COMP1511 19T1

Assignment 2 - Pokédex

COMP1511 19T1

version: 1.10 last updated: 2019-04-28 07:40:00

Introduction

For this assignment, we are asking you to implement a mini Pokédex in C. The task is split into 5 sections; each section is not weighted the same.

What is a Pokémon? What is a Pokédex?

Hello there! Welcome to the world of Pokémon! My name is Oak! People call me the Pokémon Prof! This world is inhabited by creatures called Pokémon! For some people, Pokémon are pets. Others use them for fights. Myself ... I study Pokémon as a profession. — Professor Oak

Pokémon are fictional creatures from the Pokémon franchise, most famously from the Pokémon games. The game revolves around the (questionably ethical) capturing of these creatures. Within the fiction of this universe, a device called the **Pokédex** is used to catalogue all the creatures a player finds and captures.

Where can we learn more about Pokédexes?

You can play the one of the original Pokémon games here. Quite early in the game you get introduced to the Pokédex.

There's some more information on the Pokémon Wikia.

In addition, Google will be a great resource, as the topic has been extensively written up about.

Supplied Code

This zip file contains the files that you need to get started with the assignment. It contains the following files:

`pokedex.h`	contains declarations for all functions you need to implement for this assignment. **Don't change pokedex.h**
`pokedex.c`	contains stubs for all the functions you need to implement for this assignment. Put all your Pokédex code in pokedex.c.
`pokemon.h`	contains declarations for all functions you must call to create and manipulate Pokémon. **Don't change pokemon.h**
`pokemon.c`	contains the functions you must call to create and manipulate Pokémon. **Don't change pokemon.c**
`main.c`	contains a main function and other functions that allow you to interactively test the functions you implement in pokedex.c **Don't change main.c**
`test_pokedex.c`	contains a main function and other functions which are your starting point for a test framework to automatically test the functions you implement in pokedex.c. As you implement functions in pokedex.c you should add tests to test_pokedex.c. Only put testing code in test_pokedex.c.

You can also link and copy the supplied files into your CSE account using commands like these:

```
$ mkdir ass2
$ cd ass2
$ cp -n /web/cs1511/19T1/activities/pokedex/pokedex.c .
$ cp -n /web/cs1511/19T1/activities/pokedex/test_pokedex.c .
$ ln -s /web/cs1511/19T1/activities/pokedex/pokemon.[ch] .
$ ln -s /web/cs1511/19T1/activities/pokedex/main.c .
$ ln -s /web/cs1511/19T1/activities/pokedex/pokedex.h .
```

The **ln** commands create symbolic links to a file in class account, so you are always using the latest version.

Reference implementation

A reference implementation **1511 pokedex_reference** is available to help you understand the assignment.

```
$ 1511 pokedex_reference
===========================[ Pokédex ]===========================
            Welcome to the Pokédex!  How can I help?
=================================================================
Enter command: ?
  a [pokemon_id] [name] [height] [weight] [type1] [type2]
    Add a Pokemon to the Pokedex
  p
    Print all of the Pokemon in the Pokedex (in the order they were adde
d)
  d
    Display details of the currently selected Pokemon
  >
    Move the cursor to the next Pokemon in the Pokedex
  <
    Move the cursor to the previous Pokemon in the Pokedex
  m [pokemon_id]
    Move the cursor to the Pokemon with the specified pokemon_id
  r
    Remove the current Pokemon from the Pokedex
  x [seed] [factor] [how_many]
    Go exploring for Pokemon
  f
    Set the current Pokemon to be found
  c
    Print out the count of Pokemon who have been found
  t
    Print out the total count of Pokemon in the Pokedex
  e [pokemon_A] [pokemon_B]
    Add an evolution from Pokemon A to Pokemon B
  s
    Show evolutions of the currently selected Pokemon
  q
    Quit
  ?
    Show help
Enter command: a 1 Bulbasaur 0.7 6.9 poison grass
Added Bulbasaur to the Pokedex!
```

Your **pokedex.c** (compiled with the supplied pokemon.c and main.c) should match the behaviour of the reference implementation.

Provision of a reference implementation is a common method to provide an operational specification, and it's something you will likely need to do outside UNSW.

If you discover what you believe to be a bug in the reference implementation, report it in the class forum. We may fix the bug or indicate that you do not need to match the reference implementation's behaviour in this case.

