Cairo University
Faculty of Engineering
CMP 102 & CMP N102

Spring 2019

Data Structures and Algorithms

Restaurant Delivery

Project Requirements

Objectives

By the end of this project, the student should be able to:
- Write a **complete object-oriented C++ program** with **Templates** that performs a non-trivial task.
- Use data structures to solve a real-life problem.
- Understand unstructured, natural language problem description and derive an appropriate design.
- Intuitively modularize a design into independent components and divide these components among team members.

Introduction

Finally, the data structures TAs have decided to give up their career and start a new less tiring business, a Restaurant. Yes, it is time to farewell bits and bytes and start working with stacks of wheat packs, queues of customers' orders and loops of Swiss roll. The TAs, however, believe that they can use computer simulation to assess and enhance their delivery service. They currently have '***M***' **delivery motorcycles** and they want to determine the ***service criteria*** -- that is, the criteria based on which a delivery order should be serviced (i.e. delivered) earlier or later than others, to maximize average customer satisfaction. Because the TAs are currently busy running their new business, they ask you to help them, using your programming skills and knowledge of data structures, to write a simulation program that **simulates** the **Restaurant delivery system** given certain service criteria and **calculates** some statistics that measure average customer satisfaction.

Project Phases

Project Phase	*%*
Phase 1	**35%**
Phase 2	**65%**

NOTE: Number of students per team = **3 to 4 students.**

The project code must be totally yours. The penalty of cheating any part of the project from any other source is not ONLY taking ZERO in the project grade but also taking MINUS FIVE (-5) from other class work grades, so it is better to deliver an incomplete project other than cheating it. It is totally your responsibility to keep your code private.

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.

Problem Description

Your system (your program) will receive a **list of orders** as input. This list represents the *scenario* to be simulated. For each order, the system will receive the following information:

- Arrival Time Stamp: When the order was made.
- Order Type: There are 3 types of orders: VIP orders, Frozen orders and Normal orders.
- **VIP orders** must be serviced first before frozen and normal orders.
- **Frozen orders** are the orders that needs to be delivered using motorcycles with refrigerators (*Frozen Motorcycles*) to keep its temperature low.
- **Normal orders** are the orders that neither VIP nor Frozen.

- Order Region: the restaurant has 4 branches. Each branch is in a different region. The *Order Region* indicates the region of the order and hence the branch that should deliver the order to the customer.
- Order Distance: the distance between the restaurant and the order location (in meters).
- Order Money: the total order money the customer should pay.

At startup, the system loads (from a file) information about the available **motorcycles**. For each motorcycle, the system will load the following information:

- Motorcycle Type: There are 3 types: Fast motorcycles, Frozen motorcycles and Normal motorcycles.
- **Fast motorcycles** are motorcycles with higher speed level.
- **Frozen motorcycles** are the motorcycles that have small refrigerator to save food in.
- **Normal motorcycles** are the motorcycles that neither Fast nor Frozen.

- Motorcycle Region: the branch or the region of this motorcycle (from which the motorcycle starts its delivery and to which it returns after delivery).

- Note: The **Motorcycle speed** (the number of meters it can move in one timestep) is the same for all motorcycles of the same type.

Orders Service Criteria

To determine the next order to serve (if a motorcycle is available), TAs proposed the following **Service Criteria** to be applied for all the arrived un-serviced orders **at each time step:**

- First, Serve **VIP orders** by ANY available type of motorcycles, but there is priority of motorcycles' types over the others to deliver VIP orders: First use Fast Motorcycles THEN Normal Motorcycles THEN Frozen Motorcycles. This means that we don't use Normal Motorcycles unless all Fast Motorcycles are busy, and don't use Frozen Motorcycles unless all other types are busy.

- Second, Serve **Frozen orders** using available *Frozen motorcycles ONLY (if found).*

- Third, Serve **Normal orders** using any type of motorcycles *EXCEPT Frozen motorcycles*, but First use the available Normal Motorcycles THEN Fast Motorcycles (if all Normal are busy).

