Cairo University
Faculty of Engineering
CMP 103 & **CMP N103**

Fall 2020

بسم الله الرحمن الرحيم

"نرفع درجاتٍ من نشاء وفوق كل ذي علم عليم"

Programming Techniques Project Requirements

Logic Circuit Simulator

Fall 2020

Introduction

A logic circuit is a collection of interconnected logic components, such as gates, flip-flops, wires or even embedded circuits. Each component has a set of input pins and a set of output pins. A connection can be created between any two components by connecting an output pin of a component to an input pin of another component. It is not possible, however, to connect an output pin to another output pin, or an input pin to another input pin. If an output pin happens to source multiple input pins each connection should have a separate connector. Gates, on the other hand, are components with one or more input pins and only one output pin. Each circuit component manipulates the values applied to its input pins to calculate corresponding output. Two special components are the switch and the LED. The switch is a component with no input pins but has one output pin. The LED on the other hand has one input pin and zero output pin.

Complicated logic circuits can be very hard to trace, this is why computer-aided design of logic circuits is necessary. One of the very powerful tools is a simulator, which simulates the operation of an entire logic circuit and predicts the outputs based on the inputs.

Your Task:

You are required to implement a simple logic circuit 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 logic circuit using different components, connect them with connections, add switches to control inputs and LEDs to monitor outputs, save and load a circuit, and so on. The application should accept input from the user as mouse clicks or key strokes and then take actions according to the input.

NOTE: The application should be designed so that the types of components and types of 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 Week***
Phase 1	Input-Output Classes	Week 7
Phase 2	Final Project Delivery	Week 12

Note: At any delivery:

One day late makes you lose 1/4 of the grade.
Two days late makes you lose 1/2 of the grade.
Three 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 the following operations (actions) should be supported:

1- ***Add*** a new component to the circuit. This includes
- ❑ Adding a new gate: The application should support, at least, the following gates
  - i. Buffer gate and inverter gate
  - ii. 2-input AND, OR, NAND, NOR, XOR, XNOR gates
  - iii. 3-input AND, NOR, XOR
- ❑ Adding a new **switch** (a switch is a component with one output pin and no input pins)
- ❑ Adding a new **LED** (a LED is a component with one input pin and no output pins)

2- ***Connect*** two components: this means to create a connection from an output pin of a component to and input pin of another component. To connect one output pin to many input pins, a separate connection should be created between the output pin and each of the input pins.

3- ***Label*** a component or a connection.

4- ***Edit*** a component or a connection: To edit a component is to edit its label. To edit a connection is to edit its label or to change its source or destination pins

5- ***Select***/***Unselect*** one of the components. When the user clicks on one of the components, it should be highlighted.

6- ***Delete*** an existing component: when deleting a component all its connections should be automatically deleted. Also, deleting a separate connection should be supported.

7- ***Move*** a component: user can drag a component to move it. All its connections should be moved with it.

8- ***Save*** a circuit to a file (see file format section).

9- ***Load*** a circuit from a file (see file format section).

10- ***Copy-Cut-Paste*** a component (not applicable for connections)
- ❑ Copying a component copy only the component without its connections.
- ❑ Cutting a component is to erase the original component with its connections then paste the component with no connections.

11- ***Multiple-Selection*****:** User can select multiple components (including connections) to perform multi-move or multi-delete. All selected components must be highlighted.

12- ***Switch to simulation mode*****.** (only if the circuit is fully and correctly connected)

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

Important:

Application should limit user drawings to the design area only. No drawing on the toolbar or the status bar is allowed.

[II] Simulation Mode:

Operations supported by this mode are:

1- ***Circuit Validation*:** To simulate a circuit it must be fully connected and no pins are left floating.

2- ***Simulate circuit*:** user can change the circuit inputs by changing the status of input switches. Any changes in the circuit inputs in this mode should be reflected at the output LEDs.

3- ***Create Truth Table***: The application must be able to create the truth table of the designed circuit. It should be able to display the truth table for up to 5 inputs. For circuits with inputs greater than 5, the truth table should be saved to a text **file**.

4- ***Circuit Probing*:** a probing tool is used by the user to check the value of any pin or connection in the circuit.

