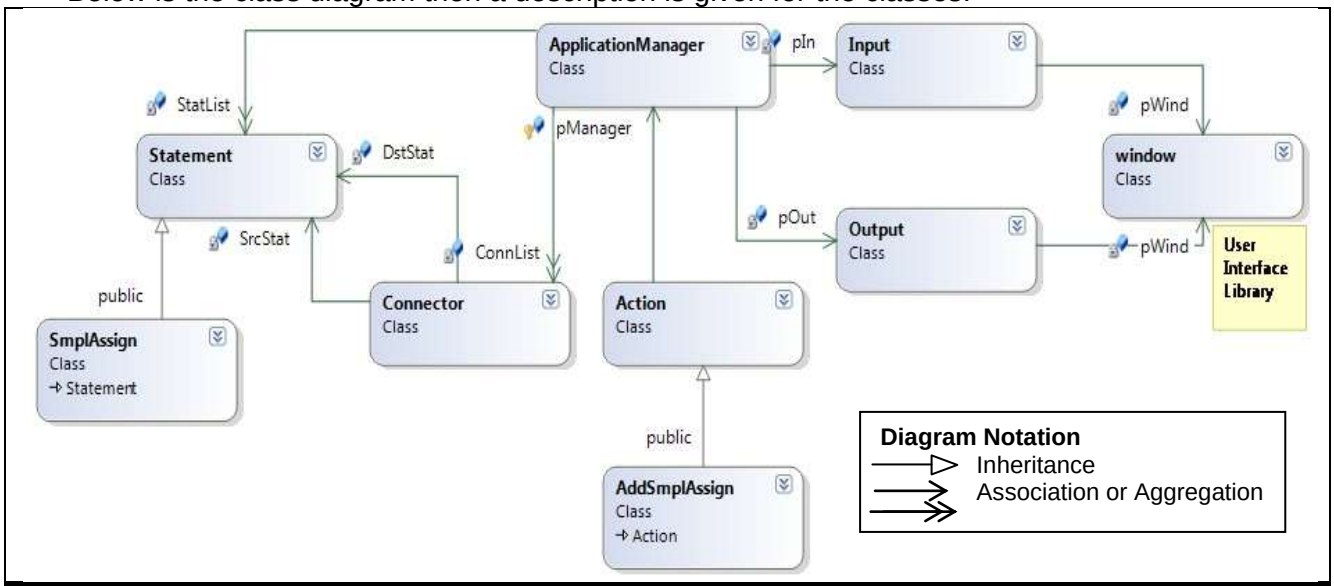

Figure 1 – Class Diagram of the application

Input Class:

There ***MUST*** be a separate class for inputs. All user inputs must come through this class. If any other class wants to read any input, it must call a member function of the input class. You should add suitable member functions for different types of inputs.

Output Class:

There ***MUST*** be a separate class for outputs. This class is responsible for all GUI outputs. It is responsible for toolbar and status bar creation, flowchart drawing, and for messages printing to the user. All outputs must be done through this class. You should extend this class and add suitable member functions for different types of outputs.

Important Note:

The only classes that are allowed to deal directly with the graphics library are Input and Output.

ApplicationManager Class:

This is the ***maestro*** class that controls everything in the application. It has pointers to objects of all other classes in the application. As its name shows, its job is to ***manage*** other classes ***not*** to do other classes' jobs. So it just instructs other classes to do their jobs. In addition, this class maintains the list of statements and connectors inside the application.

Statement Class:

This is the base class for all types of statements (assignments, conditions, I/O,… etc) to be supported by the application. To add a new type of statements (conditional statement for example), you must **inherit** it from this class. Then you should override virtual functions found in the class **Statement**. You can also add more details for the class Statement itself if needed.

Connector Class:

This is the class for the connector that connects between two statements.

Action Class:

This is the base class for all types of actions (operations) to be supported by the application. To add a new action, it must be **derived** from this class. Then you should override virtual functions found in the class **Action**. You can also add more details for the class Action itself if needed.

Implementation Guidelines

- In general, each user operation is performed in *4 steps*
    a. Get user input.
    b. Create suitable action for that input
    c. Execute action.
    d. Reflect action to the Interface.

- Use of Pointers:
    a. Nearly all the parameters passed/returned to/from the functions should be pointers to be able to exploit polymorphism and virtual functions.
    b. Many class members should be pointers for the same reason in part (a).

