Лекция 4(6)

Функції. Масиви.

Про функції

- Функція - це послідовність операцій для виконання певного завдання.

- **Виклик функції** - це вираз, який вказує процесору перервати виконання поточної функції і приступити до виконання іншої функції.

- Процесор «залишає закладку» в поточній точці виконання, а потім виконує функцію, що викликається. Коли виконання функції, що викликається завершено, процесор повертається до закладки і відновлює виконання перерваної функції.

Про функції

```
#include <iostream>
 // Оголошення функції doPrint ()
void doPrint ()
{
   std :: cout << "In doPrint ()" << std :: endl;
}

// Оголошення функції main ()
int main ()
{
   std :: cout << "Starting main ()" << std :: endl;
   doPrint (); // виклик функції doPrint ().
   std :: cout << "Ending main ()" << std :: endl;
   return 0;
}
```

Результат виконання програми:

```
Starting main ()
In doPrint ()
Ending main ()
```

Значення, що повертаються

- Функції можуть повертати значення.
- Для цього потрібно вказати тип значення (або «тип повернення»).
- Тип вказується при оголошенні функції, перед її ім'ям.
- Зверніть увагу, тип повернення *не вказує, яке саме значення буде повертатися*.
- Він вказує ***тільки тип цього значення***.

Значення, що повертаються

```
#include <iostream>

// int означає, що функція повертає
ціле значення в програму
int return7()
{
    // Ця функція повертає ціле значення,
    // тому використовується оператор return
    return 7;
}

int main()
{
    std :: cout<<return7()<<std :: endl; // виведеться 7
    std :: cout<<return7() + 3<<std :: endl; // виведеться 10
    return7();
    return 0;
}
```

Результат виконання програми:

- 7
- 10

Тип повернення void

- Функції можуть і не повертати значення. Щоб повідомити компілятору, що функція не повертає значення, потрібно використовувати тип повернення void.

```
void doPrint() // void - це тип повернення
{
    std :: cout<<"In doPrint()"<<std::endl;
    // Ця функція не повертає ніякого значення,
    // тому оператор return тут не потрібен
}
```

Ця функція має тип повернення void, який означає, що функція не повертає значення. Оскільки значення; не повертається, то і оператор return не потрібно.

Тип повернення void

```
#include <iostream>

void returnNothing()
{
    std::cout<<"Hi!"<<std::endl;
}

int main()
{
    returnNothing();
    std::cout<<returnNothing();
                    // помилка, цей рядкок не скомпілюється.
    return 0;
}
```

Повернення значень в функцію main()

- Коли програма виконується,
  - операційна система робить виклик функції main() і починається її виконання.
  - Конструкції в main() виконуються послідовно.
  - В кінці функція main() повертає цілочисельне значення (зазвичай 0) назад в операційну систему.
  - Тому main() оголошується як int main().

- Значення функції main() **є кодом стану, який повідомляє операційній системі про успішне або невдале виконання програми**.
- Зазвичай, повертається ***значення 0 (нуль) означає що все пройшло успішно, тоді як будь-яке інше значення означає невдачу / помилку.***

Повторне використання функцій

```
include <iostream>
int getValueFromUser()
{
   std::cout<<"Enter an integer:";
   int x;
   std::cin >> x;
   return x;
}
 int main()
{
   int first = getValueFromUser();
   int second = getValueFromUser();
   std::cout<<first<<"+"<<second<<"="<< a + b<<std::endl;
   return 0;
}
```

Результат виконання програми:

- Enter an integer: 4
- Enter an integer: 9
- 4 + 9 = 13

Повторне використання функцій

```
#include <iostream>
void printO()
{    std::cout<<"O"<<std::endl; }
void printK()
{    std::cout<<"K"<<std::endl; }
void printOK()
{  printO();
   printK(); }
int main()
{
   std::cout<<"Starting main()"<< std::endl;
   printOK();
   std::cout<<"Ending main()"<<std::endl;
   return 0;
}
```

Результат виконання програми:

- Starting main()
- O
- K
- Ending main()

