ЛАБОРАТОРНА РОБОТА

Технології та методології програмування за допомогою динамічних масивів.

Мета роботи

- Засвоїти технологію та методологію програмної обробки на мові C/C++ динамічних двовимірних масивів.
- Засвоїти способи виділення пам'яті для динамічних масивів.
- Навчитись будувати математичну модель задачі.
- Засвоїти методи перевірки правильності вхідних даних.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Взагалі кажучи, ім'я масиву є *константним вказівником*, який ініціалізовано базовою адресою. Таким чином, масиви і вказівники використовуються для однієї мети: доступу до пам' яті. Різниця полягає в тому, що вказівник є змінною, яка приймає в якості значення адресу комірки пам' яті. А ім' я масиву може розглядатися як константний вказівник з фіксованою (базовою) адресою (Лекція 1_12).

```
int mas[4];
int* ptr = mas;
```

Таблиця 1.

Адреса	&mas[0] або ptr	&mas[1] або ptr+1	&mas[2] або ptr+2	&mas[3] або ptr+3
Значення	mas[0] або *ptr	mas[1] або *(ptr+1)	mas[2] або *(ptr+2)	mas[3] або *(ptr+3)

Таким чином, в даному випадку записи ptr+i та &mas[i] рівносильні, де `i` — деяке зміщення (ціле число) від адреси ptr або &mas (в даному випадку `i`=0..3). Відмінність полягає в тому, що значення ptr+i змінювати можна, а `&mas[i]` — ні. Наступні вирази є некоректними :

```
mas = ptr;
++mas;
mas = mas + 3;
```

Для отримання значення елементу масиву через вказівник ptr необхідно використати операцію розіменування. Розглянемо два способи знаходження суми елементів деякого масиву :

```
const int N = 10; int mas[N];
int* ptr = mas;  // або ptr = &mas[0]

//  1 варіант:  через вказівник ptr
 int sum1 = 0;
            for   (ptr = mas;  ptr < &mas[N]; ++ptr) sum1 += *ptr;
```

```
// 2 варіант:  з використанням індексів
int sum2 = 0;
for (int i = 0; i < N; ++i)
sum2 += mas[i];
// рівносильно sum2 += *(mas + i);
```

Аналогічна адресна арифметика використовується при використанні масивів більшої розмірності. Розглянемо оголошення двовимірного масиву:

```
int mas[4][2];  // матриця розміром 4 на 2
int *ptr;
```

Тоді вираз ptr=mas вказує на перший стовпець першого рядка матриці mas. Записи mas і &mas[0][0] рівносильні. Тоді вираз ptr+1 вказуватиме на mas[0][1], далі йдуть елементи: mas[1][0] , mas[1][1] , mas[2][0] і т. д.; ptr+5 вказуватиме на mas[2][1]. Тобто двовимірні масиви розташовані в пам'яті так само, як і одновимірні масиви, займаючи послідовні комірки пам'яті:

Таблиця 2

Адреса	ptr	ptr+1	ptr+2	ptr+3	ptr+4	ptr+5	...
Значення	mas[0][0]	mas[0][1]	mas[1][0]	mas[1][1]	mas[2][0]	mas[2][1]	...

Слід зауважити, що розмірність статичного масиву є константою і не може визначатися під час виконання програми.

Динамічні масиви

Змінні у програмах повинні розміщатися в одному з трьох місць: в області даних програми, в області стеку, в області вільної пам'яті (купи).

Кожній змінній у програмі може відводитися пам'ять або статично (в момент завантаження), або динамічно (у процесі виконання програми).

До цих пір всі використовувані масиви оголошувались статично, а отже, зберігали значення своїх елементів в області даних. Якщо кількість елементів невелика, таке розміщення виправдано. Але досить часто виникають випадки, коли необхідно мати великі масиви даних або розмір масиву заздалегідь не може бути визначений. Тут на допомогу приходить можливість використання динамічної пам'яті.

Динамічним називається масив, розмірність якого стає відомою в процесі виконання програми.

Для того, щоб у пам'яті можна було розмістити будь-який динамічний об'єкт, для нього необхідно попередньо виділити відповідне місце. По завершенні роботи з об'єктом виділену пам'ять необхідно звільнити.

Виділення пам'яті можна здійснювати двома способами.

