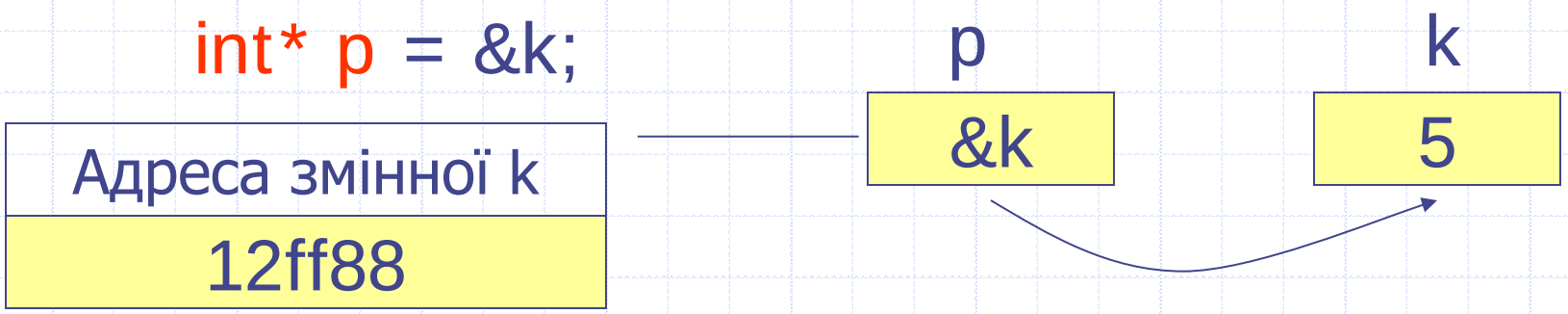
Вказівники, посилання, масиви

Проф. Куссуль Н.М.

Вказівники

- **Вказівник** - це похідний тип даних, що містить адресу змінної певного типу.
  - Для типу T тип T* – вказівник на T
  - Змінна T* містить адресу об'єкту T
- **Приклад**

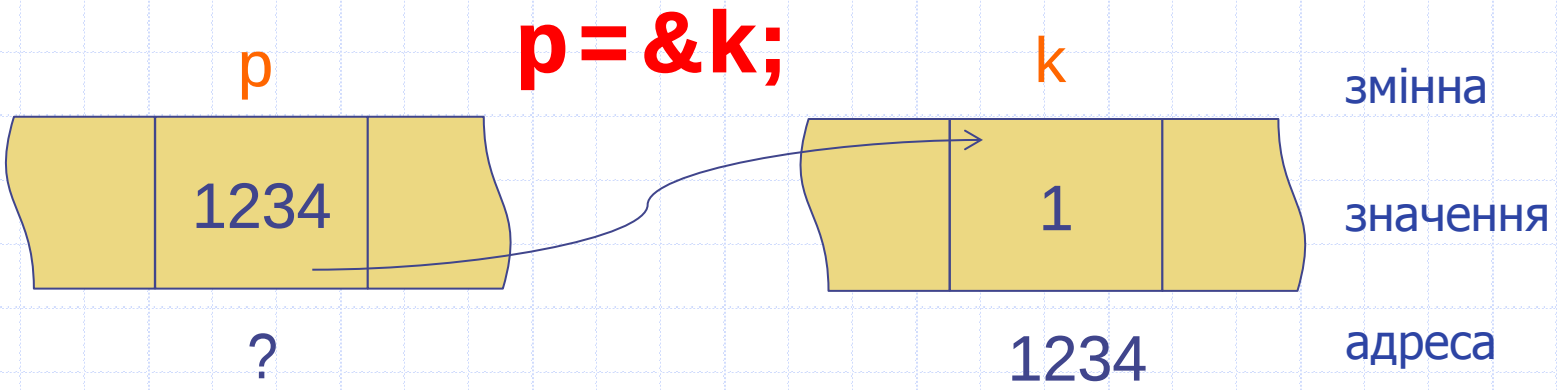
```
int k = 5;
int* p = &k;
```



Вказівники і адреси

- Вказівник містить адресу об’єкта,
- Забезпечує «непрямий» доступ до цього об’єкта.
- **Приклад**

```
int k = 1;
int* p;
```



