Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Програмування алгоритмічних структур

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 122 – «Комп’ютерні науки»

Київ
КПІ ім. Ігоря Сікорського
2020

Програмування алгоритмічних структур [електронний ресурс]: Навчальний посібник для студентів спеціальності 122 – «Комп'ютерні науки» /
КПІ ім. Ігоря Сікорського; Укладачі: О.С. Крячок, М.П. Воронько. – електронні текстові данні (1 файл: 467 Кб). – Київ: КПІ ім. Ігоря Сікорського, 2020. – 62 с.

Гриф надано Методичною радою
КПІ ім. І. Сікорського
(Протокол № ___ від ___.__.2020 р.)
За поданням Вченої ради
Теплоенергетичного факультету
(Протокол № _ від _.__.2020 р.)

Електронне мережне навчальне видання

Програмування алгоритмічних структур

Укладачі: *Крячок Олександр Степанович, канд. техн. наук, доцент*
Воронько Максим Платонович, асистент

Відповідальний редактор *Аушева Н.М., д-р техн. наук, доцент*

Рецензенти *Недайвода Ігор Володимирович – н.с., Інститут кібернетики НАН України*
Реуцький Микола Олександрович – к.т.н., доцент кафедри електромеханіки ФЕА КПІ ім. Ігоря Сікорського

Посібник розроблений на підставі робочої програми кредитного модуля «Програмування алгоритмічних структур» та призначений для якісної організації виконання практичних робіт студентами і підвищення розуміння основ програмування.

Призначений для студентів, які навчаються за освітньою програмою підготовки бакалаврів за спеціальністю 122 – «Комп'ютерні науки».

Спрямований на формування у студентів умінь та навичок програмування мовою програмування *C*. Забезпечує студентів необхідними теоретичними знаннями для опанування відповідної теми лабораторної роботи та виконання завдань, запланованих впродовж семестру.

Зміст

Вступ

Дисципліна «Програмування алгоритмічних структур» є важливою складовою у підготовці студентів спеціальності 122 – «Комп'ютерні науки». Даний курс базується на мові програмування *C*. Вивчення теоретичних основ програмування алгоритмічних структур, процесу програмування і його технологій є невід'ємною частиною раціональної організації обчислювального процесу. З метою поглиблення теоретичних знань та отримання практичних навичок програмування, проводиться цикл комп'ютерних практичних робіт. Даний посібник допоможе опанувати особливості та можливості мови програмування *C* та зробити важливий крок до вивчення курсу об'єктно-орієнтованого програмування.

Головною метою комп'ютерного практикуму є вивчення інструментальних засобів розробки програмних застосунків, що використовують складні типи даних, і використання цих навичок для проектування програм заданої тематики.

В результаті виконання циклу практичних робіт студент повинен опанувати засоби проектування та отримати навички, необхідні для самостійного проектування та супроводження програмних продуктів.

Даний цикл виконуваних робіт не передбачає створення графічних інтерфейсів (*GUI*), а лише використання консольного вводу і виводу, а також файлових операцій.

При розробці програм застосовується компілятор мови *C*. Мова *C* обрана для навчання тому, з одного боку є універсальною мовою програмування, а з іншого дозволяє опанувати керуванням комп'ютером на достатньо низькому рівні.

Практичні заняття складаються з циклу шести робіт. З метою формування навичок правильного оформлення звітів (див. Додаток А), в даних методичних вказівках наводяться цілком сформовані варіанти оформлення звітних документів. Для виконання робіт, які передбачають індивідуальне завдання, у Додатку В надано варіанти задач.

Практичне заняття № 1:

«Динамічні масиви»

1.1. Завдання

Ознайомитись з особливостями створення динамічних одно- та двовимірних масивів. Об’явити динамічний масив розмірністю ***m*n*** елементів і заповнити його даними згідно варіантів із Додатку Б (наприклад, цілими числами *array[i][j] = i+j*). Записати розмірності (***m, n***) масиву і сам масив в бінарний і текстовий файли. Розширення для бінарного файлу може бути ****.dat***, а для текстового ****.txt***. Назви файлів як і розміри масиву вводити під час виконання програмного коду в один із трьох методів:

- через інтерактивний запит;
- через параметри командного рядка;
- через використання ***config***-файла.

При виконанні завдання забезпечити наступний алгоритм вводу даних у програму:

1. Перевірити наявність ***config***-файлу, який повинен мати таке саме ім’я що й сам виконуваний файл, але з розширенням ****.conf***.

2. Перевірити наявність параметрів командного рядка, що задають необхідні параметри.

3. Якщо немає конфігураційного файлу і не задані всі параметри у командному рядку – перейти до інтерактивного запиту всіх параметрів.

4. Якщо є конфігураційний файл і немає відповідних параметрів у командному рядку, то використовувати його дані, але за наявності даних у командному рядку – використовувати їх.

1.2. Теоретичні відомості

Розглянемо розподіл важливих для розуміння ділянок пам’яті, які виділяються програмі операційною системою. Схема є доволі спрощеною, але дозволяє пояснити питання, зв’язані з динамічним виділенням ділянок пам’яті під структури даних програми.

В процесі компіляції програми створюється така послідовність байтів, яка може

бути виконана процесором після завантаження її у пам'ять комп'ютера. В момент завантаження, така послідовність вже готових байтів розміщується в пам'яті комп'ютера і процесор отримує адресу, з якої він повинен вибрати наступну команду для виконання. Більш детально тут описувати весь цей процес нема сенсу. Подібна послідовність називається виконуваним кодом і в дійсності складається з коду програми і тих даних, які можна було розмістити під час компіляції. Але крім цієї фіксованої частини пам'яті, кожна комірка якої має заздалегідь визначене фіксоване призначення, програмі доступний досить великий обсяг пам'яті іншого типу. Англійською ця пам'ять зветься Heap, а українською – Купа. Ця ділянка пам'яті зазвичай має великі розміри, не ініційована і доступ до неї може бути забезпечений шляхом динамічного виділення деякого обсягу в процесі виконання програми.

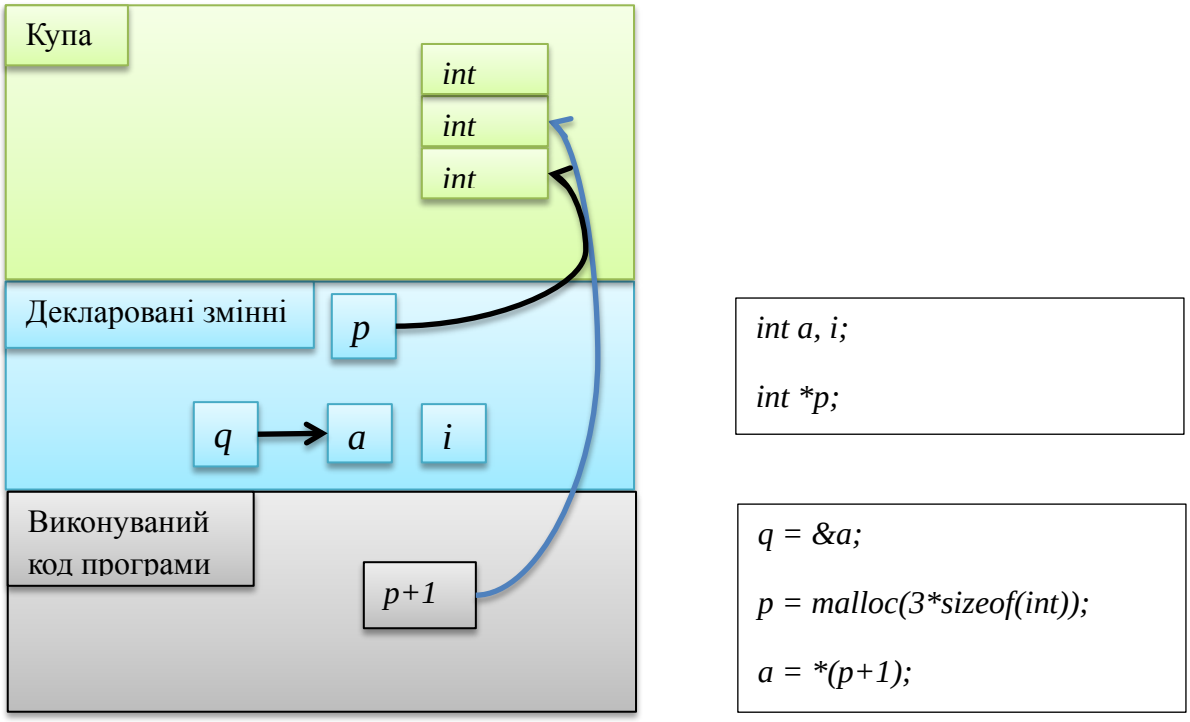

Рисунок 1.1 – Схематичне зображення трьох ділянок пам'яті, доступних програмі під час роботи

Динамічний масив – це така структура даних, яка може створюватися тоді, коли кількість елементів наперед невідома, а також може змінювати свій розмір під час виконання програми.

Формування масивів зі змінними розмірами можна організувати за допомогою вказівників та засобів динамічного розподілу пам'яті. В мові *C* передбачені широкі можливості керування пам'яттю під час виконання програми, які забезпечуються

функціями, що розташовані в бібліотеці *stdlib.h*.

Для виділення і звільнення динамічної пам’яті використовуються наступні функції:

malloc()

*void * malloc (unsigned s)*

Повертає вказівник на початок області динамічної пам’яті довжиною в *s* байт, при невдалому завершенні повертає *NULL*.

calloc()

*void * calloc (unsigned n, unsigned m)*

Повертає вказівник на початок області динамічної пам’яті для розміщення *n* елементів довжиною по *m* байт кожен, при невдалому завершенні повертає *NULL*.

realloc()

*void * realloc (void * p, unsigned s)*

Змінює розмір блоку раніше виділеної динамічної пам’яті до розміру *s* байт, *p* – адреса початку змінюваного блоку, при невдалому завершенні повертає *NULL*.

free()

*void * free (void p)*

Звільняє раніше виділену ділянку динамічної пам’яті, *p* – адреса першого байту.

В усіх функціях виділення пам’яті необхідно передавати кількість байт, а зазвичай програмісту відома кількість елементів або структур, що будуть в ній розміщені. Для визначення дійсної кількості байт, що буде відведена під дану змінну або структуру в *C* введена функція *sizeof()*, яка повертає кількість байт в даному типі змінної або структури.