I-й спосіб: функції ***malloc(), calloc(), free().***

Ці функції описані у заголовочному файлі <*stdlib.h*> або <*malloc.h*>

Функція *malloc(size)* виділяє *size* байтів з купи. У випадку успішного виділення пам'яті покажчик встановлюється на виділений блок пам'яті. При невдалому виділенні пам'яті функція повертає *NULL*.

Функція *calloc(num, size),* окрім виділення області пам'яті під масив об'єктів, ще здійснює ініціалізацію елементів масиву нульовими значеннями. Тут *num* вказує, скільки елементів буде зберігатися у масиві, а *size* - розмір кожного елемента у байтах.

Наприклад, якщо необхідно виділити пам'ять для масиву з *n* цілих чисел типу long, це можна зробити за допомогою оператора:

```
long * fptr =  (long *)malloc(n*sizeof(long));
```

Якщо необхідно виділити пам'ять під двовимірний масив розмірності *mxn,* це можна зробити за допомогою оператора

```
float* fptr = (float*)malloc(n*m*sizeof(float));
```

Функція *free(*block)* звільняє пам'ять, на яку вказує покажчик *block.*

II-й спосіб: функції ***new(), delete().***

Ці функції з'явилися в C++. *malloc(), calloc(), free()* працюють і в C, і в C++. Нові оператори гнучкого розподілення пам'яті *new()* і *delete()* мають додаткові можливості.

Якщо оператори *malloc(), calloc()* повертають пустий покажчик, який далі перетворюється до потрібного типу, то оператор *new()* повертає покажчик на той тип, для якого виділяється пам'ять, і додаткових перетворень не потребує.

Оператори new та delete використовуються для керування вільною пам'яттю. Вільна пам'ять (або куча, heap) — це область пам'яті, яка надається системою для розміщення об'єктів, час життя яких напряму керується програмістом. За допомогою оператора new виділяється пам'ять під динамічний об'єкт (який створюється в процесі виконання програми), а за допомогою оператора delete створений об'єкт видаляється з пам' яті. Оператора new має наступний синтаксис.

```
new ім'я типу;
new ім'я типу ініціалізатор;
new ім'я типу[вираз];
```

В результаті виконання оператору new в пам'яті виділяється об'єм пам' яті, який необхідний для зберігання вказаного типу, і повертається базова адреса. Якщо пам'ять недоступна, оператор new повертає значення 0 , або генерує виключення.

Оператор delete має наступний формат:

```
delete вираз; delete[] вираз;
```

Розглянемо виділення пам'яті під динамічний масив. Нехай розмірність динамічного масиву вводиться з клавіатури. Спочатку необхідно виділити пам'ять під цей масив, а потім створений динамічний масив необхідно вилучити з пам'яті. Це можна зробити наступним чином.

```
int n;  // n — розмірність масиву
cin >> n;   // вводимо з клавіатури
int* mas = new int[n]; // виділення пам'яті під динамічний масив
```

```
delete[]  mas;   // звільнення пам'яті
```

В цьому прикладі змінна mas є вказівником на масив з n елементів. Вираз int* mas = new int[n] виконує дві дії: оголошується змінна типу вказівника на int, далі вказівнику надається адреса виділеної області пам'яті у відповідності з заданим типом об'єкта.

Для цього ж прикладу можна задати наступну еквівалентну послідовність операторів:

```
int n, *mas;  // n - розмірність масиву,  mas -вказівник на тип int
cin >> n;
mas = new int[n];   // виділення пам'яті під масив
delete[]  mas;   // звільнення пам'яті
```

Оператор delete[] mas використовується для звільнення виділеної пам'яті.

Приклад використання динамічного двовимірного масиву

Задача . Заповнити прямокутну матрицю випадковими числами з інтервалу (a, b). Створити матрицю, що містить стільки ж стовпців, як і вхідна матриця, і 3 рядки: 1-й рядок - максимальні значення відповідних стовпців, 2-й - середні арифметичні значення елементів стовпця, 3-й рядок - мінімальні значення відповідних стовпців.

Математична модель задачі.

