

Міністерство освіти і науки України
Національний університет “Львівська Політехніка”



Лабораторна робота №15-16

Виконав:
Студент групи АП-11
Білий Анатолій Іванович

Прийняв:
Чайковський І.Б.

“Дослідження використання одновимірних та багатовимірних масивів”

Мета роботи: навчитися використовувати одновимірні та багатовимірні масиви у процесі програмування для обробки великої сукупності значень.

Теоретичні відомості

В усіх програмах, що розглядалися у лабораторних роботах, оброблялися поодинокі значення. На практиці часто виникає потреба обробити єдиним алгоритмом велику сукупність однорідних значень. В математиці такі сукупності мають позначення на зразок x_1, x_2, \dots, x_n .

Для підтримки обробки таких сукупностей в мові C існує поняття масиву.

Масив - це сукупність даних одного типу, що об'єднані спільним ім'ям.

Загальний запис оголошення одновимірного масиву має наступний вигляд:

Тип_назва масиву[розмірність];

Треба звернути увагу, що розмір, який вказується при оголошенні масиву, повинен бути константою, тобто не може бути значенням змінної чи, скажімо, результатом виклику функції.

Ім'я масиву є його адресою (вказує на перший елемент масиву).

Наприклад, оголошення масивів з 12 цілих чисел та з 17 дійсних має вигляд:
int trail[10];

double value[17];

Нумерація елементів в масиві починається з 0. Якщо масив містить N елементів, то це значить, що в ньому є елементи під номерами 0, 1, \dots , $N-1$. Елемента номер N в масиві немає.

Для доступу до елементу масиву слугує операція індексування, яка позначається квадратними дужками: **конструкція ім'я_масиву[номер_елементу]** означає змінну - елемент, що стоїть в даному масиві під даним номером. Номер елементу в масиві називають також індексом. Звичайно ж, номер елементу може бути заданий будь-яким виразом цілого типу. Звертання до елементу масиву нічим в принципі не відрізняється від звертання до звичайної змінної за іменем, хіба що в даному випадку в ролі імені змінної виступає, так би мовити, складне

ім'я, що складається з імені самого масиву та індексу елементу. Наприклад, розглянемо програмний фрагмент:

```
1  int m[10], k=3;
2  m[0]=1;
3  m[k]=8;
4  ++ k;
5  m[k]=8;
6  m[( k + 2)%3+ 1]=17;
7  m[k+3]=m [0]+ m[k];
8  scanf("% d",&m[k+1]);
9  printf("%d\ n",m[k]);
```

В першому рядку оголошується масив *m* з 10 елементів та допоміжна змінна *k*, яка одразу отримує початкове значення 3.

В рядку 2 показано, як присвоїти значення елементу масиву, номер якого заздалегідь відомий: в якості індексу використано константу, число 0. Оскільки нумерація елементів починається з 0, то даний оператор означає, що значення присвоюється першому елементу масиву.

Рядок 3 ілюструє, що індекс може бути не константою, а значенням змінної. Оскільки в даний момент змінна *k* має значення 3, даний оператор означає, що значення 8 присвоюється у четвертий від початку (а не третій!) елемент масиву.

Оператор **в рядку 4** збільшує значення змінної *k* на 1, отже, воно тепер дорівнює 4. Тому, хоча оператор **в рядку 5** повністю співпадає за написанням з оператором в рядку 3, тепер вираз в лівій частині присвоювання означає вже не четвертий, а п'ятий від початку елемент масиву.

Рядок 6 є прикладом того, що в якості індексу може використовуватися не лише значення змінної, але і складний вираз. Підставивши поточне значення змінної *k*, маємо, що значення 17 буде присвоєно елементові з індексом 1, тобто другому елементу масиву.

