

Лабораторна робота № 6

Тема: Розробка програм для опрацювання динамічних масивів.

Мета роботи: Навчитись виділяти та використовувати динамічну пам'ять для розміщення масивів даних.

Теоретичні відомості

Динамічна пам'ять (heap, купа) – це пам'ять, в якій змінні розміщуються динамічно. Основна потреба у динамічному виділенні пам'яті виникає тоді, коли розмір або кількість даних, які необхідно виділити, наперед невідомі і визначаються у процесі виконання програми.

Засоби динамічного управління пам'яттю дають змогу розширити можливості програмування. Під час свого виконання програма отримує можливість самостійно визначати потрібні розміри пам'яті і за потреби в реальному часі створювати та знищувати динамічні об'єкти.

Динамічно виділена пам'ять, де б вона не виділялась, живе поки її не звільнити. Якщо динамічно виділену пам'ять не звільнити, то в операційній системі може виникнути ситуація нестачі вільної пам'яті.

Робота з динамічними масивами виконується за допомогою змінних типу вказівник. Після створення масива ця змінна буде вказувати на його початок, тобто буде містити адресу його першого елемента.

Розмір динамічного масива задається змінною, а не константою, значення якої обчислюється за певними формулами або його вводить з клавіатури користувач.

Динамічне управління пам'яттю в мові C/C++ виконується за допомогою стандартних функцій, оголошення яких містяться в заголовочному файлі `alloc.h` (або `malloc.h`).

Таблиця 1.

Робота з динамічною пам'яттю
#include <alloc.h>

Назва	Звертання	Призначення
calloc	<code>void * calloc(size_t nitems, size_t size);</code>	Виділяє динамічну пам'ять для масиву <code>nitems</code> елементів, кожен з яких займає <code>size</code> байтів. Виділена пам'ять ініціалізується нулями. Повертає вказівник на виділену область, або <code>NULL</code> , якщо немає доступної пам'яті.
free	<code>void free(void *block);</code>	Звільняє блок динамічної пам'яті, закріпленої за вказівником <code>block</code> .
malloc	<code>void * malloc(size_t size);</code>	Виділяє <code>size</code> байтів динамічної пам'яті. Ініціалізація виділеної пам'яті не здійснюється. Повертає вказівник на виділену область, або <code>NULL</code> , якщо немає доступної пам'яті.

Назва	Звертання	Призначення
realloc	<code>void * realloc(void *block, size_t size);</code>	Змінює розмір динамічної пам'яті, раніше виділеної функціями <code>calloc</code> або <code>malloc</code> та закріпленої за вказівником <code>block</code> . Встановлюється новий розмір пам'яті, рівний <code>size</code> байтів. Вміст старого блоку не змінюється. Повертає вказівник на початок виділеної області пам'яті.
brk	<code>int brk(void *addr);</code>	Визначає нове значення змінної <code>break</code> , яка визначає адресу першого байта оперативної пам'яті, не доступної для програми. Тим самим змінюється розмір оперативної пам'яті, відведений для поточного процесу. Функція повертає 0 при успішній зміні розміру оперативної пам'яті, або -1, якщо розмір пам'яті не може бути змінений.
sbrk	<code>void * sbrk(int incr);</code>	Змінює значення глобальної змінної <code>break</code> на <code>incr</code> байтів, що призводить до зміни (збільшення або зменшення) розміру оперативної пам'яті, відведеної поточному процесу. Значення змінної <code>break</code> визначає адресу першого байта оперативної пам'яті, не доступної для програми. Функція повертає старе значення змінної <code>break</code> , або -1, якщо розмір пам'яті процесу не може бути змінений.
heapcheck	<code>int heapcheck(void);</code>	Перевіряє блоки динамічної пам'яті на помилки та її цілісність. Якщо перевірка була успішною, то повертає значення, більше від нуля, інакше – менше від нуля значення.
heapfillfree	<code>int heapfillfree(unsigned int fillvalue);</code>	Заповнює вільні блоки динамічної пам'яті значенням величини <code>fillvalue</code> . При успішному завершенні повертає більше від нуля значення, інакше – менше від нуля значення.
heapcheckfree	<code>int heapcheckfree(unsigned int fillvalue);</code>	Перевіряє вільні блоки динамічної пам'яті на заповненість значеннями <code>fillvalue</code> . При успішному завершенні повертає більше від нуля значення, інакше – менше від нуля значення.
coreleft	<code>unsigned long coreleft(void);</code>	Повертає розмір пам'яті у байтах, яка є доступною для динамічного розподілу. Для моделей пам'яті <code>tiny</code> , <code>small</code> , <code>medium</code> повертає вільний об'єм ближньої, а для моделей <code>compact</code> , <code>large</code> , <code>huge</code> – дальньої динамічної пам'яті.

