radix testplan
WO1 Clayton E. Williams
August 2023

Purpose

radix is a custom library that provides an API and defines various functions to build and interact with a radix trie
The purpose of this test plan is to provide a process of testing library functions to gain reasonable assurance that the functions exhibit desired behavior and do not crash on unexpected or invalid input.

Components

Components of `radix` test coverage are contained in automated tests through the use of the `make` command, and manual tests using `valgrind`.

Automated Tests - Test Suite

There is only one test suite to test `radix`: `test_radix.c`
Automation of unit testing is provided through the make tool and can be run from the command line:

```
$ make check
```

Test Cases

Each public function of `radix.c` will be tested against valid and invalid data. There are 10 total test cases, however most contain multiple assertions to test a wide range of input. Valid input is assumed to be a non-NULL, non-empty string of ASCII characters >= *a* and <= *z* The functions to be tested by `test_radix.c` are:

1. radix_create()
   The test case for this function is `test_radix_create`. It creates a trie and asserts that the pointer itself is not **NULL**, the children array is not **NULL** and the boolean flag is initialized to *false*.
2. radix_insert_word()
   The test cases for this function are `test_radix_insert_valid` and `test_radix_insert_invalid`. `test_radix_insert_valid` inserts valid strings and asserts the return value is **non-0**. `test_radix_insert_invalid` inserts invalid strings and asserts the retun value is **0**. This test case also tests the return value of **0** when a string is inserted that already exists in the trie.
3. radix_remove_word()
   The test cases for this function are `test_radix_remove_word_valid` and `test_radix_remove_word_invalid`. `test_radix_remove_word_valid` calls `radix_remove_word()` against an array of words known to be in the trie and asserts the return value is **1**. `test_radix_remove_word_invalid` calls `radix_remove_word()` against an array of

words known not to be in the trie and asserts the return value is **0**. It also tests against invalid arguments that raise an error, asserting the return value is **-1**.

4. radix_find_word()
   The test cases for this function are `test_radix_find_word_valid` and `test_radix_find_word_invalid`. `test_radix_find_word_valid` calls `radix_find_word()` against an array of words known to be in the trie and asserts the return value is '1'. `test_radix_find_word_invalid` calls `radix_find_word()` against an array of words known not to be in the trie and asserts the return value is **0**. It also tests against invalid arguments that raise an error, asserting the return value is **-1**.
5. radix_find_prefix()
   The test cases for this function are `test_radix_find_prefix_valid` and `test_radix_find_prefix_invalid`. `test_radix_find_prefix_valid` calls `radix_find_prefix()` against an array of strings known to be in the trie and asserts the return value is '1'. These prefixes can be full or partial strings, but must be valid input strings, and start at the top level of the trie. `test_radix_find_prefix_invalid` calls `radix_find_prefix()` against an array of strings known not to exist in the trie, or whose root is a sub-tree of the root of the trie and asserts the return value is '0'. It also tests against invalid arguments that raise an error, asserting the return value is **-1**.
6. radix_delete()
   The test case for this function is `test_radix_delete_valid`. This test case calls `radix_delete()` on the address of the trie and iterates over it, asserting all children are **NULL**, and that the trie pointer itself is **NULL**. It then calls `radix_delete()` on the previously deleted and freed pointer to ensure there is no **SEGFAULT** from dereferencing a null pointer.

Setup/Teardown

To run automated tests, the check library must first be installed.
Each test case runs its own `setup()` and `teardown()` functions to run automatically. The `test_radix.c` library contains defined arguments to pass to test cases for their respective asserts:

```
const char *valid_words[] = {
    "places",
    "pickling",
    "placebo",
    "play",
    "picture",
    "pickets",
    "panacea",
    "pick",
    "picket",
    "pickles"
};

const char *invalid_args[] = {
    NULL,
    "",
    " ",
    "Pickle",
```

```
    "!ickle",
    "pickl!"
};

const char *invalid_words[] = {
    "p",
    "a",
    "laces",
    "anacea",
    "pic",
    "pickl",
    "pla",
    "placeb",
    "pi"
};

const char *valid_prefixes[] = {
    "p",
    "pan",
    "pi",
    "pick",
    "pl",
    "pla",
    "placeb",
    "picke",
    "pickl",
    "pickle",
    "picklin"
};

const char *invalid_prefixes[] = {
    "ic",
    "pand",
    "pice",
    "ply",
    "placb",
    "lay",
    "anacea",
    "plce"
};
```

Manual Testing - Valgrind

Manual testing of the program is to ensure there are no memory leaks or errors reported by valgrind.
Running valgrind against radix

```
$ make clean && make debug
$ valgrind ./radix
```

Results in the following report:

```
==161574==
==161574== HEAP SUMMARY:
==161574==     in use at exit: 0 bytes in 0 blocks
==161574==   total heap usage: 77 allocs, 77 frees, 15,768 bytes allocated
==161574==
==161574== All heap blocks were freed -- no leaks are possible
==161574==
==161574== For lists of detected and suppressed errors, rerun with: -s
==161574== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```