Вкладені функції

- У мові C ++ одні функції не можуть бути оголошені всередині інших функцій (тобто бути вкладеними).

```
#include <iostream>
int main ()
{
    int boo () // ця функція знаходиться всередині функції main (), що заборонено
    {
       std :: cout << "boo!";
       return 0;
    }
     boo ();
    return 0;
}
```

Вкладені функції

```
#include <iostream>

int boo () // тепер уже не в main ()
{
    std :: cout << "boo!";
    return 0;
}

int main ()
{
    boo ();
    return 0;
}
```

Масиви

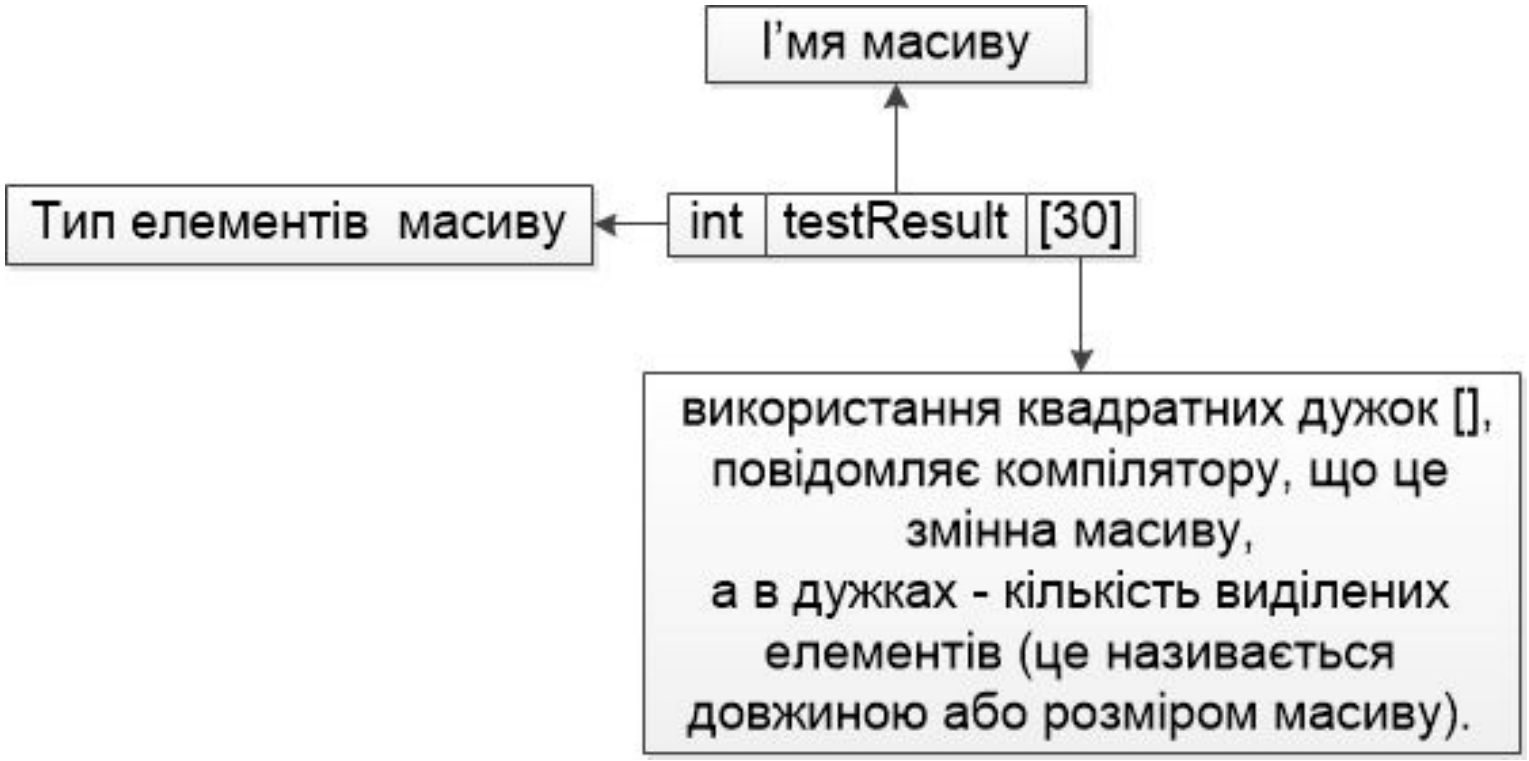
Що таке масиви?