NULL адреса

- Якщо заздалегідь невідомо, яку адресу зберігає вказівник, то йому присвоюється значення 0, але вказівник не може бути неініціалізований.
- Вказівник приймає значення 0, якщо :
  - вичерпана пам'ять, то оператор new повертає 0
  - це вказівник на кінець динамічної структури (список, дерево)
- Вказівник, значення якого рівне 0, називається **порожнім**.

```
int* p=0; //порожній вказівник
```

Операції з вказівниками

- У випадку **оголошення** змінної оператор * означає, що оголошується вказівник:

  int *p=0; //оголошення вказівника

- **Оператор розіменування** (*) дозволяє отримати значення, що зберігається за адресою, записаною в вказівнику (непряма адресація).
- У випадку розіменування операція проводиться не над адресою, а над значенням, збереженим за адресою:

  ***p=5;** //присвоює значення 5 змінній, що
  //зберігається за адресою в вказівнику

Приклади

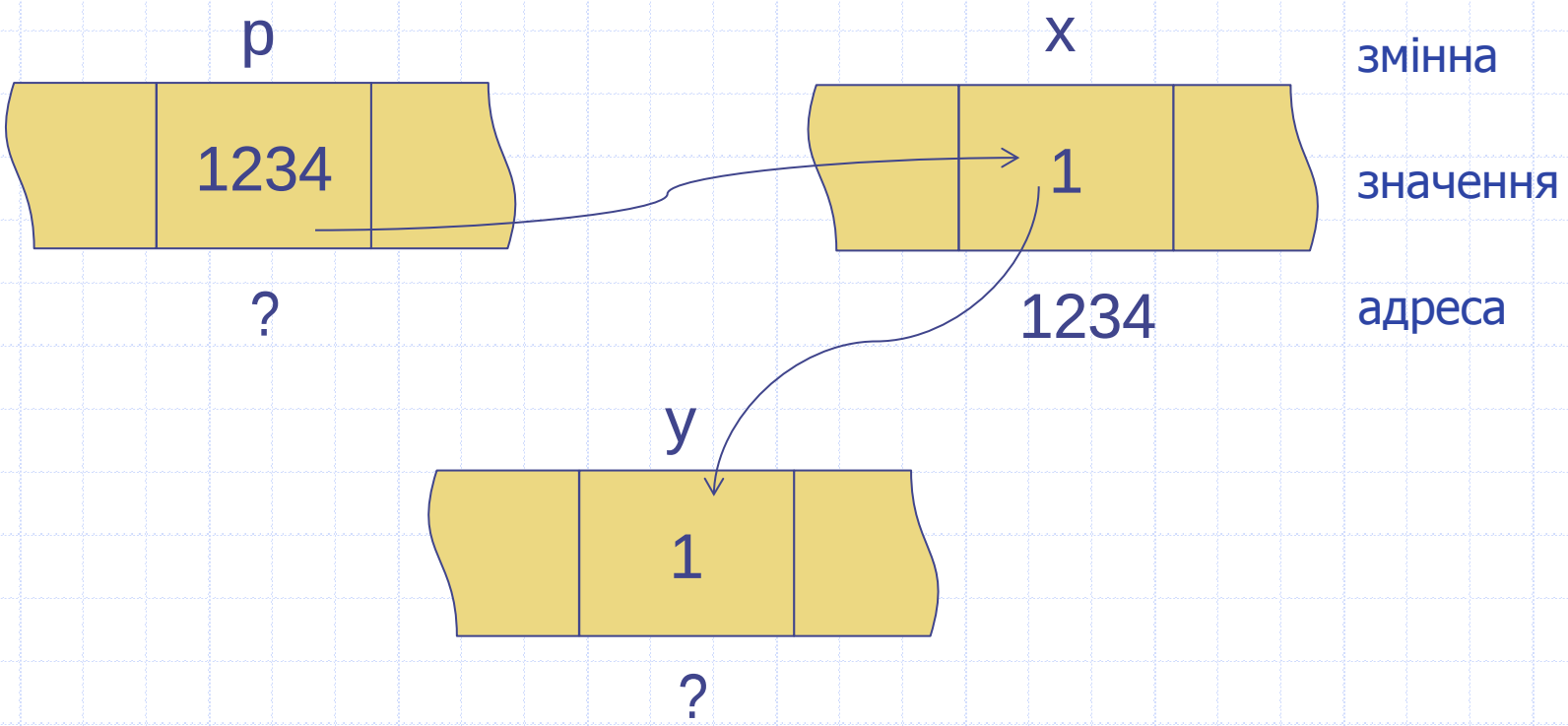
int i = 5, j; //оголошення змінних
int* p = &i;//p містить адресу змінної i
*p = 5;//присвоює змінній i значення 5
j = p; //заборонено (вказівник не може бути перетворений в цілочисельний тип int!)
j = *p+1; //j=6
p = &j; //p вказує на j

