**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

Інститут **ІКНІ**

Кафедра **ПЗ**

**ЗВІТ**

До 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-15

Марущак А.С.

**Прийняла:**

асист. каф. ПЗ

Заводовська Н.О.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2021 р.

∑= \_\_\_\_ .

Львів – 2021

**Тема роботи:** вказівники на функції. рекурсивні функції.

**Мета роботи:** поглиблене вивчення можливостей функцій в мові С з використанням механізмів рекурсії та вказівників.

**Теоретичні відомості**

Вказівник на функцію

Як і дані, функція зберігається у пам'яті, а отже, вона має адресу. Тому на функцію можна встановити вказівник. Робиться це за допомогою такої конструкції:

тип\_результату(\* ім’я\_вказівника)(список\_параметрів);

Саме розміщення вказівника у круглих дужках і означає, що він посилається на функцію. Без круглих дужок значення конструкції зовсім інше – оголошувалася б функція, що повертає вказівник на дані зазначеного типу.

Вказівники на функцію широко застосовуються в програмуванні:

• багато бібліотечних функцій як аргумент одержують вказівник на функцію;

• використання вказівників на функцію як аргументів дозволяє розробляти універсальні функції, наприклад функції чисельного рішення рівнянь, чисельного інтегрування й диференціювання;

• масиви вказівників на функції використовуються для організації меню.

Як і звичайні змінні, вказівники на функції можуть об’єднуватися в масиви. Наприклад, визначити й проініціалізувати масив вказівників на функції можна в такий спосіб. Нехай є такі функції:

float funcl(float);

float func2(float);

float func3(float);

float func4(float);

float func5(float);

Тоді масив вказівників на функції можна описати так:

float (\*fparray [5])(float) = { funcl, func2, func3, func4, func5};

Доступ до елементів масиву fparray виконується, як до звичайних елементів масиву. Наприклад:

float х = 1;

printf(“%f”, fparray[0](х)); // або printf(“%f”, (\*fparray[0]) (х); Вказівники на функції застосовуються у випадках, коли потрібно викликати різні функції залежно від певних умов.

Рекурсивні функції

В алгоритмічній мові С функції можуть бути рекурсивними. Рекурсивною називається така функція, яка звертається сама до себе (викликає саму себе). У рекурсивній функції обчислювальний процес повинен бути організований так, щоб на кожному кроці рекурсії задача поступово спрощувалася до тих пір поки для неї не стане можливим не рекурсивне рішення. Саме тому рекурсивна функція повинна повертати конкретний результат тільки для найпростішої частини задачі, тобто тоді коли можливо обчислити результат безпосередньо. В усіх інших випадках функція викликає саму себе, але з передачею нового значення аргументу, при якому нова задача є більш простою для розв’язання підзадачею вихідної задачі, причому алгоритм розв’язування цієї нової задачі є аналогічним алгоритму розв’язування вихідної задачі. Створюючи рекурсивну функцію слід дотримуватися таких правил:

1. при кожному виклику такої функції в неї повинні передаватися нові дані;
2. на якомусь етапі повинен бути припинений подальший виклик даної функції;
3. після завершення кожного виклику рекурсивної функції в точку повернення повинен передаватися деякий результат для подальшого використання.

**ЛАБОРАТОРНЕ ЗАВДАННЯ**

1. Ознайомитися з теоретичним матеріалом викладеним вище в даній інструкції і виконати приклади програм.

2. Одержати індивідуальне завдання з Додатку 1.

3. Скласти програму на мові С у відповідності з розробленим алгоритмом.

4. Виконати обчислення по програмі. Навести характеристики продуктивності комп’ютера на якому було виконано програму.

5. Одержати індивідуальне завдання з Додатку 2.

6. Розробити алгоритм розв’язання індивідуального завдання і подати його у вигляді блок-схеми.

7. Скласти програму на мові С у відповідності з розробленим алгоритмом.

8. Виконати обчислення по програмі при різних значеннях точності і порівняти отримані результати.

9. Одержати індивідуальне завдання з Додатку 3.

10. Розробити алгоритм розв’язання індивідуального завдання і подати його у вигляді блок-схеми.

11. Скласти програму на мові С у відповідності з розробленим алгоритмом. 12. Виконати обчислення по програмі при різних значеннях точності і порівняти отримані результати.Підготувати та здати звіт про виконання лабораторної роботи.

