FAKÉDEX

Machine Project Specifications

BACKGROUND

A Pokédex is a device that serves as a database for all existing creatures (Pokémon) found in each generation of the Pokémon franchise. Players use it as a reference to the total number of unique Pokémon species they can catch in the game. Each Pokédex entry contains detailed information about a Pokémon, such as its name, nickname, brief description, catch date, type, level, and battle stats. All entries in a Pokédex are initially hidden until the player catches at least one Pokémon of that species.

For your project, you are to create an altered Pokédex application called Fakédex, that manages user-generated Pokémon entries. The information for each entry has been reduced, while the catching and storage mechanics have been simplified. The program must be able to save and load the contents of the dex and storage box through text files.

FUNCTIONALITIES

The program must satisfy the functionalities listed below. Each major section (*Fakédex*, *Exploration*, *Box*, *Settings*, and *Exit*) must be accessible at the program's main menu.

I. Fakédex

The Fakédex feature stores the list of Pokémon available in the game. The dex starts off as empty, and has a maximum capacity of 150 entries. Users must manually populate the list through the *Add Dex Entry* feature.

- **Add Dex Entry**
  Users can add a dex entry with the information described in Table 1.1 below.

FIELD	DESCRIPTION
NAME	Maximum of 11 characters. Letters only. Must be unique. Must not be empty.
SHORT NAME	Exactly 5 characters. Uppercase letters only. Must be unique. Must not be empty.
DESCRIPTION	Maximum of 300 characters. Must not be empty.
GENDER	`MALE`, `FEMALE`, or `UNKNOWN` values only. Must not be empty.
CAUGHT	`YES` or `NO` values only. All new entries must have a value of `NO`.

Table 1.1 Entry Fields

From the fields listed above, only the **CAUGHT** field is automatically set by the program to **`NO`**. All other fields are given to the program through user input. The user should only be allowed to enter one field value at a time, in the following order: *name*, *short name*, *description*, *gender*. This means that the program asks for the *Name* first, validates it, then asks for the *Short Name* next, validates it, and so on. Users are not allowed to proceed to the next field until they input a valid value to the current field.

The program must validate and enforce all limitations on user input stated in the **DESCRIPTION** column of Table 1.1. Inputs that do not conform to these limitations are rejected by the program. Whenever an input is rejected, the user is shown an *error message* that states the specific limitation that was violated. Only print the first violation detected by your program at a time (see Table 1.2). The order of limitations checked by your program is up to you.

	USER INPUT	LIMITATIONS VIOLATED	ERROR MESSAGE
1	?Unknown No.7?	• NAME LENGTH • LETTERS ONLY	NAME FIELD CAN HAVE AT MOST 11 CHARACTERS.
2	?Unown No.7	• LETTERS ONLY	NAME FIELD CAN ONLY HAVE LETTERS.
3	Unown Seven	• LETTERS ONLY	NAME FIELD CAN ONLY HAVE LETTERS.
4	UnownSeven		✓

Table 1.2 Sample Error Message for Invalid Name Values

Adding a Pokémon whose *name* is already in the dex (regardless of capitalization) will show the details of the existing entry, and ask the user if they wish to overwrite the *short name*, *description*, and *gender* fields of that Pokémon. Note that selecting **`YES`** will also update the *name* field in the entry to follow the capitalization of the new name. The *caught* field should always follow the original entry's value.

The *short name* should be unique amongst all dex entries. Attempting to add a Pokémon with a unique *name* but a non-unique *short name* should prompt the user to change the *short name* value. Adding a Pokémon with a non-unique *name* and *short name* will simply prompt an overwrite (see Table 1.3).

Attempting to add an entry when the dex capacity is full should ask the user if they wish to remove an entry first. Selecting **`NO`** will bring the user back to the main menu. Selecting **`YES`** will bring up the *Remove Dex Entry* feature. Once a valid entry has been removed, the newly created entry should automatically be added to the dex. The user should also be prompted that the new entry has been successfully added. Once an entry has been successfully added or updated, it must reflect in the *View Dex Entries* feature

	NEW ENTRY	EXISTING ENTRY	RESULT
NAME	Gengar	Gengar	**PROMPT FOR OVERWRITE**
SHORT NAME	GNGAR	GENGR	
NAME	Arctovish	Arctozolt	**INVALID SHORT NAME**
SHORT NAME	ARCTO	ARCTO	
NAME	Dragapult	Dragapult	**PROMPT FOR OVERWRITE**
SHORT NAME	DRAGA	DRAGA	

Table 1.3 Sample Cases for Name and Short Name.