- If an order cannot be serviced at the current time step, it should wait for the next time step to be checked if it can be serviced at it and if not, it will wait and so on.

- **Note 1:** Any motorcycle can deliver orders in its region only.
- **Note 2:** Each motorcycle can deliver only one order at a time then it comes back to the restaurant to assign another order to it.
- **Note 3:** If the orders of a specific type cannot be serviced in the current time step, try to service the other types (e.g. if Frozen orders cannot be serviced in the current time step, this does NOT mean not to service the Normal orders)

That is how we prioritize the service between different order types, but how will we prioritize the service between the orders of **the same type**?

- **For Frozen and Normal orders,** orders that arrive first should be serviced first.
- **For VIP orders,** there is a priority equation for deciding which of the available VIP orders to serve first. VIP orders with higher priority must be serviced first.
  You should develop a reasonable **weighted** priority equation depending on all the following factors: *Order Arrival Time*, *Order Money*, *and Order Distance*.

There are some additional services the restaurant offers to its customer:

1. **For Normal orders ONLY**, their customers can **promote** their orders to become VIP orders by paying more money and can **cancel** the order.
   Note: Order cancelation or promotion request can be accepted only if the request is BEFORE assigning a motorcycle to deliver the order (only during the order waiting).

2. **For Normal orders ONLY**, if an order waits more than **N** timesteps from its arrival time to be assigned to a motorcycle (*its waiting time* >= *N*), it will be **automatically promoted** to be a VIP order without paying any extra money.(N is read from input file)

   Note: If an order is promoted (automatically or by request), the VIP priority equation is calculated for that order to be inserted in its correct position among other VIP orders.

Simulation Approach & Assumptions

You will use incremental time steps. You will divide the time into discrete time steps of 1 unit time each and simulate the changes in the system at each time step.

Some Definitions

- **Arrival Time (AT):**
  It is the time step at which the order is issued by the customer.

- **Inactive Order:**
  It is the order that its arrival time is less than the current timestep (NOT arrived yet).

- **Active Order (waiting orders):**
  It is the order that its arrival time is greater than or equal the current timestep (an arrived order) but not served yet. **At each time step**, you should choose the order**s** to deliver from the active orders (depending on the service criteria described above).

- **In-service Order:**

  It is the order that we start serving (by a motorcycle) but not delivered to its customer yet.

- **Delivered Order:**
  It is the order that is serviced and delivered to the customer.

- **Waiting Time (WT):**
  It is the time spent from the arrival of the order to assigning a motorcycle to deliver it.

- **Service Time (ST):**
  It is the time that a motorcycle needs to deliver an order from the restaurant to the customer(the time spent in the distance between the restaurant and the customer).

- **Finish Time (FT):**
  It is the time step at which the order is successfully delivered to the customer.
  FT = AT + WT + ST

- **Note:** When a motorcycle is assigned for an order, the motorcycle will not be available at time step **FT** but in **FT + ST** because it takes the same **ST** in the way back to the restaurant.

Assumptions

- If the motorcycle comes back at time step T, it can be used to deliver a new order starting from that time step.
- More than one order can arrive at the same time step. Also, more than one order can be chosen to be delivered at the same time step as long as there are motorcycles to deliver them.

File Formats

Your program should receive all information to be simulated from an input file and produces an output file that contains some information and statistics about the simulation. This section describes the format of both files and gives a sample for each.

The Input File

- First line contains three integers: **SN SF SV**
  SN is the speed of all normal motorcycles while **SF** is the speed for frozen ones, and **SV** for VIP motorcycles.
- The 2nd line contains three integers: **N F V**
  Each integer represents the number of motorcycles for different order types. **N** for normal, **F** for frozen, and **V** for VIP orders. The first line is for region A.
  This line is followed by 3 similar lines for regions B, C, and D.
- Then a line with only one integer **AutoS** that represent the number of timesteps after which an order is automatically promoted to VIP.
- The next line contains a number ***M*** that represents the number of **events** following this line
- Then the input file contains ***M*** lines (one line for **each event**). An event can be:
  1. Arrival of a new order. Denoted by letter **R**, or
  2. Cancellation of an existing order. Denoted by letter **X**, or
  3. Promotion of an order to be a VIP order. Denoted by letter **P**.

