**Лекція 4.** **Використання масивів та покажчиків.**

**Масиви покажчиків**

Покажчики можна групувати в масиви, кожен з елементів яких містить адресу рядка масиву даних у пам’яті. Такий спосіб дозволяє зберігати дані з рваними краями, наприклад, деяку текстову інформацію. Масив з «рваними» краями схожий на двовимірну таблицю, рядки якої можуть мати різну довжину. Використання масиву покажчиків (**char \*fio[ ]**) для збереження рядків дозволяє заощаджувати пам’ять, а процес обробки рядків виконується значно швидше, бо змінюються тільки покажчики, а не вміст рядків.



Рисунок 1. Приклад масиву з "рваними" краями

**Приклад 1**. Виведення даних з масиву з "рваними" краями

**#include <iostream>**

**using namespace std;**

**main ( )**

**{ system("color F0");**

**char \*fio[ ] = { "Petrenko",**

**"Golovko",**

**"Korz",**

**"Kutz",**

**"Ushko",**

**"Plush" }; *//*** *ініціалізація масиву покажчиків*

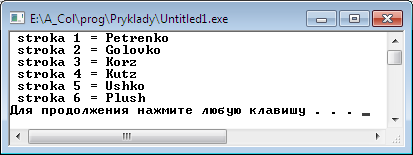
**int str;**

**for (str = 0; str <=5; str++)**

**cout << " stroka " << (str + 1) << " = " << \*(fio + str) << endl;**

**system("pause"); // для затримки екрану виведення результатів**

**}**

Результати виконання програми:  


Особливістю масиву покажчиків є те, що кожний з його елементів може вказувати на масив довільної довжини. Існує можливість записати двовимірний масив чисел і як матрицю, і як одновимірний масив покажчиків.

**int matr[5][7];**  
**або     int \*pmt[5];.**

При цьому двовимірний масив розглядається як одновимірний масив рядків, кожен елемент якого — це теж масив стовпців, тобто масив масивів, тому індексування елементів матриці записується у вигляді **mas[i][j]**.

Звернення до елемента **pm [i][j]** може здійснюватися так:

**\*(pm[i] +j)**  
**або     \*(\*(рm + і) + j) .**

У C++ можна описати змінну, що має **тип «покажчик на** **покажчик»**. Ознакою такого типу є повторення символу «\*» при описі змінної, тобто **int \*\*pmt;**.

Пам’ять для такої змінної не виділяється. Її треба привести до відповідного масиву. При описі покажчик на покажчик можна ініціювати:  
**int х = 20;**  
**int \*px1 = &х;**  
**int \*\*рх2 = &рх1;**  
**int \*\*\*рхЗ = &рх2;.**  
Доступ до змінної **х** здійснюється одним з трьох, способів:  
**\*рх1;**  
**\*\*px2;**  
**\*\*\*рхЗ;.**

Для доступу до пам’яті через покажчики на покажчики використовуються як символи «\*», так і індекси. Наприклад, еквівалентими будуть такі посилання на змінну **х**:  
**\*px1;**               *//рx1 [0];*  
**\*\*рх2;**             *//рх2 [0][0];*  
**\*\*\*рхЗ;**          //рхЗ [0][0][0];.

# Динамічні масиви

В основній пам’яті дані можуть зберігатися двома способами

* пам’ять виділяється або в сегменті стека і залишається закріпленою до завершення виконання конкретної функції, або виділяється в сегменті даних на весь час виконання програми;
* пам’ять виділяється в міру потреби (динамічне виділення  пам’яті).

Слід зазначити, що всі приклади, розглянуті раніше, демонструють роботу з даними, які зберігаються першим способом.

