Problem Set 6

Part 2: Function pointers, hash table

Problem 6.1

In this problem, we will use and create function that utilize function pointers. The file 'callback_ps.c' contains an array of records consisting of a fictitious class of celebrities. Each record consists of the first name, last name and age of the student. Write code to do the following:

- Sort the records based on first name. To achieve this, you will be using the `qsort()` function provided by the standart library: `void qsort(void *arr, int num, int size, int (*fp)(void *pa, void *pb))`. The function takes a pointer to the start of the array 'arr', the number of elements 'num' and size of each element. In addition it takes a function pointer 'fp' that takes two arguments. The function fp is used to compare two elements within the array. Similar to `strcmp()`, it is required to return a negative quantity, zero or a positive quantity depending on whether element pointed to by 'pa' is "less" than, equal to or "greater" the element pointed to by 'pb'. You are required to write the appropriate *callback* function.
- Now sort the records based on last name. Write the appropriate *callback* function.
- The function `void apply(...)` iterates through the elements of the array calling a function for each element of the array. Write a function `isolder()` that prints the record if the age of the student is greater than 20 and does nothing otherwise.

Answer:

The code is in **callback_ps.c**.

Problem 6.2

A useful data structure for doing lookups is a hash table. In this problem, you will be implementing a hash table with chaining to store the frequency of words in a file. The hash table is implemented as an array of linked lists. The hash function specifies the index of the linked list to follow for a given word. The word can be found by following this linked list. Optionally, the word can be appended to this linked list. You will require the code file 'hash.c' and data file 'book.txt' for this problem. You are required to do the following:

- The function `lookup()` returns a pointer to the record having the required string. If not found it returns NULL or optionally creates a new record at the correct location. Please complete the rest of the code.
- Complete the function `cleartable()` to reclaim memory. Make sure each call to `malloc()` is matched with a `free()`.

Answer:

The code is in **hash_ps.c**. When I dealt with function `lookup()`, I ignored the fact if `create` is 1, I have to treate the new `wordrec` as a node and add it to the linkedlist.

This problem gives me a better understanding the purposes and principles of hash table. Basically, hash table is used to map the keys of key-value pairs to some positions in the stored array through some hash functions. The reason we do this is to reduce the lookup time from *O(n)* to *O(1)*. When we look up a word from the record, we don't need to compare the word with each word in the list, all we do is making use of hash function to calculate the position in the stored array for that word. If it collides with the value stored before, we can use linkedlist to store the value in the same index.