Назва	Звертання	Призначення
heapchecknode	int heapchecknode(void far *node);	Перевіряє стан блоку динамічної пам'яті, що визначається вказівником node. При успішному завершенні повертає значення більші від нуля: _HEAPEMPTY (= 1) No heap _FREEENTRY (= 3) Node is a free block _USEDENTRY (= 4) Node is a used block Якщо помилка, то повертає значення менші від нуля: _HEAPCORRUPT (= -1) Heap has been corrupted _BADNODE (= -2) Node could not be found
heapwalk	int heapwalk(struct farheapinfo far *hi);	Сканує динамічну пам'ять від елемента до елемента. Аргументом є вказівник на структуру heapinfo: struct heapinfo { void *ptr; unsigned int size; int in_use; }; При першому виклику поле hi.ptr повинно дорівнювати NULL. Далі через це поле повертається адреса першого блоку динамічної пам'яті. Поле hi.size зберігає розмір блоку у байтах. Поле hi.in_use є прапорцем, який встановлений, якщо блок використовується у даний час. Функція heapwalk вважає, що динамічна пам'ять є коректною. Для перевірки динамічної пам'яті перед викликом heapwalk необхідно викликати heapcheck. При успішному завершенні повертає значення, більші від нуля: _HEAPEMPTY (= 1) Немає вільної ДП. _HEAPOK (= 2) ДП є перевірена (heapinfo або farheapinfo блоки містять правильні дані) _HEAPEND (= 5) Досягнуто кінець ДП. Якщо помилка, то повертає значення менше від нуля.
farcalloc	void far * farcalloc (unsigned long nunits, unsigned long unitsz);	calloc для дальньої динамічної пам'яті
farcoreleft	unsigned long farcoreleft (void);	coreleft для дальньої динамічної пам'яті
farfree	void farfree(void far *block);	free для дальньої динамічної пам'яті
farmalloc	void far * farmalloc (unsigned long nbytes);	malloc для дальньої динамічної пам'яті

Назва	Звертання	Призначення
farrealloc	void far * farrealloc (void far *oldblock, unsigned long nbytes);	realloc для дальньої динамічної пам'яті
farheapcheck	int farheapcheck(void);	heapcheck для дальньої динамічної пам'яті
farheapchecknode	int farheapchecknode (void far *node);	heapchecknode для дальньої динамічної пам'яті
farheapfillfree	int farheapfillfree(unsigned int fillvalue);	heapfillfree для дальньої динамічної пам'яті
farheapcheckfree	int farheapcheckfree (unsigned int fillvalue);	heapcheckfree для дальньої динамічної пам'яті
farheapwalk	int farheapwalk(struct farheapinfo *hi);	heapwalk для дальньої динамічної пам'яті

*Примітка: тип `size_t` визначений так: `typedef unsigned size_t;`

Якщо динамічно виділену пам'ять не звільнити, коли вона стане більше непотрібною, то в системі може виникнути нестача вільної пам'яті.

Робота з динамічними масивами здійснюється через змінну типу вказівник. Після створення масива ця змінна буде вказувати на його початок, тобто буде містити адресу його першого елемента.

Розмір динамічного масива задається не константою, а змінною, значення якої вводить з клавіатури користувач, або воно обчислюється за певними формулами.

Динамічні одновимірні масиви

Динамічні об'єкти не є локальними для певної функції (локальною може бути змінна, що містить адресу такого об'єкта). Тобто динамічний об'єкт може створюватися в одній функції, опрацьовувати його може друга функція (якщо перша повідомить їй адресу об'єкта), а знищувати – третя. Таким чином, доступ до динамічного об'єкта має будь-яка функція програми, якій відома його адреса. Час життя динамічного об'єкта – це час від моменту його створення функцією `malloc` (`calloc`, `new`) до знищення функцією `free` (`free`, `delete`), причому ці функції можуть викликатися як в одній функції програми, так і в різних.

Синтаксис оголошення динамічного одновимірного масива за допомогою функції `malloc` () такий:

```
<тип> *(<ім'я> = (<тип>*) malloc (<кількість_елементів> * sizeof(<тип>));
```

де параметр `<кількість_елементів>` – це кількість елементів, для яких потрібно виділити пам'ять, а параметр `sizeof(<тип>)` – це розмір одного елемента певного типу в байтах. Тоді розмір такого масива визначається як добуток кількості його елементів на розмір одного елемента певного типу в байтах.

Оскільки функція `malloc()` наперед не знає, дані якого типу будуть розміщені у динамічно виділеній області пам'яті, то значення, яке вона повертає, має тип абстрактного вказівника `void*`. А для того, щоб

використовувати цей вказівник як масив, його слід перетворити до відповідного типу за допомогою операції перетворення типу.

Приклад 1. Оголосити динамічний одновимірний масив за допомогою

malloc:

```
#include <iostream>
#include <stdio.h>

using namespace std;

int main() {
    int n;          // кількість елементів масива
    printf("Enter count array's elements n="); scanf("%d", &n);

    // створення динамічного одновимірного масива
    double *p = (double*) malloc(n * sizeof(double));

    // перевірка, чи масив створився
    if (p == NULL) {
        printf("\nMemory error!\n");
        return -1; // якщо масив не створився, виконується вихід з програми
    }
    // введення елементів масива
    printf("\n\nEnter array's elements\n");
    for (int i=0; i<n; i++){
        printf("p[%d]=", i);
        scanf("%lf", &p[i]);
    }

    // подальше опрацювання масива
    ...

    // звільнення пам'яті, виділеної під масив
    free(p);
    return 0;
}
```

Синтаксис оголошення динамічного одновимірного масива за допомогою функції **calloc** такий:

```
<тип> *<ім'я> = (<тип>*) calloc (<кількість_елементів>, sizeof(<тип>));
```

де параметр <кількість_елементів> – це кількість елементів, для яких потрібно виділити пам'ять, а параметр sizeof(<тип>) – це розмір одного елемента певного типу в байтах.