**Динамічна пам’ять** — це вільна пам’ять, у якій під час виконання програми можна виділяти місце залежно від потреб користувача. ***Доступ до виділених ділянок динамічної пам’яті, що називаються динамічними змінними, здійснюється*** ***тільки через покажчики.*** Час існування динамічних змінних — від початку створення до кінця програми або до явного звільнення пам’яті. У мові C++ застосовують два способи роботи з динамічною пам’яттю. Перший з них дістався в спадщину від мови С і використовує сукупність функцій **malloc()**, другий — працює з операціями **new та delete**, які здійснюють динамічний розподіл і скасування з вищим пріоритетом, ніж інші функції.

Оператор **new** виділяє пам’ять і повертає її адресу. За допомогою оператора **delete** відбувається звільнення пам’яті, на яку вказує змінна-покажчик.

Загальна форма запису оператора **new**:

**змінна-покажчик = new тип змінної;.**

Оператор **delete** має вигляд:

**delete [ ] змінна-покажчик;** .

Динамічні масиви створюють за допомогою операції new, при цьому необхідно вказати їх тип і розмірність. Наприклад, Для одновимірного масиву дійсних чисел, що має 100 елементів, треба записати:

**int n = 100;**

**float \*р = new float[n];** — змінна-покажчик на float виділяє у динамічній пам’яті ділянку для розміщення 100 елементів дійсного типу.

***Слід пам’ятати, що динамічні масиви при створенні не можна ні ініціювати, ні обнуляти.***

**Ініціалізація двовимірних масивів.**

Ініціалізація двовимірних та багатовимірних масивів подібна до ініціалізації **одновимірних масивів**. В цьому випадку список ініціалізаторів кожної розмірності береться в фігурні дужки.

Ініціалізація двовимірних (багатовимірних) масивів може бути:

* з задаванням розміру масиву;
* без задавання розміру масиву (“безрозмірна” ініціалізація).

**Приклад 2.** Ініціалізація двовимірного масиву M дійсних чисел розміром 3×4. Задається розмір масиву.

// ініціалізація масиву M дійсних чисел розміром 3\*4

float M[3][4] =

{

{ 0.5, -2.8, -1.0, 23.45 },

{ -2.3, 0.4, 10.5, 0.8 },

{ 12.5, 10.4, 5.4, 3.56 }

};

Якщо у переліку вказати не всі елементи групи, тоді відсутні елементи будуть доповнюватись нульовими значеннями автоматично (рис. 2).

// ініціалізація масиву M дійсних чисел розміром 3\*4

// відсутні елементи доповнюються нулями

float M[3][4] =

{

{ 0.5, -2.8 },

{ -2.3 },

{ 12.5, 10.4, 5.4, 3.56 }

};

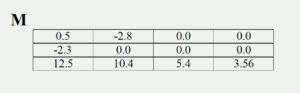
[](https://www.bestprog.net/wp-content/uploads/2017/03/05_02_02_04_02_02_02_.jpg)

Рисунок 2. Доповнення масиву нулями

**Приклад 3.** “Безрозмірна” ініціалізація масиву B цілих чисел.

// "безрозмірна ініціалізація масиву B

int B[][4] =

{

{ 2, -8, 3, 4 },

{ -3, 50, 42, -77 },

{ 11, 25, -30, 4 }

};

**Приклад 4.** “Безрозмірна ініціалізація масиву B з доповненням нулями (рис. 3).

// "безрозмірна ініціалізація масиву B

// з доповненням нулями

int B[][4] =

{

{ 2, -8 },

{ -3 },

{ 11, 25, -30, 4 }

};

[](https://www.bestprog.net/wp-content/uploads/2017/03/05_02_02_04_02_02_03_.jpg)

Рисунок 3. “Безрозмірна” ініціалізація масиву B з заповненням нулями

Другий індекс масиву (стовпці) обов’язково має бути присутній. В іншому випадку компілятор видає повідомлення про помилку.

##### **Ініціалізація масивів рядків**

Масив рядків також є двовимірний масив символів.

Так само як і з іншими типами даних, ініціалізація масиву рядків може бути:

* з задаванням розміру рядка;
* без задавання розміру рядка (“безрозмірна” ініціалізація).

