**ПР16 (ПР01-2) Розробка програми з передачею масиву за допомогою покажчиків**

**Мета:** навчитися складати програми з функціями, які передають багатовимірні масиви**.**

**Завдання**

1. Ознайомтеся з теоретичною частиною.
2. Написати програму, де створити 2 функції: 1) генерація матриці, заповнену вибірковими числами з діапазону [-10;10]; 2) над згенерованою матрицею виконати індивідуальне завдання, визначене в ПР15 Завдання.docx, де надаються розміри матриці. Матрицю переглядаємо зліва направо та згори вниз. Вивести згенеровану матрицю і матрицю після коригування за завданням або результат роботи. **Параметри передати за допомогою покажчиків**.
3. Результати у вигляді текстового файлу та скріншот виконання надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)

Файл повинен мати назву в такому форматі:

**ОП+АМ <Номер групи><Номер практичної><Прізвище англійською>**

Наприклад, 21-01Ivanov.cpp.

Для відповідей формується текстовий файл з відповідною назвою.

**Тему в заголовку листа записати**

**ОП+АМ <Номер групи>-><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ОП+АМ-Запитання-<Номер групи>-<Прізвище >**.

**Строк відсилки ЛР для ІПЗ-31 22.02.2025**

**ІПЗ-32 22.02.2025**

**ІПЗ-33 22.02.2025**

*Контрольні запитання для самоперевірки*.

1. Що ми називаємо функцією?
2. Які змінні називаються локальними?
3. Як задаються в функціях значення, що повертаються?
4. Які параметри називаються формальними, а які фактичними?
5. Розкажіть про порядок дій, які виконуються під час виклику функції.
6. Чи можна у викликаємій функції змінювати параметри при **виклику функції з передачею значень? Чи вплине така зміна значень параметрів на дії в головній функції, якщо ці параметри потім там використовуються?**
7. **Які правила встановленні для формування списку параметрів, коли** передача даних виконується за замовчуванням?

**ТЕОРЕТИЧНА ЧАСТИНА**

**Багатомірний масив як параметр функції**

На жаль, у мові С++ немає способу, що дозволив би уникнути явного завдання розміру масиву, переданого функції. Тому передавати масив без завдання кількості рядків і стовпців неможливо. Наприклад, при передачі функції двомірного масиву необхідно визначити хоча б кількість стовпців, що задається як глобальна змінна, у той час як кількість рядків передається як параметр. Утім, якщо програміст візьме на себе обов'язок представити двомірний масив у вигляді одномірного, можна скористатися способом, описаним у попередньому розділі.

Передача двовимірного масиву

#include <iostream.h>

const int COLUMNS = 2;

void twiceArray(int [][COLUMNS], int rows);

int main()

{

int actual[][COLUMNS]={1,2,3,4};

int rows = 2;

twiceArray(actual, rows);

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

{

for(int j=0; j<rows; j++) cout << actual[i][j] << " ";

cout << endl;

}

return 0;

}

void twiceArray(int formal[][COLUMNS], int rows)

{

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

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

formal[i][j] \*=2;

}

**Адресна арифметика**

До вказівників (покажчиків) можна застосувати деякі операції. Арифметика вказівників обмежується додаванням, відніманням і порівнянням. Під час виконання арифметичних операцій з вказівниками передбачається, що вказівники вказують на масив об'єктів. Додавши ціле значення до вказівника, ми переміщуємо його на відповідне число об'єктів у масиві. Якщо, наприклад, тип має розмір 10 байтів, а потім ми додали ціле число 5, вказівник переміщується на 50 байтів в пам'яті. Можна застосувати інкремент і декремент, а також операції складеного присвоювання для неконстантних вказівників.

**int** a[5] = { 1, 2, 4, 8, 16 };

**int** \*p = a; // p вказує на a[0]

cout << \*(a + 3) << endl; // 8

p++; // p вказує на a[1]

cout << \*p << endl; // 2

p += 3; // p вказує на a[4]

cout << \*p << endl; // 16

p--; // p вказує на a[3]

cout << \*p << endl; // 8

cout << \*(p - 2) << endl; // 2

a++; // Помилка! a є константним вказівником

Різниця між двома вказівниками на різні елементи масиву повертає кількість елементів, які розташовані між цими вказівниками (включаючи перший і не включаючи останній). Наприклад,

**int** a[5] = { 1, 2, 4, 8, 16 };

**int** \*p1 = a; // p1 вказує на a[0]

**int** \*p2 = a + 3; // p2 вказує на a[3]

cout << p2 - p1 // 3;

Різниця між двома вказівниками має сенс тільки тоді, коли обидва вказівники вказують на елементи одного масиву.

Перевірка виходу за межі масиву не здійснюється. У наведеному нижче прикладі р вказує на елемент, який не існує:

**int** a[5] = {1, 2, 4, 8, 16};

**int** \*p3 = a + 5; // Такого елемента немає

Розіменування p3 є небезпечним.

**Цикли з використанням вказівників**

Замість циклів, побудованих на використанні індексів, можна як параметр використовувати вказівник. Це може трохи підвищити ефективність програми.

У наведеному нижче прикладі ми здійснюємо обхід масиву для заповнення і виведення за допомогою арифметики адрес:

#include <iostream>

**int** main()

{

**const int** n = 4;

**double** a[n] = { };

**double** item = 1;

**for** (**double**\* p = a; p < a + n; p++)

{

\*p = item;

item \*= 2;

}

**for** (**const double**\* p = a; p < a + n; p++)

{

std::cout << \*p << " "; // 1 2 4 8

}

}

Модифікатор **const** перед описом вказівника не є обов'язковим, але рекомендованим у випадках, коли розіменований елемент масиву застосовують тільки для читання.

