College of Computer Studies
De La Salle University
CCPROG2 Machine Project
Pokedex

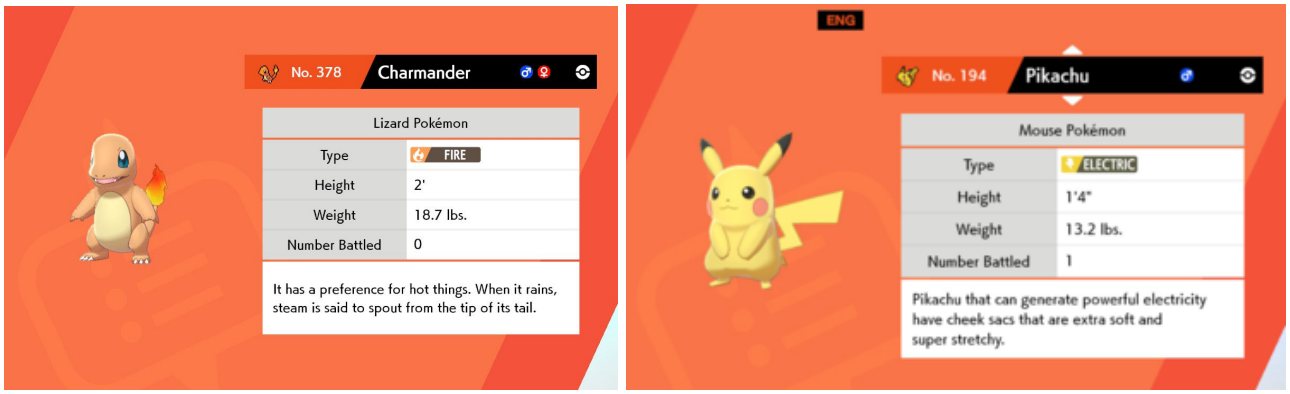
*Professor Laventon*

Hi there! I am Professor Laventon from the Hisui Region. We live in a world where strange and powerful creatures exist called Pokemon. We are on a mission to finish the very first ever Pokedex (Pokemon Index) which contains a list of Pokemon & description from this region. Problem is we're having a hard time compiling all this data. Thankfully, you volunteered to help us automate this very tedious process (*Thanks in advance!*)

As a new member of the Galaxy Expedition Team (or Galaxy Team in short), you need to develop a tool called a **Pokedex** (Pokemon Index).

A Pokedex is designed to catalog and provide information regarding the various species of Pokemon. This will help us learn and track all the Pokemon we catch and study in the Hisiu region.

*Pokedex*
*(Pokemon Index)*




*Sample Pokedex Entry*

Given that this is our first time tracking Pokemon, let's keep it simple! We would like to note the following details of a pokemon: Name, Type, and Description. Important to note that:

- **Name** can have at most 20 letters each.
- **Type** can either be: Water, Fire, Grass or Electric. A Pokemon can only have 1 type.
- **Description** can have at most 50 letters each.

Below are 2 Pokedex Entry Samples. Note that there can be at most 150 Pokedex entries and cannot have duplicate entries.

| Name | Type | Description |
|---|---|---|
| Charmander | Fire | Dinosaur like Pokemon that prefers hot things |
| Squirtle | Water | Turtle Pokemon with large eyes and chubby cheeks |

Your Pokedex should allow the user to **Manage Data**, by providing a user-friendly interface, as well as the functionality to do the following:

- ***Add Entry***
  Your Pokedex should ask for the necessary information (e.g. Name, Type, Description). It will first check if there is an existing pokemon entry based on the Name. If so, the Pokedex should prevent the user from creating the new entry. If there are no duplicates, then a new entry is created. However, the Pokedex may ask the user if he wants to encode another pokemon entry or to use a sentinel value to terminate the input series. All necessary information should each have at least 1 character.