**Приклад 5.** Ініціалізація масиву рядків з задаванням довжини рядка. Описується двовимірний масив рядків розміром 5×60 символів.

char Strings[5][60] =

{

"Text - 1",

"Text - 2",

"Text - 3",

"Text - 4",

"Text - 5"

};

У вищенаведеному прикладі кожен рядок може містити до 60 символів. Якщо довжина рядка менше за 60, то значення інших символів рівне 0 (рисунок 4).

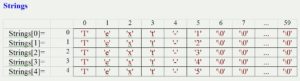
[](https://www.bestprog.net/wp-content/uploads/2017/03/05_02_02_04_02_02_04_.jpg)

Рисунок 4. Представлення двовимірного масиву рядків у пам’яті комп’ютера

**Приклад 6.** “Безрозмірна” ініціалізація масиву рядків

char \* Strings2[] =

{

"String number 1",

"String #2",

"Third string"

};

Вищенаведений код можна записати й іншим чином

char \* Strings2[3] =

{

"String number 1",

"String #2",

"Third string"

};

При “безрозмірній” ініціалізації масиву рядків не потрібно “вручну” підраховувати довжину (кількість символів) рядка. Це здійснюється автоматично на етапі компілювання.

У вищенаведеному прикладі довжина першого рядка рівна 16 символів (рис.º5). Довжина другого рядка рівна 10. Довжина третього рядка рівна 12 символів. В кінці кожного рядка слідує нульовий символ ‘\0’.

Таким чином, використання безрозмірної ініціалізації дозволяє економити пам’ять, виділену під масив.

[C++ представлення рядків рисунок](https://www.bestprog.net/wp-content/uploads/2017/03/05_02_02_04_02_02_05_.jpg)

Рисунок 5. Представлення рядків при “безрозмірній” ініціалізації

##### **Багатовимірні масиви. Приклади опису та використання**

// масив цілих чисел розміром 3\*5\*6

int MI[3][5][6];

MI[0][2][5] = 238;

MI[2][4][3] = -3994;

// масив дійсних чисел розміром 4\*6\*2\*3

float MF[4][6][2][3];

MF[0][0][0][0] = -394.32f;

MF[3][4][1][1] = 4.4f;

MF[1][5][0][2] = 0.0f;

MF[2][2][1][1] = 555.2437f;

##### Ініціалізація багатовимірних масивів. Приклад

// ініціалізація масиву M цілих чисел розміром 2\*3\*4

int M[2][3][4] =

{

{

{ 8, 3, -5, 2 },

{ -3, 2, 8, 4 },

{ 1, 0, 3, -9 }

},

{

{ -2, 9, 5, 4 },

{ 1, 0, 2, 5 },

{ -8, 3, 4, 2 }

}

};

**Робота з двовимірними масивами. Розмістити в двовимірний масив num[3][4] послідовні числа від 1 до 12.**

#include <iostream>

using namespace std;

int main ( )

{ system("color F0");

int t, i, num[3][4];

for(t=0; t<3; ++t)

{

for(i=0; i<4; ++i)

{

num[t][i] = (t\*4)+i+1;

cout << num[t][i] << " ";

}

cout << endl;

}

system("pause");

return 0;

}

Результат роботи програми – заповнення  масиву зображено на рис. 6.

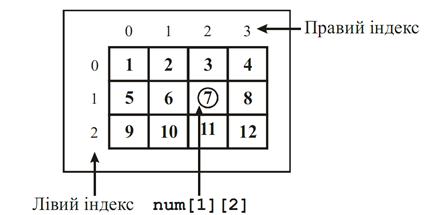
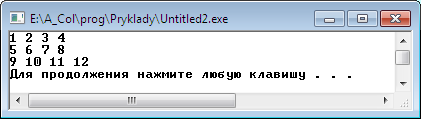


Рисунок 6 Заповнений масив числами від 1 до 12



Для визначення кількості байтів пам'яті, зайнятої двовимірним масивом, використовують формула:

**кількість байтів = кількість рядків × кількість стовпців × розмір типу в байтах**

Наприклад, двовимірний цілочисельний масив розмірністю 10×5 займає в пам'яті 10×5×2, тобто 100 байтів (якщо цілочисельний тип має розмір 2 байт).