Функція для формування одновимірного динамічного масиву:

```
int * make_mas (int n)
{
int * mas;
mas = (int *) malloc (n * sizeof (int));
for (int i = 0; i <n; i ++)
mas [i] = random (10);
return mas;
}
```

Для виділення пам’яті використовується функція *malloc()*, параметром якої є розмір ділянки, що виділяється пам’яті рівний *n * sizeof (int)*. Так як функція *malloc()* повертає нетипізований вказівник *void* *, то необхідно виконати перетворення отриманого нетипізованого вказівника на вказівник типу *int *.*

Звільнити виділену пам’ять можна функцією *free (mas).*

Після виділення деякої ділянки пам’яті така ділянка резервується і не може бути повторно використана до тих пір, поки вона не буде звільнена за допомогою явної функції *free().* Функція *free(p)* “знає” розмір ділянки, що пов’язана з вказівником, тому достатньо в ній вказати вказівник на відповідну ділянку. Після виконання функції *free()* всі вказівники на неї стають невизначеними і можуть бути використані для адресації інших ділянок. Якщо трапиться, що на дану ділянку не вказує жоден вказівник, вона стає недоступною, але не повертається до вільної частини купи і не може бути використаною. Саме така помилка частіше всього призводить до важко визначуваної помилки, що зветься “витік пам’яті” (*memory leak*).

Добрим правилом вважається звільняти динамічно виділені ділянки пам’яті в тому ж блоці коду, де вона була виділена, наприклад в функції *main()* або в будь-якій функції.

При виділенні динамічної пам’яті під масив розміри масиву повинні бути повністю визначені до операції виділення пам’яті.

Для створення двовимірного масиву можливі два підходи, кожен з яких має свої переваги і недоліки:

В першому випадку для виділення масиву *M*N* визначається повний розмір ділянки пам’яті під весь масив як *arr_size = M*N*sizeof(arr_elem)*. Недоліком такого підходу є те, що безпосереднє адресування елементу *a[i][j]* неможливо, бо під час компілювання невідомі розміри масиву, і приходиться використовувати обчислення індексу одновимірного масиву *j*M+i*, але тут є і переваги – масив витрачає пам’ять лише на зберігання необхідних даних, суттєво спрощуються операції при зміні розмірів масиву і при запису масиву у бінарний файл.

В другому випадку, створюється одновимірний масив вказівників, які далі слугують вказівниками на нові одновимірні масиви.

*// виділення динамічної пам’яті 100 * sizeof (int) байт*

*int * a = (int *) malloc(100 * sizeof (int));*

```
// Виділення динамічної пам’яті під двовимірний динамічний масив * /
int ** form_matr (int n, int m)
{
int ** matr = (int **) malloc(n*sizeof(int*)); // виділення пам’яті під масив вказівників
for (int i = 0; i <n; i ++) // виділення пам’яті для масиву значень
    matr [i] = (int*) malloc(m*sizeof(int));
return matr; // повернення вказівника на масив вказівників
}
```

Видалення з динамічної пам’яті двовимірного масиву другого типу здійснюється в зворотному порядку, тобто спочатку звільняється пам’ять, виділена під одномірний масив з даними, а потім пам’ять, виділена під одномірний масив вказівників.

Перевагою такого підходу є можливість звернення до елементів масиву звичайним методом *a[i][j]*, але розмір такої структури більший, а засоби керування розмірами складніші.

Для виділення пам’яті, заповнення масивів, видалення і додавання елементів (рядків, стовпців) написати окремі функції. У функції *main()* повинні бути розміщені тільки опис змінних і звернення до відповідних функцій:

```
int main() {
int n;
cout<<"N?";cin>>n;
person*mas=form_mas(n);
init_mas(mas,n);
print_mas(mas,n);
return 1;
}
```

Передбачити можливість вводу нечислових значень для розмірів масиву.

1.3. Опис програми

На початку програми відбувається перевірка наявності в керувальному рядку параметрів, необхідних для роботи програми. Для спрощення розбору рядка жорстко

задаємо позиції, в яких знаходяться параметри програми. Наприклад, якщо програма називається dynarr.exe, то повний керувальний рядок матиме вигляд:

dynarr.exe <m> <n> output.txt output.bin

Де замість <*m*> і <*n*> стоять необхідні числові значення, а імена вихідних файлів можуть бути довільними.

Після цього, якщо ці параметри не задані, або задані не повністю, то іде перевірка наявності конфігураційного файлу, який є текстовим файлом. При його наявності з нього читають всі чотири параметри, але використовують лише ті, які відсутні в керувальному рядку. Наприклад:

dynarr.exe 5 5

І є конфігураційний файл dynarr.conf, в якому записані такі дані:

4

8

outp.txt

outp.bin

В цьому випадку для роботи програми буде обрано розміри масиву *5 * 5* і назви вихідних файлів *outp.txt* і *outp.bin*.

Якщо після такої процедури залишаються невизначені параметри, то програма запитує параметри, яких бракує в діалоговому режимі. Після отримання всіх необхідних даних починається робота програми зі створення динамічного масиву, заповнення його згідно завдання і виводу його на екран і в два файли. В файли виводяться розміри масиву, а потім всі елементи масиву. Необхідно зрозуміти різницю між текстовим і бінарним файлами - текстовий може бути прочитаний у будь-якому текстовому редакторі бо вміщує лише символи певної кодової таблиці, а бінарний не може бути прочитаний текстовим редактором, бо не є послідовністю символів.

1.4. Результати виконання програми

В результаті виконання програми на екран монітора будуть виведені розміри масиву, які були задані одним з трьох методів, а в теці з файлами з’являться два файли.

В текстовому буде збережена та ж інформація, щ і виведена на екран, а в бінарному буде зберігатися в бінарній формі розміри і сам масив. Його можна

прочитати тільки спеціальними редакторами. Наприклад вільним редактором *HxD*.

Копію екрана з результатами роботи програми наведено на рисунку 1.1.

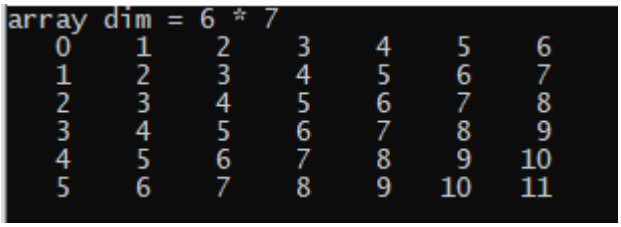

Рисунок 1.2 – Результат виконання програми

1.5. Висновки по роботі

Під час виконання комп'ютерного практикуму розглянуто найбільш вживані методи вводу даних у програму, створення динамічного масиву та виводу його у текстовий і бінарний файли. Також виконано проектування програми.

1.6. Лістинг програми

```
// lab1.cpp: Defines the entry point for the console application.
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char argv[]) {

    int m=0,n=0;   // Розміри массиву
    char tof[64]; // Буфер для назви текстового файлу
    char bof[64]; // Буфер для назви бінарного файлу
    char tmp[64]; // Допоміжний буфер
    int *arr;       // Вказівник на масив
    memset(tof,0,64); // Заповнення буферів 0-ми
    memset(bof,0,64);
    FILE *ff;             // Файлова змінна
    for (i=1; i<argc; i++){
// Читаємо і перетворюємо параметри керівного рядка
        if (i==1) m = atoi(argv[1]);
        if (i==2) n = atoi(argv[2]);
        if (i==3) strcpy(tof,argv[3]);
        if (i==4) strcpy(bof,argv[4]);
    }
```

```
    ff = fopen("init.cfg","rt");
    if (ff) {
// Якщо є конфігураційний файл і не всі параметри задані у керівному
// рядку, то вони зчитуються з кофігураційного файлу
        fgets(tmp,12,ff);
        if (m==0) m=atoi(tmp);
        fgets(tmp,12,ff);
        if (n==0) n=atoi(tmp);
        fgets(tmp,64,ff);
        if (strlen(tof)==0) strncpy(tof,tmp,strlen(tmp)-1);
        fgets(tmp,64,ff);
        if (strlen(bof)==0) strncpy(bof,tmp,strlen(tmp)-1);
        fclose(ff);
    };
// Якщо залишилися невизначені параметри і після цього, далі іде
// діалоговий ввід даних
    if (m==0) {
        printf("Input M"); scanf("%d",&m);
    };
    if (n==0){
        printf("Input N"); scanf("%d",&n);
    };
    if (strlen(tof)==0){
        printf("Name of text file"); scanf("%s",tof);
    };
    if (strlen(bof)==0){
        printf("Name of bin file"); scanf("%s",bof);
    };
    arr = malloc(m*n*sizeof(int));
// Друкуємо розміри масиву
    printf("array dim = %d * %d\n",m,n);
    for (int i=0; i<m; i++){
        for (int j=0; j<n; j++){
            arr[index(n,i,j)]=i+j;
        }
    }
//Роздруковуємо сам масив на консолі
    for (int i=0; i<m; i++){
        for (int j=0; j<n; j++){
            printf(" %3d ",arr[index(n,i,j)]);
        }
```

```
            printf("\n");
        }
    // Зберігаємо розміри і сам масив у текстовому файлі
        ff = fopen(tof,"wt");
        fprintf(ff,"%d %d\n",m,n);
        for (int i=0; i<m; i++){
            for (int j=0; j<n; j++){
                fprintf(ff," %3d ",arr[index(n,i,j)]);
            }
            fprintf(ff,"\n");
        }
        fclose(ff);
    // Зберігаємо розміри і масив у бінарному файлі
        ff = fopen(bof,"wb");
        dim[0]=m; dim[1]=n;
        fwrite(dim,sizeof(int),2,ff);
        fwrite(arr,sizeof(int),n*m,ff);
        fclose(ff);
    // Звільняємо виділену під масив пам'ять
        free(arr);
        return 0;
    }
```

Контрольні запитання

1. Які масиви називаються динамічними?
2. В чому складність індексування багатомірних динамічних масивів?
3. Які два методи формування багатомірних динамічних масивів їх недоліки і переваги?
4. Чим відрізняється текстовий файл від бінарного?
5. В яких випадках варто використовувати бінарні файли, а в яких текстові?

Практичне заняття № 2:

«Методи сортування масивів»

2.1. Завдання

Ознайомитись з алгоритмами сортування масивів та способами їхньої реалізації.

У якості індивідуального завдання необхідно написати програмний код, у якому реалізується сортування масивів 3-а методами (наприклад, бульбашки, вставок, вибору, сортування Шелла, Гоара (*Hoare*) та швидкого сортування). Вихідні дані по варіантах – використати результати, що отримано в роботі №1 (тип даних, розмірність масиву, ім'я та розширення файлу даних, ім'я та розширення файлу конфігурації). Для створення всіх масивів, що використовуються в програмі, реалізувати динамічне виділення пам'яті.

2.2. Теоретичні відомості

Метод бульбашки (Bubble Sort)

Сутність методу полягає в багаторазовому проході по масиву. На кожному кроці послідовно порівнюються пари сусідніх елементів, і якщо порядок в такій парі невірний, то елементи в парі міняються місцями. При проході алгоритму, елемент, що стоїть не на своїй позиції, «спливає» до потрібної позиції як бульбашка, звідки і назва алгоритму.

Є дві версії методу бульбашки – з фіксованою кількістю кроків і з перевіркою на закінчення сортування. Друга версія суттєво зменшує кількість кроків у випадку, коли масив частково впорядкований. Ця перевага особливо відчутна на великих масивах.

