Mini-project topics
Object-Oriented Programming

Lecturer: NGUYEN Thi Thu Trang, trangntt@soict.hust.edu.vn

Important note:

- For each topics provided below, it is compulsory for you **strictly** follow all the **specifications** stated below. The reference links are there to clarify/illustrate for the specifications only, in case of conflict between the specifications and the reference, the specifications should be prioritized.
- Some topics may require a bit more field-specific knowledge than others (topics related to physics, electronics, …). In which case, we have provided summarizations and formulae for you to quickly and aptly grasp the knowledge needed. This way, you can focus on the object-oriented aspect of the project, which is the most important goal.

Caution: Since we will only assess the use of object-oriented programming of this project, you should not spend too much time on interface design, which won't be graded.

How to conduct mini-projects:

- You will conduct mini-projects with your teammates. Each team must have from 2 to 3 members.
- You are free to choose your teammates and submit to the monitor with one of the 9 below mini-project topics. Deadline:
    - Teams submit to monitor: 30 April 2021
    - Monitor submits to professor: 1 May 2021
- Each topic must be chosen by 1 to 2 teams. The monitor may play a random game for allocating the topic to teams if there is any conflict.
- Your mid-term score will contribute 50% to your grade of the course. Your mini-project will be 50% for the mid-term score, 50% for labs.
- You will be given an individual score for the mini-project although you do it with your teammate. Please remember to clarify the responsibility of each member of your team, otherwise, your teamwork score will be very low.

1. Demonstration of operations on stack, queue, list

Overview: stack, queue, list are basic data structures that are frequently used in computer science. You will design a program to display and explain some basic operations on these structures.
Basic knowledge: create, insert, sort, find, delete elements in the structure
Specifications:
- GUI: you can freely design the GUI, however, there is no need to focus too much on the interface since the main aim of the project is to use OOP to design the application
You can refer to this source for some idea: https://visualgo.net/en/
- Design:
+ On the main menu: title of the application, options for user to choose between the three types of data structure, help menu and quit

- User must select a type of data structure before getting into the demonstration
- The help menu show basic usage and aim of the project
- Quit button exits the application. Remember to ask for confirmation

+ In the demonstration
  - The main demonstration shows options for user to choose: create, insert, sort, find and delete elements in the structure. You can choose to put it in a dropdown list or separate buttons. Remember to process input if needed.
  - After user has picked a choice, demonstrate it on the screen. For simplication, you do not need to implement revert or slow down function, but put the speed of demonstration at average.
  - Always have a Back button for user to return to main menu at any time

Note: In this project, you should build your own data structure to demonstrate your OOP design.

2. Visualization of operations on tree data structures

Overview: Tree is a useful data structure with lots of applications in computer science. In this project, you will design a program to display and explain some basic operations four types of tree.

Basic knowledge:

- o Generic tree (no special properties): A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges and contains no cycles.
- o Binary tree: A binary tree is a tree where each node has at most two children
- o Balanced tree: A balanced tree is a tree where each leaf node is "not more than a certain distance" away from the root than any other leaf.
- o Balanced binary tree: A balanced binary tree has the properties of both a binary tree and a balanced tree

Operations on the above tree structures: create, insert, delete, update, traverse.

Specifications:

- GUI: You can refer to this source for some idea: https://visualgo.net/en/bst

- Design:

+ We only consider undirected-weight trees, with integer node values and no duplicated node values allowed.

+ For the balanced tree and balanced binary tree, the maximum difference in distance from root of the leaf nodes must be chosen by the user

+ On the main menu: title of the application, navigation bar for user to choose between the four types of tree, help menu and quit

- User must select a type of data structure before getting into the visualization
- The help menu show basic usage and aim of the project
- Quit button exits the application. Remember to ask for confirmation

+ In the visualization

- User can choose to visualize one of six operations, by selecting an option on the operations menu, and then provide the necessary parameter.