- Incremental implementation:
    a. Compile each class separately first before compiling the entire project.
    b. If class B depends on class A, don't move to class B before making sure that all errors in class A code have been corrected.

- Classes responsibilities:
  Each class should perform tasks that are related to its responsibilities only. No class performs tasks of another class.

- The only classes that have a **direct** access to the graphics library are **Input** and **Output** classes. Any class that needs to read from the input window should do so by calling functions of the class Input. Similarly, any class that needs to draw or print on the output window should do so by calling functions of the class Output. In summary, **any interaction with the GUI should be done through the I/O classes.**

- Save/Load
    a. User can save/load incomplete flowcharts.
    b. Save/load function is a **virtual** function in the class Statement. Each statement-derived class should override this function to save/load itself. Also a connector can save/load itself.
    c. Save/Load Action just opens the file and then calls ApplicationManager::Save/Load function.
    d. ApplicationManager then saves/loads the list of statements/connectors by calling save/load function of each statement/connector.

- Work load must be distributed among team members. A good way to divide work load is to assign some classes to each team member. A first question to the team at the project discussion and evaluation is"who is responsible for what?" An answer like" we all worked together" is a failure.

Example Scenario

The application window **may** look like the window in the following figure. (***Note***: the tool bar in the figure is not complete and you should extend it to meet the project requirements).

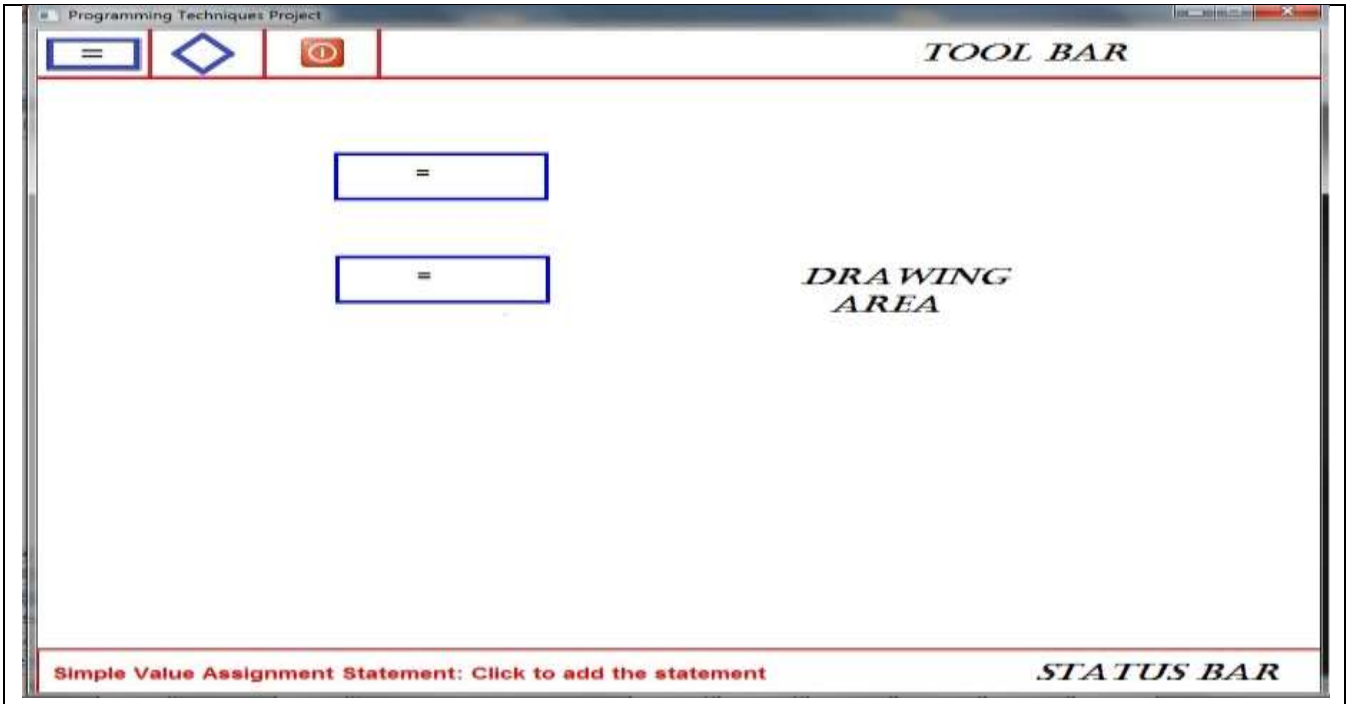

Figure 2 – Framework screenshot (here you can only add Simple Assignment)

Here is an example scenario for drawing a simple value assignment statement on the output window. It is performed through the four main steps mentioned in the previous section

Step I: Get user input

1- The ***ApplicationManager*** calls the ***Input*** class and waits for user action.
2- The user clicks on the "Simple Assignment Statement" tool to draw a simple assignment.
3- The ***Input*** class checks the area where the user clicked and recognizes that it is a "add simple assignment" operation. It returns **ADD_SMPL_ASSIGN** to the manager.