Операція розіменування

int *p=&x; // оголошення+ініц.

int y=*p; //розіменування

***p=10; // x=y=10; - розіменув.**



Приклади

double A = 5; /*оголошення дійсної змінної подвійної точності */

double *a; /*оголошення вказівника на дійсну змінну */

a = &A; /* присвоєння вказівнику адреси змінної A. a тепер вказує на A */

a = 10;** /*Присвоєння значення об'єкту, на який вказує a. ****a — операція розіменування вказівника */

У результаті змінна **A** отримує значення *10*.

Посилання

- **Посилання** (reference) являє собою видозмінену форму вказівника, яка використовується в якості **псевдоніму** (іншого імені) змінної
- Посилання **не потребують додаткової пам'яті**
- Для визначення посилання використовують символ **&** (амперсант), який ставитися перед змінною-посиланням.

```
int k = 5;   //ініціалізація змінної
int& alias = k; // створення посилання на об'єкт k
int *p=&k; // ініціалізація вказівника адресою
```

Призначення посилання

- Змінні типу посилання можуть використовуватися в наступних цілях:
  - для передачі у функцію об'єкта **за посиланням**, а не за значенням;
  - для визначення **конструктора копії** (в ООП);
  - для перевантаження операцій (в ООП);

Передача параметрів у функцію за посиланням

```
void swap (int& a, int& b)
    {  int temp;
       temp = a;
       a = b;
       b = temp; }
    swap(big_alias, small_alias); // попередньо
    //необхідно оголосити посилання
```

- Зміни значень аргументів зберігаються при виході із функції
- Використання посилань дозволяє уникнути комбінування вказівників зі звичайними змінними.

Відмінності між посиланнями і вказівниками

- Неможливо звертатись безпосередньо до посилання після його створення; кожне згадування його імені **посилається безпосередньо на об'єкт**, псевдонімом якого є.
- Після створення посилання **не може бути перевизначено**, що часто виконують з вказівниками.
- Посилання **не можуть мати значення NULL**. Кожне посилання відповідає конкретному об'єкту, незалежно від його коректності.

Відмінності між посиланнями і вказівниками

- Посилання не можуть бути **неініціалізовані**, оскільки їх неможливо перевизначити.
- Наприклад

  int& k; // **неприпустимо**

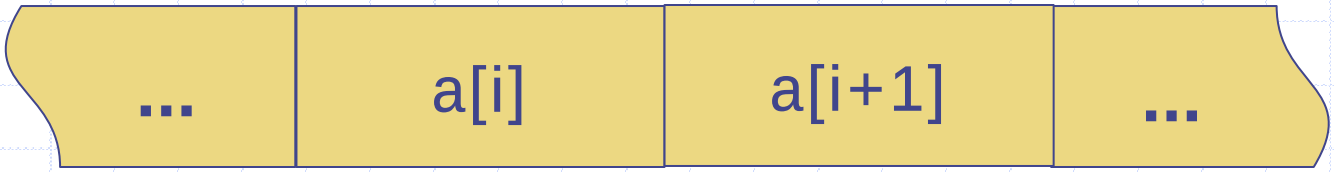
- Компілятор видасть повідомлення про помилку:

  'k' declared as reference but not initialized

  ('k' оголошена як посилання, але не ініціалізована)

Масиви

- **Загальне означення:**
  - **Масив** (англ. *array*) — одна з найпростіших **структур даних**, сукупність елементів одного типу даних, впорядкованих за індексами, які зазвичай представляються натуральними числами, що визначають положення елемента в масиві.
- **В C++:**
  - Масив – **похідний** тип даних. Всі елементи масиву повинні бути одного і того ж типу, наприклад, int, float або char;
  - Масив - спеціальна форма вказівника, яка зв'язана з **неперервним фрагментом пам'яті** для зберігання індексованих послідовностей значень.