  NOTE: The input lines of all events are **sorted by the event time** (**TS)** in **ascending** order.

  - **Arrival event line** have the following info
    R TS TYP ID DST MON REG
    where **R** means an arrival event, **TYP** is the order type (*N: normal, F: frozen, V: VIP*). **TS** is the arrival time step. The **ID** is a unique sequence number that identifies each order**. DST** is the distance (in meters) between the order delivery location and the restaurant, **MON** is the total order money and **REG** is the order region.
  - **Cancellation event line** have the following info
    X TS ID
    where **X** means an order cancellation event occurring at time **TS**, and **ID** is the id of the order to be canceled. This ID must be of a Normal order.
  - **Promotion event line** have the following info
    P TS ID ExMon
    where **P** means an order promotion event occurring at time **TS**, and **ID** is the id of the order to be promoted to be VIP. This ID must be of a Normal order.
    ExMon if the extra money the customer paid for promotion.

Sample Input File

```
2  3     6      ➔ motorcycles speeds line
5  3     1      ➔ no. of motorcycles in Region A
6  3     2      ➔ no. of motorcycles in Region B
4  2     1      ➔ no. of motorcycles in Region C
9  4     2      ➔ no. of motorcycles in Region D
25                ➔ Auto promotion limit
8                 ➔ no. of events in this file
R  7     N  1    15   110  A     ➔ Arrival event
R  9     N  2    7    56   B
R  9     V  3    21   300  B
R  12    F  4    53   42   C
X  15    1                      ➔ Cancellation event
R  19    N  5    17   95   D
P  19    2  62                  ➔ promotion event
R  25    F  6    33   127  D
```

The Output File

The output file you are required to produce should contain **M** output line of the format

FT ID AT WT ST

which means that the order identified by sequence number **ID** has arrived at time **AT.** It then waited for a period **WT** to be served**.** It has then taken **ST** ticks to be delivered at timestep **FT**.

(Read the "Definitions Section" mentioned above)

The output lines **must be sorted** by **FT** in ascending order. If more than one order is delivered at the same time step, **they should be ordered by ST**.

Then the following statistics should be shown at the end of the file

1- First for each region
   a. Total number of orders and number of each order type
   b. Total number of motorcycles and number of each type
   c. Average waiting, and average service time
2- Print the same statistics again but for the whole restaurant

Sample Output File

The following numbers are just for clarification and are not produced by actual calculations.