У методі з фіксованою кількістю кроків зазвичай використовують два вкладені цикли з довжиною циклу, рівною довжині масиву. Якщо в масиві *M* елементів, то сортування займе *M*M* проходів.

У методі із зупинкою по готовності зовнішній цикл є циклом *while* де перевіряється прапорець завершення. Наприклад:

```
int flg=1;
while (flg) {
        flg=0;
        for (…){
```

```
    if (a[i+1]<a[i]) {
        swap(a[i+1],a[i]);
        flg=1;
}}}
```

Такий метод завершується в той момент, коли жодної заміни не відбулося, а значить весь масив відсортовано. В найгіршому випадку кількість циклів буде рівна першому методу, але якщо масив частково впорядкований, то метод зупиниться раніше.

Сортування зі вставками (Insertion Sort)

Сортований масив переглядається в порядку зростання номерів і кожен елемент записується в уже переглянуту частина масиву так, щоб зберегти порядок.

Сортування вибором

Спочатку відшукується найменший елемент масиву, потім він міняється місцями з елементом, що стоїть першим у сортованому масиві. Далі, знаходиться другий найменший елемент і міняється місцями з елементом, що стоїть другим у вихідному масиві. Цей процес триває до того часу, поки весь масив не буде відсортований.

Метод Шелла

Цей метод полягає в порівнянні елементів масиву, розділених однаковою відстанню таким чином, щоб елементи на цій відстані були впорядковані. Потім ця відстань ділиться навпіл і процес триває. В кінці відстань рівна 1 і якщо змін немає, то масив відсортований.

Швидке сортування

Цей метод розглядає масив, як список значень. Спочатку виділяється середнє значення як сепаратор (фактор розбиття) списку. Список розбивається на два: в одному з них значення менше сепаратора, а в іншому – більше або рівні. Далі процедура сортування рекурсивно викликає саму себе для кожного з двох списків. Кожен раз при виклику сортування список елементів розбивається на два менших.

Швидке сортування Гоара (Hoare)

Цей метод ґрунтується на послідовному поділі набору даних на блоки меншого розміру таким чином, що між значеннями різних блоків забезпечується відношення впорядкованості (для будь-якої пари блоків всі значення одного з цих блоків не

перевищують значень іншого блоку).

2.3. Описання програми

Написання програми відбувалося методом додавання функцій, що реалізують вказані вище методи сортування, до програми попереднього комп'ютерного практикуму з метою створити програму, яка буде здатна здійснювати сортування усіма вивченими методами одразу. Крім функції *main()*, *void output(int p[], int start, int end)*, що виводить масив на екран, *int maximum(int p[], int v)*, яка повертає значення максимального елемента на ділянці масиву, *void bubble(int p[], int x[])*, *void choise(int p[], int x[])* і *void insert(int p[], int x[])*, які виконують сортування масива відповідними методами і вже були описані, було додано функції *void shell(int p[], int x[])*, *void hoar(int p[], int first, int last)*, *void quick(int p[], int first, int last)*, а також кілька допоміжних: *void swap(int &a, int &b)*, яка дозволяє міняти елементи масиву місцями, та *void refresh(int p[], int x[])*, що використовується в разі необхідності перезаписати в новостворений масив той початковий, з яким ведеться робота.

2.4. Результати виконання програми

Копію екрана з результатами роботи програми наведено на рисунку 2.1.

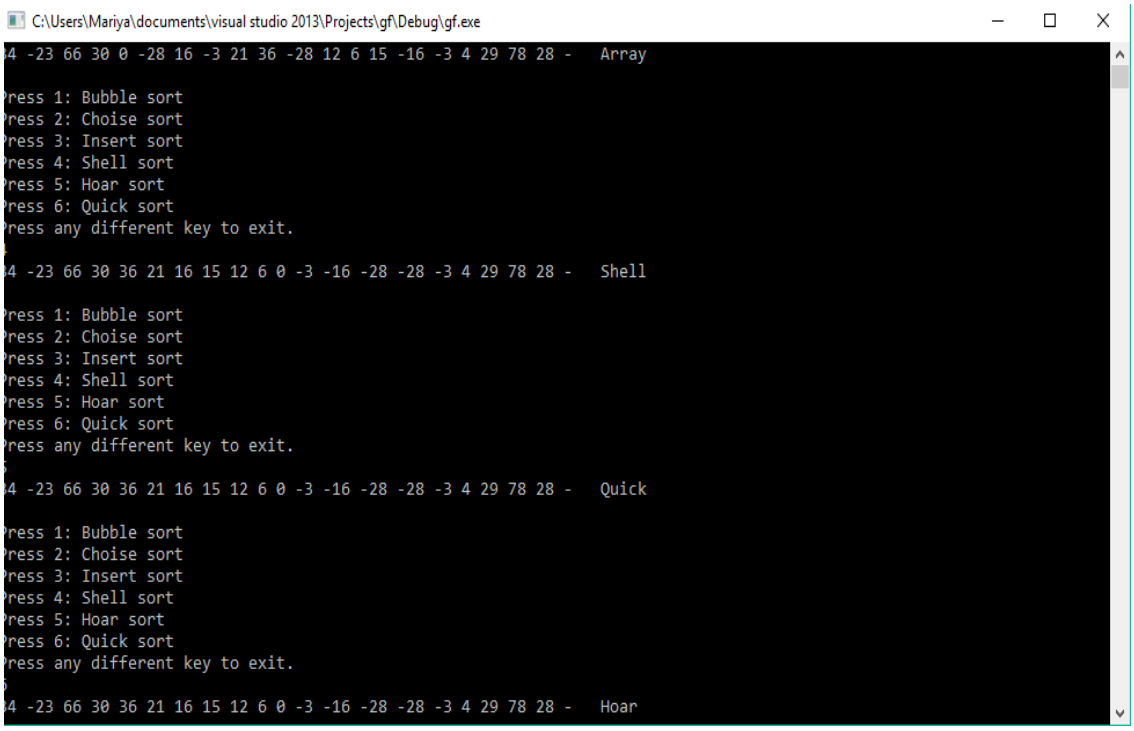

Рисунок 2.1 – Результат виконання програми

2.5. Висновки по роботі

Під час виконання комп'ютерного практикуму виконано проектування програми, в якій із головної функції викликається ряд допоміжних функцій, які дозволяють відсортувати за спаданням елементи масиву такими методами, як бульбашка, вставка, вибір, Шелла, Гоара та швидке сортування. Таким чином, поставлена у завданні задача була вирішена.

2.6. Лістинг програми

```
// lab2.cpp: Defines the entry point for the console application.
#include "stdafx.h"
#include <iostream>
#include <stdio.h>
#include <ctime>
#include <cstdlib>
#include <windows.h>
using namespace std;
void bubble(int p[], int x[]);
void choise(int p[], int x[]);
void insert(int p[], int x[]);
void shell(int p[], int x[]);
void hoar(int p[], int first, int last);
void quick(int p[], int first, int last);
int maximum(int p[], int v);
void output(int p[], int start, int end);
void swap(int &a, int &b);
void refresh(int p[], int x[]);
#define size 20
#define from 4
#define to 14

void main()
{
	int x[size], r[size];
	srand(time(NULL));
```

```
	for (int i = 0; i < size; i++)
	x[i] = rand() % (80 + 30 + 1) - 30;

	output(x, 0, size); cout << "-	Array \n";
	{for (;1;)
	{	char answer;
		printf("\nPress 1: Bubble\nPress 2: Choise\nPress 3: Insert\nPress 4: Shell \nPress 5: Hoar\nPress 6: Quick sort\nPress any different key to exit.\n");
		scanf("%s", &answer);
		switch (answer)
		{case '1': bubble(r, x); break;
		case '2': choise(r, x); break;
		case '3': insert(r, x); break;
		case '4': shell(r, x); output(r, 0, size); cout << "-	Shell \n"; break;
		case '5': refresh(r, x); quick(r, from, to); output(r, 0, size); cout << "-Quick \n"; break;
		case '6': refresh(r, x); hoar(r, from, to); output(r, 0, size); cout << "-Hoar \n"; break;
		default: exit(0);
		}}}
	system("pause");
}

void bubble(int r[], int x[])
{	refresh(r, x);
	for (int i = from; i < to-1; i++)
		for (int j = from; j < to; j++)
			if (r[j]<r[j + 1])
				swap(r[j], r[j + 1]);
	output(r, 0, size); cout << "-	Bubble\n";}

void choise(int r[], int x[])
{	refresh(r, x);
	for (int j = from; j < to+1; j++)
	{	int k = maximum(r, j);
		if (r[j]<r[k]) swap(r[j], r[k]);
	}
	output(r, 0, size); cout << "-	Choise \n";
}

void insert(int r[], int x[])
```

```
{       refresh(r, x);
        for (int i = from; i < to+1; i++)
                for (int j = i; (j > from && r[j - 1] < r[j]); j--)
                        swap(r[j], r[j - 1]);
        output(r, 0, size); cout << "-    Insert \n";
}

void shell(int p[], int x[])
        {       refresh(p, x);
                int i, j, step, n=11;
                for (step = n / 2; step > 0; step /= 2)
                        for (int i = from; i < (to+1-step); i++)
                        {       int j = i;
                                while (j >= from && p[j] < p[j + step])
                                        {swap(p[j], p[j + step]); j--;}
                        }
        }

void quick(int p[], int first, int last)
{       int hard_element, llim = first, rlim = last;
        hard_element = p[first];
        while (first < last)
        {       while (p[last] <= hard_element && first < last) last--;
                if (first != last)
                {       p[first] = p[last];
                        first++;
                }
                while ((p[first] >= hard_element) && (first < last)) first++;
                if (first != last) {p[last] = p[first]; last--;}
        }
        p[first] = hard_element; hard_element =first; first = llim; last = rlim;
        if (first < hard_element) quick(p, first, hard_element - 1);
        if (last > hard_element) quick(p, hard_element + 1, last);
}

void hoar(int p[], int first, int last)
{       int i = first, j = last, step = -1, condition = 1;
        if (first >= last) return;
        do
        {if (condition == (p[j] > p[i]))
                {swap(p[i], p[j]); swap(i, j); step = -step; condition = !condition;}
```

```
            j += step;
        } while (j != i);
        hoar(p, first, i - 1); hoar(p, i + 1, last);
}

int maximum(int p[], int v)
{       int index = v; int max = p[v];
        for (int j = v; j <= to; j++)
        {if (p[j] > max) { max = p[j]; index = j; }}
        return index;
}

void swap(int &a, int &b)
{a+=b; b=a-b; a-=b;}

void refresh(int p[], int x[])
{for (int i = 0; i < size; i++) p[i] = x[i];}
void output(int p[], int start, int end)
{for (int i = start; i < end; i++) cout << p[i] << " ";}
```

Контрольні запитання

1. Перерахуйте кілька методів сортування.
2. Поясніть особливості методу «бульбашки» і два його варіанти.
3. Поясніть особливості сортування вставками.
4. Поясніть особливості сортування вибором.
5. Поясніть особливості сортування методом Шелла.
6. Поясніть особливості швидкого сортування.
7. Поясніть особливості швидкого сортування Гоара.
8. Який з методів сортування використовує рекурсію?