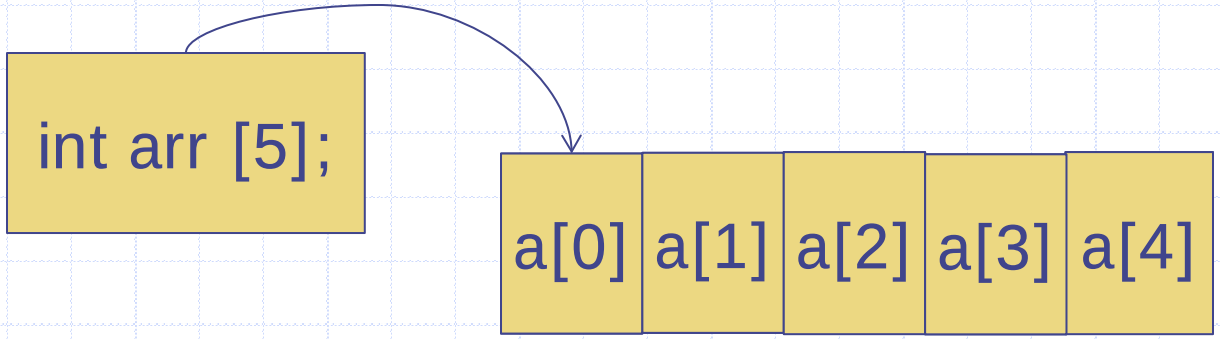
Приклад

```
int array[25]; // глобальний масив int
void main()
    {
        char arr1[256]; // масив з 256 символів
        static  long int arr2[56]; //статичний  масив
        extern int array[]; //зовнішній масив
        string arr3[7][8]; // двовимірний масив рядків
    }
```

- При визначенні масиву виділяється пам’ять, починаючи з базової адреси.
- Ім’я масиву є постійним **вказівником**, який ініціалізований цією базової адресою.

Статичний масив

- **Статичними** називають **масиви**, кількість елементів яких відома наперед і не змінюється в ході виконання програми.
- В C++ можливі масиви будь-яких типів: базових, користувацьких, масиви масивів тощо.
- Нумерація елементів масиву починається з нуля і закінчується *n*-1 (*n* - кількість елементів масиву).



Індексування

- Доступ до окремого елементу масиву організується з використанням номера цього елементу, або **індексу**.
- Використання значення індексу поза діапазоном називається **перевищенням меж масиву** або **виходом індексу за межі.**
- Ефект від такої помилки в C++ залежить від системи і може бути несподіваним! В C++ **компілятор не відслідковує межі масивів!**
- Масив може бути **одновимірним** (вектором), та **багатовимірним** (наприклад, двовимірною таблицею), тобто таким, де індексом є не одне число, а кортеж (сукупність) з декількох чисел, кількість яких збігається з **розмірністю** масива.

Багатовимірні масиви

- Багатовимірні масиви в C++ розглядаються як масиви, елементами яких є масиви.
- Визначення багатовимірного масиву має містити інформацію про тип, розмірності і кількості елементів кожної розмірності.

```
int Array1[10];
```
//одновимірний масив розмірності 10

```
int Array2[20][10];
```
//20 одномірних масивів розмірності 10

```
int Array3[30][20][10];
```
//30 двовимірних масивів розмірності 20 * 10

Динамічний масив

- **Динамічним** називається масив, розмір якого може змінюватися під час виконання програми. Для зміни розміру динамічного масиву мова програмування, що підтримує такі масиви, повинна надавати вбудовану функцію чи оператор.
- Динамічні масиви дають можливість більш гнучко працювати з даними, оскільки дозволяють не прогнозувати об’єм даних, що зберігається, а регулювати розмір масиву відповідно до реально необхідних об’ємів.

Динамічні масиви

- Динамічне оголошення масиву виконується з використанням оператора **new**, який відповідає за виділення **динамічної пам'яті**.

  `int n = 255;` // змінна може бути визначена після //початку виконання програми, але до моменту //оголошення масиву