  The description of the operations is as following:

Operations	Parameters	Description

Create	None	Create a new empty tree
Insert	Value of parent node, value of new node	Add the new node with specified value as a child of the specified parent node
Delete	Value of the node	Delete the node from the tree
Update	Current value of the node, new value of the node	Change the node with current value to new value
Traverse	Algorithm (DFS or BFS)	Traverse all node in the tree (highlight current node in each step of traversal)
Search	Search value	Search for the node value in the tree

- When an operation starts to execute, on the code panel, the pseudo code (or actual code) should be displayed, and the currently executing line is highlighted to help the user keep track of the process. On the bottom bar, the user can see the progress bar of the executing operation and choose to pause, continue, or go backward or forward a step in the execution.
- The user can also undo or redo operations from the bottom bar.
- Always have a Back button for user to return to main menu at any time

3. <u>Demonstration of sorting algorithms on array (1)</u>

<u>Overview</u>: Array is the most basic structure of computer science. Most operations as well as other data structures are built and performed on array. In this project, you will make an application in order to explain threee sorting algorithms on array: selection sort, merge sort and shellsort.
<u>Basic knowledge</u>: selection, merge and shellsort on array
<u>Specifications</u>:
- GUI: you can freely design your own GUI. However, since the basic aim of the project is to develop an application based on OOP, focus on interface is not required
You can refer to these sources to have some idea:
https://www.youtube.com/watch?v=xWBP4lzkoyM
https://www.youtube.com/watch?v=JSceec-wEyw
https://www.youtube.com/watch?v=SHcPqUe2GZM
- Design:
+ On the main menu: title of the application, 3 types of sort algorithms for user to choose, help menu, quit

- User must select a sort type in order to start the demonstration
- Help menu show the basic usage and aim of the program
- Quit exits the program. Remember to ask for confirmation

+ In the demonstration

- A button for creating the array: User can choose to randomly create an array or input an array for the program
- A button for starting the algorithm with the created array. Remember to show clearly each step of the sorting
- A back button for user to return to main menu at any time

Note: You MUST use pure array for this project. If you use any Java implementations, you must design your own wrapper to show your OOP design in the project
Hints: In order to design this program with OOP method, treat each sort type as a class – not array. By this way, you will find there are some similarities between the sortings (input, functions, attributes, etc.)

4. Demonstration of sorting algorithms on array (2)

Overview: Array is the most basic structure of computer science. Most operations as well as other data structures are built and performed on array. In this project, you will make an application in order to explain three sorting algorithms on array: bubble sort, quicksort and insertion sort.
Basic knowledge: selection, merge and shellsort on array
Specifications:
- GUI: you can freely design your own GUI. However, since the basic aim of the project is to develop an application based on OOP, focus on interface is not required
You can refer to these sources to have some idea:
https://www.youtube.com/watch?v=nmhjrI-aW5o
https://www.youtube.com/watch?v=PgBzjlCcFvc
https://www.youtube.com/watch?v=OGzPmgsI-pQ
- Design:
+ On the main menu: title of the application, 3 types of sort algorithms for user to choose, help menu, quit

- User must select a sort type in order to start the demonstration
- Help menu show the basic usage and aim of the program
- Quit exits the program. Remember to ask for confirmation

+ In the demonstration

- A button for creating the array: User can choose to randomly create an array or input an array for the program
- A button for starting the algorithm with the created array. Remember to show clearly each step of the sorting
- A back button for user to return to main menu at any time

Note: You MUST use pure array for this project. If you use any Java implementations, you must design your own wrapper to show your OOP design in the project
Hints: In order to design this program with OOP method, treat each sort type as a class – not array. By this way, you will find there are some similarities between the sortings (input, functions, attributes, etc.)

