Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт

Про виконання лабораторної роботи №7

на тему:

«Вказівники на функції. Рекурсивні функції»

з дисципліни «Основи програмування»

Лектор:
ст. викл. каф. ПЗ
Муха Т.О.

Виконав:
ст. гр. ПЗ-11
Ясногородський Н.В.

Прийняв:
асист. каф. ПЗ
Дивак І.В.

« __ » _____ 2021 р.

∑ = _____ _____ .

Львів – 2021

Тема: Вказівники на функції. Рекурсивні функції

Мета: Поглиблене вивчення можливостей функцій в мові С з використанням механізмів рекурсії та вказівників.

ЗАВДАННЯ

Завдання 1.

Використовуючи вищенаведені функції `swap` та `qs_sort`, які реалізують алгоритм швидкого сортування масиву, написати програму мовою С для порівняння ефективності алгоритмів сортування масивів великих обсягів (наприклад, 100000 елементів). Програма повинна також реалізувати один з класичних алгоритмів сортування масиву згідно з варіантом індивідуального завдання. У програмі використати два одинакових масиви, які заповнити випадковими числами, здійснити перевірку впорядкованості елементів масиву, перевірку ідентичності масивів до і після сортування, а також за допомогою стандартної функції `time`, оцінити час виконання реалізованих алгоритмів сортування.

5. Сортування в порядку зростання “бульбашковим” методом без додаткової перевірки чи масив вже відсортований.

Завдання 2.

Написати мовою С три функції, щоб протабулювати, задану згідно варіанту, функцію на проміжку $[a\ ,\ b]$ з кроком h, використавши:

а) для першої функції оператор циклу for;

б) для другої – оператор циклу while;

в) для третьої – – оператор циклу do…while.

Вибір способу табулювання реалізувати через вказівник на відповідну функцію.

$$3.\ f=\cos^2(4x), a=-\pi/2, b=0\ ;$$

Завдання 3.

6. Ввести стрічку (число). Використовуючи рекурсивну функцію, отримати з введених символів все можливі перестановки знаків.

ТЕКСТ ПРОГРАМИ

Завдання 1

```
#include <stdio.h>
#include <time.h>
#include <math.h>

void swap(int array[], long pos1, long pos2)
{
   long tmp = array[pos1];
```

```
    array[pos1] = array[pos2];
    array[pos2] = tmp;
}

void quickSort(int array[], long start, long end)
{
    long head = start, tail = end - 1, tmp;
    long diff = end - start;
    long pe_index;
    // якщо залишилося менше двох елементів – кінець рекурсії
    if (diff < 1)
        return;
    if (diff == 1)
        if (array[start] > array[end])
        {
            swap(array, start, end);
            return;
        }
    // пошук індексу розділяючого елементу pe_index
    long m = (start + end) / 2;
    if (array[start] <= array[m])
    {
        if (array[m] <= array[end])
            pe_index = m;
        else if (array[end] <= array[start])
            pe_index = start;
        else
            pe_index = end;
    }
    else
    {
        if (array[start] >= array[end])
            pe_index = start;
        else if (array[end] >= array[m])
            pe_index = m;
        else
            pe_index = end;
    }
    long pe = array[pe_index]; // сам розділяючий елемент
    swap(array, pe_index, end);
    while (1)
    {
        while (array[head] < pe)
            ++head;
        while (array[tail] > pe && tail > start)
            --tail;
        if (head >= tail)
            break;
```

```
        swap(array, head++, tail--);
    }
    swap(array, head, end);
    long mid = head;
    quickSort(array, start, mid - 1); // рекурсивний виклик для 1-ої підмножини
    quickSort(array, mid + 1, end);   // рекурсивний виклик для 2-ої підмножини
}

void bubbleSort(int array[], int length)
{
    int j, i;
    for (i = 0; i < length - 1; i++)
    {
        for (j = 0; j < length - i - 1; j++)
        {
            if (array[j] > array[j + 1])
            {
                swap(array, j + 1, j);
            }
        }
    }
}

void printArray(int arr[], int size)
{
    printf("\nArray[%d] {", size);

    for (int i = 0; i < size; i++)
    {
        printf("%d", arr[i]);
        if (i >= 100)
        {
            printf(" ...%d more elements}\n", size - i);
            return;
        }
        if (i != size - 1)
        {
            printf(", ");
        }
    }

    printf("}\n");
}

int isArraySortedAsc(int arr[], int length)
{
    for (int i = 0; i < length - 1; i++)
    {
```

```
        if (arr[i] > arr[i + 1])
        {
            return 0;
        }
    }
    return 1;
}

int areArraysEqual(int arr1[], int arr2[], int length)
{
    for (int i = 0; i < length; i++)
    {
        if (arr1[i] != arr2[i])
        {
            return 0;
        }
    }
    return 1;
}

void printExecutionTime(clock_t start, clock_t end)
{
    printf("Time taken: %f sec.\n", (float)(end - start) / CLOCKS_PER_SEC);
}

#define ARRAY_SIZE 200000
int main(void)
{
    printf("Task 5, Section 1:\n");

    int arr1[ARRAY_SIZE], arr2[ARRAY_SIZE];
    clock_t start, end;

    srand(time(NULL));
    for (int i = 0; i < ARRAY_SIZE; i++)
    {
        arr1[i] = rand() % ARRAY_SIZE;
        arr2[i] = arr1[i];
    }

    printf("Initial arrays are %s\n\n", areArraysEqual(arr1, arr2, ARRAY_SIZE) ? "identical" :
"different");

    // bubble sort
    start = clock();
    bubbleSort(arr1, ARRAY_SIZE);
    end = clock();

    printf("Bubble sort %s:\n", isArraySortedAsc(arr1, ARRAY_SIZE) ? "worked" : "failed");
```

```
    printExecutionTime(start, end);
    printArray(arr1, ARRAY_SIZE);

    // quick sort
    start = clock();
    quickSort(arr2, 0, ARRAY_SIZE - 1);
    end = clock();

    printf("\nQuick sort %s:\n", isArraySortedAsc(arr2, ARRAY_SIZE) ? "worked" : "failed");
    printExecutionTime(start, end);
    printArray(arr2, ARRAY_SIZE);

    printf("\nSorted arrays are %s\n", areArraysEqual(arr1, arr2, ARRAY_SIZE) ? "identical" :
"different");

    return 0;
}
```