Нехай маємо матрицю A розмірністю n x m.

$$A = \begin{array}{|c|c|c|c|c|} \hline A_{00} & A_{01} & A_{02} & \dots & A_{0,m-1} \\ \hline A_{10} & A_{11} & A_{12} & \dots & A_{1,m-1} \\ \hline A_{20} & A_{21} & A_{22} & \dots & A_{2,m-1} \\ \hline \dots & \dots & \dots & \dots & \dots \\ \hline A_{n-1,0} & A_{n-1,1} & A_{n-1,2} & \dots & A_{n-1,m-1} \\ \hline \end{array}$$

Необхідно отримати матрицю B:

$$B = \begin{array}{|c|c|c|c|c|} \hline B_{00} & B_{01} & B_{02} & \dots & B_{0,m-1} \\ \hline B_{10} & B_{11} & B_{12} & \dots & B_{1,m-1} \\ \hline B_{20} & B_{21} & B_{22} & \dots & B_{2,m-1} \\ \hline \end{array}$$

$B_{0j} = \max\{A_{ij}, i=0, \dots,n-1\}$ - максимальне значення серед елементів j-ого стовпця (j=0,.. .,m-1);

$B_{1j} = \text{avg}\{A_{ij}, i=0, n-1\} = (A_{0j}+A_{1j}+...+A_{n-1}j)/n$ - середнє значення елементів стовпця (j=0,.,m-1);

$B_{2j} = \min\{Aij, i=0, ..., n-1\}$ - максимальне значення серед елементів стовпця (j=0,.,m-1).

Формалізація завдання

1. Вхідні дані: n, m - кількість рядків і стовпців матриці A відповідно, a, b - границі інтервалу для елементів матриці.

2 Вихідні дані: матриця B, що містить максимальні, середні, мінімальні значення стовпців.

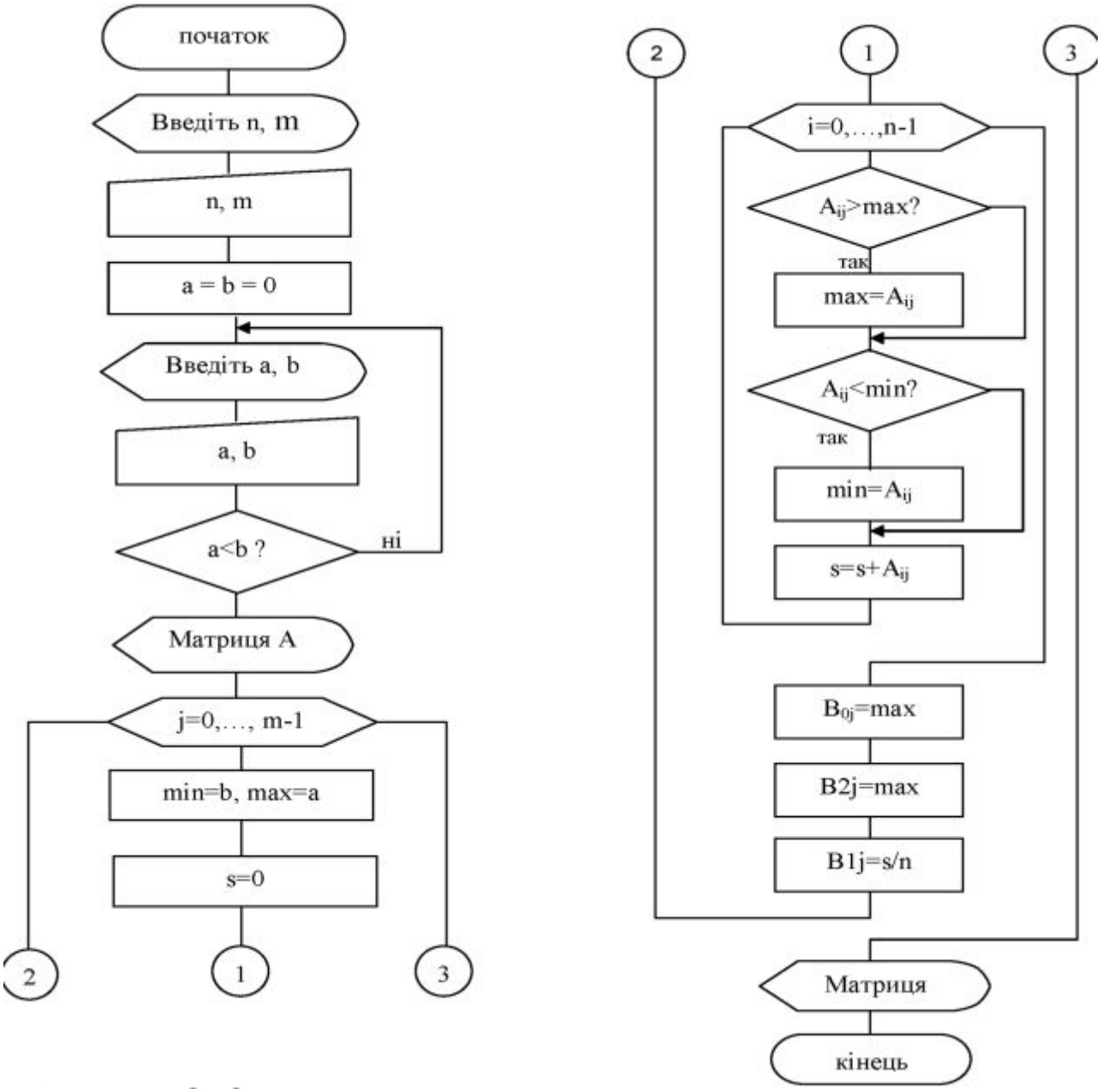
3. Розмірність матриці задається у процесі роботи програми, отже, для неї пам'ять будемо виділяти динамічно (з купи).