- ***Modify Entry***
  Your Pokedex should allow users to modify your Pokedex Entries. This option will first display the list of Pokemon entries (Follow [Display All Entries] format) entered before asking which entry you want to modify. The input for this is a number referring to the index of the Pokemon's entry. Note that the first entry is referred to as entry 1, but should be stored in index 0. If an invalid number is given, a message is displayed before going back to the Manage Data menu. After this, your Pokedex should ask the user which information to edit using numbers (**1** - Name, **2** - Type or **3** - Description). Your Pokedex will then ask the user to enter the updated details & refresh the data as needed. Make sure to double check if there is any conflicting/incorrect information before saving.

- ***Delete Entry***
  Similar to the modify functionality, this option will first display the list of Pokemon entries (Follow [Display All Entries] format) entered before asking which entry you want to delete. The input for this is a number referring to the index of the Pokemon's entry. Note that the first entry is referred to as entry 1, but should be stored in index 0. If an invalid number is given, a message is displayed before going back to the Manage Data menu. After this, your Pokedex should delete this entry.

- ***Display All Entries***
  Provide a list of all Pokedex entries. An entry is displayed by providing the entry number of the Pokedex Entry (note: entry number starts at 1, index number starts at 0), the Pokemon Name, Type, and Description. Make sure that garbage (uninitialized) values are not displayed by this list. Do not display all 150 entries if there are only 5 entries initialized.

- ***Search Pokemon by Name***
  This option first asks for an input word, then the Pokedex proceeds to show a listing of all entries where that input word appears in the Pokemon Name. Display is similar to how the Display All Entries work (with a way to view next, previous, and exit) but only those that matches the given word. [For example, if the word to be searched is **chu**, then **Pikachu** and **Raichu** should be shown.]. If no word matches, a message should be shown prior to reverting back to the Manage Data menu.

- ***Search Pokemon by Type***
  This option is similar to *[Search Pokemon] by Name*, except this displays the ***list of all Pokemons under the entered type.*** Display is similar to how the *[Display All Entries]* work. For example, if the users search for Pokemon with type *Water*, the user should be seeing a list, not just one example of pokemon on said type. There should be a message showing the end of the list (e.g. *These are all of the Water type Pokemon)*. Alternatively, if there is no match, a message should be shown stating that no match is found.

- ***Export***
  Your Pokedex should allow all data to be saved into a text file. The data stored in the text file can be used later on. The PokeDex should allow the user to specify the filename. Filenames

have at most 30 characters including the extension. If the file exists, the data will be overwritten.

This is a sample content of the text file. Make sure to follow the format. Those that are in <> are supposed to be replaced with the character. No <> will be saved in the text file. No additional content should be in stored in the text file. The user should explicitly choose this option for all modifications done (within the options of Manage Data) to be reflected (or stored) to the text file.

```
Name: Pikachu<next line>
Type: Electric<next line>
Description: Mouse-like Pokemon that have yellow fur<next line>
<nextline>
Name: Squirtle<next line>
Type: Water<next line>
Description: Turtle Pokemon with large eyes and chubby
cheeks<next line>
<nextline>
<end of file>
```

- ***Import***

Your Pokedex should allow the data stored in the text file to be added to the list of entries in the Pokedex. The user should be able to specify which file (input filename) to load. If there are already some entries added (or loaded previously) in the current run, the Pokedex shows one entry loaded from the text file and asks if this is to be added to the list of entries (in the array). If yes, it is added as another entry. If no, this entry is skipped. Whichever the choice, the Pokedex proceeds to retrieve the next entry in the file and asks the user again if this is to be included in the array or not, until all entries in the file are retrieved. The data in the text file is following the format indicated in Export.

- ***Exit***

The exit option will just allow the user to quit the Manage Data menu. Only those exported to files are expected to be saved. The information in the lists should be cleared after this option. The Pokedex goes back to the Main Menu.

**Research Tasks**

We need to know more about these Pokemon! That's why our Galaxy Team will need to log and review their research tasks every time they get back from their Pokemon adventure. Our Pokedex should allow us to track how many Pokemon have been seen and defeated by our team.