5- ***Switch back to Design Mode***

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 other operations are.

Bonus Operations:

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

1- **Neat Layout**: Connections should be drawn as vertical and horizontal lines only and should not overlap. Components also should not overlap.

2- **Complex Circuit Blocks**: User can select part of the designed circuit and convert it into a block diagram that operates exactly as the selected components.

3- **Undo/Redo** actions

4- **Detecting Circuit Errors**: To detect the following errors
   - ❑ First circuit level contains components other than switches
   - ❑ Last circuit level contains components other than LEDs
   - ❑ A feedback in the circuit

Main Classes

Because this is your first object oriented application, you are given a ***code framework*** where we have ***partially*** written the code of some of the project classes. For the graphical user interface (GUI), we have integrated an open-source ***graphics library*** that you will use to easily handle GUI.

You should **stick to** the given design and complete the given framework by either: extending some classes or inheriting from some classes (or even creating new base classes but after instructor approval)

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

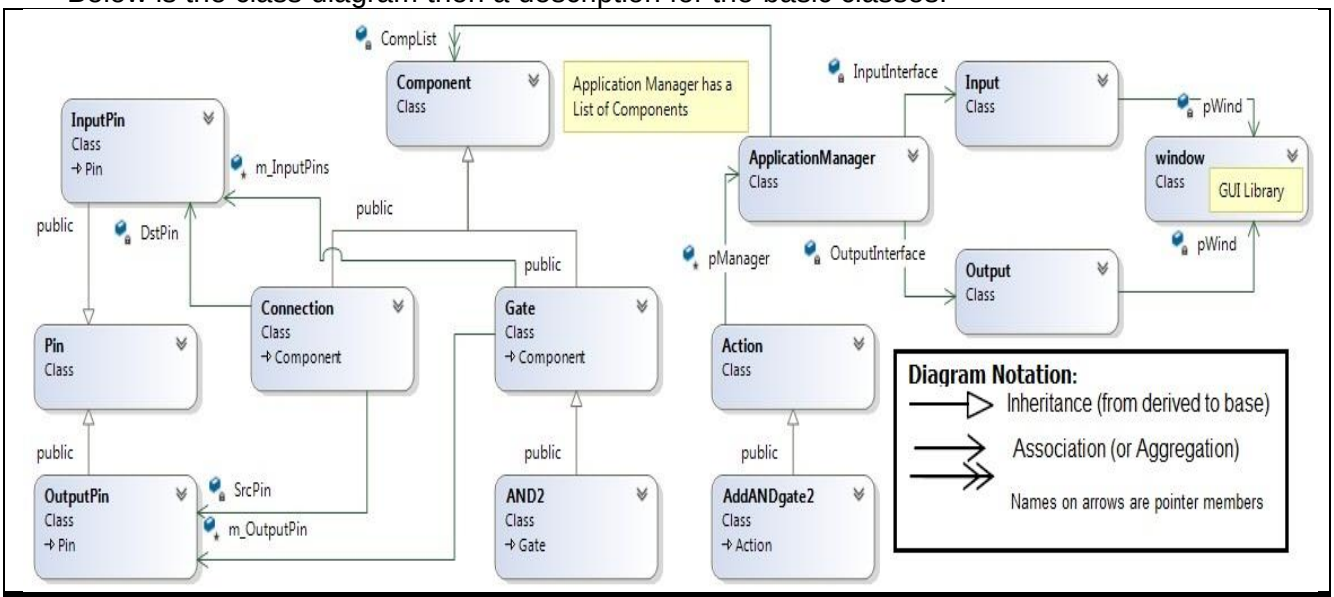

Figure 1 – Class Diagram of the Application

Input Class:

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:

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

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 components inside the application.

Pin Class:

This is the base class for input and output pins (**InputPin** class and **OutputPin** class).

Component Class:

This is the base class for all types of circuit components (switches, gates, LEDs, and connections). To add a new component (a new gate for example), you must **inherit** it from this class. Then you should override virtual functions found in the class **Component**. You can also add more details for the class Component itself if needed.

