ЗМІСТ

|  |  |
| --- | --- |
| ВСТУП |  |
| 1 ТЕОРЕТИЧНА ЧАСТИНА |  |
| 1.1 Покажчики |  |
| 1.2 Арифметика покажчиків |  |
| 1.2.1 Операції інкремента і декремента |  |
| 1.2.2 Додавання й віднімання цілого числа до покажчика |  |
| 1.2.3 Різниця однотипних покажчиків |  |
| 1.3 Динамічна пам’ять |  |
| 1.4 Динамічні одновимірні масиви  1.5 Динамічні двовимірні масиви (матриці) |  |
| 2 ПРАКТИЧНА ЧАСТИНА |  |
| 2.1 Створення програми обчислення математичних виразів |  |
| 2.2 Створення програми реалізації циклічних обчислювальних процесів |  |
| 2.3 Створення програми обробки одновимірних масивів величин |  |
| 2.4 Створення програми обробки двовимірних масивів величин |  |
| 2.5 Створення програми обробки рядків |  |
| 2.6 Створення програми обробки структур |  |
| 2.7 Створення програми реалізації функцій |  |
| 2.8 Створення програми обробки файлів |  |
| ВИСНОВКИ |  |
| СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ |  |

ВСТУП

С – проста й вишукана мова програмування, що була розроблена співробітником фірми Bell Labs Денісом Рітчі спільно з Кеном Томпсоном у 1972 році. Спочатку її використовували як інструментальну мову операційної системи UNIX. Однак вона виявилась настільки вдалою, що досить швидко стала однією з найпопулярніших мов системного програмування. Швидке зростання популярності привело до розробки в 1983 році стандарту мови С після чого її стали використовувати повсюдно.

Мова С має ряд достоїнств, які роблять її універсальним інструментом програміста:

- простота;

- відносно висока швидкодія та відносно малий розмір програм, написаних на С;

- високий рівень мобільності програм, написаних на С;

- суміщення можливостей мови низького рівня зі зручністю мови програмування високого рівня.

Все перелічене вище робить мову С зручним засобом для розробки програм різноманітного призначення. Однак програмування – це галузь знань, що досить швидко розвивається, виникають нові концепції та методи програмування. Нові методи вимагають нових можливостей, нових засобів розробки. Так розвиток концепції об’єктно-орієнтованого програмування привів до модифікації мови програмування С та розробки на її основі мови «С з класами», яка в подальшому розвинулась у мову програмування С++.

C++ — мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом в AT&T Bell Laboratories у 1979. Базується на мові С. У 1990-х роках С++ стала однією з найуживаніших мов програмування загального призначення.

Назву мові "С++" придумав Рік Масситті, яка походить від оператора мови С інкремента "++" (збільшення значення змінної на одиницю). У цей час також був поширений спосіб присвоєння нових імен комп'ютерним програмам, що полягав у додаванні до назви символу "+" для позначення її нових модифікацій. Згідно з задумом Б'ярна Страуструпа, "ця назва вказує на еволюційну природу змін мови C". Виразом "С+" називали попередню мову програмування, зовсім не пов'язану з мовою С++. Багато зі фахівців можуть зауважити, що введення назви мови С++ не змінює самої мови програмування С, тому найточнішим іменем мало б бути " С+1".

Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм таких як відеоігри. С++ суттєво вплинула на інші, популярні сьогодні, мови програмування: С# та Java. Також програми, складені мовою C++, є мобільними, тобто мо­жуть бути виконані на комп’ютерах різних виробників і в різ­них операційних системах, завдяки чому C++ є особливо популярною.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Покажчики

Пам’ять комп’ютера являє собою сукупність комірок для зберігання інформації, кожна з яких має власний номер. Ці номери називаються адресами. Розмір однієї комірки пам’яті становить 1 байт. При оголошенні змінної в пам’яті виділяється ділянка обсягом, відповідним до типу змінної, наприклад 4 байти для типу int. Ця ділянка пам’яті пов’язується з іменем змінної. Можна сказати, що адреса змінної – це адреса першої комірки цієї ділянки. Існують спеціальні змінні – покажчики, в яких можна зберігати адреси комірок пам’яті, тобто адреси змінних.

Слід звернути увагу на те, що адреса і значення змінної – це зовсім різні поняття. Зазвичай адреси змінних не є важливими, найголовніше – значення цих змінних. Покажчики використовуються для прямого доступу до даних у пам’яті.

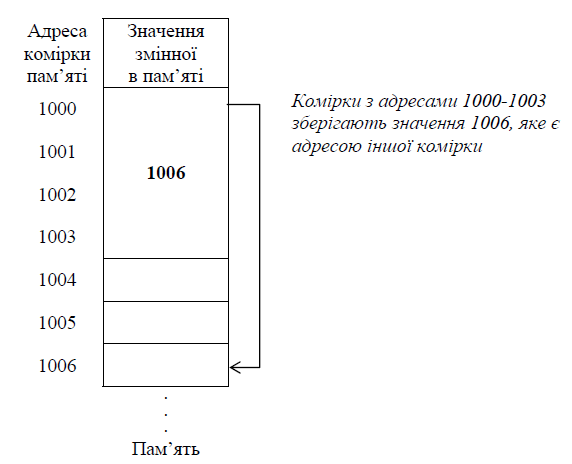


Рисунок 1.1 – Схема заповнення комірок пам’яті значеннями

Покажчик оголошується за допомогою \*:

<тип> \*<ім’я>;

Тут тип – це базовий тип покажчика, котрим може бути який завгодно тип. Ім’я є ідентифікатором змінної-покажчика. Слід звернути увагу на те, що тип – це

не тип покажчика, а тип даних, адресу яких буде записано у покажчику.

Наприклад, покажчик на пам’ять, в якій зберігатиметься ціле число:

int \*A;

Покажчик на дійсне число:

float \*В;

При цьому змінна А – це адреса ділянки пам’яті, в якій розташоване ціле число, B – адреса ділянки пам’яті, в якій розташоване дійсне число. Обидві змінні – А та B – є покажчиками, тобто їхніми значеннями є адреси. Окрім того, змінні типу int та float займають у пам’яті однакову кількість байтів. Але насправді змінні А та В мають різний тип: А – покажчик на ціле, B – покажчик на дійсне.

Базовий тип покажчика визначає тип об’єкта, адресу якого містить покажчик. Фактично покажчик будь-якого типу може посилатися на яке завгодно місце в пам’яті. Однак операції, які можуть виконуватися з покажчиком, суттєво залежать від його типу. Наприклад, якщо оголошено покажчик типу int \*, компілятор припускає, що адреса, на яку він посилається, містить змінну типу int, хоча це може бути і не так. Отже, при оголошенні покажчика слід переконатися, що його тип є сумісним з типом об’єкта, на який він буде посилатися.

Покажчикові можна присвоїти лише адресу змінної відповідного типу, оскільки С++ не виконує автоматичного перетворення типів покажчиків. Тому такі команди є помилковими:

A = B; // Не можна переприсвоювати один одному покажчики на різні типи;

А = 10; // не можна присвоювати числа покажчикам.

У мові С++ визначено дві операції для роботи з покажчиками:

1) \* – розадресація, чи розіменування покажчика;

2) & – адресація, тобто отримання адреси змінної.

Для прикладу у програмі оголосимо цілу змінну Х зі значенням 7:

int Х = 7;

тоді, щоб її адресу записати до покажчика А, слід записати команду:

А = &Х; // Значенням змінної А є адреса змінної Х.

Якщо пам’ять, у якій зберігається змінна Х, починається з комірки під номером 2000, тоді змінній А після цієї команди буде присвоєно значення 2000. Тепер до значення 7, яке зберігається у змінній X, можна звернутися не лише за ім’ям X, але й через покажчик. При цьому слід застосувати операцію розадресації покажчика (вона позначається символом “\*”), наприклад: \*А. Цей запис означає: “значення, яке зберігається за адресою, записаною у змінній A”.

Щоб збільшити на 2 оголошену раніше змінну X, можна записати так:

X = X + 2; /\* За ім’ям X комп’ютер визначить адресу ділянки пам’яті, в якій вона розташована, візьме значення, що зберігається у пам’яті за цією адресою, збільшить його на 2 і запише назад у ту ж саму ділянку пам’яті \*/

чи так:

\*A = (\*A) + 2; /\* Комп’ютер з ділянки пам’яті за адресою, записаною у змінній A, візьме значення, яке зберігається в ній, збільшить це значення на 2 і запише в ту ж саму ділянку \*/

У першому разі відбувається звертання до змінної з ім’ям X, у другому – до змінної через її адресу, яку записано у змінній-покажчику A. Звернімо увагу на те, що “зірочки” при оголошенні покажчика та операції розадресації – це зовсім різні речі, які лише позначаються однаковим символом.

Покажчик, який не вказує на жодне значення, називається порожнім, чи нульовим. Такий покажчик має значення 0 чи NULL. Покажчики використовуються при роботі з динамічними змінними і при передаванні параметрів до функцій.

* 1. Арифметика покажчиків

У С++ для покажчиків визначено наступні арифметичні операції:

1.2.1 Операції інкремента і декремента

Операції інкремента (++) і декремента (--), тобто збільшення чи зменшення покажчика на 1. При цьому адреса, яка зберігається у цьому покажчику, збільшується на розмір базового типу в байтах. Наприклад, покажчик ptr вказує на ціле число:

int \*ptr;

Команда

ptr++;

збільшує його значення на 4, оскільки розмір змінної типа int дорівнює 4.

Припустімо, що число 10, подане на рис. 1.2, зберігається за адресою 1000, тобто значення покажчика ptr є 1000. Після команди ptr++; значення покажчика буде 1004 і ptr вказуватиме на число 20.



Рисунок 1.2 – Операція інкремента покажчика

Зменшення покажчика на 1 означає віднімання розміру в байтах базового типу від адреси, яка зберігається у покажчику.

1.2.2 Додавання й віднімання цілого числа до покажчика

При цьому адреса, яка зберігається у покажчикові, змінюється кратно до розміру даних, на які вказує покажчик. Для оголошеного вище покажчика ptr команда

ptr += 3;

збільшить адресу, яка міститься у ptr, на 4\*3=12 байт, і ptr вказуватиме на число 40 (рис. 1.3).

Загальна формула, яка обчислює значення покажчика при додаванні (відніманні) цілого числа N:

<Нова адреса> = <стара адреса> +/- <розмір базового типа>\*N;



Рисунок 1.3 – Арифметика покажчиків

1.2.3 Різниця однотипних покажчиків.