- ***Масив*** - сукупний тип даних, який дозволяє отримати доступ до всіх змінним одного і того ж типу даних через використання одного ідентифікатора.

- //Виділяємо 30 цілочисельних змінних (кожна з різним ім'ям)
- int testResultStudent1;
- int testResultStudent2;
- int testResultStudent3;
- // ...
- int testResultStudent30;

- З використанням масиву все набагато простіше.
- int testResult[30]; // виділяємо 30 цілочисельних змінних, використовуючи фіксований масив

Що таке масиви?



Элементи массиву

- Кожна з змінних в масиві називається ***елементом***.
- Елементи *не мають* своїх власних унікальних імен.
- Для доступу до елементів масиву використовується ім'я масиву разом з ***оператором індексу [] і параметром***, який називається ***індексом***. Цей процес називається **індексуванням масиву.**

- *Важливо:*
- Відлік індексів масивів в програмуванні в С ++ завжди починається з **0**, а не з 1!

Елементи массиву

```
#include <iostream>



int main()
{
    int array[5]; // массив з п’яти чисел
    array[0] = 3; // індекс першого елементу - 0 (нульовий елемент)
    array[1] = 2;
    array[2] = 4;
    array[3] = 8;
    array[4] = 12; // індекс останнього елементу - 4

    std::cout << "The lowest number is " << array[0] << "\n";
    std::cout << "The sum of the first 5 numbers is " ;
     std::cout<< array[0] + array[1] + array[2] + array[3] + array[4] << "\n";

    return 0;
}
```

Результат виконання програми

The lowest number is 3
The sum of the first 5 numbers is 29

Типи данних та массиви

```
#include <iostream>

int main()
{
    double array[3]; // виділяємо 3 змінні типу double
    array[0] = 3.5;
    array[1] = 2.4;
    array[2] = 3.4;

    std::cout << "The average is " ;
    std::cout << (array[0] + array[1] + array[2]) / 3 << "\n";

    return 0;
}
```

Масив може бути будь-якого типу даних.

Індекси масивів

- У C ++ індекси масивів завжди повинні бути **інтегрального типу даних**
  (char, short, int, long, long long, bool і т.д.).
- Ці індекси можуть бути або константними значеннями, або неконстантнимі значеннями.

- **int array[4];** // оголошуємо масив довжиною 4
- // Використовуємо літерал (константу), як індекс
- **array[2] = 8;**
- // Використовуємо перерахування (константу), як індекс
- **const int ANIMAL_CAT = 3;**
- **array[ANIMAL_CAT] = 5;**
- // Використовуємо змінну (НЕ константу), як індекс
- **short index = 4;**
- **array[index] = 8;**

Оголошення масивів фіксованого розміру

- int array[7]; // Використовуємо макрос-об'єкт з текстом-заміною як символьної константи
- **#define ARRAY_WIDTH 4**
- **int array[ARRAY_WIDTH];** // синтаксично добре, але не робіть цього
- // Використовуємо символьну константу
- **const int arrayWidth = 7;**
- **int array[arrayWidth];**
- // Використовуємо неконстантную змінну
- **int width;**
- **std::cin >> width;**
- **int array[width];** // погано: width повинна бути константою типу compile-time!

Ініціалізація фіксованих масивів

- int array[5]; // масив містить 5 простих чисел
- array[0] = 4;
- array[1] = 5;
- array[2] = 8;
- array[3] = 9;
- array[4] = 12;

- int array[5] = { 4, 5, 8, 9, 12 };

- Якщо в цьому списку ініціалізаторів більше, ніж може містити масив, то компілятор видасть помилку.
- Однак, якщо в списку ініціалізаторів менше, ніж може містити масив, то інші елементи будуть ініційовані значенням 0.