Приклад 2. Оголосити динамічний одновимірний масив за допомогою calloc:

```
#include <iostream>

using namespace std;

int main() {
    int n;          // кількість елементів масива
    cout << "Enter count array's elements n="; cin >> n;
    // створення динамічного одновимірного масива
    double *p = (double*) calloc(n, sizeof(double));
    // перевірка, чи масив створився
    if (p == NULL) {
        cout << "\nMemory error!\n";
        return -1; // якщо масив не створився, виконується вихід з програми
    }
    // введення елементів масива
    cout << "\n\nEnter array's elements" << endl;
    for (int i=0; i<n; i++){
```

```

        cout << "p[" << i << "]=";
        cin >> p[i];
        cout << endl;
    }
    // подальше опрацювання масива
    ...

    // звільнення пам'яті, виділеної під масив
    free(p);
    return 0;
}

```

Синтаксис оголошення динамічного одновимірного масива за допомогою функції `new` такий:

```
<тип_даних> *<ім'я_вказівника> = new <тип_даних> [кількість_елементів];
```

Параметр `[кількість_елементів]` в операторі `new` необов'язковий.

Оператор `new` виділяє місце в пам'яті для певної кількості елементів певного типу, а адреса цієї ділянки пам'яті записується у змінну-вказівник.

Приклад 3. Оголосити динамічний одновимірний масив за допомогою `new`:

```

#include <iostream>

using namespace std;

int main() {
    int n;          // кількість елементів масива
    cout << "Enter count array's elements n="; cin >> n;
    // створення динамічного одновимірного масива
    int *b = new int [n];
    // перевірка, чи масив створився
    if (b == NULL) {
        cout << "\nMemory error!\n";
        return -1; // якщо масив не створився, виконується вихід з програми
    }
    cout << "\n\nEnter array's elements" << endl;
    for (int i=0; i<n; i++){
        cout << "b[" << i << "]=";
        cin >> b[i];
        cout << endl;
    }
    // подальше опрацювання масива
    ...

    // звільнення пам'яті, виділеної під масив
    delete [] b;
    return 0;
}

```

Звертання до значення певного елемента такого масиву виконується за допомогою певного індекса елемента:

а) `b[i]`;

б) `*(b+i)`, що еквівалентно `b[i]`.

Приклад 4. Змінити розмір виділеної пам'яті під одновимірний масив з 10-ти дійсних елементів до 20-ти елементів:

```

int Count=10, newCount=20;
// оголошення та створення одновимірного масива
float *a = (float*) calloc(Count, sizeof(float));
// зміна розміру масива

```

```
a = (float*) realloc(a, newCount);
```

Приклад 5. Увести масив і обчислити за допомогою функції добуток елементів динамічного одновимірного масива.

```
double dob(double *a, int elCount) {
    int i;
    double p=1;
    for(i=0; i<elCount; i++)
        p *= a[i];
}

void main() {
    double P;
    int i, Count;
    cout << "Enter elements count: " << endl; cin >> Count;
    double *a = new double [Count];
    for(i=0; i<Count; i++)
        a[i]=rand()%100+1;
    cout << dob(a, Count);
    delete []a;
}
```

Динамічні двовимірні масиви

Адреси нульових елементів таких динамічних двовимірних масивів зберігаються в допоміжному масиві, пам'ять під який потрібно виділяти завчасно. Оскільки елементами такого масива будуть адреси значень певного типу, тому їхнім типом буде “вказівник на значення певного типу”, тобто `<тип>*`. Виходячи з цього, оголошуючи такий масив слід писати дві зірочки. Після цього з динамічним двовимірним масивом можна працювати як зі звичайним двовимірним масивом, звертаючись до кожного елемента за його індексом, наведеним у квадратних дужках:

```
<ім'я_масива> [<індекс_рядка>][<індекс_стовпчика>]
```

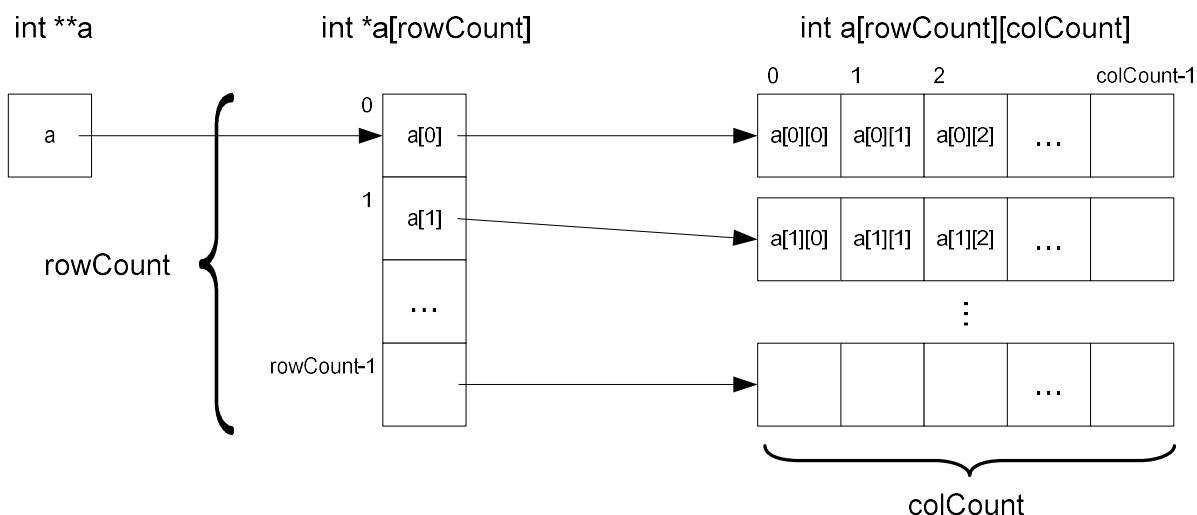


Рис. 1. Розподіл у пам'яті двовимірного масива як двовимірного

Приклад 6. Оголосити динамічний двовимірний масив як двовимірний:

1) за допомогою `malloc`:

```
int rowCount, // кількість рядків
```

```

colCount;    // кількість стовпчиків
cout << "Enter rows count: "; cin >> rowCount;
cout << "Enter columns count: "; cin >> colCount;

// Оголошення та розміщення в пам'яті основного масива
int **m = (int**) malloc(rowCount*sizeof (int*));

// Оголошення та розміщення в пам'яті допоміжного масива
for (int i=0; i<rowCount; i++)
    m[i] = (int*) malloc(colCount*sizeof(int));

// ініціалізація елементів та виведення масива
for (int i=0; i<rowCount; i++) {
    for (int j=0; j<colCount; j++){
        m[i][j]=Low + rand() % (High-Low+1);
        cout << setw(5) << m[i][j];
    }
    cout << endl;
}

//подальше опрацювання масива */
...

// звільнення пам'ті, виділеної під допоміжний масив
for (int i=0; i<rowCount; i++)
    free(m[i]);
// звільнення пам'ті, виділеної під основний масив
free(m);

```

2) за допомогою calloc:

```

// Оголошення та створення (розміщення в пам'яті) основного масива
int **c = (int**) calloc(rowCount, sizeof (int*));
// Оголошення та розміщення в пам'яті допоміжного масива
for (int i=0; i<rowCount; i++)
    c[i] = (int*) calloc(colCount, sizeof(int));
// звільнення пам'ті, виділеної під допоміжний масив
for (int i=0; i<rowCount; i++)
    free(c[i]);
// звільнення пам'ті, виділеної під основний масив
free(c);

```

3) за допомогою new:

```

// Оголошення та створення (розміщення в пам'яті) основного масива
int **n = new int*[rowCount];

// Оголошення та розміщення в пам'яті допоміжного масива
for (int i=0; i<rowCount; i++)
    n[i] = new int[colCount];

// ініціалізація масива
for (int i=0; i<rowCount; i++) {
    for (int j=0; j<colCount; j++){
        n[i][j]=Low + rand() % (High-Low+1);
        cout << setw(5) << n[i][j];
    }
    cout << endl;
}

// подальше опрацювання масива
...

// звільнення пам'ті, виділеної під допоміжний масив
for (int i=0; i<rowCount; i++)

```



```

    delete [] n[i];
// звільнення пам'ті, виділеної під основний масив
delete [] n;

```

При передаванні у функцію багатовимірного масива всі розмірності, якщо вони невідомі на етапі компіляції, мають передаватися як параметри. Наприклад, заголовок функції, яка обчислює суму елементів динамічного двовимірного масива, може мати вигляд

```
int sum(const int **a, const int row, const int col);
```

Виклик цієї функції у програмі може бути таким:

```
cout << "Сума елементів a: " << sum((const int**)a,m,n);
```

Передавання динамічних масивів у функцію

Якщо у функцію передається масив, то насправді у функцію **передається адреса його першого елемента**, тому **масив завжди передається за адресою, а не за значенням**. При цьому інформація про кількість елементів масива втрачається, тому кількість елементів кожної розмірності потрібно передавати за допомогою окремих параметрів.

Якщо кількість елементів масива – стала величина, то її можна оголосити як глобальну константу і використовувати в усіх функціях. Проте, це не є хороший стиль програмування.

Всередині функції масив сприймається як одновимірний, а його індекс переобчислюється (лінійний індекс формується з набору індексів багатовимірного масиву).

Приклад 7. Обчислити суму елементів двох двохвимірних масивів:

```

#include <iostream>
#include <stdlib.h>
int Sum(const int *a, const int rowCount, const int colCount);
using namespace std;
void main(){
    int b[3][3] = {{3, 2, 9}, {2, 7, 6}, {3, 3, 5}};
    cout << "Suma elements of b = " << Sum(&b[0][0], 3, 3)) << endl;
    int i, j, rowCount, colCount, *a;
    cout << "Input rows and columns count:" << endl;
    cin >> rowCount >> colCount;
    // виділення пам'яті для масива
    a = (int *) malloc(rowCount * colCount * sizeof(int));
    // ініціалізація масива
    for (i = 0; i<rowCount; i++)
        for (j = 0; j<colCount; j++)
            cin >> a[i * colCount + j];
    cout<<"Suma elements of a = "<<Sum(a, rowCount, colCount)<<endl;
    // звільнення пам'яті
    free(a);
}
int Sum(const int *a, const int rowCount, const int colCount){
    int i, j, s = 0;
    for (i = 0; i<rowCount; i++)
        for (j = 0; j<colCount; j++)
            s += a[i * colCount + j];
    return s;
}

