COMP1917 Computing 1

Session 2, 2016

Assignment 2- Pokedex

Due: Midnight end of Sunday 23rd October (end of Week 12)
Marks: 20% of final assessment

1 Preliminary

For this assignment, you'll implement a simplified Pokedex.

1.1 What is a Pokemon? What is a Pokedex?

1.2 Where can we learn more about Pokedexes?

- You can play the one of the original Pokemon games here. Quite early in the game you get introduced to the Pokedex. http://emulator.online/gameboy/pokemon-blue-version/
- More information: http://pokemon.wikia.com/wiki/Pok%C3%A9dex

2 Setting Up the Assignment

You are provided with several files with some of the code implemented for you.

- *Pkmn.h*: Contains typedefs, #defines and function prototypes. *Do not edit this file.*
- *Pkmn.c*: Contains the start of the function implementations. Only add code where the line `ADD YOUR CODE HERE` appears.
- *main.c*: Contains the start of an implementation which accepts commands from the user and calls the functions in Pkmn.h to create a functional application.

3 Implementation Stages

3.1 Stage 0: Main Menu

Commands:

- `a - Add a Pokemon to the list`
  Allow the user to enter the details of a Pokemon and adds that Pokemon to the list.
- `p - Print list`
  Prints the list of Pokemon.
- `d - Display details of the current Pokemon`

- `> - Move to the next Pokemon`
- `< - Move to the previous Pokemon`
- `j - Jump to a specific Pokemon`
- `r - Remove the current Pokemon`
- `f - Find Pokemon`
- `c - Count the number of Pokemon which have been found`
- `e - Add an evolution to a Pokemon`
- `s - Show the evolutions of the current Pokemon`
- `q - Quit`

3.2 Stage 1: Adding Pokemon and Printing the List

Functions to implement:

- `Pkmn createPkmn(int id, char * name, double height, double weight, int type1, int type2);`

  When a Pokemon is first entered into the Pokedex, the following information needs to be stored:

  - The id or number of the Pokemon
  - Name
  - Height in metres
  - Weight in kgs
  - The type of the Pokemon

  When a Pokemon is first added to the Pokedex, it has not yet been 'found'.

- `PkmnList createPkmnList();`

  What you store in your 'list' struct is entirely up to you. Store whatever you need to to help you implement the functionality required. (The same goes for your 'pkmn' struct - the design is up to you.)

- `void freePkmnList(PkmnList list);`
- `void addPkmnToList(PkmnList list, Pkmn pkmn);`

  New Pokemon should be added to the end of the existing list.

- `void printPkmnList(PkmnList list);`

  When printing a whole list of Pokemon, only display the number and name in the following format. (Using asterisks for Pokemon which have not been found, and full names for Pokemon which have been found.)

  Also required, is an indication of the 'current' Pokemon. When the Pokemon are initially added to the list, the 'current' Pokemon is the first one in the list. The 'current' Pokemon is indicated using an arrow pointing to the corresponding row in the list.

```
--> #001: Bulbasaur
    #004: **********
    #007: ********
    #010: Caterpie
    #013: Weedle
    #016: Pidgey
```

- `void printCurrentPkmn(PkmnList list);`

  This function should print out the 'current' Pokemon of the provided list.

  When printing out the details of a specific Pokemon, if the Pokemon has been 'found', then all the details are printed out. If the Pokemon hasn't been 'found', then only the id is printed out, and the name is printed out as asterisks.

  For example, for a Squirtle that has not been found:

  ```
  Id: 007
  Name: ********
  Height: --
  Weight: --
  Type: --
  ```

  And a Squirtle after it has been found:

  ```
  Id: 007
  Name: Squirtle
  Height: 0.51m
  Weight: 9.0kg
  Type: Water
  ```

  Note: All the data is stored in the list from the time the struct is created. It is only how the data is displayed that changes.

3.3 Stage 2: Navigating the List

- `void nextPkmn(PkmnList list);`

  Moves the current Pokemon to be the next one in the list. If the current Pokemon is already the last Pokemon in the list, then the current Pokemon should remain unchanged.

- `void prevPkmn(PkmnList list);`

  Moves the current Pokemon to be the previous one in the list. If the current Pokemon is already the first Pokemon in the list, then the current Pokemon should remain unchanged.

- `void jumpToPkmn(PkmnList list, int id);`

  Moves the current Pokemon to be the one with the given id. If the id provided does not match a Pokemon in the list, then the current Pokemon should remain unchanged.

- `void removePkmn(PkmnList list);`

  Removes the current Pokemon from the list. The current Pokemon should be updated to indicate the Pokemon after the Pokemon being removed. If the Pokemon being removed is at the end of the list, then the newly current Pokemon should be the Pokemon at the end of the list.