Stage 1: Adding and Printing

When you compile and run the given code, you will get the following message if you try to add a Pokemon:

```
$ dcc -o pokedex main.c pokedex.c pokemon.c
$ ./pokedex
============================[ Pokédex ]============================
             Welcome to the Pokédex!  How can I help?
==================================================================
Enter command: a 1 Bulbasaur 0.7 6.9 poison grass
exiting because you have not implemented the add_pokemon function in poke
dex.c
```

When you run the `a` command, this function in pokedex.c is called:

```
void add_pokemon(Pokedex pokedex, Pokemon pokemon) {
    fprintf(stderr, "exiting because you have not implemented the add_pokemon function in pokedex.c\n");
    exit(1);
}
```

Your first task is to implement **add_pokemon**. All the functions you have to implement in pokedex.c have descriptions in pokedex.h:

```
// Add a new Pokemon to the Pokedex.
// Note: just adding the Pokemon to the Pokedex does not mean it has
// been 'found'.
//
// The new Pokemon should be added to the end of the Pokedex (i.e.
// directly after the Pokemon that was added when add_pokemon was last
// called).
//
// When the first Pokemon is added to the Pokédex, the currently
// selected Pokemon becomes this Pokemon.
//
// The currently selected Pokemon remains the first Pokemon that was
// inserted into the Pokedex, until the `change_current_pokemon`,
// `next_pokemon`, or `prev_pokemon` functions are called.
//
// If there is already a Pokemon in the Pokedex with the same pokemon_id
// as this Pokemon, the function should print an appropriate error
// message and exit the program.
//
// Pokedex Order:
// --------------
// The Pokemon in the Pokedex are stored in the order in which they were
// added, i.e. the first Pokemon in the Pokedex will be the first
// Pokemon that was added, the second Pokemon in the Pokedex will be the
// second Pokemon that was added etc, and the last Pokemon in the
// Pokedex will be the Pokemon that was added most recently.
//
// For example, if Pikachu (#025) was added into an empty Pokedex, the
// Pokedex would now contain Pikachu at the start of the list, and
// Pikachu would be the currently selected Pokemon.
//
// [Start of Pokedex]
// #025: Pikachu <-- currently selected Pokemon
// [End of Pokedex]
//
// If Squirtle (#007) was then added, it would be the second Pokemon in
// the Pokedex, after Pikachu. The currently selected Pokemon would
// remain Pikachu.
//
// [Start of Pokedex]
// #025: Pikachu <-- currently selected Pokemon
// #003: Squirtle
// [End of Pokedex]
//
// If Diglett (#050) was then added, it would be the third Pokemon in the
// Pokedex, after Squirtle.
//
// [Start of Pokedex]
// #025: Pikachu <-- currently selected Pokemon
// #003: Squirtle
// #050: Diglett
// [End of Pokedex]
void add_pokemon(Pokedex pokedex, Pokemon pokemon);
```

You need to put code in the **add_pokemon** function to store the `pokemon` it is given into the `pokedex` it is given.

You don't need to know any details about what the **Pokemon** type is (i.e. the type of the **Pokemon pokemon** argument passed to **add_pokemon**) to implement **add_pokemon** or any other function in pokedex.c.

Pokemon Tip #0: Pokemon IDs

In the Pokemon game franchise, Pokemon IDs start at 1 and go up to 809 (at the time of writing this). The first Pokemon is Bulbasaur, who has a Pokemon ID of 1. This assignment, however, allows any Pokemon ID from 0 upwards (because programmers love counting from 0).

Throughout this spec we'll be making use of Bulbasaur (as well as some of the other official Pokemon) in our examples, but do note that you can make your own Pokemon with any ID from 0 to the maximum number that can be stored in an integer.

Hint: your code should also operate under the assumption that a pokemon_id of 0 is valid.

If you look in pokedex.h you will discover this definition: **typedef struct pokedex *Pokedex**

In other words the type **Pokedex** is the same as **struct pokedex ***.

struct pokedex is defined at the top of pokedex.c.

It already has one field: a pointer to **struct pokenode** which can be used to store a linked list.

Put in code in **add_pokemon** to add the `pokemon` it is given to the linked list whose head is in the `pokedex` it is given.

`Hint`: allocate a **struct pokenode** with `malloc`.