```

Для того, щоб працювати з двохвимірним масивом звичайним способом, можна застосувати альтернативний спосіб виділення пам'яті. У цьому випадку пам'ять виділяється спочатку під стовпчик вказівників на рядки матриці, а потім у циклі під кожний рядок. Звільнення пам'яті потрібно виконувати у зворотному порядку.

```
#include <iostream>
int sum(int **a, const int rowCount, const int colCount);
using namespace std;
void main() {
    int rowCount, colCount;
    cin >> rowCount >> colCount;
    int **a, i, j;
    // виділення пам'яті для масива
    a = new int* [rowCount];
    for (i = 0; i<rowCount; i++)
        a[i] = new int [colCount];
    // ініціалізація масива
    for (i = 0; i<rowCount; i++)
        for (j = 0; j<colCount; j++)
            cin >> a[i][j];
    cout << sum(a, rowCount, colCount);
    // звільнення пам'яті
    for(j=0; j<colCount; j++)
        delete []a[j];
    delete []a;
}
int sum(int **a, const int rowCount, const int colCount){
    int i, j, s = 0;
    for (i = 0; i<rowCount; i++)
        for (j = 0; j<colCount; j++)
            s += a[i][j];
    return s;
}
```

Приклад 8. Заповнити матрицю розмірності $M \times N$ випадковими числами за допомогою функції.

```
void fillrandom(int** a, int M, int N) {
    for(int i=0; i<M; i++)
        for(int j=0; j<N; j++)
            a[i][j]=rand()%100;
}
void main() {
    int M, N, i, j;
    cout << "Enter rows and columns count: "; cin >> m >> n;
    // виділення пам'яті для масива
    int **a = new int* [m];
    for(i=0; i<m; i++)
        a[i] = new int [n];
    // ініціалізація масива
    fill_random(a, m, n);
    for (int i=0; i<m; i++) {
        for (int j=0; j<n; j++)
            cout << setw(4) << a[i][j];
```

```

        cout << endl;
    }
    for(i=0; i<m; i++)
        delete []a[i];
    delete []a;
}

```

Хід роботи

Завдання 1.

1. Написати програму для виконання певних дій над **динамічним** двовимірним масивом, реалізувавши доступ до елементів масива двома способами: за допомогою індексів та вказівників. Кожний спосіб реалізувати окремою функцією.
2. Алгоритм формування початкового масива реалізувати двома способами: за допомогою введення даних з клавіатури та випадкової генерації чисел. В окремій функції передбачити виділення пам'яті під динамічний масив функціями malloc, calloc та new (під час виконання програми користувач має вибрати, якою з функцій створити масив).
3. Вивести на екран у вигляді матриці елементи початкового масива. Якщо є кілька початкових масивів, то вивести їх у різних рядках. Якщо масив був змінений, то у новому рядку вивести модифікований масив. Якщо потрібно визначити певні індекси, елементи тощо, то вивести їх у новому рядку.
4. **Використання глобальних змінних у підпрограмах не допускається.** Інформація у підпрограми повинна передаватися лише за допомогою параметрів.
5. Введення-виведення даних і виконання певних дій над ними потрібно реалізувати в окремих функціях. У головній програмі потрібно виконувати лише їхній виклик. Введення-виведення даних супроводжувати відповідними повідомленнями.
6. Побудувати блок-схему алгоритму програми та блок-схеми алгоритмів усіх функцій із вказівниковим доступом до елементів масива.

№ з/п	Завдання
1.	Написати програму, яка транспонує квадратну матрицю. Нової матриці створювати не можна.
2.	Написати програму для пошуку максимального елемента бічної діагоналі квадратної матриці.
3.	Написати програму для пошуку найбільший з мінімальних елементів по парних стовпчиках прямокутної матриці.
4.	Написати програму для визначення суми мінімальних елементів по непарних рядках прямокутної матриці.
5.	Написати програму для пошуку найбільшого з мінімальних елементів по непарних рядках прямокутної матриці.
6.	Написати програму для пошуку найменшого з максимальних елементів по всіх рядках прямокутної матриці.

7.	Написати програму для пошуку мінімального елемента головної діагоналі квадратної матриці.
8.	Написати програму для визначення суми елементів бічної діагоналі квадратної матриці.
9.	Написати програму для пошуку максимального елемента головної діагоналі квадратної матриці.
10.	Написати програму для пошуку мінімального елемента бічної діагоналі квадратної матриці.
11.	Написати програму для пошуку найменшого з максимальних елементів по непарних стовпчиках прямокутної матриці.
12.	Написати програму для визначення суми мінімальних елементів по парних стовпчиках прямокутної матриці.
13.	Написати програму для визначення суми максимальних елементів по всіх рядках прямокутної матриці.
14.	Написати програму для визначення суми максимальних елементів по непарних стовпчиках прямокутної матриці.
15.	Написати програму для визначення суми елементів головної діагоналі квадратної матриці.
16.	Написати програму для пошуку найбільшого з мінімальних елементів по всіх рядках прямокутної матриці.
17.	Написати програму, яка впорядковує за зростанням елементи головної діагоналі квадратної матриці.
18.	Написати програму для визначення суми мінімальних елементів по всіх рядках прямокутної матриці.
19.	Написати програму, яка впорядковує за спаданням елементи бічної діагоналі квадратної матриці.
20.	Написати програму для пошуку найменшого з максимальних елементів по парних рядках прямокутної матриці.
21.	Написати програму, яка міняє місцями елементи парних рядків квадратної матриці так, щоб максимальні елементи по парних рядках розмістилися на головній діагоналі.
22.	Написати програму, яка міняє місцями елементи парних стовпчиків квадратної матриці так, щоб максимальні елементи по парних стовпчиках розмістилися на головній діагоналі.
23.	Написати програму, яка міняє місцями максимальний і мінімальний елементи кожного непарного стовпчика прямокутної матриці.
24.	Написати програму, яка міняє місцями максимальний і мінімальний елементи кожного рядка прямокутної матриці.
25.	Написати програму, яка міняє місцями елементи рядків квадратної матриці так, щоб мінімальні елементи по всіх рядках розмістилися на бічній діагоналі.
26.	Написати програму, яка міняє місцями елементи непарних рядків квадратної матриці так, щоб мінімальні елементи по непарних рядках розмістилися на бічній діагоналі.