## Використання покажчиків на масиви

Пригадаємо. Покажчики — це змінні, котрі містять адресу пам’яті, розподіленої для об’єкта відповідного типу. Змінні зберігають якісь значення (дані). Ці дані можуть бути різних типів: символьного, цілого, дійсного тощо. При оголошенні змінної-покажчика слід вказати тип даних, адресу яких буде містити змінна, та ім’я покажчика з символом «\*».

У мові програмування C++ передбачено два способи доступу до елементів масивів: за допомогою індексування елементів масивів і арифметики покажчиків. Арифметичні операції над покажчиками іноді виконуються швидше, ніж індексування елементів масивів, особливо під час доступу до елементів, розташування яких відрізняється строгою впорядкованістю.

Оскільки швидкодія часто є визначальним чинником при виборі тих або інших рішень під час програмування, то використання покажчиків для доступу до елементів масиву – характерна особливість багатьох С++ програм. Окрім цього, іноді покажчики дають змогу написати дещо компактніший код програми порівняно з використанням індексування елементів масивів.

Покажчик на елемент масиву з індексом х можна записати як: ptr = & array[x];

**Приклад. Робота з покажчиками на масив.**

Обхід елементів масиву за допомогою покажчиків на елементи масиву.

**#include <iostream>**

**using namespace std;**

**int main ( )**

**{ system("color F0");**

**int t, i, num[3][4];**

**int ia[9] = { 0, 1, 1, 2, 3, 5, 8, 13, 21 };**

**int \*pbegin = ia;**

**int \*pend = ia + 9;**

**while ( pbegin != pend ) {**

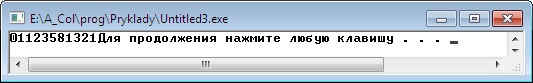
**cout << \*pbegin;**

**++pbegin;}**

**system("pause");**

**return 0;**

**}**



**Приклад.** Виктористання покажчика на масив.

**#include <iostream>**

**using namespace std;**

**void show\_float(float \*array,**

**int number\_of\_elements)**

**{**

**int i;**

**for (i = 0; i < number\_of\_elements; i++) cout << \*array++ << endl;**

**}**

**int main()**

**{**

**system("color F0");**

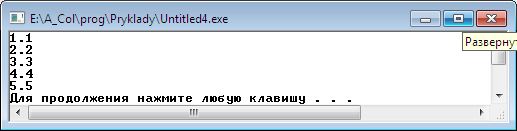
**float values[5] = {1.1, 2.2, 3.3, 4.4, 5.5};**

**show\_float(values, 5);**

**system("pause");**

**return 0;**

**}**

}

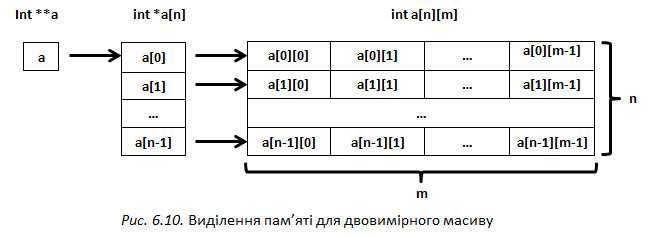
***Слід пам’ятати, що динамічні масиви при створенні не можна ні ініціювати, ні обнуляти.***

Розглянемо універсальний i безпечний спосіб виділення динамічної пам'яті під двовимірний масив, коли обидві його розмірності задаються на етапі виконання програми. Наприклад, розподіл динамічної пам'яті для матриці, що має **n** рядків i **m** стовпців та елементи цілого типу, можна здійснити так:

**int n, m;**  
**cout << ” Введите кількість рядків і стовпців: “;**  
**cin >> n >> m;**  
**int \*\*a = new int \*[n];** /\* оголошення змінної тип «покажчик на покажчик на int» і виділення пам’яті для масиву покажчиків на рядки матриці;\*/

**for (int і = 0; і < n; і++)**  
**a[і] = new int [m];** — кожному елементу масиву покажчиків на рядки присвоюється адреса початку ділянки пам’яті, виділеної для ряд­ка матриці.

Наочно це представлено на наступному рисунку***.***



Звільнення пам’яті з-під масиву будь-якої кількості вимірів виконується за допомогою операції **delete[].**

***Контрольні запитання для самоперевірки***.

1. В чому полягає особливість масиву покажчиків?
2. Як здійснюється "безрозмірна" ініціалізація багатовимірного масиву?
3. В чому проявляються переваги використання покажчиків при роботі з масивами?
4. Які оператори використовують для роботи з динамічними масивами?
5. Які є застереження щодо використання динамічних масивів?
6. Що вказує покажчик на покажчик?

**Для самостійного вивчення** *(4 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції: розглянути приклади, надані після списку літератури. Скомпілювати та виконати приклади, виправивши помилки.

**Література**

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL: <http://www.ph4s.ru/bookprogramir_1.html>
3. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL: <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>

**Завдання для самостійної роботи**

Розібрати приклади**.**

***Приклад 1.*** З використанням динамічної пам’яті обробити вiдомість успішності групи студентів з дисципліни “Програмування” підрахувавши середній бал групи і кількість відмінників.

*/\* использование динамической памяти при обработке одномерного массива \*/*

**#include <iostream.h>**

**#include <conio.h>**

**#include <math.h>**

**main ( )**

**{ const int n = 14;**             *// количество студентов*

**int \*mas;** *// объявление указателя на массив*

**int i, k = 0;***// k — количество отличников*

**float s = 0;**                      *// s — сумма оценок группы*

**mas = new int [n];**             *//выделение динамической памяти*

**cout << " Ввод оценок " << endl;**

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

**сin >> mas[i];**

*//---------------- подсчет суммы оценок и количества отличников*

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

**{ s = s + mas[i];**

**if (mas[i] == 5)**

**k = k+1; }**

**cout << endl;**

**cout.precision(3);**

**cout << "Средний балл = " << s/n << endl;**

**cout << "Количество отличников = " << k << endl;**

**getch ();**

**delete[ ]mas;** *// освобождение динамической памяти*

**}**

Результати обчислень:  
**Ввод оценок**  
**33345343245333**  
**Средний балл = 3.43  
Количество отличников = 2**

У цiй програмi змiнна **s** служить для обчислення суми oцiнок групи, а змiнна **k** — для пiдрахунку кiлькостi вiдмiнникiв. Перед обчисленням треба надати цим змiнним початкове нульове значення.

При створеннi динамiчного багатовимiрного масиву необхiдно в операцiї **new** вказати всi його розмiрностi (перший може бути змiнною), наприклад:  
**int n = 5;** *// n — количество строк*  
**int \*\*m = (int \*\*) new int [n][5];.**

**int n, m;**  
**cout << ” Введите количество строк и столбцов: “;**  
**cin >> n >> m;**  
**int \*\*a = new int \*[n];** — оголошення змiнної тип «покажчик на покажчик на int» i видiлення пам’ятi для масиву покажчикiв на рядки матрицi;  
**for (int i = 0; i < n; i++)**  
**a[i] = new int [m];** — кожному елементу масиву покажчикiв на ряд­ки присвоюється адреса початку дiлянки пам’ятi, видiленої для ряд­ка матрицi.

