Cairo University
Faculty of Engineering
Computer Engineering Department
CMP N102

Spring 2021

Data Structures and Algorithms

Mars Exploration

Project Requirements

Objectives

By the end of this project, the student should be able to:
- 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.
- Build and use data structures to implement the proposed design.
- Write a **complete object-oriented C++ program** that performs a non-trivial task.

Introduction

In the hope of getting to know more about Mars and the possibility of life on its surface, a huge number of exploratory missions to different regions of the planet need to be conducted. Imagine (at some future time) that we have several rovers of different types and capabilities on the surface of Mars ready to carry out missions to its different regions. There is also a hypothetical base station which acts as the central point from which the different rovers begin their exploratory missions and to which they return after mission completion. The base station basically assigns missions to the different rovers.

Using your programming skills and your knowledge of the different data structures, you are going to develop a program that **simulates** the **mission assignment process** and **calculates** some related **statistics** in order to help improve the overall process.

Project Phases

Project Phase	***%***	***Deadline***
Phase 1	**20%**	**Week 7**
Phase 2	**80%**	**Week 14**

Late Submission is not allowed

NOTE: Number of students per team = 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 rather than cheating it. It is totally your responsibility to keep your code away from others.

Missions and Rovers

Missions:

The following pieces of information are available for each mission:

- **Formulation Day:** the day when the mission formulation is finalized and the mission becomes ready to be assigned to a rover.
- **Mission Type:** There are 3 types of missions: Emergency, Mountainous and Polar missions.
    - **Emergency missions** must be assigned first before mountainous and polar missions.
    - **Mountainous missions** are missions to mountainous regions of Mars and must be conducted by rovers equipped to navigate in such regions.
    - **Polar missions** are missions to the polar ice caps of Mars and must be conducted by rovers equipped to navigate in such regions.
- **Target Location:** The distance (in kilometers) from the base station to the target location of the mission.
- **Mission Duration:** The number of days needed to fulfill the mission requirements at the target location (assumed constant regardless of rover type).
- **Significance:** A number representing the importance of the mission and how beneficial it is (the higher the number, the more significant it is).

Rovers:

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

- **Rover Type:** There are 3 types of rovers: Emergency rovers, Mountainous rovers, and Polar rovers.
    - **Emergency rovers** are rovers which are over-equipped and ready for emergency missions in almost any region.
    - **Mountainous rovers** are rovers which can navigate in mountainous regions.
    - **Polar rovers** are rovers which can navigate in polar regions.
- **Checkup Duration:** The duration (in days) of checkups that a rover needs to perform after completing **N** missions.
- **Speed:** in kilometers/hour. (Note that a day on Mars is 25 hours.)

Missions Assignment Criteria

To determine the next mission to assign (if a rover is available), the following **assignment criteria** should be applied for all the formulated un-assigned missions **on each day:**

1. First, assign **emergency missions** to ANY available rover of any type. However, there is a priority based on the rover type: first choose from emergency rovers THEN mountainous rovers THEN polar rovers. This means that we do not use mountainous rovers unless all emergency rovers are busy, and we do not use polar rovers unless rovers of all other types are busy.

2. Second, assign **polar missions** using the available polar rovers ***ONLY***. If all polar rovers are busy, wait until one is available.

3. Third, assign **mountainous missions** using any type of rovers *EXCEPT polar rovers*. First use the available mountainous rovers THEN emergency rovers (if all mountainous rovers are busy).

4. If a mission cannot be assigned on the current day, it should wait for the next day. On the next day, it should be checked whether the mission can be assigned now or not. If not, it should wait again and so on.

NOTES: If missions of a specific type cannot be assigned on the current day, try to assign the other types (e.g. if polar missions cannot be assigned on the current day, this does NOT mean not to assign the mountainous missions).

This is how we prioritize the assignment of missions of different types, but how will we prioritize the assignment of missions of **the same type**?

- **For polar and mountainous missions,** assign them based on a first-come first-served basis. Missions that are formulated first are assigned first.
- **For emergency missions,** you should design a priority equation for deciding which of the available emergency missions should be assigned first. Emergency missions with a higher priority are the ones to be assigned first.