When you run your code you should see this:

```
$ dcc -o pokedex main.c pokedex.c pokemon.c
$ ./pokedex
============================[ Pokédex ]============================
            Welcome to the Pokédex!  How can I help?
===================================================================
Enter command: a 1 Bulbasaur 0.7 6.9 poison grass
Added Bulbasaur to the Pokedex!
```

You'll need to implement 4 more functions to find out if add_pokemon really works:

```
void detail_pokemon(Pokedex pokedex);
Pokemon get_current_pokemon(Pokedex pokedex);
void find_current_pokemon(Pokedex pokedex);
void print_pokemon(Pokedex pokedex);
```

See pokedex.h for information on each function.

detail_pokemon and **print_pokemon** need to call functions defined in pokemon.h to get the information about a Pokemon they need to print.

Note the information **detail_pokemon** and **print_pokemon** print about a Pokemon depends on whether it has been found.

Hint: add a field to **struct pokenode** to indicate if a Pokemon has been found.

Pokemon Tip #1: finding Pokemon.

Your Pokedex is like an encyclopedia that stores information about Pokemon.

But unlike a normal encyclopaedia, some of the information in your Pokedex is hidden until you've discovered a Pokemon of that kind.

Once you have "found" a certain kind of Pokemon, its details will become visible to you in your Pokedex.

You will need to keep track of whether a Pokemon has been found in your Pokedex implementation.

Hint: add a field to **struct pokenode** to indicate if a Pokemon has been found.

Pokemon Tip #2: the currently selected Pokemon.

Your Pokedex allows you to scroll through the Pokemon stored within, allowing you to see the details of a specific Pokemon of your choice. This means that there will always be one Pokemon that is currently selected by the Pokedex.

When you add the first Pokemon to the Pokedex, that Pokemon becomes the currently selected Pokemon. When you add additional Pokemon, the currently selected Pokemon does not change.

In Stage 2, you will implement functions to allow you to scroll back and forth between Pokemon. But for now, you won't need to change the currently selected Pokemon.

Hint: your currently selected Pokemon won't change in Stage 1, and so will always be the first Pokemon added to your Pokedex (for now).

Pokemon Tip #3: adding Pokemon to the Pokedex.

The Pokemon in your Pokedex are stored in the order that they were added (which is not necessarily by order of pokemon_id!).

For example, if you have an empty Pokedex and then add **Pikachu (#025)** to your Pokedex, your Pokedex now contains:

#025: Pikachu

If you next added **Squirtle (#007)**, your Pokedex now contains:

#025: Pikachu -> #007: Squirtle

If you then added **Diglett (#050)**, your Pokedex now contains:

#025: Pikachu -> #007: Squirtle -> #050: Diglett

Hint: every time you add a Pokemon to your Pokedex, add it to the end of your linked list.

When all four functions are working you should see something like this:

```
$ dcc -o pokedex main.c pokedex.c pokemon.c
$ ./pokedex
===========================[ Pokédex ]===========================
            Welcome to the Pokédex!  How can I help?
=================================================================
Enter command: a 1 Bulbasaur 0.7 6.9 poison grass
Enter command: d
Id: 001
Name: *********
Height: --
Weight: --
Type: --
Enter command: p
--> #001: *********
Enter command: f
Enter command: d
Id: 001
Name: Bulbasaur
Height: 0.7m
Weight: 6.9kg
Type: Poison Grass
Enter command: p
--> #001: Bulbasaur
Enter command: a 7 Squirtle 0.5 9 water none
Added Squirtle to the Pokedex!
Enter command: p
--> #001: Bulbasaur
    #007: ********
Enter command: d
Id: 001
Name: Bulbasaur
Height: 0.7m
Weight: 6.9kg
Type: Poison Grass
Enter command:
```

Stage 2: Navigating the List And Destroying the List

For this stage, you are providing the ability to move a cursor through a Pokédex, either stepping through or jumping to a specific pokemon_id. You will also implement the ability to remove a Pokémon from the Pokedex.

Pokemon Tip #4: scrolling through your Pokedex.

You can change the currently selected Pokemon in your Pokedex (see Pokemon Tip #2) to view the details or information about a specific Pokemon.

You can do this by either scrolling to the **next** and **previous** Pokemon in your Pokedex (using the **next_pokemon** and **prev_pokemon** functions).

