radix design plan
WO1 Clayton E. Williams
August 2023

Project structure

Project structure break down:

```
.
├── doc
│   ├── design.pdf
│   ├── radix_description.pdf
│   ├── radix.h.3
│   ├── testplan.pdf
│   └── writeup.pdf
├── include
│   └── radix.h
├── Makefile
├── README.md
├── src
│   ├── driver.c
│   └── radix.c
└── test
    ├── p_dict.txt
    ├── p_test.txt
    └── test_radix.c
```

`doc` - documentation for the project. Includes design plan, man page, writeup, and test plan
`inlcude` - Custom header file for radix library.
`src` - C source code files for driver and library.
`test` - C source code directore containing test suite and .txt files for testing.

Data Structures

The main data structure needed is a struct for the trie, that contains an array of trie structs, character array for string stored at index, and flag to indicate whether or not it is a word:

```c
struct trie_t {
    struct trie_t *children[NUM_CHARS];
    char *word;
    bool b_is_word;
};
```

Functions Needed

Public Functions

The following functions will be made public by the API:

```
/*
 Callocs and returns a pointer to the trie_t struct. Calloc is used so
members
 are initialized to 0.
 */
trie_t *radix_create(void);

/*
 Takes a string and inserts it, if possible, into the trie.
 */
int radix_insert_word(trie_t *trie, const char *word);

/*
 Removes the string 'word' from the trie if it is found.
 */
int radix_remove_word(trie_t *trie, const char *word);

/*
 Searches trie for target and returns 1 if it is found.
 */
int radix_find_word(trie_t *trie, const char *target);

/*
 Searches trie for prefix and returns 1 if it is found. Prefix must begin
at the top-level of the trie to be valid.
 */
int radix_find_prefix(trie_t *trie, const char *prefix);

/*
 free() the memory allocated to the trie and sets the address to NULL.
 */
void radix_delete(trie_t **trie);
```

Static Functions

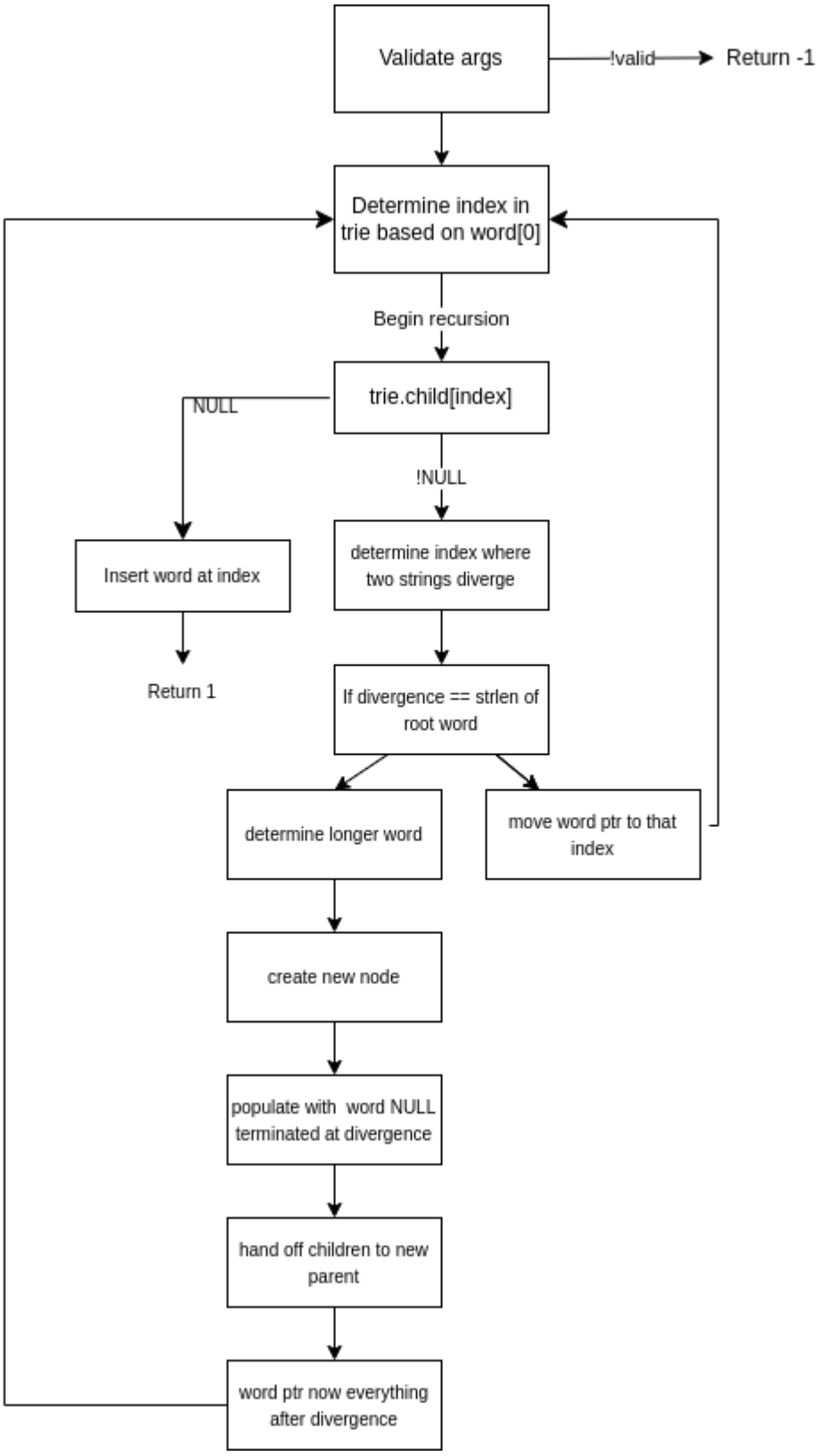
The following functions will not be made public by the API, and are defined as static helper functions:

```
/*
 This helper function is used to validate a word argument passed to public
functions. It checks that the string is all ASCII lower case ('a' - 'z').
It returns true on success, else false.
 */
static bool validate_input(const char *word);

/*
 This helper function is a recursive function to locate a word, if it
exists, in the trie. It has an additional argument, 'b_to_remove' which is
a boolean flag to specify whether this function is being called from
```

