# Лабораторна робота №2

# Використання операторів циклу та вибору мови C#

## *1. Мета завдання:*

1) Отримання практичних навиків використання операторів циклу.
2) Отримання практичних навиків використання операторів вибору.
3) Загальне практичне ознайомлення з програмами, що містять багато методів.

## *2. Теоретичні відомості*

### *2.1. Складені оператори*

До складених операторів відносять власне складені оператори та блоки. У обох випадках це послідовність операторів, взята у фігурні дужки. Блок відрізняється від складеного оператора наявністю оголошень змінних у тілі блоку.

```
{
  n++;                    //це складений оператор
  summa+=n;
}

{
  int n=0;
  n++;                    //це блок
  summa+=n;
}
```

### *2.2. Оператори вибору*

Оператори вибору – це умовний оператор і перемикач.
1. Умовний оператор має повну і скорочену форму.

```
if (вираз-умова) //скорочена форма
  оператор;
```

Як вираз-умова має використовуватися логічний вираз. Якщо значення цього виразу-умови істинне (`true`), то виконується оператор.

Наприклад:

```
if (x <= y && x <= z)
  min = x;
```

```
if (вираз-умова) //повна форма
  оператор1;
else
  оператор2;
```

Якщо значення виразу-умови істинне (`true`), то виконується `оператор1`, при хибному (`false`) значенні виразу-умови виконується `оператор2`.

```
if (d >= 0)
{
  x1 = (-b - Math.Sqrt(d)) / (2 * a);
  x2 = (-b + Math.Sqrt(d)) / (2 * a);
  Console.WriteLine("x1 = {0},  x2 = {1}", x1, x2);
}
else
  Console.WriteLine("Розв'язку нема");
```

2. Перемикач визначає множинний вибір.

```
switch (вираз)
{
  case константа1 :
    оператор1;
    break;
  case константа2 :
    оператор2;
    break;
. . . . . .
 [default:
    оператор;
    break;]
}
```

(Квадратні дужки навколо варіанту `default` позначають необов'язковість такого варіанту, а не потребу так із квадратними дужками й писати у програмі.)

При виконанні оператора `switch`, обчислюється вираз, записаний після `switch`. У багатьох С-подібних мовах є вимога, що він повинен бути цілочисельним. У C# він може бути також інших типів (у відносно старіших версіях C# – або цілочисельний, або `char`, або `string`, або `bool`, або `enum`, у найновіших – майже будь-якого типу). Якщо значення виразу, записаного після `switch`, не збіглося ні з однією константою, записаною після `case`, то виконуються оператори, які слідують за `default` (ця гілка може бути відсутня).

## *2.3. Оператори циклу*

- Цикл з передумовою:

```
while (вираз-умова)
  оператор;
```

Поки `вираз-умова` істинний, тіло циклу (`оператор`) повторюється; коли `вираз-умова` стає хибним, повтори припиняються. Оскільки `вираз-умова` вперше перевіряється перед початком циклу, такий цикл може (якщо `вираз-умова` відразу хибний) не виконатися жодного разу.

- Цикл з післяумовою:

```
do
  оператор
while (вираз-умова);
```

Тіло циклу (`оператор`) повторюється, поки `вираз-умова` істинний; коли `вираз-умова` стає хибним, повтори припиняються. Оскільки `вираз-умова` вперше перевіряється після завершення першої ітерації (проходу) циклу, такий цикл завжди виконується щонайменше один раз.

- Цикл з параметром:

```
for (вираз_1; вираз2-умова; вираз_3)
  оператор;
```

`вираз_1` визначає, з яким значенням параметра цикл починається;

`вираз2-умова` є умовою продовження циклу, цілком аналогічною умові циклу з передумовою `while` (причому, перша перевірка теж відбувається перед першою ітерацією (проходом) циклу, так що якщо `вираз2-умова` відразу хибний, `оператор` не виконається жодного разу);

`вираз_3` виконується після кожної ітерації циклу (отже, не виконується перед першою ітерацією циклу), перед тим, як повторне знаходження значення `вираз2-умова` визначає, чи буде цикл повторюватися знову.