При відніманні покажчиків обчислюється кількість комірок між адресами, які містяться у покажчиках. Нехай ptr та pq – покажчики на цілі числа (див. рис. 1.3). Різниця pq – ptr є цілим числом і дорівнює 2:

int k = pq - ptr; // k = 2

Загальна формула, яка обчислює значення кількості комірок при відніманні однотипних покажчиків:

<Кількість комірок>=<перший покажчик>–<другий покажчик>/<розмір типу>

Приклад:

double z; // Змінна z займає у пам’яті 8 байтів;

double \*p = &z; /\* p – покажчик на тип double, йому можна присвоїти адресу змінної z типу double;\*/

p++; // значення p збільшилось на 8 байтів;

p--; // значення p зменшилося на 8 байтів;

p+=5; // значення p збільшилося на 8\*5=40 байтів.

Покажчики можна порівнювати (на рівність/нерівність та більше/менше). При цьому порівнюються адреси, які містяться у покажчиках. Два покажчики вважаються за рівні, якщо вони містять однакові адреси (вказують на одну комірку пам’яті). За більший вважається той покажчик, який містить більшу адресу. Наприклад:

if(p < q) printf("p посилається на меншу за q адресу\n");

Приклад порівняння покажчика ptr з ненульовим значенням:

if(ptr != NULL) ShowMessage("Ненульовий покажчик");

* 1. Динамічна пам’ять

Окрім звичайної пам’яті, в якій автоматично розміщуються змінні за їхнього оголошення, існує ще й динамічна пам’ять, в якій змінні можуть розміщуватися динамічно. Це означає, що пам’ять виділяється під час виконання програми, й лише тоді, коли у програмі зустрінеться спеціальна інструкція. Основна потреба у динамічному виділенні пам’яті виникає, коли розмір або кількість даних заздалегідь є невідомі, а визначаються в процесі виконання програми. У С++ існує кілька команд для виділення динамічної пам’яті. Найчастіше використовується оператор new, який у загальному вигляді записується як:

<тип> \*<ім’я\_покажчика> = new <тип>;

Приклади:

1) int \*p = new int;

Тут виділяється місце в пам’яті під ціле число й адреса цієї ділянки пам’яті записується у змінну-покажчик p. Звернутися до цього числа можна буде через покажчик на нього: \*p = 2;

2) int \*p = new int (5);

Ця команда не лише виділить місце у пам’яті під ціле число, а й запише в цю ділянку пам’яті значення 5. Адреса першої комірки виділеної ділянки пам’яті присвоюється змінній-покажчику p. Звернутися до числа, на яке вказує p, можна аналогічно до попереднього прикладу. Приміром, щоб збільшити таке число на 2, слід написати: (\*p) += 2;

3) int \*p = new int [5];

У цьому разі виділяється пам’ять під 5 цілих чисел, тобто фактично створюється так званий динамічний масив з 5-ти елементів. Звернутися до кожного з цих чисел можна за його номером: p[0], p[1] і т. д. або через покажчик: \*p − те ж саме, що p[0]; \*(p+1) − те ж саме, що p[1] і т. д. Пам’ять, яку було виділено динамічно, автоматично не звільнюється, тому програміст повинен обов’язково звільнити її самостійно за допомогою спеціальної команди. При виділенні пам’яті за допомогою оператора new, для звільнення пам’яті використовується delete:

delete <покажчик>;

Наприклад:

delete a;

Якщо оператором new було виділено пам’ять під кілька значень водночас (як у розглянутому вище прикладі 3), використовується форма команди delete

delete []<покажчик>;

Квадратні дужки повинні бути порожніми, операційна система контролює кількість виділеної пам’яті і при звільненні їй відома потрібна кількість байтів.

Наприклад:

delete [] p;

Окрім операторів new та delete, існують функції, які перейшли до С++ з

С, але вони використовуються на практиці набагато рідше.

Функція

void \*malloc(size\_t size);

(від англ. “memory allocation” – виділення пам’яті) робить запит до ядра операційної системи щодо виділення ділянки пам’яті заданої кількості байтів. Єдиний аргумент цієї функції size − кількість байтів, яку треба виділити. Функція повертає покажчик на початок виділеної пам’яті. Якщо для розміщення заданої кількості байтів є недостатньо пам’яті функція malloc() повертає NULL. Вміст ділянки лишається незмінним, тобто там може залишитися “бруд”. Якщо аргумент size дорівнює 0, функція повертає NULL. Наприклад, команда

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

виділяє пам’ять під ціле число й адресу початку цієї ділянки пам’яті записує у покажчик а. Виділення пам’яті під 10 дійсних чисел за допомогою цієї функції:

float \*a = (float\*) malloc(sizeof(float)\*10);

Функція

void \* calloc(size\_t num, size\_t size);

виділяє блок пам’яті розміром num×size (під num елементів по size байтів кожен) і повертає покажчик на виділений блок. Кожен елемент виділеного блока ініціалізується нульовим значенням (на відміну від функції malloc). Функція calloc() повертає NULL, якщо не вистачає пам’яті для виділення нового блока, або якщо значення num чи size дорівнюють 0.Виділення пам’яті під 10 дійсних чисел за допомогою функції calloc():

float \*a = (float\*) calloc(10,sizeof(float));

Функція

void \*realloc(\*bl, size);

змінює розмір виділеного раніш блока пам’яті з адресою \*bl на новий обсяг, який становитиме size байтів. Якщо змінення відбулося успішно, функція realloc() повертає покажчик на виділену ділянку пам’яті, а інакше повертається NULL. Якщо \*bl є NULL, функція realloc() виконує такі самі дії, що й malloc(). Якщо size є 0, виділений за адресою \*bl блок пам’яті звільнюється – і функція повертає NULL. Для ілюстрації роботи функції realloc() наведемо приклад змінення розміру раніш виділеної пам’яті під одновимірний масив з 10-ти дійсних елементів до 20-ти елементів:

a = (float\*) realloc(a, 20);

Пам’ять, виділену за допомогою функцій malloc() і сalloc(), звільнюють за допомогою функції free():

void free(\*bl);

де \*bl – адреса початку виділеної раніше пам’яті, наприклад:

free(а);

Використання згаданих функцій разом з покажчиками надає можливість керувати розподілом динамічної пам’яті.

* 1. Динамічні одновимірні масиви

Відмінності динамічного масиву від звичайного полягають у тому, що:

- пам’ять під динамічний масив виділяється динамічно за допомогою вищерозглянутих функцій;

- кількість елементів динамічного масиву може бути задано змінною (але у програмі її неодмінно має бути визначено до виділення пам’яті під масив).

Синтаксис оголошення динамічного одновимірного масиву за допомогою оператора new є такий:

<базовий тип> \*<ім’я>=new <базовий тип>[<кількість елементів>];

Приклад оголошення дійсного динамічного масиву зі змінною кількістю елементів:

int N;

cin >> N;

float \*a=new float [N];

Тут кількість елементів масиву є змінною і вводиться з клавіатури перед оголошенням масиву. Звільнення пам’яті від цього масиву a матиме вигляд:

delete []a;

Синтаксис оголошення динамічного одновимірного масиву за допомогою функції malloc() є такий:

<тип> \*<ім’я> = (<тип>\*) malloc(sizeof(<тип>)\*<кільк. ел.>);

Приклад оголошення дійсного динамічного масиву зі змінною кількістю елементів за допомогою функції malloc():

int N;

cin>>N;

float \*a=(float \*)malloc(sizeof(float)\*N);

Приклад оголошення дійсного динамічного масиву зі змінною кількістю

елементів за допомогою функції calloc():

int N;

cin>>N;

float \*a=(float \*)calloc(sizeof(float), N);

Звільнення пам’яті з-під цього масиву a:

free (a);

* 1. Динамічні двовимірні масиви (матриці)

Двовимірний динамічний масив з m рядків і n стовпчиків займає в пам’яті сусідні m\*n комірок, тобто зберігається так само, як і одновимірний масив з m\*n елементів. При розміщенні елементи двовимірних масивів розташовуються в пам’яті підряд один за одним з першого до останнього без проміжків у послідовно зростаючих адресах пам’яті. Наприклад, дійсний масив 3×5 зберігається у пам’яті в такий спосіб (рис. 1.4):

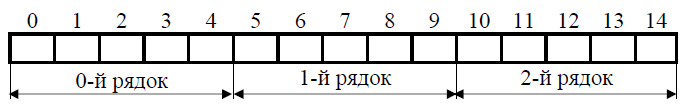


Рисунок 1.4 – Приклад збереження масиву у пам’яті комп’ютера

У такому масиві перші п’ять елементів належать до першого рядка, наступні п’ять – до другого і останні п’ять – до третього.

Нагадаємо, що a – покажчик на початок масиву, тобто на елемент a[0][0]. Щоб звернутися, наприклад, до елемента a[1][3], слід “перестрибнути” від початку масиву через 5 елементів нульового рядка й 3 елементи першого рядка, тобто написати: \*(a+1\*5+3). У загальному випадку до елемента a[i][j] можна звернутися в такий спосіб: \*(a+i\*5+j). Але цей спосіб роботи з двовимірним масивом є не надто зручний, тому що в програмі при звертанні до елемента масиву доводиться розадресовувати покажчик і обчислювати індекс елемента.

Оголосити дійсний динамічний масив 3×5 можна як одновимірний з 15-ти елементів:

float \*a=new float [3\*5];

чи то

float \*a=(float \*)сalloc(3\*5, sizeof(float));

Пам’ять від створеного в такий спосіб масиву очищується за допомогою операцій відповідно delete й free:

delete []a;

чи то

free(a);

Розглянемо інший спосіб роботи з динамічним двовимірним масивом. Для цього розмістимо в пам’яті машини матрицю 3×5 (рис. 1.5):

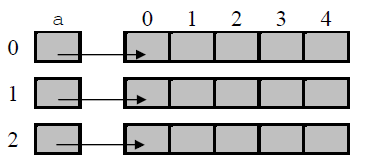


Рисунок 1.5 – Схема масиву 3х5

При цьому буде виділено пам’ять під кожний рядок матриці окремо, тобто буде утворено три різні одновимірні масиви. Адреси нульових елементів цих масивів зберігатимуться в допоміжному масиві a (пам’ять під нього слід виділити заздалегідь). Елементами цього масиву будуть адреси дійсних чисел, тому вони матимуть тип “покажчик на дійсне число”, тобто float∗. Нагадаємо, що загальний вигляд оголошення покажчика на динамічний масив є такий:

<тип елемента> \* <ім’я масиву>;

Тоді при оголошенні масиву а слід записати дві зірочки:

float \*\* a = new float\* [3]; /\* Оголошення й розміщення в пам’яті допоміжного масиву з 3-х елементів типу float\* \*/

for(int i=0; i<3; i++) a[i] = new float [5]; /\* У циклі виділяється пам’ять під 3 масиви по 5 елементів (рядки матриці) й адреси нульових елементів цих масивів записуються у відповідні елементи масиву a \*/

Після цього можна працювати з матрицею як зі звичайним двовимірним масивом, звертаючись до кожного елемента за його індексом, наведеним у квадратних дужках: a[i][j], – що є більш природним і зручним, аніж попередній спосіб.