5. Interactive simulation of composition of forces

Overview: Newton's laws of motion are the foundation of dynamics. These laws provide an example of the breadth and simplicity of principles under which nature functions. In this project, you will create a simple interactive simulation for demonstrating Newton's laws of motion.
Basic knowledge: Gravitation force, normal force, friction, Newton's laws of motion, Newton's law for rotation
Specifications:

- GUI: You can refer to this source: http://phet.colorado.edu/sims/html/forces-and-motion-basics/latest/forces-and-motion-basics_en.html ("Acceleration" module)
- Design: The design of this simulation is closely similar to the one in the reference above
+ In the simulation, the user controls a physical system. The system includes three components: one main object, the surface (which is always horizontal) and an actor who can apply a horizontal force on the object. The user can control all the components of the physical system and observe the motion of the main object, specifically:

- The main object: The user has two option for the main object, either a cube-shaped object or a cylinder-shaped object. For each type of object, the user can also specify the desired parameter as follows:

Object type	User controlled parameters
Cube-shaped object	- Side-length (cannot exceed a maximum threshold) - Mass
Cylinder-shaped object	- Radius (cannot exceed a maximum threshold) - Mass

  To set up the main object in the system, the user can drag an option from the object menu on the bottom left onto the surface, then click on the object, and provide the parameters in the context menu that pops up.
- The actor: The actor always applies force on the center of mass of the main object. Unlike the reference, you don't need to represent the actor with an actual figure, instead, instead, the actor is represented by the force it applies. This force can be represented with a horizontal arrow. To control the strength and direction of the applied force, the user can use the bottom center panel by using the sliding bar or specify the number of Newtons in the textbox.
- The surface: The user can control the friction coefficients of the surface in the bottom right panel. There are two friction coefficients: static friction coefficient and kinetic friction coefficient. For each coefficient, the user can control its value through a sliding bar and a text box. Note that the value of the static coefficients must be higher than the value of the kinetic coefficient.

+ To simulate motion, we recalculate the position of the main object after each time interval Δt.
+ Throughout the motion simulation process, the user can
- Change the applied force as well as the friction coefficients of the surface.
- Pause, continue and reset the simulation.
- Choose to show or hide detailed information such as the forces, the sum of forces, the values of forces, the mass, speed and acceleration of the main object through the corresponding tick-boxes on the panel on the upper right:
  - The forces and sum of forces are displayed as arrows like in the reference.
  - The masses are displayed as text on the main object like in the reference.
  - The speed and acceleration are displayed as text on the upper left corner.

Knowledge summary:
The solution and formulae to calculate the motion of the main object will be presented below:
The main object is under the effect of four forces:

Forces	Applied point	Direction	Value
Gravitational force	Center of mass	Vertical, downward	Mass x 10

Normal force	Point of contact with the surface	Vertical, upward	Equal to gravitational force
Actor's applied force	Center of mass	Horizontal	
Friction	Point of contact with the surface	Horizontal, opposite direction of motion	Depending on the applied force and the shape of the main object

The value of friction is dependent on the value of the applied force as follows:

Cases of applied force	Object shape	Value of friction
Applied force <= (Normal force x Surface's static friction coefficient)	Cube	Equal to applied force
Applied force > (Normal force x Surface's static friction coefficient)	Cube	(Normal force) x (Surface's kinetic friction coefficient)
Applied force <= 3 x (Normal force x Surface's static friction coefficient)	Cylinder	Equal to applied force / 3
Applied force > 3 x (Normal force x Surface's static friction coefficient)	Cylinder	(Normal force) x (Surface's kinetic friction coefficient)

The position of the main object after each time interval Δt is calculated as follows:

- The translational motion of the main object is calculated as follows (assuming x, v, x', v' are the previous position, the previous velocity, the new position and the new velocity of the main object, respectively):

$$a = \frac{|Applied\ force - friction|}{mass}$$
$$v' = v + a * \Delta t$$
$$x' = x + v * \Delta t$$

- If the main object is cylinder-shaped, it also has an additional rotation motion. The rotation motion is calculated as follows: (assuming θ, ω, θ', ω' are the previous angular position, the previous angular velocity, the new angular position and the new angular velocity of the main object, respectively):

$$\gamma = \frac{Friction}{\frac{1}{2} mass * radius^2}$$
$$\omega' = \omega + \gamma * \Delta t$$
$$\theta' = \theta + \omega * \Delta t$$