Наприклад:

```
for(int k = 1; k <= 5; k++)
  Console.WriteLine(k);
```

виведе (у стовпчик) значення 1, 2, 3, 4, 5, бо спочатку виконується ініціалізація (надання початкового значення) `k`=1 (так написано у `вираз_1`), потім тіло циклу (виведення поточного значення змінної `k`) виконується кілька разів, спочатку при `k`=1, потім (згідно написаного у `вираз_3`) відбувається збільшення `k` на 1, і все це повторюється кілька разів; причому, після зміни `k` з 4 на 5 `вираз2-умова` все ще істинний, тому при `k`=5 тіло циклу (виведення `k`) ще виконується, а після подальшої зміни `k` з 5 на 6 `вираз2-умова` стає хибним і цикл припиняється.

Аналогічно:

| | |
|---|---|
| `for(int k = 0; k < 5; k++)`<br>`  Console.WriteLine(k);` | виведе (у стовпчик) значення 0, 1, 2, 3, 4 (починаємо з 0, і продовжуємо лише поки `k` строго менше 5, тож остання ітерація відбувається при `k`=4) |
| `for(int k = 5; k > 0; k--)`<br>`  Console.WriteLine(k);` | виведе (у стовпчик) значення 5, 4, 3, 2, 1 (починаємо з 5 і йдемо у бік зменшення `k`, останнім цілим, строго більшим 0, є 1) |
| `for(int k = 4; k >= 0; k--)`<br>`  Console.WriteLine(k);` | виведе (у стовпчик) значення 4, 3, 2, 1, 0 |

У більшості випадків `for` використовується саме у вигляді одного з чотирьох досі перелічених варіантів. Втім, принципово можливі й значно складніші конструкції. Наприклад, цілком собі працює (принаймні, у Visual Studio 2013) такий цикл:

```
for (int i = 3, j = 100; i < j; i *= 2, j--)
  Console.WriteLine("({0}, {1})", i, j);
```

Він виведе (у стовпчик) пари (3, 100), (6, 99), (12, 98), (24, 97), (48, 96), бо спочатку виконується ініціалізація (надання початкових значень) зразу двох змінних `i`=3, `j`=100, вони виводяться, потім щоразу значення змінної `i` збільшується удвічі, а змінної `j` зменшується на 1 (все це написано у `вираз_3`); це повторюється при (6, 99), (12, 98), (24, 97), (48, 96), а при (96, 95) `вираз2-умова` стає хибним і цикл припиняється. Само

собою, можливість використання у циклі `for` таких складних конструкцій не означає, ніби ними варто постійно користуватися. Як правило, якраз не варто. Як правило, варто брати щось із чотирьох раніше перелічених найстандартніших конструкцій (звісно, замінюючи нижню та верхню межі циклу відповідно до потреб конкретної задачі). Але принципова можливість писати у `for` щось складніше в мові C# є, й іноді це доцільно.

## *2.4. Оператори переходу*

Оператори переходу виконують безумовну передачу управління. Водночас, рідко коли варто передавати управління справді безумовно, тому ці оператори майже завжди використовуються у поєднанні з `if`-ом.

- `break` – оператор переривання циклу.

```
Якийсь_заголовок_циклу
{
     оператори_1;
     if (вираз_умова_виходу)
       break;
     оператори_2;
}
```

Якщо `вираз_умова_виходу` істинний, то негайно з цього місця цикл обривається (незалежно від того, істинна чи хибна «основна» умова продовження циклу). Причому, `оператори_2` (якщо такі є) теж пропускаються.

- `continue` – перехід до наступної ітерації циклу.

```
Якийсь_заголовок_циклу
{
     оператори_1;
     if (вираз_умова_пропуску)
       continue;
     оператори_2;
}
```

Якщо `вираз_умова_пропуску` істинний, то `оператори_2` пропускаються, і відбувається перехід до дій, які визначають, чи продовжувати цикл (чи виконувати наступну ітерацію). Якщо цикл є циклом `for`, то (так само, як було б і без `continue`) спочатку виконається `вираз_3`, а потім перевірятиметься `вираз2-умова`); якщо цикл є циклом `while` чи `do … while`, то (теж так само, як було б і без `continue`) негайно перевірятиметься `вираз-умова`. Залежно від результату виразу-умови, далі цикл може як продовжитися, так і завершитися; але завершення, якщо й відбудеться, то не внаслідок істинності тієї умови, котра в `if` перед `continue`, а внаслідок того, що «основна» умова продовження циклу стала хибною (`false`).