***Приклад 2.*** З використанням динамiчної пам’ятi створити програму обчислення матрицi С за формулою:

**С [n][q] = A[n][m] \* B[m][q].**

Згiдно з умовою, елемент матрицi **C[n][q]** дорiвнює сумi добуткiв елементiв вiдповiдного рядка матрицi **A[n][m]** на елементи вiдповiдного стовпця матрицi **B[m][q].**

У запропонованiй програмi для обчислення елементiв добутку матриць органiзовано три вкладених цикли: цикл переборi рядкiв матрицi **А**, цикл перебору стовпцiв матрицi **В**, а також цикл накопичення суми для одержання елемента матрицi **С**. Для видiлення динамiчної пам’ятi пiд двовимiрнi масиви **A[n][m], B[m][q] i С [n][q]** скористаємося розглянутим вищi способом.

*/\* Р6\_17.СРР — использование динамической памяти при работе с матрицами \*/*

**#include <iostream.h>**

**#include <math.h>**

**#include <conio.h>**

**main ( )**

**{ int i, j, k;**

**int n, m, q;**

**cout << " Введите размерности матриц: \n";**

**cout << " n = " ;**

**cin >> n;**

**cout << " m = " ;**

**cin >> m;**

**cout << " q = " ;**

**cin >> q;**

*//----------------- создание динамических массивов*

*/\* выделение динамической памяти под массивы указателей и инициализация массивов указателей \*/*

**int \*\*С = new int \*[n];**

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

**C[i] = new int [q];**

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

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

**A[i] = new int [m];**

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

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

**B[i] = new int [q];**

*//------------------------------ ввод матриц A[n][m] и B[m][q]*

**cout<<"\n Ввод матрицы A[n][m] \n";**

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

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

**сin >> \*(А[i] + j);**

**cout << "\n Ввод матрицы B[m][q] \n”;**

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

**for (j = 0; j < q; j++)**

**cin >> \*(B[i] + j);**

*//-----------------------* *вычисление матрицы C[n][q] = A[n][m]\*B[m][q]*

**for** **(i = 0; i < n; i++)  *//*** *перебор строк матрицы A[n][m]*

**for (k = 0; k<q;** **k++)** *//перебор столбцов матрицы B[m][q]*

**{ C[i][k] = 0;**

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

**C[i][k] += (A[i][j]\* B[j][k]);** *// определение элемента C[i][k]*

**}***//-------------------- вывод на экран матрицы С[n][q]*

**cout << "\n Матрица C[n][q] = A[n][m] \* B[m][q] \n";**

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

**{ for (j = 0; j < q; j++)**

**cout << \*(C[i] + j) << " ";**

**cout << endl;}**

**delete [ ]A;**

**delete [ ]B;**

**delete [ ]C;**   *// освобождение памяти*

**getch ();**

**}**

Результати обчислень:  
**Введите размерности матриц:**  
**n = 4**   
**m = З**   
**q = 5**  
**Ввод матрицы A[n][m]**  
**1 1 1**  
**3 3 3**  
**2 2 2**  
**4 4 4**  
**Ввод матрицы B[m][q]**  
**8  8  8  8  8**  
**7  7  7  7  7**  
**5  5  5  5  5**  
**Матрица C[n][q] = A[n][m] \* B[m][q]**  
**20 20 20 20 20**  
**60 60 60 60 60**  
**40 40 40 40 40**  
**80 80 80 80 80**

-------------------------------------------------------------------------------------

***Приклад 3.*** Сформувати матрицю цiлих чисел **С(5,5),** елементи якої обчислюються за формулою **Сij = i+j**. Пiдрахувати добуток елементiв, розташованих нижче побiчної дiагоналi, та обнулити цi елементи. Вивести на екран елементи, розташованi в трикутниках нижче голов­ної та вище побiчної дiагоналей.

Перший варiант програмної реалiзацiї даної задачi передбачає, що матриця описується явним способом i робота ведеться з її елементами.