Завдання 2

```
#include <stdio.h>
#include <math.h>
#define _USE_MATH_DEFINES

typedef struct tabulatedRes
{
    double x;
    double value;
} tabulatedRes;
typedef double doubleMathFunc(double);
typedef void getResultsFunc(double, double, double, doubleMathFunc *, tabulatedRes *);

double mathFunc(double x)
{
    return pow(cos(4 * x), 2);
}
void getResultsWithFor(double start, double end, double step, doubleMathFunc *func,
tabulatedRes *results)
{
    int length = (end - start) / step;

    for (int i = 0; i < length; i++, start += step)
    {
        results[i].x = start;
```

```
            results[i].value = func(start);
        }
    }
    void getResultsWithWhile(double start, double end, double step, doubleMathFunc *func,
tabulatedRes *results)
    {
        while (start < end)
        {
            *results++;
            results->x = start;
            results->value = func(start);
            start += step;
        }
    }
    void getResultsWithDoWhile(double start, double end, double step, doubleMathFunc
*func, tabulatedRes *results)
    {
        do
        {
            *results++;
            results->x = start;
            results->value = func(start);
            start += step;

        } while (start < end);
```

```
}

void printTabulateResults(double start, double end, double step, doubleMathFunc *func,
int option)
{
    int length = (end - start) / step;
    tabulatedRes *results = (tabulatedRes *)malloc(length * sizeof(tabulatedRes));

    getResultsFunc *getResult;

    switch (option)
    {
    case 1:
        getResult = &getResultsWithFor;
        break;
    case 2:
        getResult = &getResultsWithWhile;
        break;
    case 3:
        getResult = &getResultsWithDoWhile;
        break;
    default:
        break;
    }
```

```
    getResultsWithFor(start, end, step, func, results);

    printf("Printing tabulated results:\n");
    for (int i = 0; i < length; i++)
    {
        printf("x=%lf\ty=%lf\n", results[i].x, results[i].value);
    }

    free(results);
}

int main(void)
{
    printf("Task 3, Section 2:\n");

    const double start = -M_PI / 2, end = 0;
    double step;
    int option;
    printf("Enter step: ");
    scanf("%lf", &step);

    printf("Enter option number [1/2/3]: ");
    scanf("%d", &option);

    printTabulateResults(start, end, step, &mathFunc, option);
```

```
    return 0;
}
```

Завдання 3

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void printNumber(int *a, int n);
void swap(int *a, int *b);
void permutation(int *a, int start, int end);
int *numberToArray(int number, int *length);

int main(void)
{
  printf("Task 6, Section 3:\n");

  int number, digitsCount;
  printf("Enter number: ");
  scanf("%d", &number);

  int *digitsArr = numberToArray(number, &digitsCount);
  permutation(digitsArr, 0, digitsCount - 1);
```