- `goto мітка` – передає управління операторові, який містить мітку. У тілі тієї ж функції повинна бути присутньою конструкція:
  `мітка: оператор;`

Оператор `goto` передає управління операторові, що стоїть після мітки. Застосування `goto` порушує принципи структурного та модульного програмування (згідно яких, кожен з блоків, з яких складається програма, повинен мати лише один вхід і лише один вихід). Тому використовувати його без крайньої потреби не слід. Зокрема, використання

оператора `goto` може бути (а може й не бути) виправданим, якщо необхідно виконати перехід з декількох вкладених циклів або перемикачів вниз по тексту програми.

- `return` – оператор повернення з функції (методу). Він завжди завершує виконання методу (функції) і передає управління в точку його (її) виклику. У частковому випадку, якщо він вживається у методі `Main`, він по суті завершує виконання всієї програми. Вигляд оператора:
  `return [вираз];`
  Квадратні дужки тут знов позначають необов'язковість, а не потребу так і писати квадратні дужки в самому тексті програми. Тобто, `return` може використовуватися із виразом після нього, а може без такого виразу. Але це не на вільний вибір програміста, а цілком визначається типом функції (методу). Якщо (як рекомендується в цій лабораторній) функція (метод) має тип `void`, то `return` слід використовувати лише без виразу, а якщо інший тип, то лише із виразом.

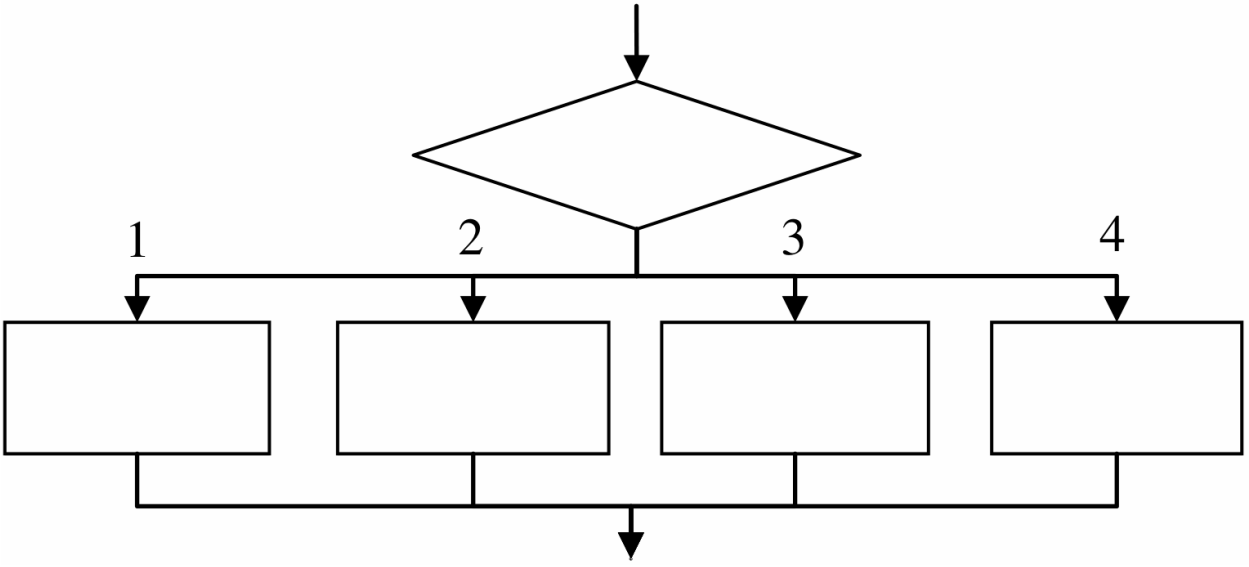
## *2.5. Зображення розглянутих алгоритмічних конструкцій на блок-схемах*

Зображення на блок-схемі кожного з циклів вже згадувалося наприкінці лабораторної роботи №1.

Зображення на блок-схемі `break` (того, що впливає на цикл, а не `switch`) полягає головним чином у тому, що стрілочка переходить на перший оператор після циклу (чи, якщо цикл наприкінці методу, на блок «Кінець»). При цьому можна додатково також підписати стрілочку словом «break», але це другорядне й не обов'язкове.

Із зображенням на блок-схемі `continue` аналогічно (головне – стрілочка, яка веде до перевірки, чи продовжувати цикл, і цю стрілочку можна при бажанні додатково підписати словом «continue»).

