Assignment 5 – Towers Report Template

Jayden Sangha

CSE 13S – Winter 24

Purpose

Audience for this section: Pretend that you are working in industry, and write this paragraph for your boss. You are answering the basic question, "What does this thing do?". This section can be short. A single paragraph is okay.

Do not just copy the assignment PDF to complete this section, use your own words.

The purpose of this program is to read a file and output the number of unique lines in that file.

Questions

Please answer the following questions before you start coding. They will help guide you through the assignment. To make the grader's life easier, please do not remove the questions, and simply put your answers below the text of each question.

- In Part I, you implemented garbage collection—routines that clean up dynamically-allocated memory. How did you make sure the memory was all cleaned up? How did you check?

  I made sure the memory was cleaned up by creating a reference count. The program increases the reference count linked to a dynamically created block each time a pointer to that block is assigned or duplicated. It decrements the reference count when a pointer is deliberately released or goes out of scope. Blocks can be securely deallocated when the reference count drops to zero, which indicates that no pointers are referring to them. I checked that all my memory was cleaned up by making a test that manipulates pointers, allocates memory in different ways, and verifies that the memory is properly reclaimed.

- In Part II, you made a major optimization to the linked list optimization. What was it, and why do you think it changed the performance of bench1 so much?

  I added a variable that keeps track of the end of the list, so when adding an item to the list the program does not have to increment through all of the nodes, it simply adds the new node where the variable is and pushes the variable back.

- In Part III, you implemented hash tables. What happens to the performance of bench2 as you vary the number of buckets in your hash table? How many buckets did you ultimately choose to use?

  As you increase the number of buckets the less elements you have in each, so it improves performance and speed. I choose to use 200 buckets because it makes my program efficient without using too much memory overhead.

- How did you make sure that your code was free from bugs? What did you test, and how did you test it? In particular, how did you create inputs and check the output of unique?

I used valgrind to check for data leaks in my program. I also tested that my memories will be properly cleared. I also created a test that insure that the program knows what to do when it is given an invalid file name or type.

Testing

List what you will do to test your code. Make sure this is comprehensive. [1] Be sure to test inputs with delays and a wide range of files/characters.

I will create a test that tests for memory leaks. Another one that checks that the memory gets cleaned up. I will create a test that checks for a valid file.

How to Use the Program

Audience: Write this section for the user of your program. You are answering the basic question, "How do I use this thing?". Don't copy the assignment exactly; explain this in your own words. This section will be longer for a more complicated program and shorter for a less complicated program. You should show how to compile and run your program. You should also describe any optional flags or inputs that your program uses, and what they do.

To use this program you must first compile it with

```
make uniqq
```

then you runt the program using

```
./uniqq < file_name.txt
```

the program will then output the number of unique lines in the file.

To show "code font" text within a paragraph, you can use `\lstinline{}`, which will look like this: `text`.

For a code block, use `\begin{lstlisting}` and `\end{lstlisting}`, which will look like this:

```
Here is some code in lstlisting.
```

And if you want a box around the code text, then use `\begin{lstlisting}[frame=single]` and `\end{ lstlisting}`

which will look like this:

```
Here is some framed code (lstlisting) text.
```

Want to make a footnote? Here's how.[2]

Do you need to cite a reference? You do that by putting the reference in the file `bibtex.bib`, and then you cite your reference like this[1][2][3].

Program Design

Audience: Write this section for someone who will maintain your program. In industry you maintain your own programs, and so your audience could be future you! List the main data structures and the main algorithms. You are answering the basic question, "How is this thing organized so that I can have a chance of fixing it?". This section will be longer for a more complicated program and shorter for a less complicated program.

Pseudocode

Give the reader a top down description of your code! How will you break it down? What features will your code have? How will you implement each function.

Here is my code for ll.c I implemented a tail in the list add function to improve efficiency, I also created list remove and list destroy.

[1]This question is a whole lot more vague than it has been the last few assignments. Continue to answer it with the same level of detail and thought.

[2]This is my footnote.

```
function list_create:
    l = allocate_memory_for_LL

    // Check if memory allocation was successful
    if l is NULL:
        return NULL

    // Initialize the head and tail pointers to NULL
    l.head = NULL
    l.tail = NULL

    // Return the created linked list
    return l

function list_add(l, i):
    n = allocate_memory_for_Node

    // Check if memory allocation was successful
    if n is NULL:
        return false

    n.data = i
    n.next = NULL

    // If the list is empty, set both head and tail to the new node
    if l.head is NULL:
        l.head = n
        l.tail = n
    // Otherwise, add the new node to the end of the list
    else:
        l.tail.next = n


    l.tail = n

    // Return true to indicate successful addition
    return true

function list_find(l, cmp, i):
    // Start from the head of the list
    n = l.head

    // Traverse the list until the end is reached
    while n != NULL
        // If the comparison function returns true for the current node's data and the given item
        if cmp(n.data, i):
            // Return the data of the current node
            return n.data

        // Move to the next node in the list
        n = n.next

    // Return NULL if the item is not found in the list
    return NULL

function list_remove(l, cmp, i):

    active = l.head
    last = NULL

    // Traverse the list until the end is reached
    while (active != NULL)
        // If the comparison function returns true for the current node's data and the given item
        if cmp(active.data, i):
            // If the node to remove is not the head of the list, update the previous node's next
                pointer
            if last is not NULL:
                last.next = active.next
            // If the node to remove is the head of the list, update the head pointer
            else:
                l.head = active.next

            // If the node to remove is also the tail of the list, update the tail pointer
            if active == l.tail:
                l.tail = last

            // Free the memory allocated for the removed node
            free_memory(active)

            // Exit the loop after removing the node
            break

        // Move to the next node in the list
        last = active
        active = active.next

function list_destroy(l):

    if l is NULL:
        return

    active = l.head
    // Initialize a variable to hold the next node to be freed
    next = NULL

    // Traverse the list until the end is reached
        // Store the next node in the list
        next = active.next
        // Free the memory allocated for the current node
        free_memory(active)
        // Move to the next node in the list
        active = next

    // Free the memory allocated for the linked list structure itself
    free_memory(l)
```