Практичне заняття № 3:
«Структури. Операції над масивами структур»

3.1. Завдання

Ознайомитись з операціями додавання записів, редагування, пошуку, виведення та видалення структур.

У якості індивідуального завдання необхідно написати програмний код, у якому реалізовано масив структур за індивідуальним завданням (Додаток Б), результати розрахунків.

3.2. Теоретичні відомості

Структура в мові *C* – це тип даних, який складається з визначеної кількості елементів, що називаються членами структури.

Перш ніж використовувати екземпляр структури в програмі, його необхідно ініціалізувати. Зробити це безпосередньо в оголошенні неможливо, оскільки оголошення структури являє собою лише абстрактний опис схеми структури і не зв'язано з виділенням конкретної пам'яті для її полів. Отже, початкові значення полів просто нікуди записати. Ініціалізація можлива лише при визначенні екземпляра структури.

```
#include <iostream.h>
int main()
{
struct Complex
{
double Re, Im;
} A = {0, 1};
cout << A.Re << endl << A.Im;
return 0;
}
```

Структури можна присвоювати одна одній. Присвоювання можливе лише в тому випадку, коли структури мають однаковий тип.

Із структурами можна виконувати наступні дії:

- доступ до членів структури;
- копіювання та присвоєння структур;
- взяття адреси структури.

З однієї сторони, структури можуть бути елементами масивів, а з іншої, – масиви можуть бути членами структур.

Визначення масиву структур нічим не відрізняється від визначення масиву вбудованого типу. Природно, оголошення (логічна схема) структури повинне передувати її визначенню.

```
#include <iostream.h>
int main()
{
struct Complex
{
double Re, Im;
};
Complex A[100];
return 0;
}
```

Для доступу до екземпляра структури, що є елементом масиву, використовується її індекс. Наприклад, щоб вивести на екран дійсну і уявну частини 50-го елемента масиву A, можна виконати наступну інструкцію.

```
cout << A[49].Re << "+ i*" << A[49].Im;
```

Структури можуть передаватися функціям шляхом передачі окремих елементів структури, передачі всієї структури або передачі вказівника на структуру. Коли структури або окремі їх елементи передаються функції, проходить передача по значенню. А тому функція, що викликає не може змінювати елементи у структурі що викликається. Щоб здійснити передачу структури по посиланню, необхідно передати їй адресу. Масиви структур автоматично передаються за посиланням.

3.3. Описання програми

Код програми розбито на декілька функцій, кожна з яких реалізує пункти меню. У функції main описано алгоритм зацикленого меню. Функція *void insert()* відповідає за додавання записів в базу даних і містить у собі підфункції *void insert_surname(), void insert_name(), void insert_dadname(), void insert_card_number(), void insert_balance(), void insert_date()*, які дозволяють заповнити поля прізвища, імені, по батькові, номера картки, баланс та дату останнього відповідно. Функції *void correct_surname(char name[]), void correct_dadname(), void correct_card_number()* у свою чергу викликають функції коригування введених даних під єдиний шаблон. Функції *void ask_me(char name[])* та *void center_out(int answer)* є допоміжними при коригуванні і забезпечують інтерфейс з користувачем під час виконання цього процесу. Функції *void search(), void change(), void deleting()* та *void output()* відповідають за пошук записів, зміну їх полів, видалення та виведення відповідно.

3.4. Результати виконання програми

При запуску програми користувач побачить меню і може виконати описані в ньому дії.

Копію екрана можна побачити на рисунку 3.1.

3.5. Висновки по роботі

Під час виконання комп’ютерного практикуму виконано проектування програми, в якій із головної функції викликається ряд допоміжних функцій, які охоплюють наступні завдання по роботі з структурами: створення запису, його пошук та редагування, видалення та виведення на екран. Очевидно, що під час виконання поставленого завдання було отримано необхідні навички по роботі зі структури та усвідомлення механізмів їх роботи у мові *C*. Таким чином, поставлена у завданні задача була вирішена.