You can also select a Pokemon of your choice by passing its pokemon_id to the **change_current_pokemon** function.

Using the example from Pokemon Top #3: if your Pokedex contains the following three Pokemon in the following order, with Pikachu currently selected:

#025: Pikachu <--
#007: Squirtle
#050: Diglett

Calling **next_pokemon** would lead to Squirtle becoming the currently selected Pokemon:

#025: Pikachu
#007: Squirtle <--
#050: Diglett

Calling **prev_pokemon** would lead to Pikachu becoming the currently selected Pokemon again:

#025: Pikachu <--
#007: Squirtle
#050: Diglett

And calling **change_current_pokemon** with the pokemon_id 50 would lead to Diglett becoming the currently selected Pokemon:

#025: Pikachu
#007: Squirtle
#050: Diglett <--

You must implement these functions in pokedex.c:

```
void next_pokemon(Pokedex pokedex);
void prev_pokemon(Pokedex pokedex);
void change_current_pokemon(Pokedex pokedex, int pokemon_id);
void remove_pokemon(Pokedex pokedex);
void destroy_pokedex(Pokedex pokedex);
```

Again, see pokedex.h for information on what each function should do, and use **1511 pokedex_reference** to see what the correct behaviour is.

Note that as time goes on, you may need to update `destroy_pokedex` to make sure it's cleaning up all the memory in a `Pokedex` instance. Keep this in mind.

Stage 3: Finding Pokémon

A Pokémon trainer will be venturing out into the wild, meeting and cataloguing Pokémon. These functions will allow you to go exploring and to mark certain Pokémon as "found" in your Pokédex.

For this, you need to complete:

```
void go_exploring(Pokedex pokedex, int seed, int factor, int how_many);
int count_found_pokemon(Pokedex pokedex);
int count_total_pokemon(Pokedex pokedex);
```

Again, see pokedex.h for information on what each function should do, and use **1511 pokedex_reference** to see what the correct behaviour is.

Stage 4: Evolutions

What's this? Your Pokédex code is evolving? To finish this evolution of your Pokédex, implement these functions in `pokedex.c`:

```
void add_pokemon_evolution(Pokedex pokedex, int from_id, int to_id);
void show_evolutions(Pokedex pokedex);
int get_next_evolution(Pokedex pokedex);
```

Again, see pokedex.h for information on what each function should do, and use **1511 pokedex_reference** to see what the correct behaviour is.

Stage 5: Getting Sub-Lists

Set up your Pokédex to do some nice searching!

```
Pokedex get_pokemon_of_type(Pokedex pokedex, pokemon_type type);
Pokedex get_found_pokemon(Pokedex pokedex);
Pokedex search_pokemon(Pokedex pokedex, char *text);
```

Again, see pokedex.h for information on what each function should do, and use **1511 pokedex_reference** to see what the correct behaviour is.

Testing

As you implement functions in **pokedex.c**, you should add tests to test_pokedex.c .

It already has some basic tests.

```
$ dcc -o test_pokedex test_pokedex.c pokedex.c pokemon.c
$ ./test_pokedex
Welcome to the COMP1511 Pokedex Tests!

==================== Starter Tests ====================

>> Running a very basic new_pokedex test
    ... Creating a new Pokedex
    ... Checking that the returned Pokedex is not NULL
Test passed!
```

As usual autotest is available with some simple tests - but your own tests in **test_pokedex** will be more useful.

```
$ 1511 autotest pokedex
```

File Redirection

Every time you run your application, instead of re-typing all the details of each given Pokémon you're adding to the Pokédex, you can store the input for your application in a separate file, and run it using:

```
$ ./pokedex < test1.in > test1.out
```

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

A example *test1.in* might be

```
a 1 Bulbasaur 0.7 6.9 poison grass
p
```

And the output would be

```
--> #1: *********
```

Sample Pokemon

To help you along you can look at a set of 151 add_pokemon commands in sample_pokemon you can use to play around.

Attribution of Work

This is an individual assignment. The work you submit must be your own work and only your work apart from exceptions below. Joint work is not permitted.
You may use small amounts (< 10 lines) of general purpose code (not specific to the assignment) obtained from site such as Stack Overflow or other publically available resources. You should attribute clearly the source of this code in a comment with it.