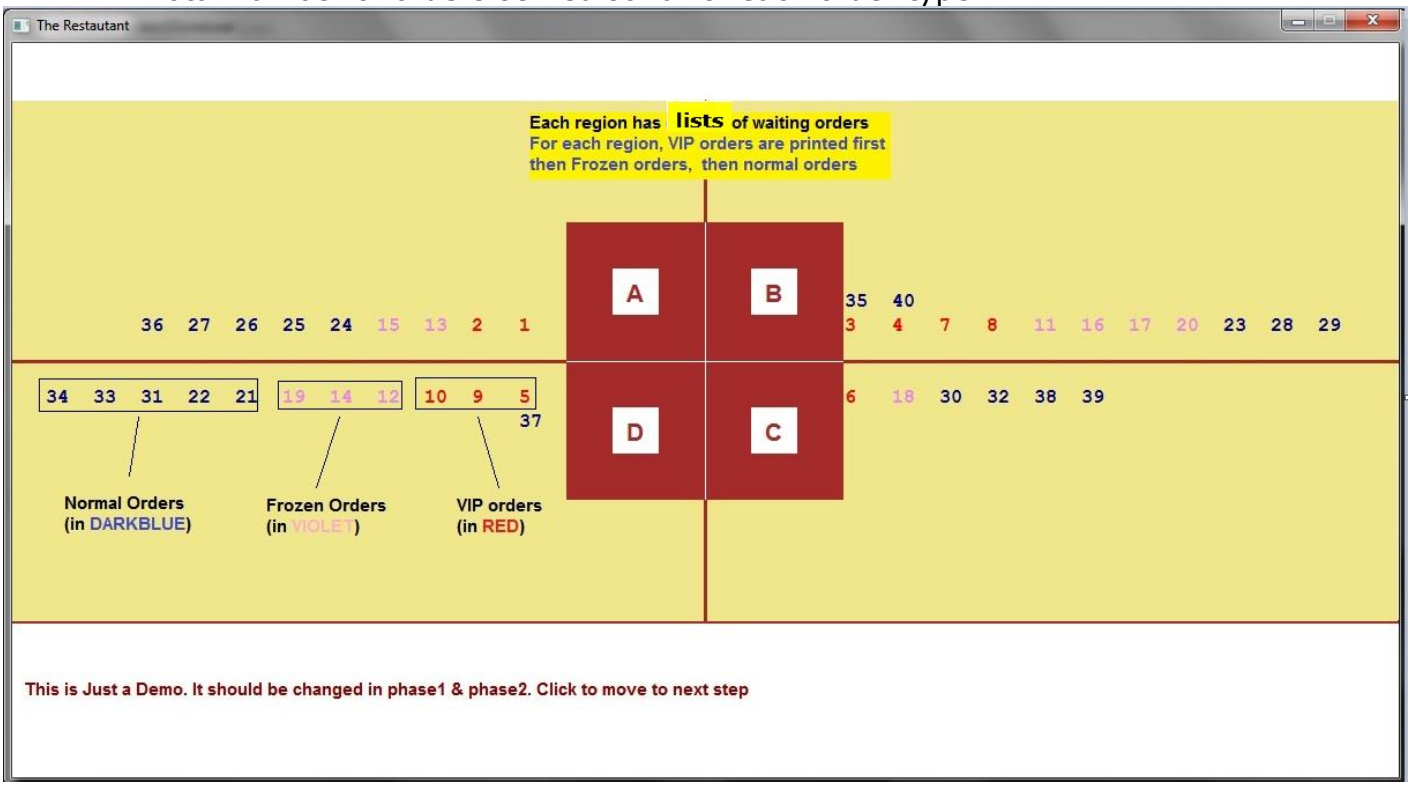
```
FT    ID    AT    WT    ST
18    1     7      5    6
44    10    24    2     18
49    4     12    20    17
.....................................................
.....................................................
Region A:
        Orders: 124 [Norm:100, Froz:15, VIP:9]
        MotorC: 9    [Norm:5, Froz:3, VIP:1]
        Avg Wait = 12.3,  Avg Serv = 25.65
Repeat for each region
Then Repeat for whole restaurant
```

Program Interface

The program can run in one of three modes: **interactive, step-by-step** or **silent mode**. When the program runs, it should ask the user to select the program mode.

Interactive mode allow user to monitor the orders waiting in each region. VIP orders are printed in red while frozen ones are printed in violet and normal are printed in dark blue.
At each time step, program should provide output **similar** to that in the following figure. In this mode, program pauses for a user mouse click to display the output of the next time step.

At the bottom of the screen, the following information should be printed:
- Simulation Timestep Number
- **For each region,** print: ***[Note that the following information is for each region.]***
  - Number of active(waiting) orders of each order type
  - Number of available motorcycles of each type
  - Type & ID for ALL motorcycle**s** and order**s** that were assigned in the last timestep. e.g. **N6(V3)** ➔ normal motorcycle#6 assigned VIP order#3
  - Total number of orders served so far of each order type



Program Interface

Step-by-step mode is identical to interactive mode except that each time step, the program waits for one second (not for mouse click) then resumes automatically.