Step II: Create a suitable action

4- ***ApplicationManager::ExecuteAction*** is called. It creates an object of type ***AddSmplAssign*** action and calls ***AddSmplAssign::Execute*** to execute the action

Step III: Execute the action

5- ***AddSmpleAssign::Execute***
   a. Calls the ***Input*** class to get the statement position from the user. Here, to print a message to the user, ***Execute*** calls the Output::PrintMessage function.
   b. Creates a statement of type ***SmplAssign*** and asks the ***ApplicationManager*** to add it to the current list of Statements. (by calling AddStatement)

At this step, the action is complete but it is not yet reflected to the user interface.

Step IV: Reflect the action to the Interface (function ***ApplicationManager::UpdateInterface)***

6- ***ApplicationManager::UpdateInterface*** calls the virtual function ***Statement::Draw*** for each statement. (in this example, function SmplAssign::Draw is called)
7- ***SmplAssign::Draw*** calls ***Output::DrawAssign*** to draw assignment statement.

File Format

Your application should be able to save and load a flowchart from a simple text file. In this section, the file format is described together with an example and an explanation for that example. The application should enable the user to create a new flowchart or to load an existing one. If the user wants to load an existing flowchart, the application loads the flowchart from the required file. Otherwise, a new empty window is created.

- **Flowchart File Format**

```
Number_of_Statements
Statement_1_Type    Statement_ID    Stat _Graphics_info    Stat_Code_Info    Comment
Statement_2_Type    Statement_ID    Stat _Graphics_info    Stat_Code_Info    Comment
……..
……..
Statement_n_Type    Statement_ID    Stat _Graphics_info    Stat_Code_Info    Comment
Number of Connectors
Source_ Statement_ID    Target_ Statement_ID    Outlet_branch
……..
Source_ Statement_ID    Target_ Statement_ID    Outlet_branch
```

Notes:

- ❑ Outlet_branch is only applicable for connections going out of the conditional statements because conditional statement has two output branches. For other connections it is set to 0.

- **Example:** The flowchart shown in figure 2.a below is represented by the file in figure 2.b