27.	Написати програму, яка міняє місцями елементи рядків квадратної матриці так, щоб максимальні елементи по всіх рядках розмістилися на головній діагоналі.
28.	Написати програму, яка впорядковує за спаданням елементи головної діагоналі квадратної матриці.
29.	Написати програму, яка впорядковує за зростанням елементи бічної діагоналі квадратної матриці.
30.	Написати програму для визначення суми максимальних елементів по парних рядках прямокутної матриці.

Завдання 2.

1. Написати програму для виконання певних дій над **динамічним** двовимірним масивом, реалізувавши доступ до елементів масива двома способами: за допомогою індексів та вказівників. Кожний спосіб реалізувати окремою функцією.
2. Алгоритм формування початкового масива реалізувати двома способами: за допомогою введення даних з клавіатури та випадкової генерації чисел. В окремій функції передбачити виділення пам'яті під динамічний масив функціями `malloc`, `calloc` та `new` (під час виконання програми користувач має вибрати, якою з функцій створити масив).
3. Вивести на екран у вигляді матриці елементи початкового масива. Якщо є кілька початкових масивів, то вивести їх у різних рядках. Якщо масив був змінений, то у новому рядку вивести модифікований масив. Якщо потрібно визначити певні індекси, елементи тощо, то вивести їх у новому рядку.
4. **Використання глобальних змінних у підпрограмах не допускається.** Інформація у підпрограми повинна передаватися лише за допомогою параметрів.
5. Введення-виведення даних і виконання певних дій над ними потрібно реалізувати в окремих функціях. У головній програмі потрібно виконувати лише їхній виклик. Введення-виведення даних супроводжувати відповідними повідомленнями.
6. Побудувати блок-схему алгоритму програми та функції, що виконує вказані дії над масивом.

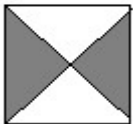
№ з/п	Завдання
1.	Задано двовимірний масив цілих чисел з діапазону (18,...,74). Знайти всі елементи, крім непарних або кратних 7.
2.	Задано двовимірний масив цілих чисел з діапазону (-41,...,48). Знайти всі елементи, крім від'ємних кратних 4.
3.	Задано двовимірний масив цілих чисел з діапазону (5,...,67). Знайти всі елементи, крім парних або кратних 13.

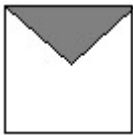

4.	Задано двовимірний масив цілих чисел з діапазону $(-22, \dots, 27)$. Знайти всі елементи, крім додатних або кратних 3.
5.	Задано двовимірний масив цілих чисел з діапазону $(11, \dots, 64)$. Знайти всі парні елементи або, всі крім кратних 6.
6.	Задано двовимірний масив цілих чисел з діапазону $(-21, \dots, 24)$. Знайти всі елементи кратні 3, крім від'ємних.
7.	Задано двовимірний масив цілих чисел з діапазону $(5, \dots, 43)$. Знайти всі елементи кратні 7 або, крім непарних.
8.	Задано двовимірний масив цілих чисел з діапазону $(-41, \dots, 27)$. Знайти всі елементи, крім додатних кратних 3.
9.	Задано двовимірний масив цілих чисел з діапазону $(7, \dots, 62)$. Знайти всі елементи, крім парних або кратних 3.
10.	Задано двовимірний масив цілих чисел з діапазону $(6, \dots, 23)$. Знайти всі елементи, крім парних, хоча б один індекс яких кратний 3.
11.	Задано двовимірний масив цілих чисел з діапазону $(12, \dots, 39)$. Знайти всі, непарні елементи, або всі, крім кратних 7.
12.	Задано двовимірний масив цілих чисел з діапазону $(-26, \dots, 23)$. Знайти всі елементи, крім додатних парних.
13.	Задано двовимірний масив цілих чисел з діапазону $(-29, \dots, 34)$. Знайти всі елементи, крім додатних або кратних 4.
14.	Задано двовимірний масив цілих чисел з діапазону $(-32, \dots, 61)$. Знайти всі елементи, кратні 6, крім від'ємних.
15.	Задано двовимірний масив цілих чисел з діапазону $(7, \dots, 29)$. Знайти всі непарні елементи, крім кратних 3.
16.	Задано двовимірний масив цілих чисел з діапазону $(13, \dots, 57)$. Знайти всі парні елементи, або крім тих, сума індексів яких кратна 6.
17.	Задано двовимірний масив цілих чисел з діапазону $(4, \dots, 45)$. Знайти всі парні елементи, крім тих, хоча б один індекс яких непарний.
18.	Задано двовимірний масив цілих чисел з діапазону $(-38, \dots, 49)$. Знайти всі додатні елементи або ті, обидва індекси яких непарні.
19.	Задано двовимірний масив цілих чисел з діапазону $(-34, \dots, 26)$. Знайти всі додатні або непарні елементи.
20.	Задано двовимірний масив цілих чисел з діапазону $(14, \dots, 93)$. Знайти всі елементи, крім парних з сумою індексів, кратних 13.
21.	Задано двовимірний масив цілих чисел з діапазону $(-6, \dots, 31)$. Знайти всі додатні елементи, крім тих, хоча б один індекс яких кратний 3.
22.	Задано двовимірний масив цілих чисел з діапазону $(-8, \dots, 42)$. Знайти всі додатні елементи, або крім тих, обидва індекси яких парні.
23.	Задано двовимірний масив цілих чисел з діапазону $(-34, \dots, 36)$. Знайти всі додатні елементи, або крім тих, обидва індекси яких кратні 5.
24.	Задано двовимірний масив цілих чисел з діапазону $(-28, \dots, 37)$. Знайти всі додатні елементи і обидва індекси яких парні.
25.	Задано двовимірний масив цілих чисел з діапазону $(4, \dots, 51)$. Знайти всі елементи, крім непарних, або з двома індексами, кратними 3.