Зображення на блок-схемі `switch` має приблизно такий вигляд:

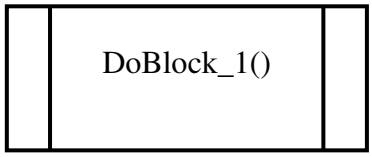



Всередині ромба пишеться те, по чому береться `switch`; на відміну від решти ситуацій, тут у ромбі зазвичай буде числове чи рядкове значення (а не логічне). Варіанти (котрі на цьому рисунку-прикладі підписані як «1», «2», «3», «4») відповідають `case`-випадкам, у разі наявності `default`-випадку він зображається аналогічно, зі словом «default». На рисунку-прикладі в кожному з варіантів є лише одна дія (один прямокутник), але якщо їх треба кілька, або всередині гілки є якісь складніші алгоритмічні конструкції, їх слід просто зображати відповідними кількома блоками замість відповідного прямокутника.

При цьому, ті `break`-и, котрі є складовими кожен своєї гілки `switch`, слід просто не зображати.

Зображення на блок-схемі `return` залежить від того, чи йдеться про `return` з функції типу `void` з метою дострокового завершення, чи про `return` з метою повернення значення з методу (функції) іншого (не `void`) типу. У першому випадку (дострокове завершення функції типу `void`) головне – стрілочка, яка веде до блоку «Кінець», її можна (не обов'язково) підписати словом «return», але робити блок не варто. У другому випадку (повернення значення методом іншого типу; конкретно у лабораторній №2 такого ніби не повинно бути, але повинно бути в подальших лабораторних) це слід відобразити аналогічно виведенню (або паралелограмом, або тим блоком, який зліва, згори і справа прямокутник, а знизу одна велика хвиля), тільки вказуючи слово «return» (або «повернення»); звісно, після такого блоку теж повинна бути стрілка, що веде до блоку «Кінець».

Щоб зобразити алгоритм, що містить допоміжні алгоритми (програму, що містить кілька методів) блок-схемами, зображають окрему блок-схему для кожного методу (включаючи `Main`). У блоці «початок» решти, крім `Main`, функцій (методів) слід писати не саме лише слово «початок», а крім нього ще й назву відповідного метода та список параметрів (конкретно в цій лабораторній рекомендується зробити все без параметрів, відповідно й писати їх не треба; однак, якщо самі зробите методи з параметрами — зображайте їх у блоку початку).




Якщо блок має форму прямокутника і при цьому містить виклик написаної Вами функції (методу), то ліва і права сторони цього прямокутника повинні бути подвоєними лініями. Втім, це правило стосується лише прямокутників. Якщо виклик функції (методу) потрапляє у розгалуження чи виведення, то слід використати відповідний стандартний блок розгалуження чи стандартний блок виведення, тож у цих випадках форма блоку не буде звертати увагу, що відбувається виклик функції (методу).)

## *3. Постановка завдання*

Розв'язати вказані згідно розподілених варіантів три завдання, використовуючи основні оператори мови C#. Якщо при вирішенні вказаних Вам трьох завдань Ви використаєте всі три типи циклів (`for`, `while`, `do … while`), бали за лабораторну будуть дещо збільшені.

***Обов'язково*** зробити всі три завдання в одній програмі, ***обов'язково*** винісши кожне з них в окремий метод. Дозволяється (а тим, хто сумнівається у своїх силах, наполегливо рекомендується) взяти за основу опублікований разом із завданням файл `Program.cs` і залишити метод `Main` незмінним, дописавши вміст методів `DoBlock_1()`, `DoBlock_2()`,`DoBlock_3()`.

Варіанти:

1. Дана послідовність з n цілих чисел. Знайти середнє арифметичне цієї послідовності.
2. Дана послідовність з n цілих чисел. Знайти суму парних (за значенням) елементів цієї послідовності.
3. Дана послідовність з n цілих чисел. Знайти суму елементів з парними номерами з цієї послідовності.
4. Дана послідовність з n цілих чисел. Знайти суму непарних (за значенням) елементів цієї послідовності.

