**1. У чому полягає суть успадкування?**

**-** Успадкування являє собою один із ключових аспектів ооп, що дозволяє наслідувати функціонал одного класу ( базового класу) в іншому - класі-нащадку

(<https://metanit.com/cpp/tutorial/5.10.php>)

**-** Суть успадкування полягає в тому, що абстрактний тип даних може наслідувати дані і функціональність деякого існуючого типу (батьківського класу), що сприяє повторному використанню компонентів одного класу в інших (класах-нащадках). Таким чином клас-нащадок може мати увесь функціонал батьківського класу, при цьому модифікувати його під конкретні потреби, і ці модифікації не впливатимуть на клас-предок.

(<https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)>)

**2. Що таке клас-предок?**

Клас-предок - клас, дані та методи якого наслідуються іншими класами (підкласами, класам-нащадками).

**3. Що таке клас-нащадок?**

Клас-нащадок - це клас, що наслідує дані та методи батьківського класу (або інтерфейсу) і здійснює їх реалізацію.

У мові С# клас-нащадок позначається наступним чином:

Class DerivedClassName : ParentClassName {}

(<https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))>

**4. Для чого використовують класи-предки?**

- перекривати поля, методи і властивості предка;

- перевизначати в нащадку методи з цими самими іменами і параметрами, але з іншою реалізацією тіла методу;

- перевантажувати метод, залишаючи ім’я, але змінюючи список параметрів

**5. Скільки класів-предків може мати клас?**

В мові C#, на відміну від, наприклад C++, клас може мати тільки один клас-предок.

**6. Як описати клас-нащадок?**

Клас нащадок розширює функціонал класу-предка. Зазвичай клас нащадок конкретизує модель, яку описує його клас-предок, наприклад, клас «Cat» описує більш конкретну сутність, ніж клас «Animal», від якого «Cat» успадковується.

**7. Що таке перевизначення методів?**

Перевизначення методів використовується тоді, коли потрібно змінити функціонал метода, який описаний всередині класу-предка, для класа-нащадка.

**8. Як перевизначити метод класу-предка?**

До метода класу-предка додається модифікатор virtual, котрий дозволяє перевизначити цей метод. Перевизначенний метод в класі-нащадка визначається модифікатором override. Перевизначенний метод повинен містити той самий набір аргументів, що і базовий віртуальний метод.

**7. Що таке перевизначення методів?**

Перевизначення методу в ООП — функція, що дозволяє підкласу чи нащадку класу забезпечувати специфічну реалізацію методу, вже реалізованого у одному з класів-предків

**8. Як перевизначити метод класу-предка?**

Для того, щоб метод предку можна було перевизначати, потрібно використати virtual при оголошенні методу, і при оголошенні методу класа-нащадка використати override

class Person{

…

public virtual void Action(){

....

}

}

**9. Як здійснити доступ до перевизначених методів класу-предка?**

При оголошенні методу в класі нащадку, який будем перевизначати, використаємо override.

class Person{

…

public virtual void Action(){

....

}

}

class Student : Person{

…

public override void Action(){

....

}

}

**10. Чи має об’єкт-нащадок доступ до перевизначених методів класу-предка?**

Доступ до перевизначених методів класу-предка здійсюється за допомогою ключового слова base

class Person{

…

public virtual void Action(){

....

}

}

class Student : Person{

…

public override void Action(){

....

base.Action();

}

}

**11. Які члени класу-предка є доступними для класу-нащадка?**

Для класу-нащадка доступні усі члени класу-предка, окрім тих, що оголошені з модифікатором доступу private.

**12. Чи має можливість клас-нащадок змінювати область видимості членів класу-предка?**

Клас-нащадок не має можливості змінювати область видимості членів класу-предка.

**13. Як описати члени класу-предка, щоб вони були доступними для класу-нащадка і недоступними у програмі-клієнті?**

Потрібно описати члени класу-предка з модифікатором protected.

**14. Як описати члени класу-предка, щоб вони не були доступними як для класу-нащадка, так і для програми-клієнта?**

Для цього потрібно встановити у відповідних членах класу-предку модифікатор доступу private.

[Access Modifiers - C# Programming Guide | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers)

**15. Чи може змінна типу класу-предка приймати значення об’єкта-нащадка?**

Так, може. В такому випадку усі віртуальні члени класу-предку, що присутні у класі-нащадку будуть мати реалізацію, властиву для класу-нащадка.