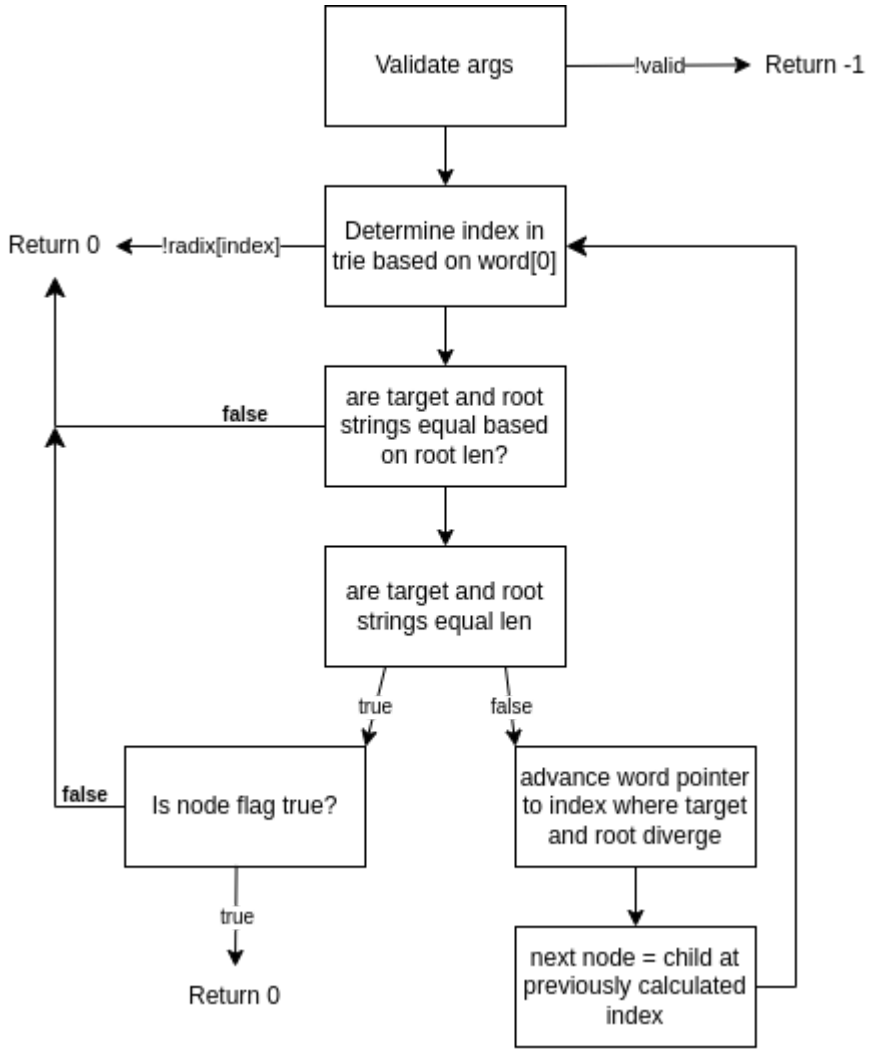
```
radix_insert_word or radix_find_word to return a '0' failure, or from
radix_remove_word to set the node flag to false.
 */
static int radix_find_rec(trie_t * root, const char *word, bool
b_to_remove);

/*
 This helper function is a recursive function to insert a new word into the
trie.
 */
static trie_t *radix_insert_rec(trie_t * node, char *word, int len, int
index);

/*
 This helper function is called from radix_insert_rec to create a new node,
assign a string, and boolean flag for a new word.
 */
static trie_t *radix_create_node(int len);

/*
 This helper function is used to get the index position where two
divergence occurs between two strings being compared. This is used when
bifurcation is needed to know where to NULL terminate one string and where
to the pointer should be moved on the other.
 */
static int get_prefix_index(const char *word, const char *new_word);

/*
 This helper function is called by radix_find_prefix to return a pointer to
the node where a prefix terminates. That node is passed as an argument to
print_word_by_prefix for its operations.
 */
static trie_t *get_prefix_node(trie_t * node, const char *word);

/*
 This helper function is a recursive function called by radix_find_prefix
that will iterate over the trie and print any full word based on prefix
passed into radix_find_prefix.
 */
static void print_word_by_prefix(trie_t * node, char *word, int len);
```