В рядку 7 показано, що звертання до елементів одного й того самого масиву може здійснюватися і в лівій, і в правій частинах присвоювання. В перший

(з індексом 0) елемент раніше було занесене значення 1, поточне значення змінної *k* дорівнює 4, а елементу з індексом 4 було присвоєно значення 8. Отже, елемент з індексом 7 (восьмий від початку) отримає значення 9.

Значення елементів масиву можна вводити з клавіатури так само, як і значення звичайних змінних, за допомогою функції `scanf`, що показано **в рядку 8**. Як і завжди, перед іменем змінної, в яку треба розмістити введене значення, ставиться знак `&` - амперсанд.

З рядка 9 видно, що значення елементів масиву можна передавати до функцій в якості аргументів, в тому числі - друкувати на екран.

Одразу ж при оголошенні масиву можна присвоювати значення його елементам, або, як кажуть, ініціалізувати масив. Для цього достатньо записати знак рівності та послідовність значень елементів в фігурних дужках:

```
int m[4]={ -1 , 3, 0 , 27 };
```

Більш того, в такому випадку розмір масиву можна не вказувати:

```
int m[ ]={ -1 , 3, 0 , 27 };
```

Тоді транслятор сам визначить розмір, порахувавши ініціалізатори.

Підкреслимо наступне: присвоїти масиву одразу цілу сукупність значень можна лише при оголошенні масиву, тобто задавши початкові значення його елементам. По ходу виконання програми, серед операторів, присвоїти один масив іншому масиву (так, щоби з одного масиву в інший скопіювалися значення одразу всіх елементів) неможливо.

При ініціалізації масиву з рядка символів, рядок повинен закінчуватися нуль-символом `'\0'`. При спрощеній ініціалізації нуль-символ автоматично дописується у кінець рядка.

```
char S[]="Рядок";
```

Для обробки рядків в С визначено багато різних бібліотечних функцій. Найчастіше використовуються функції, які представлено у таблиці 1.

Функції для обробки рядків

Функція	Виконувана дія
strcpy(s1,s2)	Копіювання s2 в s1
strcat(s1,s2)	Конкатенація (приєднання) s2 в кінець s1
strlen(s1)	Повертає довжину рядка s1
strcmp(s1,s2)	Повертає 0, якщо s1 і s2 збігаються, негативне значення, якщо s1 < s2 і позитивне значення, якщо s1 > s2
strchr(s1,ch)	Повертає вказівник на перше входження символу ch в рядок s1
strstr(s1,s2)	Повертає вказівник на перше входження рядка s2 в рядок s1

Ці функції оголошені в заголовному файлі <string.h>. Застосування бібліотечних функцій обробки рядків ілюструється наступним прикладом:

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char s1[80], s2[80];
    gets(s1);
    gets(s2);
    printf("Dovzuna: %d %d\n", strlen(s1), strlen(s2));
    if(!strcmp(s1, s2)) printf("Ryadku rivni\n");
    strcat(s1, s2);
    printf("%s\n", s1);
    strcpy(s1, "Perevirka.\n");
    printf(s1);
    return 0;
}
```

Слід пам'ятати, що strcmp () приймає значення неістини, якщо рядки збігаються (хоч це і дещо нелогічно). Тому в тесті на збіг потрібно використовувати логічний оператор заперечення! як у наведеному прикладі.

Схема обробки масивів. Найбільша вигода від масивів, їх гнучкість та потужність, яка проявляються там, де потрібно обробити весь масив як ціле. Для цього використовується ідея послідовної обробки, елемент за елементом. В кожен момент часу алгоритм продирається один елемент масиву - поточний елемент. На початку поточним робимо перший елемент масиву. Далі на кожній ітерації циклу поточний елемент обробляється, після чого алгоритм переходить до наступного елементу, тобто наступний елемент стає поточним. Процес закінчується, коли всі елементи масиву вичерпано. Розпишемо загальну схему алгоритму в деталях.

1. Взяти поточним перший елемент масиву.
2. Масив вже закінчився?
 - Якщо так, то закінчити;
 - Якщо ні, то продовжувати з наступного кроку.
3. Обробити поточний елемент.
4. Взяти поточним наступний елемент масиву.
5. Перейти до кроку 2.

Нехай масив має ім'я m та містить N елементів. Щоб виділити в масиві поточний елемент, зрозуміло, потрібно використати цілочисельну змінну k - номер (індекс) поточного елементу. Тоді сам поточний елемент є $m[k]$. Фраза <взяти поточним перший елемент масиву> зі словесного опису алгоритму уточнюється в мові C оператором $k=0$, а фраза <взяти поточним наступний елемент масиву> - оператором $++k$. Умова <масив ще не закінчився> (умова продовження циклу) природно уточнюється виразом $k < N$ (нагадаємо, що оскільки нумерація елементів масиву починається з 0, то останній елемент має номер $k-1 < N$; якщо ж номер поточного елементу після чергового збільшення досягає значення N , то це означає, що всі елементи масиву вже перебрано).

Отже, основний цикл обробки масивів виглядає так:

```
for(k=0; k < N; ++ k)
    обробити m[k];
```

Наприклад, наведений нижче програмний фрагмент вводить масив дійсних чисел елемент за елементом.

```

. . .
#define N 10
int main() {
double m[ N ];
int k;
for( k = 0; k < N; ++k ) {
    printf( "Значення_номер_%d_", k );
    scanf( "%ld", &m[k] );
}
. . .

```

В наведеному прикладі проілюстровано також ще одну важливу деталь. З точки зору гарного професійного стилю, константу <кількість елементів масиву> абсолютно необхідно оформлювати як макрос. Справді, якби в тексті програми всюди, де потрібна кількість елементів, було жорстко вписане конкретне число 10, то при зміні умови задачі (скажімо, замість 10 ввести 12 чисел), було б дуже важко внести зміну в усі місця, де потрібно.

Нижче представлено програму для обчислень значення елементів одновимірної масиви цілого типу з індексами від 0 до 99.

```

#include <stdio.h>
int main(void)
{
    int x[100]; /* оголошення масиву цілого типу з 100 чисел */
    int t;
    for(t=0; t<100; ++t) x[t] = t; /* присвоєння масиву значень від 0 до 99 */
    for(t=0; t<100; ++t) printf("%d ", x[t]);
    return 0;
}

```

Багатовимірні масиви. Масив може бути одновимірним чи багатовимірним. Кількість пар квадратних дужок після імені масиву вказує на вимірність масиву. Якщо масив багатовимірний, то кількість елементів визначається як добуток кількостей елементів по всіх вимірах.

Загальний опис двовимірної масиви:

```

Тип_назва масиву[M][N];

```

де M і N - розмірність масиву. Кількість рядків визначає перше число у квадратних дужках, а кількість стовбців дорівнює другому числу.

Двовимірний масив елементів символьного типу з $3 \times 5 = 15$ елементів:

```

char symbol_matrix[3][5];

```

Тривимірний масив з $3 \times 3 \times 3 = 27$ елементів з плаваючою крапкою:

`float velocity[3][3][3];`

Наприклад, масив, описаний як `int a[4][3]`, можна представити у вигляді таблиці 2.

Таблиця 2

Графічне представлення двовимірного масиву

Масив a	Стовбець 0	Стовбець 1	Стовбець 2
Рядок 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>
Рядок 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>
Рядок 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>
Рядок 3	<code>a[3][0]</code>	<code>a[3][1]</code>	<code>a[3][2]</code>

Елементи двовимірного масиву розташовані у пам'яті ЕОМ за рядками.

Ініціалізація двовимірного масиву також відбувається за рядками, наприклад:

```
int a[][3] = { { 18, 21, 5 },  
              { 6, 7, 11 },  
              { 30, 52, 34 },  
              { 24, 4, 67 } };
```

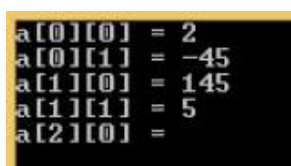
У такому випадку першу розмірність масиву буде визначена компілятором.

Друга розмірність масиву обов'язково повинна бути вказана.

Введення та виведення двовимірного масиву здійснюється поелементно за допомогою вкладених циклів. Наприклад, для введення двовимірного масиву елементів типу `double` розмірністю $N \times M$ за рядками, слід записати наступний фрагмент коду:

```
for (i=0; i<N; i++)  
{ for (j=0; j<M; j++)  
  { printf("a[%d][%d] = ", i, j);  
    scanf ("%f", &a[i][j]);  
  }  
}
```

Тоді стає зрозумілим, який елемент слід вводити :



```
a[0][0] = 2  
a[0][1] = -45  
a[1][0] = 145  
a[1][1] = 5  
a[2][0] =
```

Рис.1. Введення двовимірного масиву

У наступному прикладі елементам двовимірного масиву присвоюються числа від 1 до 12 і значення елементів виводяться на екран порядково:

```
#include <stdio.h>
int main(void)
{
    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;
    /* вивід на екран */
    for(t=0; t<3; ++t) {
        for(i=0; i<4; ++i)
            printf("%3d ", num[t][i]);
        printf("\n");
    }
    return 0;
}
```

У цьому прикладі num [0] [0] має значення 1, num [0] [1] - значення 2, num [0] [2] - значення 3 і так далі. Наочно двовірний масив num можна представити як:

num[t][i]				
	0	1	2	3
	---+-----			
0	1	2	3	4
2	5	6	7	8
3	9	10	11	12

Двовірні масиви розміщуються в матриці, що складається з рядків і стовпців. Перший індекс вказує номер рядка, а другий - номер стовпця. Це означає, що коли до елементів масиву звертаються в тому порядку, в якому вони розміщені в пам'яті, правий індекс змінюється швидше, ніж лівий.

Обсяг пам'яті в байтах, займаний двовірним масивом, обчислюється за такою формулою:

Кількість_байтів = Розмір_1-го_ виміру × розмір_2-го_ виміру × sizeof (базовий_тип).

Наприклад, двовірний масив 4-байтових цілих чисел розмірністю 10 × 5 займає ділянку пам'яті об'ємом: 10 × 5 × 4 тобто 200 байтів.

Якщо двомірний масив використовується в якості аргументу функції, то в неї передається тільки вказівник на початковий елемент масиву.

У відповідному параметрі функції, який отримує двомірний масив, обов'язково повинен бути зазначений розмір правого виміру [1], який дорівнює довжині рядка масиву.

Розмір лівого вимірювання вказувати не обов'язково. Розмір правого виміру необхідний компілятору для того, щоб усередині функції правильно обчислити адресу елемента масиву, так як для цього компілятор повинен знати довжину рядка масиву. Наприклад, функція, яка отримує двомірний масив цілих розмірністю 10×10 , повинна бути оголошена так:

```
void func1(int x[][10])  
{  
    /* ... */  
}
```

Компілятор повинен знати довжину рядка масиву, щоб усередині функції правильно обчислити адресу елемента масиву. Якщо при компіляції функції це невідомо, то неможливо визначити, де починається наступний рядок, і обчислити адресу елемента.

Масиви рядків. У програмах на мові C часто використовуються масиви рядків. Наприклад, сервер бази даних звіряє команди користувачів з масивом допустимих команд.

В якості масиву рядків в мові C служить двомірний символьний масив. Розмір лівого вимірювання визначає кількість рядків, а правого - максимальну довжину кожного рядка. Наприклад, в наступному операторі оголошений масив з 30 рядків з максимальною довжиною 79 символів:

```
char str_array [30] [80];
```

Щоб звернутися до окремого рядку масиву, потрібно вказати тільки лівий індекс. Наприклад, виклик функції `gets ()` з третім рядком масиву `str_array` як аргумент можна записати так:

```
gets (str_array [2]);
```

Цей оператор еквівалентний наступному:

```
gets (& str_array [2] [0]);
```

З цих двох форм запису кращою є перша. Для кращого розуміння властивостей масиву рядків розглянемо наступну коротку програму, в якій на основі застосування масиву рядків створений простий текстовий редактор:

```
#include <stdio.h>
#define MAX 100
#define LEN 80
char text[MAX][LEN];
int main(void)
{
    register int t, i, j;
    printf("Dlya vuxody vvedit pystuyi ryadok.\n");
    for(t=0; t<MAX; t++) {
        printf("%d: ", t);
        gets(text[t]);
        if(!*text[t]) break; /* вихід при пустому рядку */
    }
    for(i=0; i<t; i++) {
        for(j=0; text[i][j]; j++) putchar(text[i][j]);
        putchar('\n');
    }
    return 0;
}
```

Специфікатор `register` дає вказівку компілятору зберегти змінну способом, що дозволяє здійснювати найшвидший доступ.

Користувач вводить в програму рядки тексту, закінчуючи введення пустим рядком. Потім програма виводить текст посимвольно.

Багатовимірні масиви. У мові C можна користуватися масивами, розмірність яких більше двох. Загальна форма оголошення багатовимірного масиву наступна:

тип імя_масиву [Розмір1] [Розмір2] ... [РозмірN];

Масиви, у яких число вимірювань більше трьох, використовуються досить рідко, тому що вони займають великий обсяг пам'яті.

Наприклад, чотиривимірний масив символів розмірністю 10х6х9х4 займає 2160 байтів. Якби масив містив 2-байтові цілі, треба було б 4320 байтів. Якби елементи масиву мали тип `double`, причому кожен елемент (дійсне число подвійної точності) займав би 8 байтів, то для зберігання масиву треба було б 17280 байтів.

Обсяг необхідної пам'яті з ростом числа вимірювань зростає експоненціально. Наприклад, якщо до попереднього масиву додати п'ятий вимір,

причому його товщину за цим виміром зробити рівною 10, то його обсяг зросте до 172800 байтів.

При зверненні до багатовимірних масивів комп'ютер багато часу витрачає на обчислення адреси, так як при цьому доводиться враховувати значення кожного індексу. Тому доступ до елементів багатовимірного масиву відбувається значно повільніше, ніж до елементів одновимірного.

Ініціалізація масивів. У мові C масиви при оголошенні можна ініціалізувати. Загальна форма ініціалізації масиву аналогічна ініціалізації змінної:

тип ім'я_масиву [розмір1] ... [розмір №] = {список_значень};

Список_значень – це список констант, розділених комами. Типи констант повинні бути сумісними з типом масиву. Перша константа присвоюється першому елементу масиву, друга - другому і так далі. Після закриття фігурної дужки крапка з комою обов'язкова.

У наступному прикладі масив цілих з 10 елементів ініціалізується числами від 1 до 10:

int i [10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

Тут елементу *i* [0] присвоюється 1, а *i* [9] - 10. Символьні масиви, що містять рядки, можна форматовувати строковими константами:

char ім'я_масиву [розмір] = "рядок";

У наступному прикладі масив *str* ініціалізується фразою "Мова C":

char str [9] = "Мова C";

Це оголошення можна записати так:

char str [9] = {'M', 'o', 'v', 'a', ' ', 'C', '\0'};

Рядок закінчується нульовим символом, тому при оголошенні необхідно ставити розмір масиву, достатній для того, щоб цей символ помістився в ньому.

Припустимо, що необхідно створити таблицю повідомлень про помилки, використовуючи ініціалізацію масивів:

char e1 [12] = "Не можу прочитати \ n";

char e2 [13] = "Не можу записати \ n";

char e3 [18] = "Не можу відкрити файл \ n";

Для задання розміру масиву довелося б вручну підраховувати кількість символів в кожному повідомленні. Однак в мові C є конструкція, завдяки якій компілятор автоматично визначає необхідну довжину рядка.

Якщо в операторі ініціалізації масиву не вказано розмір масиву, компілятор створює масив такого розміру, що в ньому вміщаються всі ініціалізовані елементи. Таким чином створюється безрозмірний масив.

Використовуючи цей метод, попередній приклад можна записати так:

```
char e1 [] = "Не можу прочитати \n";
```

```
char e2 [] = "Не можу записати \n";
```

```
char e3 [] = "Не можу відкрити файл \n";
```

Тоді оператор

```
printf ("%s має довжину %d \n", e2, sizeof e2);
```

виведе на екран довжину e2.

Крім зменшення трудомісткості, ініціалізація безрозмірних масивів корисна тим, що дозволяє змінювати довжину будь-якого повідомлення, не піклуючись про дотримання меж масивів.

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Здійснити виконання прикладів, представлених у теоретичних відомостях, після чого представити скріни їх коду та результати виконання у звіті.
3. Напишіть програму, яка заповнює масив довільними цілими числами, введеними з клавіатури (розмір масиву становить 10), обчислює суму всіх елементів та виводить на екран сам масив і значення суми його елементів.
4. Оформити звіт.

Приклад 1

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

```
#include <string.h>
```

```
int main(void) {
```

```
    char s1[80], s2[80];
```

```
    printf("Введіть перший рядок:");
```

```
    fgets(s1, sizeof(s1), stdin);
```

```

    printf("Введіть другий рядок: ");
    fgets(s2, sizeof(s2), stdin);
    printf("Довжина:%zu %zu\n", strlen(s1), strlen(s2));
    if (!strcmp(s1, s2))
        printf("Рядки рівні:\n");
    strcat(s1, s2);
    printf("%s\n", s1);
    strcpy(s1, "Перевірка.\n");
    printf("%s", s1);
    return 0;
}

```

Введіть перший рядок: Hello
 Введіть другий рядок: World
 Довжина:6 6
 Hello
 World
 Перевірка.

Приклад 2

```

#include <stdio.h>

int main(void) {
    int x[100]; //оголошення масиву цілого типу з 100 чисел
    int t;
    for(t=0;t<100;++t) x[t]=t; //присвоєння значення від0 до 99
    for(t=0;t<100;++t) printf("%d\n",x[t]);}

```

```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

```

Приклад 3

```

#include<stdio.h>

int main(void){
    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;
//вивід на екран
for(t=0;t<3;++t){
for(i=0;i<4;++i)
printf("%3d",num[t][i]);
printf("\n");
}
return 0;
}

```

```

1 2 3 4
5 6 7 8
9 10 11 12

```

Приклад 4

```

#include <stdio.h>
#define MAX 100
#define LEN 80
char text[MAX][LEN];
int main(void) {
    int t, i, j;
    printf("Для виходу введіть пустий рядок.\n");
    for (t = 0; t < MAX; t++) {
        printf("%d: ", t);
        gets(text[t]);
        if (!*text[t]) break; } // вихід при пустому рядку
    for (i = 0; i < t; i++) {
        for (j = 0; text[i][j]; j++)
            putchar(text[i][j]);
        putchar('\n');    }
    return 0;}

```

Для виходу введіть пустий рядок.

0: Hello

1: World

2:

Hello

World

Приклад 5