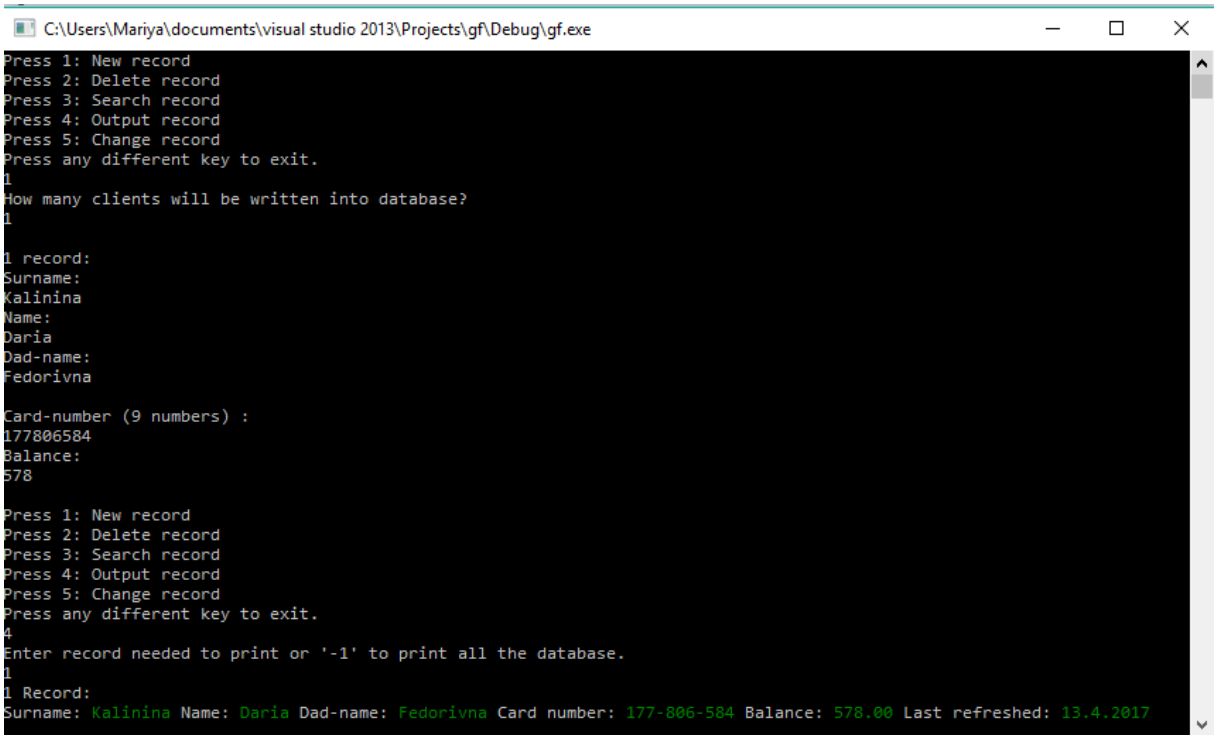

Рисунок 3.1 – Результат виконання програми

3.6. Лістинг програми

```
// lab3.cpp: Defines the entry point for the console application.
#include "stdafx.h"
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <cstdlib>
#include <cstring>
#include <windows.h>
#include<string>
#include <time.h>
#define def_settings SetConsoleTextAttribute(hConsole, (WORD)((Black << 4) | LightGray))
#define correction_settings SetConsoleTextAttribute(hConsole, (WORD)((Black << 4) | Red))
#define good_settings SetConsoleTextAttribute(hConsole, (WORD)((Black << 4) | Green))
#define int_array 10
#define float_array 11
using namespace std;
void insert();
void insert_surname(); void correct_surname(char name[]); void insert_name();
void insert_dadname(); void correct_dadname();
```

```
void insert_card_number(); void correct_card_number();
void insert_balance(); void insert_date();
void ask_me(char name[]);
void output(); void center_out(int answer);
void search(); void change(); void deleting();
enum ConsoleColor {
  Black = 0, Blue = 1, Green = 2, Cyan = 3, Red = 4, Magenta = 5, Brown = 6,
LightGray = 7, DarkGray = 8, LightBlue = 9, LightGreen = 10, LightCyan = 11,
LightRed = 12, LightMagenta = 13, Yellow = 14, White = 15 };
HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
struct client {
  char surname[100], name[100], char dadname[100];
  long int card_number;
  float balance;
  short int day, month, year; };
struct client array[100], tmt;
unsigned int records = 0;
short int qual, clicker;
char int_numbers[int_array] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' };
char float_numbers[float_array] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.' };
bool test(char ch, const char *keys, unsigned int size)
{for (int i = 0; i<size; i++) if (keys[i] == ch) return true;
  return false; }
int getNumber(int size, char array[])
{ string n; char ch;
  while ((ch = getch()) != '\r')//enter
        if (test(ch, array, size)) { n += ch; cout << ch; }
  cout << endl; return atoi(n.c_str()); }
void main()
{ cout << "Welcome to Privat Bank.\n";
  {for (; 1;)
  {     char answer;
        printf("\n1:New\n2:Delete\n3:Search\n4:Output\n5:Change\nAny dif key to
exit.\n");
        scanf("%s", &answer);
        switch (answer)
        {case '1': insert(); break;
         case '2': if (records != 0) deleting(); else cout << "\nEmpty database\n";
break;
        case '3': if (records != 0)search(); else cout << "\nEmpty database\n";   break;
        case '4': if (records != 0)output(); else cout << "\nEmpty database\n";   break;
```

```
            case ‘5’: if (records != 0)change(); else cout << "\nEmpty database\n"; break;
            default: exit(0);}}}
  system("pause");}
 void insert()
 {cout << "How many clients will be written into database?\n";
  cin >> qual;
  for (int i = 0; i < qual; i++)
  {     cout << "\n" << i + 1 << " record:\n";
        insert_surname(); insert_name(); insert_dadname();
        insert_card_number(); insert_balance(); insert_date();
        records++; }}
 void insert_surname()
 {cout << "Surname:\n";
  cin >> array[records].surname;
  clicker = 0;
  correct_surname(array[records].surname);}
 void correct_surname(char name[])
 {int letter_code = 1, i = 0, length = strlen(name), mistake = 0; //видалення сміття
  for (int j = 0; j < length; j++)
        {letter_code = name[i]; if (clicker != 2) {
            if ((letter_code < 65 || (letter_code < 90 && letter_code>97) || letter_code>122) && letter_code != 45 && letter_code != 39)
            {for (int j = i; j < length; j++) name[j] = name[j + 1];
            } i++;}
        else if ((letter_code<65 || (letter_code<90 && letter_code>97) || letter_code>122))
            {for (int j = i; j < length; j++) name[j] = name[j + 1];
            } i++; }
  name[length - mistake + 1] = ‘\0’;
  for (int j = 0; j < length; j++) //верхні-нижні реєстри
  {letter_code = name[j];
  if (j==0 && letter_code>96 && letter_code<123) {int new_code=letter_code-32; name[j]=new_code; mistake++;}
  if (name[j-1]==45||name[j-1]==39) {int new_code=letter_code-32; name[j]=new_code; mistake++;}
        if (j != 0 && name[j - 1] != 45 && name[j - 1] != 39 && letter_code > 64 && letter_code < 91) { int new_code = letter_code + 32; name[j] = new_code; mistake++; }}
  if (strlen(name) == 0)
  {     correction_settings;
        cout << "WRONG INPUT\n";
```

```
            def_settings; ask_me(name);
            insert_surname(); }
      else if (mistake != 0) ask_me(name); }
    void insert_name()
    {cout << "Name:\n";          cin >> array[records].name;
      clicker = 1;
      correct_surname(array[records].name); }
    void insert_dadname()
    {cout << "Dad-name:\n";
      cin >> array[records].dadname;
      clicker = 2;
      correct_surname(array[records].dadname); }
    void insert_card_number()
    {cout << "\nCard-number (9 numbers) :\n";
      array[records].card_number = getNumber(int_array, int_numbers);
      correct_card_number(); }
    void correct_card_number()
    {if                    (array[records].card_number<100000000                    ||
array[records].card_number>999999999)
      {     correction_settings;
            cout << "WRONG INPUT";
            def_settings;
            insert_card_number(); }}
    void insert_balance()
    {cout << "Balance:\n";
      array[records].balance = getNumber(float_array, float_numbers);}
    void insert_date()
    {struct tm *tim;
      time_t tt = time(NULL);
      tim = localtime(&tt);
      array[records].day = tim->tm_mday;
      array[records].month = tim->tm_mon;
      array[records].year = 1900 + tim->tm_year;}
    void ask_me(char name[])
    {char answer;
      correction_settings;
      cout << "INPUT CORRECTION\n";
      def_settings;
      cout << name << "\nIs that right? (y/n) "; cin >> answer;
      if (answer != '1' && answer != 'y' && answer != 'Y')
      {     if (clicker == 0) insert_surname();
```

```
            else insert_name(); }}
        void output()
        { int answer;
          cout << "Enter record needed to print or '-1' to print all the database.\n"; cin >> answer;
          if (answer <= records && answer != -1) center_out(answer);
          else for (int i = 1; i < records + 1; i++) center_out(i); }
        void center_out(int ans)
        { cout << ans << " Record:\n";
          cout << "Surname: "; good_settings; cout << array[ans - 1].surname; def_settings;
          cout << " Name: "; good_settings; cout << array[ans - 1].name; def_settings;
         cout << " Dad-name: "; good_settings; cout << array[ans - 1].dadname; def_settings;
          int first = array[ans - 1].card_number / 1000000, second = (array[ans - 1].card_number - first * 1000000) / 1000, third = array[ans - 1].card_number - first * 1000000 - second * 1000;
          cout<<"Card number:"; good_settings; printf("%d-%d-%d",first,second,third); def_settings;
          cout<<"Balance:"; good_settings; printf("%.2f",array[ans-1].balance); def_settings;
          cout<<"Last refreshed:"; good_settings; printf("%d.%d.%d\n", array[ans-1].day, array[ans-1].month, array[ans - 1].year); def_settings; }
        void change()
        { int numrec = records; int anssswer;
          cout << "Enter record number:"; cin >> records; if (records > 0 && records <= numrec)
        { records--; printf("\nWhat variable to change?\n 1:Surname\n2: Name\n3: Dad name\n4: Card number\n5: Balance\n 6: Date\nPress any different key to exit.\n");
                scanf("%d", &anssswer);
                switch (anssswer)
                {case 1:   insert_surname(); break;
                 case 2:   insert_name();   break;
                 case 3:   insert_dadname();   break;
                 case 4: insert_card_number(); break;
                 case 5: insert_balance(); break;
                 case 6: insert_date();   break;
                 default: exit(0); }}
        records = numrec; }
        void search()
        { char answer, ansi[100]; int count = 0, iansi;
        printf("\nPress 1: Search by surname\nPress 2: Search by Name\nPress 3: Search by Dad-name\nPress 4: Search by card number\n Press 5: Search by balance\n Press 6 Search by refreshed date\n");
```

```
cin >> answer;
switch (answer)
{case ‘1’: {
        cout << "Enter needed surname\n"; cin >> ansi;
        for (int j = 1; j <= records; j++)
            if (strcmp(array[j - 1].surname, ansi) == 0) { count++; center_out(j); }
        cout << "\n Total:" << count << " records\n"; count = 0; break; }
  case ‘2’: cout << "Enter needed name\n"; cin >> ansi;
        for (int j = 1; j <= records; j++)
            if (strcmp(array[j - 1].name, ansi) == 0) { count++; center_out(j); }
        cout << "\n Total:" << count << " records\n"; count = 0; break;
case ‘3’:cout << "Enter needed dad-name\n"; cin >> ansi;
        for (int j = 1; j <= records; j++)
            if (strcmp(array[j - 1].dadname, ansi) == 0) { count++; center_out(j); }
        cout << "\n Total:" << count << " records\n"; count = 0; break;
case ‘4’:cout << "Enter needed card number\n"; cin >> iansi;
        for (int j = 1; j = records; j++)
            if (array[j - 1].card_number == iansi) { count++; center_out(j); }
        cout << "\n Total:" << count << " records\n"; count = 0; break;
case ‘5’: char anss;
        printf("\nPress 1: Balance less than.. \nPress 2: Balance more than ...\n");
        cin >> anss;
        int less, more;
        if (anss == ‘1’) {
            cout << "Less than?\n";   less = getNumber(float_array, float_numbers);
            for (int j = 1; j <= records; j++)
                if (array[j - 1].balance < less) { count++; center_out(j); }
            cout << "\n Total:" << count << " records\n"; count = 0; break;}
        else   { cout << "More than?\n";      more = getNumber(float_array,
float_numbers);
            for (int j = 1; j <= records; j++)
            if (array[j - 1].balance > more) { count++; center_out(j); }
            cout << "\n Total:" << count << " records\n"; count = 0; break; }
  default: exit(0);}}
void deleting()
{char answer; int aaa;
  printf("\nPress 1: To delete all records\nPress 2: To delete ‘n’ record\n");
  cin >> answer;
  switch (answer)
  {case’1’: {for (int k=0; k<records; k++) array[k]=tmt; records=0; break;
            cout << "\nEmpty database\n"; }
```

```
case'2':   {cout << "\nWhich record should be deleted?\n"; cin >> aaa;
           if (aaa <= records && aaa>0)
       for (int f = aaa; f <= records; f++)
       {array[f - 1] = array[f]; records--; cout << "\nDone."; }
break; }}}
```

Контрольні запитання

1. Дайте визначення структури даних.

2. Як здійснюється ініціалізація структури?

3. Чи можна присвоювати структури одна одній?

4. Які дії можна виконувати над структурами?

5. Чи можуть структури бути елементами масивів? Наведіть приклади.

6. Чи можуть масиви бути елементами структур? Наведіть приклади.

7. Що використовується для доступу до екземпляра структури, що є елементом масиву?

8. Поясніть поняття передачі змінної до функції по значенню.

9. Що відбувається, якщо в функції змінити значення змінної, яка була передана по значенню?

10. Поясніть поняття передачі змінної до функції по посиланню.

11. Що відбувається, якщо в функції змінити значення змінної, яка була передана по посиланню?

Практичне заняття № 4:
«Файли. Створення, запис, читання та видалення»

4.1. Завдання

Ознайомитись з функціями, що призначені для роботи з файлами. Дослідити особливості їх застосування. Написати програму, що виконує копіювання файлу будь-якого типу.

У якості індивідуального завдання необхідно написати код програми, що виконує копіювання даних структурного типу (використати варіанти завдань комп'ютерного практикум № 3) у файл в вигляді таблиці.

4.2. Теоретичні відомості

Вказівник на файл (змінна-вказівник типу *FILE* *) – це вказівник на інформацію, що визначає різні параметри файлу, включаючи його ім'я, стан і поточну позицію. Вказівник на файл ідентифікує конкретний дисковий файл і використовується потоком для виконання операцій вводу та виводу. Для читання або запису файлу програма використовує вказівник на файл. Файлова змінна *file* створюється наступним оператором:

*FILE *file;*

Операція, за допомогою якої програма запитує доступ до файлу, має назву відкриття файлу. Звільнення файлу наприкінці роботи програми називається закриттям файлу.

Функція *fopen()* відкриває потік. Цією функцією управляють три основних параметри: ім'я файлу, який необхідно відкрити, вид застосування файлу й вказівник на файл, що повертає функція. Синтаксис: *FILE * fopen(char *filename, char *mode)*. Функція *fopen()* відкриває файл, іменований параметром *filename* і зв'язує його з відповідним потоком *stream*. Функція *fopen()* повертає як результат адресний вказівник, який ідентифікує потік *stream* у наступних операціях. Файл може бути відкритий у таких режимах (*mode*):

r відкриття файлу тільки для читання;

w створення файлу для запису;

a приєднання, відкриття для запису в кінець файлу або створення файлу для запису, якщо він не існує.

Якщо даний файл відкривається або створюється у текстовому режимі, можна додати символ *t* до значення параметра *mode* (*rt, w+t*). Для специфікації бінарного режиму можна до значення параметра *mode* додати символ *b* (*wb, a+b*). Значення, що повертає функція: при успішному завершенні *fopen()* повертає вказівник на відкритий потік *stream*. У випадку помилки функція повертає *NULL*.

Для закриття потоку використовується функція *fclose()*. Синтаксис: *int fclose(FILE *stream)*. Аргументом є вказівник на відкритий файл. Якщо файл успішно закритий, то функція *fclose()* повертає значення *0*. Функція *fclose()* зберігає у файл дані, які знаходяться у дисковому буфері, і виконує операцію системного рівня по закриттю файлу. Виклик *fclose()* звільняє блок управління файлом, що пов'язаний з потоком, і робить його доступним для повторного використання.

Крім основних функцій вводу та виводу, система буферизованого вводу/виводу містить функції *fprintf()* і *fscanf()*, які працюють із дисковими файлами. Якщо швидкість роботи програми або розмір створюваних файлів мають важливе значення, краще використовувати функції, які дозволяють читати й записувати блоки даних – *fread()* і *fwrite()*. Одним з найбільш важливих застосувань цих функцій є читання й запис даних типу масив або структура. Функції мають наступні прототипи:

*size_t fread(void *ptr, size_t size, size_t n, FILE *stream),*

де **ptr* – вказівник на область пам'яті, що одержує дані з файлу;

size – довжина кожного елемента в байтах;

n – визначає, скільки елементів довжиною *size* буде прочитано;

**stream* – файловий вказівник на раніше відкритий файл.

*size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream),*

де **ptr* – вказівник на інформацію, записану у файл.