Action Class:

This is the base class for all types of actions (operations) to be supported by the application. To add a new action, you must **inherit** it 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 action type.
  b. Create suitable action object 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 the **Input** and the **Output** classes. Any class that needs to read input from the input window should do that by calling functions of the class Input. Similarly, any class that needs to draw or print on the output window should do that by calling functions of the class Output. In summary, **any interaction with the user interface should be done through the I/O classes.**

- All gates classes **should** be implemented **similar** to the **AND2** class implementation. You **must inherit Gate** class.

- Save/Load
  a. User can save/load incomplete designs.
  b. Save/load function is a **virtual** function in the class Component. Each derived class should override this function to 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 components/connectors by calling save/load function of each Component.

- 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

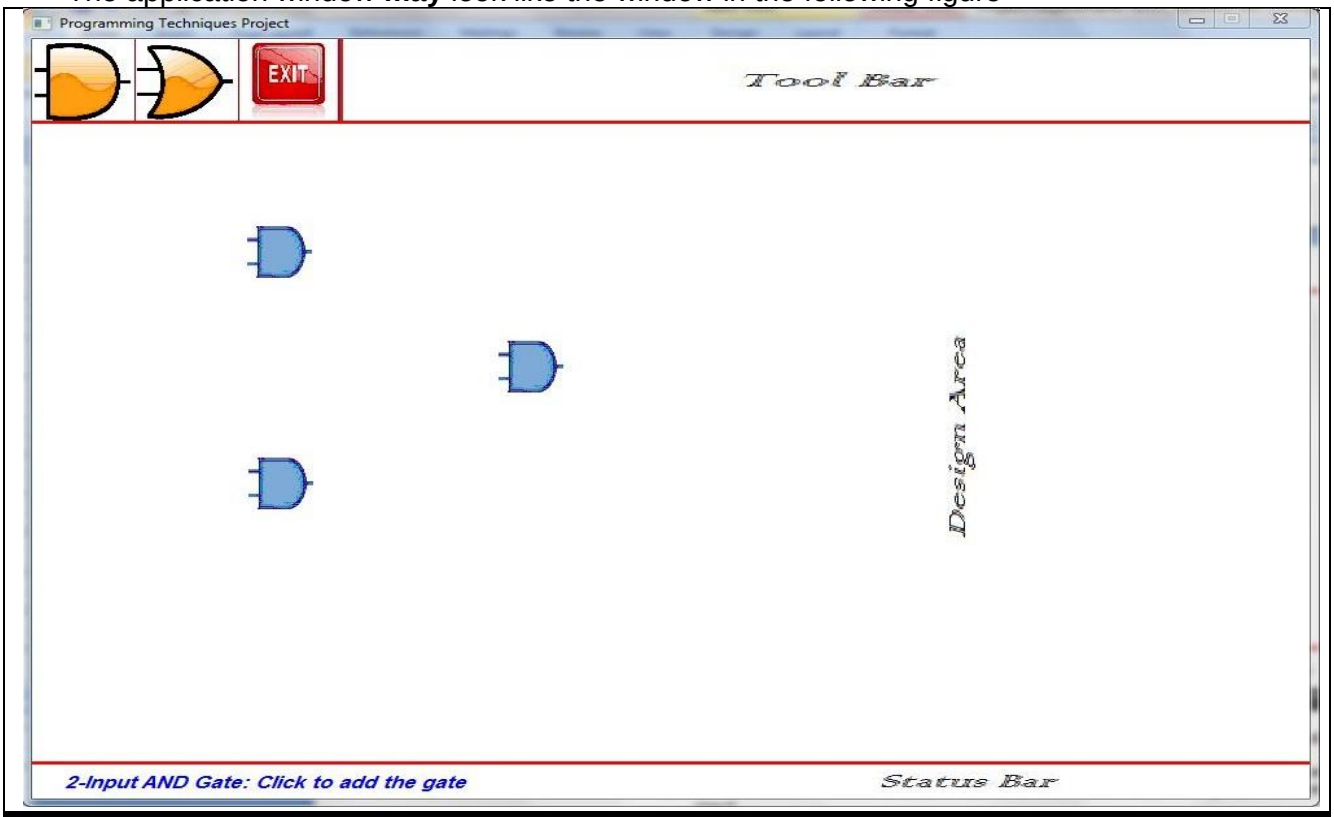

Figure 2 – Application Screenshot

Here is an example scenario for adding an AND gate and drawing it on the output window. It is performed through the four main steps mentioned in the previous section (see main( ) function in the given framework code)

