GIORGIO CIACCHELLA | 2481024C | LAB 1

Systems Programming

Assessed Exercise 1 Report

Note on Design

Several possible designs have been identified for the program. This version follows a "functional" design approach: most operations on the BST are performed recursively, and rely on a single but modular post-order traversal function; the only exception is the insertion operation, which is powered by iterative search. The traversal function operates with several parameters, most notably a second function providing the logic, and the required arguments; this post-order traversal-based approach was chosen due to its unique suitability for deallocation operations, and enabled by the irrelevance of internal ordering within the structures.

State of Implementation

The program is compiled and linked using `clang` with the `-Wall` and `-Werror` flags.
It compiles and links without any errors or warnings.

The program executes without any runtime errors.

The program is also capable of producing the desired output -- however, the final percentage of TLD matches for any given TLD is printed a number of times equal to all occurrences of said TLD in the input log file. Still, aside from the increased time complexity, this output is fundamentally equivalent to the specified requirement from an application standpoint, as it is trivial to pipe the output through a basic `uniq` call. This effectively makes this discrepancy equivalent to unsorted ordering of the output.

`date.c`

`date.c` works entirely in accordance with the specification.

Additionally, verbose `printf` statements are provided throughout the functions for simple debugging.

Functions of `date.h`

`date_create`

The function `Date* date_create(char* datestr)`:

- Performs rudimentary but effective validation on the input string;
- Prevents memory leaks by:
  - Only dynamically allocating memory if the input string passes validation;
  - Only proceeding in case of successful allocation;
- Parses the input string using `strtok` and sets all fields of the `Date` object.

`date_duplicate`

The function `Date* date_duplicate(Date* d)` operates full duplication by:

- Dynamically allocating memory and only proceeding in case of successful allocation;
- Copying over the values of all fields of the `Date` object.

`date_compare`

The function `int date_compare(Date* date1, Date* date2)` operates optimised "waterfall" comparison between two dates:

- Firstly comparing the years;
- Secondly comparing the months, only in case of equal years;
- Lastly comparing the days, only in case of equal years and months.

`date_destroy`

The function `void date_destroy(Date* d)` deallocates the chunk of memory occupied by a `Date` object.

The responsibility for appropriately and timely calling this function falls on the application. However, this is done extensively in `tldmonitor.c` and requires calls to `date_duplicate` within the BST construction, which avoids the risk of memory leaks.

`tldlist.c`

Functions of `tldlist.h`

`tldlist.c` works mostly in accordance with the specification.

Additionally, verbose `printf` statements are provided throughout the functions for simple debugging.

`tldlist_create`

The function `TLDList* tldlist_create(Date* begin, Date* end)` allocates and initialises a `TLDList` object:

- Prevents memory leaks by:
  - Only proceeding in case of successful dynamic memory allocation;
  - Only allocating a single "header" for the entire structure
- Sets all fields of the `TLDList` object with the appropriate contents of the `Date` argument.

`tldlist_destroy`

The function `void tldlist_destroy(TLDList* tld)` operates post-order traversal on the BST, deallocating the chunk of memory occupied by each node visited. This is accomplished by relying on the framework function `tldlist_traverse` to traverse the tree, and on its plugin helper function `destroy` to deallocate the nodes after visiting.

This specific operation fixed the traversal to be operated post-order, as pointers from deallocated nodes would have been referenced otherwise.

`tldlist_add`

The function `int tldlist_add(TLDList* tld, char* hostname, Date* d)`:

- Checks that the date of the new node falls between the time window considered by the `TLDList` object, relying on `date_compare`;
- Prevents memory leaks by:
  - Reserving a statically allocated chunk of memory for its string field;
  - Only dynamically allocating memory if the input date passes validation;
  - Only proceeding in case of successful dynamic memory allocation;
- Relies on *iterative search* to insert the newly-created node at the appropriate location.

`tldlist_count`

The function `long tldlist_count(TLDList* tld)` operates post-order traversal on the BST, increasing a counter at each node visited. This is accomplished by relying on the framework function `tldlist_traverse` to traverse the tree, and on its plugin helper function `count` to increase the counter.

`tldlist_iter_create`

The function `TLDIterator* tldlist_iter_create(TLDList* tld)` allocates and initialises a `TLDIterator` object:

- Prevents memory leaks by:
  - Only proceeding in case of successful dynamic memory allocation;
- Sets all fields of the `TLDIterator` object with the appropriate contents.

`tldlist_iter_next`

The function `TLDNode* tldlist_iter_next(TLDIterator* iter)` operates post-order traversal on the BST, increasing a counter at each node visited and returning the currently visited node when the counter reaches a limit. This is accomplished by relying on the framework function `tldlist_traverse` to traverse the tree, and on its plugin helper function `next` to increase the counter, compare it with the limit, and break out of recursion.

This specific operation fixed the traversal function to be of `int` type to enable breaking out of recursion.

`tldlist_iter_destroy`

The function `void tldlist_iter_destroy(TLDIterator *iter)` deallocates the chunk of memory occupied by a `TLDIterator` object.

The absence of helper data structures in the "functional" design removes the need for further iterative or recursive deallocation calls.

`tldnode_tldname`

The function `char* tldnode_tldname(TLDNode* node)` simply retrieves the domain field associated with a `TLDNode` object.

`tldnode_count`

The function `long tldnode_count(TLDNode* node)` operates post-order traversal on the BST, increasing a counter at each node visited if its value matches a condition. This is accomplished by relying on the framework function `tldlist_traverse` to traverse the tree, and on its plugin helper function `next` to compare the node value with the condition and increase the counter.

Additional Functions (internal use within `tldlist.c`)

`tldlist_traverse`

```
int tldlist_traverse (TLDNode* root_node, int (*func)(TLDNode*, TLDNode**, void*, void*), TLDNode** arg_node, void* arg1, void* arg2)
```

A number of helper functions have been designed to be used as plugins:

- `destroy` for `tldlist_destroy`;
- `count` for `tldlist_count`;
- `next` for `tldlist_iter_next`;
- `count_if` for `tldnode_count`.