Here is my item.c with my new cmp implementation.

```
function cmp(a, b):
    // Compare the keys of the two items using strcmp function
    // Return true if the keys are equal, otherwise return false
    return strcmp(a.key, b.key) == 0
```

Here is my hash.c

```
Function hash_create:
    Allocate memory for an array of Hashtable structures called htable, with size buckets.
    If htable is NULL:
        Return NULL.
    For each index i from 0 to buckets - 1:
        Set htable[i].key to NULL.
        Set htable[i].next to NULL.
    Return htable.

Function hash_put:
    If htable or key is NULL:
        Return false.
    Calculate the hash value for the key.
    Determine the bucket index using the hash value.
    Allocate memory for a new Hashtable structure called n_node.
    If n_node is NULL:
        Return false.
    Allocate memory for the key string inside n_node.
    If the key allocation fails:
        Free n_node.
        Return false.
    Copy the key string into n_node.
    Set n_node.val to val.
    Set active_node to reference the head of the bucket at the calculated index.
    Traverse the linked list until the last node.
    Set the next pointer of the last node to n_node.
    Set n_node.next to NULL.
    Return true.

Function hash_destroy:
    If htable is NULL:
        Return.
    For each index i from 0 to buckets - 1:
        Set active_node to reference the first node in the linked list of the bucket at index i.
        While active_node is not NULL:
            Set holder to active_node.
            Move active_node to the next node.
            Free the key memory inside holder.
            Free holder.
    Free the memory allocated for htable.
    Set htable to NULL.

Function hash_get:
    If htable or key is NULL:
        Return NULL.
    Calculate the hash value for the key.
    Determine the bucket index using the hash value.
    Set active_node to reference the first node in the linked list of the bucket at the calculated
     index.
    While active_node is not NULL:
        If the key in active_node matches the input key:
            Return a reference to the value stored in active_node.
        Move active_node to the next node.
    Return NULL.
```

Here is my uniqq.c psudo code.

```
int main()
    Declare variables:
        buf: Character array of size BUFFER_SIZE.
        table: Pointer to Hashtable structure.
        unique: Size_t variable initialized to 0.
        len: Size_t variable initialized to 0.
```

```
Allocate memory for the hash table using hash_create function and assign the pointer to table.
Initialize unique to 0.

Loop while not at the end of file (EOF) and no error in input:
   Read a line from standard input into buf using fgets.
   Get the length of the string stored in buf and store it in len.

   If the length is greater than 0 and the last character is a newline:
      Replace the newline character with null terminator.
      Decrement len by 1.

   If the string in buf is not found in the hash table (unique entry):
      Insert the string into the hash table with value 0 using hash_put.
      Increment unique.

Print the value of unique.
Destroy the hash table using hash_destroy.

Return 0.
```

Function Descriptions

For each function in your program, you will need to explain your thought process. This means doing the following

- The inputs of every function (even if it's not a parameter)
- The outputs of every function (even if it's not the return value)
- The purpose of each function, a brief description about a sentence long.
- For more complicated functions, include pseudocode that describes how the function works
- For more complicated functions, also include a description of your decision making process; why you chose to use any data structures or control flows that you did.

Do not simply use your code to describe this. This section should be readable to a person with little to no code knowledge. **DO NOT JUST PUT THE FUNCTION SIGNATURES HERE. MORE EXPLANATION IS REQUIRED.**

hash_create (void) This function allocates memory in the heap for the hash table.

hash_put (Hashtable *, char *key, int val) This function will take a key value and integer value, then it stores both in the hash table.

hash_get (Hashtable *, char *key) This function takes the hash table and the pointer to a key value and return the integer value of the key.

hash_destory(Hashtable *) This function frees all the allocated memory in the hash table.

list_remove(*item) This function will take a pointer and access that element in the list and remove it.

list_destroy (LL *) This function will take a pointer to another pointer to a linked list and sets the pointer to NULL. It then frees the memory that was stored in the heap earlier in the code.

References

[1] Wikipedia contributors. C (programming language) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/C_(programming_language), 2023. [Online; accessed 20-April-2023].

[2] Robert Mecklenburg. *Managing Projects with GNU Make, 3rd ed.* O'Reilly, Cambridge, Mass., 2005.

[3] Walter R. Tschinkel. Just scoring points. *The Chronicle of Higher Education*, 53(32):B13, 2007.