*Sample Research Task*

Once data is loaded, the following menu options are shown:

- ***Review Research Task per Pokemon***
  In this feature, the Pokedex displays all the stats based on the finished research tasks for a Pokemon (see above picture for an example). The feature will first display the list of Pokemon similar to how [Display All entries] work. The user will input the pokemon entry number in order to see all research task progress for said pokemon.

- ***Review Research Task per Task Type***
  In this feature, the Pokedex displays progress for all pokemons for a single task type. The feature will first ask for the research type via number input (1 - Seen, 2 - Defeated). Afterwards, it will display a list of the progress for each pokemon. If the progress counter is currently 0 for a pokemon, do not include them in the display. There should be a message showing the end of the list. Alternatively, if there is no match, a message should be shown stating that no match is found.

- ***Update Research Task***
  In this feature, the Pokedex will ask the user to input what type of research task was completed. The user will input in number (1 - Seen, 2 - Defeated). The Pokedex will then show the list of Pokemon similar to how [Display All entries] work. The input for this is a number referring to the index of the Pokemon's entry. Note that the entry number should start at 1, but the index number should start at 0. If an invalid number is given, a message is displayed before going back to the Research Task menu. After this, the Pokedex should ask the user how many are seen/defeated and should add the number inputted to the existing research task.

- ***Exit***
  The exit option will just allow the user to quit to the Language Tools menu. The information in the lists should be cleared after this option. The Pokedex then reverts back to the Main Menu.

The Pokedex terminates when the user chooses to exit from the Main Menu.

***Bonus***

A **maximum** of **10 points** may be given for features **over & above** the requirements, like (1) producing the top 5 pokemon (sorted by ranking) with the highest research progress (based on total of each task); (2) allowing the addition of additional task types. **Required features** must be **completed first** before bonus features are credited. Note that use of conio.h, or other advanced C commands/statements may **not** necessarily merit bonuses.

***Submission & Demo***

**Final MP Deadline: June 17, 2022 (F), 1200noon via Canvas**. After the indicated time, the submission page will remain open for submission until June 20, 1200noon only but for every day late, there will be a 20% deduction in the MP grade. Note that any amount of time (even a few seconds after 12noon of June 17) will already be considered the next day. Thus, submissions from June 17 12:01PM to June 18 12:00PM (noon) will incur 20% deductions; submissions from June 18, 12:01PM to June 19, 12:00PM will incur 40% deductions; submissions from June 19, 12:01PM until June 20, 12PM will incur 60% deductions in the MP grade. At June 20, 12:01PM, the submission page will be locked and thus considered no submission (equivalent to 0 in the MP grade).

**Requirements:** Complete Program

- Make sure that your implementation has considerable and proper use of arrays, strings, structures, files, and user-defined functions, as appropriate, even if it is not strictly indicated.
- It is expected that each feature is supported by at least one function that you implemented. Some of the functions may be reused (meaning you can just call functions you already implemented [in support] for other features. There can be more than one function to perform tasks in a required feature.
- Debugging and testing was performed exhaustively. The Program submitted has
  a. NO syntax errors
  b. NO warnings - make sure to activate -Wall (show all warnings compiler option) and that C99 standard is used in the codes
  c. NO logical errors -- based on the test cases that the Program was subjected to

***Important Notes:***

1. Use **gcc -Wall** to compile your C Program. Make sure you **test** your Program completely (compiling & running).
2. Do not use brute force. Use **appropriate conditional** statements **properly**. Use, **wherever appropriate**, **appropriate loops** & **functions properly**.
3. You **may** use topics outside the scope of CCPROG2 but this will be **self-study**. Goto *label*, exit(), break (except in switch), continue, global variables, calling main() are **not allowed**.
4. Include **internal documentation** (comments) in your Program.
5. The following is a checklist of the deliverables:

**Checklist:**

- ❒ Upload via AnimoSpace submission:
  - ❒ source code*
  - ❒ test script**
  - ❒ sample text file exported from your Program
- ❒ email the softcopies of all requirements as attachments to **YOUR** own email address on or before the deadline

Legend:

***** Source code exhibit readability with supporting inline documentation (not just comments before the start of every function) and follows a coding style that is similar to those seen in the course notes and in the class discussions. The first page of the source code should have the following declaration (in comment) [replace the pronouns as necessary if you are working with a partner]:

/**********************************************************************************************

This is to certify that this project is my own work, based on my personal efforts in studying and applying the concepts learned. I have constructed the functions and their respective algorithms and corresponding code by myself. The

Program was run, tested, and debugged by my own efforts. I further certify that I have not copied in part or whole or otherwise plagiarized the work of other students and/or persons.

<your full name>, DLSU ID# <number>
**********/

Example coding convention and comments before the function would look like this:

```
/* funcA returns the number of capital letters that are changed to small letters
   @param strWord - string containing only 1 word
   @param pCount - the address where the number of modifications from capital to small are
                   placed
   @return 1 if there is at least 1 modification and returns 0 if no modifications
   Pre-condition: strWord only contains letters in the alphabet
*/
int     //function return type is in a separate line from the
funcA(char strWord[20] ,    //preferred to have 1 param per line
      int * pCount)            //use of prefix for variable identifiers
{  //open brace is at the beginning of the new line, aligned with the matching close brace
   int    ctr;        /* declaration of all variables before the start of any statements –
                         not inserted in the middle or in loop- to promote readability */

   *pCount = 0;
   for (ctr = 0; ctr < strlen(strWord); ctr++)  /*use of post increment, instead of pre-
                                                  increment */
   {  //open brace is at the new line, not at the end
      if (strWord[ctr] >= 'A' && strWord[ctr] <= 'Z')
      {    strWord[ctr] = strWord[ctr] + 32;
           (*pCount)++;
      }
      printf("%c", strWord[ctr]);
   }

   if (*pCount > 0)
      return 1;
   return 0;
}
```

**Test Script should be in a table format. There should be at least 3 categories (as indicated in the description) of test cases **per function**. There is no need to test functions which are only for screen design (i.e., no computations/processing; just printf).

Sample is shown below.

| Function | # | Description | Sample Input Data | Expected Output | Actual Output | P/F |
|---|---|---|---|---|---|---|
| sortIncreasing | 1 | Integers in array are in increasing order already | aData contains: 1 3 7 8 10 15 32 33 37 53 | aData contains: 1 3 7 8 10 15 32 33 37 53 | aData contains: 1 3 7 8 10 15 32 33 37 53 | P |
| | 2 | Integers in array are in decreasing order | aData contains: 53 37 33 32 15 10 8 7 3 1 | aData contains: 1 3 7 8 10 15 32 33 37 53 | aData contains: 1 3 7 8 10 15 32 33 37 53 | P |
| | 3 | Integers in array are combination of positive and negative numbers and in no particular sequence | aData contains: 57 30 -4 6 -5 -33 -96 0 82 -1 | aData contains: -96 -33 -5 -4 -1 0 6 30 57 82 | aData contains: -96 -33 -5 -4 -1 0 6 30 57 82 | P |

Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested. Given the sample code in page 6, the following are four distinct classes of tests:

i.) testing with strWord containing all capital letters (or rephrased as "testing for at least 1 modification")

ii.) testing with strWord containing all small letters (or rephrased as "testing for no modification")

iii.) testing with strWord containing a mix of capital and small letters
iv.) testing when strWord is empty (contains empty string only)

The following test descriptions are **incorrectly formed**:

- Too specific: testing with strWord containing "HeLlo"
- Too general: testing if function can generate correct count OR testing if function correctly updates the strWord
- Not necessary -- since already defined in pre-condition: testing with strWord containing special symbols and numeric characters

