Pokedex Report

**Repl.it:** <https://repl.it/@danieltangYSJ/danieltangAssessmentProg03#main.c>

# Describe the **struct**s you have created - Pokedex, PokemonNode and PlayerNode. Explain why you have created them the way you have

For Pokdex, PolemonNode and PlayerNode I have used typedef because it simplified use type definition status in C programming to declare structure variable, and alongside I do not have to write struct all over the place, to save time and keystrokes it makes it easier. For PokemonNode I also use binary tree structure, unlike Arrays, the tree is a hierarchical structure, it is useful to store information and extracting information (searching for node). Alongside I have chosen tree structure because there is no limit on the number of nodes that are linked. This is useful because I could add how many pokemon in the pokedex. In a binary search tree, you can in order traverse which produces sorted output.

Like Java class without the methods or constructor, it can allocate on the stack or the heap, which is useful for defining new structured types of data.

# Explain what type of lists you have used for the Pokémon and Player lists. Explain why you have created then the way you have and how they work.

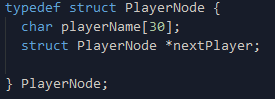
The list I have used for the Pokémon and Player list is simple linked list, for Pokemon I have use the tree link to store the node in a hierarchical setting and use the linked list to find and display the nodes, unlike arrays where memory is stored at random in Linked List it stored in order. It is a simple data structure, which forms a link. With its dynamic size alongside with it ease of insertion and deletion of nodes which give more reason to use linked list unlike using Arrays, however the draw back from it, it is the random access is not allow, and cannot do binary search with linked list, alongside I can traverse though the list with linked list which can display nodes just like an array but quicker.

For the struct I have use char, which is one size which range from -128 to 127 in A character.

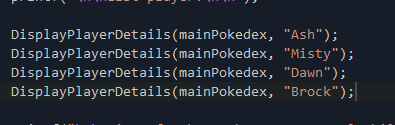
However, I have used linked list because it is set of dynamically allocated nodes which is arrange in a way that each node contains a single value and a pointer, the pointer always point to the next node on the list. If the pointer is null the it is the end of the list.

# Explain where in memory the Pokedex variable and the Pokémon and Player lists are. Explain what this means.

# Explain how *DisplayPlayerDetails(pokedex, name)* works. You should explain the functionality and also which types of memory are used. You should discuss what happens on the Stack and the Heap.

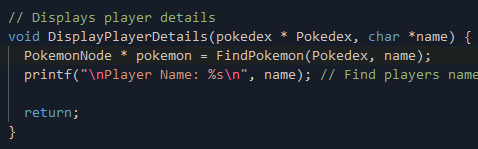
To display player details which includes the Pokémon name, I first I must create a typedef struct which avoids the need of qualify with struct or enum. The naming conventions allow for the function and type by same name. Also, it saves time to use typedef struct [name] {. 

This image below shows what players are in the mainPokedex, which is used to display the player details.



The void is to return the function that we are calling, example we are calling the void and calling the function that is named DisplayPlayerDetails which return the function of (pokedex \* Pokedex, char \* name) {. This basically process what information that is specifically instructed, basically finding all the nodes in the linked list in the Pokedex.

The variables can be accessed globally which is used to find the playerName and the pokemonName.



malloc(sizeof \*)

malloc() can be used to allocate memory from the heap. Which is the Heap, the malloc give use memory address alongside gives us how much we need in memory as in a node. Malloc will manage the heap, allocating bytes. sizeof will return number bytes of the structure require.

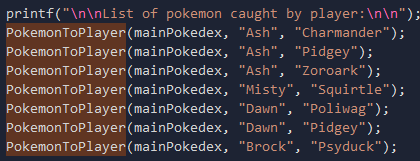


Malloc allocates a block of memory of a given size, which returns a pointer to the first byte of that memory, which malloc return it as NULL, if memory could not be allocated. The sizeof function calculate the size of memory is needed. NULL is 0x00000000, attempt to deference NULL will cause a segment fault. Unlike the stack, variables can be created on the heap which is accessible by any functions anywhere in the program, the heap variables are essentially global scope.

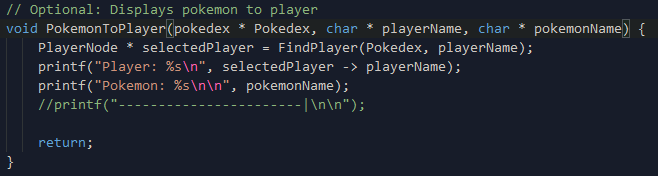
The heap is a large pool of unallocated/ unused memory that is used for dynamic allocation of data. Which malloc allocates sections of data in the heap.

# [Bonus] Bonus marks are available for explaining how any optional bundles you’ve implemented work and how they use the memory.

Player with Pokemon bundle, I have assigned a void called PokemonToPlayer which allow to display the player name and the pokemon that the player caught. Example Ashe caught Charmander node.

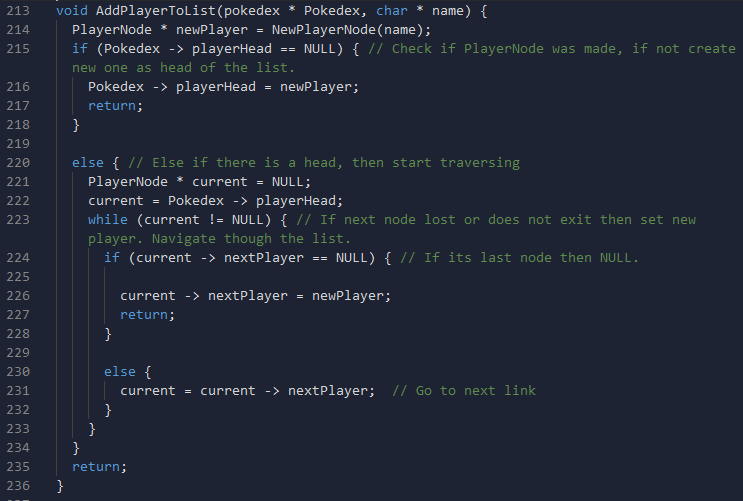


The void calls the function from pokedex and finds playersName and pokemonName. The PlayerNode \* selectedPlayer = FindPlayer(Pokedex, playerName); refers to finding the select player which is FindPlayer in the node which contains Pokdex in the playerName, to my understanding this will enable to print out the selectedPlayer as playerName, alongside the pokemonName.



This will be similar for abilities and type for the pokemon display.

The void returns the function from the pointer pokedex. The if statement check is the PlayerNode is made, if not then creates a new head on the list. Then goes though the Pokdex to the playerHead. Then returns it. For the else if statement it will start traversing though the link, starting from the PlayerNode.



The while (current != NULL) checks if the next node is lost or does not exist then set a new player, and navigate though the list. Then if it is the last node then set that last node to NULL. Then else traverse to the next node the finally return the function.