- **View Dex Entries**
  Users should be able to view the complete list of Pokémon through this feature. Each entry should list a Pokémon's corresponding *name*, *short name*, *description*, *gender*, and *caught* values. All uncaught Pokémon's *short name*, *description*, and *gender* values should show as `???` in the dex, until at least one instance of that Pokémon has been caught.

- **Update Dex Entry**
  Users should be allowed to edit the *name*, *short name*, *description*, and *gender* of all dex entries. The edited values must still follow the previously stated limitations[1]. Once an entry has been successfully updated, it must reflect in the *View Dex Entries* feature.

- **Remove Dex Entry**
  Users should be able to select a dex entry (through its *name*) and remove it from the list. When an entry is removed, the list should adjust to occupy the removed entry's space. Dex entries that are already marked as caught cannot be removed from the dex anymore. Once an entry has been successfully removed, it must reflect in the *View Dex Entries* feature.

II. Exploration

Catching Pokémon and updating the Fakédex is done through the exploration feature. The player traverses a row of grass tiles to encounter a random Pokémon listed in the dex. Encountered Pokémon are not limited to uncaught entries, so it is possible to catch more than one Pokémon of the same name.

The player should not be allowed to select this feature from the main menu if their box has reached maximum capacity. Additionally, the player should be kicked out of this feature once their box reaches maximum capacity while in the middle of exploration.

- **Grass Tiles**
  A row has 7 tiles that serve as patches of grass. The player (denoted by a token of any symbol) starts at column 0, and can be commanded to move **`FORWARD`** or **`BACKWARD`** within the grass tiles. The token's position should update accordingly whenever the player is issued

a valid move command. The player should also be prevented from stepping past the specified bounds.

Everytime the player successfully takes a step on a different patch of grass, they have a random chance of encountering a Pokémon (see Encounters). Typing **`EXIT`** in the grass tiles screen returns the user to the main menu[3].

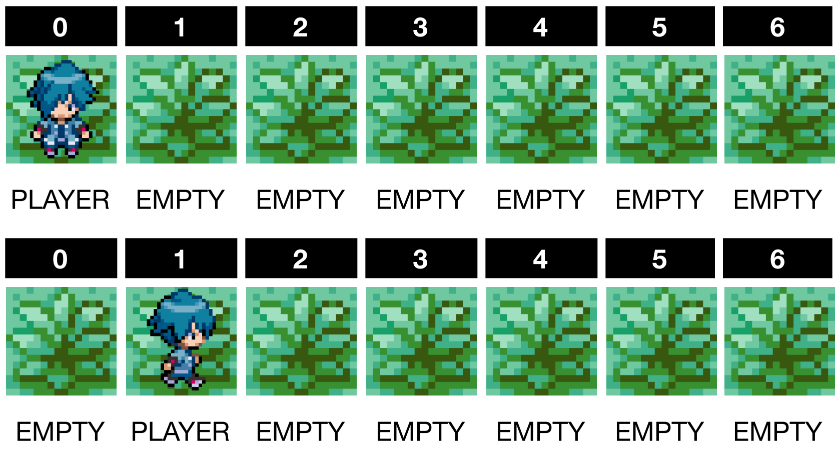


**Figure 2.1 Initial Configuration of Grass Tiles (Top)
Player Moves Forward (Bottom)**

- **Encounters**
  Each grass tile has a 40% chance of triggering a Pokémon encounter. Only entries currently listed in the Fakédex can be encountered, so removed entries should not appear anymore. If the dex is empty, then no encounters can trigger during exploration.

  When an encounter is triggered, a random Pokémon should be displayed on screen. Each Pokémon listed in the dex should have an equal chance of appearing. The *name* and *gender* of the encountered Pokémon should be shown to the user. The user is then prompted if they want to **`CATCH`** or **`RUN`** (see Catching).