In **silent mode**, the program produces only an output file (See the "File Formats" section). It does not draw anything graphically.
No matter what mode of operation your program is running in, **the output file** should be produced.

You are provided a code library (set of functions) for drawing the above interface. (See - Appendix A)

Project Phases

You are given a partially implemented code that you should add your phase 1 and phase 2 codes to. It is implemented using classes. You are required to write **object-oriented** code with **Templates** for data structure classes. The graphical user interface GUI for the project is almost all implemented and given to you.

Before explaining the requirement of each phase, All the following are NOT allowed to be used in your project:

- You are not allowed to use **C++ STL** or any external resource that implements the data structures you use. ***This is a data structures course where you should build data structures by yourself from scratch.***
- You need to get instructor approval before making any **custom (new)** data structure. **Note:** No approval is needed to use the known data structures.
- ***Do NOT allocate the same Order more than once***. Allocate it once and make whatever data structures you chose points to it (pointers). Then, when another list needs an access to the same order, DON'T create a new copy of the same order. Just **share** it by making the new list point to it or **move** it from current list to the new one.
  SHARE, MOVE, DON'T COPY…
- You are not allowed to use **global variables** in your implemented part of the project.
- You need to get instructor approval before using **friendships.**

Phase 1

In this phase you should finish implementing any data structures needed for phase 2 without implementing logic related to servicing the order**.**

The required parts to be finalized and delivered at this phase are:

1- **Full data members** of Order, Motorcycle, and Restaurant Classes.

2- **Full "Template" implementation for ALL data structures DS** that you will use to represent **the lists of orders and motorcycles**. All data structure part must be finished in phase 1.

Important: Keep in mind that you are **NOT** selecting the DS that would **work in phase1 only**.
You must choose the DS that would work efficiently for both phase1 & phase2.

When choosing the DS think about the following:

a. How will you store **active orders**? Do you need a separate list for each type?
b. What about the **motorcycles data structure**?
c. Will you use **one list for all regions** or **a separate list for each region?**
d. Do you need to store **delivered orders**? When should you delete them?
e. **Which list type** is much suitable to represent each list? You must take into account the **complexity of the main operations** needed for each list (e.g. insert, delete, retrieve, shift, sort …etc.). For example, If the most frequent operation in a list is deleting from the middle, use a data structure that has low complexity for this deleting.

You need to justify your choice for each DS and why you separated or joined lists. Selecting the appropriate DS for each list is the core target of phase 1 and the project as a whole. Most of the discussion time will be on that.

Note: you need to read *"File Format"* section to see how the input data and output data are sorted in each file because this will affect the selection of the data structures.

3- **File loading function.** The function that reads input file to:
   a. Load Orders and Motorcycle data
   b. Create new orders and motorcycles to populate (fill) events list(s) and motorcycles list(s).

4- **Simple Simulator function for Phase 1**. The main purpose of this function is to test your data structures and how to move orders and motorcycles between lists. This function should:
   a. Perform any needed initializations
   b. Call file loading function
   c. At ***each time step*** do the following:
      i. Move orders to active list(s) according to arrival time.
      ii. Update the Interface to display the active orders of each list in each region
      iii. **For each region**:
         1. Print in the status bar:
            a. Number of active(waiting) orders of each order type
            b. Number of available motorcycles of each type
         2. Pick one order from each order type list and delete it.
            Note 1: the one order you choose to delete from each type must be the first order that should be assigned to an available motorcycle in phase 2 (for example, in VIP list, delete the order with the highest priority – not any order).
            Note 2: deleting here does NOT mean servicing or assigning to motorcycles. *No assigning to motorcycles is required in Phase 1*.
         3. If cancellation events exist for this time step, cancel them ONLY IF

found in the Normal Orders list.
Note: ignore executing promotion events in Phase 1.

iv. Update the interface again

d. The simulation function stops when there are no more events nor active orders in the system

Notes about phase 1:

- No output files should be produced at this phase.
- In this phase, you can go to the next timestep by mouse click
- No delivery (service) or assigning motorcycles will be done in this phase. However, all the lists of the project should be implemented in that phase.
- Make sure you read **Project Evaluation** and **Individuals Evaluation** section mentioned below.