**Індивідуальне 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.

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

**Код програми:**

(lab07\_01\_func.h):

#ifndef LAB07\_01

#define LAB07\_01

void swap(int array[], long pos1, long pos2);

void qs\_sort(int array[], long start, long end);

void bubble\_sort(int array[], int size);

int is\_sorted(int array[], int size);

int is\_equal\_arrays(int array1[], int array2[], int size);

#endif

(lab07\_01\_func.c):

#include "lab07\_01\_func.h"

void swap(int array[], long pos1, long pos2)

{

    long tmp;

    tmp=array[pos1];

    array[pos1]=array[pos2];

    array[pos2]=tmp;

}

void qs\_sort(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;

    qs\_sort(array, start, mid-1); // рекурсивний виклик для 1-ої підмножини

    qs\_sort(array, mid+1, end); // рекурсивний виклик для 2-ої підмножини

}

void bubble\_sort(int array[], int size){

    for(int i = 0; i < size; i++){

        for(int j = 1; j < size - i; j++){

            if(array[j] < array[j-1]) swap(array, j, j-1);

        }

    }

}

int is\_sorted(int array[], int size){

    for(int i = 1; i < size; i++){

        if(array[i] < array[i-1]) return 0;

    }

    return 1;

}

int is\_equal\_arrays(int array1[], int array2[], int size){

    for(int i = 0; i < size; i++){

        if(array1[i] != array2[i]) return 0;

    }

    return 1;

}

(lab07\_01.c):

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "lab07\_01\_func.h"

#define SIZE 25000

int main(){

    clock\_t qs\_start\_time = 0, qs\_end\_time = 0, bs\_start\_time = 0, bs\_end\_time = 0;

    srand(time(NULL));

    int arr\_qs[SIZE], arr\_bs[SIZE];

    for(int i = 0; i < SIZE; i++){

        arr\_qs[i] = arr\_bs[i] = rand();

    }

    if(!is\_equal\_arrays(arr\_qs, arr\_bs, SIZE)){

        printf("Error: start arrays aren't equal.\n");

    }

    qs\_start\_time = clock();

    qs\_sort(arr\_qs, 0, SIZE-1);

    qs\_end\_time = clock();

    bs\_start\_time = clock();

    bubble\_sort(arr\_bs, SIZE);

    bs\_end\_time = clock();

    if(!is\_equal\_arrays(arr\_qs, arr\_bs, SIZE)){

        printf("Error: result arrays aren't equal.\n");

    }

    else if(!is\_sorted(arr\_qs, SIZE)){

        printf("Error: arrays aren't sorted.\n");

    }

    else{

        printf("\nSorting was successful.\n");

        printf("Quick sort: %.3lf seconds.\n", (double)(qs\_end\_time - qs\_start\_time)/ CLOCKS\_PER\_SEC);

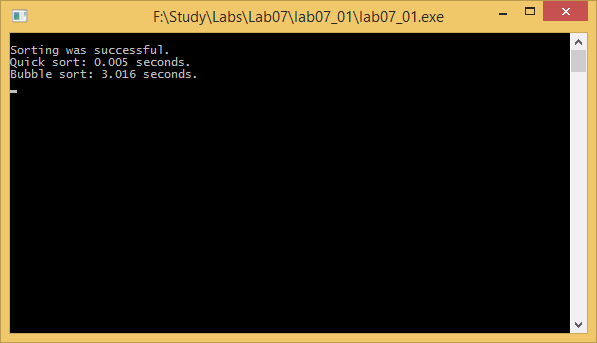
        printf("Bubble sort: %.3lf seconds.\n", (double)(bs\_end\_time - bs\_start\_time)/ CLOCKS\_PER\_SEC);

    }

}

**Протокол роботи:**

Програма сортує 2 однакових масиви різними способами, перевіряє результати, а також проводить виміри часу:



Як бачимо, алгоритм швидкого сортування відпрацьовує значно швидше за бульбашковий варіант. Це пов’язано з різною швидкістю росту к-ті операцій. А також швидке сортування працює швидше через відносно рівномірне розподілення елементів як наслдіок випадкового наповнення масивів.

**Висновок:**

Ми порівняли роботу двох алгоритмів сортування і пересвідчились в зручності використання рекурсії в реалізації функцій. Також це допомогло нам навчитися оцінювати час проведення операцій.

№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.

25. f = ,a=0,b=1;

**Код програми:**

(lab07\_02\_func.h):

#ifndef LAB07\_02

#define LAB07\_02

double f(double x);

void forTab(double a, double b, double h);

void whileTab(double a, double b, double h);

void doWhileTab(double a, double b, double h);

#endif

(lab07\_02\_func.c):

#include <math.h>

#include <stdio.h>

double f(double x){

    if(x - 1 > - 1e-6  || fabs(x) < 1e-6 || x < -1) return NAN;

    return asin(sqrt(x))/sqrt(x \* (1-x));

}

void forTab(double a, double b, double h){

    if(b < a) return;

    printf("\n  [For tabulation]  \n");

    printf("%-8s | %8s\n", "X", "Y");

    printf("---------+---------\n");

    for(double x = a; x < b; x += h){

        double res = f(x);

        if(isnan(res)){

            printf("%-8.3lf | %8s\n", x, "Not def.");

        }

        else

            printf("%-8.3lf | %8.3lf\n", x, res);

    }

}

void whileTab(double a, double b, double h){

    if(b < a) return;

    printf("\n [While tabulation] \n");

    printf("%-8s | %8s\n", "X", "Y");

    printf("---------+---------\n");

    double x = a;

    while(x < b){

        double res = f(x);

        if(isnan(res)){

            printf("%-8.3lf | %8s\n", x, "Not def.");