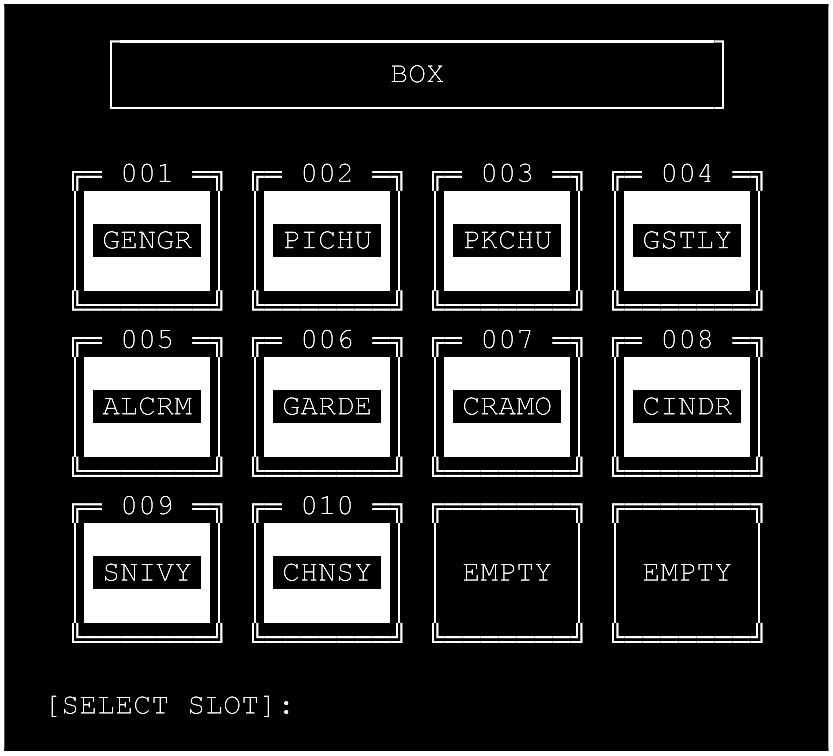
- **Catching**
  If the user selects **`CATCH`** during an encounter, they have an 80% chance of successfully catching the Pokémon. If the catch is successful, then the *short name*, *description*, and *gender* fields of that Pokémon's dex entry are revealed (as opposed to **`???`**) and the value of the *caught* field is changed to **`YES`**. A copy of the dex entry is also immediately shown to the user before the user is returned to the grass tiles[4] screen.

  Note that this only needs to be done on the very first time a Pokémon of a certain name is caught. No need to update the dex, nor show the dex entry after a successful catch, if the Pokémon has been previously caught before. Changes made to any dex entry should carry over to the *View Dex Entries* feature when it is accessed later on.

  If the catch is unsuccessful, then the player is returned to the grass tiles screen. The same thing occurs if they selected the **`RUN`** option. Whenever a player returns to the grass tiles screen after an encounter (regardless of its outcome), they should remain on the same tile they previously were when the encounter was triggered.

III. Box

Pokémon caught from explorations are immediately placed in a box. When this feature is accessed, it should print the *slot number* along with the *name* or *short name* of the Pokémon occupying each slot. When printing the box contents, make it so that there are at most 4 slots per row (see Figure 3.1). Only print the occupied slots, or alternatively, until the row of the last occupied slot.



```
                BOX

   001      002      003      004
  GENGR    PICHU    PKCHU    GSTLY

   005      006      007      008
  ALCRM    GARDE    CRAMO    CINDR

   009      010
  SNIVY    CHNSY    EMPTY    EMPTY

[SELECT SLOT]:
```

Figure 3.1 Sample Box Display with 10 Occupied Slots

The box should start off as empty when no Pokémon has been caught. It should have a maximum capacity of 100, with the first slot number starting at 1. Newly added Pokémon appear after the last occupied entry in the box.

The box must have the following **`SEARCH`** features described below:

- **Search Pokémon by Name**
  Accessing this feature allows users to type a search string with at least 1 character and at most 11 characters. The program should validate the search string to show an error message if the user input does not follow these limitations.

  After the user enters a valid search string, this feature should reprint the box contents to only show slots containing Pokémon whose name matches the search string (regardless of capitalization). If no slots match the search string, then display **`"NO MATCHES FOUND"`** (or any similar message) instead. Note that this feature does **not** include partial matches in its search (e.g. a search string of "**`ABRA`**" can display Pokémon with the name of "**`Abra`**" but not Pokémon with the name of "**`KADABRA`**").

- **Search Pokémon by Short Name**
  Accessing this feature allows users to type a search string with exactly 5 characters. The program should validate the search string to show an error message if the user input does not follow this limitation.

  After the user enters a valid search string, this feature should reprint the box contents to only show slots containing Pokémon whose short name matches the search string (regardless of capitalization). If no slots match the search string, then display "**`NO MATCHES FOUND`**" (or any similar message) instead.

