@mainpage

# Лабораторна робота № 10

### 1.1 Розробник

@author Іовов Микита ( КН-923Б)

@date 10.05.2024

@version 1.8.17

### 1.2 Загальне завдання

Дано масив з N цілих чисел. Визначити, чи є в масиві елементи, що повторюються;

якщо такі є, то створити масив, в якому вказати, скільки разів які елементи повторюються.

Таким чином, в результуючому масиві кожен непарний елемент - число, що повторюються;

кожен парний елемент - кількість повторювань.

## 2 Опис програми

### 2.1 Функціональне призначення

Програма визначає кількість повторувань чисел у масиві.

### 2.2 Обмеження на застосування

Програма може бути обмежена обробкою певних типів даних або масивів конкретних розмірів.

### 2.3 Опис логічної структури

* main.c: Основний файл програми, де ініціалізується функція “append” з вказаними аргументами для додавання елементів в масив:

#include "lib.h"  
  
  
  
  
  
  
  
#define N 10  
  
  
  
  
  
  
  
int main()  
  
  
  
{  
  
  
  
  
  
  
  
 int arr[N] = {1, 2, 3, 4, 5, 6, 7, 8, 10, 10}; //Ініцілізація масиву  
  
  
  
  
  
  
  
 int \*result = malloc(4);  
  
  
  
  
  
  
  
 int \*result\_size = malloc(4);   
  
  
  
  
  
  
  
 //Цикл для перевірки кожного елемента масиву  
  
  
  
 for (int i = 0; i < N; i++)  
  
  
  
 {  
  
  
  
  
  
  
  
 int count = 0;  
  
  
  
  
  
  
  
 for (int el = 0; el < N; el++)  
  
  
  
 {  
  
  
  
  
  
  
  
 if (\*(arr + i) == \*(arr + el))  
  
  
  
 {  
  
  
  
  
  
  
  
 count++;  
  
  
  
 }  
  
  
  
 }  
  
  
  
  
  
  
  
 if (count > 1)  
  
  
  
 {  
  
  
  
  
  
  
  
 result = append(result, result\_size, \*(arr + i));  
  
  
  
  
  
  
  
 result = append(result, result\_size, count);  
  
  
  
 }  
  
  
  
 }  
  
  
  
  
  
  
  
 free(result);  
  
  
  
 free(result\_size);  
  
  
  
  
  
  
  
 return 0;  
  
  
  
}

* lib.с: Файл з функцією для додавання елементів до масиву:

#include "lib.h"  
  
  
  
  
  
  
  
int \* append(int \*arr, int \*size, int element)  
  
  
  
{  
  
  
  
  
  
  
  
 (\*size)++;   
  
  
  
  
  
  
  
 arr = (int \*)realloc(arr, (size\_t)(\*size) \* sizeof(int));  
  
  
  
  
  
  
  
 \*(arr + \*size - 1) = element; //Додаємо новий елемент до масиву   
  
  
  
  
  
  
  
 return arr;  
  
  
  
}

* lib.h: Заголовочний файл, що містить прототипи функцій для роботи з динамічним масивом:

#ifndef LIB\_H  
  
  
  
  
  
  
  
#define LIB\_H  
  
  
  
  
  
  
  
#include <stdio.h>  
  
  
  
  
  
  
  
#include <stdlib.h>  
  
  
  
  
  
  
  
  
  
int \*append(int \*arr, int \*size, int element);  
  
  
  
  
  
  
  
#endif

* test.c: Містить тестовий запуск функцій для завідомо відомих даних:

#include "../src/lib.h"  
  
  
  
#include <assert.h>  
  
  
  
#include <check.h>  
  
  
  
  
START\_TEST(test\_append\_basic) {  
  
  
  
 int initial\_size = 0;  
  
  
  
 int \*arr = NULL;  
  
  
  
  
  
  
  
 // Перевіряємо додавання одного елемента   
  
  
  
 int element1 = 10;  
  
  
  
 arr = append(arr, &initial\_size, element1);  
  
  
  
 ck\_assert\_int\_eq(initial\_size, 1);  
  
  
  
 ck\_assert\_int\_eq(arr[0], 10);  
  
  
  
  
  
  
  
 // Перевіряємо додавання одного елемента   
  
  
  
 int element2 = 20;  
  
  
  
 arr = append(arr, &initial\_size, element2);  
  
  
  
 ck\_assert\_int\_eq(initial\_size, 2);  
  