[Casting and type conversions - C# Programming Guide | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions)

**16. До яких полів і методів об’єкта-нащадка може здійснити доступ об’єкт-предок?**

Сам екземпляр може здійснити доступ до тих, що були успадковані об’єктом-нащадком. З тіла класу-предку доступ до полів класу-нащадку не передбачений.

**17. Які види масивів можна створювати в C#?**

В C# передбачено створення одновимірних, багатовимірних та зубчастих масивів.

[C# Arrays - javatpoint](https://www.javatpoint.com/c-sharp-arrays)

1. **Як оголошуються різні види масивів?**

Одновимірний масив - int[] nums = new int[4];

Двовимірний масив - int[,] nums2 = new int[2, 3];

Трьохвимірний масив - int[,,] nums3 = new int[2, 3, 4];

Зубчатий масив - int[][] nums = new int[3][];

1. **Оператор foreach - його синтаксис та приклад роботи з масивами**

foreach(data\_type var\_name in collection\_variable) {}

1. **Елементи класу: Rank, Length, GetLength, Clear, Sort, Clone, IndexOf, Reverse, GetLowerBound, GetUpperBound, GetType. Їх опис (що вони виконують)**

Rank - кількість вимірів масиву

Length - довжина масиву

GetLength - повертає довжину масиву

Clear - видаляє всі елементи зі списку

Sort - сортує список

Clone - робить глибоку або неглибоку копію об’єкта

IndexOf - індекс деякого елементу масиву

Reverse - змінює послідовність елементів одновимірного масиву на протилежну

GetLowerBound - повертає найменший індекс масиву (зазвичай це 0)

GetUpperBound - повертає найбільший індекс масиву

GetType - повертає тип об’єкту

**21. Загальний вигляд опису одновимірного масиву**

char M[20];

**22. Ініціалізатори одновимірних масивів.**

string[] array = new string[2]; // creates array of length 2, default values

string[] array = new string[] { "A", "B" }; // creates populated array of length 2

string[] array = { "A" , "B" }; // creates populated array of length 2

string[] array = new[] { "A", "B" }; // created populated array of length 2

**23. Виділення пам’яті для одновимірних масивів.**

Загальна форма виділення пам’яті для масиву об’єктів, що є екземплярами класу ClassName (size це розмірність), має вигляд:

ClassName classNameArray = new **ClassName**[size]

**24) Введення/виведення одновимірних масивів.**

В мові C# масиви можна вводити в час декларації двома способами:

int[] first = new int[]{1, 2, 3, 4}, та

int[] first = {1, 2, 3, 4}

Можна ініціалізувати за допомогою циклу for:

int[] arr;

for(int i = 0; i < arr.Length; i++) {

arr[i] = <значення для присвоєння>;

}

Для того щоб вивести масив, наприклад, у консоль, можна скористатися як циклом for, так і циклом foreach:

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

foreach(int val in arr) {

Console.WriteLine(val);

}

**25) Звертання до елементів одновимірного масиву.**

Для звертання до елементів одновимірного масиву в мові програмування C# використовується оператор [<індекс елемента масиву>]. Приклад використання:

int[] arr = {1, 4, 6, 7};

arr[3] = 12;

**26) Копіювання одновимірних масивів.**

В мові C# для копіювання одновимірних масивів може використовуватися метод Clone(). Варто зауважити, що цей метод ствоює неглибоку копію масиву. Також можна скористатися статичним методом Copy() класу Array. На вхід цей метод приймає 3 аргументи: перший – масив, з якого потрібно скопіювати дані, другий – масив у який потрібно скопіювати дані, третій – кількість елементів для копіювання. Також дієвим способом для копіювання масивів є прохід в циклі певного масиву з присвоєнням його елементів іншому масиву. Наприклад у нас є два масиви: int[] first = {1, 3, 5, 7} та масив int[] second = new int[4]. Тоді для копіювання елементів масиву first у масив second використаємо цикл.

for(int i = 0; i < second.Length; i++) {

second[i] = first[i];

}

**27. Опис прямокутних багатовимірних масивів.**

<https://www.bestprog.net/uk/2018/01/22/arrays-part-2-multidimensional-arrays-stepped-arrays_ua/>

<https://forum.itvdn.com/t/urok-9-pryamougolnye-massivy/3188>

Багатовимірний масив – це масив, що має два і більше вимірів. Для доступу до елементу багатовимірного масиву використовується комбінація з двох і більше індексів. Загальна форма оголошення багатовимірного масиву наступна:

тип[,...,] ім’я\_масиву = new тип[розмір1, розмір2, ..., розмірN];

Загальна форма ініціалізації багатовимірного масиву має такий вигляд:

тип[,] ім’я\_масиву = { { значення, значення, ..., значення },

{ значення, значення, ..., значення },

...

