

@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 повторювань 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}; //Ініціалізація  
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.c: 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 декількох 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

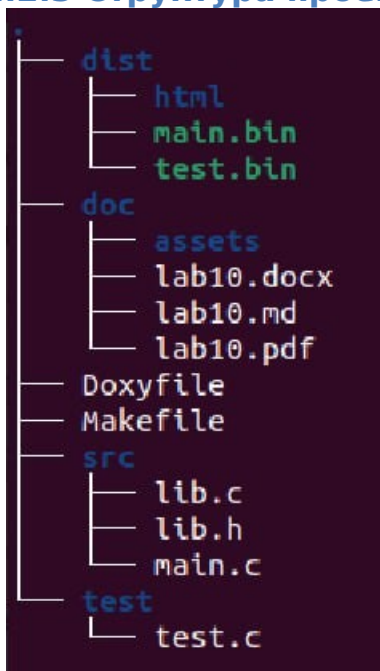


Рисунок 1 - Структура проекту

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

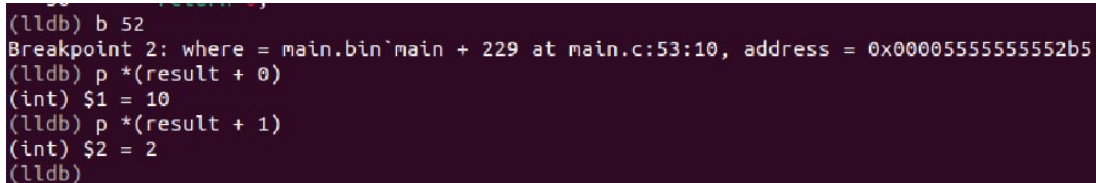
- Виявлення дублікатів елементів масиву.

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 допомогою команди:
 - `p *(result + 0);`
 - `p *(result + 1);`



```
(lldb) b 52
Breakpoint 2: where = main.bin`main + 229 at main.c:53:10, address = 0x00005555555552b5
(lldb) p *(result + 0)
(int) $1 = 10
(lldb) p *(result + 1)
(int) $2 = 2
(lldb)
```

Рисунок 2 - Скріншот виводу програми

Результат:

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

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`.