

Національний університет «Одеська політехніка»
Кафедра кібербезпеки та програмного забезпечення

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни «Програмування»
першого (бакалаврського) рівня вищої освіти
спеціальності 125 – Кібербезпека

Одеса, 2022

Національний університет «Одеська політехніка»
Кафедра кібербезпеки та програмного забезпечення

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни «Програмування»
першого (бакалаврського) рівня вищої освіти
спеціальності 125 – Кібербезпека

Затверджено
на засіданні кафедри КБПЗ
Протокол № 1 від 26.08.22 р.

Одеса, 2022

Методичні вказівки до лабораторних робіт з дисципліни «Програмування» першого (бакалаврського) рівня вищої освіти спеціальності 125 – Кібербезпека / Укл.: *О.Ю. Лебедєва*. - Одеса, 2022. – __ с.

Укладач: **Лебедєва О.Ю.**, к.т.н., доцент

ЗМІСТ

Лабораторна робота № 1. Системи числення.	5
1. Теоретичний матеріал	5
2. Завдання до лабораторної роботи №1	8
Лабораторна робота № 2. Двійкові операції	10
1. Теоретичний матеріал	10
2. Завдання до лабораторної роботи №2	15
Лабораторна робота № 3. Створення блок-схем	17
1. Теоретичний матеріал	17
2. Завдання до лабораторної роботи №3	20
Лабораторна робота № 4. Типи даних. Розрахунок математичних виразів	22
1. Теоретичний матеріал	22
2. Завдання до лабораторної роботи №4	22
Лабораторна робота № 5. Програмування алгоритмів, що розгалужуються	23
1. Теоретичний матеріал	23
2. Завдання до лабораторної роботи №5	23
Лабораторна робота № 6. Програмування циклів	24
1. Теоретичний матеріал	24
2. Завдання до лабораторної роботи №6	24
Лабораторна робота № 7. Одновимірні масиви	25
1. Теоретичний матеріал	25
2. Завдання до лабораторної роботи №7	25
Лабораторна робота № 8. Створення власних функцій	26
1. Теоретичний матеріал	26
2. Завдання до лабораторної роботи №8	26
Лабораторна робота № 9. Двовимірні масиви	27
1. Теоретичний матеріал	27
2. Завдання до лабораторної роботи №9	27
Лабораторна робота № 10. Робота зі строками	28
1. Теоретичний матеріал	28
2. Завдання до лабораторної роботи №10	28
Лабораторна робота № 11. Створення віконних додатків	29
1. Теоретичний матеріал	29
2. Завдання до лабораторної роботи №11	29

Лабораторна робота № 1. Системи числення.

Мета роботи: ознайомитися з позиційними системами числення. Навчитися переводити числа з однієї системи числення до іншої.

1. Теоретичний матеріал

Різні системи рахунку і записи чисел тисячоліттями існували і змагалися між собою, але до кінця «докомп'ютерної епохи» особливу роль за рахунку стало грати число десять, а найпопулярнішою системою кодування чисел виявилася так звана *десятькова позиційна система*. У цій системі значення цифри в числі залежить від місця (позиції) всередині числа.

У десятичній системі лише десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, але інформацію несе не тільки цифра, але також і місце, на якому вона стоїть. Серед 444, наприклад, три однакові цифри позначають кількість і одиниць, і десятків, і сотень. А ось в числі 400 перша цифра 4 позначає число сотень, а дві цифри 0 власними силами внесок у число не дають, а потрібні лише для вказівки позиції цифри 4.

Отже, у десятичній позиційній системі числення особливу роль відіграє число десять та його ступеня: 10, 100, 1000, 10000, ...

Найправіша цифра числа показує число одиниць, наступна цифра – число десятків, наступна – число сотень тощо. Наприклад, число 1995 становлять: 5 одиниць, 9 десятків, 9 сотень і тисяча:

$$1995 = 5 + 9 \cdot 10 + 9 \cdot 100 + 1 \cdot 1000.$$

Оскільки $1000 = 10^3$, $100 = 10^2$, $10 = 10^1$. Для одноманітності можна замінити 5 на $5 \cdot 10^0$. Тоді можна написати ще й так

$$1995 = 5 \cdot 10^0 + 9 \cdot 10^1 + 9 \cdot 10^2 + 1 \cdot 10^3.$$

Число 10, ступеня якого записані у формулі вище, називають *основою системи числення*. Основа позиційної системи числення дорівнює кількості цифр в алфавіті системи числення.

Вибір числа 10 як основа позиційної системи значною мірою пояснюється традицією, а не якимись чудовими властивостями числа 10. Можна розглянути й системи числення з іншою основою р.

Записати число N у р-вій системі числення – це означає записати його у вигляді

$$N = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \dots + a_1 \cdot p + a_0 \quad (1)$$

де кожен із коефіцієнтів a_i може бути 0, 1, 2, ..., $p - 1$, причому старша цифра a_n ненульова.

Взявши основу рівним 2, отримуємо двійкову систему числення всього з двома цифрами 0 та 1.

Зазвичай при записі числа значення основи системи обчислення записується у вигляді нижнього індексу після останньої цифри числа.

Приклади найчастіше використовуваних систем числення:

Систем числення	Основа (p)	Алфавіт системи числення	Приклад запису
Двійкова	2	0, 1	101101 ₂
Восьмирічна	8	0, 1, 2, 3, 4, 5, 6, 7	12345 ₈
Десятькова	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	1234 ₁₀
Шістнадцятирічна	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9,	F4D9 ₁₆

		A (=10), B (=11), C (=12), D (=13), E (=14), F(=15)	
--	--	--	--

Переведення цілого десяткового числа в недесяткову систему числення виконується шляхом послідовного ділення числа з залишком на основу системи числення з наступним записом отриманого результату та залишків на кожному кроці поділу в порядку, зворотному порядку їх отримання. Ділення проводиться до тих пір, поки отриманий на черговому кроці результат не буде меншим за основу системи числення.

Наприклад, переведемо число 12345_{10} у трійкову систему числення:

$$\begin{array}{r}
 12345 \div 3 = 4115 \text{ (залишок 0)} \\
 4115 \div 3 = 1371 \text{ (залишок 2)} \\
 1371 \div 3 = 457 \text{ (залишок 0)} \\
 457 \div 3 = 152 \text{ (залишок 1)} \\
 152 \div 3 = 50 \text{ (залишок 2)} \\
 50 \div 3 = 16 \text{ (залишок 2)} \\
 16 \div 3 = 5 \text{ (залишок 1)} \\
 5 \div 3 = 1 \text{ (залишок 2)} \\
 1 \div 3 = 0 \text{ (залишок 1)}
 \end{array}$$

Залишки зчитуються знизу вгору: 121221020.

Результат: $12345_{10} = 121221020_3$

Переклад числа з недесяткової системи числення у десяткову здійснюється шляхом виконання обчислень за розгорнутим записом вихідного числа (1).

Наприклад, переведемо числа 101110_2 у десяткову систему числення:

$$101110_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 46_{10}$$

Переведення десяткового дробу в не десяткову систему числення виконується шляхом послідовного множення числа на основу системи числення з відкиданням одержуваних цілих частин на кожному кроці множення і наступним записом отриманих значень цілих частин по порядку їх отримання. Умноження проводиться до одержання значення з нульовою дробовою частиною або до досягнення необхідної точності подання дробу (необхідної кількості значущих цифр після коми).

Наприклад, потрібно перевести число $0,123_{10}$ в п'ятирічну систему числення з точністю до 5 значущих цифр після коми:

$$0,123 \cdot 5 = 0,615$$

$$0,615 \cdot 5 = 3,075$$

$$0,075 \cdot 5 = 0,375$$

$$0,375 \cdot 5 = 1,875$$

$$0,875 \cdot 5 = 4,375$$

Результат: $0,123_{10} = 0,03014$

Переведення дійсного десяткового числа в недесяткову систему числення виконується у два етапи:

1) окремо здійснюється переведення цілої частини числа шляхом послідовного поділу на основу системи числення;

2) окремо виконується переведення дробової частини числа шляхом послідовного

множення на основу системи числення.

Запис цілої частини числа в системі обчислення доповнюється правою комою і записом дробової частини в системі числення.

Приклад: потрібно перевести число $15,12_{10}$ у п'ятирічну систему числення з точністю до 5 цифр після коми:

- 1) $15_{10} = 30_5$;
- 2) $0,12_{10} = 0,03_5$.

В результаті: $15,12_{10} = 30,03_5$.

Переведення чисел між недесятковими системами числення зазвичай найзручніше проводити через десятичну систему числення:

- 1) перевести вихідне число до десяткової системи числення;
- 2) перевести отримане десяткове число в необхідну систему числення.

Переведення чисел між системами числення з кратними основами. Якщо основи вхідної та кінцевої системи кратні одна одній, то переведення чисел між цими системами числення можна виконувати за спрощеною схемою.

Переведення *двійкового числа у вісімкову систему числення* провадиться за тріадами цифр:

- вхідне двійкове число розбивається на групи по три цифри ("тріади") з права наліво; при необхідності крайня ліворуч група цифр доповнюється незначними нулями ліворуч;
- кожна тріада двійкових цифр замінюється відповідним їй вісімковим значенням відповідно до таблиці:

Двійкова тріада	000	001	010	011	100	101	110	111
Вісімкове значення	0	1	2	3	4	5	6	7

Наприклад: потрібно перевести число 1011010_2 у вісімкову систему числення:

001 011 010
1 3 2

В результаті: $1011010_2 = 132_8$

Переклад *восьмеричного числа в двійкову систему числення* також провадиться за тріад цифр:

- вхідне вісімкове число розбивається на окремі цифри;
- кожна вісімкова цифра замінюється відповідною їй тріадою двійкових цифр за таблицею (див. вище);
- шукане двійкове число складається з одержаних тріад; незначні нулі зліва відкидаються.

