gist (https://gist.github.com/)

Jalanorian / gist:947f992ed6181545dd5c

Created just now

gistfile1.cpp C++

```cpp
#include <iostream>
#include <time.h>
#include <algorithm>
#include <math.h>

using namespace std;


// Prototyping functions
string Disemvowel(string);
int RecursiveSum(int[], int, int);
int LinearSearch(int, int[], int);
int BinarySearch(int[], int, int, int);

int main() {

        // Constants for array lengths
        const int kSumArrayLength = 10;
        const int kSearchArrayLength = 20;
        const int kSortedArrayLength = 15;

        // Creates string and sends it to Disemvowel function
        string s = "Hello, World!";
        string result_s = Disemvowel(s);


        // Creates array with random numbers and sends it to RecursiveSum function
        int int_array [kSumArrayLength];
        srand(time(NULL));
        for (int i = 0; i < kSumArrayLength; i++) {
                int_array[i] = (rand() % 100);
        }
        int recursive_sum = RecursiveSum(int_array, 0, 9);


        // Creates arry with random numbers and sends it to LinearSearch function
        int search_array [kSearchArrayLength];
        for (int i = 0; i < kSearchArrayLength; i++) {
                search_array[i] = (rand() % 10);
        }
        int index_linear = LinearSearch(5, search_array, kSearchArrayLength);

        // Creates arry with random numbers, sorts the array, and sends it to BinarySearch function
        int binary_array [kSortedArrayLength];
        for (int i = 0; i < kSortedArrayLength; i++) {
                binary_array[i] = (rand() % 20);

        }
        sort (binary_array, binary_array + kSortedArrayLength);
        int index_binary = BinarySearch(binary_array, 13, 0, kSortedArrayLength);

        return 0;
}


string Disemvowel(string s) {
        // Initializes output string
        string buff;
        char vowel_array [10] = {'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'};
        bool is_vowel;

        for (int i = 0; i < s.length(); i++) {
                is_vowel = false;
                for (int j = 0; j < 10; j++){
                        if (s[i] == vowel_array[j]){
                                is_vowel = true;
                        }
                }

                if (!is_vowel){
                        buff += s[i];
                }
        }

        return buff;
}

int RecursiveSum(int int_array[], int start, int end) {
        // BASE-CASE
        if (start == end) {
                return int_array[start];
        }
        // RECURSIVE-CASE
        return int_array[start] + RecursiveSum(int_array, start + 1, end);
}

int LinearSearch(int target, int search_array[], int array_length) {
        // Initializes as "not found"
        int index = -1;
        for (int i = 0; i < array_length; i++) {
                if (search_array[i] == target) {
                        // Store each variable
                        index = i;
                        // Break out if found
                        break;
                }
        }
        return index;
}

int BinarySearch(int binary_array[], int target, int left, int right) {

        int mid = floor((left + right)/ 2);

        // BASE-CASE #1: Found
        if (binary_array[mid] == target) {
                return (mid);
        }
        // BASE-CASE #2: Not found and at end of list
        else if (right - left < 2)
        {
                if (binary_array[mid + 1] == target) {
                        return (mid + 1);
                } else {
                        return -1;
                }

        } // RECURSIVE-CASE
        else if (binary_array[mid] > target) {
                return BinarySearch(binary_array, target, left, mid - 1);
        } // RECURSIVE-CASE
        else {
                return BinarySearch(binary_array, target, mid + 1, right);
        }
}
```