4.3. Описання програми

На початку виконання програми створюються вказівники на файли **infile, *outfile* та перевіряється кількість параметрів, що передаються головній функції, і успішність

відкриття файлів для роботи. Копіювання даних виконується у циклі *while(feof(infile) == 0)* з використанням функції *getc(infile)* та *putc(ch, outfile)*. Перед завершенням роботи програми файли закриваються за допомогою функції *fclose()*.

4.4. Результати виконання програми

Результатом виконання програмного коду має бути копіювання даних вихідного файлу у новий, який буде створено. Імена вихідного та нового файлів передаються розробленій програмі як додаткові параметри командного рядка. На рисунку 7.1 наведено приклад виклику програми з додатковими параметрами.

4.5. Висновки по роботі

При виконанні комп'ютерного практикуму ми ознайомились з основними функціями, що призначені для роботи з файлами. Розроблено програму, що копіює файл будь-якого типу. В циклі копіювання реалізовано завершення процедури при зчитуванні ознаки кінця файлу – *EOF*. Імена файлів передаються програмі як додаткові параметри командного рядка.

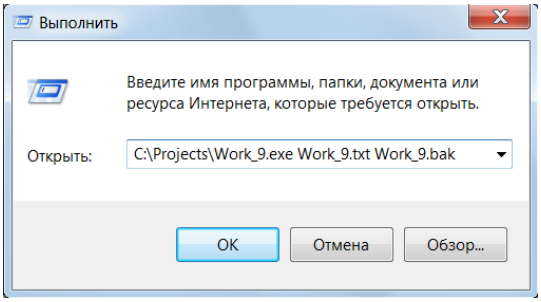

Рисунок 4.1 – Виклик програми з додатковими параметрами

4.6. Лістинг програми

```
// lab4.cpp: Defines the entry point for the console application.
#include "stdafx.h"
#include <stdio.h>
```

```
int main(int argc, char *argv[])
{
FILE *infile, *outfile;
char ch;
if(argc != 3)  {
        printf("Too few parameters...");
        return 1;
        }
if((infile = fopen(argv[1], "r+b")) == NULL)
        {
        printf("The source file was not opened");
        return 1;
        }
if((outfile = fopen(argv[2], "w+b")) == NULL)
        {
        printf("The destination file was not opened");
        return 1;
        }
while(feof(infile) == 0)
        {
        ch = getc(infile);
        if(feof(infile) == 0)        putc(ch, outfile);
        }
fclose(infile);
fclose(outfile);
return 0;
}
```

Контрольні запитання

1. Як створюється файлова змінна?
2. Перерахуйте операції над файлами.

3. Опишіть призначення функції *fopen()*.

4. Наведіть синтаксис функції *fopen()*.

5. Яке значення повертає функція *fopen()*?

6. В яких режимах може бути відкритий файл?

7. Опишіть призначення функції *fclose()*.

8. Опишіть призначення функцій *fprintf()* та *fscanf()*.

9. Чим відрізняється текстовий файл від бінарного.

Практичне заняття № 5:
«Списки»

5.1. Завдання

На прикладі заданої у завданні структури дослідити особливості створення однонаправлених і двонаправлених списків. Вивчити і реалізувати механізми додавання нових записів у список, пошуку записів у списку за певними полями, видалення записів зі списку та редагування знайдених записів, а також збереження всього списку у бінарному файлі та зчитування списку із файлу до пам’яті з відновленням всіх зв’язків.

Як приклад, розглянемо задачу розробки коду програми, що виконує Всі ці функції як для однонаправленого, так і двонаправленого списків.

В якості індивідуального завдання необхідно написати програму, що виконує вказаний перелік функцій, в якості структури даних використати варіанти завдання із Додатку В.

5.2. Теоретичні відомості

Структурою називається сукупність змінних, об’єднаних у єдине ціле, наприклад:

```
struct myStruct {
  int i, j, k;
  double x, y;
  char s[32];
};
```

Декларація змінної в такому випадку буде мати вигляд:

```
myStruct ss;
```

Але значно частіше для цього застосовують динамічне створення таких структур:

```
myStruct *pp;
pp = malloc(sizeof(myStruct));
```

В такий спосіб будуються списки або більш складні конструкції. В такому підході використовується той факт, що полями структури можуть бути вказівники на таку ж структуру, наприклад для одно- або двонаправленого списку:

```
struct data {
  // будь-які дані необхідні для конкретної задачі
};
struct listElem {
  data dat;          // дані в конкретному елементі списку
  listElem *next;   //вказівник на наступний елемент списку
  listElem *prev;   //вказівник на попередній елемент списку, якщо список
                    //двонаправлений
}
```

Однонаправлений список має тільки вказівник на наступний елемент списку, а двонаправлений має два вказівника на наступний і попередній. В однонаправленому списку можна рухатись лише від голови списку до хвоста, а в двонаправленому і від хвоста до голови.

```
listElem *head = 0; // декларований вказівник на перший елемент списку
listElem *curr = 0; // допоміжний вказівник, що використовується
                    // для руху по списку
```

Тепер можна будувати список із довільного числа елементів, зв’язаних між собою вказівниками.

```
void addElem(listElem *h) { // Додає новий елемент в голову списку
   listElem* t;
   If (!h) {
h = malloc(sizeof(listElem)); // Створення першого елементу списку
h->next = 0;
h -> prev=0;
   } else {
// додавання нового елементу перед “головою” списку
t = malloc(sizeof(listElem));
t->next = h;
t->prev = 0;
h ->prev = t;
h = t;
```

};

getData(h->dat); // функція заповнення структури даними

};

Видалення списку відбувається у зворотньому порядку.

Для доступу до списку необхідно, щоб був хоча б один вказівник на нього. У випадку однонаправленого списку це повинен бути обов’язково вказівник на “голову” списку, бо в однонаправленому списку неможливо перейти до попереднього елементу, а в двонаправленому хоча і варто дотримуватись цього ж правила, але не обов’язково, бо по такому списку можна рухатись в обох напрямках. Якщо завдяки помилкового коду раптом зникають всі вказівники на список – він стає недоступним, але продовжує займати пам’ять в купі, що приводить до досить складної для аналізу помилки, яка зветься “витік пам’яті”.

5.3. Опис програми

У програмі реалізовано створення двонаправленого списку, пошук елементів списку, редагування або видалення поточного елементу списку, сортування списку та виведення на екран всього списку. Керування програмою відбувається за допомогою меню, яке може бути розширено за необхідності іншими функціями. Всі дії програми реалізовані в формі функцій, що дозволяє зручно лагодити і тестувати програму.

5.4. Результати виконання програми

На рисунку 5.1 наведено приклад виклику програми.

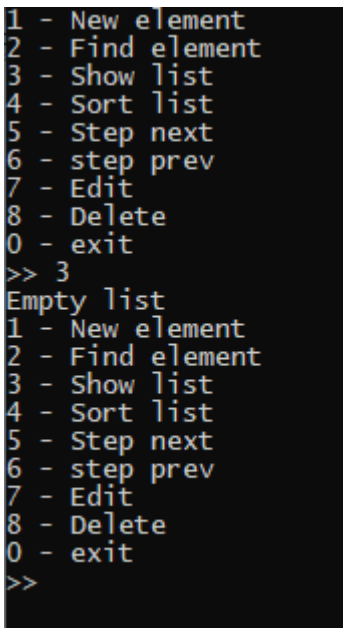

Рисунок 5.1 – Результат виконання програми

5.5. Висновки по роботі

У результаті виконання комп'ютерного практикуму досліджено особливості побудови таких динамічних структур, як списки, які часто зустрічаються на практиці.

Для прикладу було розроблено код програми, що дозволяє створювати списки, шукати, редагувати і видаляти елементи списку а також сортувати весь список за певними критеріями.

5.6. Лістинг програми

```
// lab5.cpp: Defines the entry point for the console application.
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
// List of structures program
// Global declarations
struct myData {
    int i, j;
    char nam[32];
    double waight;
};

struct listElem {
    myData data;
    listElem *next;
    listElem *prev;
};

listElem *head=0;
listElem *curr=0;

//==================================
```

```
int fromMenu() {
   int res = -1;
   char a[20];
   char menu[] = "1 - New element\n2 - Find element\n3 - Show list\n4 - Sort list\n5 - Step
next\n6 - step prev\n7 - Edit\n8 - Delete\n0 - exit\n";
   while (res<0) {
      memset(a,0,20);
      printf(menu);
      scanf("%s",a);
      if (strlen(a)==0) res = -1;
      res = atoi(a);
   }
   return res;
};

//=====================================

void newElem(listElem *h){

};

//=====================================

listElem *findElem(listElem *h){
      return 0;
};

//=====================================

void sortList(listElem *h){
   if (!h) {printf("Empty list"); return;};
```

```
}

//====================================

void showElem(listElem *c){
    if (!c) {printf("No element to show.\n"); return;};
};

//====================================

void editElem(listElem *c){
    if (!c) {printf("No element to edit.\n"); return;};

}

//====================================

void deleteElem(listElem *c){
    if (!c) {printf("No element to delete.\n"); return;};

}

//====================================

void showList(listElem *h){
    listElem *tmp;
    if (h==0){ printf("Empty list\n"); return;};
    tmp = h;
    while (tmp) {
        showElem(tmp);
        tmp=tmp->next;
    }
```

```
};

//==================================

int main() {
    int ctrlflg=1;
    do {
        ctrlflg = fromMenu();
        switch (ctrlflg) {
            case 1:
                newElem(head);
                break;
            case 2:
                curr = findElem(head);
                if (!curr) printf("Not found\n");
                break;
            case 3:
                showList(head);
                break;
            case 4:
                sortList(head);
                break;
            case 5:
                if ((curr)&&(curr->next)){
                    curr = curr->next;
                }
                showElem(curr);
                break;
            case 6:
                if ((curr)&&(curr->prev)){
                    curr = curr->prev;
                }
```

```
                showElem(curr);
                break;
            case 7:
                editElem(curr);
                break;
            case 8:
                deleteElem(curr);
                break;
        }
    } while (ctrlflg);
    printf("Exiting!");
    return 0;
}
```

Контрольні запитання

1. В якій частині пам’яті розташовуються списки?

2. Чим відрізняється двонаправлений список від однонаправленого?

3. Як обчислюється розмір пам’яті, що необхідний для розміщення одного елементу списку?

4. Який декларований статичний вказівник обов’язково повинен створюватись для роботи зі списками?

5. Що трапиться, якщо не залишиться жодного вказівника на список?

6. Що таке “витік пам’яті” і до чого він може призвести?

7. Що трапиться, якщо в хвостовому елементі списку вказівник *next* не буде мати значення *0* або *NULL*?