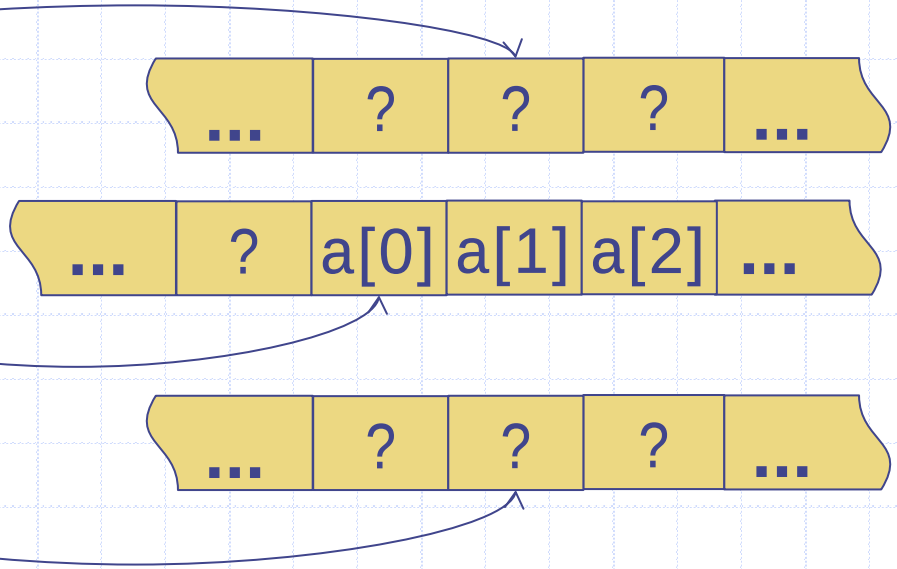
  ```
  int*array = new int[n];
  ```

- Оператор **delete** звільняє пам'ять, виділену оператором **new**, та повинен виконуватися для кожного виклику new, щоб уникнути витікання пам'яті.

  ```
  delete[] array;
  ```

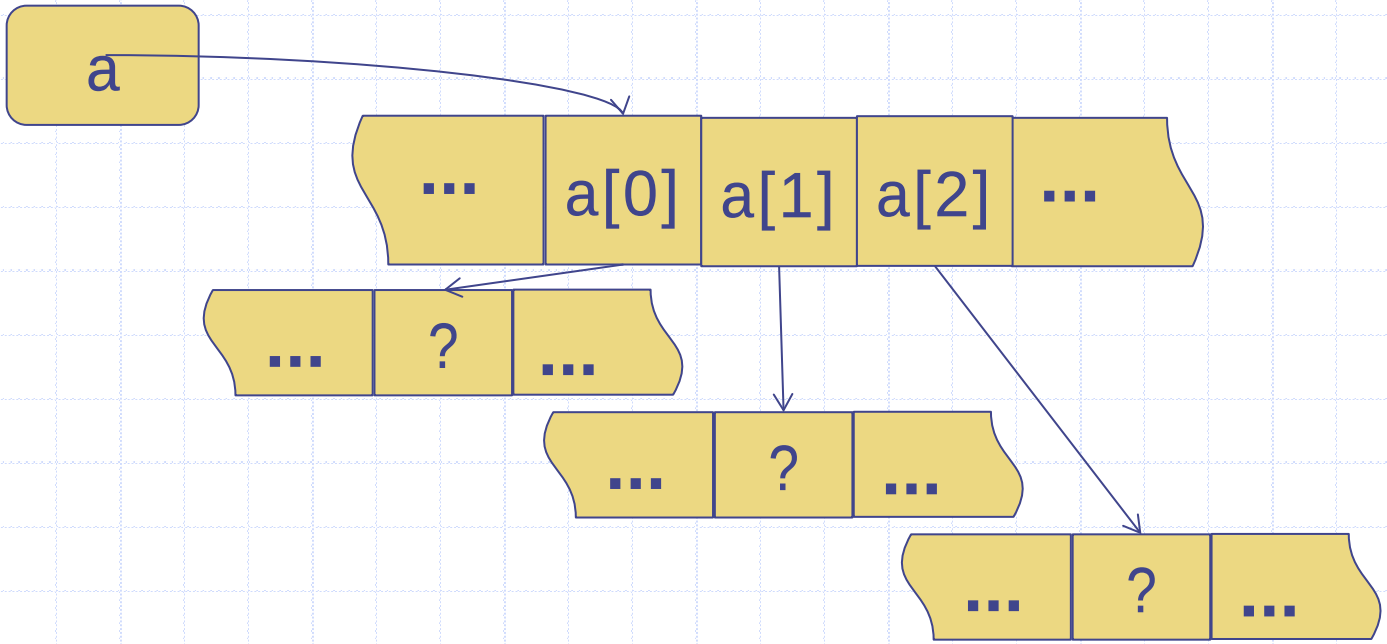
Динамічний масив (1-вимірний)

```
int *arr;

arr=new int [n];

    ...

delete []arr;
```