☞ You should develop a reasonable **weighted** priority equation depending on at least the following factors: *the mission formulation day, how far is the mission's target location, the mission's duration, and the mission's significance.*

There are some additional services that the base station has to accommodate:

- **For mountainous missions ONLY**, a request can be issued to **promote** the mission to become an emergency one. A request of mission **cancellation** could also be issued.
- **For mountainous missions ONLY**, if a mission waits more than **AutoP** days from its formulation day to be assigned to a rover, it should be **automatically promoted** to be an emergency mission. (**AutoP** is read from the input file).

Simulation Approach & Assumptions

You will use incremental **day** steps. Simulate the changes in the system every **1 day.**

Some Definitions

- **Formulation Day (FD):**
  The day on which the mission is formulated and is ready to be assigned.

- **Waiting Mission:**
  The mission that has been formulated (i.e. mission's FD < current day but the mission is not assigned yet). On each day, you should choose the mission(s) to assign from the waiting missions.

- **In-Execution Mission:**
  The mission that has been assigned to a rover but is not completed yet.

- **Completed Mission:**
  The mission that has been completed.

- **Waiting Days (WD):**
  The number of days from the formulation of a mission until it is assigned to a rover.

- **Execution Days (ED):**
  The days that a rover needs to complete a mission (the days it takes to reach the target location, fulfill mission requirements, and then get back to the base station).

- **Completion Day (CD):**
  The day at which the mission is successfully completed by the rover.
  (CD = FD + WD + ED)

Assumptions

- If the rover is available on day D, it can be assigned to a new mission starting from that day.
- More than one mission can be formulated on the same day. Also, more than one mission can be assigned to different rovers on the same day as long as there are available rovers.
- A rover can only be executing one mission at a time.
- A rover cannot be assigned a mission during its checkup time
- Checkup duration and rover speed are the same for all rovers of the same type.

Input/Output 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 missions. This section describes the format of both files and gives a sample for each.

The Input File

- First line contains three integers. Each integer represents the total number of rovers of each type.
  - **M:** for mountainous rovers
  - **P:** for polar rovers
  - **E:** for emergency rovers
- The 2nd line contains three integers:
  - **SM:** is the speed of all mountainous rovers (kilometers/hour)
  - **SP:** is the speed of all polar rovers (kilometers/hour)
  - **SE:** is the speed of all emergency rovers (kilometers/hour)
- The 3rd line contains four integers:
  - **N:** is the number of missions the rover completes before performing a checkup
  - **CM:** is the checkup duration in days for mountainous rovers
  - **CP:** is the checkup duration in days for polar rovers
  - **CE:** is the checkup duration in days for emergency rovers
- Then a line with only one integer **AutoP** which represents the number of days after which a mountainous mission is automatically promoted to an emergency mission.
- The next line contains a number ***E*** which represents the number of **events** following this line.
- Then the input file contains ***E*** lines (one line for **each event**). An event can be:
  - Formulation of a new mission. Denoted by letter **F**, or
  - Cancellation of an existing mission. Denoted by letter **X**, or
  - Promotion of a mission to be an emergency mission. Denoted by letter **P**.

  NOTE: The input lines of all events are **sorted by the event day** in **ascending** order.

Events

- **Formulation event line** has the following information:
  - **F** (letter F at the beginning of the sentence) means a mission formulation event.
  - **TYP** is the mission type (*M: mountainous, P: polar, E: emergency*).
  - **ED** is the event day.
  - **ID** is a unique sequence number that identifies each mission.
  - **TLOC** is the mission's target location (in kilometers from the base station).
  - **MDUR** is the number of days needed to fulfill the mission requirements at target location.
  - **SIG** is the mission's significance.
- **Cancellation event line** has the following information:
  - **X** (Letter X) means a mission cancellation event.
  - **ED** is the event day.

- **ID** is the ID of the mission to be canceled. This ID must be of a mountainous mission.

- **Promotion event line** has the following information:
  - **P** (Letter P) means a mission promotion event.
  - **ED** is the event day.
  - **ID** is the ID of the mission to be promoted to emergency. This ID must be of a mountainous mission.

Sample Input File

```
3   3   2                     → no. of rovers of each type
1   2   2                     → rover speeds of each type (km/h)
3   9   8   7                 → no. of missions before checkup and the checkup durations
25                            → auto promotion limit
8                                       → no. of events in this file
F   M   2   1   100   4   5             → formulation event example
F   M   5   2   250   4   4
F   E   5   3   500   6   3
F   P   6   4   900   7   4
X   10  1                               → cancellation event example
F   M   18  5   560   5   9
P   19  2                               → promotion event example
F   P   25  6   190   3   1
```

The Output File

The output file you are required to produce should contain **M** output lines of the following format:
CD ID FD WD ED
which means that the mission identified by sequence number **ID** has been formulated on day **FD.** It then waited for a period **WD** to be assigned**.** It has then taken **ED** to be completed at the day **CD**. *(Read the "Definitions Section" mentioned above)*
The output lines **must be sorted** by **CD** in ascending order. If more than one mission is completed on the same day, **they should be ordered by ED**.