{ значення, значення, ..., значення } };

Багатовимірні масиви можуть бути двох видів:

* прямокутний масив у вигляді таблиці;
* ступінчатий масив. У цьому масиві кількість елементів кожного рядка (масиву) може бути різною. Ступінчатий масив ще називають масивом масивів.

Прямокутні масиви – масиви, які містять кілька вимірів, де всі рядки мають однакову довжину.

Для збільшення вимірності масиву потрібно в квадратних дужках дописати кому, наприклад:

byte[,] array0 = new byte[2, 2]; // створить массив 2х2

byte[ , , ] array1 = new byte[2, 2, 2]; // створить массив 2х2х2

…

Двовимірний масив можна порівняти з книжковою шафою, в якій номер полиці відповідає першому індексу, а номер книги на полиці – другому.

**28. Виділення пам’яті для прямокутних багатовимірних масивів.**

<http://cpp.dp.ua/dynamichni-masyvy/>

Багатовимірний масив – це масив, що має два і більше вимірів. Для доступу до елементу багатовимірного масиву використовується комбінація з двох і більше індексів. Загальна форма оголошення багатовимірного масиву наступна:

тип[,...,] ім’я\_масиву = new тип[розмір1, розмір2, ..., розмірN];

Багатовимірні масиви можуть бути двох видів:

* прямокутний масив у вигляді таблиці;
* ступінчатий масив. У цьому масиві кількість елементів кожного рядка (масиву) може бути різною. Ступінчатий масив ще називають масивом масивів.

Прямокутні масиви – масиви, які містять кілька вимірів, де всі рядки мають однаков Під час запуску програми операційна система налаштовує різні області пам'яті:

Під час запуску програми операційна система налаштовує різні області пам'яті:

* Глобальні змінні знаходяться в глобальному просторі імен, для якого виділяється спеціальний сегмент пам'яті.
* Регістри утворюють особливу область пам'яті, вбудовану в центральний процесор.
* Стек (stack, стек викликів) це спеціальна область пам'яті, виділена для зберігання даних окремих функцій. Ознакою стеків як структур даних є принцип last-in, first-out (LIFO, першим зайшов, останнім вийшов).
* Решта пам'яті, розподіленої для програми, – це так звана динамічна пам'ять (free store).

Загальна форма ініціалізації багатовимірного масиву має такий вигляд:

тип[,] ім’я\_масиву = { { значення, значення, ..., значення },

{ значення, значення, ..., значення },

...

{ значення, значення, ..., значення } };

Ініціалізація двовимірних (багатовимірних) масивів може бути:

* з задаванням розміру масиву;

динамічної пам’яті для матриці, що має n рядкiв і m стовпців Наприклад, розподіл та елементи цілоготипу, можна здійснити так:

int n, m;

cout << ” Введіть кількість рядків і стовбців: “;

cin >> n >> m;

int \*\*a = new int \*[n]; — оголошення змінної типу int і виділення пам’яті для масиву покажчиків на рядки матриці;

for (int і = 0; і < n; і++)

a[і] = new int [m]; — кожному елементу масиву покажчиків на рядки присвоюється адреса початку ділянки пам’яті, виділеної для рядка матриці.

* без задавання розміру масиву (“безрозмірна” ініціалізація).

Використовуючи "безрозмірну" ініціалізацію масивів, можна створювати таблиці різної довжини, даючи змогу компіляторові автоматично виділяти область пам'яті, достатню для їх зберігання.

У такому прикладі масив Array[][] оголошується як "безрозмірний":

int Array[][2] = { 1, 1, 2, 4, 3, 9, 4, 16, 5, 25, 6, 36, 7, 49, 8, 64, 9, 81, 10, 100 };

Перевага такої форми оголошення перед "габаритною" (з точним указанням усіх розмірностей) полягає у тому, що програміст може подовжувати або укорочувати таблицю значень ініціалізації, не змінюючи розмірності масиву.