Аналогічне є оголошення за допомогою сalloc():

float ∗∗ a = (float∗\*) calloc (3, sizeof(float\*));

for (int i=0; i<3; i++)

a[i] = (float\*) calloc (5, sizeof(float));

2 ПРАКТИЧНА ЧАСТИНА

2.1 Створення програми обчислення математичних виразів

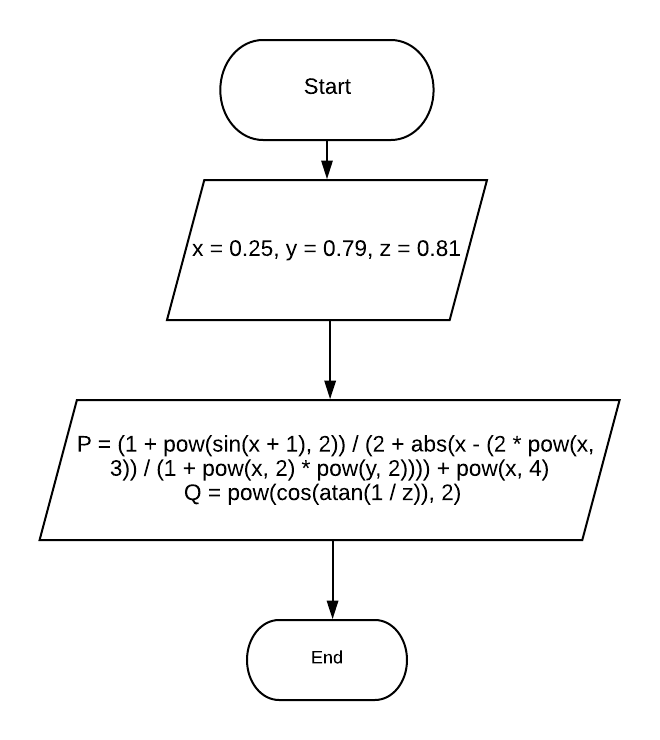
Постановка задачі:

Обчислити

якщо x = 0.25; y = 0.79; z = 0.81

Блок-схема алгоритму:

В даній задачі розраховуються змінні P та Q за значеннями сталих змінних х, у та z. Першим кроком є представлення цих змінних разом з їх значеннями, що представленні у дано. Наступний блок виконує розрахунок шуканих P та Q за зазначеними у задачі формулами.



Лістинг програми:

int main()

{

float x = 0.25, y = 0.79, z = 0.81; // Оголошення змінних

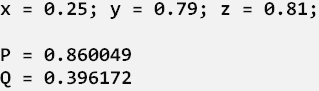
cout << "x = " << x << "; y = " << y << "; z = " << z << ";" << endl;

cout << endl << "P = " << (1 + pow(sin(x + 1), 2)) / (2 + abs(x - (2 \* pow(x, 3)) / (1 + pow(x, 2) \* pow(y, 2)))) + pow(x, 4) << endl;// Виведення на екран змінної Р разом з обчисленням її значення

cout << "Q = " << pow(cos(atan(1 / z)), 2) << endl;// Виведення на екран змінної Q разом з обчисленням її значення

}

Результат виконання:



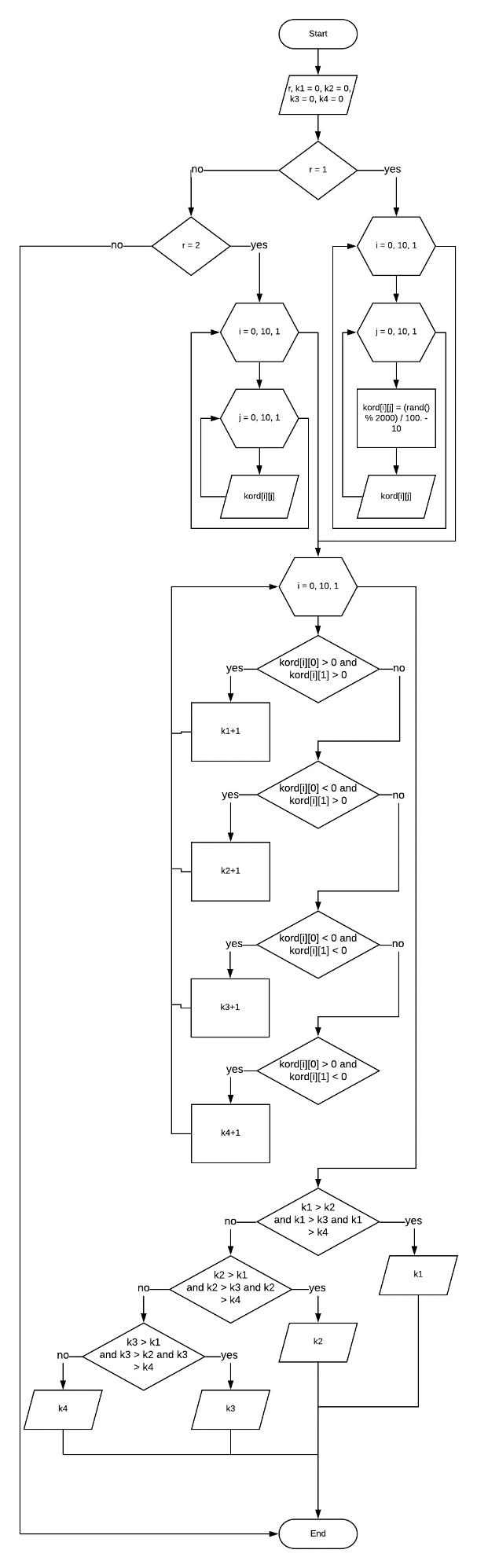
2.2 Створення програми реалізації циклічних обчислювальних процесів

Постановка задачі:

Ввести координати десяти точок на площині і визначити, яка чверть координатної площини містить найбільшу кількість заданих точок.

Блок-схема алгоритму:

В даній задачі розраховуються знаходження точок на площині. Для початку користувач обирає яким способом ввести точки. Якщо користувач присвоює змінній r значення 1, то точкам надаються випадкові значення в діапазоні від -10 до 10. Якщо змінній r надається значення 2, то користувач вводить числа самостійно з клавіатури. При інших значеннях програма не працюватиме і одразу завершить свою роботу. Після призначення точкам значень проходить перевірка кожної точки. Якщо значення х та у більше нуля, то до змінної k1 додається 1. Так само перевіряються інші значення х та у. Після цього проходить визначення у якій чверті найбільше точок. Перевіряємо, чи k1 має найбільше значення, якщо ні, то перевіряємо наступне k2. Найбільше серед k виводимо на екран.



Лістинг програми:

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251); // Підключення української мови

srand(time(NULL));

// Оголошення змінних

float kord[10][2];

int r, k1 = 0, k2 = 0, k3 = 0, k4 = 0;

cout << "Як вказати координати: 1 - рандом, 2 - вручну" << endl << "Ви обираєте = ";

cin >> r;

if (r == 1 and r == 2) {

if (r == 1) { // Наповнення масиву рандомними значеннями

cout << " x y " << endl;

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 2; j++)

{

kord[i][j] = (rand() % 2000) / 100. - 10;

cout.width(5);

cout << kord[i][j]<<" ";

}

cout << endl;

}

}

if (r == 2){ // Наповнення масиву значеннями з клавіатури

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 2; j++)

{

cout << "x" <<i+1 <<" y"<<j+1<<" = ";

cin >> kord[i][j];

}

}

cout << " x y " << endl;

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 2; j++)

{

cout.width(5);

cout << kord[i][j]<<" ";

}

cout << endl;

}

}

// Перевірка значень за чвертями

for (int i = 0; i < 10; i++){

if (kord[i][0] > 0 and kord[i][1] > 0) k1++;// х + у + (перша чверть)

if (kord[i][0] < 0 and kord[i][1] > 0) k2++;// х - у + (друга чверть)

if (kord[i][0] < 0 and kord[i][1] < 0) k3++;// х - у - (третя чверть)

if (kord[i][0] > 0 and kord[i][1] < 0) k4++;// х + у - (четверта чверть)

}

cout << endl;

// Перевірка у якій чверті найбільше точок та виведення цієї чверті

if (k1 > k2 and k1 > k3 and k1 > k4) cout << "Найбільше точок знаходиться у 1 чверті = " << k1;

if (k2 > k1 and k2 > k3 and k2 > k4) cout << "Найбільше точок знаходиться у 2 чверті = " << k2;

if (k3 > k1 and k3 > k2 and k3 > k4) cout << "Найбільше точок знаходиться у 3 чверті = " << k3;

if (k4 > k1 and k4 > k2 and k4 > k3) cout << "Найбільше точок знаходиться у 4 чверті = " << k4;

cout << endl;

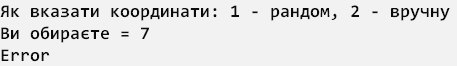
}

else { cout << "Error\n"; return(0); }

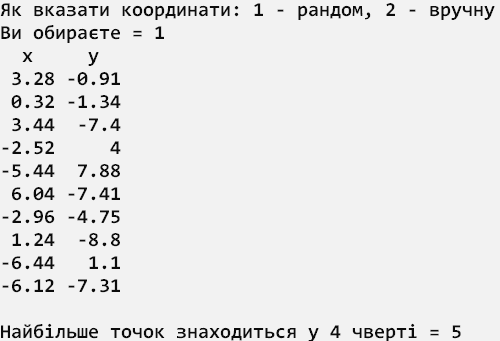
}

Результат виконання:

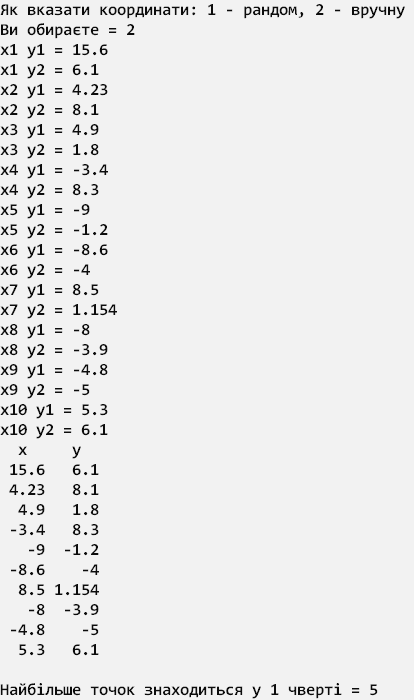
При неправильному виборі режиму:



Перший режим заповнення:



Другий режим заповнення:



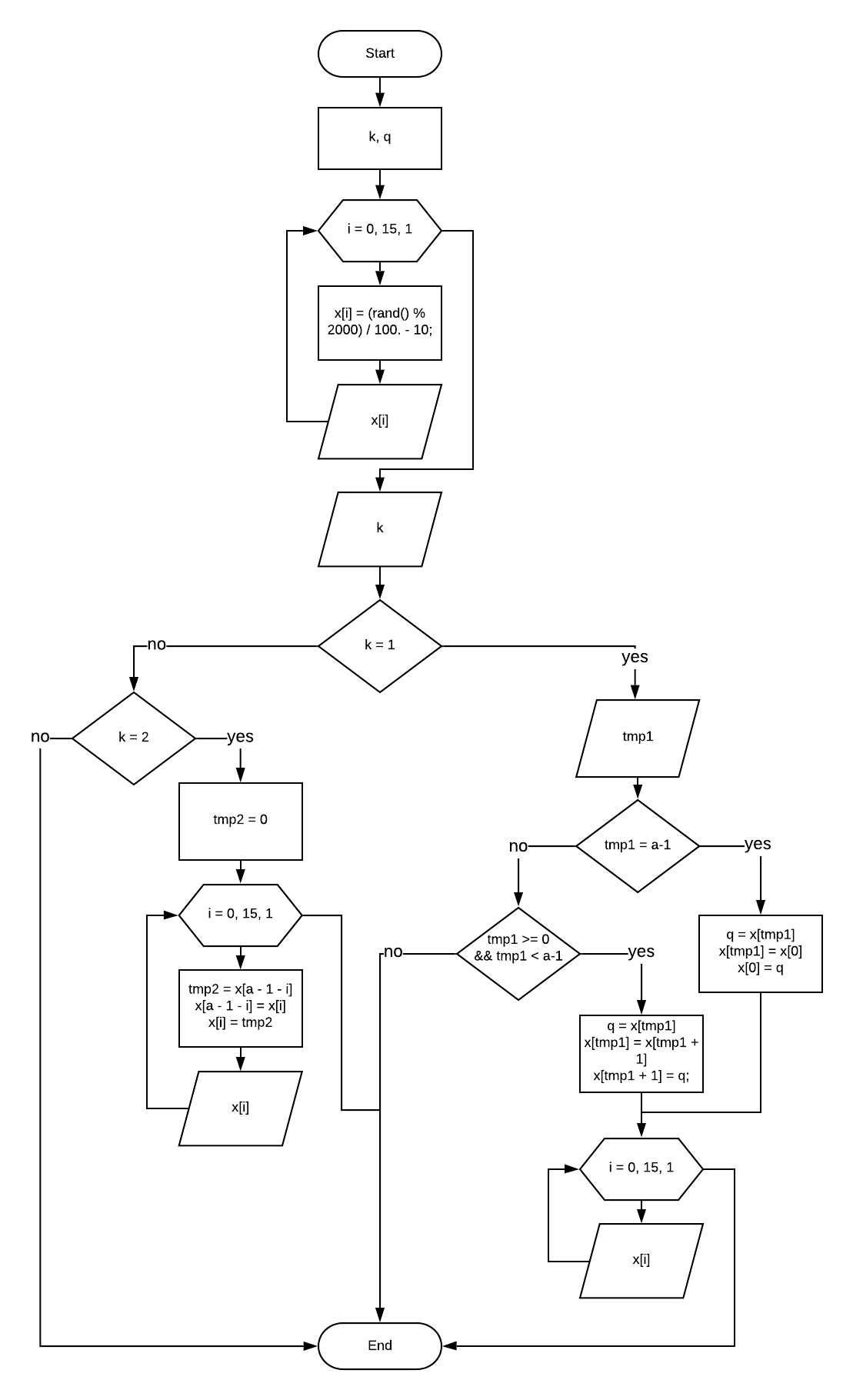
2.3 Створення програми обробки одновимірних масивів величин

Постановка задачі:

За вказівкою користувача в масиві х(16) або поміняти місцями сусідні елементи, або переставити елементи в зворотному порядку.

Блок-схема алгоритму:

В даній задачі масив, заповнений випадковими числами, обробляється за одним з передбачених в програмі режимів: перший замінює місцями елемент, що обрав користувач з наступним, другий переставляє елементи масиву у зворотному порядку. Після виконання кожного з режимів здійснюється виведення на екран обробленого масиву.



Лістинг програми:

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);// Підключення української мови

srand(time(NULL));

// Оголошення змінних

const int a = 16;

double x[a];

int k;

// Наповнення масиву рандомними значеннями

for (int i = 0; i < a; i++) {

x[i] = (rand() % 2000) / 100. - 10;

}

// Виведення елементів масиву

cout << "|";

for (int i = 0; i < a; i++) { cout << x[i] << "|"; }

// Вибір дії

cout << endl << "Оберіть дію:" << endl << "1 - переставити сусідні елементи місцями " << endl << "2 - переставити елементи у зворотньому порядку" << endl;

cout << "Ваша відповідь = "; cin >> k;

if (k != 1 && k != 2) { cout << "Error "; return (0); } // Перевірка

// Дія 1

if (k == 1) {

// Оголошення локальних змінних

int tmp1 = 0;

double q;

cout << "Ведіть номер елементу, який хочете поміняти з наступним (від 0 до 15) = "; cin >> tmp1;

// Заміна значеннь

if (tmp1 == a-1) {

q = x[tmp1];

x[tmp1] = x[0];

x[0] = q;

}

if (tmp1 >= 0 && tmp1 < a-1) {

q = x[tmp1];

x[tmp1] = x[tmp1 + 1];

x[tmp1 + 1] = q;

}

else { cout << "Error "; return (0); }

// Виведення оновленого масиву

cout << "|";

for (int i = 0; i < a; i++) { cout << x[i] << "|"; }

}

// Дія 2

if (k == 2) {

// Оголошення локальних змінних

double tmp2 = 0;

// Заміна значеннь у зворотньому порядку

for (int i = 0; i < a / 2; i++)

{

tmp2 = x[a - 1 - i];

x[a - 1 - i] = x[i];

x[i] = tmp2;

}

// Виведення оновленого масиву

cout << "|";

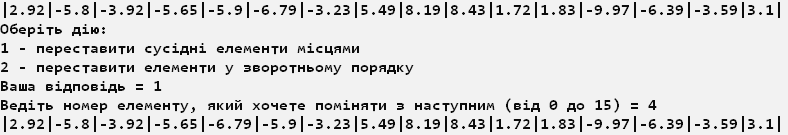
for (int i = 0; i < a; i++) { cout << x[i] << "|"; }

}

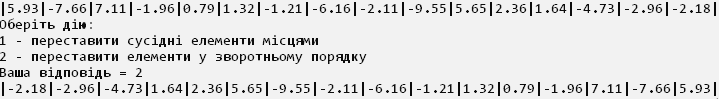
}

Результат виконання:

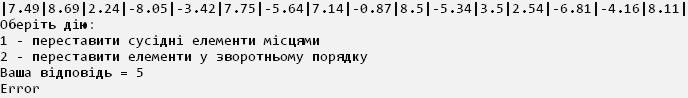
Перший режим:



Другий режим:



При неправильно обраному режимі:



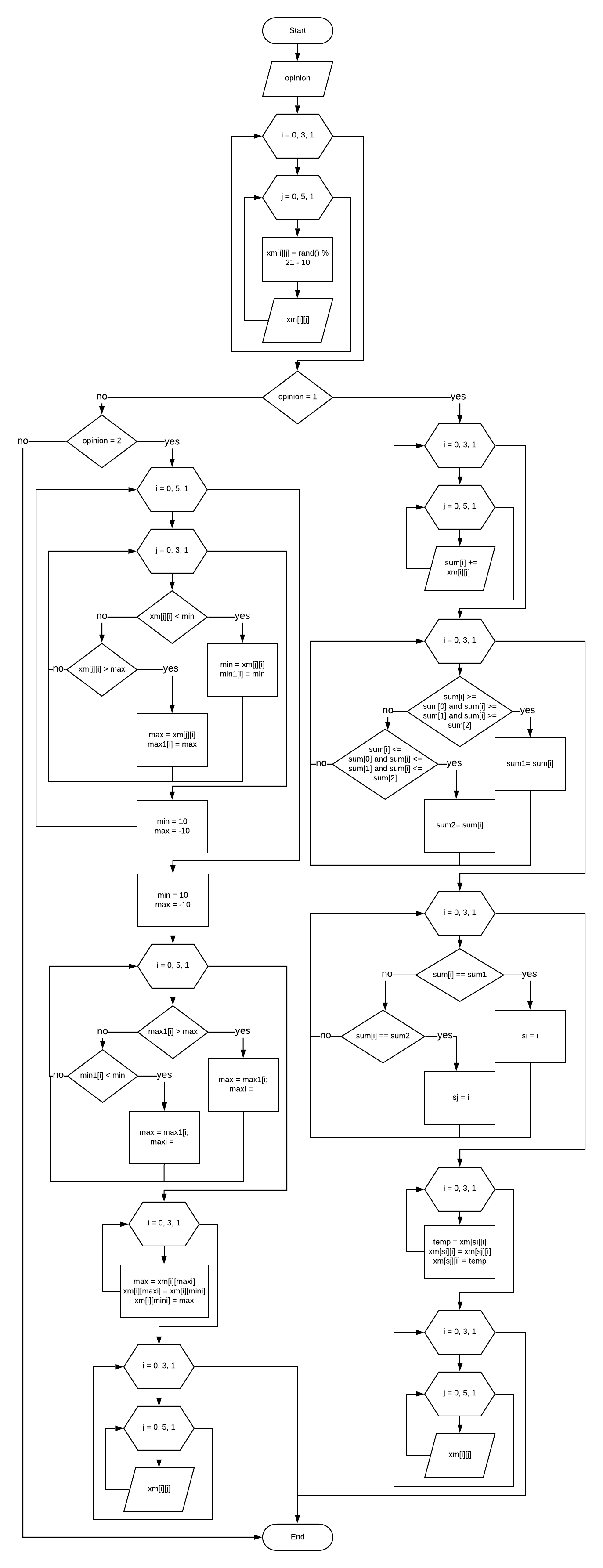
2.4 Створення програми обробки двовимірних масивів величин

Постановка задачі:

За вказівкою користувача у матриці хm(3,5) поміняти місцями або ряди з найбільшою та найменшою сумою елементів, або стовпці з мінімальним та максимальним елементом.