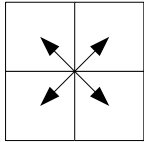
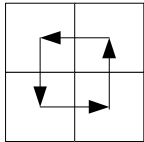
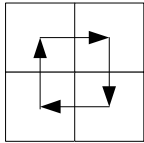

26.	Задано двовимірний масив цілих чисел з діапазону $(-16, \dots, 22)$. Знайти всі елементи кратні 3 або всі, крім від'ємних.
27.	Задано двовимірний масив цілих чисел з діапазону $(-21, \dots, 34)$. Знайти всі від'ємні парні елементи.
28.	Задано двовимірний масив цілих чисел з діапазону $(-25, \dots, 54)$. Знайти всі додатні елементи з сумою індексів, не кратною 5.
29.	Задано двовимірний масив цілих чисел з діапазону $(7, \dots, 62)$. Знайти всі парні елементи, крім тих, сума індексів яких кратна 8.
30.	Задано двовимірний масив цілих чисел з діапазону $(-15, \dots, 17)$. Знайти всі елементи, крім від'ємних або тих, обидва індекси яких непарні.

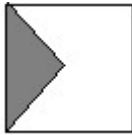

Завдання 3.

1. Написати програму для виконання певних дій над **динамічним** двовимірним масивом, реалізувавши доступ до елементів масива двома способами: за допомогою індексів та вказівників. Кожний спосіб реалізувати окремою функцією.
2. Алгоритм формування початкового масива реалізувати двома способами: за допомогою введення даних з клавіатури та випадкової генерації чисел. В окремій функції передбачити виділення пам'яті під динамічний масив функціями `malloc`, `calloc` та `new` (під час виконання програми користувач має вибрати, якою з функцій створити масив).
3. Вивести на екран у вигляді матриці елементи початкового масива. Якщо є кілька початкових масивів, то вивести їх у різних рядках. Якщо масив був змінений, то у новому рядку вивести модифікований масив. Якщо потрібно визначити певні індекси, елементи тощо, то вивести їх у новому рядку.
4. **Використання глобальних змінних у підпрограмах не допускається.** Інформація у підпрограми повинна передаватися лише за допомогою параметрів.
5. Введення-виведення даних і виконання певних дій над ними потрібно реалізувати в окремих функціях. У головній програмі потрібно виконувати лише їхній виклик. Введення-виведення даних супроводжувати відповідними повідомленнями.
6. Побудувати блок-схему алгоритму програми та функції, що виконує вказані дії над масивом.

№ з/п	Завдання
1.	Обчислити суму модулів всіх елементів квадратної матриці, що знаходяться в зафарбованій частині такої матриці: 

	Розташувати рядки матриці у порядку зростання кількості невід'ємних елементів у рядках виділеної області.
2.	Коефіцієнти системи лінійних рівнянь задані у вигляді прямокутної матриці. За допомогою допустимих перетворень привести систему до трикутного вигляду. Визначити кількість рядків, середнє арифметичне елементів яких менше заданої величини.
3.	Обчислити суму всіх елементів квадратної матриці, що знаходяться в зафарбованій частині такої матриці:  Виконати нормування всіх елементів матриці, що не входять у виділену область.
4.	Виконати циклічне переміщення елементів прямокутної матриці на x елементів вниз, x може бути більше кількості елементів в стовпчику.
5.	Виконати циклічне переміщення квадратної матриці вправо на k елементів таким чином, що елементи першого рядка переміщуються в останній стовпчик зверху вниз, з нього – в останній рядок справа наліво, з нього – в перший стовпець від низу до верху, з нього – в перший рядок; для решти елементів – аналогічно.
6.	Обчислити суму всіх елементів квадратної матриці, що знаходяться в зафарбованій частині матриці:  Розташувати стовпчики заданої матриці відповідно до спадання сум непарних елементів у непарних стовпчиках.
7.	Впорядкувати рядки прямокутної матриці цілих чисел за зростанням кількості однакових елементів у кожному рядку. Визначити номер першого зі стовпчиків, що не містить жодного від'ємного елементу.
8.	Переставити елементи квадратної дійсної матриці так, щоб її максимальний елемент розмістився у позицію (0, 0), наступний за величиною – у позицію (1, 1), наступний за величиною – у позицію (2, 2) і т. д., заповнивши таким чином всю головну діагональ. Визначити номер першого з рядків, що не містить жодного додатного елемента.
9.	Визначити кількість рядків прямокутної матриці, що містять хоча б один нульовий елемент, а також номер стовпчика, в якому знаходиться найдовша послідовність однакових елементів.
10.	У квадратній матриці цілих чисел обчислити суму елементів тих рядків, які не містять від'ємних елементів, а також мінімум серед сум елементів діагоналей, паралельних головній діагоналі матриці.
11.	Визначити кількість від'ємних елементів прямокутної матриці в тих рядках, які містять хоча б один нульовий елемент, а також номери рядків і стовпчиків всіх сідлових точок матриці. Елемент A_{ij} є сідловою точкою, якщо він є мінімальним елементом в i -му рядку та