Інший приклад демонструє проходження елементів масиву у зворотному порядку:

#include <iostream>

**int** main()

{

**double** a[] = { 1, 2, 4, 8 };

**int** n = **sizeof**(a) / **sizeof**(**double**);

**double** sum = 0;

**for** (**const double** \*p = a + n - 1; p >= a; p--)

{

std::cout << \*p << " ";

}

}

Так можна знайти суму елементів з парними індексами:

#include <iostream>

**int** main()

{

**const int** n = 4;

**double** a[] = { 1, 2, 4, 8};

**double** sum = 0;

**for** (**double** \*p = a; p < a + n; p += 2)

{

sum += \*p;

}

std::cout << sum; // 5

}

**Використання вказівників для передачі масивів у функції**

Вказівники можна використовувати для опису параметра функції типу масиву. Саме вказівники зазвичай використовують замість масивів під час опису параметрів. Наприклад:

#include <iostream>

**double** sum(**double** \*arr, **int** n)

{

**double** result = 0;

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

{

result += arr[i];

}

**return** result;

}

**int** main()

{

**double** a[] = { 1, 2, 4, 8, 16 };

std::cout << sum(a, 5) << std::endl; // 31

std::cout << sum(a, 4) << std::endl; // 15

**return** 0;

}

Вихідний масив може бути змінений всередині функції:

**void** modifyStartingElement(**int** \*p)

{

p[0] = 0;

}

**void** main()

{

**int** a[] = {1, 2, 3};

modifyStartingElement(a);

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

{

cout << a[i] << ' '; // 0 2 3

**}**

}

Для того, щоб випадково не змінити значення елементів масиву, його можна передавати через вказівник на константний об'єкт:

**void** f(**const int**\* p)

{

p[0] = 0; // Помилка компіляції

...

}

Завдяки вказівникам можна передавати багатовимірні масиви як параметри. Але для забезпечення гнучкості слід описувати такі параметри як вказівники на дані (одновимірний масив). Під час виклику функції необхідно явно перетворювати тип. Наприклад, наведена нижче функція може бути використана для обчислення суми всіх елементів двовимірного масиву з m рядків і n стовпців. Реалізація функції передбачає обчислення індексу з урахуванням розташування елементів двовимірного масиву:

**double** sum(**double\*** a, **int** m, **int** n)

{

**double** result = 0;

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

{

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

{

result += a[i \* n + j];

}

}

**return** result;

}

**int** main()

{

**double** arr[][3] = { { 1, 2, 3 },

{ 4, 5, 6 } };

cout << sum((**double**\*) arr, 2, 3) << endl; // явне перетворення типів

**return** 0;

}

ПРИКЛАД

### Знаходження суми матриць

Ми працюватимо з масивами в динамічній пам'яті (масивами вказівників). Користувач повинен ввести розміри матриць. Вихідні матриці заповнюватимуться псевдовипадковими числами. Окремі дії виконуватимуться в різних функціях. Програма буде такою:

#include <cstdio>

#include <iostream>

**using namespace** std;

// Повертає дійсне значення в діапазоні від 0 до 100

**double** nextRandom()

{

**return** (rand() % 10000) / 100.;

}

// Здійснює читання з клавіатури m і n

// з перевіркою допустимості

**bool** readData(**int**& m, **int**& n)

{

cout << "Enter m and n: ";

cin >> m >> n;

**if** (m <= 0 || n <= 0)

{

cout << "Wrong values";

**return false**;

}

**return true**;

}

// Створює двовимірний масив необхідних розмірів

// та повертає вказівник на нього

**double**\*\* createMatrix(**int** m, **int** n)

{

**double**\*\* matrix = **new double**\* [m];

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

{

matrix[i] = **new double**[n];

}

**return** matrix;

}

// Заповнює масив випадковими значеннями

**void** fillMatrix(**double**\*\* matrix, **int** m, **int** n)

{

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

{

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

{

matrix[i][j] = nextRandom();

}

}

}

// Виводить елементи масиву на екран

**void** showMatrix(**double**\*\* matrix, **int** m, **int** n)

{

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

{

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

{

cout << matrix[i][j] << "\t";

}

cout << endl;

}

cout << endl;

}

// Знаходить та повертає суму матриць

**double**\*\* sumOfMatrices(**double**\*\* a, **double**\*\* b, **int** m, **int** n)

{

**double**\*\* c = createMatrix(m, n);

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

{

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

{

c[i][j] = a[i][j] + b[i][j];

}

}

**return** c;

}

// Видаляє масив з динамічної пам'яті

**void** deleteMatrix(**double**\*\* matrix, **int** m)

{

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

{

**delete**[] matrix[i];

}

**delete**[] matrix;

}

**int** main()

{

**int** m, n;

**if** (readData(m, n))

{

**double** \*\*a = createMatrix(m, n);

fillMatrix(a, m, n);

showMatrix(a, m, n);

**double**\*\* b = createMatrix(m, n);

fillMatrix(b, m, n);

showMatrix(b, m, n);

**double** \*\*c = sumOfMatrices(a, b, m, n);

showMatrix(c, m, n);

deleteMatrix(a, m);

deleteMatrix(b, m);

deleteMatrix(c, m);

**return** 0;

}

**return** 1;

}

Як видно з коду, циклі, побудовані на діапазоні, не можна використовувати.