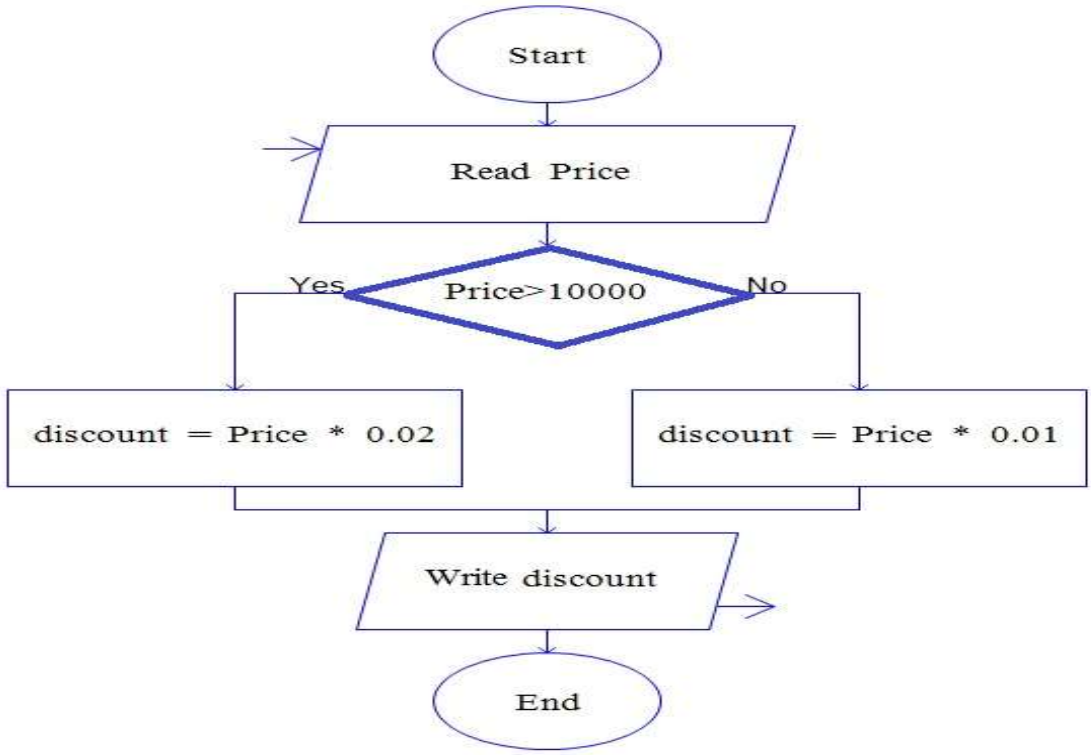

Figure 3.a – Flow Chart Diagram

```
7
STRT      1     200   90    ""
READ      2     150   145   Price   "get the price from the user"
COND      3     200   190   Price   GRT   10000   "Check Price to decide the discount"
SNGLOP    4     100   250   discount        Price   MUL   0.02   ""
SNGLOP    5     230   250   discount        Price   MUL   0.01   ""
WRITE     6     160   312   discount        ""
END       7     200   376   ""
7
1         2     0
2         3     0
3         4     1
3         5     2
4         6     0
5         6     0
6         7     0
```

Figure 3.b - File that represents the flow chart

Figure 2.b represents the flowchart in a text file. However, the order of the text file is not always the same as the order of the statements in the flowchart. The user usually creates the statements and the connections in any order and then saves the chart at any time. Figure 2.c gives another file that represents the flowchart in figure 2.a but just with different ordering.

```
7
READ      1     150   145   Price   "get the price from the user"
SNGLOP    3     100   250   discount        Price   MUL   0.02   ""
WRITE     19    160   312   discount        ""
STRT      10    200   90    ""
SNGLOP    15    230   250   discount        Price   MUL   0.01   ""
END       17    200   376   ""
COND      13    200   190   Price   GRT   10000   "Check Price to decide the discount"
7
10        1     0
3         19    0
19        17    0
1         13    0
13        3     1
13        15    2
15        19    0
```

Figure 3.c - File that represents the flow chart with different ordering

- **Explanation of the above example**

Here is the explanation of figure 2.b file (the same concept applies for figure 2.c)

7 //The flowchart has 7 statements

STRT 1 200 90 “”
//Start statement, ID=1, Center Point (200,90)

READ 2 150 145 Price “get the price from the user”
//Read statement, ID=2, Left Corner (150,145), Var to read: Price

COND 3 200 190 Price GRT 10000 “Check Price to decide the discount”
//Conditional statement, ID=3, Diamond upper Point (200,190), LHS: Price, Operator: greater_than, RHS: 10000

SNGLOP 4 100 250 discount Price MUL 0.02 “”
//Single Operator Assignment, ID=4, Left Corner (100,250), LHS: discount, RHS: Operator: multiplication, Operand1:Price, Operand2: 0.02

SNGLOP 5 230 250 discount Price MUL 0.01 “”
//Single Operator Assignment, ID=5, Left Corner (230,250), LHS: discount, RHS: Operator: multiplication, Operand1:Price, Operand2: 0.01

WRITE 6 160 312 discount “”
//Write statement, ID=6, Left Corner (160,312), Var to write :discount

END 7 200 376 “”
//End statement, ID=7, Center Point (200,376)

7 //Chart has 7 connectors

1	**2**	**0**	//Statement **1** (Start) is connected to Statement **2**(Read)
2	**3**	**0**	//Stat **2** (Read) is connected to Stat **3**(Conditional)
3	**4**	**1**	//Stat **3** (Conditional) is connected to Stat **4**(SNGLOP) through branch 1 (YES_branch)
3	**5**	**2**	// Stat **3** (Conditional) is connected to Stat **5**(SNGLOP) through branch 2 (NO_branch)
4	**6**	**0**	// Stat **4** (SNGLOP) is connected to Stat **6**(Write)
5	**6**	**0**	// Stat **5** (SNGLOP) is connected to Stat **6**(Write)
6	**7**	**0**	// Stat **6** (Write) is connected to Stat **7**(End)

Notes:

- The comment associated with each statement is written between double quotes (***“Comment”***). Empty quotes mean no comment.
- You can select any IDs for the statements. Just make sure ID is unique for each statement.
- You are allowed to add some modifications to this file format if necessary. But get approval from your instructor first.
- You can use numbers instead of text to simplify the "load" operation. For example you can give each statement type a number. This may be done by using ***enum*** statement in C.