The box must also have the following **`SORT`** features:

- **Sort Pokémon by Short Name**
  Since newly caught Pokémon get added at the end of the box by default, accessing this feature lets users sort the Pokémon in the box by short name. Users may select whether they wish to sort the box in ascending (A-Z) or descending (Z-A) order. Note that any caught Pokémon after the box has been sorted still gets added at the end of the box. Users must re-sort the box contents if they wish to place newly caught Pokémon in the proper order.

Every time the box is reprinted due to searching or sorting, it should still include the *slot number*, *name*, and *short name* of each Pokémon being displayed. A Pokémon's slot number should remain the same even after the box has been searched. However, a Pokémon's slot number **may** change after the box has been sorted, or after another Pokémon has been deleted (see Release).

Along with displaying, searching, and sorting the box contents, this feature should also allow users to select one of the occupied slot numbers. Doing so will give the user 3 options, namely: **`VIEW ENTRY`**, **`RELEASE`**, and **`CANCEL`**. Inputting **`EXIT`** or **`-1`** instead of a slot number should return the user to the main menu.

- **View Entry**
  Displays the full contents of the selected Pokémon's dex entry. The entry shown by this feature must match the entry of the same Pokémon from the *View Dex Entries* feature.

- **Release**
  This is equivalent to deleting the selected Pokémon from the box. Make sure to have a confirmation prompt first before doing the deletion process. After successfully deleting a Pokémon, the remaining Pokémon at the latter slots should adjust to occupy the recently emptied slot. There should be no empty slots before and in-between occupied slots.

- **Cancel**
  Returns to the previous set of options (allowing the user to re-enter a different slot number or type **`EXIT`** to return to the main menu).

IV. Settings

This feature allows the user to **`SAVE`** their current progress or **`LOAD`** an existing file. The generated saved file should be in the form of a text file.

- **Save**

  Users must be able to save the Fakédex entries, including each entry's current information, along with the complete box contents, into a text file. Users should be asked to input their preferred filename first, which has to end with the extension "**`txt`**". Selecting a filename of an existing save file will overwrite it.

  Both the dex and the box information will be in a single text file. Each section end, meaning after the last entry for that section, a "**`--------- DEX END ---------`**" and "**`--------- BOX END ---------`**" marker should be placed to signify the end of the dex and the box sections respectively. The stored information for each section should follow the specified format (see Table 4.1).

	TEXT FILE FORMAT	SAMPLE FILE CONTENTS
1	`SLOT: <dex index><\n>`	`SLOT: 0`
2	`NAME: <name><\n>`	`NAME: Gengar`
3	`SHORT NAME: <short_name><\n>`	`SHORT NAME: GENGR`
4	`DESCRIPTION: <description><\n>`	`DESCRIPTION: A spooky ghost.`
5	`GENDER: <gender><\n>`	`GENDER: MALE`
6	`CAUGHT: <caught><\n>`	`CAUGHT: NO`
7	`<\n>`	
8	.	`SLOT: 1`
9	.	`NAME: Chansey`
10	.	`SHORT NAME: CHNSY`
11	`SLOT: <dex index><\n>`	`DESCRIPTION: The lucky egg.`
12	`NAME: <name><\n>`	`GENDER: FEMALE`
13	`SHORT NAME: <short_name><\n>`	`CAUGHT: YES`
14	`DESCRIPTION: <description><\n>`	
15	`GENDER: <gender><\n>`	`--------- DEX END ---------`
16	`CAUGHT: <caught><\n>`	`SLOT: 0`
17	`<\n>`	`NAME: Dragapult`
18	`--------- DEX END ---------`	`SHORT NAME: DRAGA`
19	`SLOT: <box index><\n>`	
20	`NAME: <name><\n>`	`SLOT: 1`
21	`SHORT NAME: <short_name><\n>`	`NAME: Luxray`
22	`<\n>`	`SHORT NAME: LXRAY`
23	.	
24	.	`--------- BOX END ---------`
25	.	
26	`SLOT: <box index><\n>`	
27	`NAME: <name><\n>`	
28	`SHORT NAME: <short_name><\n>`	
29	`<\n>`	
30	`--------- BOX END ---------`	

Table 4.1 Save File Content Format

- **Load**

  Users should be able to load any text file generated by the **`SAVE`** feature. Users should specify the filename, with the "**`txt`**" extension, of the file they wish to load. After loading a file, it should **overwrite** the contents of the dex and box arrays in the program. The overwritten contents of those features won't be accessible anymore, unless the user saved it prior to loading a file. Make sure to warn the user first then ask if they wish to proceed.