5. Дана послідовність з n цілих чисел. Знайти суму елементів з непарними номерами з цієї послідовності.
6. Дана послідовність з n цілих чисел. Знайти мінімальний елемент в цій послідовності.
7. Дана послідовність з n цілих чисел. Знайти номер максимального елементу в цій послідовності.
8. Дана послідовність з n цілих чисел. Знайти номер мінімального елементу в цій послідовності.
9. Дана послідовність з n цілих чисел. Знайти максимальний елемент в цій послідовності.
10. Дана послідовність з n цілих чисел. Знайти суму мінімального і максимального елементів в цій послідовності.
11. Дана послідовність з n цілих чисел. Знайти різницю максимального і мінімального елементів в цій послідовності.
12. Дана послідовність з n цілих чисел. Знайти кількість непарних елементів цієї послідовності.
13. Дана послідовність з n цілих чисел. Знайти кількість парних елементів цієї послідовності.
14. Дана послідовність з n цілих чисел. Знайти кількість елементів цієї послідовності, кратних числу K (число K вводиться додатково на самому початку перед n).
15. Дана послідовність з n цілих чисел. Знайти кількість елементів цієї послідовності, кратних її першому елементу.
16. Дана послідовність з n цілих чисел. Знайти кількість елементів цієї послідовності, кратних числу K1 і не кратних числу K2 (числа K1 та K2 вводяться додатково на самому початку перед n).
17. Дана послідовність з n цілих чисел. Визначити, яких чисел в цій послідовності більше: додатних чи від'ємних.
18. Дана послідовність цілих чисел, за якою слідує 0. Знайти середнє арифметичне цієї послідовності.
19. Дана послідовність цілих чисел, за якою слідує 0. Знайти суму парних (за значенням) елементів цієї послідовності.
20. Дана послідовність цілих чисел, за якою слідує 0. Знайти суму елементів з парними номерами з цієї послідовності.
21. Дана послідовність цілих чисел, за якою слідує 0. Знайти суму непарних (за значенням) елементів цієї послідовності.
22. Дана послідовність цілих чисел, за якою слідує 0. Знайти суму елементів з непарними номерами з цієї послідовності.
23. Дана послідовність цілих чисел, за якою слідує 0. Знайти мінімальний елемент в цій послідовності.
24. Дана послідовність цілих чисел, за якою слідує 0. Знайти номер максимального елементу в цій послідовності.
25. Дана послідовність цілих чисел, за якою слідує 0. Знайти номер мінімального елементу в цій послідовності.
26. Дана послідовність цілих чисел, за якою слідує 0. Знайти максимальний елемент в цій послідовності.
27. Дана послідовність цілих чисел, за якою слідує 0. Знайти суму мінімального і максимального елементів в цій послідовності.
28. Дана послідовність цілих чисел, за якою слідує 0. Знайти різницю мінімального і максимального елементів в цій послідовності.
29. Дана послідовність цілих чисел, за якою слідує 0. Знайти кількість непарних елементів цієї послідовності.