Ініціалізація фіксованих масивів

- Щоб ініціалізувати всі елементи масиву **значенням 0,** потрібно:
- **int array[5] = { };**

- Ініціалізіруем всі елементи масиву значень C++ замість цього може використовуватися синтаксис uniform-ініціалізації:
- **int array[5] { 4, 5, 8, 9, 12 };**

Довжина масиву

- Якщо ви ініціалізіруете фіксований масив за допомогою списку ініціалізаторів, то компілятор може **визначити довжину масиву замість вас**, і вам вже не буде потрібно її оголошувати.
-
- Наступні два рядки виконують одне і те ж:
- // явно вказуємо довжину масиву
- **int array[5] = { 0, 1, 2, 3, 4 };**

- // список ініціалізаторів автоматично визначить довжину масиву
- **int array [] = {0, 1, 2, 3, 4};**

Оператор sizeof та масиви

- Оператор sizeof можна використовувати і з масивами: ***він повертає загальний розмір масиву*** (довжина масиву помножена на розмір одного елемента) в байтах.

```
#include <iostream>
int main()
{
    int array[] = { 1, 3, 3, 4, 5, 9, 14, 17 };
    std::cout << sizeof(array) << '\n';
                // виводиться розмір масива
     return 0;
}
```

- Результат виконання програми: 32

Індексування масиву в межах дії

```
int main()
{
    int array[5]; // масив містить 5 простих чисел
    array[5] = 14;

    return 0;
}
```

- ***Правило: При використанні масивів переконайтеся, що ваші індекси коректні і відповідають діапазону вашого масиву.***

Використання циклів з масивами

- const int numStudents = 5;
- **int student0 = 73;**
- **int student1 = 85;**
- **int student2 = 84;**
- **int student3 = 44;**
- **int student4 = 78;**
- **int totalScore = student0 + student1 + student2 + student3 + student4;**
- double averageScore = static_cast<double>(totalScore) / numStudents;
- ---
- const int numStudents = 5;
- **int students[numStudents] = { 73, 85, 84, 44, 78};**
- **int totalScore = students[0] + students[1] + students[2] + students[3] + students[4];**
- double averageScore = static_cast<double>(totalScore) / numStudents;

Використання циклів з масивами

```
int students[] = { 73, 85, 84, 44, 78};
const int numStudents = sizeof(students) / sizeof(students[0]);
int totalScore = 0;

// Використовуємо цикл для обчислення totalScore
for (int person = 0; person < numStudents; ++person)
    totalScore += students[person];

double averageScore = static_cast<double>(totalScore) / numStudents;
```

- ***Оскільки доступ до кожного елементу масиву виконується через цикл, то формула підрахунку суми всіх значень автоматично налаштовується з урахуванням кількості елементів в масиві.***

Використання циклів з масивами

- Цикли з масивами зазвичай використовуються для виконання однієї з трьох наступних завдань:

1. Обчислити значення (наприклад, середнє або суму всіх значень).
2. Знайти значення (наприклад, найбільше або найменше).
3. Відсортувати елементи масиву (наприклад, за зростанням або спаданням).

Масиви і «помилка неврахованої одиниці»

```
#include <iostream>

int main()
{
   int students[] = { 73, 85, 84, 44, 78 };
   const int numStudents = sizeof(students) / sizeof(students[0]);
   int maxScore = 0; // відстежуємо найвищу оцінку
   for (int person = 0; person <= numStudents; ++person)
   if (students[person] > maxScore)
                        maxScore = students[person];
  std::cout << "The best score was " << maxScore << '\n';
  return 0;
}
```