6. Upload the softcopies via Submit Assignment in Canvas. You can submit multiple times prior to the deadline. However, only the last submission will be checked. Send also to your **mylasalle account** a **copy** of all deliverables.
7. Use **<surnameFirstInit>.c** & **<surnameFirstInit >.pdf** as your filenames for the source code and test script, respectively. You are allowed to create your own modules (.h) if you wish, in which case just make sure that filenames are descriptive.
8. During the MP **demo**, the student is expected to appear on time, to answer questions, in relation with the output and to the implementation (source code), and/or to revise the Program based on a given demo problem. Failure to meet these requirements could result to a grade of 0 for the project.
9. It should be noted that during the MP demo, it is expected that the Program can be compiled successfully and will run. If the Program does not run, the grade for the project is automatically 0. However, a running Program with complete features may not necessarily get full credit, as implementation (i.e., code) and other submissions (e.g., non-violation of restrictions evident in code, test script, and internal documentation) will still be checked.
10. The MP should be an HONEST intellectual product of the student/s. For this project, you are allowed to do this individually or to be in a group of 2 members only. Should you decide to work in a group, the following mechanics apply:

**Individual Solution:** Even if it is a group project, each student is still required to create his/her INITIAL solution to the MP individually without discussing it with any other students. This will help ensure that each student went through the process of reading, understanding, solving the problem and testing the solution.

**Group Solution:** Once both students are done with their solution: they discuss and compare their respective solutions (ONLY within the group) -- note that learning with a peer is the objective here -- to see a possibly different or better way of solving a problem. They then come up with their group's final solution -- which may be the solution of one of the students, or an improvement over both solutions. Only the group's final solution, with internal documentation (part of comment) indicating whose code was used or was it an improved version of both solutions) will be submitted as part of the final deliverables. It is the group solution that will be checked/assessed/graded. Thus, only 1 final set of deliverables should be uploaded by one of the members in the Canvas submission page. [Prior to submission, make sure to indicate the members in the group by JOINing the same group number.]

**Individual Demo Problem:** As each is expected to have solved the MP requirements individually prior to coming up with the final submission, both members should know all parts of the final code to allow each 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 of the group. Both students should be present/online during the demo, not just to present their individual demo problem solution, but also to answer questions pertaining to their group submission.

**Grading:** the MP grade will be the same for both students -- UNLESS there is a compelling and glaring reason as to why one student should get a different grade from the other -- for example, one student cannot answer questions properly OR do not know where or how to modify the code to solve the demo problem (in which case deductions may be applied or a 0 grade may be given -- to be determined on a case-to-case basis).

11. Any form of **cheating** (**asking other people not in the same group for help**, **submitting as your [own group's] work part of other's work**, **sharing your [individual or group's] algorithm and/or code to other students not in the same group, etc.**) can be punishable by a grade of **0.0** for the **course** & a **discipline case**.

**Any requirement not fully implemented or instruction not followed will merit deductions.**

----------There is only **1 deadline** for this project: June 17, 12:00noon, but the following are **suggested** targets.----------
*Note that each milestone assumes fully debugged and tested code/function, code written following coding convention and included internal documentation, documented tests in test script.

1.) Milestone 1 : April 22
    a. Menu options and transitions
    b. Preliminary outline of functions to be created

2.) Milestone 2: May 20
    a. Add Entry
    b. Display all entries
    c. Search Pokemon by Name
    d. Search Pokemon by Type

3.) Milestone 3: May 30
    a. Modify Entry
    b. Delete Entry
    c. Update Research Task
    d. Review Research Task per Pokemon
    e. Review Research Task per Task Type

    [+]For these features, do not clear the contents of the array of entries upon exit from Manage Data menu IF you have not implemented the import and export yet.

4.) Milestone 4: June 10
    a. Export
    b. Import

5.) Milestone 5: June 17
    a. Integrated testing (as a whole, not per function/feature)
    b. Collect and verify versions of code and documents that will be uploaded
    c. Recheck MP specs for requirements and upload final MP
    d. [Optional] Implement bonus features