30. Дана послідовність цілих чисел, за якою слідує 0. Знайти кількість парних (за значенням) елементів цієї послідовності.
31. Дана послідовність цілих чисел, за якою слідує 0. Знайти кількість елементів цієї послідовності, кратних числу K (число K вводиться додатково на самому початку).
32. Дана послідовність цілих чисел, за якою слідує 0. Знайти кількість елементів цієї послідовності, кратних її першому елементу.
33. Дана послідовність цілих чисел, за якою слідує 0. Знайти кількість елементів цієї послідовності, кратних числу K1 і не кратних числу K2 (числа K1 та K2 вводяться додатково на самому початку перед n).
34. Дана послідовність цілих чисел, за якою слідує 0. Визначити, яких чисел в цій послідовності більше: додатних чи від'ємних.
35. $S = 1 - 2 + 3 - 4 + 5 - \ldots$, всього $n$ доданків;
36. S=1+3+5+7+ ., всього n доданків;
37. S=1+2–3+4+5–6+7+8–9+…, всього $n$ доданків;
38. S=15+17–19+21+23–25+…, всього $n$ доданків;
39. $S = \sin X + \sin X^2 + \sin X^3 + \sin X^4 + \ldots + \sin X^n$;
40. $S = \sin X + \sin^2 X + \sin^3 X + \sin^4 X + \ldots + \sin^n X$;
41. $S = \sqrt{3} + \sqrt{6} + \sqrt{9} + \ldots + \sqrt{3n}$;
42. $S = \sin x + \cos 2x + \sin 3x + \cos 4x + \sin 5x + \cos 6x + \ldots$ (до $\sin(nx)$ чи $\cos(nx)$ включно, $\sin(nx)$ чи $\cos(nx)$ залежить від парності $n$);
43. Знайти перше від'ємне число послідовності $u_i$=cos(ctg($i$)), де $i$ =1, 2, 3, ... (гарантовано, що серед цих чисел є від'ємні).
44. Визначити чи є число $K$ степенем 3.
45. Визначити чи є число $K$ простим.
46. Дана послідовність з $n$ чисел. Знайти номер першого від'ємного числа (або з'ясувати, що таких нема).
47. Знайти кількість цифр в десятковому натуральному числі $K$.
48. Знайти суму цифр в десятковому натуральному числі $K$.
49. Сформувати $n$ чисел Фібоначчі ($a_1$=1, $a_2$=1, $a_i$=$a_{i-1}$+$a_{i-2}$).
50. Сформувати всі числа Фібоначчі, що не перевищують задане число $Q$.
51. Дано натуральне число $K$. Визначити, чи є воно числом Фібоначчі.
52. $P = \frac{2}{3} \cdot \frac{4}{5} \cdot \frac{6}{7} \cdot \ldots \cdot \frac{2N}{2N+1}$.
53. $P = a \cdot (a+1) \cdot \ldots \cdot (a+n-1)$.
54. $S = \frac{1}{a} + \frac{1}{a^2} + \frac{1}{a^4} + \frac{1}{a^8} + \ldots + \frac{1}{a^{2^n}}$.
55. $P = \frac{(x-1)(x-3)(x-7) \cdot \ldots \cdot (x-63)}{(x-2)(x-4)(x-8) \cdot \ldots \cdot (x-64)}$.
56. $P = (1+\sin 0{,}1)(1+\sin 0{,}2)\ldots\ldots(1+\sin 10)$.
57. $P = (1-\frac{1}{2^2})(1-\frac{1}{3^2}) \cdot \ldots \cdot (1-\frac{1}{n^2})$, де n>2.
58. $P = (1-\frac{1}{2})(1-\frac{1}{4})(1-\frac{1}{6}) \cdot \ldots \cdot (1-\frac{1}{2n})$
59. $S = \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \ldots + \frac{1}{(2n+1)^2}$

Лише охочим, лише з числа тих, хто захистить свої варіанти щонайменше за тиждень до кінця відведеного терміну, можна виконати на додаткові бали також такі завдання підвищеної складності:

60. $S = \sqrt{3+\sqrt{6+\sqrt{9+....\sqrt{3n}}}}$ ;
61. $S = \sin(x+\sin(2x-\sin(3x+\sin(4x+\sin(5x-\sin(6x+...)...)$ (до $\sin(nx)$ включно; на кожні три рази двічі відбувається додавання, один раз віднімання);
62. $S = \sin(x+\cos(2x+\sin(3x+\cos(4x+\sin(5x+\cos(6x+...)...)$ (до $\sin(nx)$ чи $\cos(nx)$ включно, $\sin(nx)$ чи $\cos(nx)$ залежить від парності $n$);
63. $S = \sin(x+\cos(2x-\sin(3x+\cos(4x+\sin(5x-\cos(6x+...)...)$ (до $\sin(nx)$ чи $\cos(nx)$ включно, $\sin(nx)$ чи $\cos(nx)$ залежить від парності $n$; на кожні три рази двічі відбувається додавання, один раз віднімання).

## *4. Загальні рекомендації*

1. Вимога зробити всі три завдання в одній програмі, але рознісши по різним методам, абсолютно обов'язкова.
2. Введення даних у завданнях здійснюється з клавіатури (за виключенням деяких варіантів завдання №3, де нічого вводити не треба). У завданні №1, з клавіатури слід ввести також і кількість елементів. А в завданні №2 вводити кількість елементів заборонено, там програма повинна читати, поки не дійде до 0, не знаючи наперед кількості елементів. Але і в завданні 1, і в завданні 2 треба вводити самі елементи. Ви не знаєте наперед, якими будуть елементи. Які введе користувач, такі й будуть. (Однак, все-таки відомо, що цілі, і що поміщаються в тип `int`.)
3. У завданні №2 значення 0 слідує після елементів послідовності, не будучи частиною цієї послідовності (його треба використати як межу вхідних даних, але треба не вважати частиною цих даних). Для деяких варіантів це байдуже (наприклад, якщо треба рахувати суму, то байдуже, чи буде додано до неї зайвий 0, чи не буде), а для деяких може істотно вплинути на відповідь (наприклад, якщо треба знайти мінімум, і всі «справжні» значення додатні, то знайти слід мінімум лише серед цих додатних, не «зіпсувавши» його нулем).
4. Наполегливо рекомендується виконати всі завдання, ніде не використовуючи масиви. Використання масивів не те щоб зовсім заборонене, але не рекомендується, і за їх використання бали будуть зменшені.
5. Наполегливо рекомендується виконати всі завдання, ніде не використовуючи рядки (`string`). Використання рядків не те щоб зовсім заборонене, але не рекомендується, і за їх використання бали будуть зменшені. (Звісно, «використання», коли рядок читається з клавіатури й тут же перетворюється в `int` чи `double` тут не рахується, мова про крайню небажаність більш істотного використання.)
6. При вирішенні завдання №1, як правило, доцільно використати цикл з параметром (тобто `for`), оскільки кількість елементів послідовності відома наперед (читається з клавіатури перед початком циклу), а при вирішенні завдання №2 – циклом з передумовою. Однак, ця рекомендація може входити у протиріччя з рекомендацією використати у трьох завданнях всі три види циклів, тому остаточне рішення, що де використати, за вами.