	максимальним в j -му стовпчику.
12.	<p>Переставити елементи квадратної дійсної матриці так, щоб елемент головної діагоналі $(0, 0)$ перемістився на місце її мінімального елемента, елемент $(1, 1)$ – у позицію наступного за величиною мінімального елемента і т. д., перемістивши таким чином всю головну діагональ. Визначити номер другого з рядків, що не містить жодного нульового елемента.</p>
13.	<p>На основі квадратної матриці цілих чисел отримати нову матрицю, переставляючи її блоки згідно такої схеми:</p>  <p>Визначити кількість рядків матриці, у яких кількість додатних елементів більша за кількість від'ємних.</p>
14.	<p>На основі квадратної матриці цілих чисел отримати нову матрицю, переставляючи її блоки згідно такої схеми:</p>  <p>Визначити кількість від'ємних елементів у кожному непарному рядку парного стовпчика.</p>
15.	Визначити кількість стовпчиків прямокутної матриці, що містять хоча б один нульовий елемент, а також номер рядка, в якому знаходиться найдовша послідовність однакових елементів.
16.	Дана прямокутна матриця цілих чисел. Визначити номер першого зі стовпчиків, що містить хоча б один нульовий елемент. Переставити рядки заданої матриці так, щоб розташувати їх відповідно до спадання сум їхніх від'ємних парних елементів.
17.	<p>На основі квадратної матриці цілих чисел отримати нову матрицю, переставляючи її блоки згідно такої схеми:</p>  <p>Визначити добуток найбільших елементів кожного рядка матриці і їхніх координат (індексів).</p>
18.	<p>Обчислити суму всіх елементів квадратної матриці, що знаходяться в зафарбованій частині такої матриці:</p>  <p>Визначити найменшу суму від'ємних елементів у парних стовпчиках парних рядків виділеної області.</p>
19.	Ущільнити задану прямокутну матрицю, видаливши з неї рядки і стовпчики, заповнені нулями. Визначити номер першого з рядків, що

	містить хоча б один додатній елемент.
20.	Обчислити кількість локальних мінімумів заданої квадратної матриці. Елемент матриці називається локальним мінімумом, якщо він строго менший за всіх його сусідів. Сусідами елемента A_{ij} в матриці є елементи A_{kl} , де $i - 1 \leq k \leq i + 1, j - 1 \leq l \leq j + 1$. Визначити суму модулів елементів, розташованих вище головної діагоналі.
21.	Знайти суму елементів тих стовпчиків прямокутної матриці, які містять хоча б один від'ємний елемент. Переставити стовпчики заданої матриці так, щоб розташувати їх відповідно до зростання сум модулів від'ємних непарних елементів кожного стовпчика.
22.	Визначити кількість рядків прямокутної матриці, що не містять жодного нульового елемента, а також максимальне з чисел, що зустрічаються в заданій матриці більше одного разу.
23.	Визначити кількість стовпчиків прямокутної матриці, що не містять жодного нульового елемента. Переставляючи рядки заданої матриці, розташувати їх відповідно до зростання сум їхніх додатних парних елементів.
24.	Обчислити суму квадратів всіх елементів квадратної матриці, що знаходяться в зафарбованій частині такої матриці:  Визначити кількість парних елементів, які містяться в кожному рядку виділеної області.
25.	Визначити добуток елементів прямокутної матриці в тих рядках, які не містять від'ємних елементів, а також максимум серед сум елементів діагоналей, паралельних головній діагоналі матриці.
26.	Визначити суму елементів прямокутної матриці в тих рядках, які містять хоча б один від'ємний елемент, а також номери рядків і стовпчиків всіх сідлових точок матриці. Елемент A_{ij} є сідловою точкою, якщо він є мінімальним елементом в i -му рядку та максимальним в j -му стовпчику.
27.	Визначити суму елементів прямокутної матриці в тих стовпчиках, які не містять нульових елементів, а також мінімум серед сум модулів елементів діагоналей, паралельних бічній діагоналі матриці.
28.	Для заданої квадратної матриці цілих чисел знайти такі k , що k -ий рядок матриці співпадає з k -им стовпчиком. Обчислити суму елементів тих рядків, які містять хоча б один від'ємний елемент.
29.	Обчислити середньоарифметичну суму всіх елементів квадратної матриці, що знаходяться в зафарбованій частині такої матриці:  Розташувати стовпчики матриці у порядку спадання кількості додатних елементів у стовпчиках виділеної області.

30.	Згладити задану квадратну від'ємну матрицю. Операція згладжування матриці формує нову матрицю того ж розміру, кожен елемент якої рівний середньому арифметичному наявних сусідів відповідного елемента початкової матриці. Сусідами елемента A_{ij} в матриці є елементи A_{kl} , де $i - 1 \leq k \leq i + 1, j - 1 \leq l \leq j + 1$. У згладженій матриці знайти суму модулів елементів, розташованих нижче головної діагоналі.
-----	--