nfl testplan

WO1 Clayton E. Williams

September 2023

1 Purpose

nfl loads a player/team database of NFL players from 1960 to 2019. Through the use of command line options, nfl will display statistics and metrics of specified teams or players. nfl relies on custom functionality, as well as previously written libraries for interacting with hashtables, linked lists, and conducting a BFS
The purpose of this test plan is to provide a process of testing the functions of auxiliary libraries to gain reasonable assurance that the program exhibits desired behavior and does not crash on unexpected or invalid input or arguments.

2 Components

nfl contains both automated and manual testing

2.1 Automated Tests - Test Suites

Automation of unit testing is provided through the make tool and can be run from the command line as:

```
$ make check
```

There are four test suites to test maze

- test_hashtable
- test_io_helper
- test_llist
- test_trie

2.2 Test Cases

- test_hashtable
  These test cases test the previously written hashtable library against NULL or invalid entries.

- test_hash_table_create_valid
  This tests that creation of a hashtable with valid hash function and size ¿ 0 returns non-NULL pointer.
- test_hash_table_create_invalid
  This tests that creation of a hashtable with NULL hash function or size ¡ 1 return NULL.
- test_hash_table_insert
  Given a valid hashtable, this test asserts the return value after calling hash_table_insert() with each arg being NULL returns 0, and calling with valid, non-NULL values returns 1.
- test_find
  Given a valid hashtable with populated values, this test asserts find() returns NULL when passed NULL arguments, and that a non-NULL value is returned when tested called with valid hashtable and valid key.

• test_io_helper

- test_validate_file_invalid
  This tests validate_file against an array on non-regular and size 0 files and asserts return value 0.
- test_validate_file_valid
  This tests against known, valid files, and asserts return value 1.
- test_get_num_entries_valid
  This tests get_num_entries() against three files with known number of lines, and asserts that the return value equals known value.
- test_get_num_entries_invalid
  This tests get_num_entries() against NULL file pointer and asserts return value 0.

• test_llist

- test_llist_create
  This tests that llist_create() returns a non-NULL pointer.
- test_llist_enqueu_valid
  With a valid linked list created and valid, non-NULL pointer to data, asserts llist_enqueue() returns 1.
- test_llist_enqueue_invalid
  This tests llist_enqueue() against a NULL linked list and NULL pointer to data and asserts the return value is 0.
- test_llist_dequeue_valid
  With a valid, populated linked list, llist_dequeue() is called and asserts the returned values equal known values of the linked list.

  - test_llist_dequeue_invalid
    This test asserts calling llist_dequeue() on NULL or empty linked list return NULL.
  - test_llist_is_empty
    This test creates a NULL linked list pointer, and asserts that llist_is_empty() returns 0. It is then created as an empty list, asserting return value is 1. linked list is populated with 10 elements and asserts the return value is 0. linked list is then dequeued 10 times and asserts the return value is then 1.
  - test_llist_create_iter
    This tests the llist_create_iter() function, asserting 0 is returned when called with NULL arguments.

- test_trie
  - test_trie_create
    This test asserts that trie_create() returns a non-NULL pointer.
  - test_trie_insert
    trie_insert() is called with NULL arguments and asserts that the return value is 0.

Results of successful test run with make check:

```
100%: Checks: 14, Failures: 0, Errors: 0
test/test_hashtable.c:20:P:hashtable:*curr++:0: Passed
test/test_hashtable.c:26:P:hashtable:*curr++:0: Passed
test/test_hashtable.c:38:P:hashtable:*curr++:0: Passed
test/test_hashtable.c:51:P:hashtable:*curr++:0: Passed
test/test_llist.c:21:P:llist:*curr++:0: Passed
test/test_llist.c:29:P:llist:*curr++:0: Passed
test/test_llist.c:37:P:llist:*curr++:0: Passed
test/test_llist.c:45:P:llist:*curr++:0: Passed
test/test_llist.c:59:P:llist:*curr++:0: Passed
test/test_llist.c:73:P:llist:*curr++:0: Passed
test/test_llist.c:85:P:llist:*curr++:0: Passed
test/test_llist.c:89:P:llist:*curr++:0: Passed
test/test_trie.c:9:P:trie:*curr++:0: Passed
test/test_trie.c:18:P:trie:*curr++:0: Passed
```

2.3 Manual Tests - Valgrind

Manual testing of the program is to ensure there are no memory leaks or errors reported by valgrind: NOTE: Output will differ based on options and/or file provided

```
$ make clean && make debug
```

```
$ valgrind ./nfl --roster "Dallas Cowboys" 1970
==30037== HEAP SUMMARY:
==30037==     in use at exit: 0 bytes in 0 blocks
==30037==   total heap usage: 386,266 allocs, 386,266 frees,
13,271,638 bytes allocated
==30037==
==30037== All heap blocks were freed -- no leaks are possible
==30037==
==30037== For lists of detected and suppressed errors, rerun with: -s
==30037== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

2.4 Manual Tests - Stress Test

Manual stress testing of the program is provided to test against a variety of valid and invalid arguments, options, and files. The controller for these tests are located at ./test/stress_test.txt. To run the stress test:

```
$ make clean
$ make
$ while read -r line; do eval $line; done < test/stress_test.txt
```

NOTE: This will run valgrind against multiple options, including –oracle, and may take significant time (greater than 1 hour) to complete.