  
  
 ck\_assert\_int\_eq(arr[1], 20);  
  
  
  
  
  
  
  
 // Перевіряємо додавання декiлькох елементів   
  
  
  
 int element3 = 30;  
  
  
  
 int element4 = 40;  
  
  
  
 arr = append(arr, &initial\_size, element3);  
  
  
  
 arr = append(arr, &initial\_size, element4);  
  
  
  
 ck\_assert\_int\_eq(initial\_size, 4);  
  
  
  
 ck\_assert\_int\_eq(arr[2], 30);  
  
  
  
 ck\_assert\_int\_eq(arr[3], 40);  
  
  
  
  
  
  
  
   
  
  
  
 free(arr);  
  
  
  
}  
  
  
  
  
  
  
  
Suite \*append\_suite(void) {  
  
  
  
 Suite \*s;  
  
  
  
 TCase \*tc\_core;  
  
  
  
  
  
  
  
 s = suite\_create("Append");  
  
  
  
  
  
  
  
 tc\_core = tcase\_create("Core");  
  
  
  
  
  
  
  
 tcase\_add\_test(tc\_core, test\_append\_basic);  
  
  
  
  
  
  
  
 suite\_add\_tcase(s, tc\_core);  
  
  
  
  
  
  
  
 return s;  
  
  
  
}  
  
  
  
  
  
  
  
int main(void) {  
  
  
  
 int number\_failed;  
  
  
  
 Suite \*s;  
  
  
  
 SRunner \*sr;  
  
  
  
  
  
  
  
 s = append\_suite();  
  
  
  
 sr = srunner\_create(s);  
  
  
  
  
  
  
  
 srunner\_run\_all(sr, CK\_NORMAL);  
  
  
  
 number\_failed = srunner\_ntests\_failed(sr);  
  
  
  
 srunner\_free(sr);  
  
  
  
  
  
  
  
 return (number\_failed == 0) ? EXIT\_SUCCESS : EXIT\_FAILURE;  
  
  
  
}

### 2.2.3 Структура проекту

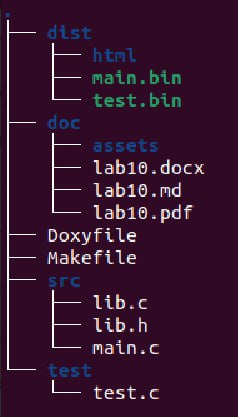


Рисунок 1 - Структура проекту

### 2.4 Важливі фрагменти програми

* Включення заголовочних файлів.
* Створення тестового набору.
* Запуск тестів та вивід результатів.

# 2 Варіанти використання

Для представлення виконання кожного завдання використовується:

* послідовне виконання програми в інструменті lldb;
* виведення результатів у консоль за допомогою функції виводу;
* юніт-тест;

### Варіант використання 1

* Виявлення дублікатів елементів масиву.

Інструкція для запуску програми за допомогою юніт-тесту:

* Виклик юніт тесту за допомогою команди ./test.bin.

Інструкція для запуску програми у режимі відлагодження:

* Виклик програми у відлагоджувачі lldb main.bin;
* Встановлення точки зупинки на строчці 52;
* Вивести масив result за дпомого команди:
  + p \*(result + 0);
  + p \*(result + 1);

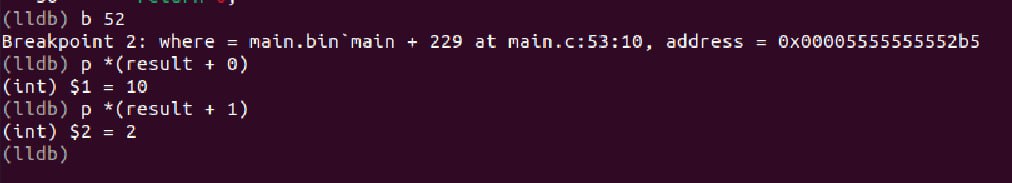


Рисунок 2 - Скріншот виводу програми

### Результат:

Як бачимо, результатом є масив = [10, 2], що є правильною відповідю.

# Висновок

Наданий код виконує функцію підрахунку кількості елементів масиву, які потворюються. Ця програма реалізована мовою програмування C і використовує вбудовані функції для роботи з динамічним виділенням пам’яті. Логіка роботи: програма знаходить елементи, які повторюються, за допомогою двох циклів, а також підраховує кількість повторювань елементів.

Результат роботи записується у масив result.