

Jonathan Chiu (A12113428), Adrian Cordova y Quiroz (A12010305)

CSE 100, Alvarado

Winter 2017

2/22/17

Refactor DictionaryTrie.cpp

1. Got rid of magic numbers in the code

- It improves readability of the code and it's easier to maintain
- These will never change, but they make the meaning clearer, and provide some protection against typos
- Here is one example of the magic number refactoring which makes the code more legible and easier to understand

Original Code

```
bool DictionaryTrie::find(std::string word) const
{
    // Check for invalid (empty) strings
    if (word.length() == 0) { return false; }

    // Current position in the trie
    MWNode * current = root;
    int next;

    // Traverse down valid paths within the trie
    for (unsigned int level = 0; level < word.length(); level++) {
        if (!current) { return false; }

        // Move down a level
        next = word[level] - 'a';

        // Next char is space
        if (word[level] == ' ') {
            next = SPACE;
        }

        // Invalid char
        if (next < 0 || next > 26) {
            return false;
        }
    }
}
```

Refactored Code

```
bool DictionaryTrie::find(std::string word) const
{
    int zero = 0;
    int twenty-six = 26;
    // Check for invalid (empty) strings
    if (word.length() == zero) { return false; }

    // Current position in the trie
    MWNode * current = root;
    int next;

    // Traverse down valid paths within the trie
    for (unsigned int level = zero, level < word.length(); level++) {
        if (!current) { return false; }

        // Move down a level
        next = word[level] - 'a';

        // Next char is space
        if (word[level] == ' ') {
            next = SPACE;
        }

        // Invalid char
        if (next < zero || next > twenty-six) {
            return false;
        }
    }
}
```

2. Made variable names easier to understand

- More descriptive variable names make it easier to read
- The less generic they are the less confusing it will be

Original Code

```
// current position in the trie
MWTNode * current = root;
int next;

// Traverse down valid paths within the trie
for (unsigned int level = 0; level < word.length(); level++) {
    if (!current) { return false; }

    // Move down a level
    next = word[level] - 'a';
```

Refactored Code

```
// Current position in the trie
MWTNode * currentNode = root;
int nextPosition;

// Traverse down valid paths within the trie
for (unsigned int level = zero; level < word.length(); level++) {
    if (!currentNode) { return false; }

    // Move down a level
    nextPosition = word[level] - 'a';
```