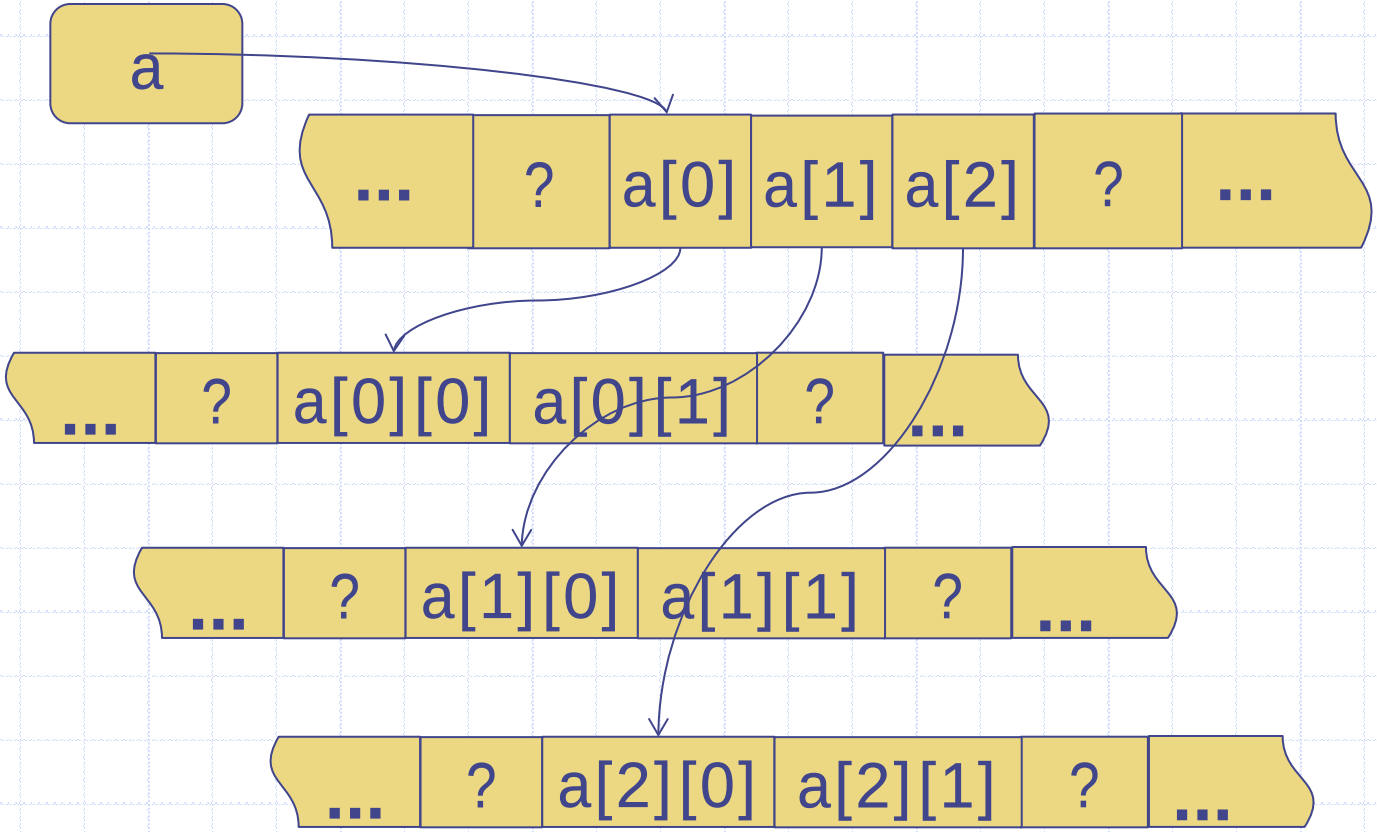
Динамічний масив (2-вимірний)

```
int **a=new int *[n]; //перший крок
for (int i=0; i<n; i++)
        a[i]=new int [m];
```



Динамічний масив (2-вимірний)

```
for (int i=0; i<n; i++)
    a[i]=new int [m];
```