Наприклад, потрібно перевести число 12345_8 у двійкову систему числення:

1 2 3 4 5
001 010 011 100 101

В результаті: $12345_8 = 1010011100101_2$

Переклад *двійкового числа в шістнадцяткову систему числення* провадиться за тетрадами цифр:

- вхідне двійкове число розбивається на групи по чотири цифри («тетрад») з права на наліво; при необхідності крайня ліворуч група цифр доповнюється незначними нулями ліворуч;
- кожен тетрад двійкових цифр замінюється відповідним їй шістнадцятковим значенням згідно з таблицею:

Двійкова тетрада	0000	0001	0010	0011	0100	0101	0110	0111
Шістнадцяткове значення	0	1	2	3	4	5	6	7
Двійкова тетрада	1000	1001	1010	1011	1100	1101	1110	1111
Шістнадцяткове значення	8	9	A	B	C	D	E	F

Наприклад, потрібно перевести число 1011010_2 в шістнадцяткову систему числення:

0101 1010
5 A

В результаті: $1011010_2 = 5A_{16}$

Переведення шістнадцяткового числа в двійкову систему числення також проводиться за зошитами цифр:

- вхідне шістнадцяткове число розбивається деякі цифри;
- кожна шістнадцяткова цифра замінюється з відповідним їй тетрадом двійкових цифр за таблицею (див. вище);
- шукане двійкове число складається з отриманих тетрад; незначні нулі зліва відкидаються.

Наприклад, потрібно перевести число $1ADA_{16}$ в двійкову систему числення:

1 A D A
0001 1010 1101 1010

В результаті: $1ADA_{16} = 1101011011010_2$

Переведення цілого числа між шістнадцятковою та вісімковою системами числення можна використовувати їх зв'язок з двійковою системою. Для цього треба:

- отримати двійкове число;
- розбити його на тетради (для шістнадцяткової системи) або на тріади (для вісімкової системи);
- записати їх цифрами необхідної системи числення.

Наприклад:

$$35_8 = 011101_2 = 00011101_2 = 1D_{16}$$

$$1A_{16} = 00011010_2 = 011010_2 = 32_8$$

2. Завдання до лабораторної роботи №1

1. Перевести наступні числа з десяткової системи числення в двійкову:

- 9490;
- 763;
- 994,125;

— 523,25;

— 203,82;

2. Перевести наступні числа з десяткової системи числення в шістнадцяткову:

— 123456;

— 563;

— 234,25;

— 53181,125;

— 286,16.

3. Перевести наступні числа з двійкової системи числення в десяткову:

— 111000111₂;

— 100011011₂;

— 1001100101,1001₂;

— 1001001,011₂;

— 1100010010₂.

4. Перевести наступні числа з шістнадцяткової системи числення в десяткову:

— 3CD,78₁₆;

— FF12₁₆;

— 987A,A₁₆;

— 7A5F3D₁₆;

— FFFF₁₆.

5. Перевести наступні числа з вісімкової системи числення в шістнадцяткову:

— 3567₈;

— 1234₈;

— 775₈;

— 5731₈;

— 111₈.

Лабораторна робота № 2. Двійкові операції

Мета роботи: ознайомитися з двійковими операціями, а саме: додавання, віднімання, множення, ділення, логічні операції та зсуви. Навчитися виконувати на практиці двійкові операції.

1. Теоретичний матеріал

Виконання арифметичних дій у будь-яких позиційних системах числення проводиться за тими самими правилами, що використовуються в десятковій системі числення.

Так само, як і в десятковій системі числення, для виконання арифметичних дій необхідно знати таблиці складання (віднімання) та множення.

Додавання в двійковій системі числення виконується за тими самими правилами, що й у десятковій. Два числа записуються в стовпчик з вирівнюванням по роздільнику цілої та дробової частини та за необхідності доповнюються незначними нулями ліворуч для цілої частини та праворуч для дробової частини. Додавання починається з крайнього правого розряду.

Маємо наступну таблицю додавання для двійкової системи числення:

Сложение
$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 10$

Розглянемо приклад, виконаємо додавання чисел $1011,1_2 + 1010,11_2$:

$$\begin{array}{r}
 1 \leftarrow \quad 1 \leftarrow 1 \leftarrow 1 \leftarrow \\
 \begin{array}{r}
 1 \ 0 \ 1 \ 1,1 \ 0 \\
 + \ 1 \ 0 \ 1 \ 0,1 \ 1 \\
 \hline
 1 \ 0 \ 1 \ 1 \ 0,0 \ 1
 \end{array}
 \end{array}$$

Цікавою є також ситуація, коли складаються більше двох чисел. В цьому випадку можливе перенесення через кілька розрядів. Наприклад виконаємо додавання чисел $111,1_2 + 111_2 + 101,1_2$:

$$\begin{array}{r}
 1 \leftarrow \quad 1 \leftarrow \quad 1 \leftarrow \quad 1 \leftarrow \\
 \begin{array}{r}
 1 \ 1 \ 1,1 \\
 + \ 1 \ 1 \ 1,0 \\
 \hline
 1 \ 0 \ 1,1 \\
 \hline
 1 \ 0 \ 1 \ 0 \ 0,0
 \end{array}
 \end{array}$$

Разряды: 4 3 2 1 0 -1

При додаванні в розряді одиниць (розряд 0) виявляється 4 одиниці, які, об'єднавшись, дають 100_2 . Тому з нульового розряду до першого розряду переноситься 0, а до другого – 1. Аналогічна ситуація виникає у другому розряді, де з урахуванням двох перенесених одиниць виходить число 101_2 . 1 залишається у другому розряді, 0 переноситься у третій та 1 переноситься у четвертий.

Маємо наступну таблицю віднімання для двійкової системи числення:

Вычитание
$0 - 0 = 0$
$1 - 0 = 1$
$1 - 1 = 0$
$10 - 1 = 1$

У разі віднімання в поточному розряді з нуля одиниці відбувається позика зі старшого розряду. По суті, ми віднімаємо не з одиниці, а з двійкового числа 10. Якщо займається одиниця через кілька розрядів, то вона дає по одній одиниці у всіх проміжних нульових розрядах і 10 у тому розряді, для якого займалася.

Розглянемо приклад, виконаємо віднімання чисел $10110,01_2 - 1001,1_2$

$$\begin{array}{r}
 \overset{\rightarrow 10}{1} \overset{\rightarrow 1}{0} \overset{\rightarrow 10}{1} \overset{\rightarrow 1}{1} \overset{\rightarrow 10}{0}, \overset{\rightarrow 10}{0} \overset{\rightarrow 10}{1} \\
 - \quad 1 \ 0 \ 0 \ 1, 1 \ 0 \\
 \hline
 1 \ 1 \ 0 \ 0, 1 \ 1
 \end{array}$$

Маємо наступну таблицю множення для двійкової системи числення

Умножение
$0 \cdot 0 = 0$
$0 \cdot 1 = 0$
$1 \cdot 0 = 0$
$1 \cdot 1 = 1$

Множення багаторозрядних двійкових чисел виконується в стовпчик аналогічно до множення десяткових чисел. Однак при цьому для двійкової системи числення можна скористатися такими простими правилами:

- завжди множити більше на менше;
- множення замінюється додаванням копій множини, записаних один під одним зі зрушенням щоразу на одну позицію вліво для кожного одиничного розряду множника (для нульових розрядів множника копія множника в шуканій сумі пропускається).

Розглянемо приклад, виконаємо множення чисел $1101_2 \cdot 101_2$:

$$\begin{array}{r}
 \times 1101 \\
 \quad 101 \\
 \hline
 + 1101 \\
 + 0000 \\
 + 1101 \\
 \hline
 1000001
 \end{array}$$

Ділення багаторозрядних двійкових чисел виконується аналогічно поділу на стовпчик

десятичних чисел. При цьому на кожному кроці поділу виконується послідовне віднімання дільника з чергового поділеного до отримання залишку, що дорівнює 0 або 1, а під лічену кількість віднімань записується як чергове значення частки.

Розглянемо приклад, виконаємо ділення чисел $1000110_2 : 111_2$:

$$\begin{array}{r|l} 1000110 & 111 \\ - 111 & 1010 \\ \hline 00111 & \\ - 111 & \\ \hline 00 & \end{array}$$

Арифметичні операції інших системах числення виконуються аналогічно.

До основних логічних операцій над двійковими числами відносяться операції інверсія (NOT), логічне "І" (AND), логічне "АБО" (OR) та "Виключне АБО" (XOR).

У логіці *логічними операціями* називають дії, внаслідок яких породжуються нові поняття з використанням вже існуючих. У більш вузькому значенні поняття логічної операції використовується в математичній логіці та програмуванні.

До операцій належать:

- логічне додавання (OR);
- логічне множення (AND);
- логічне віднімання (XOR);
- інверсія (NOT).

Інверсія (заперечення, доповнення, операція "НЕ", NOT) здійснює зміну значення операнду на протилежне. У тому випадку, якщо йдеться про інверсію двійкового числа з кількох розрядів, ця операція здійснюється порозрядно (побітове), тобто інвертується кожен розряд (біт) числа.

А	не А	not A
0	1	\overline{A}
1	0	$\neg A$

Наприклад, виконаємо інверсію для числа 01110010_2 :

$$(NOT) 01110010_2 = 10001101_2$$

Логічне множення (операція «І», AND) є бінарною операцією, що повертає 1, тільки коли обидва операнду дорівнюють 1. При роботі з двійковими числами з кількох розрядів, ця операція здійснюється порозрядно. Порозрядні операції зазвичай здійснюються з операндами рівної довжини. У разі різної довжини операндів той, який має меншу довжину, доповнюється нулями у старших ступенях.

А	В	А и В	A and B
0	0	0	A&B
0	1	0	$A \wedge B$
1	0	0	$A \cdot B$
1	1	1	AB

Розглянемо операцію порозрядного логічного множення з прикладу двох двійкових чисел а та b: $a \text{ and } b = 101101_2 \text{ and } 100110_2$:

$$\begin{array}{r} \text{and} \quad 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ \quad \quad 1 \ 0 \ 0 \ 1 \ 1 \ 0 \\ \hline \quad \quad 1 \ 0 \ 0 \ 1 \ 0 \ 0 \end{array}$$

Логічне додавання (операція «АБО», OR) є бінарною операцією, що повертає 1, коли хоча б один операнд дорівнює 1. При роботі з двійковими числами з кількох розрядів, ця операція здійснюється порозрядно.