3.4 Stage 3: Finding Pokemon

- `void findPkmn(int seed, int factor, int numberOfNewPkmn, PkmnList list);`

  This function emulates what happens when a user goes out exploring and comes across different Pokemon. When exploring, some of the Pokemon they come across are Pokemon they've already found previously. Some of them are Pokemon that they're finding for the first time.

  The parameters work in the following way:

- `seed` - Is used to start a random number generator.
- `factor` - Random numbers are generated between 0 and the `factor` provided (not including the `factor` itself). The generated number is the id of the Pokemon that has been found. If the Pokemon has been previously found, then nothing happens. If the Pokemon was previously undiscovered (had never been found) then this Pokemon is now found.
- `numberOfNewPkmn` - This number is the number of **new** Pokemon which are found using the process described above.

- `int totalFound(PkmnList list);`

  This function returns the number of Pokemon in the list which have been found.

3.5 Stage 4: Evolutions

As a Pokemon ages and gains experience, it sometimes transforms into another Pokemon. This is called an Evolution.

For example: #007 Squirtle, turns into a #008 Wartortle, which then turns into a #009 Blastoise. (See http://pokemondb.net/pokedex/squirtle→ Evolution chart, for more information.)

- `void addEvolution(PkmnList list, int pkmnId, int evolutionId);`

  Add the information that the Pokemon with `pkmnId` can evolve into the Pokemon with the id `evolutionId`.

  If the Pokemon with id `pkmnId` is not in the list, then the list remains unchanged.

- `void showEvolutions(PkmnList list);`

  Show the evolutions of the current Pokemon. It should include the Pokemon it evolves into (if any), as well as any evolutions that its evolved state can evolve into, and so on.

3.6 Stage 5: Getting Sub-Lists

- `PkmnList getPkmnOfType(PkmnList list, int type);`
  - Creates a new PkmnList containing only the Pokemon from the original list which are of the specified type.
  - If a Pokemon has more than one type, only one of their types has to match the given type to be included in the list.
  - This function should not alter the original list. It should create copies of any Pokemon which are in the original list, and insert those into the new list.
  - The Pokemon should appear in this list in the same order as they appeared in the original list.

- `PkmnList getFoundPkmn(PkmnList list);`
  - Creates a new PkmnList containing only the Pokemon from the original list which have been found.
  - This function should not alter the original list. It should create copies of any Pokemon which are in the original list, and insert those into the new list.
  - The Pokemon should appear in this list in ascending id order (regardless of the order in which they appeared in the original list).

- `PkmnList searchByName(PkmnList list, char text[]);`

- Creates a new PkmnList containing only the Pokemon from the original list which have the given string appearing in its name. (Eg if the text provided is 'basau' then 'Bulbasaur' should be one of the Pokemon in the returned list.)
- The text provided is not case sensitive. (Eg if the text provided is 'bulb' then 'Bulbasaur' should be one of the Pokemon in the returned list.)
- This function should not alter the original list. It should create copies of any Pokemon which are in the original list, and insert those into the new list.
- Do not use any functions from the `string.h` library to complete this function.

4 Hints

4.1 File Redirection

Remember the lecture we had on file redirection. Every time you run your application, instead of re-typing all the details of each given Pokemon you're adding to the Pokedex, put the input for your application into a separate file and run it using:

```
./a.out < test1.in > test1.out
```

Then check that your file test1.out contains the correct information given the input file.

5 Submission

You must submit two ANSI C source file which must be called Pkmn.c and main.c which, when compiled with `-Wall` and `-Werror`, will produce an executable that performs exactly as described in the specification. Once submissions are open, you should submit by typing

```
give cs1917 assign2 main.c Pkmn.c
```

You can submit as many times as you like - later submissions will overwrite earlier ones. You can check that your submission has been received by using the following command:

```
1917 classrun -check
```

The submission deadline is Sunday 23rd October 23:59:59. If you submit the assignment after the deadline, you will lose 10% of the maximum marks for each day you are late. You'll get 0 marks if you submit more than five days late.

Additional information may be found in the FAQ and will be considered as part of the specification for the assignment. Questions relating to this assignment can also be posted to the comments section of the assignment 1 page of the course website.

If you have a question that has not already been answered in the FAQ or in the comments, you can email it to your tutor, or to `s.mautner@unsw.edu.au`.

6 Marking Scheme

Assignment 2 is worth 20 marks (which will make up 20% of the overall marks for the course).

The grades are broken down as follows:

- Pkmn.c Function Implementations: 15 marks

- main.c: 1 mark
- Style: 4 marks

  This assignment assesses the use of Linked Lists. Any assignment which fails to implement the application in a Linked List will get 0.