6. <u>Traditional game: Ô ăn quan</u>

<u>Overview & gameplay</u>: https://hocvienboardgame.vn/huong-dan-tro-choi-o-an-quan/
<u>Demo game</u>: http://choinhanh.vn/game-tri-tue/o-an-quan
<u>Specifications</u>:
In this project, you will make an application for 2 users to play the traditional game Ô ăn quan.
- GUI: You can freely design your own UI; however, since the aim of this project is to design a program using OOP, there is no need to pay too much focus on the interface. You can also use some image references of the web game in the link provided, or look for any materials you like.
- Design: The application must have these functions:

+ On the main screen:

- Start: start the game. For convenient, you do not have to create different difficulties
- Exit: exit the program. Be sure to ask users if they really want to quit the game
- Help: Show guide for playing the game

+ In the game:

- Gameboard: The gameboard consists of 10 squares, divided into 2 rows, and 2 half-circle on the 2 ends of the board. Initially, each square has 5 small gems, and each half-circle has 1 big gem. Each small gem equals 1 point, and each big gem equals 5 points.
- For each turn, the application must show clearly whose turn it is. A player will select a square and a direction to spread the gems. He got points when after finishing spreading, there is one empty square followed by a square with gems. The score the got for that turn is equal to the number of gems in that followed square (see the gameplay for more details about streaks)
- The game ends when there is no gem in both half-circles. The application must notify who is the winner and the score of each player.
- For simplicity, you do not have to build a bot to play with human

7. Demonstration of types of viruses and its mechanism

Overview: COVID-19 has been spreading all over the world and there is the need of understanding different type of viruses, as well as the way they infect in order to have the basic knowledge to prevent them.

Basic knowledge:

- Basic structure of virus:

+ Every virus has 2 basic elements: acid nucleic and capsid.

+ Based on their structure, viruses are divided into 2 categories: with and without lipid envelop.

- Virus without envelop will dissolve its capsid when reach the target cell
- Virus with envelop usually has anchors, called glycoprotein. The mechanism for infecting in this case is by lock – key: when reaching the host cell with the suitable outer structure, it uses its glycoproteins to attach, then injects its acid nucleic into the cell

You can refer to these materials for more information:
https://en.wikipedia.org/wiki/Virus#Structure
https://www.vinmec.com/vi/tin-tuc/thong-tin-suc-khoe/suc-khoe-tong-quat/dac-diem-cau-tao-cua-virus-gay-benh/
https://opentextbc.ca/microbiologyopenstax/chapter/the-viral-life-cycle/

Specifications:

- GUI: You can freely design the GUI with your favor. However, this project focus on structuring the application with OOP design; therefore, too much focus on the interface is not necessary

- Design: the application must have these functions:

+ On the main screen: Title of the application, options to choose between virus with lipid envelop and virus without lipid envelop, help menu and quit

- User can choose to investigate one of the two types of viruses in the main menu to start the application
- After choosing the desired type, the application will show a variety of viruses in order for user to select (for example, after choosing virus with lipid envelop, the application

displays 2 viruses: HIV, COVID and Rotaviruses for the user to choose. The choice of viruses to demonstrate depends on you)

- The help menu shows basic usage and aim of the application
- The quit button exits the application. Be sure to ask for confirmation

+ In the demonstration:

- Display the structure of the virus. Note that each virus has different structure, you should clearly display and explain them.
- One button to start demonstrating the progress of virus infecting the host cell. Different viruses have the same basic mechanism of spreading with minor difference - remember to show that

There is always return button for user to get back to the main menu at any time.

8. <u>Electrical circuit simulator</u>

<u>Overview</u>: In this project, you will create a simple electrical circuit simulator that let users build an electrical circuit and perform circuit analysis on it.

<u>Basic Knowledge</u>: Electrical circuit elements: resistors, capacitors, inductors, Ohm's law, DC voltage source, AC voltage source

<u>Specifications</u>:

- GUI: You can refer to this source: https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-parallel-and-series-resistor

- Design:

+ This simulator will demonstrate two types of circuits: parallel circuit and serial circuit. To pick a type of circuit, the user can choose one or two tabs in the navigation bar, like in the reference above. Then, the user can start to construct a circuit and press submit when they are done.