4. Границі інтервалу (a, b) вводимо з клавіатури, елементи матриці заповнюємо за допомогою генератора випадкових чисел.

5. Матрицю B доцільно оголосити дійсною, оскільки серед її елементів будуть значення середнього арифметичного, а це - результат розрахунку. Пошук мінімального і максимального елементів будемо здійснювати одночасно, перебираючи і порівнюючи усі елементи стовпця по черзі (використовуємо оператор циклу).

6. Для знаходження середнього арифметичного значення треба накопичувати суму елементів стовпця, що також можна робити у тому ж сам циклі.

Схема роботи програми



Лістинг програми

```
#include <iostream>
#include <conio.h>
#include <new.h>
#include <locale.h>
using namespace std;
int main()
{ setlocale(0,"");
    int n, m, a=0, b=0;
    cout<<" \n Уведіть розмірності масиву n і m:\n"; cin>>n>>m;
    cout<<"\n n="<<n<<" m="<<m;
    while (a>=b)
    {           //перевірка правильності
       cout<<" \n\n  Уведіть границі інтервалу a і b:\n"; cin>>a>>b;
       cout<<"\n a="<<a<<" b="<<b;
     }
     int** arr = new int*[n];
     cout<<"\n\n  Початковий  масив:\n";
     for (int i=0;i<n;i++)
        { arr[i] = new int[m];
          cout<<"\n";
          for (int j=0;j<m;j++)
            {arr[i][j]=rand()%(b-a)+a;
                printf("%7d",arr[i][j]);
            }
        }
     float** brr = new float*[3];
     for (int i=0;i<3;i++)
          brr[i] = new float[m];
      for (int j=0;j<m;j++)
      {  // по стовпцях
           int min=b; // початкове значення мінімуму у стовпці
                int max=a;  // початкове значення максимуму  у стовпці
                float s=0; // для накопичення суми у сер. арифм.у стовпці
                for (int i=0;i<n;i++)
                {
                  if (arr[i][j]>max) max = arr[i][j];
                  if (arr[i][j]<min) min = arr[i][j];
                  s += arr[i][j];
                }
                s = s/n;    // обчислюємо сер.арифм.
                brr[0][j] = max; brr[1][j] = s; brr[2][j] = min;
        }
        cout<<"\n\n  Результуючий масив:\n";
        for (int i=0;i<3;i++)
```

```
{ cout<<"\n";
  for (int j=0; j<m; j++)
    if (i==1) printf("%7.2f",brr[i][j]); else printf("%7.0f",brr[i][j]);
  }
delete[] arr;            delete[] brr;
_getch();
return 0;
}
```

Результат роботи програми

```
 Введіть розмірності масиву n і m:
5
10

 n = 5    m = 10

 Введіть границі інтервалу a і b:
-50
100

 a = -50    b = 100

 Початковий  масив:

   -9   -33   -16    50    69    74    28    58    62   -36
  -45    45   -19   -23    11    -9    95    42   -23   -14
   91     4   -48   -47    92    32   -29    66    18    45
   -3    76    21    88    19    62   -33    -1    35    94
    3    61    72   -17    73   -36    91    11     3    68

 Результуючий масив:

   91    76    72    88    92    74    95    66    62    94
 7,40 30,60  2,00 10,20 52,80 24,60 30,40 35,20 19,00 31,40
  -45   -33   -48   -47    11   -36   -33    -1   -23   -36
```