A	B	A или B
0	0	0
0	1	1
1	0	1
1	1	1

A or B
 $A \vee B$
 $A+B$

Розглянемо операцію порозрядного логічного складання з прикладу двох двійкових чисел a та b: $a \text{ or } b = 101101_2 \text{ or } 100110_2$:

$$\begin{array}{r} \text{or} \quad 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ \quad \quad 1 \ 0 \ 0 \ 1 \ 1 \ 0 \\ \hline \quad \quad 1 \ 0 \ 1 \ 1 \ 1 \ 1 \end{array}$$

Операція «Виключне АБО» (XOR) є бінарною операцією, що повертає 1, коли один операнд дорівнює 0, а другий 1 і повертає 0 при рівності операндів. Працюючи з двійковими числами з кількох розрядів, ця операція здійснюється порозрядно.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

A xor B
 $A \oplus B$

Розглянемо операцію виключне АБО на прикладі двох двійкових чисел a та b: $a \text{ xor } b = 101101_2 \text{ xor } 100110_2$

$$\begin{array}{r} \text{xor} \quad 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ \quad \quad 1 \ 0 \ 0 \ 1 \ 1 \ 0 \\ \hline \quad \quad 0 \ 0 \ 1 \ 0 \ 1 \ 1 \end{array}$$

Розрядна сітка містить число, яке має n-розрядну цілу частину, k-розрядну дробову частину і знаковий розряд зліва від основних розрядів (рисунок 2.1).

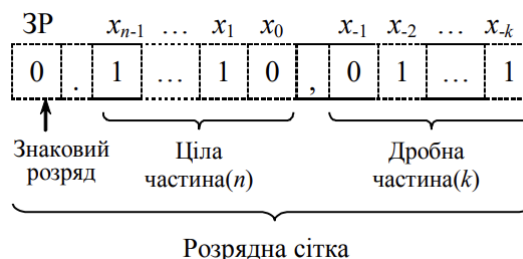


Рисунок 2.1 – Вигляд розрядної сітки

Існують два різновиди машинних зсувів:

- логічний зсув;
- арифметичний зсув.

Логічний зсув це зміщення розрядів машинного слова у просторі із втратою розрядів, що виходять за межі розрядної сітки. Розряди, що звільняються заповнюються нулями.

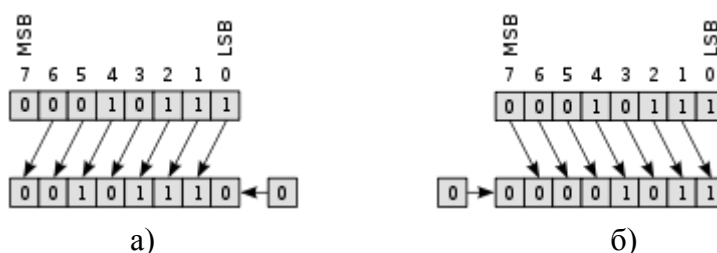
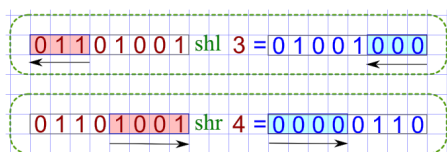


Рисунок 2.2 – Види логічних зсувів: а) логічний зсув вліво (SHL); б) логічний зсув вправо (SHR)

Наприклад, число 6, що зберігається у вигляді 32-бітного числа 00000000 00000000 00000000 00000110. Зсунемо це число вліво на один біт:

00000000 00000000 00000000 00001100

Наприклад, треба виконати логічний зсув 105 на 3 біта вліво та 105 на 4 біта вправо. Число 105 в двійковому поданні має вигляд 0110 1001.



Арифметичний зсув виконується з урахуванням знакового розряду (рисунок 2.3).

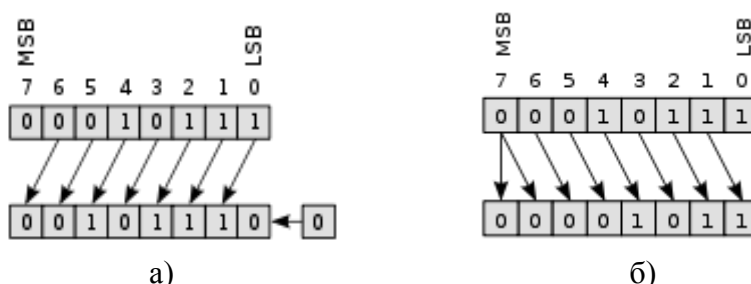


Рисунок 2.3 – Види логічних зсувів: а) арифметичний зсув вліво (SAL); б) арифметичний зсув вправо (SAR)

Арифметичний зсув ліворуч означає множення числа на 2 у ступені кількості біт (тобто на

основу системи числення), а зсув праворуч – ділення числа на 2 у ступені кількості біт. Таким чином, $6 \ll 1$ еквівалентно $6 * 2$, $6 \ll 3$ еквівалентно $6 * 8$.

Розглянемо приклад арифметичний зсув вправо на 1 біт числа $11011,1001_2$:

$11011,1001_2 \text{ sar } 1$

Результат: $11101,1100_2$

Циклічні зсуви (ROL і ROR) зазвичай не мають усталеного позначення у мовах високого рівня; вони заповнюють звільнені розряди значеннями, витісненими в інший бік (рисунок 2.4).

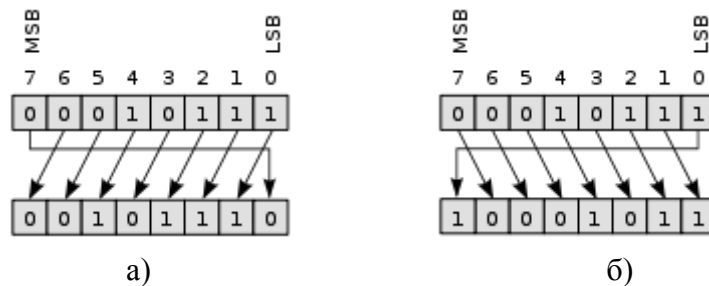


Рисунок 2.4 – Види логічних зсувів: а) циклічний зсув вліво (ROL); б) циклічний зсув вправо (ROR)

Приклад. Нехай маємо число 11111010_2 . Потрібно зробити циклічний зсув вліво на 1 біт цього числа, то отримаємо число

11110101_2

Якщо зробити циклічний зсув вправо на 1 біт, то отримаємо число

01111101_2

2. Завдання до лабораторної роботи №2

1. Виконайте додавання чисел.

- $1110101010_2 + 10111001_2$;
- $10111010_2 + 10010100_2$;
- $111101110,1011_2 + 1111011110,1_2$;
- $10111111_2 + 110010000_2$;
- $110010100_2 + 1011100001_2$.

2. Виконайте віднімання чисел.

- $1000000100_2 - 101010001_2$;
- $1010111101_2 - 111000010_2$;
- $1101000000,01_2 - 1001011010,011_2$;
- $1000001001_2 - 111110100_2$;
- $1111000101_2 - 1100110101_2$.

3. Виконайте множення чисел.

- $1001011_2 * 1010_2$;
- $111101_2 * 111_2$;
- $1001001_2 * 1001_2$;
- $1001000_2 * 1011_2$;
- $10111010_2 * 1111_2$.

4. Виконайте ділення чисел.

- $10000000_2 / 100_2$;

- $1111001_2 / 1011_2$;
- $1100011_2 / 11_2$;
- $1101010110_2 / 111_2$;
- $1101001_2 / 101_2$.

5. Виконайте наступні логічні операції.

- 1001001_2 and 1011_2 ;
- 1111001_2 or 1100011_2 ;
- 1000000100_2 xor 1010111101_2 .

6. Виконайте логічний зсув вправо та вліво, арифметичний зсув вправо та вліво та циклічний зсув вправо та вліво на один біт:

- 1001011_2 ;
- 1010011_2 ;
- 1100110101_2 ;

Лабораторна робота № 3. Створення блок-схем

Мета роботи: ознайомитись із блок-схемами. Навчитися записувати лінійні, що розгалужуються та циклічні алгоритми у вигляді блок-схеми.

1. Теоретичний матеріал

Блок-схемою називається наочне зображення алгоритму, коли окремі дії (етапи алгоритму) зображуються за допомогою різних геометричних фігур (блоків), а зв'язки між етапами (послідовність виконання етапів) зазначаються за допомогою стрілок, що з'єднують ці фігури.

Виконання блок-схем здійснюється за ГОСТ 19701-90.

При виконанні блок-схем усередині кожного блоку вказується інформація, що характеризує дії, що виконуються цим блоком.

Потоки даних у схемах відображаються лініями. Напрямок потоку зліва направо та зверху вниз вважається стандартним. Якщо потік має напрямок відмінний від стандартного, на лініях використовуються стрілки, що вказують цей напрямок.

У схемах слід уникати перетину ліній. Лінії, що перетинаються, не мають логічного зв'язку між собою, тому зміни напрямку в точках перетину не допускаються.


Якщо дві або більше вхідних ліній об'єднуються в одну вихідну лінію, місце об'єднання ліній зміщується.

Кількість вхідних ліній не обмежена, лінія, що виходить з блоку, повинна бути одна, за винятком логічного блоку.

Розглянемо основні елементи блок-схем:

–  Блок початок-кінець (пуск-зупинка)

Елемент відображає вхід із зовнішнього середовища або вихід з нього (найчастіше застосування – початок та кінець програми). Усередині фігури записується відповідна дія.

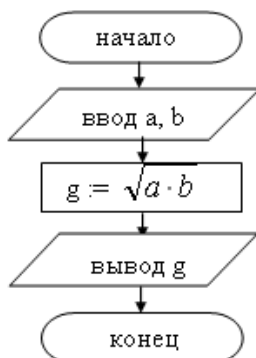
–  Дані (введення-виведення)

Перетворення даних на форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Всередині фігури безпосередньо записують дані для введення або виведення.

–  Блок дії

Виконання однієї чи кількох операцій, обробка даних будь-якого виду. Усередині фігури безпосередньо записують самі операції, наприклад, операцію присвоєння: $a = 10 \cdot b + c$

Розглянемо приклад блок-схеми алгоритму обчислення середнього геометричного між двома числами:



Базові структури алгоритмів – це певний набір блоків та стандартних способів їхнього

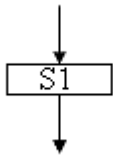
з'єднання для виконання типових послідовностей дій.

До основних структур належать такі:

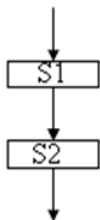
- лінійні;
- що розгалужуються;
- циклічні.

Лінійними називаються алгоритми, у яких дії здійснюються послідовно один за одним.

Блок має один вхід та один вихід:



Команда слідування складається лише з найпростіших команд S1 і S2. З команд слідування утворюються лінійні алгоритми.