```
  return 0;
}

int *numberToArray(int number, int *length)
{
  *length = (int)(log10((float)number)) + 1;
  int *arr = (int *)malloc(*length * sizeof(int)), *curr = arr;

  do
  {
    *curr++ = number % 10;
    number /= 10;
  } while (number != 0);

  return arr;
}

void swap(int *a, int *b)
{
  int temp = *a;
  *a = *b;
  *b = temp;
}
```

```
void permutation(int *a, int start, int end)
{
  if (start == end)
  {
    printNumber(a, end + 1);
    return;
  }

  for (int i = start; i <= end; i++)
  {
    // swapping numbers
    swap((a + i), (a + start));
    // fixing one first digit
    // and calling permutation on
    // the rest of the digits
    permutation(a, start + 1, end);
    swap((a + i), (a + start));
  }
}

void printNumber(int *a, int n)
{
  static int num = 1;
```

```
  printf("%d: ", num++);


  for (int i = 0; i < n; i++)
  {
    printf("%d", a[i]);
  }
  printf("\n");
}
```

РЕЗУЛЬТАТИ

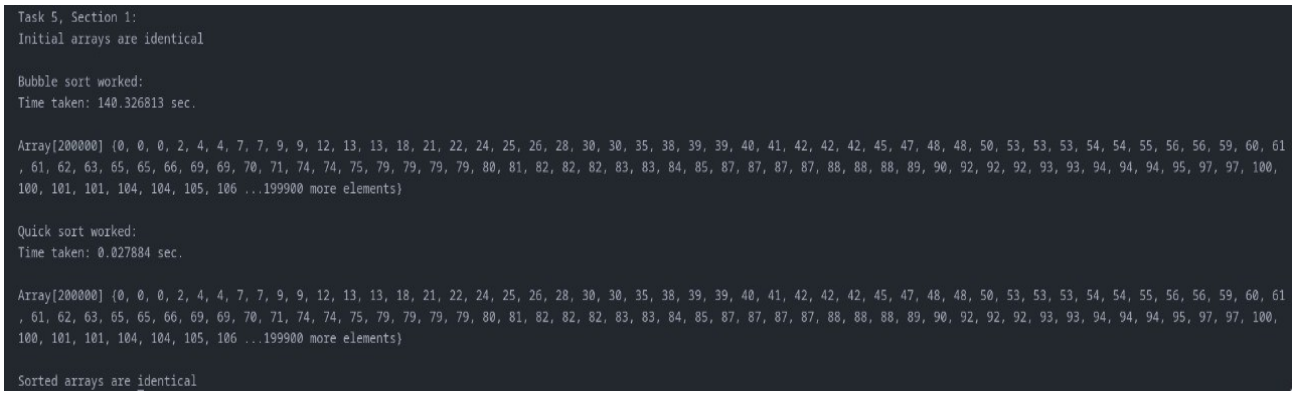

Рис 1. Результат виконання програми №1

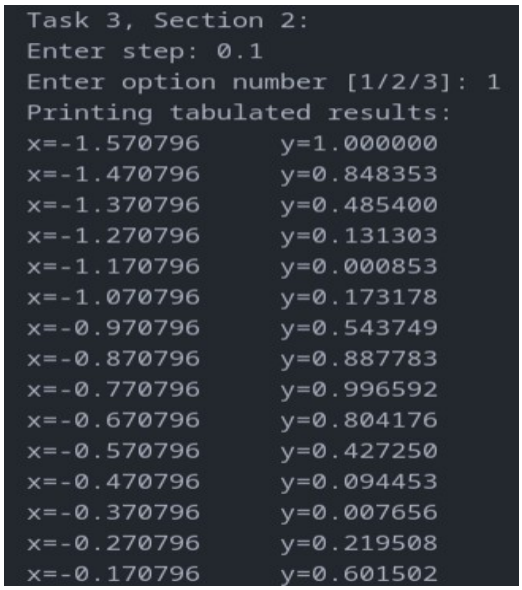

Рис 2. Результат виконання програми №2

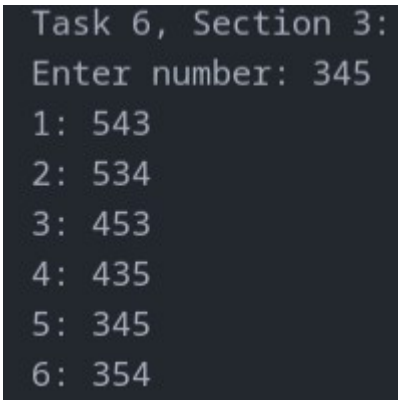

Рис 3. Результат виконання програми №3

ВИСНОВКИ

На даній лабораторній роботі створено програму, що порівнює ефективність алгоритмів сортування масивів великих обсягів.