При створенні динамічного багатовимірного масиву необхiдно в операції **new** вказати всі його розмірності (перший може бути змінною), наприклад:  
**int n = 5;** *// n — количество строк*  
**int \*\*m = (int \*\*) new int [n][5];.**

**29. Ініціалізатори прямокутних масивів.**

<https://lektsia.com/14x2285.html>

У мові програмування C++ передбачено можливість ініціалізації елементів масиву. Формат ініціалізації елементів масиву подібний до формату ініціалізації інших змінних:

тип ім'я\_масиву[розмір] = [перелік\_значень];

У цьому записі елемент перелік\_значень є перелік значень ініціалізації(або ініціалізаторів) елементів масиву, розділених між собою комами. Тип кожного значення ініціалізації повинен бути сумісний з базовим типом масиву (елементом тип). Перше значення ініціалізації буде збережено в першій позиції масиву, друге значення – в другій і т.д.

Наприклад, так масив Array ініціалізувався числами від 1 до 10 і квадратами цих чисел.

int Array[10][2] = {1, 1, 2, 4, 3, 9, 4, 16, 5, 25, 6, 36, 7, 49, 8, 64, 9, 81, 10, 100};

Значення в фігурних дужках ще називають ініціалізатор.

**30. Звертання до елементів прямокутних масивів.**

<https://ravesli.com/urok-78-mnogomernye-massivy/>

Для доступу до всіх елементів двовимірного масиву потрібно два цикли: один для рядків і один для стовпців. Оскільки доступ до двовимірних масивів зазвичай виконується по рядках, то лівий індекс використовується як зовнішній цикл:

for (int row = 0; row < numRows; ++row) // доступ по рядках

for (int col = 0; col < numCols; ++col) // доступ до кожного елементу в рядку

std::cout << array[row][col];

Доступ до елементів тривимірного масиву здійснюється так само, як і до елементів двовимірного масиву:

std::cout << array[3][2][1];

**31. Введення/виведення прямокутних масивів.**

У мові C# ведення з прямокутними масивами демонструють такі приклади:

int[,] arr = new[,] { { 1, 2, 3}, { 5, 6, 7 }, { 9, 10, 11 }, { 13, 14, 15 } };

int[,] arr2 = new int[]{ { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 13, 14, 15, 16 } };

string[,] studList = {{"Андрій","Банах"}, {"Ірина","Бужська"}, {"Семен","Вовк"} };

Також ввід масиву можна виконати вручну за допомогою циклу for із вкладеним циклом:

int m = 4;

int n = 5;

int[,] arr = new int [m, n];

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

{

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

{

arr[i, j] = <дані для введення>;

}

}

**32. Опис багатовимірних масивів з різними довжинами рядків.**

Багатовимірний масив з різними довжинами рядків (зубчастий масив) - це масив масивів, тобто масив елементами якого є масиви.

Оголошення масиву:

int[][] a = new int[5][];

a[0] = new int[8];

a[1] = new int[4];

На відміну від прямокутних масивів кожен індекс в зубчастих масивах виокремлено набором квадратних дужок:

a[0][0] = 100;

**33. Виділення пам’яті для багатовимірних масивів з різними довжинами рядків.**

int[][] a = new int[5][];

a[0] = new int[8];

a[1] = new int[4];

**34. Що таке інтерфейс?**

Інтерфейс – це «полегшений» клас, де всі функції є віртуальними і відсутні поля (але можуть бути властивості). Кількість функцій, визначених у конкретному інтерфейсі, залежить від того, яку поведінку ми намагаємося змоделювати за допомогою цього інтерфейсу.

**35. Що спільного між інтерфейсами та абстрактними класами?**

Абстрактні класи, у яких оголошено тільки абстрактні члени (методи, властивості, індексатори) називаються інтерфейсами. Для інтерфейсів, як і для класів, визначено поняття успадкування.

**36. Чим відрізняються інтерфейси від абстрактних класів?**

Оскільки всі члени інтерфейсу є абстрактними (реалізації для них не наводиться), то, на відміну від абстрактних класів, при описанні членів у інтерфейсі специфікатор аbstrаct не використовується. Ще однією характерною особливістю інтерфейсів є те, що всі члени інтерфейсу є неявно відкритими (public) і не можуть супроводжуватися ніякими специфікаторами доступу.

**37. Який інтерфейс є базовим для всіх інтерфейсів?**

Базовими інтерфейсами інтерфейсу є явні базові інтерфейси та їхні базові інтерфейси.