- **Code File**
  As mentioned in the simulation mode, your application should be able to create a C++ code file that is equivalent to the flowchart. Here is an equivalent code file of the above example

```
#include <iostream>
using namespace std;
int main()
{
   double Price, discount;
   cin >> Price;  //get the price from the user
   if (Price>10000)     //Check Price to decide the discount
   {
      discount =Price*0.02;
   }
   else
   {
      discount =Price*0.01;
   }
   cout << discount;
   return 0;
}
```

Notes:

- ❑ It is required to support “double” variables only in your application.
- ❑ Any other equivalent valid C++ code is acceptable.
- ❑ Each statement should be able to write its code to the output code file through the virtual function “***Statement::GenerateCode***(…)”

Project Phases

1- Phase 1 (Input / Output Classes)

In this phase you will implement the input and the output classes because they don't depend on any other classes. The Input and Output classes should be **finalized** and ready to run and test.

Input and Output Classes Code and Test Code

You are given a code that contains both the input and output classes partially implemented. Each team should complete such classes as follows:

1- ***Input Class***:
- Complete the function Input::***GetValue*** to read a "double" value from the user.
- Complete the function Input::***GetUserAction*** where the input class should detect all possible actions according to the coordinates clicked by the user.
- Add any other needed member data or functions

2- ***Output Class***:
- Add/update member functions to:
  - Create the full tool bars. Output class should create two tool bars; design tool bar and simulation tool bar.
  - Draw different types of statements (Single Operand Assignment, Variable Assignment, Conditional, Read, Start,.. etc).
  - Draw each statement in all possible cases; normal, highlighted, empty and filled with code.
  - Draw connectors.
- Add any other needed member data or functions

3- **Test Code**: (this is NOT part of the input or the output class)
Each team should write a short test program that tests both Input and Output classes. Examples of required tests are
- Call Output class functions to draw the full interface (the full toolbar, the drawing area, and the status bar)
- Draw ALL possible statements in ALL possible cases.
- Ask the user to click on the toolbar and print the corresponding operation.

Deliverables*:*

Each team should upload on the Blackboard a compressed folder that contains IDs.txt, Input and Output classes and a test program.

2- Phase 2 (Project Delivery)

In this phase, the remaining classes should be implemented. Start by implementing the base classes then move to derived classes. To save time, work should be divided between team members.

Deliverables*:*

(1) **Workload division**: a printed page containing team information and a table that contains members' names and the classes each member has implemented.

(2) A compressed folder, should be uploaded on the Blackboard, that contains the following:
   a. ID.txt file. (Information about the team: name, IDs, team email)
   b. The work division document.
   c. The project code and resources files.
   d. Sample flowchart text files: at least three different charts.
   e. Sample code files equivalent to the above three text files.

Evaluation Criteria

The project is to be evaluated according to the following criteria

Basic Actions:

- (60%) Add, Edit statement, and Delete
- (40%) Connect, Edit connection, Comment and Select

Advanced Actions:

- (40%) Move
- (60%) Copy-Cut-Paste

Chart Save/Load

- (20%) Sticking to file format
- (80%) Each object saves/loads itself correctly

Flowchart Simulation

- (20%) Run the flowchart and produce final output
- (40%) Step-by-step and watch
- (40%) Interactive Run

C++ Code Generation

- (70%) Equivalent Code with no unnecessary ***goto*** statements
- (30%) Code indentation

GUI Input/Output:

- (40%) Input class
- (60%) Output class

Object Oriented Concepts

- (20%) Encapsulation
- (40%) Each class is doing its job. No class is performing the function of another class.
- (40%) Polymorphism: use of pointers and virtual functions

Integration & Run:

- (30%) No Compilation errors
- (20%) No Warnings
- (50%) No runtime errors

Code Organization & Style:

- (50%) Naming: variables, classes, constants, .. etc.
- (50%) Indentation & Comments

NOTE: Each of the above criteria will have its own weight. The summation of them constitutes the group grade (GG)

Individuals Evaluation:

Each student must be responsible for some classes and must answer some questions showing that he/she understands both the program logic and the implementation details. Each student will get a percentage grade (**IG**) according to this evaluation

The overall grade for each student will be the product (GG * IG).