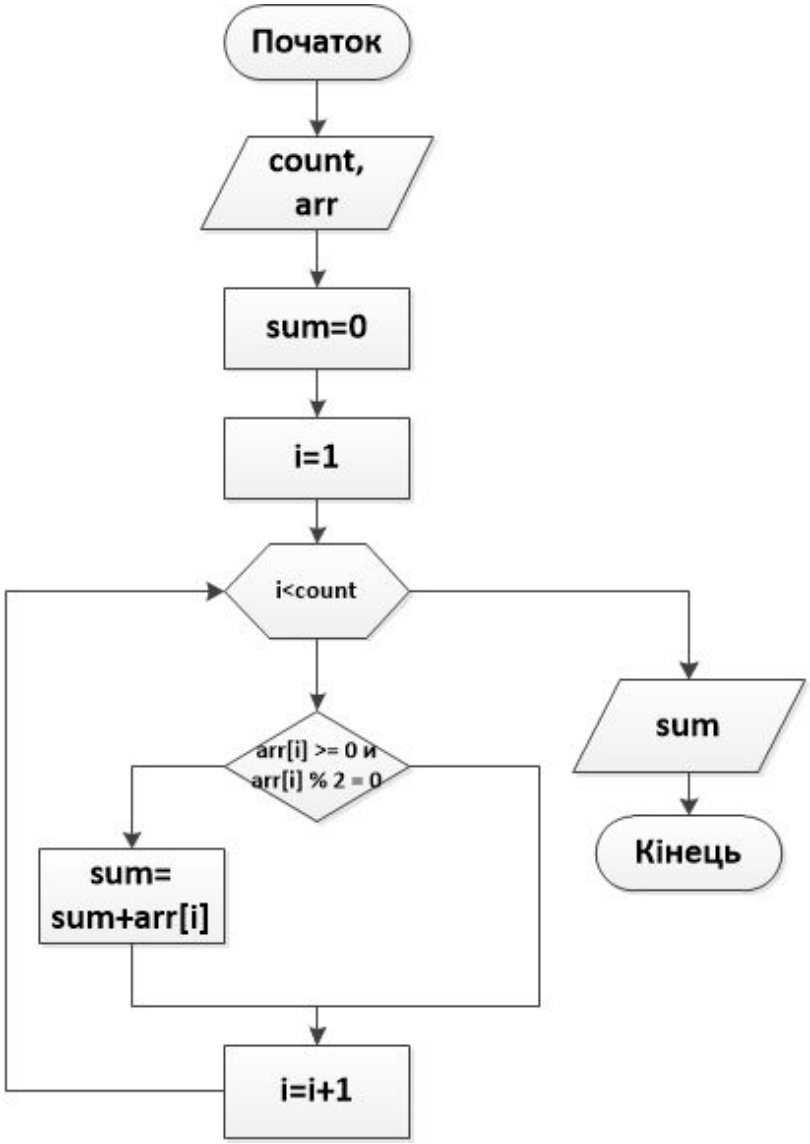
Масиви і «помилка неврахованої одиниці»

- *При використанні циклів з масивами, завжди перевіряйте умови в циклах, щоб переконатися, що їх виконання не призведе до помилки неврахованої одиниці.*

Застосування статичних масивів

- ***Завдання 1.***
- У масиві, що містить додатні та від'ємні цілі числа, обчислити суму парних додатних елементів масиву .