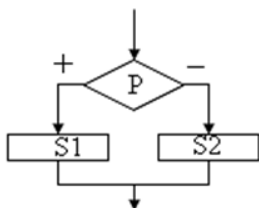
Що розгалужується називають алгоритм, в якому дія виконується по одній з можливих гілок рішення задачі, залежно від виконання умов.

На відміну від лінійних алгоритмів, в яких команди виконуються послідовно одна за одною, в алгоритмах, що розгалужуються, входить умова, залежно від виконання або невиконання якої виконується та чи інша послідовність команд (дій).

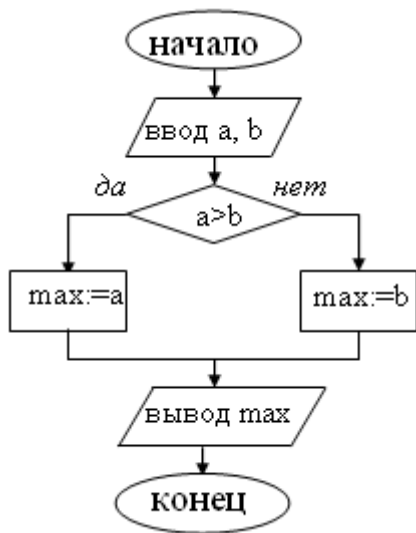
Як умова в алгоритмі, що розгалужується, може бути використане будь-яке зрозуміле виконавцю твердження, яке може бути істинним або бути хибним. Таке твердження може бути виражене як словами, і формулою.

Команда розгалуження – це складова команда алгоритму, у якій залежно від умови P виконується або S1 або S2.

З команд прямування і команд розгалуження складаються алгоритми, що розгалужуються.



Розглянемо приклад знаходження більшого із двох чисел:



Циклічним називається алгоритм, у якому деяка частина операцій (тіло циклу – послідовність команд) виконується багаторазово.

Організація циклів, будь-коли призводить до зупинки у виконанні алгоритму, порушення вимоги його результативності, тобто. отримання результату за кінцеве число кроків.

Ітераційним називається цикл, число повторень якого задається, а визначається під час виконання циклу. І тут одне повторення циклу називається ітерацією.

Рекурсія – це така ситуація, коли деякий алгоритм безпосередньо або через інші алгоритми викликає себе як допоміжний. Сам алгоритм називається рекурсивним.

Розрізняють (рисунок 3.1):

- цикл із передумовою (цикл типу поки що)
- цикл із постумовою (цикл типу до)

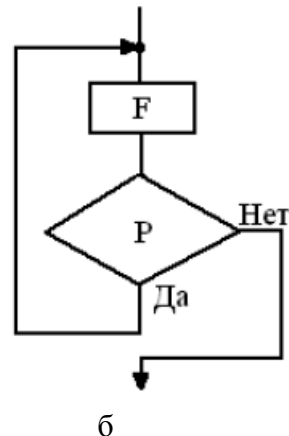
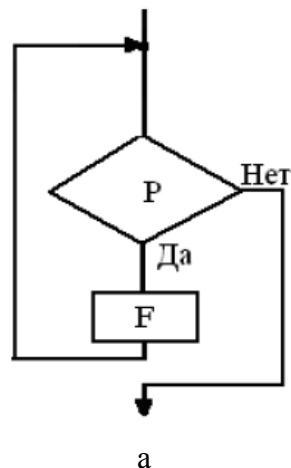
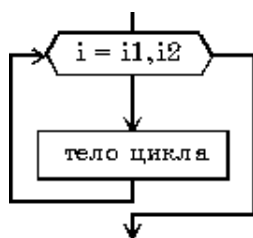
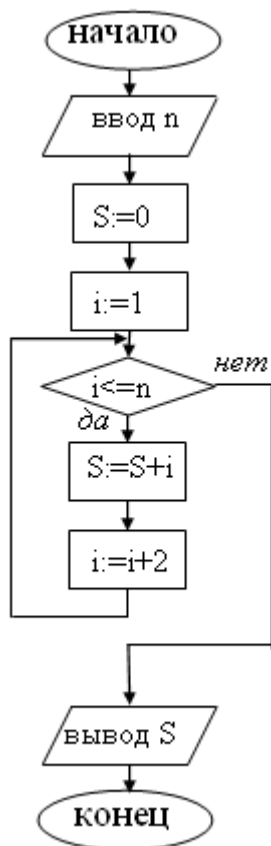


Рисунок 3.1 – Цикли: з передумовою (а); з постумовою (б)

Базова структура циклу забезпечує багаторазове виконання деякої сукупності дій, яка називається тілом циклу, наприклад, цикл для i від i_1 до i_2 крок i_3 .



Розглянемо приклад обчислення суми перших n непарних чисел:



2. Завдання до лабораторної роботи №3

1. Лінійний алгоритм:

- Обчислити площу поверхні і об'єм усіченого конуса за такими формулами:

$$S = \pi (R + r) l + \pi R^2 + \pi r^2 ;$$

$$V = (1/3) \pi (R^2 + r^2 + Rr) h .$$

- Обчислити медіана трикутника зі сторонами a , b , c по формулам:

$$m_a = 0.5\sqrt{2b^2 + 2c^2 - a^2} ;$$

$$m_b = 0.5\sqrt{2a^2 + 2c^2 - b^2} ;$$

$$m_c = 0.5\sqrt{2b^2 + 2a^2 - c^2} ;$$

- Обчислити площу кола і довжину окружності по введеному значенню радіуса.

2. Алгоритм, що розгалужується:

- Визначити, чи можна з відрізків з довжинами x , y і z побудувати трикутник.
- Якщо серед трьох чисел a , b , c є хоча б одне парне, то знайти максимальне число, інакше – мінімальне.
- Визначити, в якому квадранті знаходиться точка з координатами x і y і вивести номер квадранта на екран.

3. Циклічний алгоритм:

- Знайти суму чисел, кратних трьом, в діапазоні від 0 до 50.
- Вводяться позитивні числа. Припинити введення, коли сума введених чисел перевищить 100.
- Ви поклали в банк 1500 гривень. Визначити, скільки грошей буде на Вашому вкладі через 1 рік, якщо кожен місяць внесок збільшується на 0.76% від суми попереднього місяця.

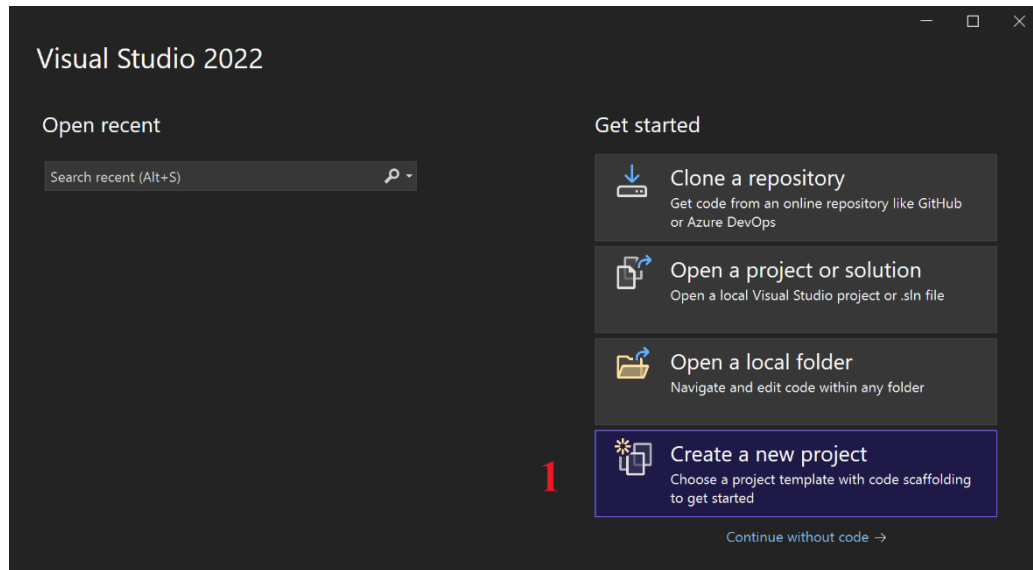
Лабораторна робота № 4. Типи даних. Розрахунок математичних виразів

Мета роботи: ознайомитися з типами даних мови програмування C#, арифметичними операціями та бібліотекою математичних функцій. Навчитися створювати змінні необхідного типу та математичні вирази для обчислення.

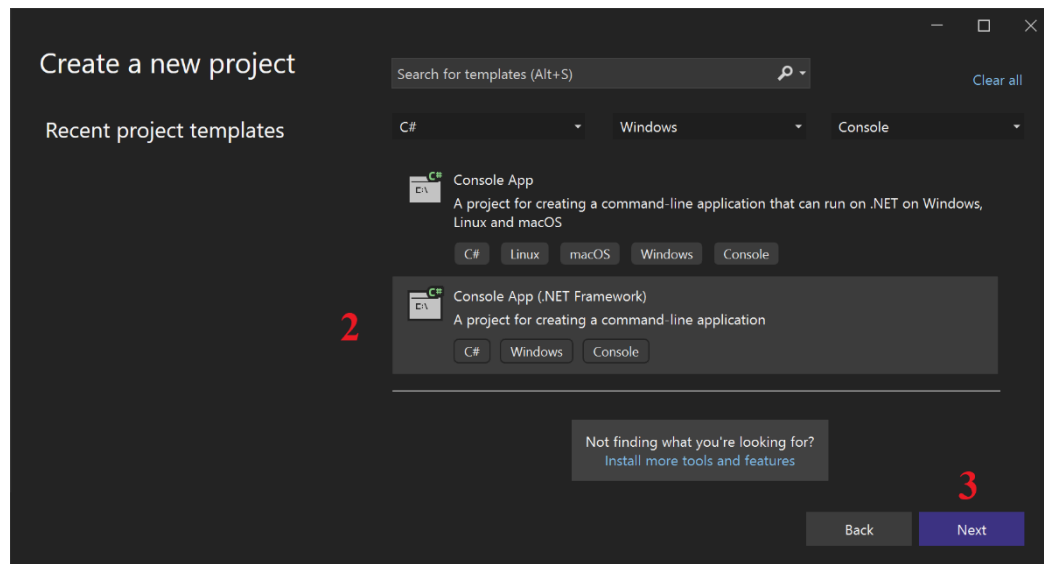
1. Теоретичний матеріал

Розглянемо основні кроки для створення консольного додатку за допомогою Visual Studio.