Блок-схема алгоритму:

В даній задачі прямокутна матриця 3х5 обробляється за одним з передбачених в програмі режимів: перший режим міняє місцями ряди з найбільшою та найменшою сумою елементів, другий режим стовпці з мінімальним та максимальним елементом. Після виконання кожного з режимів здійснюється виведення на екран обробленого масиву.



Лістинг програми:

int main()

{

// Підключення української мови

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

// Оголошення змінних

const int x = 3;

const int m = 5;

int xm[x][m], max1[m], min1[m], max = -10, min = 10;

int opinion, maxi=0, mini;

int sum[x], sum1=0, sum2=0, si =0, sj=0, temp = 0;

// Наповнення масиву рандомними значеннями та вивід його на екран

for (int i = 0; i < x; i++)

{

for (int j = 0; j < m; j++)

{

xm[i][j] = rand() % 21 - 10;

cout.width(4);

cout << xm[i][j];

}

cout << endl;

}

// Вибір режиму роботи програми

cout << "Оберіть дію: " << endl << " 1 - Поміняти місцями ряди з найбільшою та найменшою сумою елементів" << endl;

cout << " 2 - Поміняти стовпці з максимальним та мінімальним елементом";

cout << endl << "Ви обираєте? - "; cin >> opinion;

// Перевірка

if (opinion != 1 or opinion != 2) { cout << "Error\n"; return(0);}

// Перший режим

if (opinion == 1) {

// Блок обчислень

for (int i = 0; i < x; i++)

{

sum1 = 0;

for (int j = 0; j < m; j++) {

sum1 += xm[i][j];

}

sum[i] = sum1;

cout << "sum"<<i+1<<" = " << sum1 << endl;

}

// Знаходимо ряд з найбільшою та найменшою сумою

for (int i = 0; i < x; i++) {

if (sum[i] >= sum[0] and sum[i] >= sum[1] and sum[i] >= sum[2]) { sum1 = sum[i]; }

else if (sum[i] <= sum[0] and sum[i] <= sum[1] and sum[i] <= sum[2]) { sum2 = sum[i]; }

}

for (int i = 0; i < x; i++) {

if (sum[i] == sum1) si = i;

else if (sum[i] == sum2) sj = i;

}

for (int i = 0; i < m; i++)

{

temp = xm[si][i];

xm[si][i] = xm[sj][i];

xm[sj][i] = temp;

}

// Вивід оновленої матриці

for (int i = 0; i < x; i++)

{

for (int j = 0; j < m; j++)

{

cout.width(4);

cout << xm[i][j];

}

cout << endl;

}

}

// Другий режим

else if (opinion == 2) {

// Блок обчислень

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 3; j++) {

if (xm[j][i] < min) {

min = xm[j][i];

min1[i] = min;

}

if (xm[j][i] > max) {

max = xm[j][i];

max1[i] = max;

}

}

min = 10;

max = -10;

}

max = -10; min = 10;

for (int i = 0; i < 5; i++)

{

if (max1[i] > max) { max = max1[i]; maxi = i; }

else if (min1[i] < min) { min = min1[i]; mini = i; }

}

cout << endl << "min = " << min << " ("<<mini + 1 <<" ряд)" << " max = " << max << " (" << maxi+1 << " ряд)" << endl;

cout << "Міняємо ряди "<<mini+1<<" та "<<maxi+1 << endl;

max = 0;

for (int i = 0; i < x; i++)

{

max = xm[i][maxi];

xm[i][maxi] = xm[i][mini];

xm[i][mini] = max;

}

// Виведення опрацьованої матриці

cout << endl;

for (int i = 0; i < x; i++)

{

for (int j = 0; j < m; j++)

{

cout.width(4);

cout << xm[i][j];

}

cout << endl;

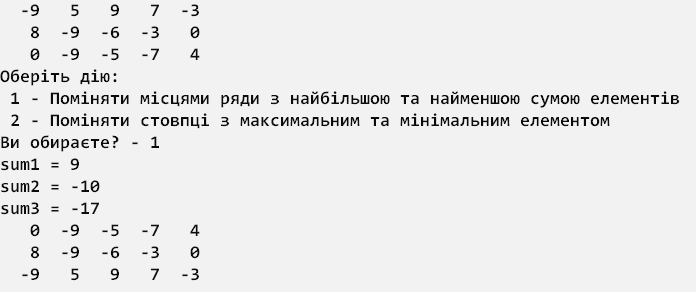
}

}

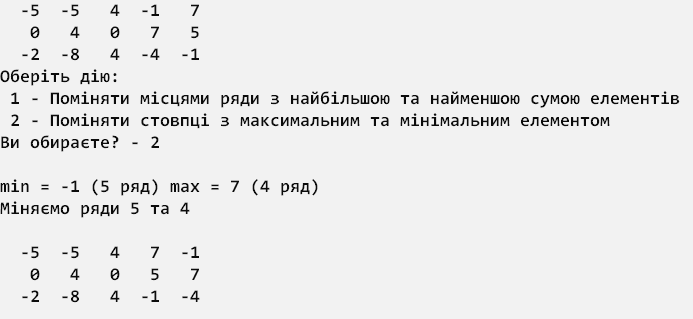
}

Результат виконання:

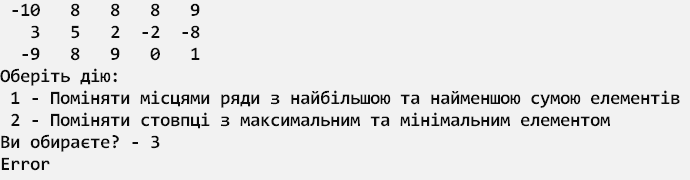
Перший режим:



Другий режим:



При неправильному вводі режиму:



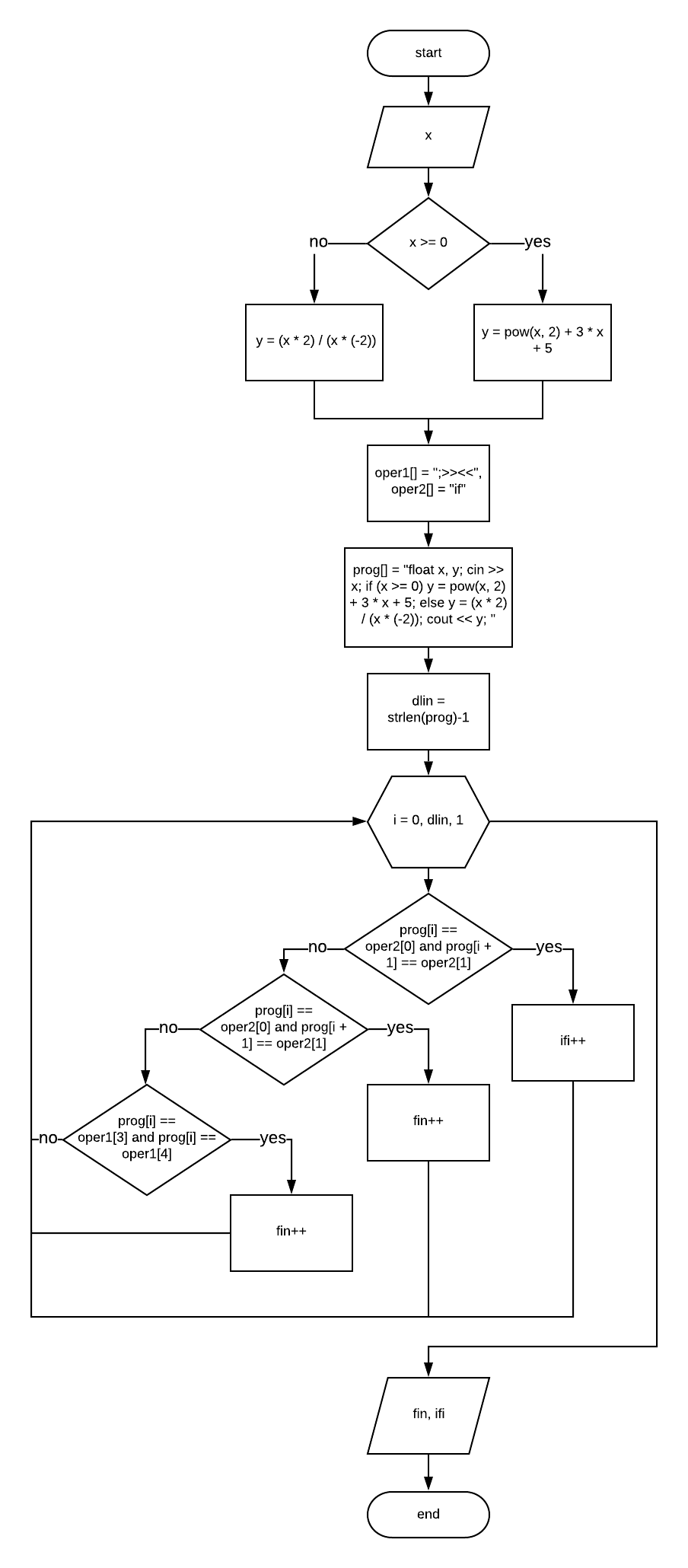
2.5 Створення програми обробки рядків

Постановка задачі:

Ввести невелику програму, написану мовою С++. Визначити, скільки в ній операторів і скільки разів повторюється оператор іf.

Блок-схема алгоритму:

В даній задачі опрацьовується невелика програма, що при різних значеннях х рахує значення у. Після код цієї програми записується у символьний масив. Далі програма порівнює кожен елемент масиву з веденими операторами і підраховує їх кількість. Далі відбувається виведення результатів на екран.