Алгоритм програми



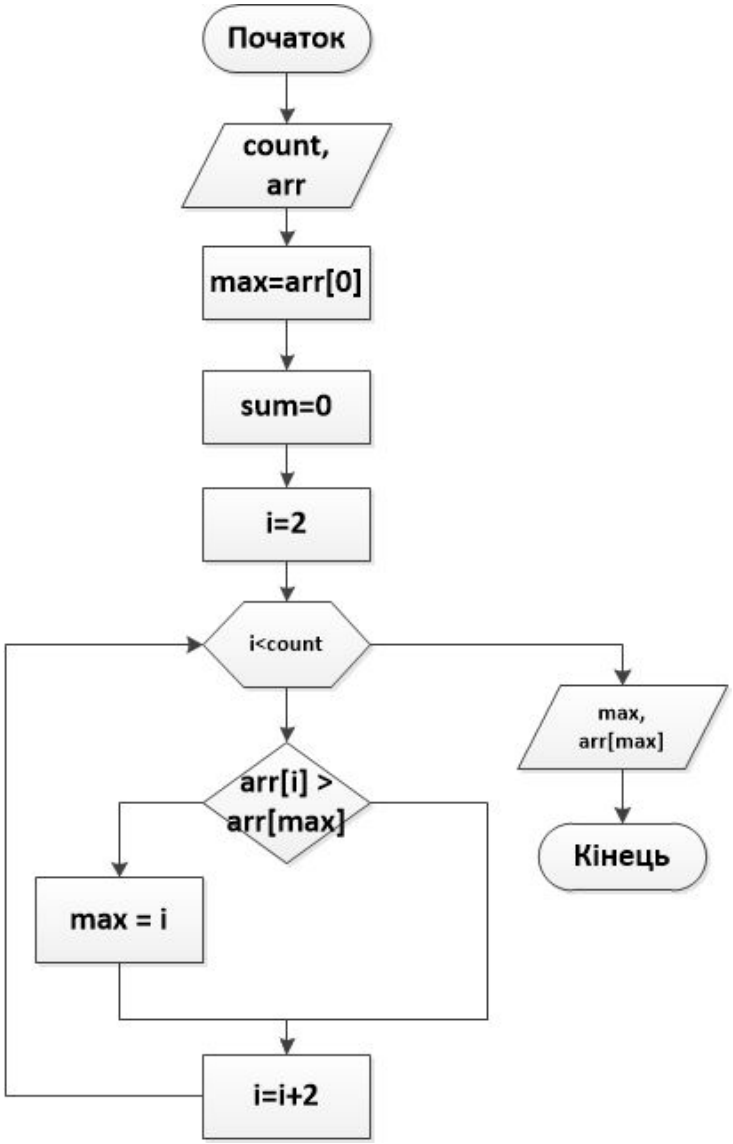
Текст програми

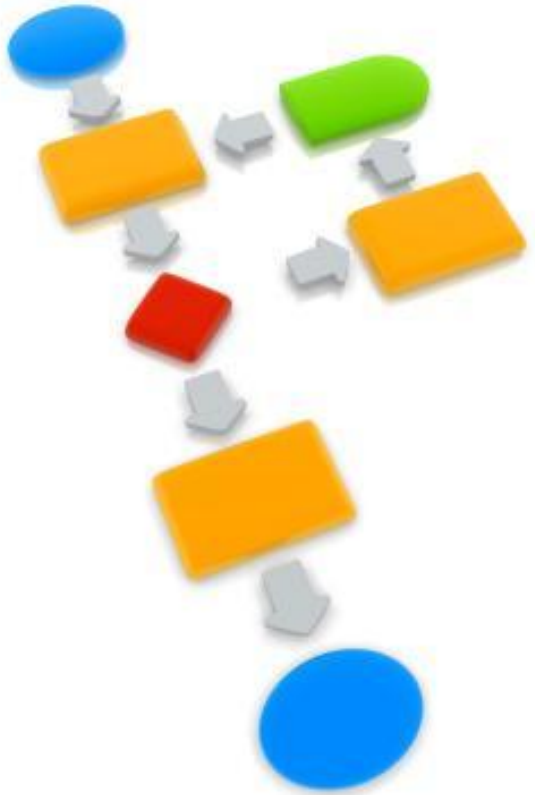
```
#include <iostream>
#define N 100
int main()
{ int arr[N] = {};
 int i, sum = 0, count = 0;
 std::cout << "Enter count of array members n="; std::cin >> count;
 for (i = 0; i < count; i++) {
      arr[i] = rand() % 10 - 5;          //  -5 …5
      std::cout << "\narr[" <<i<< "]="<<arr[i];   }
for (i = 0; i < count; i++)
{
      if (arr[i] >= 0 && arr[i] % 2 == 0)
          sum += arr[i];
}
std::cout << "\n\nSum of members = " << sum << "\n";
system("pause");  return 0;
}
```

Застосування статичних масивів

- ***Завдання 2.***
- У масиві знайти максимальний елемент з парним індексом.

