/// Andrew Souza

/// Comp 210 -- Spring 2024

/// Assignment 1 -- Problem 1

#include <iostream>

using namespace std;

// Programming Project 11.14.1

// Maps each index of the char array to a letter on the keypad

// For simplicity's sake we will treat 0 and 1 as their numerical values

char keyPad[11][5] = {"0","1","ABC","DEF","GHI","JKL","MNO","PQRS","TUV","WXYZ"};

// Builds out the array based on the number of potential letter combinations

int GetSizeOfArray(int code, int positions) {

int size = 1;

for (int i = 0; i < positions; i++) {

if ((code % 10) == 7 || (code % 10) == 9) {

size \*= 4; // 7 and 9 add four letter possibilities

} else if ((code % 10) != 1) {

size \*= 3; // All other numbers add three, besides 1,

// which only adds 1

}

code /= 10; // As code is divided by 10 the number in the

// following place is shifted up in significance

}

return size;

}

// Recursively traverses the code and assigns each possible value to it's space in the array

void CreateArrayOfStrings(char\*\* wordList, int code, int size, int positions) {

// Index of each letter in each sub-word of the keyPad array

int letterPos = 0;

// If positions is equal to 0 then the cycle completes itself

if (positions == 0) {

return;

}

for (int i = 0; i < size; i++) {

// For each subword we insert a letter from the keyPad array

// at the respective positions index point

wordList[i][positions - 1] = keyPad[(code % 10)][letterPos];

// letterPos increments each time i increments positions number of times

if ((i - positions) % positions == 0) {

letterPos++;

}

if (code % 10 == 7 || code % 10 == 9) {

if (letterPos > 3) { // 7 and 9 have 4 potential letters

letterPos = 0; // thus we reset letterPos after

} // each 4 iterations

} else if (code % 10 != 1 && code % 10 != 0) {

if (letterPos > 2) { // All other letters besides 1 and 0

letterPos = 0; // have 3 potential letters and so

} // we reset after 3 iterations

} else {

letterPos = 0;

}

}

// We call CreateArrayOfStrings with code divided by 10 and positions

// minus 1

CreateArrayOfStrings(wordList, code / 10, size, positions - 1);

}

int main() {

int code = -1;

cout << "Enter your PIN: ";

cin >> code;

// Counts if code goes to 1, 10, 100, 1000...

int positions = 0;

int temp = code;

// Positions increments while temp is divided by 10 until it reaches 0

while (temp != 0) {

positions++;

temp /= 10;

}

// Count of total letter combinations

int size = GetSizeOfArray(code, positions);

// Char array of size number \* size positions

char\*\* arr = new char\*[size + 1];

for (int i = 0; i < size; i++) {

arr[i] = new char[positions + 1];

arr[i][positions] = '\0';

}

cout << "You Entered: " << code << endl;

cout << "Possible Codes: " << endl;

CreateArrayOfStrings(arr, code, size, positions);

for (int i = 0; i < size; i++) {

for (int j = 0; j < positions; j++) {

cout << arr[i][j];

}

cout << endl;

}

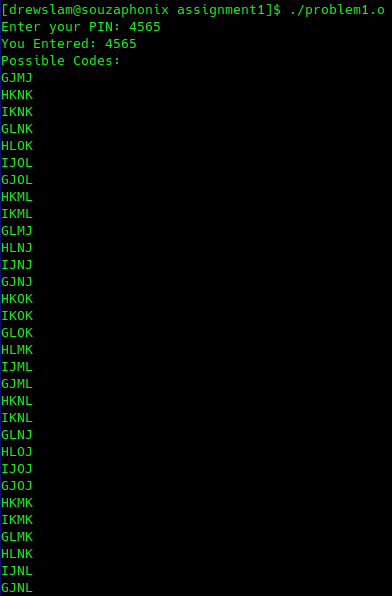
for (int i = 0; i < size; i++) {

delete arr[i];

}

return 0;

}



/// Andrew Souza

/// Comp 210 -- Spring 2024

/// Assignment 1 -- Problem 2

#include <iostream>

using namespace std;

// Write a version of the binary search algorithm that can be used to search a list of strings.

// Make sure the list is sorted before using the search.

// Write a driver to test your code.

string StringSearch(string\* dictionary, int low, int high, string searchTerm) {

// Ints low and high represent the low and high index point of the dictionary array

if (low > high) {

return "Not Found!";

}

// Detects the middle point of the array

int mid = (low + high) / 2;

// searchTerm is compared to dictionary at the middle index

// If searchTerm is alphabetically greater than we call StringSearch again

// using mid + 1 as the low parameter.

if (dictionary[mid] < searchTerm) {

return StringSearch(dictionary, mid + 1, high, searchTerm);

} else if (dictionary[mid] > searchTerm) {

// If searchTerm is lesser than we call StringSearch using

// mid - 1 as the high parameter.

return StringSearch(dictionary, low, mid - 1, searchTerm);

}

// If the mid index point holds the value of searchTerm

return dictionary[mid];

}

int main() {

const int size = 10;

string dictionary[size] = {"Apple", "Banana", "Centipede", "Dagger", "Elephant", "Fountain", "Georgia", "Hamlet", "Icarus", "Monorail"};

string searchTerm;

cout << "Search for a word: ";

cin >> searchTerm;

cout << "Result: " << StringSearch(dictionary, 0, size, searchTerm) << endl;

return 0;

}