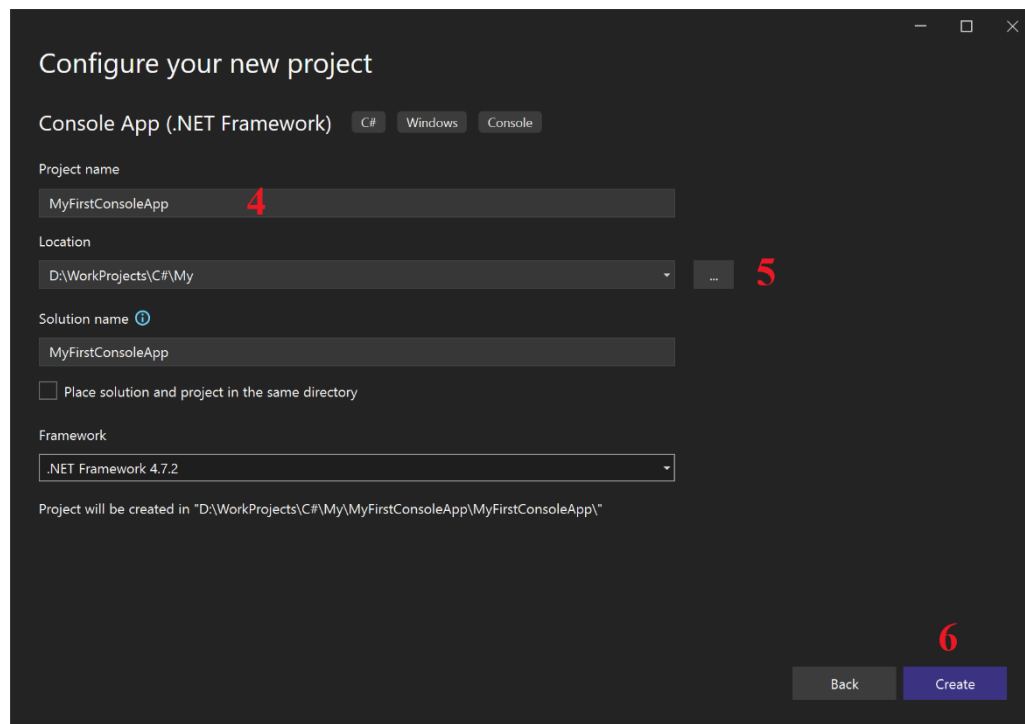
1. Запускаємо Visual Studio та натискаємо кнопку Create a new project



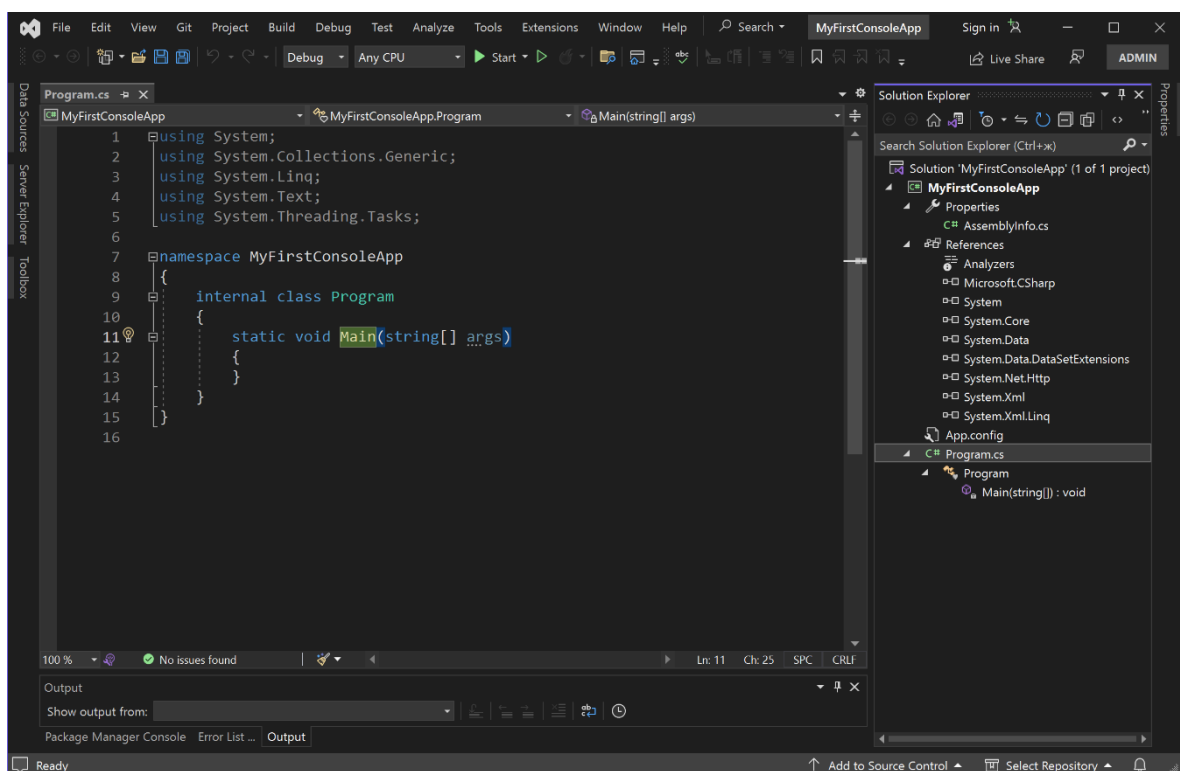
2. Обираємо тип проекту, що створюється.



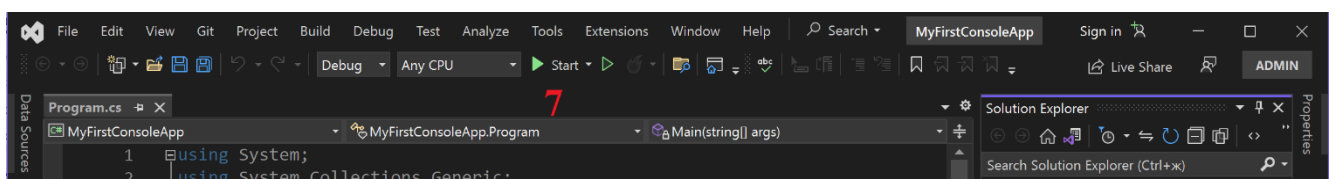
3. Необхідно ввести ім'я проекту



4. Відкривається середовище для програмування та створюється консольний додаток



5. Для запуску програми на виконання натискаємо кнопку Start



У кожному проекті проекту C# є файл, який відповідає за загальну конфігурацію проекту. За замовчанням цей файл називається Назва_проекту.csproj.

Для зберігання даних у програмі використовуються змінні. Змінна являє саме іменовану область пам'яті, в якій зберігається значення певного типу. Змінна має тип, ім'я та значення. Тип визначає, якого роду інформацію може зберігати змінна.

Перед використанням будь-яку змінну слід визначити. Синтаксис визначення змінної виглядає так:

тип ім'я_змінної;

Спочатку йде тип змінної, потім її ім'я.

Як ім'я змінної може виступати будь-яка довільна назва, яка задовольняє наступним вимогам:

- ім'я може містити будь-які цифри, букви та символ підкреслення, при цьому перший символ в імені має бути буквою або символом підкреслення;
- в імені не повинно бути знаків пунктуації та пропусків;
- ім'я не може бути ключовим словом мови C#. Таких слів не так багато, і під час роботи у Visual Studio середовище розробки підсвічує ключові слова синім кольором.

Хоча ім'я змінною може бути будь-яким, але слід давати змінним описові імена, які будуть говорити про їхнє призначення.

Оскільки визначення змінної є інструкцією, то після нього ставиться крапка з комою.

При цьому слід враховувати, що C# є регістрозалежною мовою, тому два однакові визначення змінних, але записані різними малими або заголовними літерами будуть представляти дві різні змінні. Наприклад, name та Name – це різні імена змінних.

Після визначення змінної можна надати деяке значення:

ім'я_змінної = значення;

Причому змінної можна присвоїти лише те значення, яке відповідає її типу.

Надалі за допомогою імені змінної ми зможемо звертатися до тієї області пам'яті, де зберігається її значення.

Також ми зможемо відразу при визначенні присвоїти змінної значення. Цей прийом називається *ініціалізацією*:

тип ім'я_змінної = значення;

Як і в багатьох мовах програмування, C# є своя система типів даних, яка використовується для створення змінних. *Тип даних* визначає внутрішнє представлення даних, безліч значень, які може приймати об'єкт, і навіть допустимі дії, які можна застосовувати над об'єктом.

У мові C# є такі базові типи даних:

- bool: зберігає значення true чи false (логічні літерали) **та займає** .
- byte: зберігає ціле число від 0 до 255 та займає 1 байт.
- sbyte: зберігає ціле число від -128 до 127 та займає 1 байт.
- short: зберігає ціле число від -32768 до 32767 і займає 2 байти.
- ushort: зберігає ціле число від 0 до 65535 і займає 2 байти.
- int: зберігає ціле число від -2147483648 до 2147483647 і займає 4 байти. Всі цілі **цілочисельні?** літерали за умовчанням представляють значення типу int
- uint: зберігає ціле число від 0 до 4294967295 і займає 4 байти.

- long: зберігає ціле число від -9223372036854775808 до 9223372036854775807 і займає 8 байт.
- ulong: зберігає ціле число від 0 до 18446744073709551615 і займає 8 байт.
- float: зберігає число з плаваючою точкою від $-3.4 \cdot 10^{38}$ до $3.4 \cdot 10^{38}$ і займає 4 байти.
- double: зберігає число з плаваючою точкою від $\pm 5.0 \cdot 10^{-324}$ до $\pm 1.7 \cdot 10^{308}$ та займає 8 байти.
- decimal: зберігає десяткове дробове число. Якщо використовується без десяткової коми, має значення від $\pm 1.0 \cdot 10^{-28}$ до $\pm 7.9228 \cdot 10^{28}$, може зберігати 28 знаків після коми і займає 16 байт.
- char: зберігає одиночний символ у кодуванні Unicode і займає 2 байти.
- string: зберігає набір символів Unicode і займає.

Літерали становлять незмінні значення. Літерали можна передавати змінним як значення. Літерали бувають логічними, цілими, речовими, символьними і малими. І окремих літералів є ключовим словом null.