        }

        else

            printf("%-8.3lf | %8.3lf\n", x, res);

        x += h;

    }

}

void doWhileTab(double a, double b, double h){

    if(b < a) return;

    printf("\n[DoWhile tabulation]\n");

    printf("%-8s | %8s\n", "X", "Y");

    printf("---------+---------\n");

    double x = a;

    do{

        double res = f(x);

        if(isnan(res)){

            printf("%-8.3lf | %8s\n", x, "Not def.");

        }

        else

            printf("%-8.3lf | %8.3lf\n", x, res);

        x += h;

    }while(x < b);

}

(lab07\_02.c):

#include <stdio.h>

#include <conio.h>

#include "lab07\_02\_func.h"

int main(){

    double a, b, h;

    int choice;

    printf("Enter start and end values: ");

    scanf("%lf %lf", &a, &b);

    printf("Enter step: ");

    scanf("%lf", &h);

    printf("\n(0) for\n(1) while\n(2) do while\nWhich cycle would you choose? ");

    do{

        scanf("%d", &choice);

    }while(choice < 0 && choice > 2);

    void (\*tabFuncs[3]) (double, double, double) = {forTab, whileTab, doWhileTab};

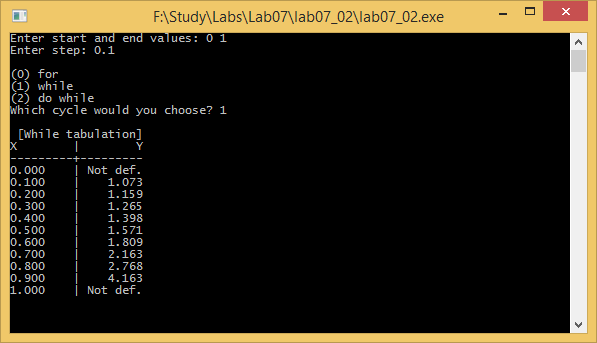
    tabFuncs[choice](a, b, h);

    getch();

}

**Протокол роботи:**

Програма зчитує з клавіатури межі табуляції, крок, а також питає користувача, який цикл використати при табуляції. Потім табулює функцію і результат подає у вигляді таблиці.



Вибір функції табуляції був реалізований на основі масиву вказівників на відповідні функції – це займає менше місця порівнянно з громіздкою конструкцією if/else або switch. Як бачимо, якщо значення функції при заданому аргументі вдається порахувати, то функція робить це, якщо значення функції не визначене, то в правій колонці опиняється запис «Не визначено».

**Висновок:**

Написання цієї програми допомогло нам навчитися використовувати вказіники на функції. А також, ми опанували викорситання масиву вказівників на функції, який зручно використовувти при реалізації меню або сукупності операцій.

№3

Використовуючи рекурсію, ввести групу даних і вивести їх у зворотному порядку, не оголошуючи масиву.

**Код програми:**

(lab07\_03\_func.h):

#ifndef LAB07\_03

#define LAB07\_03

void get\_and\_print();

#endif

(lab07\_03\_func.c):

#include <stdio.h>

#include "lab07\_03\_func.h"

void get\_and\_print(){

    char str;

    str = getchar();

    if(str != EOF){

        get\_and\_print();

    }

    printf("%c", str);

}

(lab07\_03.c):

#include <conio.h>

#include "lab07\_03\_func.h"

int main(){

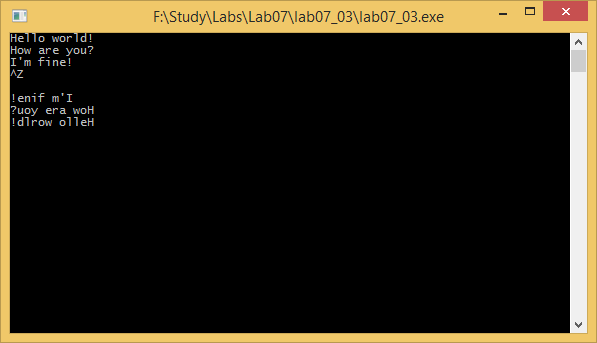
    get\_and\_print();

    getch();

}

**Протокол роботи:**

Функція зчитує набір символів до символу кінця файлу, який ми штучно імітуємо натисканням Ctrl+Z а потім Enter. Потім виводить ці символи в зворотньому порядку за рекурсивним механізмом.



Як бачимо функція відпрацьовує ідеально. Вибір EOF’у як кінцевого символу пов’язаний з тим, що всі інші символи можуть зустрітися у введеному тексті.

**Висновок:**

Під час написанння цієї програми ми навчилися викорситвувти рекурсію у своїх програмах. Також засвоїли правило, що операції, написані перед рекурсивним викликом виконуються у прямому порядку, а ті, що після нього – у зворотньому.