V. Exit

Users should be able to exit the program when they type **`EXIT`** in the main menu. Exiting the program will clear the contents of the dex and box features, unless saved by the user prior to closing. Note that this feature should **not** delete any generated save files.

NAVIGATION

On top of the requirements stated above, students must ensure that users will always have a way to navigate the program at any given time. Users should **never** get stuck inside a feature that forces them to manually close the program, or give a significant amount of input, in order to get out of it (e.g. The box feature does not have an exit command, which prevents the user from returning to the main menu. Or the player can only exit the exploration feature once the box limit has been reached). You may choose to use either the **`EXIT`** string or **`-1`** as an input to exit any feature that does not have an exit command explicitly stated in the specs. Make sure that your program displays what the exit command value is whenever asking for user input.

UI FEEDBACK

Make sure that users can understand what input your program is asking for at any given time. Also make sure that the user receives feedback (through print statements) for all necessary changes (e.g. “Pokémon released”), updates (e.g. “Pokémon caught and added to the box”), and prompts (e.g. “File saved”) made by your program.

BONUS

A maximum of 10 points may be given for features over & above the requirements, such as, but not limited to:

1. Allowing users to re-arrange the box contents by inputting two slot numbers and swapping them. This change should also reflect in the save file, whenever the user uses the save feature.

2. Giving users an option to assign unique nicknames to Pokémon right after being caught, and allowing users to search Pokémon in the box via their nicknames. An additional box feature should also be added to allow users to change these nicknames.

3. Integrating a Pokéball feature that limits the number of times the user can use the **`CATCH`** command during an exploration. Users should not be allowed to enter the exploration feature when they have zero Pokéballs at hand. Users will immediately be kicked out of the exploration feature when their held Pokéballs become zero. This feature must come with a **`SHOP`** feature that allows users to purchase or receive Pokéballs for free (this is up to the student, but if the purchase feature is chosen, then there should also be a way to gain and keep track of the user’s **`MONEY`**).

No bonus feature should change the requirements (subject to evaluation of the teacher). Required features must be completed first in order to receive any bonus point(s).

DEADLINE

Upload your submissions (see SUBMISSIONS) to the Canvas assignment page. Submission deadline is June 17, 2022 (F), 12:00PM. The submission page will remain open until June 20, 12:00PM. Please take note of the corresponding deductions for late submissions:

START (DATE, TIME)	END (DATE, TIME)	DEDUCTION
June 17, 12:01PM	June 18, 12:00PM	**20%**
June 18, 12:01PM	June 19, 12:00PM	**40%**
June 19, 12:01PM	June 20, 12:00PM	**60%**

Submissions from June 20, 12:01PM will no longer be accepted and will receive a grade of 0 for the MP.

WORKING IN PAIRS
You may work on this project individually or in pairs. If you decide to work with a pair, take note of the following policies below:

1. You must first sign-up with your pair through the Canvas MP Groupings page. You are not allowed to switch pairs with other students once the deadline for forming pairs has passed.
2. Each student must create their own **INDIVIDUAL** solution to the MP, **WITHOUT** discussing it with their pair.
3. After both students are done with their individual solution, they can discuss and compare it with their pair's solution to create their *final solution*. The final solution may be the solution of one of the students in the pair, or an improved version of both solutions.
4. Make sure to include in the **internal documentation** whose solution was used, or whether it was an improved version of both solutions.
5. Only the final solution should be uploaded in Canvas. Only one member is needed to upload this submission.
6. Since each student is expected to have provided an individual solution prior to creating the final solution, both members should know all parts of the final code to be able to individually complete the demo problem within a limited amount of time (to be announced nearer the demo schedule). This demo problem is given only on the day/time of the demo, and may be different per member.
7. Both students should be present during the demo, not just to present their individual demo problem solution, but also to answer questions pertaining to their group submission.
8. Students in the same pair will receive the same MP grade, unless there is a compelling and glaring reason for one student to get a different grade than the other (e.g. one student is unable to properly answer questions regarding the solution, or modify the code according to the given demo problem). These will be handled on a case-to-case basis.

Students working on their project individually are **not** allowed to ask any other student for help in any form. Students working in pairs are **only** allowed to discuss their solutions with their assigned partner **after** completing it individually.