Phase 1 Deliverables:

Each team is required to deliver a **CD** that contains:

- A text file named ***ID.txt*** containing team members' names, IDs, and emails.
- **Phase 1 full code** [Do not include executable files].
- **Three Sample input files** (test cases).
- **Work Load Document:** how the load is divided between members in this phase. **Print** it and bring it with you in the discussion day.
- **Phase 1 document** with 1 or more pages describing:
  1. Each order and motorcycle list you chose
  2. The DS you chose for each list
  3. Your justification of all your choices with the complexity of the most frequent or major operation for each list.
- Write your team number and team members name on the back of the CD cover.

Phase 2

In this phase, you should extend code of phase 1 to build the full application and produce the final output file. Your application should support the different operation modes described in "Program Interface" section.

Phase 2 Deliverables:

Each team is required to deliver a **CD** that contains:

- A text file named ***ID.txt*** containing team members' names, IDs, and emails.
- **Final Project Code** [Do not include executable files].
- **Six Comprehensive sample input files (test cases) and their output files.**
- **Work Load Document**. Don't forget to **Print** it and bring it with you in the discussion day.
- **A project document** with 2 or more pages describing your solution modules and any clever or innovative alternatives you followed in implementing the solution.

Project Evaluation

These are the main points that will be graded in the project:

- **Successful Compilation:** Your program must compile successfully with zero errors. Delivering the project with any compilation errors will make you lose a large percentage of your grade.

- **Object-Oriented Concepts:**
  - **Modularity:** A **modular** code does not mix several program features within the same unit (module). For example, the code that does the core of the simulation process should be separate from the code that reads the input file which, in turn is separate from the code that implements the data structure. This can be achieved by:
    - adding classes for each different entity in the system and each DS has its class.
    - dividing the code in each class to several functions. Each function should be responsible for a single job. Avoid writing very long functions that does everything.
  - **Maintainability:** A maintainable code is the one whose modules are easily modified or extended without a severe effect on other modules.
  - **Separate each class in .h and .cpp** *(if not a template class).*
  - **Class Responsibilities:** No class is performing the job of another class.

- **Data Structure & Algorithm**: After all, this is what the course is about. You should be able to provide a concise description and a justification for: (1) the data structure(s) and algorithm(s) you used to solve the problem, (2) the **complexity** of the chosen algorithm, and (3) the logic of the program flow.

- **Interface modes**: Your program should support the three modes described in the document.

- **Test Cases**: You should prepare comprehensive different test cases (at least 6). Your program should be able to simulate different scenarios not just trivial ones.

- **Coding style**: How elegant and **consistent** is your coding style (indentation, naming convention ...etc.)? How useful and sufficient are your comments? This will be graded.

Individuals Evaluation

Each member must be responsible for writing some project modules (e.g. some classes or some functions) and must answer some questions showing that he/she understands both the program logic and the implementation details. The work load between team members must be almost equal.

- ✓ The grade of **each student** will be divided as follows:
  - **[70%]** of the student grade is on his individual work (the project part he was responsible for).
  - **[25%]** of the student grade is on integrating his work with the work of ALL students who Finished or nearly finished their project part.
  - **[5%]** of the student grade is on cooperation with other team members and helping them if they have any problems implementing their parts (helping does NOT mean implementing their parts).

- ✓ If one or more members didn't make their work, the other members will NOT be affected **as long as** the students who finished their part integrated all their parts together AND their part can be tested to see the output.
- ✓ If a student couldn't finish his part, try to still integrate it with the other parts to at least take the integration grade.
- ✓ You should **inform the TAs** before the deadline **with a sufficient time (some weeks before)** if any problems happened between team members to be able to take grading action appropriately.