Project Flow

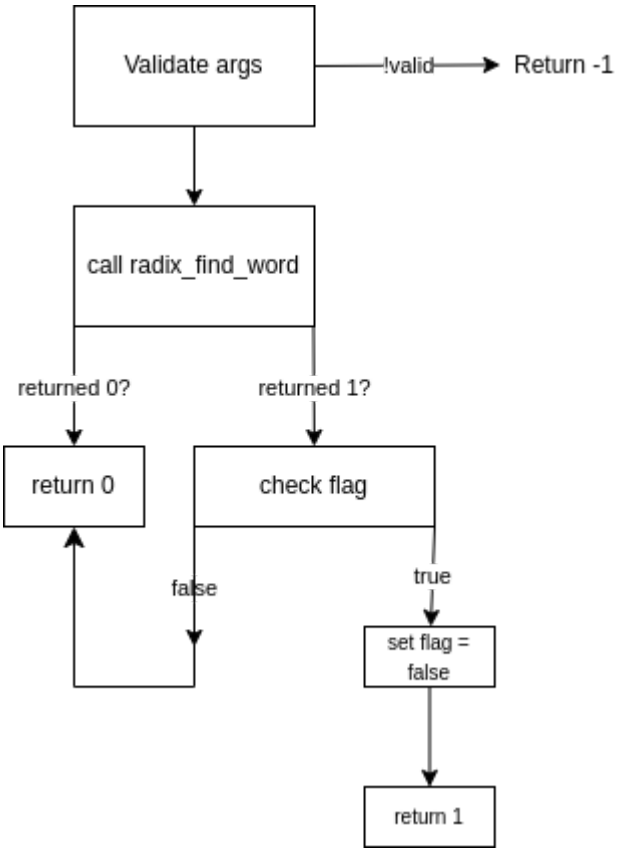
Insert Word


Validate args
!valid
Return -1
Determine index in trie based on word[0]
Begin recursion
trie.child[index]
NULL
!NULL
Insert word at index
Return 1
determine index where two strings diverge
If divergence == strlen of root word
determine longer word
move word ptr to that index
create new node
populate with word NULL terminated at divergence
hand off children to new parent
word ptr now everything after divergence