+ After the user press submit, the application will draw the circuit diagram, output the circuit analysis results in a table, and calculate the equivalent resistance.

+ Circuit construction: Each circuit will allow the user to pick a single voltage source and several electrical elements. There are three types of electrical element: resistor, capacitor, and inductor. Likewise, there are two types of source, AC and DC:

- To add an element, the user can pick one of three buttons: add resistors, add capacitor, or add inductor. After a button is picked, a row is added after the last row in the panel for the user to specify the parameter of the new element, like in the reference. The parameters corresponding to each element:

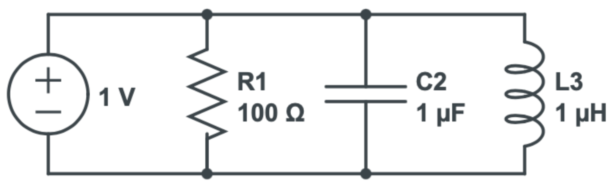
Element type	Parameter	Unit	Name prefix
Resistor	Resistance	Ω	R
Capacitor	Capacitance	F	C
Inductor	Inductance	H	L

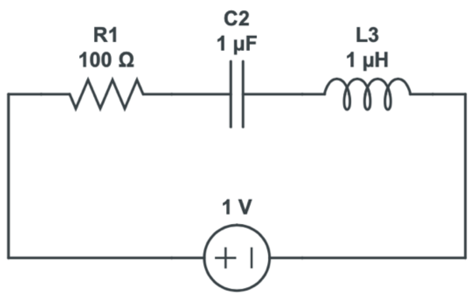
  Note that each new element must be named according to a specific convention: The prefix is a letter depending on the type of the element (as specified in the table above), and the remaining part of the name is a number that is incremented for each new element. For example, if the user adds two resistors in a row and then an inductor, their names will be R1, R2, L3, respectively. A maximum of 5 elements can be added.

- To pick the source, the user first chooses the type of source (AC or DC) from a dropdown button and based on the option, some text boxes appear next to the dropdown for the user to provide the associated parameters for the type of source chosen:

Type of source	Parameters (unit)
DC	Voltage(V)
AC	Voltage(V), frequency (Hz)

+ Circuit diagram: After the user has constructed the circuit and press the submit button, the simulator will show the corresponding circuit diagram. You are free to choose any formyou're your diagram, but here is a suggestion:





- For the parallel circuit, each element will be in its own vertical branch and the source will be in the left-most branch.
- For the serial circuit, each element will be placed serially in the same horizontal upper branch and only the source will be in the lower branch. This diagram can also be inspected, when clicking an element, it will show the current and voltage at that element.
- For both types of circuit, each element will be shown with their name and their parameter values.

+ Circuit analysis:
The circuit analysis table's format is as follows:

	R1	C2	L3	
U (Voltage)				V
I (Current intensity)				A
R (Resistance)				Ω

You will need to calculate values in the circuit to fill in the empty cells in the above table.
+ Short circuit: If a short circuit is detected, you must inform the user of the element causing the short circuit and ask them to change it.
<u>**Knowledge summary**</u>:
The solution and formulae to calculate the circuit analysis table and the equivalent circuit will be presented below. Note that for these formulae, complex number is used, so you have to implement by yourself a complex number class in Java to apply these formulae (denote $\hat{\imath}$ as the imaginary unit):

- For the R (Resistance) row, the values are calculated as in the table below

Element type	R (Resistance)	
	AC ($f \neq \infty$)	DC ($f = \infty$)
Resistor	R	
Capacitor	$R_c = -2\pi f L \hat{\imath}$	$R_c = \infty$
Inductor	$R_l = \dfrac{1}{2\pi f C}\hat{\imath}$	$R_l = 0$

- For the U (Voltage) and I (Current intensity) rows and the equivalent resistance:

- For the serial circuit:

Equivalent resistance: $R_{eq} = \sum R_i$

I (Current intensity) row: $I_i = \frac{V_{source}}{R_{eq}}$

U (Voltage) row: $U_i = I_i \times R_i$

- For parallel circuit:

If the user uses an inductor element in a parallel DC circuit, notify them that it will cause a short circuit and ask them to change that circuit element.