Step I: Get user action

1- The ***ApplicationManager*** calls the ***Input*** class and waits for user action.
2- The user clicks on the "AND gate" icon in the toolbar
3- The ***Input*** class checks the area where the user clicked and recognizes he wants to add AND gate to the circuit. It returns ***ADD_AND_GATE_2*** (a constant indicating the required action) to the manager.

Step II: Create a suitable action object

4- ***ApplicationManager::ExecuteAction*** is called. It creates an object of type ***AddANDgate2*** action and calls ***AddANDgate2::Execute*** to execute the action.

Step III: Execute the action

5- ***AddANDgate2::Execute***
   - ***a.*** Calls the ***Input*** class to get the gate position from the user. Here, to print a message to the user, ***Execute*** calls the Output::PrintMsg function.
   - ***b.*** Creates a Component of type ***AND2*** and asks the ***ApplicationManager*** to add it to the current list of Components. (by calling AddComponent)

Note: 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 ***Component::Draw*** for each Component. (in this example, function AND2::Draw is called)
7- ***AND2::Draw*** calls ***Output::DrawAND2*** to draw AND gate.

File Format

Your application should be able to save and load a circuit 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 circuit or to load an existing circuit. If the user wants to load an existing circuit, the application loads the circuit from the required file. Otherwise, a new empty window is created.

- **File Format**

```
Number_of_Components
Comp_1_Type     Comp_ID     Label   Component_Graphics_info
Comp_2_Type     Comp_ID     Label   Component_Graphics_info
……..
……..
Comp_n_Type     Comp_ID     Label   Component_Graphics_info
Connections
Source_Comp_ID          Target_Comp_ID      Pin_number
-1
```

- **Example:** The circuit shown in figure 3.a below is represented by the file in figure 3.b

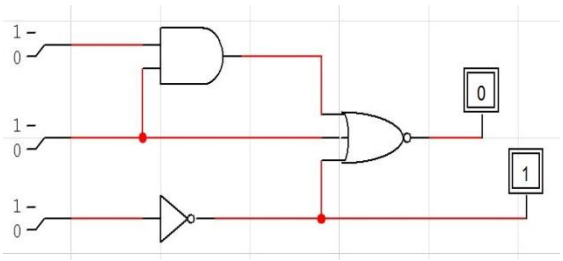

Figure 3.a - Circuit Diagram

```
8
SWTCH   1   sw1     90      154
SWTCH   2   sw2     90      227
AND2    4   $       170     159
NOR3    6   out     290     235
SWTCH   13  sw3     90      330
LED     8   L1      406     276
NOT     5   gate2   173     343
LED     7   Result  382     231
Connections
1       4       1
2       4       2
2       6       2
3       5       1
4       6       1
5       6       3
5       8       1
6       7       1
-1
```

Figure 3.b - File that represents the circuit

- **Explanation of the above example**

  Here is the example and the explanation of each line

```
8  //Circuit has 8 components

SWTCH   1     sw1   90    154
//Switch, ID=1, label="sw1", its rectangular area corner is (90,154)

SWTCH   2     sw2   90    227
//Switch, ID=2, label="sw2", its rectangular area corner is (90,227)

AND2    4     $     170   159
//2-input AND, ID=4, label=No Label, its rectangular area corner is (170,159)

NOR3    6     out   290   235
//3-input NOR, ID=6, label="out", its rectangular area corner is (290,235)

SWTCH   3     sw3   90    330
//Switch, ID=3, label="sw3", its rectangular area corner is (90,330)

LED     8     L1    406   276
//LED, ID=8, label="L1", its rectangular area corner is (406,276)

NOT     5     gate2 173   343
//NOT, ID=5, label="gate2", its rectangular area corner is (173,343)

LED     7     Result 382  231
//LED, ID=7, label="Result", its rectangular area corner is (382,231)

Connections   //Start of connections part
1       4     1  //Component 1 (Switch) is connected to Comp 4 (AND2) at pin# 1
2       4     2 //Comp 2 (Switch) is connected to Comp 4 (AND2) at pin# 2
2       6     2 //Comp 2 (Switch) is connected to Comp 6 (NOR3) at pin# 2
3       5     1 //Comp 3 (Switch) is connected to Comp 5 (NOT) at pin# 1
4       6     1 //Comp 4 (AND2) is connected to Comp 6 (NOR3) at pin# 1
5       6     3 //Comp 5 (NOT) is connected to Comp 6 (NOR3) at pin# 3
5       8     1 //Comp 5 (NOT) is connected to Comp 8 (LED) at pin# 1
6       7     1 //Comp 6 (NOR3) is connected to Comp 7 (LED) at pin# 1
-1   //End of file signal
```