Bonus Criteria (maximum 10%)

- **[3%] Motorcycle speed:** Motorcycles of the same type may have different speeds. The motorcycles of a specific type must be sorted by its speed. Higher speed motorcycles of a type have the higher priority to be used in delivery than lower speed motorcycles of the same type.
- **[3%] Motorcycles Damage and Traffic Problems:** Motorcycles can get damaged and become out of order for a certain time. Some orders can be hard to deliver due to traffic problems. A motor cycle which delivered such an order should wait when it comes back and not assigned another order for a certain time and if the system needs this motorcycle to be used before its rest period is over, this will cause some damage to it.
- **[4%] More order types:** Think about **two** more order types other than those given in the document. The load of the logic of the two orders must be acceptable (**Take a TA Approval**).

Appendix A – Guidelines on the Provided Framework

The main objects of the game are Restaurant, Order and Motorcycle. There is a class for each of them.

Event classes:

There are three types of events; Arrival, Cancel, and Promote events. You are given a base class called "***Event***" that stores event time and related order ID. This is an abstract class with a pure virtual function "**Execute**". The logic of Execute function depends on the Event type.

For each type of the three events, there should be a class derived from ***Event*** class. Each derived class should store data related to its operation. Also each of them should override function **Execute** as follows:

1- ArrivalEvent::Execute ➔ should create a new order and adds it to the appropriate list
2- CancelEvent::Execute ➔ should cancel the requested order if found
3- PrompteEvent::Execute ➔ should move a normal order to the VIP list and update order's data.

Class Restaurant has a queue of ***Event*** pointers to store all events loaded from the file at system startup. At each timestep, the code loops on the events queue to dequeue and execute all events that should take place at current timestep.

GUI class

It is the class responsible for ALL drawings and ALL inputs. It contains the input and output functions that you should use to show status of the orders at each timestep.

Main members of class GUI:

1. ***OrdListForDrawing****:* an array of **pointers to Order**
   - This array should point to ALL orders to be drawn at current timestep.
   - Orders to be drawn are only those that has arrived and still waiting to be served.
   - It should be filled so that for each region VIP orders added first, then frozen orders then normal orders.
   - At the end of each timestep, it should be reset to be reloaded again with orders of the next tiemstep.
2. Function ***AddOrderForDrawing(Order *pO)***
   - Used to add an order to the array of pointers (*OrdListForDrawing*)
   - It does not create a new order. Rather, it makes the first available pointer points to the passed order
3. Function ***ResetDrawingList( )***
   - Resets the drawing list (at the end of each time step)
4. Function ***DrawOrders( )***
   - Responsible for drawing ALL orders in array *OrdListForDrawing*
   - For each order in the array it calls ***DrawSingleOrder*** function to draw it

5. Function ***DrawSingleOrder(Order* pO, int regionCount)***
   - Draws the passed order (pO) by printing order ID on the screen.
   - Each order type is printed in a different color
   - regionCount parameter is used to place the order in its correct relative position with respect to other orders in the same region.
6. Function ***UpdateInterface( )***
   - It first clears the interface then draws everything again
   - Called every timestep to display current system status.
7. Function ***PrintMsg(string msg)***
   - Prints ***msg*** on the status bar
   - Use this function to print information about each region at every timestep
8. Function ***string GetString( )***
   - Reads a string from the GUI window
   - Use this function to read input file name

[Important Note]:

The drawing list (OrdListForDrawing) is just used to draw orders.

It has nothing to do with the data structure you will be using to store orders in the system. You can pick whatever data structure that is suitable for order manipulation. For each order to be drawn on the screen, just prepare a pointer to it then call **AddOrderForDrawing** to add it to that list then call **UpdateInterface** to show them all

Finally, a demo code (***Demo_Main.cpp***) is given just to test the above functions and show you how they can be used. It has nothing to do with phase 1 or phase 2. You should write your main function of each phase by yourself.