## *6. Зміст звіту та інших матеріалів, які слід здати*

1. Треба здати в гуглклас звіт, у форматі PDF. Звіт має містити всі такі складові:
   1.1. Постановка завдання для конкретного варіанту (для кожного з трьох завдань).
   1.2. Алгоритм вирішення завдання у вигляді блок-схеми (як описано вище, для Main і кожного з додаткових методів).
   1.3. Текст програми для вирішення завдань мовою C# (один текст однієї програми, куди три розв'язки трьох завдань включені як методи). Цей текст програми слід і включити у звіт, і приєднати в гуглклас окремим файлом (див. далі).

1.4. Опис (текстом, своїми словами) формату вхідних даних, тобто що за чим вводити для завдання №1, для завдання №2, для завдання №3; якщо введення у `Main` абсолютно таке ж, як у наданому зразку, то написати «введення у `Main` згідно наданого зразку», а якщо відрізняється, то введення у `Main` також розписати своїми словами. Це особливо важливо при здачі в гуглклас без захисту, зокрема, тому, що в досить простих випадках викладачу може бути зручніше виконувати Вашу програму на сайті ideone.com (див. також наступний пункт), а там інтерактивності не виходить, треба спочатку ввести вхідні дані, потім запустити.

1.5. Посилання на сайт ideone.com, де розміщена ця сама програма. Це повинно бути посилання, подібне до https://ideone.com/4IckXZ (тільки з іншим кодом наприкінці). Само собою, треба не написати неіснуюче посилання, а справді викласти свій розв'язок на сайт ideone.com. Це повинно бути одне посилання, за яким повинна бути одна програма, куди три розв'язки трьох завдань включені як методи.

1.6. Приклад(и) вхідних даних та результатів, причому обов'язково окремо для завдання №1, окремо до завдання №2, окремо до завдання №3, і обов'язково з деякими мінімальними коментарями (як-то «ми ввели послідовність з трьох чисел, це були числа 3, 14 та 5, програма порахувала, що їх сума дорівнює 22, і це правильно, бо справді просили порахувати суму, і справді 3+14+5=22», тільки відповідно до Вашого варіанту та Вашого прикладу). До речі, це означає також необхідність вибирати якісь прості приклади, щоб можна було пояснити, чому це правильно-то... (У завданні 3 є деякі варіанти, де неможливо вибрати щось просте; їх це не стосується; однак скрізь, де можна вибрати простий приклад, слід так і зробити.)

1.7. В кінці звіту повинен бути один короткий висновок до лабораторної в цілому.

2. Той самий, вже включений у звіт, текст програми приєднати також у вигляді окремого .cs-файлу. Оскільки сказано включити всі три блоки в одну програму, це буде один .cs-файл (якщо Ви самі нічого не мінятимете, він називатиметься `Program.cs`).
3. Те саме, вже включене у звіт, посилання на Вашу програму на сайті `ideone.com`

Вимога викласти розв'язки на сайт `ideone.com` не означає, ніби ви повинні там і писати свої програми. Якщо раптом Вам так зручно – в принципі можна. Але, як правило, значно зручнішим є повноцінне середовище (Visual Studio, Rider, тощо) на своєму комп'ютері. Викласти на `ideone.com` пропонується вже наприкінці. Однак, наполегливо рекомендується все ж перевірити, чи працює Ваша програма також і безпосередньо на сайті `ideone.com`.