Notes:

- ❑ You can select any IDs for the components. Just make sure ID is unique for each component.
- ❑ You are allowed to add some modifications to this file format if necessary. But before adding such modifications, get the approval from your instructor.
- ❑ You can use numbers instead of text to simplify the "load" operation. For example you can give each component type a number. This is **must** be done by using **ComponentType *enum*** in DEF.h file.

Project Phases

1- Phase 1 (Input and Output Classes) [15% of total project grade]

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. Any expected user interaction (input/output) that will be needed by phase 2, should be implemented at this phase.

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::***GetString*** to read a string 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 with full items.
  - Draw different types of Components (OR2, NAND2, Switch, LED,.. etc).
  - Draw each component in all possible cases; normal, highlighted.
  - Draw connections.
- Add any other needed member data or functions

3- **Test Code**: (this is NOT part of the input or the output class)
Complete the code given in TestCode.cpp file to test both Input and Output classes.

Deliverables*:*

Each team should:
1. Deliver a CD that contains IDs.txt, Input and Output classes and a test program
2. Submit to Elearn Platform a compressed file containing the same data using the contact email provided in teams' sheet. Folder name should be **TYP-T#-P#.zip**, where TYP is SEM for semester or CRD for credit, T# is your team number and P# is current phase number. For example: SEM-T1-P1.zip for team 1 semester – phase 1, or CRD-T4-P2.zip for team 4 credit – phase 2.

2- Phase 2 (Project Delivery) [85% of total project grade]

In this phase, the I/O classes should be added to the project framework code and 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 containing members' names and the classes each member has implemented.

(2) A compressed folder (through Elearn) and A CD. Both should contain the following
   a. ID.txt file. (Information about the team: name, IDs, team email)
   b. The workload division document.
   c. The project code and resources files.
   d. Sample circuit files: Three different circuits. For each circuit, provide:
      i. Circuit text file
      ii. Circuit screenshot for the circuit generated by your program
      iii. Screenshot of one case of circuit simulation
      iv. Circuit truth table generated by the program.

On CD cover, each team should write: semester or credit, team number and phase number.

Final Phase Evaluation Criteria

Note: Number of students per team = 3 to 4 students

Basic Actions [30%]

- (50%) Add (create): components and connections
- (50%) Label, Delete, Edit, and Select

Advanced Actions [15%]

- (20%) Move
- (20%) Copy-Cut-Paste
- (60%) Multiple selection, Multiple delete, Multiple move

Circuit Save/Load [5%]

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

Circuit Simulation [25%]

- (20%) Circuit Validation
- (50%) Simulate Circuit Operation
- (20%) Truth table
- (10%) Circuit Probing

GUI Input/ Output [5%]

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

Object Oriented Concepts [10%]

- (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 [7%]

- (30%) No Compilation errors
- (20%) No Warnings (other than graphics library warnings, if any)
- (50%) No runtime errors

Code Organization & Style [3%]

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

Bonus Operations [max 10%]

The total grade of all bonus operations can never exceed 10% of project grade

Individuals Evaluation (IG):

Each team member 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 member will get a percentage grade (**IG**) of the total team grades according to this evaluation.

Note: we will reduce the IG in the following cases:

- Not working enough
- Neglecting or preventing other team members from working enough

** Cheating results in zero grade for **all teams involved**.