Цілочисленні літерали представляють позитивні та негативні цілі числа, наприклад, 1, 2, 3, 4, -7, -109. Числа в двійковій формі передуються символами 0b, після яких йде набір з нулів та одиниць. Для запису числа у шістнадцятковій формі застосовуються символи 0x, після яких йде набір символів від 0 до 9 і від A до F, які власне представляють число.

Наприклад:

```
int a = 0xFF; // 255
int b = -101;
int c = 0b11; // 3
```

Всі цілі літерали за замовчанням розглядаються як значення типу int. Щоб явним чином вказати, що цілий літерал представляє значення типу uint, в коді слід треба використовувати суфікс U/u, для типу long – суфікс L/l, а для типу ulong – суфікс UL/ul. Наприклад:

```
uint a = 10U;
long b = 20L;
ulong c = 30UL;
int d = 150;
```

Дійсні літерали є дробовими числами. Цей тип літералів має дві форми. Перша форма – дійсні числа з фіксованою комою, при якій дробову частину відокремлюється від цілої частини крапкою. Також дійсні літерали можуть визначатися в експоненційній формі ME_p, де M – мантиса, E – експонента, яка фактично означає " $\cdot 10^p$ " (помножити на десять у ступені), а p – порядок.

Приклади дійсних літералів:

```
double c = 100.001;
double d = -0.38;
double e = 3.2e3; // 3.2 * 10^3 = 3200
```

При присвоєнні значень слід пам'ятати таку тонкість: всі речові літерали (дрібні числа) розглядаються як значення типу double. І щоб вказати, що дрібне число представляє тип float або тип decimal, необхідно до літералу додавати суфікс: F/f – для float і M/m – для decimal.

```
float b = 30.6f;
double a = 10.003;
decimal c = 1005.8M;
```

Символьні літерали є одиночними символами.

Наприклад:

```
char f = 'W';
char g = '1';
```

Рядкові літерали представляють рядки **символів**. Рядки **полягають** у подвійні лапки.

Якщо всередині рядка необхідно вивести подвійну лапку, то така внутрішня лапка попереджається зворотним слішем. Також у рядках можна використовувати керуючі послідовності.

Наприклад:

```
string m = "hello word";
string n = "Компания \"Рога и копыта\"";
string p = "Hello \nWorld!!!";
```

C# надає низку операторів. Багато хто з них підтримується вбудованими типами і дозволяє виконувати базові операції зі значеннями цих типів. До цих операторів входять такі групи:

- Арифметичні оператори виконують арифметичні операції з числовими операндами.
- Оператори порівняння, що порівнюють числові операнди.
- Логічні оператори виконують логічні операції з операндами bool.
- Бітові оператори та оператори зсуву виконують бітові операції або операції зсуву з операндами цілих чисел.
- Оператори рівності перевіряють **рівність чи нерівність** своїх операндів.

Наприклад:

```
int x = 10;
int z = x + 12; // 22
int y = x - 6; // 4
int m = x * 5; // 50
double b = 3;
double c = x / b; //3.33333333
```

При діленні варто враховувати, що **оскільки** й обидва операнди представляють цілі числа, то результат також округляється до цілого числа.

```
double z = 10 / 4; //результат равен 2
```

Незважаючи на те, що результат операції в результаті поміщається в змінну типу double, яка дозволяє зберегти дробову частину, але в самій операції беруть участь два літерали, які за умовчанням розглядаються як об'єкти int, тобто цілі числа, і результат те ж буде цілий.

Для виходу з цієї ситуації необхідно визначати літерали або змінні, що беруть участь в операції саме як типи double або float:

```
double z = 10.0 / 4.0; //результат равен 2.5
```

Операція % отримання залишку від цілісного поділу двох чисел. Наприклад:

```
double x = 10.0;
double y = x % 4; //результат равен 2
```

Операція інкременту.

Інкремент буває префіксним: ++x - спочатку значення змінної x збільшується на 1, а потім її значення повертається як результат операції.

І також існує постфіксний інкремент: x++ - спочатку значення змінної x повертається як результат операції, а потім до нього додається 1.

Наприклад:

```
int x1 = 5;
```

```
int z1 = ++x1; // z1=6; x1=6
int x2 = 5;
int z2 = x2++; // z2=5; x2=6
```

Операція декременту чи зменшення значення на одиницю. Також існує префіксна форма декременту (--x) та постфіксна (x--).

Наприклад:

```
int x1 = 5;
int z1 = --x1; // z1=4; x1=4
int x2 = 5;
int z2 = x2--; // z2=5; x2=4
```

При виконанні кількох арифметичних операцій слід враховувати порядок виконання. Пріоритет операцій від найвищого до нижчого:

- Інкремент, декремент;
- **Ступінь?** Множення, розподіл, отримання залишку;
- Додавання, віднімання.

Для зміни порядку проходження операцій застосовуються дужки.

Крім базової операції присвоєння в C# є ще ряд комбінованих операцій:

- += присвоєння після додавання. Надає лівому операнду суму лівого та правого операндів: вираз $A += B$ рівнозначний виразу $A = A + B$
- -= присвоєння після віднімання. Надає лівому операнду різниця лівого і правого операнду: $A -= B$ еквівалентно $A = A - B$
- *= присвоєння після множення. Надає лівому операнду добуток лівого та правого операндів: $A *= B$ еквівалентно $A = A * B$
- /= присвоєння після розподілу. Надає лівому операнду приватне лівого та правого операндів: $A /= B$ еквівалентно $A = A / B$
- %= присвоєння після поділу по модулю. Надає лівому операнду залишок від цілого розподілу лівого операнду на правий: $A \% = B$ еквівалентно $A = A \% B$

Наприклад:

```
int a = 10;
a += 10; // 20
a -= 4; // 16
a *= 2; // 32
a /= 8; // 4
```

Для виконання різних математичних операцій у бібліотеці класів .NET призначено клас Math (using System).

Розглянемо основні математичні функції.

Math.Abs	Повертаємо абсолютне число, має 7 навантажень. Тобто метод приймає різні типи змінних.	int i = Math.Abs(x);
Math.Acos	Арккосинус. Визначається кут, косинус якого дорівнює зазначеній кількості.	double i = Math.Acos(0.5);
Math.Asin	Арксинус. Визначається кут, синус якого дорівнює зазначеній кількості	double i = Math.Asin(0.5);
Math.Atan	Арктангенс.	double i = Math.Atan(0.5);
Math.Cos	Повертає косинус кута.	double x = Math.Cos(1.04);
Math.Cosh	Повертає гіперболічний косинус кута.	double x = Math.Cosh(radian);
Math.Exp	Експонента.	double x = Math.Exp(2);
Math.Log	Обчислення логарифму. X – число яке	double x = Math.Log(X,Osn);

	потрібно знайти, Osn – основа логарифму.	
<code>Math.Log10</code>	Обчислення десяткового логарифму.	<code>double x = Math.Log10(10)</code>
<code>Math.Max</code>	Повертає з 2-х чисел більше.	<code>int x = Math.Max(10,20);</code>
<code>Math.Min</code>	Повертає з 2-х чисел менше.	<code>int x = Math.Min(10,20);</code>
<code>Math.PI</code>	Повертає число Пі.	<code>double pi = Math.PI;</code>
<code>Math.Pow</code>	Повертає число зведений у ступінь: a^x	<code>double i = Math.Pow(a, x);</code>
<code>Math.Sin</code>	Повертає синус кута.	<code>double p = Math.Sin(0.5);</code>
<code>Math.Sinh</code>	Повертає гіперболічний синус кута.	<code>double p = Math.Sinh(0.5);</code>
<code>Math.Sqrt</code>	Повертає квадратний корінь.	<code>double r = Math.Sqrt(7);</code>
<code>Math.Tan</code>	Повертає тангенс кута.	<code>double p = Math.Tan(1.04);</code>
<code>Math.Tanh</code>	Повертає гіперболічний тангенс кута.	<code>double p = Math.Tanh(1.04);</code>

Розглянемо функції округлення.

`Ceiling(double value)` – повертає найменше ціле число з плаваючою точкою, яке не менше за value.

```
double result = Math.Ceiling(2.34); // 3
```

`Floor(decimal d)` - повертає найбільше ціле число, яке не більше d.

```
double result = Math.Floor(2.56); // 2
```

`Round(double d)` – повертає число d, округлене до найближчого цілого числа.

```
double result1 = Math.Round(20.56); // 21
double result2 = Math.Round(20.46); // 20
```

`Round(double a, int b)`: повертає число a, округлене до певної кількості знаків після коми, представленої параметром b.

```
double result1 = Math.Round(20.567, 2); // 20,57
double result2 = Math.Round(20.463, 1); // 20,5
```

`Truncate(double value)` - відкидає дрібну частину числа value, повертаючи лише ціле значення.

```
double result = Math.Truncate(16.89); // 16
```

`DivRem(int a, int b, out int result)` – повертає результат від поділу a/b, а залишок поміщається у параметр result.

```
int result;
int div = Math.DivRem(14, 5, out result);
//result = 4
// div = 2
```

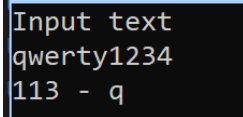
Консоль — це вікно операційної системи, в якому користувачі взаємодіють із операційною системою або текстовою консольною програмою, **вводячи введення** тексту за допомогою клавіатури комп'ютера та зчитуючи текстові вихідні дані з терміналу комп'ютера.

Клас Console використовується в консольних проектах, дозволяючи вводити вихідні дані з консолі та виводити результати на консоль. За замовчуванням, при введенні з консолі дані вводяться з клавіатури і відображаються на дисплеї, при виведенні на консоль – дані відображаються на екрані дисплея.

Методи `Read()` та `ReadLine()` дозволяють читати з консолі текст, який відображається на екрані комп'ютера. Методи не мають вхідних аргументів.

Оператор `Read()` читає по одному символу із вхідного рядка і повертає як результат код прочитаного символу, що має тип `int`. Посимвольне введення застосовується досить **рідко**.

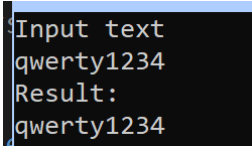
```
Console.WriteLine("Input text");
int code = Console.Read();
char ch = Convert.ToChar(code);
Console.WriteLine("'" + code + "' - " + ch);
```



```
Input text
qwerty1234
113 - q
```

Основним методом, який використовується для читання даних з консолі, є метод `ReadLine()`. Він читає з консолі рядок тексту, що завершується ознакою кінця рядка. Цей рядок є результатом, що повертається методом `ReadLine()`.

```
Console.WriteLine("Input text");
string str = Console.ReadLine();
Console.WriteLine("Result: ");
Console.WriteLine(str);
```



```
Input text
qwerty1234
Result:
qwerty1234
```

ReadKey() отримує наступний натиснутий користувачем символ або функціональну клавішу. Натиснута клавіша відображається у вікні консолі.