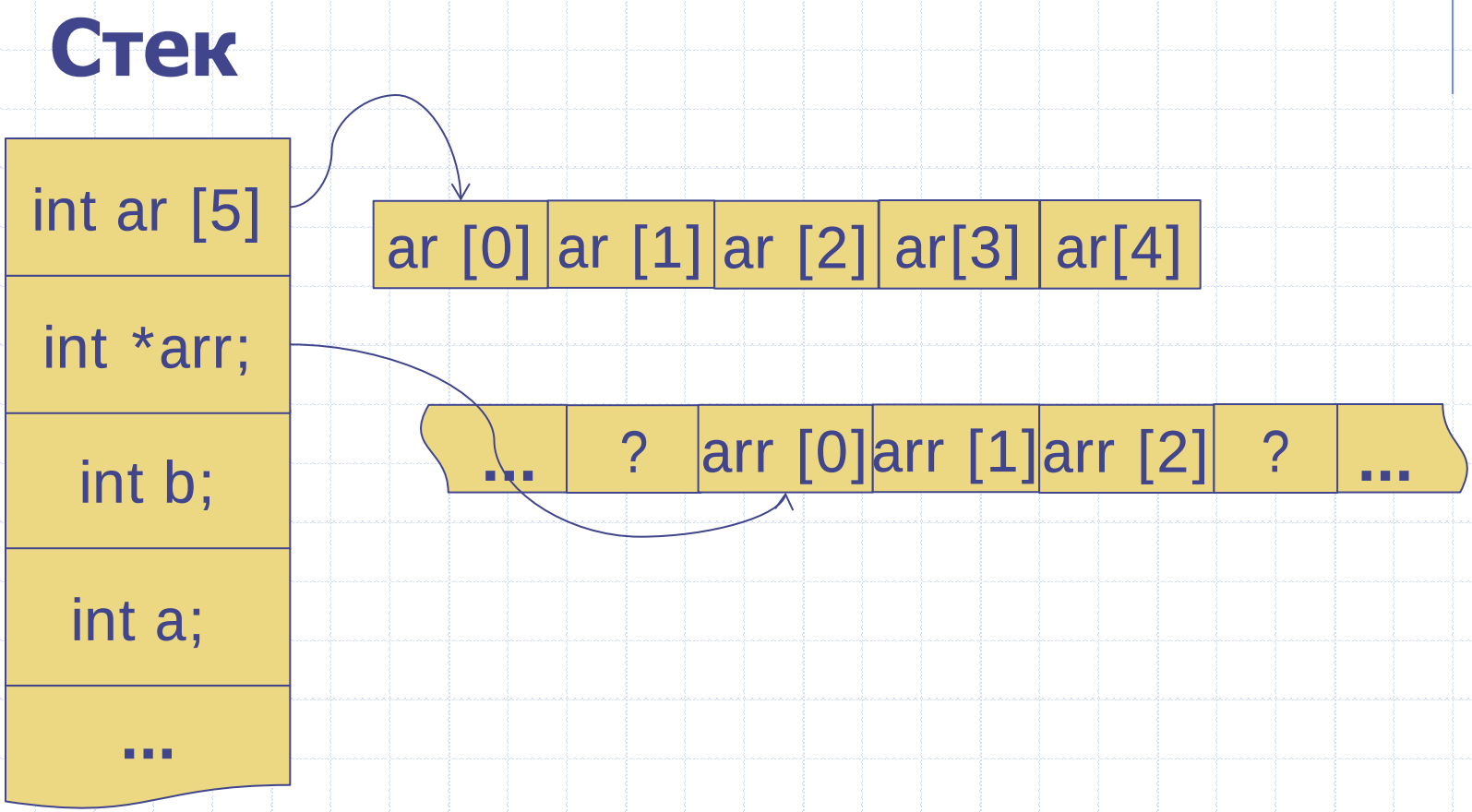


Приклад

- Багатовимірний динамічний масив:

```
char** b; //b - вказівник на масив вказівників
b=new char* [k]; //виділення пам’яті для масиву вказівників на k кортежів
for (int i=0; i<k; i++)
        b[i]=new char [m]; //виділення пам’яті для кожного рядка масиву розмірністю kxm
...
for (int i=0; i<k; i++)
                delete [] b[i];
delete []b;
```

Виділення пам’яті



Ініціалізація масивів

- Статичне оголошення масиву

int Array [10]; //масив типу int на 10 елементів

int Array**[]** = {1,2}; //кількість елементів масиву визначається **ініціалізатором**

int Array[2] = {1,2}; //важливо, щоб кількість елементів при ініціалізації **не перевищувала** оголошеного розміру.

int Array[3] = {1,2};//**часткова** ініціалізація масиву, при якій значення отримують перші елементи масиву, значення останнього елемента не визначено

Ініціалізатор масиву

- Список значень

```
int v1[] = {1, 2, 3, 4};
char v2[] = {‘a’, ‘b’, 0};
```

- Якщо масив оголошено **без вказання розміру**, але ініціалізовано списком, то розмір обчислюється автоматично шляхом підрахунку **елементів списку**!
- Якщо розмір вказується **явно**, то кількість елементів не повинна перевищувати заданого розміру!

```
char v3[2] = {‘a’, ‘b’, 0}; // помилка!
```

- Якщо в списку **недостатньо** елементів, інші елементи ініціалізуються **нулями**!

```
int v[5] = {1, 2, 3}; ⇔
v[5] = {1, 2, 3, 0, 0};
```

Ініціалізатор масиву (прод.)