Рекомендації

1. Динамічне виділення пам'яті для одновимірного масиву *A* (цілих чисел) розміром *n* і заповнення елементів масиву випадковими числами:

```
int* A = new int[n];
for (int i=0;j<n;j++)
      A[i]=rand()% RAND_MAX+1;
```

2. Динамічне виділення пам'яті для двовимірного масиву *arr* (цілих чисел) розміром *n* x *m* та його заповнення випадковими числами:

```
int** A = new int*[n];
for (int i=0;i<n;i++)
{ arr[i]  = new int[m];
  for (int j=0;j<m;j++)
      arr[i][j]=rand()%RAND_MAX+1;
}
```

Порядок виконання роботи

1. Ознайомитись з теоретичним матеріалом.
2. Дослідити процес реалізації завдань прикладів, відлагодити наведені програму на своєму комп'ютері.
3. Розробити власні програми, які реалізує індивідуальне завдання.
4. Підготувати звіт, який включатиме:

- варіант і текст індивідуального завдання;
- формалізацію завдання (включаючи математичну модель задачі);
- схему роботи програми;
- лістинг програми;
- роздруківку трьох контрольних прикладів (вигляд екрану з результатами (різні значення вхідних даних);
- висновки.

Варіанти індивідуальних завдань

Задача . Розробити програму, дотримаючись таких вимог: використовувати динамічні масиви; максимальні розміри масиву *(N і M)* - статичні константи; реальні розміри масиву *n і m (n<N, m<M)* - ввести з клавіатури (при цьому здійснювати перевірку правильності введення даних); елементи масиву - псевдовипадкові числа, згенеровані на інтервалі [*a*, b], де *a* і *b (a<b)* вводяться з клавіатури; усі вхідні дані і елементи вхідного і вихідного масивів виводяться на екран у зручному для перегляду вигляді. Підготувати звіт у повному обсязі.

1	Реалізувати програму, яка міняє місцями перший і останній стовпці квадратної матриці.
2	Реалізувати програму, яка додає перший і останній рядки квадратного масиву і записує результат у останній стовпець.
3	Реалізувати програму, яка міняє значення елементів квадратної матриці ні значення відповідних елементів заданого одновимірного масиву.
4	Реалізувати програму, яка додає відповідні елементи двох заданих масивів і заносить результат у третій масив. Усі три масиви мають однакові розмірності *(nxm)*.
5	Реалізувати програму, яка міняє місцями перший рядок і останній стовпець квадратної матриці.
6	Реалізувати програму, яка міняє місцями діагоналі квадратної матриці.
7	Реалізувати програму, яка сумує елементи рядків двовимірного масиву і заносить результат в одновимірний масив, розмірність якого дорівнює числу рядків двовимірного масиву.
8	Реалізувати програму, яка знаходить максимальний за модулем елемент заданого двовимірного масиву.
9	Реалізувати програму, яка міняє місцями останній рядок і перший стовпець квадратної матриці.
10	Реалізувати програму, яка додає перший і останній стовпці квадратної матриці і записує результат на місце першого рядка.
11	Реалізувати програму, яка міняє елементи заданого стовпця на значення відповідних елементів одновимірного масиву.
12	Реалізувати програму, яка перемножує відповідні елементи двох заданих масивів і заносить результат у третій масив. Розмірності усіх масивів однакові.
13	Реалізувати програму, яка міняє місцями останній рядок і перший стовпець квадратної матриці.

14	Реалізувати програму, яка сумує елементи стовпців двовимірного масиву і зносить результат в одновимірний масив, розмірність якого дорівнює числу стовпців двовимірного масиву.
15	Реалізувати програму, яка знаходить номер рядка заданого двовимірного масиву, що має максимальну за модулем суму елементів.

Контрольні питання

1. В чому різниця між статичними та динамічними масивами?
2. Коли і звідки виділяється пам'ять для статичних і динамічних масивів?
3. Як оголосити одновимірний динамічний масив?
4. Як оголосити динамічний двовимірний масив?
5. Наведіть фрагмент коду для заповнення масиву цілими (дійсними) випадковими числами.
6. Наведіть фрагмент коду для обчислення суми (добутку) елементів одновимірного масиву?
7. Наведіть фрагмент коду для підрахунку суми (добутку) елементів певного рядка (стовпця) двовимірного масиву?