Алгоритм програми





Текст програми

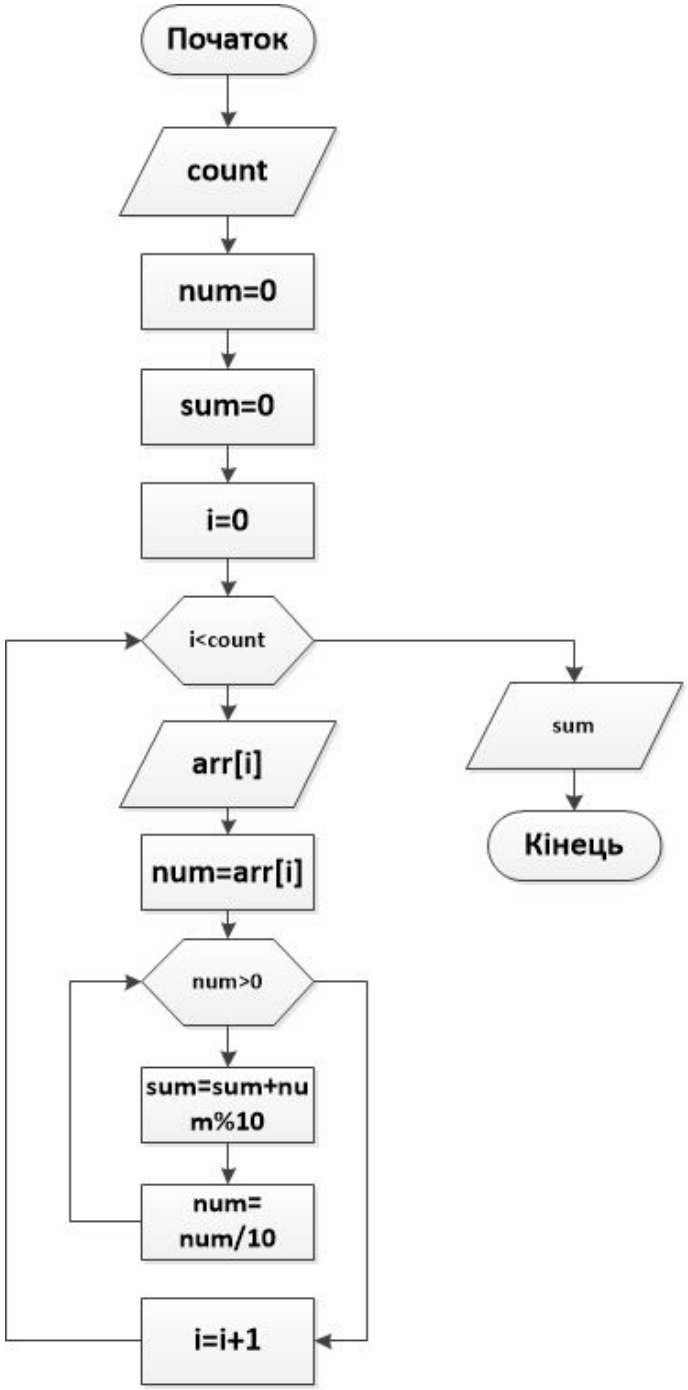
```
#include <iostream>
#define N 100


int main()
{
int arr[N] = {};
int i, max = 0, count = 0;
std::cout << "Enter count of array members n="; std::cin >> count;
for (i = 0; i < count; i++)
        {
        arr[i] = rand() % 100;
        std::cout << "\narr[" <<i<< "]="<<arr[i];
        }
max=arr[0];
for (i = 2; i < count; i+=2)
        {
             if (arr[i] > arr[max]) max = i;
        }
std::cout << "\n\nMaximum index= " << max << "\n";
std::cout << "\n\nMaximum = " << arr[max] << "\n";
system("pause");
return 0;
}
```

Застосування статичних масивів

- ***Завдання 3.***
- Знайти суму всіх цифр цілочисельного масиву. Наприклад,
- якщо дано масив [12, 104, 81], то сума всіх його чисел буде дорівнює 1 + 2 + 1 + 0 + 4 + 8 + 1 = 17.

Алгоритм програми

num=0
sum=0
i=0...count
arr[i]
num=arr[i]
num>0
sum=sum+num%10
num=num/10
i=i+1
num=0



Текст програми

```
#include <iostream>
#define N 100
int main() {
int arr[N] = {};
int i, sum = 0, num = 0, count;
std::cout << "\nCount="; std::cin >> count;
for (i = 0; i < count; i++)
{
	arr[i] = rand() %20;
	std::cout <<arr[i]<<"  ";
	num = arr[i];
	while (num > 0)
	{
		sum += num % 10;
		num /= 10;
	}
}
std::cout << "\nsum = "<<sum<<"\n";
system("pause"); return 0;
}
```

Результат виконання програми

count=15

41 17 34 0 19 24 28 8 12 14 5

45 31 27 11

sum = 91

Дякую за увагу!