8. Що трапиться, якщо вказівник на “голову” однонаправленого списку буде зміщений на деякий інший запис у списку без збереження?

9. Чи має сенс при збереженні списків у бінарний файл зберігати і значення вказівників *next* і *prev*?

10. Яким чином треба читати список з бінарного файлу, щоб зберегти порядок розташування записів у списку?

Практичне заняття № 6:
«Розв'язання системи лінійних алгебраїчних рівнянь»

6.1. Завдання

Ознайомитись з методами розв'язання системи лінійних алгебраїчних рівнянь (СЛАР). Розглянути два методи метод Гаусса і метод Крамера. Побудувати для методу Гаусса програмну реалізацію, яка дозволяє вирішувати системи довільного порядку і визначати сумісність системи і її визначеність.

Як індивідуальне завдання необхідно написати програму вирішення довільної СЛАР методом Гаусса з вибором найбільшого коефіцієнту при головному елементі.

6.2. Теоретичні відомості

СЛАР називається сукупність рівнянь вигляду:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots a_{2n}x_n = b_2 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots a_{nn}x_n = b_n \end{cases}$$

Рішенням такої системи називається сукупність значень невідомих $x_1, x_2, \dots, x_n$, при яких одночасно виконуються всі рівняння.

Не всі системи рівнянь мають єдине рішення і не всі мають рішення взагалі. Наприклад система:

$$\begin{cases} x_1 + x_2 = 10 \\ x_1 + x_2 = 5 \end{cases}$$

Не має жодного рішення бо при будь-яких значеннях невідомих почленне віднімання другого рівняння від першого приведе до рівняння $0 = 5$, що не є вірним. Такі системи рівнянь називаються несумісними.

Якщо система сумісна, то вона може бути визначеною, якщо має єдиний розв'язок, або невизначеною, якщо розв'язків безліч. Члени $b_1, b_2, \dots b_n$ називаються вільними членами.

Є кілька відомих методів розв'язування СЛАР, але в цьому практикумі ми приділяємо увагу лише методу Гаусса.

В нашому практикумі ми матимемо справу тільки з квадратними матрицями, де кількість рівнянь точно рівна кількості невідомих

Метод Гаусса спирається на дві важливі властивості:

1. Система має одне і теж рішення, якщо в будь-якому рівнянні помножити всі

члени при невідомих і вільний член на одне і те ж число.

2. Система має одне і те ж рішення при заміні будь-якого рівняння на зважену суму його з будь-яким іншим рівнянням з системи.

Тобто, завжди можна замінити *j*-те рівняння системи

$$a_{1j}x_1 + a_{2j}x_2 + \dots a_{nj}x_j$$

на будь яке типу

$$(a_{1j}*c_1 + a_{1k}*c_2)x_1 + (a_{2j}*c_1 + a_{2k}*c_2)x_2 + \dots + (a_{nj}*c_1 + a_{nk}*c_2)\, xn = b_j*c_1 + b_k*c_2$$

при будь-яких множниках c_1 і c_2.

Метод Гаусса складається з двох частин – прямого проходу і зворотнього.

На прямому проході система приводиться до ступінчастого вигляду, при якому всі елементи системи під головною діагоналлю є нульовими а на діагоналі ненульові. Це робиться наступним чином:

1. Вибираємо рівняння з найбільшим за абсолютною величиною коефіцієнтом при першій невідомій a_{t1}, де *t* значить номер рівняння;
2. Переносимо це рівнянні в перший рядок, тобто коефіцієнти цього рівняння стають a_{1i} $i = 1, 2, \dots, n$; і b_t стає b_1; На цьому кроці необхідно пам'ятати, що всі переноси виконуються з усім рівнянням разом з вільним членом;
3. Користуємось властивістю 1 і ділимо всі коефіцієнти на перший а також і вільний член, щоб отримати рівняння вигляду $x_1+a_{12}/a_{11}x_2+\dots+a_{1n}/a_{11}x_n=b_1/a_{11}$;
4. Користуємось властивістю 2 для того, щоб у всіх рівняннях починаючи з 2-го зробити коефіцієнти при x_1 рівними *0*: для *k*-го ($k = 2,...n$) рядка це має вигляд: $0 + (a_{12}/a_{11}*a_{k1}-a_{k2})x_2 + \dots + (a_{1n}/a_{11}*a_{k1}-a_{kn})x_n = (b1/a_{11}*a_{k1}-b_k)$;
5. Залишаємо перше рівняння незмінним і повторюємо кроки з 1 по 4 для другого і наступних;
6. В результаті ми отримуємо ступінчасту систему:

$$\begin{cases} x_1 + c_{12}x_2 + c_{13}x_3 + \dots + c_{1n}x_n = b_1 \\ 0 + x_2 + c_{13}x_3 + \dots + c_{2n}x_n = b_2 \\ 0 + 0 + x_3 + \dots + c_{3n}x_n = b_3 \\ \dots\dots\dots\dots\dots\dots\dots \\ 0 + 0 + 0 + \dots + x_n = b_n \end{cases}$$

Тепер переходимо до зворотнього ходу:

Очевидно, що останнє рівняння вже є рішенням для невідомої x_n. Тепер для рівняння *n-1* маємо вираз $c_{n-1,n-1}x_{n-1} + c_{n-1,n}x_n = d_{n-1}$. Підставляємо сюди значення x_n і обчислюємо x_{n-1}. Такик чином, послідовно підстановками визначаємо всі значення невідомих.

Може трапитись, що на деякому кроці прямого ходу не знайдеться ненульового

значення коефіцієнту для певного невідомого, що значитиме, що система як мінімум невизначена і єдиного рішення не має. На цьому етапі можна зупинити програму і дати сповіщення про неможливість отримати єдине рішення.

Саме тут ми не будемо розбирати питання яким чином відрізняються несумісні і невизначені системи.

6.3. Опис програми

При розробці коду програми реалізовано метод Гаусса для системи з довільною кількістю невідомих за умови, що кількість рівнянь рівна кількості невідомих бо саме ця умова є необхідною для існування єдиного рішення системи.

Було обрано варіант методу Гаусса з пошуком максимального коефіцієнта, а не просто ненульового тому що такий підхід робить метод більш стійким.

Результати були виведені на екран.

6.4. Результати виконання програми

Результат виконання програми показано на рисунку 6.1.

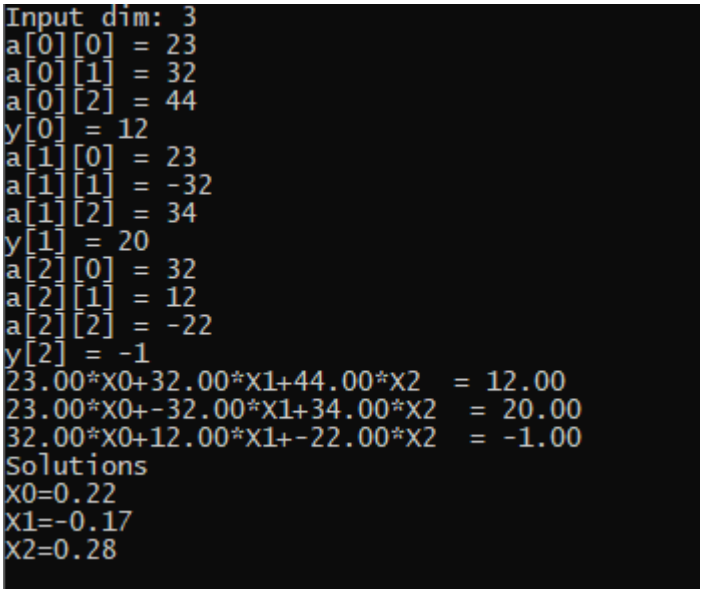

Рисунок 6.1 – Результат виконання програми

6.5. Висновки по роботі

У результаті проведеного практикуму була створена програма обчислення системи лінійних алгебраїчних рівнянь (СЛАР) і знаходження коренів системи за умови існування єдиного рішення.

6.6. Лістинг програми

```
// lab6.cpp: Defines the entry point for the console application.
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <math.h>
#include <string.h>

//=====================================

int iask(char *q) {
	char tmp[8];
	int res;
	do {
		printf("%s",q); scanf("%s",tmp);
		res = atoi(tmp);
	} while (res==0);
	return res;
}

//=====================================

double dask(char *q) {
	char tmp[16];
	char *err;
	double res;
	do {
		printf("%s",q); scanf("%s", tmp);
		res=strtod(tmp,&err);
	} while (*err!=0);
	return res;
}

//=====================================

void prn(int d, double **a, double *y){
	char c = ' ';
	for (int i=0;i<d;i++){
		for (int j=0;j<d;j++){
			c=(j==d-1)?' ':'+';
			printf("%4.2f*X%d%c",a[i][j],j,c);
```

```
            }
            printf(" = %4.2f\n",y[i]);
        }
}

//=====================================

double * gauss(double **a, double *y, int n)
{
  double *x, max;
  int k, index;
  const double eps = DBL_EPSILON;   // точність
  x = new double[n];
  k = 0;
  while (k < n)
  {
    max = abs(a[k][k]);
    index = k;
    for (int i = k + 1; i < n; i++)
    {
      if (abs(a[i][k]) > max)
      {
        max = abs(a[i][k]);
        index = i;
      }
    }
    if (max < eps)
    {
      printf("Null column\n");
      return 0;
    }
    for (int j = 0; j < n; j++)
    {
      double temp = a[k][j];
      a[k][j] = a[index][j];
      a[index][j] = temp;
    }
    double temp = y[k];
    y[k] = y[index];
    y[index] = temp;
    for (int i = k; i < n; i++)
```

```
      {
        double temp = a[i][k];
        if (abs(temp) < eps) continue; // коеффіцієнт <eps == 0
        for (int j = 0; j < n; j++)
          a[i][j] = a[i][j] / temp;
        y[i] = y[i] / temp;
        if (i == k)   continue; // не віднімати рівняння саме від себе
        for (int j = 0; j < n; j++)
          a[i][j] = a[i][j] - a[k][j];
        y[i] = y[i] - y[k];
      }
      k++;
    }
    // зворотня підствновка
    for (k = n - 1; k >= 0; k--)
    {
      x[k] = y[k];
      for (int i = 0; i < k; i++)
        y[i] = y[i] - a[i][k] * x[k];
    }
    return x;
}

//=====================================

int main() {
      int dim;
      char qq[40];
      double **a, *x, *y;

      dim = iask("Input dim: ");

//      a = new double*[dim];
      a = (double **)calloc(dim,sizeof(double*));
      y = (double*) calloc(dim,sizeof(double));
      x = (double*) calloc(dim,sizeof(double));

      for (int i=0;i<dim;i++){
            a[i] = (double*) calloc(dim,sizeof(double));
            x[i] = 0;
            for (int j=0;j < dim;j++){
```

```
            sprintf(qq,"a[%d][%d] = ",i,j);
            a[i][j] = dask(qq);
        }
        sprintf(qq,"y[%d] = ",i);
        y[i] = dask(qq);
    }
    prn(dim,a,y);
    x = gauss(a,y,dim);
    if (!x)
        printf("No solutions\n");
    else {
        printf("Solutions\n");
        for (int i=0;i<dim;i++) printf("X%d=%4.2f\n",i,x[i]);
    }
    return 0;
}
}
```

Контрольні запитання

1. У чому полягають головні кроки методу Гаусса?
2. Назвіть інші методи рішення СЛАР крім методу Гаусса.
3. Чому проводиться пошук максимального коефіцієнта при поточному невідомому, а не ненульовий?
4. Наведіть приклад несумісної СЛАР.
5. Наведіть приклад невизначеної системи.
6. Чому в цьому випадку краще використовувати двовимірний масив, організований як масив вказівників на одновимірні масиви дійсних чисел?