Лістинг програми:

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);// Підключення української мови

float x, y;// Оголошення зміних у програмі прикладі

cin >> x;

// Обчислення програми прикладу

if (x >= 0) y = pow(x, 2) + 3 \* x + 5;

else y = (x \* 2) / (x \* (-2));

// Вивід обчислень програми прикладу

cout << y;

// Оголошення операторів, які будемо шукати у програмі

char oper1[] = ";>><<", oper2[] = "if";

int ifi = 0, fin = 0;

// Оголошення коду програми прикладу у чарівському масиві

char prog[] = "float x, y; cin >> x; if (x >= 0) y = pow(x, 2) + 3 \* x + 5; else y = (x \* 2) / (x \* (-2)); cout << y; ";

int dlin = strlen(prog) - 1;

cout << endl << "Код реалізованої програми:" << endl;

// Розрахунок кількості операторів серед програми прикладу

for (int i = 0; i < dlin; i++)

{

if (prog[i] == oper2[0] and prog[i + 1] == oper2[1]) ifi++;

if (prog[i] == oper1[0] or prog[i] == oper1[1] and prog[i] == oper1[2]) fin++;

if (prog[i] == oper1[3] and prog[i] == oper1[4]) fin++;

cout << prog[i];

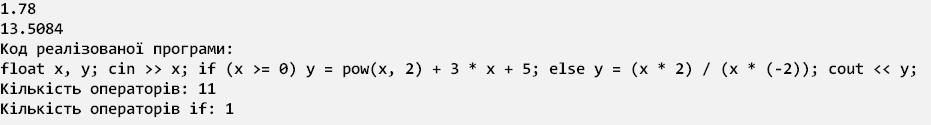
}

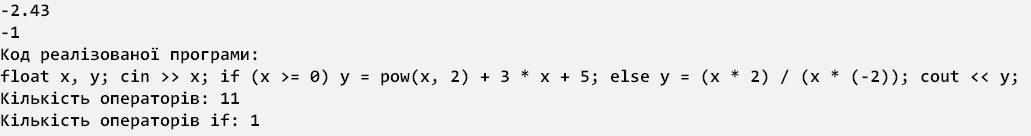
// Вивід результатів

cout << endl << "Кількість операторів: " << fin+ifi << endl << "Кількість операторів if: " << ifi << endl;

}

Результат виконання:





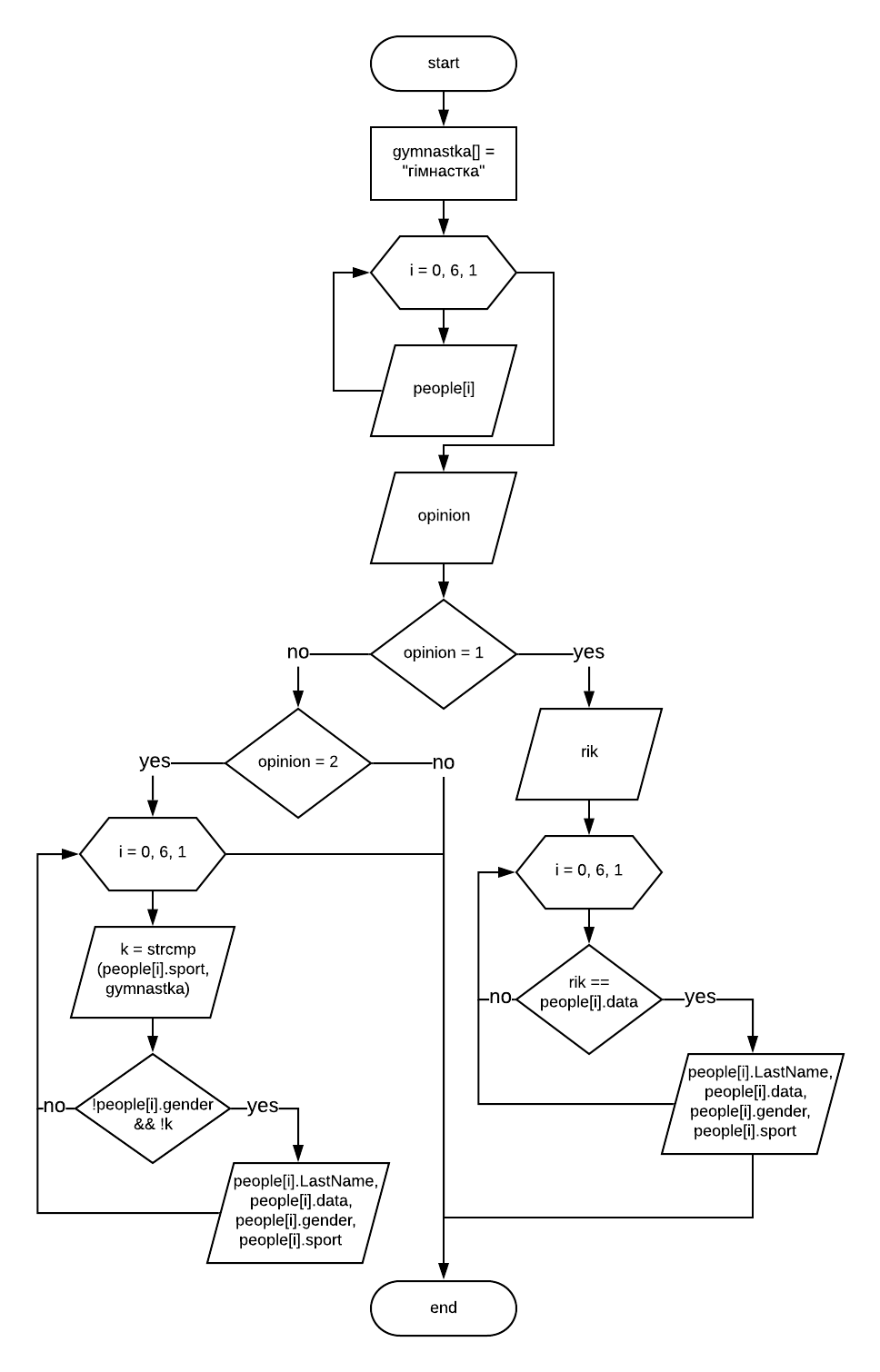
2.6 Створення програми обробки структур

Постановка задачі:

Увести список спортсменів, що складається з прізвища, статі, року народження і виду спорту. Вивести за вказівкою користувача список легкоатлетів заданого року народження або список жінок-гімнасток.

Блок-схема алгоритму:

В даній задачі введений користувачем масив структур обробляється за одним з передбачених режимів: перший режим за вказівкою користувача виводить список легкоатлетів заданого року народження, другий режим виводить список жінок-гімнасток.



Лістинг програми:

// Оголошення структури

struct People

{

char LastName[20];

int data;

bool gender;

char sport[20];

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);// Підключення української мови

// Оголошення зміних

char gymnastka[] = "гімнастка";

const int N = 6;

People people[N];

// Наповнення масиву значеннями

for (int i = 0; i < N; i++) {

cout << "Введіть призвіще: ";

cin >> people[i].LastName;

cout << "Введіть рік народження: ";

cin >> people[i].data;

cout << "Введіть стать [1-чоловік][0-жінка]: ";

cin >> people[i].gender;

cout << "Введіть назву спорту: ";

cin >> people[i].sport;

cout << "\n";

}

// Вибір режиму роботи програми

int rik = 0, opinion;

cout << "Оберіть дію: " << endl << " 1 - задайте рiк народження легкоатлетiв, що хочете дiзнатись " << endl;

cout << " 2 - Вивести список жiнок гімнасток ";

cout << endl << "Ви обираєте? - "; cin >> opinion;

// Перший режим

if (opinion == 1) {

cout << "Введіть рік ";

cin >> rik;

for (int i = 0; i < N; i++) {

if (rik == people[i].data) {

cout << people[i].LastName << "\t" << people[i].data << "\t" << people[i].gender << "\t" << people[i].sport;

cout << endl;

}

}

}

// Другий режим

else if (opinion == 2) {

int k;

for (int i = 0; i < N; i++) {

k = strcmp(people[i].sport, gymnastka);

if (!people[i].gender && !k ) {

cout << people[i].LastName << "\t" << people[i].data << "\t" << people[i].gender << "\t" << people[i].sport;

cout << endl;

}

}

}

// При неправильному вводі режиму дії

else {

cout << "Помилка"; return(0);

}

}

Результат виконання:

Перший режим:



Другий режим:



При неправильному виборі режиму:



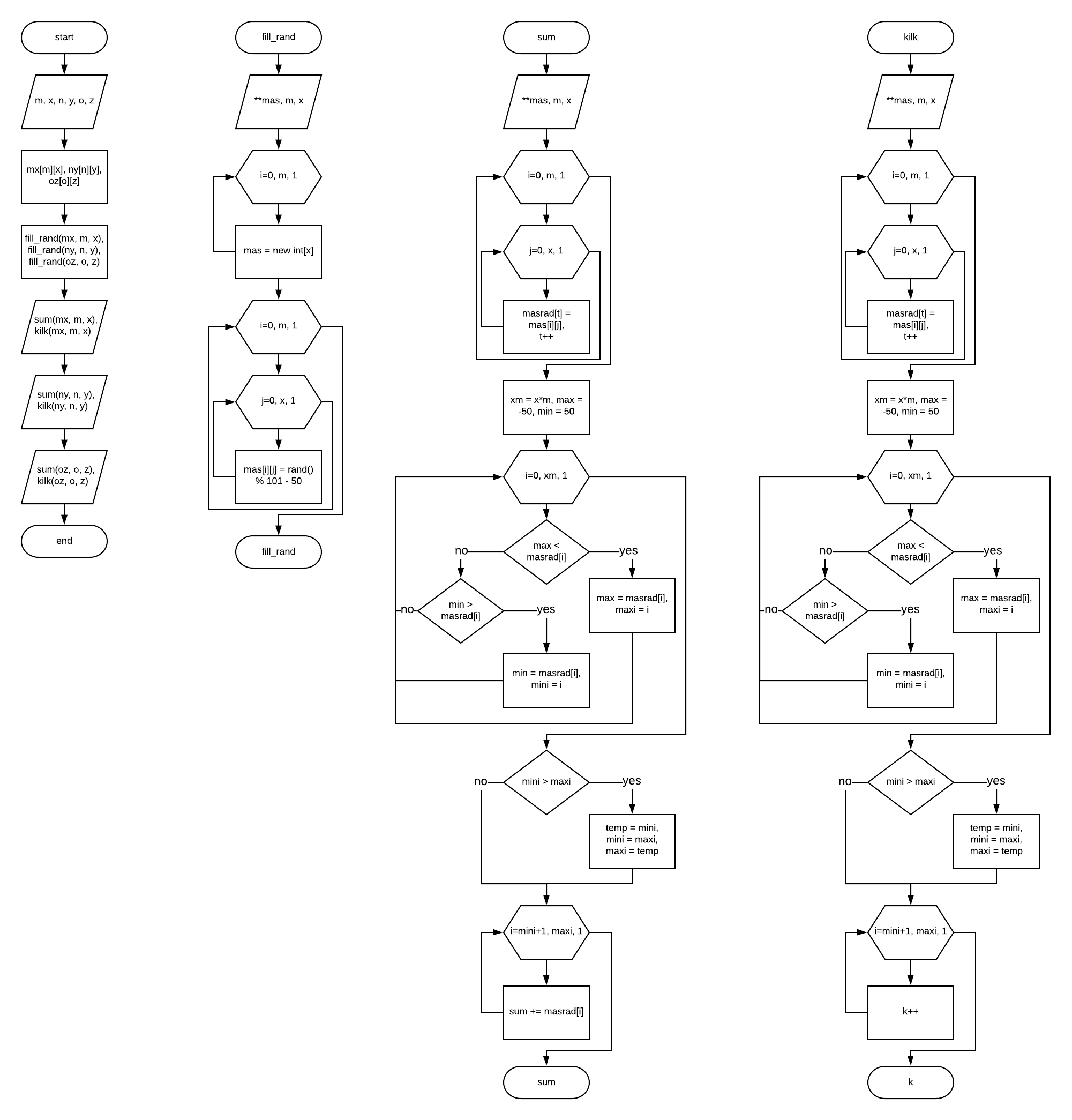
2.7 Створення програми реалізації функцій