```
#include <stdio.h>
#include<windows.h>
#define SIZE 10
int main(void) {
    SetConsoleCP(65001);
    SetConsoleOutputCP(65001);
    int arr[SIZE];
    int sum = 0;
    printf("Введіть %d цілих чисел:\n", SIZE);
    for (int i = 0; i < SIZE; i++) {
        printf("Елемент %d: ", i + 1);
        scanf("%d", &arr[i]);
        sum += arr[i];}
    printf("Введений масив: ");
    for (int i = 0; i < SIZE; i++) {
        printf("%d ", arr[i]);}
    printf("\nСума елементів масиву: %d\n", sum);
    return 0;}
```

Введіть 10 цілих чисел:

Елемент 1: 1

Елемент 2: 2

Елемент 3: 3

Елемент 4: 4

Елемент 5: 5

Елемент 6: 6

Елемент 7: 7

Елемент 8: 8

Елемент 9: 9

Елемент 10: 10

Введений масив: 1 2 3 4 5 6 7 8 9 10

Сума елементів масиву: 55

Контрольні питання

1. Дайте визначення поняття масив в мові C.

У мові програмування C масив — це набір елементів одного типу, збережених разом під одним іменем. Масиви дозволяють зберігати багато значень в одній змінній. Кожен елемент масиву має свій номер (індекс), який починається з 0.

2. Назвіть види масивів.

Одновимірні масиви: Це звичайні списки елементів, розташовані один за одним в один ряд.

Двовимірні масиви: Це таблиці, де елементи розташовані в рядках і стовпцях, подібно до електронних таблиць (наприклад, Excel).

Багатовимірні масиви: Це розширені таблиці, які можуть мати більше двох вимірів, наприклад, куби або ще складніші структури.

Масиви символів (рядки): Це послідовності символів, що використовуються для зберігання тексту.

3. Назвіть перевагу використання багатовимірних масивів.

Організація даних: Багатовимірні масиви дозволяють зручно організувати та зберігати складні структури даних, такі як матриці, таблиці, або навіть тривимірні об'єкти. Це робить код більш зрозумілим і легким для підтримки.

Ефективний доступ до даних: Використання багатовимірних масивів дозволяє легко і швидко доступатися до конкретних елементів даних, використовуючи індекси. Це особливо корисно в таких задачах, як обробка зображень або наукові обчислення.

Моделювання реальних систем: Багатовимірні масиви можуть бути використані для моделювання реальних систем, таких як тривимірні простори або багатовимірні графіки, що робить їх корисними в науці, інженерії та комп'ютерній графіці.

Скорочення коду: Використання багатовимірних масивів може зменшити кількість коду, необхідного для управління складними даними, оскільки всі дані зберігаються в одній структурі.

4. Для чого у масивах використовується матриця?

Матриця - це двовимірний масив, який використовується для зберігання даних у вигляді таблиці з рядками та стовпцями. Матриці дозволяють ефективно працювати з даними, які мають двовимірну структуру, наприклад, зображення, графіки, математичні розрахунки та інші дані, організовані у вигляді таблиці.

Основні використання матриць у масивах включають:

Математичні операції: Матриці використовуються для виконання матричних операцій, таких як додавання, віднімання, множення та ділення матриць, що є важливими для багатьох обчислень у математиці, фізиці та інших науках.

Зображення і графіка: У комп'ютерній графіці та обробці зображень матриці використовуються для зберігання пікселів зображення, де кожен елемент матриці може відповідати кольору чи іншій властивості пікселя.

Обробка даних: Матриці можуть використовуватися для зберігання та обробки даних у формі таблиць, що дозволяє здійснювати швидкий доступ до різних елементів даних.

Моделювання реальних систем: У науці та інженерії матриці використовуються для моделювання різних фізичних та технічних систем, де дані можна представити у вигляді таблиці.

Аналіз даних: У статистиці та машинному навчанні матриці використовуються для зберігання та обробки великих обсягів даних для аналізу та виявлення закономірностей.

5. Яка загальна форма ініціалізації масиву?

тип_даних - тип даних елементів масиву,

ім'я_масиву - ім'я масиву,

розмір - кількість елементів у масиві,

{елемент_1, елемент_2, ..., елемент_n} - список значень, які ініціалізують масив.

Висновок: Під час виконання цієї лабораторної роботи я навчився використовувати одновимірні та багатовимірні масиви у процесі програмування для обробки великої сукупності значень.