**#include <iostream.h>**

**const int n = 5;**

**int main()**

**{ int i, j, C[n][n];**

**int Pr;**  *// Pr —     переменная для подсчета произведения*

*//-------------------* *формирование матрицы С[n][n]*

**cout << "------ Матрица С[n][n]: \n";**

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

**{ for (j = 0; j < n; j++)**

**{ C[i][j]=i+j;**

**cout << C[i][j] << " " ;**

**}**

**cout << endl;**

**}**

*/\* вычисление произведения (Pr) элементов ниже побочной диагонали и их последующее обнуление \*/*

**Pr = 1;**       *//начальное значение Рr*

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

**for (j = n - i; j < n; j++)**

**{ Pr\*= C[i][j];**

**C[i][j] = 0; }**

*//-------------- вывод па экран полученных результатов*

**cout << "\n Произведение ниже побочной диагон. = " <<Pr << endl;**

**cout << "\n-------- Преобразованная матрица \n";**

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

**{ for (j = 0; j < n; j++)**

**cout << C[i][j] << " ";**

**cout << endl; }**

*//---------------------- вывод элементов ниже главной диагонали*

**cout << "\n Элементы ниже главной диагонали ";**

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

**{ for (j=0; j<n; j++)**

**if (j < i) cout << C[i][j] << " ";**

**else cout << " " ;**

**cout << endl;**  }

*//--------------------- вывод элементов выше побочной диагонали*

**cout << "\n Элементы выше побочной диагонали \n";**

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

**{ for (j = 0; j < n; j++)**

**if (j < n-1-i) cout << C[i][j] << " ";**

**else cout << " ";**

**cout << endl; }**

**getch();**

**return 0;**

**}**

Результати обчислень:  
**——–Матрица C[n]|n]:**  
**0 1 2 3 4**  
**1 2 3 4 5**  
**2 3 4 5 6**  
**3 4 5 6 7**  
**4 5 6 7 8**  
**Произведение ниже побочной диагон. = 52920000**  
**——– Преобразованная матрица**  
**0 1 2 3 4**  
**1 2 3 4 0**  
**2 3 4 0 0**  
**3 4 0 0 0**  
**0 0 0 0 0**  
**Элементы ниже главной диагонали**  
**1**  
**2 3**  
**3 4 0**  
**4 0 0 0**  
**Элементы выше побочной диагонали**  
**0 1 2 3**  
**1 2 3**  
**2 3**  
**3**

Другий варiант програмної реалiзацiї використовує масив покажчикiв:

**#include <iostream.h>**

**#include <conio.h>**

**const int n = 5;**

**int main()**

**{** **int i, j, Pr, C[n][n], \*pm[n];**

*//---------------- инициализация массива указателей*

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

**pm[i] = &C[i][0];**

*//--------------- формирование матрицы С[n][n]*

**cout << " Матрица C[n][n]"<< endl;**

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

**for(j = 0; j < n; j++)**

**\*(pm[i] + j) = i+j;**

*//------------------ вывод матрицы*

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

**{ for (j = 0; j < n; j++)**

**cout << \*(pm[i] + j) << " ";**

**cout << endl; }**

*/\* вычисление произведения (Pr) элементов ниже побочной диагонали и их последующее обнуление \*/*