Постановка задачі:

Скласти функцію визначення суми та кількості елементів, що розташовані між мінімальним і максимальним елементами матриці, та застосувати її для обробки трьох довільних двовимірних масивів.

Блок-схема алгоритму:

В даній задачі опрацьовується три довільних масиви, розмірності яких вводить користувач. На початку масиви наповнюються випадковими числами, завдяки роботі функції fill\_rand. Після масив опрацьовується функціями sum та kilk. Для опрацювання масивів функція переробляє їх у рядок чисел. Після знаходиться мінімум та максимум матриці. Функція sum рахує суму чисел, що знаходяться між мінімумом та максимумом, коли функція kilk рахує кількість таких чисел. Після опрацювання матриць вони виводяться на екран.



Лістинг програми:

// Функція знаходження суми

int sum(int\*\* mas, int m, int x) {

int min = 10, max = -10, maxi = 0, mini = 0;

int sum = 0, temp = 0, t = 0;

const int m1 = m;

int xm = x \* m;

int\* masrad = new int[xm];

// Запис елементів з двовимірної матриці mas в одновимірний масив masrad

for (int i = 0; i < m; i++)

{

for (int j = 0; j < x; j++) {

masrad[t] = mas[i][j];

t++;

}

}

cout << endl;

// Знаходження мінімуму та максимуму

for (int i = 0; i < xm; i++) {

if (max < masrad[i]) { max = masrad[i]; maxi = i; }

else if (min > masrad[i]) { min = masrad[i]; mini = i; }

cout.width(4);

cout << masrad[i];

}

cout << endl << "max = " << max << " maxi = " << maxi << endl;

cout << "min = " << min << " mini = " << mini << endl;

if (mini > maxi) {

temp = mini; mini = maxi; maxi = temp;

}

// Знаходження суми елементів між мінімумом і максимумом

for (int i = mini+1; i < maxi; i++)

{

sum += masrad[i];

}

return sum;

}

// Функція знаходження кількості між мінімумом і максимумом

int kilk(int\*\* mas, int m, int x) {

int min = 10, max = -10, maxi = 0, mini = 0;

int k = 0, temp = 0, t = 0;

const int m1 = m;

int xm = x \* m;

int\* masrad = new int[xm];

// Запис елементів з двовимірної матриці mas в одновимірний масив masrad

for (int i = 0; i < m; i++)

{

for (int j = 0; j < x; j++) {

masrad[t] = mas[i][j];

t++;

}

}

// Знаходження мінімуму та максимуму

for (int i = 0; i < xm; i++) {

if (max < masrad[i]) { max = masrad[i]; maxi = i; }

else if (min > masrad[i]) { min = masrad[i]; mini = i; }

}

if (mini > maxi) {

temp = mini; mini = maxi; maxi = temp;

}

// Знаходження кількості елементів між мінімумом і максимумом

for (int i = mini + 1; i < maxi; i++)

{

k++;

}

return k;

}

// Функція заповнення масиву випадковими елементами

void fill\_rand(int\*\* mas, int m, int x) {

for (int i = 0; i < m; i++) mas[i] = new int[x];

for (int i = 0; i < m; i++)

{

for (int j = 0; j < x; j++)

{

mas[i][j] = rand() % 101 - 50;

cout.width(4);

cout << mas[i][j];

}

cout << endl;

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);// Підключення української мови

srand(time(NULL));

// Введення розмірностей 3-х матриць

int m, x;

cout << "Введіть розмірність першої матриці: " << endl << "Рядків = "; cin >> m; cout << "Стовпчиків = "; cin >> x;

int n, y;

cout << "Введіть розмірність другої матриці: " << endl << "Рядків = "; cin >> n; cout << "Стовпчиків = "; cin >> y;

int o, z;

cout << "Введіть розмірність третьої матриці: " << endl << "Рядків = "; cin >> o; cout << "Стовпчиків = "; cin >> z;

int\*\* mx = new int\* [m];

int\*\* ny = new int\* [n];

int\*\* oz = new int\* [o];

// Заповнення матриць, після використання функції sum i kilk і вивід результату

fill\_rand(mx, m, x);

cout << endl << "Сума між максимальним та мінімальними значеннями: " << sum(mx, m, x) << endl;

cout << "Кількість елементів між максимальним та мінімальними значеннями: " << kilk(mx, m, x) << endl << endl;

fill\_rand(ny, n, y);

cout << endl << "Сума між максимальним та мінімальними значеннями: " << sum(ny, n, y) << endl;

cout << "Кількість елементів між максимальним та мінімальними значеннями: " << kilk(ny, n, y) << endl << endl;

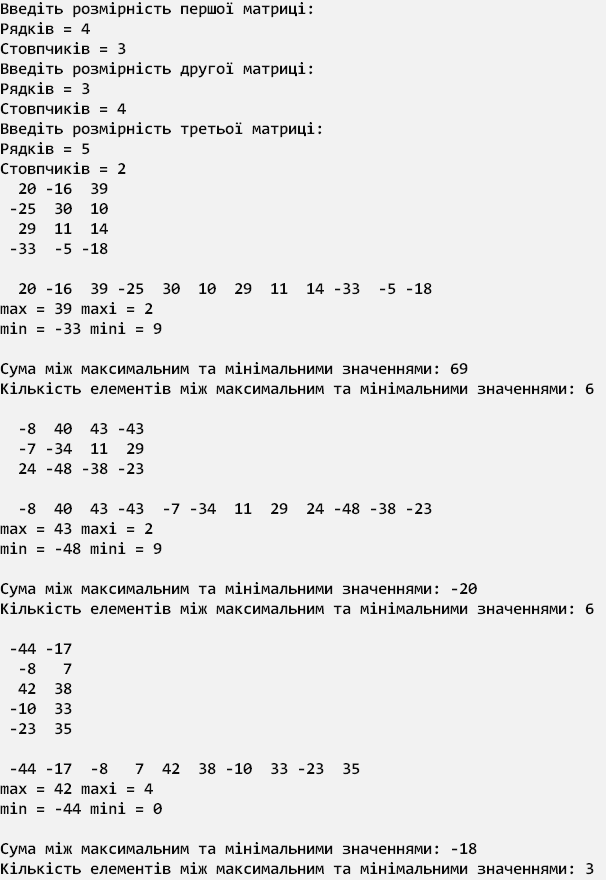
fill\_rand(oz, o, z);

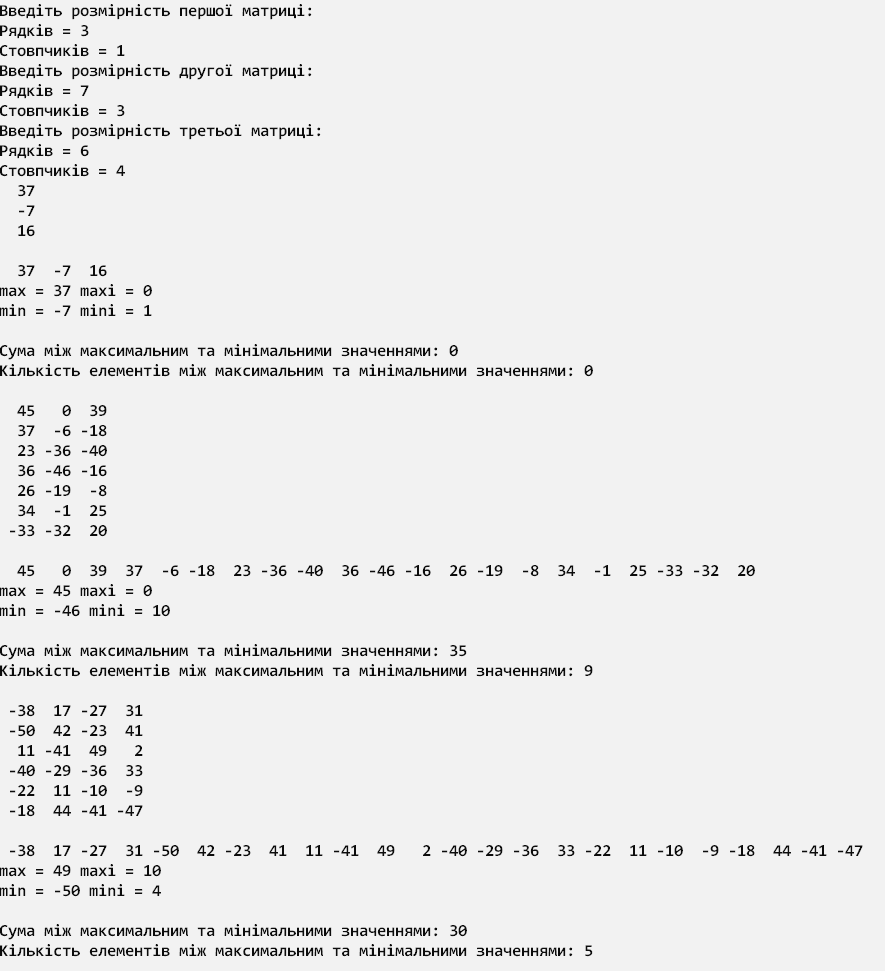
cout << endl << "Сума між максимальним та мінімальними значеннями: " << sum(oz, o, z) << endl;

cout << "Кількість елементів між максимальним та мінімальними значеннями: " << kilk(oz, o, z) << endl << endl;