Іншими словами, набір базових інтерфейсів є повним транзитивним закриттям явних базових інтерфейсів, їх явних базових інтерфейсів тощо. Інтерфейс успадковує всі члени своїх базових інтерфейсів.

**38. Що повинен містити клас, який реалізує інтерфейс?**

Клас, який реалізує інтерфейс, повинен містити реалізацію всіх нереалізованих методів, описані в даному інтерфейсі. Статичні методи або методи з реалізацією за замовчуванням, описані в інтерфейсі, можуть залишитись без реалізації в класі.

**39. Чи обов’язково клас, який реалізує інтерфейс має містити реалізації всіх його методів?**

Так. Клас, який реалізує інтерфейс, повинен містити реалізацію всіх нереалізованих методів, описані в даному інтерфейсі. Проте, статичні методи або методи з реалізацією за замовчуванням, описані в інтерфейсі, можуть залишитись без реалізації в класі.

**40. Чи можна змінній типу інтерфейс надавати значення об’єкта типу класу, в якому реалізовано цей інтерфейс?**

Можна змінній типу інтерфейс надавати значення об’єкта типу класу, в якому реалізовано цей інтерфейс.

Triangle tr = new Triangle();

IPoint p = (IPoint) tr;

Console.WriteLine (p.GetNumberOfPoints());

**41. Чи може інтерфейс мати багато реалізацій?**

Інтерфейс може мати багато реалізацій. Оскільки за допомогою нього відбувається множинне наслідування. Якщо спадкування здійснюється не від інтерфейсу, то серед прямих «предків» клас може бути тільки один - всі інші повинні бути інтерфейсами. Імена інтерфейсів перераховуються слідом за ключовим словом implements і розділяються комами.

Наприклад:

interface CanFight {

void fight();

}

interface CanSwim {

void swim();

}

interface CanFly {

void fly();

}

class ActionCharacter {

public void fight() {}

}

class Hero extends ActionCharacter

implements CanFight, CanSwim, CanFly {

public void swim() {}

public void fly() {}

}

<https://schoolboyprog10.blogspot.com/p/83.html>

**42. У чому відміна колекцій від масивів?**

Основна відмінність колекцій від масивів полягає у функціоналі.

Масив оголошує тип елементів, що містяться в ньому, а колекція не оголошує.

Масив статичний. Примірник масиву має фіксований розмір. Після створення його ємність не можна змінити. Колекція може динамічно розширювати свою ємність і динамічно змінювати свій розмір за необхідності.

У колекціях багато різних структур даних, із різними властивостями. Наприклад, Set - безліч унікальних елементів, TreeSet - безліч унікальних відсортованих елементів, List - список, що зберігає порядок вставки і т. д. Масиви краще використовувати якщо ви працюєте з примітивними типами даних.

<https://russianblogs.com/article/33801419309/> (**прости господи**)

**43. Назвіть основні властивості та методи класу System.Array.**

**Таблиця 4.1 – Властивості класу *Array***

|  |  |
| --- | --- |
| **Властивість** | **Опис** |
| Length | Число елементів масиву |
| Rank | Розмірність масиву |

**Таблиця 4.2 – Статичні методи класу *Array***

|  |  |
| --- | --- |
| **Метод** | **Опис** |
| BinarySearch | Двійковий пошук. Визначає індекс першого входження зразка у відсортований масив, використовуючи алгоритм двійкового пошуку. |
| Clear | Виконує початкову ініціалізацію елементів. Залежно від типу елементів встановлює значення 0 для арифметичного типу, false – для логічного типу, Null – для посилань, "" – для рядків. |
| Copy | Копіювання частини або всього масиву в інший масив. |
| IndexOf | Індекс першого входження зразка в масив. |
| LastIndexOf | Індекс останнього входження зразка в масив. |
| Reverse | Обернення одномірного масиву. Виконує обернення масиву, переставляючи елементи у зворотному порядку. |
| Sort | Сортування масиву. |