**Pr = 1;**

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

**for (j = n-i; j < n; j++)**

**{ Pr\*=\*(pm[i] + j);**

**\*(pm[i]+j) = 0; }**

*//------------ вывод на экран полученных результатов*

**cout << endl <<" Произведение ниже побочной диагон. = " << Pr << endl;**

**cout << endl << " Преобразованная матрица " << endl;**

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

**{ for (j = 0; j < n; j++)**

**cout << \*(pm[i] + j) << " ";**

**cout << endl; }**

*//----------------- вывод элементов ниже главной диагонали*

**cout << endl << " Элементы ниже главной диагонали";**

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

**{ for (j = 0; j < n; j++)**

**if (j < i) cout << \*(pm[i] + j) << ' ';**

**else cout << " ";**

**cout << endl;}**

**cout << endl;**

*//------------ вывод элементов выше побочной диагонали*

**cout << endl << " Элементы выше побочной диагонали " << endl;**

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

**{ for (j = 0; j < n; j++)**

**if (j < n-1-i) cout << \*(pm[i] +j) << ' ';**

**else cout << " ";**

**cout << endl; }**

**getch();**

**return 0;**

**}**

У розглянутiй програмi для виведення матрицi можна ви­користати iнший вигляд оператора виведення, наприклад:  
**cout << ( (j = = 0) ? ‘\t’: ‘ ‘)<< \*(pm[i] +j) << ((j= =n)? ‘\n‘:’ ‘);.**

***Iм’я двовимiрної матрицi є покажчиком-константою на масив покажчикiв-констант, кожний з елементiв якого вка­зуе на початок вiдповiдного рядка матрицi.*** Наприклад, для матрицi **mat[2][2]** маємо:  
**mat [0]** — покажчик-константа на нульовий рядок матрицi;  
**mat [1]** — покажчик-константа на перший рядок матрицi;  
**mat [2]** — покажчик-константа на другий рядок матрицi;,  
тобто:  
**mat [0] = = &mat [0][0];**  
**mat [1] = = &mat [1]|0];**  
**mat [2] = = &mat [2][0];.**  
Виведення матрицi можна реалiзувати з використанням од­ного з наведених нижче операторiв, наприклад:  
**cout << mat[i][j];**  
**cout << \*(mat [i] + j);**  
**cout << \*(\*(mat + i) + j);.**  
***Приклад 4.*** Задана матриця **aij (i = 1…n, j = 1…m) n = 3, m = 4**, необхiдно її парнi елементи переписати до масиву **b**, а непарнi — до масиву **с**.

*/\* формирование массивов четных и нечетных элементов \*/*

*// использование имени массива как указателя на его начало*

**#include <iostream.h>**

**#include <conio.h>**

**const int n = 3, m = 4;**

**void main ()**

{ **int a[n][m], b[m\*n], c[m\*n], i, j, kc = 0, kn = 0;**

// kc и kn — счетчики подсчета четных и нечетних элементов

*//-------------------- ввод исходной матрицы а[n][m]*

**cout << " Vvod massiva a[n][m] " << endl;**

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

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

**cin >> \*(\*(a+i)+j);**

*//------------ формирование массивов b[ ] и c[ ]*

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

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

**if (\*(\*(a+i)+j)%2 == 0) \*(b+kc++) = \*(\*(a+i)+j);**

**else \*(c+kn++) = \*(\*(a+i)+j);**

*//—-------- вывод массива b[]* ***—*** *четных элементов*

**cout << endl << "massiv chetn. elementov — b[ ] " << endl;**

**for (i = 0; i < kc; i++)**

**cout << \*(b+i) << " ";**

**cout << endl;**

*//----------------- вывод массива c[ ]* ***—*** *нечетных элементов*

**cout << endl << "massiv nechetn. elementov — c[ ]" << endl;**

**for (i = 0; i < kn; i++)**

**cout << \*(c+i) << " ";**

**cout << endl;**

*//------------------  вывод исходной матрицы a[n][m] в естественном  виде*

**cout << endl << "ishodny massiv";**

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

**{ cout << endl;**

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

**cout << \*(\*(a+i)+j) << " "; }**

**getch();**

**}**

Результати обчислень:  
**Vvod massiva a[n][m]**  
**8 2 4 -1**  
**6 1 0 -5**  
**2 9 3 -1**  
**massiv chetn. elementov — b[ ]**  
**8 2 4 6 0 2**  
**massiv nechetn. elementov — c[ ]**  
**-1 1 -5 9 3 -1**  
**ishodny massiv**  
**8 2 4 -1**  
**6 1 0 -5**  
**2 9 3 -1.**