SUBMISSIONS

Make sure that your submission conforms to the following:

1. The program implementation observes proper use of arrays, strings, structures, files, and user-defined functions whenever appropriate, and despite not being explicitly stated in the specs.
2. Brute force was **not** used in the program solution, and proper use of loops and conditional statements were applied.
3. The following were **not** used in the program solution: `GOTO` statements, `exit()`, `break` (outside of switch statements), `continue`, global variables, and calling `main()`.77
4. Internal documentation is present in the code and follows the format shown in the sample below.

SAMPLE DOCUMENTATION

```
/* getDescription returns the description to be displayed in the dex
   @param pPokemon - pointer pointing to a single Pokemon structure
   @return "???" if the pokemon has not been caught, otherwise,
   return the actual description value
   Pre-condition: pPokemon is not null
*/

char[] //function return type is in a separate line from the rest
getDescription(Pokemon *pPokemon) //use prefix for identifiers
{ //open brace is on a new line, aligned with matching close brace
   int caught; /* declare all variables before any statements -
                  not inserted in the middle or inside a loop -
                  for readability */

   caught = pPokemon->caught;
   if(caught == 0)
   { //open brace at new line
      return "???";
   }

   return pPokemon->description;
}
```

Any requirement not fully implemented or instruction not followed will merit deductions. Place the following files in a ZIP archive, to be submitted in Canvas:

- Source Code
- Test Script (see TEST SCRIPTS)
- Sample Save File (exported by your program)

TEST SCRIPTS

Test Script should be in a table format. There should be three categories of test cases per function. A sample test script table is shown below.

FUNCTION	CASE	DESCRIPTION	INPUT	OUTPUT		P / F
				EXPECTED	ACTUAL	

`sortBox`	1	Names in the array are in ascending order.	`aBox = {`
			`“Chansey”,`
			`“Dragapult”,`
			`“Gengar”`
			`}`
				`aBox = {`
				`“Gengar”,`
				`“Dragapult”,`
				`“Chansey”`
				`}`
					`aBox = {`
					`“Gengar”,`
					`“Dragapult”,`
					`“Chansey”`
					`}`
						P

	2	Names in the array are in descending order.	`aBox = {`
			`“Gengar”,`
			`“Dragapult”,`
			`“Chansey”`
			`}`
				`aBox = {`
				`“Chansey”,`
				`“Dragapult”,`
				`“Gengar”`
				`}`
					`aBox = {`
					`“Gengar”,`
					`“Dragapult”,`
					`“Chansey”`
					`}`
						F

	3	Names in the array are a mix of letters and numbers.	`aBox = {`
			`“Unown2”,`
			`“Unown1”,`
			`“1Unown”`
			`}`
				`aBox = {`
				`“1Unown”,`
				`“Unown1”,`
				`“Unown2”`
				`}`
					`aBox = {`
					`“Unown1”,`
					`“Unown2”,`
					`“1Unown”`
					`}`
						F

Do not include functions that are only used to display the Text-based UI.

DEMO POLICIES
Take note of the policies stated below:

1. A sign-up sheet will open a week before the start of the demo. Once the demo week has started, you are not allowed to change your schedule anymore. Make sure that the schedule you chose does not conflict with any major exams or requirements from your other courses.

2. You are expected to arrive on your scheduled demo slot on time. You should be able to answer any questions related to the output and implementation of your program, and modify your code according to the given demo problems. Failure to meet these requirements may result in a grade of 0 for the project.

3. You will be running the latest version of the program you submitted on the MP submission page. You are **not** allowed to modify your code in any way, aside from the required modifications for the demo problems given by your instructor.

4. Your program is expected to run during the MP demo, without needing any modifications. If your program does not run, the grade for your project is automatically 0.

5. A running program with complete features does not automatically mean a perfect grade for the project. Your program will still be checked for complying with restrictions stated in this document, as well as proper test script results, and presence of internal documentation.

Further demo policies and the demo sign-up sheet will be announced through Canvas.

ACADEMIC HONESTY

Any form of cheating (e.g. asking other people besides your pair for help, submitting part of another pair or student's work as your own, sharing your algorithm or code to other students not in the same pair, etc.) can be punishable by a grade of 0.0 for the course & a discipline case.

Comply with the DLSU Policy on Academic Honesty (Student Handbook Section 4.15). Academic dishonesty (i.e. cheating) is a major offense (Student Handbook Section 5.3.1.1). Please make sure that you read the details of these sections (see Student Handbook).