U (Voltage) row: $U_i = V_{source}$

I (Current intensity) row: $I_i = \frac{V_{source}}{R_i}$ $, if\ branch\ i\ is\ a\ resistor$

$I_i = 0$ $, if\ branch\ i\ is\ a\ capacitor$

Equivalent resistance: $R_{eq} = \frac{1}{\sum \frac{1}{R_i}}$ $, where\ branch\ i\ is\ a\ resistor$

In the above formulae, in any case there is a divide by zero (because a resistance value is equal to zero), it means that there is a short circuit. Another case to consider is divide by infinity, in such cases the result is assumed to be 0.

9. <u>Logic expression normalizer:</u>

<u>Overview</u>: Logic expression normalization (simplification) is an important step to reduce the original expression into a canonical normal form with fewer number of terms and operations. This way, the same expression can be implemented using fewer logic gates wich means higher reliability and lower manufacturing cost. In this project, you will create a normalizer for logic expression with using the Quine-McCluskey method.

<u>Basic Knowledge</u>: logic expression, canonical normal form: SOP & POS, Quine-McCluskey method

<u>Specifications</u>:

GUI: You can refer to this source: http://www.32x8.com/index.html

Design: The design of this normalizer is closely similar to the one in the reference above

This application will receive a Boolean expression from the user, perform logic minimization on the expression and output the canonical normal form of the expression (you don't need to do the logic circuit implementation).

+ On the main menu: The user can choose one of the two cases from the navigation bar: 3 variables and 4 variables expression.

+ Input interface: The user can input a logic expression through a truth table, like in the reference. Don't cares values are not allowed. They can also pick a canonical normal form for the simplified expression – either SOP (Sum of products) or POS (Product of Sums). After the user has finished picking values for the truth table, they can press submit to see the output.

+ Output interface: Like in the reference, the application will show to the user: the intermediate columns, the PI table, and the final simplified expression.

<u>Knowledge summary</u>:

The method for minimization of Boolean expression used is Quine-McCluskey.

- If the user chooses SOP as the final canonical form, we perform Quine-McCluskey directly on the expression.
- If the user chooses POS as the final canonical form, we perform Quine-McCluskey on the negation of the expression (replace all the "1" entries on the truth table with "0", and vice versa). Then, in the resulting simplified expression, we replace the AND

(multiplication) operation with OR (addition) operation, and vice versa. This gives us the canonical POS form of the expression.

The steps in the Quine-McCluskey method are as follows:

Step 1 – In column 0, arrange the given min terms in an ascending order and make the groups based on the number of ones present in their binary representations.

Step 2 – In the current column, compare the terms present in adjacent groups. If there is a change in only one-bit position, then combine those two min terms and put a tick mark next to them. A new term is created by placing the symbol ‘x’ in the differed bit position and keep the remaining bits as it is. After trying all possible pairings, we obtain a new set of terms that are smaller by one literal. Put these new terms into a new column, column 1.

Step 3 − Repeat step2 and form new columns 2, 3, … until no more pairing can be done. All the unticked terms from all the columns are PIs (prime implicants).

Step 4 − Formulate the PI table. It consists of set of rows and columns. Each PI is placed in a row and each min term is placed in a column. Place ‘1’ in the cells corresponding to the min terms that are covered in each PI.

Step 5 − Find the essential PI by observing each column. If the min term is covered only by one PI, then it is essential PI. Those essential prime implicants will be part of the simplified Boolean expression. For the other PI, we pick the ones for the simplified Boolean expression by finding the minimum set of PI that can cover all the min terms that are not covered yet by the essential PI. Since we only deal with at most 4 variables, for simplicity, use brute-force search to pick this minimum set of PI. Finally, the simplified expression is constructed by taking the sum of all chosen PIs (each corresponding to a product of variables)