}

Результат виконання:





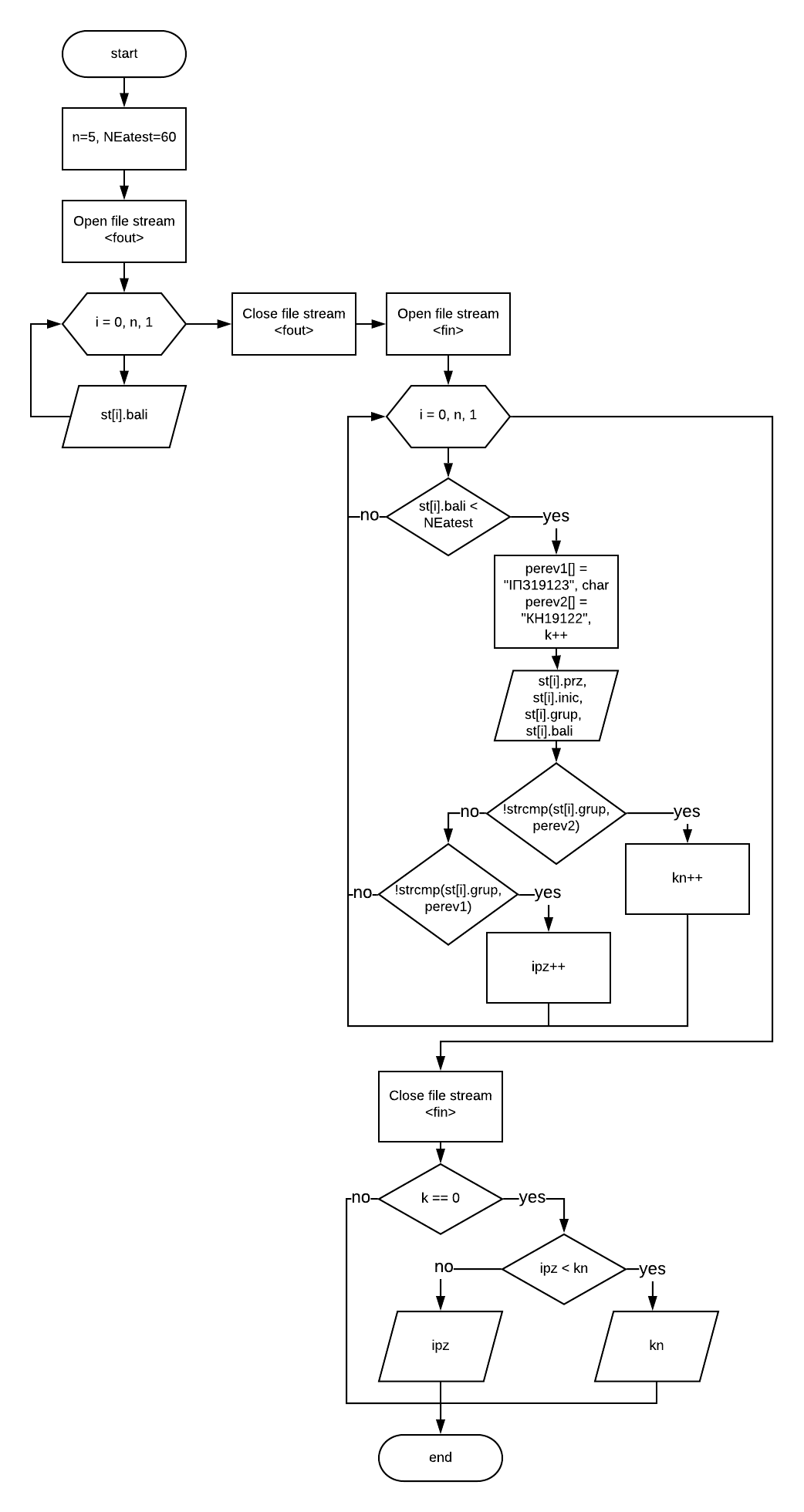
2.8 Створення програми обробки файлів

Постановка задачі:

Створити файл, який містить повідомлення про студентів (прізвища, ініціали, група) і результати атестації з програмування. Вивести список неатестованих студентів. Зазначити, в якій групі таких студентів найбільше.

Блок-схема алгоритму:

В даній задачі введений користувачем масив структур записується у файл stud.dat. Аналізується ряд st[i].bali. При тому, коли число менше 60, студент вважається неатестованим. Після йде перевірка, з якої він групи. Розраховується скільки неатестованих у кожній з груп. В групі, у якій найбільше таких студентів, виводиться на екран.



Лістинг програми:

// Оголошення структури

struct STUDENT {

char prz[10];

char inic[4];

int bali;

char grup[10];

};

void main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);// Підключення української мови

// Оголошення змінних

const int n = 5;

int NEatest = 60;

int ipz = 0, k =0, kn=0;

STUDENT st[n];

ofstream fout("stud.dat"); // відкрити запис у файл stud.dat

if (!fout) cout << "Error";

cout << "Вводити бали студентів від 0 до 100\n";

// Наповнення масиву значеннями

for (int i = 0; i < n; i++)

{

cout << "Призвіще: "; cin >> st[i].prz;

cout << "Ініціали: "; cin >> st[i].inic;

cout << "Назва групи: "; cin >> st[i].grup;

cout << "Бали: "; cin >> st[i].bali;

cout << endl;

fout<<setw(12) << st[i].prz << setw(6) << st[i].inic << setw(12) << st[i].grup << setw(5) << st[i].bali << endl;

}

fout.close();// закрити запис у файл stud.dat

ifstream fin("stud.dat"); // відкрити запис у файл stud.dat

if (!fin) cout << "Error";

// Блок розв'язку

for (int i = 0; i < n; i++)

{

if (st[i].bali < NEatest) {

k++;

char perev1[] = "ІПЗ19123";

char perev2[] = "КН19122";

cout << st[i].prz << " " << st[i].inic << " з групи " << st[i].grup << " Неастестований\n";

cout << "Набрав - " << st[i].bali << " що менше за мінімальні 60 балів\n" << endl;

if (!strcmp(st[i].grup, perev2)) { kn++; }

if (!strcmp(st[i].grup, perev1)) { ipz++; }

}

}

// Перевірка відповіді і вивід результату

if (k == 0) cout << "Помилка, усі студенти атестовані\n";

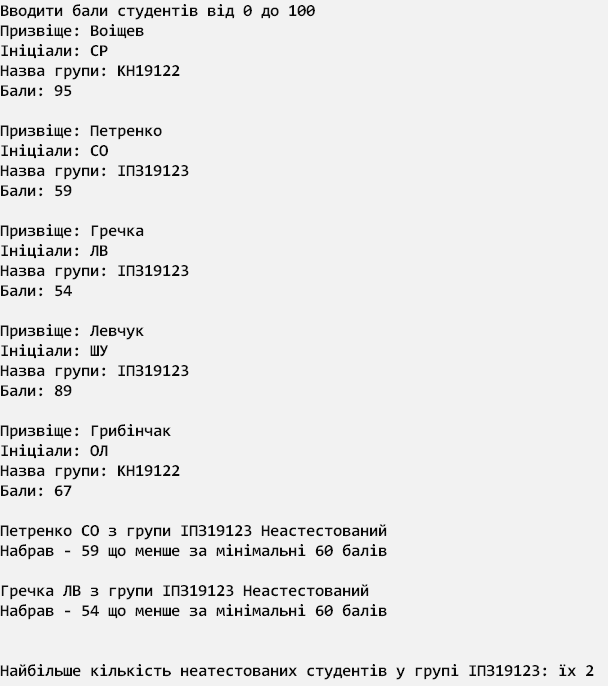
if (ipz < kn) cout << "\nНайбільше кількість неатестованих студентів у групі КН19122: їх "<< kn << "\n";

if (ipz > kn) cout << "\nНайбільше кількість неатестованих студентів у групі ІПЗ19123: їх "<<ipz<<"\n";

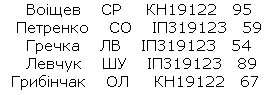
fin.close();

}

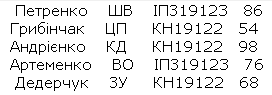
Результат виконання:



Як виглядає файл, після опрацювання програмою:







ВИСНОВОК

Під час виконання курсової роботи було розглянуто роботу покажчиків, динамічної пам’яті та динамічних масивів. Було зазначено принципи їх використання, показані переваги використання динамічних матриць над звичайними. Також показано, як саме виділяється пам’ять у комп’ютері і що відбувається при оголошені динамічних масивів.

У практичній частині курсової роботи були створені консольні додатки з використанням базових алгоритмічних структур. Мною було вивчено як виконується обробка масивів, функцій та структур, як розробляються блок-схеми до програм. Розглянуто роботу алгоритмів задач.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Мєшков А. Visual C ++ і MFC : монографія. БХВ-Петербург, 2013. 546 c.

2. Дьюхарст Програмування на C ++: монографія. ДиаСофт, 2015. 272 c.

3. Фрідман, А. C. C ++. Архів програм: монографія. ЗАТ Видавництво БІНОМ, 2016. 640 c.

4. Секун, Н.Ю. Самовчитель Visual C ++ : монографія. СПб: BHV, 2014. 960 c.

5. Довбуш, Галина Visual C ++ на прикладах : монографія. БХВ-Петербург, 2012. 528 c.

6. Динамічні масиви: вебсайт. URL: <http://cpp.dp.ua/dynamichni-masyvy/> (дата звернення 10.04.2020)

7. Покажчики та масиви: вебсайт. URL: <http://cpp.dp.ua/pokazhchyky-ta-masyvy/> (дата звернення 07.04.2020)

8. Виділення пам‘яті для покажчика: вебсайт. URL: https://www.bestprog.net/uk/2017/04/30/покажчики-частина-5-виділення-памят/ (дата звернення 08.04.2020)

9. Трофименко О.Г. Основи програмування. Теорія і практика: монографія. Одеса: Фенікс, 2010. 544 с.

10. Грицюк Ю. Програмування мовою С++: монографія. Львів: Вид-во ЛДУ БЖД, 2011. 292 с.