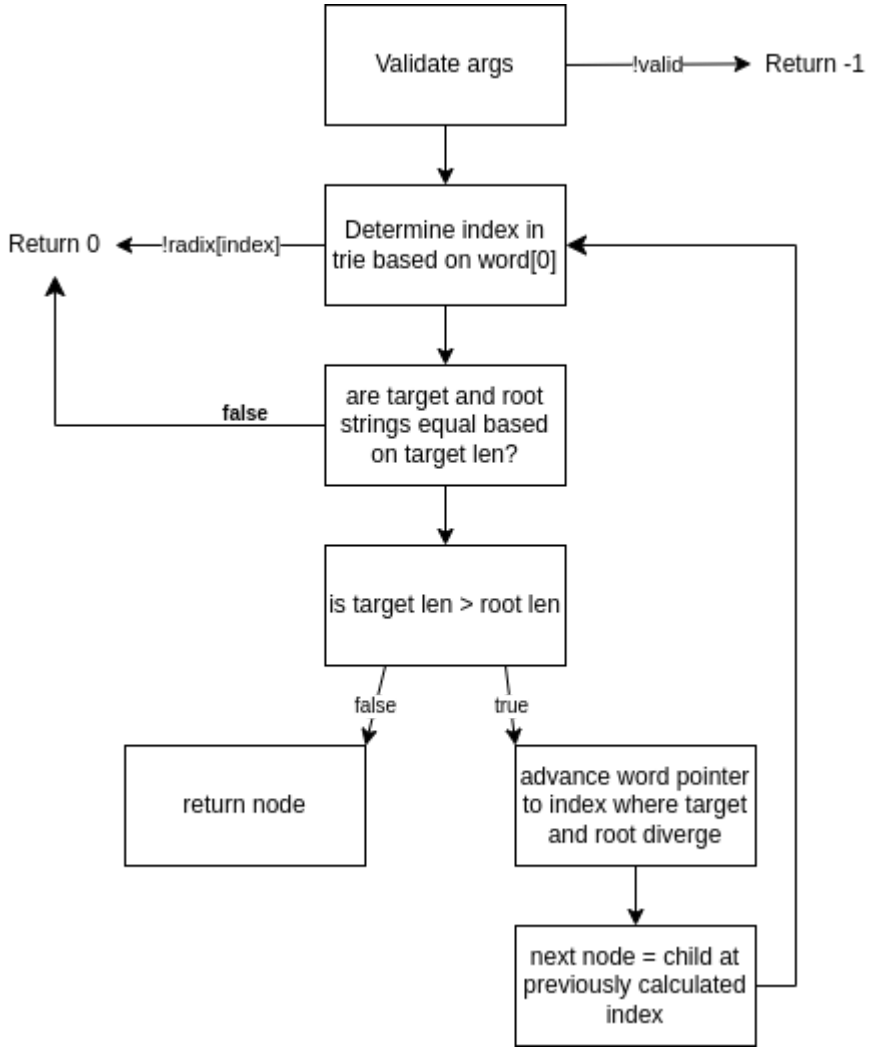
Find Word


Validate args
!valid
Return -1
Determine index in trie based on word[0]
Return 0
!radix[index]
are target and root strings equal based on root len?
false
are target and root strings equal len
true
false
Is node flag true?
false
true
Return 0
advance word pointer to index where target and root diverge
next node = child at previously calculated index

Remove Word


Validate args
!valid
Return -1
call radix_find_word
returned 0?
returned 1?
return 0
check flag
false
true
set flag = false
return 1

Find Prefix


Validate args
!valid
Return -1
Determine index in trie based on word[0]
Return 0
!radix[index]
are target and root strings equal based on target len?
false
is target len > root len
false
true
return node
advance word pointer to index where target and root diverge
next node = child at previously calculated index