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

1. Total number of missions and number of missions of each type
2. Total number of rovers and number of rovers of each type
3. Average waiting time and average execution time in days
4. Percentage of automatically-promoted missions (relative to the total number of mountainous missions)

Sample Output File

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

```
CD   ID   FD   WD   ED
18   1    7    5    6
44   10   24   2    18
49   4    12   20   17
………………………………………………
………………………………………………
Missions: 124  [M: 100, P: 15, E: 9]
Rovers: 9      [M: 5, P: 3, E: 1]
Avg Wait = 12.3,  Avg Exec = 25.65
Auto-promoted: 20%
```

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.

1. Interactive Mode: Allows user to monitor the missions and rovers. The program should print an output like that shown below. In this mode, the program prints the current day then pauses for an input from the user ("Enter" key for example) to display the output of the next day.

```
Current Day:78
7 Waiting Missions: [11,13] (6,8) {9,12,14}
-------------------------------------------------------
4 In-Execution Missions/Rovers: [2/1, 10/7] (3/5) {7/9}
-------------------------------------------------------
4 Available Rovers: [4, 6] (10) {8}
-------------------------------------------------------
2 In-Checkup Rovers: [2] {3}
-------------------------------------------------------
3 Completed Missions: {5} [4] (1)
```

Output Screen Explanation

- The numbers shown are the IDs of missions and rovers printed according to their types. The IDs of **[emergency missions or rovers]** are printed within **[ ]**, the IDs of **(polar ones)** are printed within **( )**, and the IDs of {**mountainous ones}** are printed within { }.
- In line 3, a notation like **[2/1,…]** means emergency mission #2 is being executed by rover#1
- Other lines are self-explanatory.
- The above screen is just for explanation and is not generated by actual simulation.

2. Step-By-Step Mode is identical to the interactive mode except that after each day, the program waits for one second (not for user input) then resumes automatically.

3. Silent Mode, the program produces only an output file (See the "File Formats" section). It does not print any simulation steps on the console. It just prints the following screen

```
Silent Mode
Simulation Starts...
Simulation ends, Output file created
```

NOTE: No matter what mode of operation your program is running in, **the output file** should be produced.

Project Phases

You are required to write **object-oriented** code with **templates** for data structure classes.

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 yourself from scratch.***

- You need to get instructor's approval before making any **custom (new)** data structure. **NOTE:** No approval is needed to use the known data structures.

- ***Do NOT allocate the same Mission more than once***. Allocate it once and make whatever data structures you chose point to it (pointers). Then, when another list needs an access to the same mission, DON'T create a new copy of the same mission; 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 code.

- You need to get instructor approval before using **friendships.**

Phase 1

In this phase you should decide the data structures that you will use in your project.

Selecting the appropriate DS for each list is the core target of phase 1 and the project as a whole.

You should deliver a report in the following format:

Cairo University, Faculty of Engineering
Computer Engineering Department
Data Structures and Algorithms
CMPN102

Spring 2021

Data Structures and Algorithms
Project Phase1 Report

Team Name: **Number of members:**
Team Email:

Members' Info:

Member Name	ID	Email

Project Data Structures

List Name	Chosen DS	Justification
e.g. Events List	Write the DS you have chosen e.g. Queue/Stack/ Pri-Q/ Tree .. etc	Write here the **operations** you need to perform on this list and the **complexity** of each operation Then justify why you have chosen this DS to implement that list

➔ Repeat he above for each list in the project

You should cover all lists in your project
See "**Data Structures Selection Guidelines**" before writing this report

Data Structures Selection Guidelines:

1- Do you need a separate list for each mission type? Mission assignment criteria described above should affect your decision.
2- Do you need a separate list for each rover type?
3- Do you need a separate list for each mission status. i.e. a list for waiting and another one for in-execution and a third one for completed or just one list for all?
4- Do you need a separate list for each rover status. For example, would you make a list for available rovers and another one for those under checkup,…etc. or just one list for all.
5- Do you need to store **completed** missions? When should you get rid of them to save memory?
6- **Which list type** is much suitable to represent the lists taking into account the **complexity of the operations** needed for each list (e.g. insert, delete, retrieve, shift, sort, etc.). You may need to make a survey about the complexity of the operations. Then, decide what are the most frequent operations needed according to the project description. Then, for this list, choose the DS with best complexity regarding those frequent operations.
7- Keep in mind that you are **NOT** selecting the DS that would **work in phase1**. **You should choose the DS that would work efficiently for both phases.**
8- **Important**: if you find out that a list can be implemented using two types, you should choose the more restricted type. For example if a list can be implemented using queue general list, you should use queue.

Note: You need to read *"File Format"* section and to see how the input data and output data are sorted in each file because this will affect your DS selection. important !!!!!

Phase 1 Deliverables:

Each team is required to submit phase1 report

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 the following:

- 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.**
- **Workload document**. Don't forget to **print** it and bring it with you on the discussion day.

Project Evaluation

These are the main points for project evaluation:

- **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 structures. This can be achieved by:
    - Adding classes for each different entity in the system and each DS used.
    - Dividing the code in each class to several functions. Each function should be responsible for a single job. Avoid writing very long functions that do everything.
  - **Maintainability:** A maintainable code is the one whose modules are easily modified or extended without a severe effect on the other modules.
  - **Class responsibilities:** No class is performing the job of another class.

- **Data Structures & Algorithms**: 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 different comprehensive 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, ...)? How useful and sufficient are your comments? This will be graded.

Notes:

- ❑ The code of any operation does NOT compensate for the absence of any other operation.
- ❑ There is no bonus on anything other than the points mentioned in the bonus section.

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% of Project Grade)

❑ **[3%] Rover Speed:** Rovers of the same type may have different speeds. The rovers of a specific type must be sorted by their speed. Higher speed rovers of a type have the higher priority to be assigned to missions than lower speed rovers of the same type.

❑ **[3%] Rovers Maintenance:** Rovers can sometimes need maintenance apart from their regular checkups. If this happens, they should be unavailable for some time. If the system needs this rover before its maintenance is over, this will cause the rover's speed to be decreased to its half till the end of the simulation.
Think about reasonable cases when should a rover get into maintenance.

❑ **[4%] Mission failure**: with a very low probability, a mission might fail due to problems in rovers executing it.
In this case: (1) the rover should be moved to checkup, (2) the mission is not completed and needs to be re-formulated and added to the list of waiting missions to be assigned later to another rover of the same type.

Appendix A – Guidelines for Project Framework

The main classes of the project should be MarsStation, Mission, Rover, Event, and UI (User Interface). In addition, you need lists of appropriate types to hold events, missions, rovers,..etc.

Event Classes:

There are three types of events: Formulation, Cancellation, and Promotion events. You should create a base class called "***Event***" that stores the event day and the related mission ID. This should be an abstract class with a pure virtual function "**Execute**". The logic of the Execute function should depend 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 the function **Execute** as follows:

1. FormulationEvent::Execute ➔ should create a new mission and add it to the appropriate list
2. CancelEvent::Execute ➔ should cancel the requested mountainous mission (if found and is waiting)
3. PrompteEvent::Execute ➔ should move a mountainous mission to the emergency list and update the mission's data (if found and is waiting)

Mission and Rover Classes:

Should have data members to store all info of missions and rovers. In addition to appropriate member functions.

MarsStation Class

It should have an appropriate ***list of Event pointers*** to store all events loaded from the file at system startup.

This class is the maestro class that manages the system. It should have member functions to:

1- At program startup, open the input file and load rovers data to fill rovers list(s) and to load the events list.

2- One each day,
   a. Execute the events that should be executed at that day
   b. Check waiting mission to assign them to available rovers
   c. Move missions from waiting to in-execution to completed
   d. Move rovers from available to in-execution to checkup to available again
   e. Collect statistics that are needed to create output file
   f. Calls UI class functions to print day details on the output screen

3- Produce the output file at the end of simulation

UI Class

This should be the class responsible for taking in any inputs from the user and printing any information on the console. It contains the input and output functions that you should use to show status of the missions on each day.

At the end of each day, this class should print the output screen (see Program Interface section) showing what has happened during that day.

The is the only class whose function can have input and output (cin and cout) lines

Input and output lines must NOT appear anywhere else in the program