- Не існує присвоювання, яке аналогічно ініціалізації:
  `v[3] = {1, 2, 3};` **//недопустимо!**
- Багатовимірний масив ініціалізується аналогічно.

```
int Array[3][3][3] =
  {0,1,2,3,4,5,6,7,8,9,10,11};
```
//першими ініціалізуються елементи з найменшими індексами

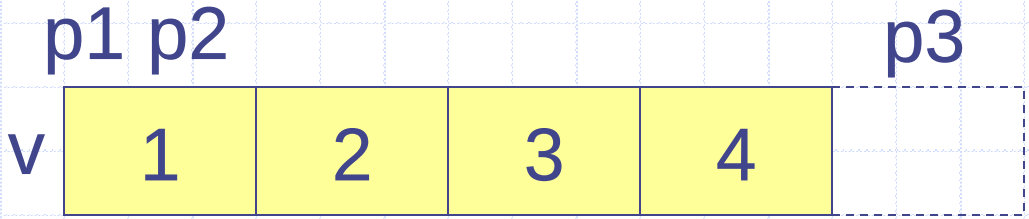
```
int Array[3][3][3] = { {{0}},
  {{10},{2,3},{9}}, {{7},{2,4},{6,3,0}} };
```
//додаткові фігурні дужки дозволяють ініціалізувати окремі фрагменти багатовимірного масиву, де кожна пара фігурних дужок специфікує значення, пов'язані з певною розмірністю; порожні дужки не допускаются.

Вказівники на масиви

- Ім’я масива може використовуватися в якості вказівника на його перший елемент
- int array[5]; //масив із 5 елементів типу int

  int *array[5]; //5 вказівників на об'єкти типу int

  int *array = new int[5];//вказівник на масив із 5 об'єктів типу int
- int v[] = {1, 2, 3, 4};
  int* p1 = v;
  int* p2 = &v[0];
  int* p3 = &v[4];

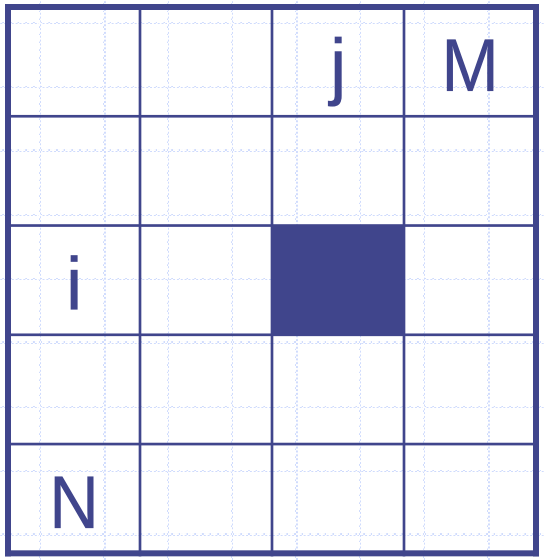


Двовимірний масив

- Масив виду

  A[N][M]

- int a[4][5];
  int* p;
  *p = a;
  a[i][j] = 4; ⇔
  *(p+i*M+j) = 4;



Двовимірний масив (прод.)

- Розміщення в пам’яті

p	p+1	p+2	p+3	p+4	p+5
a[0][0]	a[0][1]	a[1][0]	a[1][1]	a[2][0]	a[2][1]

Двовимірний масив

```
int** data;
data = new int*[N]; // створення рядків
for (int j=0; j<N; j++)
        data[j] = new int[M]; //створення стовпчиків

for (int i=0; i<N; i++)
        for(int j=0; j<M; j++)
                data[i][j] = i+j;
for(int i=0; i<N; i++)
delete [] data[i]; //звільнення пам’яті від кожного
рядка
delete [] data; //звільнення пам’яті від масиву
```