**Таблиця 4.3 – Динамічні методи класу *Array***

|  |  |
| --- | --- |
| **Метод** | **Опис** |
| ToString | Перетворення елемента масиву в рядкову змінну. |
| Clone | Дозволяє створити пласку або глибоку копію масиву. У першому випадку створюються тільки елементи першого рівня, а посилання вказують на ті ж самі об'єкти. У другому випадку копіюються об'єкти на всіх рівнях. Для масивів створюється тільки пласка копія. |
| CopyTo | Копіюються всі елементи одномірного масиву в інший одномірний масив,  починаючи із заданого індексу: col1.CopyTo(col2,0); |
| GetLength | Повертає кількість елементів масиву у зазначеному вимірі. |
| GetLowerBound, GetUpperBound | Повертає нижню й верхню границю у зазначеному вимірі. Для масивів нижня границя завжди дорівнює нулю. |
| GetValue, SetValue | Повертає або встановлює значення елемента масиву із зазначеними індексами. |

<https://studfile.net/preview/9859146/page:4/>

**44. Наведіть приклади опису масивів та колекцій.**

масиви:

тип\_переменной[] название\_массива;

можна ще так

int[] nums2 = new int[4] { 1, 2, 3, 5 };

int[] nums3 = new int[] { 1, 2, 3, 5 };

int[] nums4 = new[] { 1, 2, 3, 5 };

int[] nums5 = { 1, 2, 3, 5 };

<https://metanit.com/sharp/tutorial/2.4.php>

багатовимірний масив

int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };

int[,] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };

колекції:

|  |  |
| --- | --- |
| 1 | List<string> people = new List<string>();  https://metanit.com/sharp/tutorial/4.5.php |

**45. Як передавати та повертати масиви та колекції з методів.**

Масиви можна передавати як аргументи в параметри методу. Оскільки масиви являють собою типи посилань, метод може змінювати значення елементів.

Ініціалізований одновимірний масив можна передати в метод. Наприклад, наступний оператор передає масив метод друку.

int[] theArray = { 1, 3, 5, 7, 9 };

PrintArray(theArray);

<https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/arrays/passing-arrays-as-arguments>

повернення масива з метода

/\*функции принимает три параметра,

а возврашает массив\*/

public double[] myFunction(int m, int n, int p){

double[] x=new double[2]; // объявляем массив на два элемента

x[0]=...; // заполняем элементы

x[1]=...; // теперь второй

return x;

}

<http://fkn.ktu10.com/?q=node/5593>

**46. Поясніть принцип роботи циклу foreach.**

Оператор циклу foreach працює наступним чином. При входженні в цикл змінній *identifier* присвоюється перший елемент масиву (колекції) *container*. На кожному наступному кроці ітерації вибирається наступний елемент з *container*, який зберігається у змінній *identifier*. Цикл завершується, коли буде переглянуто усі елементи масиву (колекції) *container*.

**47.** **Розкажіть про плюси та мінуси використання двозв'язних списків.**

Списки мають деякі переваги над масивами. Вони досить ефективні щодо операцій додавання або видалення елементу на початку, або у кінці списка, виконуючи їх за постійний час O(1), тоді як масиви для цього потребують часу O(n) (окрім видалення останнього елемента), тобто час зростає з ростом кількості елементів масиву. При цьому додавання/видалення довільного елемента залежить від розміру списку і складність такого алгоритму O(n).

В списках також не існує проблеми «розширення», яка рано чи пізно виникає в масивах фіксованого розміру, коли виникає необхідність включити в нього додаткові елементи. Точно так, фіксований масив, з якого було видалено багато елементів (або вони просто не використовуються) є дуже неефективним з точки зору використання пам'яті. Функціонування списків можливо в ситуації, коли пам'ять комп'ютера фрагментована, тоді як масиви переважно потребують неперервної області для зберігання.

З іншого боку, масиви дозволяють безпосередній доступ до будь-якого елементу. Однобічно зв'язані списки, натомість, потребують проходження усіх попередніх елементів. Це призводить до складнощів застосування списків в задачах, де необхідно швидко знаходити елемент за його [індексом](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_%D0%B1%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85), наприклад в [алгоритмах сортування](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F). [Кешування](https://uk.wikipedia.org/wiki/%D0%9A%D0%B5%D1%88%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) списків в таких випадках майже не дає ефекту.

Іншим очевидним недоліком списків є необхідність разом з корисною інформацією додаткового збереження інформації про вказівники, що позначається на ефективності використання пам'яті цими структурами.

**Двобічне та однобічне зв'язування**

Двобічне зв'язування потребує більше пам'яті на елемент та більших обчислювальних затрат на виконання елементар

**48. Наведіть практичні приклади ефективного використання розглянутих колекцій.**

Додавання невідому кількість елементів під час виконання програми –

використання списків є дуже ефективним.