You are not permitted to request help with the assignment apart from in the course forum, help sessions or from course lecturers or tutors.

Do not provide or show your assignment work to any other person (including by posting it on the forum) apart from the teaching staff of COMP1511.

The work you submit must otherwise be entirely your own work. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted. The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline. Assignment submissions will be examined both automatically and manually for such submissions.

Relevant scholarship authorities will be informed if students holding scholarships are involved in an incident of plagiarism or other misconduct. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted you may be penalized, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.

Note, you will not be penalized if your work is taken without your consent or knowledge.

Submission of Work

You are required to submit intermediate versions of your assignment.
Every time you work on the assignment and make some progress you should copy your work to your CSE account and submit it using the `give` command below.

It is fine if intermediate versions do not compile or otherwise fail submission tests.

Only the final submitted version of your assignment will be marked.

All these intermediate versions of your work will be placed in a git repo and made available to you via a web interface at this URL, replace *z5555555* with your zID:

`https://gitlab.cse.unsw.edu.au/z5555555/19T1-comp1511-ass2_pokedex/commits/master`

This will allow you to retrieve earlier versions of your code if needed.

You submit your work like this:

```
$ give cs1511 ass2_pokedex pokedex.c test_pokedex.c
```

Assessment

This assignment will contribute 13% to your final mark.

80% of the marks for this assignment will be based on the performance of the functions you write in *pokedex.c*.

20% of the marks for assignment 2 will come from hand marking of the readability of the C you have written. These marks will be awarded on the basis of clarity, commenting, elegance and style. In other words, your tutor will assess how easy it is for a human to read and understand your program.

Tutors will also examine the tests you add to **test_pokedex.c** function. Here is an indicative marking scheme.

HD (85-100)	well-explained beautiful C correctly implementing of stages 1-5 plus thorough tests in test_pokedex.c
DN (75+)	Readable C correctly implementing stages 1-3 and some of stage 4 or 5 plus good testing in test_pokedex.c
CR (65+)	Readable C correctly implementing stages 1-2 with some testing in test_pokedex.c
PS (55+)	Reasonably readable code which implements stage 1 successfully
45-50%	Major progress on stage 1 with some things working/almost working
-70%	Knowingly providing your work to anyone and it is subsequently submitted (by anyone).
-70%	Submitting any other person's work. This includes joint work.
0 FL for COMP1511	Paying another person to complete work. Submitting another person's work without their consent.

The lecturer may vary the assessment scheme after inspecting the assignment submissions but it will remain broadly similar to the description above.

Due Date

This assignment is due Tuesday 13 August 11:59:59

If your assignment is submitted after this date, each hour it is late reduces the maximum mark it can achieve by 3%. For example if an assignment worth 74% was submitted 5 hours late, the late submission would have no effect. If the same assignment was submitted 15 hours late it would be awarded 55%, the maximum mark it can achieve at that time.

Change Log

Version 1.0 (2019-04-13 17:00:00)	• Assignment released.
Version 1.1 (2019-04-14 12:00:00)	• Added Pokemon Tips #1 and #2
Version 1.2 (2019-04-14 14:00:00)	• Added Pokemon Tips #3 and #4
Version 1.2.1 (2019-04-14 14:07:00)	• Corrected prototype of get_current_pokemon in spec
Version 1.2.2 (2019-04-14 19:15:00)	• Fix bug in do_count_found wrapper function
Version 1.3 (2019-04-15 21:30:00)	• Added Pokemon Tip #0 (clarification about Pokemon IDs)
Version 1.4 (2019-04-16 16:00:00)	• Added missing kg/m unit to detail_pokemon sample output
Version 1.5 (2019-04-16 17:00:00)	• Fixed example where Bulbasaur was accidentally 6.9m and 0.7kg
Version 1.6 (2019-04-23 12:40:00)	• Assorted minor changes/clarifications to the provided files (see the version numbers + changelog at the top of each file for more details)
Version 1.7 (2019-04-24 14:40:00)	• More autotests added.
Version 1.9 (2019-04-26 15:40:00)	• More autotests added for level 1 & 2. 'g' command to call get_current_pokemon added to main.c help
Version 1.10 (2019-04-28 07:40:00)	• Autotest levels labelled and reordered to make it clear which level they implement

Credits

Originally created by Sim Mautner. Updated by Zain Afzal & Andrew Bennett.