Одним з найбільш поширених способів використання методу `ReadKey()` є зупинка виконання програми доти, доки користувач не натисне клавішу і програма не завершить роботу або не відобразить додаткове вікно з інформацією.

Метод `ReadLine()` повертає рядок тексту навіть коли вводяться числа. Щоб у подальшому використовувати введений рядок як число його потрібно конвертувати у число.

Клас `Convert` представляє спосіб перетворення значень. Для цього в ньому визначено такі статичні методи:

- `ToBoolean(value)`
- `ToByte(value)`
- `ToChar(value)`
- `ToDateTime(value)`
- `ToDecimal(value)`
- `ToDouble(value)`
- `ToInt16(value)`
- `ToInt32(value)`
- `ToInt64(value)`
- `ToSByte(value)`
- `ToSingle(value)`
- `ToUInt16(value)`
- `ToUInt32(value)`
- `ToUInt64(value)`

Як параметр у ці методи може передаватися значення різних примітивних типів, необов'язково рядки.

Наприклад:

```
Console.WriteLine("Input a number: ");
string str = Console.ReadLine();
int number = Convert.ToInt32(str);
number *= 10;
Console.WriteLine(number);
```

```
Input a number:
7
70
```

Цей запис можна записати компактніше:

```
Console.WriteLine("Input a number: ");
int number = Convert.ToInt32(Console.ReadLine());
number *= 10;
Console.WriteLine(number);
```

Якщо методу з класу `Convert` не вдасться перетворити значення до потрібного типу, він **викидає виняток** `FormatException`. Наприклад, якщо при запуску попереднього коду замість числа ввести строку, то програма **викидає виняток** `FormatException`.

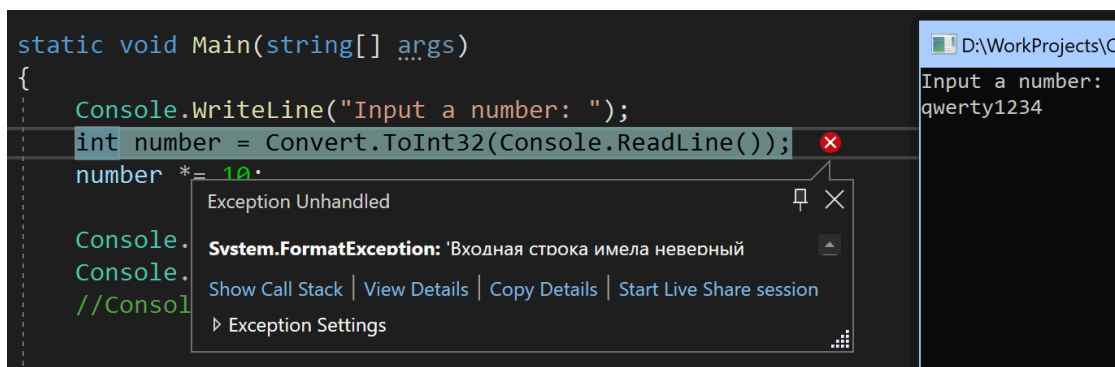


Рисунок 4.1 – Виняток `FormatException`

Для уникнення **вильоту програми на помилці переважно** застосування методу `TryParse()`. Він намагається перетворити рядок до потрібного типу і якщо перетворення пройшло успішно, то повертає `true`, інакше повертається `false`.

Перепишемо попередній код з використанням `TryParse()`.

```
Console.WriteLine("Input a number: ");
bool result = int.TryParse(Console.ReadLine(), out var number);
if (result == true)
    Console.WriteLine(number * 10);
else
    Console.WriteLine("Wrong input!");
```

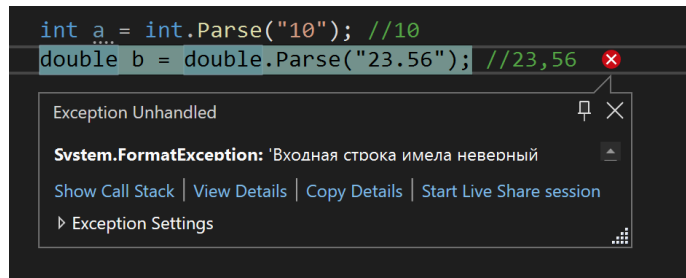
```
Input a number:
qwerty1234
Wrong input!
```

Існує ще один метод конвертації даних. Метод `Parse()` **як параметр приймає рядок** та повертає об'єкт поточного типу.

Наприклад:

```
int a = int.Parse("10"); //10
double b = double.Parse("23,56"); //23,56
```

Розглянемо ситуацію коли ми замість коми у дійсному числі поставимо крапку.



Методи Parse та TryParse у числових типах враховують місцеві налаштування культури, що можна змінити, вказуючи об'єкт CultureInfo (System.Globalization). Часто вказівка інваріантної культури є вдалою ідеєю. Наприклад, число "1.234" у double дає 1234 для Німеччини. Причина в тому, що символ точки в Німеччині використовується як роздільник тисяч, а не як десяткова точка. Проблема усуває зазначення інваріантної культури:

```
double b = double.Parse("23.56", CultureInfo.InvariantCulture); //23,56
```

CultureInfo.InvariantCulture визначає екземпляр інваріантної мови та регіональних параметрів. Він пов'язаний з англійською мовою, але не з якоюсь країною чи регіоном.

CultureInfo.CurrentCulture відображає налаштування панелі керування комп'ютера під час виконання.

Можна встановлювати специфічну культуру. У наступному прикладі ми просимо специфічну культуру (англійська (english) у Великій Британії (Great Britain)):

```
CultureInfo uk = CultureInfo.GetCultureInfo("en-GB");
double b = double.Parse("23.56", uk); //23,56
```

Щоб не залежати від регіональних культурних відмінностей, ми можемо встановити чіткий формат за допомогою класу NumberFormatInfo та його властивості NumberDecimalSeparator.

```
IFormatProvider formatter = new NumberFormatInfo { NumberDecimalSeparator = "." };
double b = double.Parse("23.56", formatter); //23,56
```

Для виведення інформації на екран використовуються методи Write та WriteLine. Метод Write дуже нагадує WriteLine, але відмінність у цьому, що WriteLine завершує свій вивід поверненням каретки.

Методи Write та WriteLine виводять у консоль рядок, який передається як параметр:

```
Console.WriteLine("Hello world!");
int x = 10; int y = -3;
Console.WriteLine("Point (" + x + "; " + y + ")");
```

```
Hello world!
Point (10; -3)
```

При виведенні рядків у консолі за допомогою методу Console.WriteLine для вбудовування значень у рядок ми можемо використовувати форматування замість конкатенації. Для цього використовується рядок складеного формату та список об'єктів.

Рядок складеного формату складається з блоків фіксованого тексту, числом від нуля і більше, що перемежуються одним або декількома елементами форматування.

Фіксованим текстом може бути довільний рядок, а кожен елемент форматування повинен відповідати об'єкту зі списку.

Під час складеного форматування створюється новий результуючий рядок, у якому всі елементи форматування замінені на рядкове подання відповідних об'єктів зі списку.

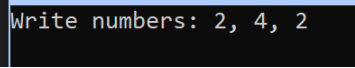
Кожен елемент форматування має такий вигляд і складається з наступних компонентів:

{index[,alignment][:formatString]}

Обов'язковий компонент `index`, що також називається описувачем параметра, - це число, що визначає відповідний об'єкт зі списку. Індексація елементів ведеться від нуля. Тобто елемент формату, описувач параметрів якого дорівнює 0, форматує перший об'єкт у списку. Елемент формату, описувач параметрів якого дорівнює 1, форматує другий об'єкт у списку тощо.

Наприклад:

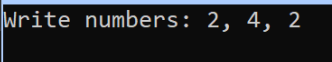
```
Console.WriteLine("Write numbers: {0}, {1}, {0}", 2, 4);
```



```
Write numbers: 2, 4, 2
```

Те саме форматування в рядку ми можемо зробити не тільки в методі `Console.WriteLine`, але і в будь-якому місці програми за допомогою методу `string.Format`:

```
string output = string.Format("Write numbers: {0}, {1}, {0}", 2, 4);
Console.WriteLine(output);
```



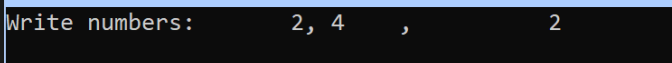
```
Write numbers: 2, 4, 2
```

Необов'язковий компонент `alignment` – це ціле число зі знаком, яке служить для вказівки бажаної ширини поля форматування. Якщо значення вирівнювання менше довжини форматowanego рядка, вирівнювання ігнорується, а довжина форматowanego рядка використовується як ширина поля.

Форматовані дані вирівнюються в полі правому краю, якщо `alignment` має позитивне значення, або по лівому краю, якщо `alignment` має негативне значення. За потреби відформатований рядок доповнюється пробілами. Якщо встановлено вирівнювання потрібна кома.

Наприклад:

```
Console.WriteLine("Write numbers: {0,7}, {1,-5}, {0,10}", 2, 4);
```



```
Write numbers:      2, 4      ,      2
```

Необов'язковий компонент `formatString` — це рядок формату, який відповідає типу відформатованого об'єкта.

Можна вказати:

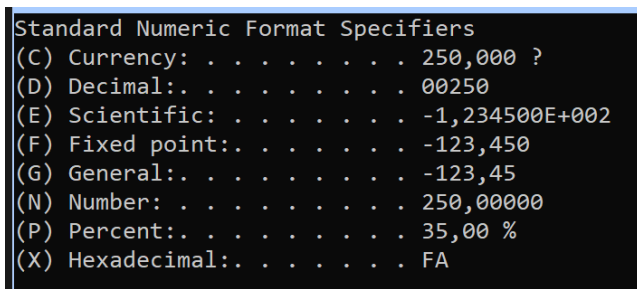
- Рядок стандартного чи налаштованого числового формату, якщо відповідний об'єкт є числовим значенням.
- Рядок стандартного або налаштованого формату дати та часу, якщо відповідний `DateTime` об'єкт є об'єктом.
- Рядок формату перерахування, якщо об'єкт є значенням перерахування.