Список використаних джерел (посилань)

1. Б. Керниган, Д. Ритчи. Язык программирования Си. М.: Финансы и статистика, 1992. С. 272.

2. Б. Страуструп. Язык программирования C++. М.: Невский Диалект Издательство Бином, 1999. С. 991.

3. *K.N. King.* C Programming: A Modern Approach, 2nd Edition. New York - London, W.W. Norton & Company, 2008, P.661.

4. *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein,* Introduction to Algorithms, 3rd Edition, The MIT Press, 2009, P. 1320.

Додаток А. Вимоги щодо оформлення робіт

При оформленні протоколів робіт необхідно дотримуватись певних вимог. Рекомендується роздруковувати (або оформити рукописно) звіти на папері формату *А4* зі звичайними полями (верхнє, нижнє – *2 см*, праве – *1,5 см*, ліве – *3 см*), шрифтом *Times New Roman* розміром *14* пунктів, об'ємом *4-5* аркушів.

Структурно звіт з комп'ютерного практикуму повинен складатися із наступних розділів:

- завдання до роботи;
- теоретичні відомості;
- результати роботи;
- висновки по роботі;
- програмний код (додаток).

Звіт повинен вміщувати лише відредаговані матеріали. Зразок титульного аркуша приводиться нижче в Додатку Б. В завданні до роботи стисло пояснюються її мета та очікувані результати.

Теоретичні відомості повинні містити алгоритм роботи програми та її блок-схему, відомості про використані/розроблені функції.

В якості результатів роботи надаються копії екранів з вихідними даними.

У висновках аналізуються отримані результати.

В додатку наводиться програмний код, що пояснює деякі спеціальні аспекти роботи, а також та частина коду, що на погляд студента, є суттєво важливою.

Програмний код повинен надаватись лише в формі лістингу *.c або *.cpp добре коментований, намагатися уникати використання в коді кирилиці, краще користуватись транслітом.

В заголовному багаторядковому коментарі до коду повинно бути вказано ПІБ студенту, номер групи, номер і назва лабораторної роботи, якщо є варіант, то номер варіанту.

Для побудови блок-схеми програми використовуються наступні (Рисунок Д.А.1) графічні символи:

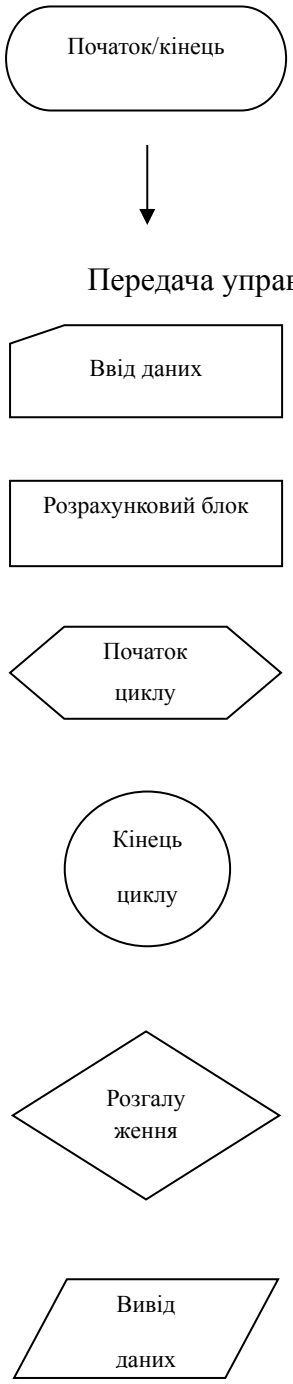

Рисунок Д.А.1 – Елементи блок-схеми

Додаток Б. Титульний аркуш (зразок)

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ПРОГРАМУВАННЯ АЛГОРИТМІЧНИХ СТРУКТУР

ЗВІТ ДО

КОМП’ЮТЕРНОГО ПРАКТИКУМУ № ___

«_____»
(Тема роботи)

Варіант № ___

Дата «___» _____ 20___

Виконав: студент ___ курсу

гр. ___-___

_____
(П.І.Б.)

Оцінка _____

Дата «___» _____ 20___

Перевірив: _____

_____
(П.І.Б., підпис)

Київ – 20___

Додаток В. Індивідуальні завдання

Практичне заняття № 1: «Динамічні масиви»
Вхідні дані та завдання дивись стор. 5.

Практичне заняття № 2: «Методи сортування масивів»
Вхідні дані та завдання дивись стор. 15.

Практичне заняття № 3: «Структури. Операції над масивами структур»
Завдання див на стор. 23. Індивідуальні структури для виконання завдань вибираються за номером студенту у списку групи.

1. Структура «Автосервіс»: реєстраційний номер автомобіля, марка, пробіг, майстер, який виконав ремонт, сума ремонту.
2. Структура «Співробітник»: прізвище, ім’я, по батькові; посада; рік народження; заробітна плата.
3. Структура «Держава»: назва; столиця; чисельність населення; займана площа.
4. Структура «Людина»: прізвище, ім’я, по батькові; домашня адреса; номер телефону; вік.
5. Структура «Читач»: прізвище, ім’я, по батькові, номер читацького квитка, назва книги, термін повернення.
6. Структура «Школяр»: прізвище, ім’я, по батькові; клас; номер телефону; оцінки з предметів (математика, фізика, іноземна мова, література).
7. Структура «Студент»: прізвище, ім’я, по батькові; домашня адреса; група; рейтинг. ‘
8. Структура «Покупець»: прізвище, ім’я, по батькові; домашня адреса; номер телефону; номер кредитної картки.
9. Структура «Пацієнт»: прізвище, ім’я, по батькові; домашня адреса; номер медичної карти; номер страхового поліса.
10. Структура «Інформація»: носій; об’єм; назва; автор.
11. Структура «Клієнт банку»: прізвище, ім’я, по батькові, номер рахунку, сума на рахунку, дата останнього зміни.
12. Структура «Склад»: найменування товару, ціна, кількість, відсоток торгової надбавки.
13. Структура «Авіарейси»: номер рейсу, пункт призначення, час вильоту, дата вильоту, вартість квитка.
14. Структура «Вокзал»: номер поїзда, пункт призначення, дні проходження, час прибуття, час стоянки.
15. Структура «Кінотеатр»: назва кінофільму, сеанс, вартість квитка, кількість глядачів.
16. Структура «Файл»: назва, розмір, дата створення, тип.

17. Структура «Підприємство»: назва, податковий номер, юридична адреса.

18. Структура «Автовокзал»: номер автобуса, пункт призначення, дні проходження, час прибуття, час стоянки.

19. Структура «Митниця»: марка автомобіля, номер автомобіля, дата перетину кордону.

20. Структура «Ресторан»: номер столу, офіціант, кількість місць.

21. Структура «Магазин»: назва, адреса, час роботи, тип товару.

22. Структура «Перукарня»: назад, адреса, кількість майстрів, час роботи.

23. Структура «Річпорт»: номер судна, пункт призначення, дні проходження, час прибуття.

24. Структура «Телекомпанія": назва програми, час показу, тривалість, рейтинг.

25. Структура «Зоопарк»: номер вольєра, назва тварини, корм.

Практичне заняття № 4: «Файли. Створення, запис, читання та видалення»

Вхідні дані та завдання дивись стор. 34.

Практичне заняття № 5: «Структури»

Завдання див на стор. 39. Індивідуальні структури для виконання завдань вибираються за номером студенту у списку групи.

1. Структура «Автосервіс»: реєстраційний номер автомобіля, марка, пробіг, майстер, який виконав ремонт, сума ремонту.

2. Структура «Співробітник»: прізвище, ім’я, по батькові; посада; рік народження; заробітна плата.

3. Структура «Держава»: назва; столиця; чисельність населення; займана площа.

4. Структура «Людина»: прізвище, ім’я, по батькові; домашня адреса; номер телефону; вік.

5. Структура «Читач»: прізвище, ім’я, по батькові, номер читацького квитка, назва книги, термін повернення.

6. Структура «Школяр»: прізвище, ім’я, по батькові; клас; номер телефону; оцінки з предметів (математика, фізика, іноземна мова, література).

7. Структура «Студент»: прізвище, ім’я, по батькові; домашня адреса; група; рейтинг. ‘

8. Структура «Покупець»: прізвище, ім’я, по батькові; домашня адреса; номер телефону; номер кредитної картки.

9. Структура «Пацієнт»: прізвище, ім’я, по батькові; домашня адреса; номер медичної карти; номер страхового поліса.

10. Структура «Інформація»: носій; об’єм; назва; автор.

11. Структура «Клієнт банку»: прізвище, ім’я, по батькові, номер рахунку, сума на рахунку, дата останнього зміни.

12. Структура «Склад»: найменування товару, ціна, кількість, відсоток торгової надбавки.

13. Структура «Авіарейси»: номер рейсу, пункт призначення, час вильоту, дата вильоту, вартість квитка.

14. Структура «Вокзал»: номер поїзда, пункт призначення, дні проходження, час прибуття, час стоянки.

15. Структура «Кінотеатр»: назва кінофільму, сеанс, вартість квитка, кількість глядачів.

16. Структура «Файл»: назва, розмір, дата створення, тип.

17. Структура «Підприємство»: назва, податковий номер, юридична адреса.

18. Структура «Автовокзал»: номер автобуса, пункт призначення, дні проходження, час прибуття, час стоянки.

19. Структура «Митниця»: марка автомобіля, номер автомобіля, дата перетину кордону.

20. Структура «Ресторан»: номер столу, офіціант, кількість місць.

21. Структура «Магазин»: назва, адреса, час роботи, тип товару.

22. Структура «Перукарня»: назад, адреса, кількість майстрів, час роботи.

23. Структура «Річпорт»: номер судна, пункт призначення, дні проходження, час прибуття.

24. Структура «Телекомпанія": назва програми, час показу, тривалість, рейтинг.

25. Структура «Зоопарк»: номер вольєра, назва тварини, корм.

Практичне заняття № 6: «Розв’язання системи лінійних алгебраїчних рівнянь»

Вхідні дані та завдання дивись стор. 47.