Розглянемо компонент `formatString` для числового формату.

C / c	Задає формат грошової одиниці, вказує кількість десяткових розрядів після коми
D / d	Цілочисельний формат, вказує мінімальну кількість цифр
E / e	Експоненційне подання числа, що вказує кількість десяткових розрядів після коми
F / f	Формат дробових чисел із фіксованою точкою, вказує кількість десяткових розрядів після коми
G / g	Задає більш короткий із двох форматів: F або E
N / n	Також задає формат дійсних чисел із фіксованою точкою, визначає кількість розрядів після коми
P / p	Задає відображення знаку відсотків поряд з числом, вказує кількість десяткових розрядів після коми
X / x	Шістнадцятковий формат числа

Наприклад:

```
Console.WriteLine("Standard Numeric Format Specifiers");
Console.WriteLine(
    "(C) Currency: ..... {0:C3}\n" +
    "(D) Decimal:..... {0:D5}\n" +
    "(E) Scientific: ..... {1:E}\n" +
    "(F) Fixed point:..... {1:F3}\n" +
    "(G) General:..... {1:G}\n" +
    "(N) Number: ..... {0:N5}\n" +
    "(P) Percent:..... {2:P}\n" +
    "(X) Hexadecimal: ..... {0:X}\n",
    250, -123.45f, 0.35);
```



```
Standard Numeric Format Specifiers
(C) Currency: . . . . . 250,000 ?
(D) Decimal:.. . . . . 00250
(E) Scientific: . . . . . -1,234500E+002
(F) Fixed point:.. . . . . -123,450
(G) General:.. . . . . -123,45
(N) Number: . . . . . 250,00000
(P) Percent:.. . . . . 35,00 %
(X) Hexadecimal:.. . . . . FA
```

Щоб визначити спосіб форматування числових даних, можна створити рядок числового формату, що налаштовується, що складається з одного або декількох описників настроюваного формату.

Спеціальні форматні рядки – з їх допомогою контролюється кожен символ за допомогою шаблону.

Спеціальні форматні рядки для чисел:

- `"#"` – Заповнювач для цифр. Обмежує кількість цифр після десяткової точки;
- `"0"` – Заповнювач для нуля. Обмежує кількість цифр після десяткової точки, але також доповнює нулями до та після десяткових позицій;
- `"."` – Розділювач. Визначає розташування роздільника цілої та дробової частин у результуючому рядку;
- `","` – Розділювач груп. Як роздільник груп вставляє локалізований символ-розділювач груп між усіма групами;
- `"%"` – Відсотки. Помножує число на 100 і вставляє локалізований символ відсотка в результуючий рядок.

- "E0", "E+0", "E-0" – Експонентна нотація. Якщо за цим описувачем слід щонайменше один нуль (0), результат форматується за допомогою експоненційної нотації. Регістр ("E" або "e") визначає регістр символу експоненти в результуючому рядку. Мінімальна кількість цифр експоненти визначається кількістю нулів, що стоять за символом "E" або "e". Знак "+" вказує на те, що перед експонентою завжди повинен ставитися символ знаку. Знак "-" вказує на те, що символ знаку повинен ставитися лише у випадку, якщо експонент має негативне значення.
- "\" – Escape-символ. Вказує на те, що наступний за ним символ повинен розглядатися як літерал, а не як описувач формату, що настраюється.

Наприклад:

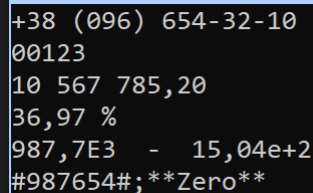
```
long number = 380966543210;
string result = string.Format("{0:## (###) ###-##-##}", number);
Console.WriteLine(result); // +38 (096) 654-32-10
```

```
int value = 123;
Console.WriteLine("{0:00000}", value); // 00123
```

```
double a = 10567785.2;
Console.WriteLine("{0:##,##.00}", a);
```

```
double x = 0.3697;
Console.WriteLine("{0:###.00 %}", x);
```

```
int y = 987654; double m = 1503.92311;
Console.WriteLine("{0:000.0E0} - {1:00.00e+0}", y, m);
Console.WriteLine("{0:\\#0\\#};**Zero**", y);
```



```
+38 (096) 654-32-10
00123
10 567 785,20
36,97 %
987,7E3 - 15,04e+2
#987654#; **Zero**
```

Інтерполяція рядків має спростити форматування рядків. Для запису використовується знак долара (\$) перед рядком.

Інтерполяція насправді представляє більш лаконічне форматування. При цьому всередині фігурних дужок ми можемо вказувати як властивості, а й різні висловлювання мови C#. Усередині рядка використовуються {...}, тільки всередині фігурних дужок можна прямо писати ті висловлювання, які ми хочемо вивести.

Наприклад:

```
string name = "Tom";
int age = 23;
Console.WriteLine($"Имя: {name} Возраст: {age}");
Console.WriteLine($"Имя: {name,-10} Возраст: {age, 10}");
```

```
int x = 8; int y = 7;
string result = $" {x} + {y} = {x + y}";
Console.WriteLine(result); // 8 + 7 = 15
```

```
Имя: Том  Возраст: 23
Имя: Том      Возраст:      23
8 + 7 = 15
```

Наприклад: Обчислити функцію при $x = 2.45$, $y = -0.423 \cdot 10^{-2}$, $z = 1.232 \cdot 10^3$

$$h = \frac{x^{2y} + e^{y-1}}{1 + x|y - \operatorname{tg} z|} + 10 \cdot \sqrt[3]{x} - \ln(z)$$

Введемо значення однієї зі змінних в кодї програми, а інші значення буде вводити користувач. Тоді програма має вигляд:

```
double x, y, z;
z = 1.232e+3;

Console.WriteLine("Input value of x: ");
CultureInfo uk = CultureInfo.GetCultureInfo("en-GB");
x = double.Parse(Console.ReadLine(), uk);
Console.WriteLine("Input value of y: ");
y = double.Parse(Console.ReadLine(), uk);

Console.WriteLine("Data for calculation x = {0:0.00}; y = {1:0.000e+0}; z = {2:0.000e+0} ", x, y, z);

double a = Math.Pow(x, 2*y) + Math.Exp(y-1);
double b = 1 + x * Math.Abs(y - Math.Tan(z));
double c = 10 * Math.Pow(x, 1 / .3) - Math.Log(z);
double h = a / b + c;

Console.WriteLine($"Result: h = {h}");
Console.ReadKey();
```

Результат роботи програми:

```
Input value of x:
2.45
Input value of y:
-0.00423
Data for calculation x = 2,45; y = -4,230e-3; z = 1,232e+3
Result: h = 191,718733120925
```

2. Завдання до лабораторної роботи №4

Написати програму для обчислення наступних функцій. У першій двох функцій організувати введення даних безпосередньо у програмі, у останніх за допомогою функції введення. Вивести інформація про введенні значення x , y , z та результат обчислення функції.

$$1. \quad s = \frac{2 \cos\left(x - \frac{2}{3}\right)}{\frac{1}{2} + \sin^2 y} \left(1 + \frac{z^2}{3 - z^2/5}\right).$$

При $x = 14.26$; $y = -1.22$; $z = 3.5 \times 10^{-2}$

$$2. \quad s = \frac{\sqrt[3]{9 + (x - y)^2}}{x^2 + y^2 + 2} - e^{|x-y|} \operatorname{tg}^3 z. \quad \text{При } x = -4.5; \ y = 0.75 \times 10^{-4}; \ z = -0.845 \times 10^2$$

$$3. \quad s = \frac{1 + \sin^2(x + y)}{\left| x - \frac{2y}{1 + x^2 y^2} \right|} x^{|y|} + \cos^2 \left(\operatorname{arctg} \frac{1}{z} \right). \quad \text{При } x = 3.74 \times 10^{-2}; \ y = -0.825; \ z = 0.16 \times 10^2$$

$$4. \quad s = |\cos x - \cos y|^{(1+2\sin^2 y)} \left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4} \right). \quad \text{При } x = 0.4 \times 10^4; \ y = -0.875; \ z = -0.475 \times 10^{-3}$$

$$5. \quad s = \ln \left(y^{-\sqrt{|x|}} \right) \left(x - \frac{y}{2} \right) + \sin^2(\operatorname{arctg}(z)). \quad \text{При } x = -15.246; \ y = 4.642 \times 10^{-2}; \ z = 21$$

Лабораторна робота № 5. Програмування алгоритмів, що розгалужуються

Мета роботи: ознайомитися з умовними операторами в C#. Навчитися використовувати на практиці умовні оператори.

1. Теоретичний матеріал

2. Завдання до лабораторної роботи №5

Написати програму для обчислення наступних функцій. При виконанні завдання передбачити вибір функції $f(x)$. Ця функція може обчислюватись як $\sin(x)$, $x^2 + x$ або e^x .

1.
$$a = \begin{cases} (f(x) + y)^2 - \sqrt[3]{|f(x)|}, & xy > 0 \\ (f(x) + y)^2 + \sin(x), & xy < 0 \\ (f(x) + y)^2 + y^3, & xy = 0. \end{cases}$$
2.
$$b = \begin{cases} \ln(f(x)) + \sqrt[3]{|f(x)|}, & x/y > 0 \\ \ln|f(x)/y| \cdot (x+y)^3, & x/y < 0 \\ (f(x)^2 + y)^3, & \text{інаше.} \end{cases}$$
3.
$$c = \begin{cases} f(x)^2 + \sqrt[3]{y} + \sin(y), & x - y = 0 \\ (f(x) - y)^2 + \ln(x), & x - y > 0 \\ (y - f(x))^2 + \operatorname{tg}(y), & x - y < 0. \end{cases}$$
4.
$$d = \begin{cases} \sqrt[3]{|f(x) - y|} + \operatorname{tg}(f(x)), & x > y \\ (y - f(x))^3 + \cos(f(x)), & y < x \\ (y + f(x))^2 + x^3, & y = x. \end{cases}$$
5.
$$g = \begin{cases} e^{f(x) - |y|}, & 0.5 < xy < 10 \\ \sqrt[3]{|f(x) + y|}, & 0.1 < xy < 0.5 \\ 2f(x)^2, & \text{інаше.} \end{cases}$$

Лабораторна робота № 6. Програмування циклів

Мета роботи: ознайомитися з циклічними операторами в C#. Навчитися використовувати на практиці циклічні оператори.

1. Теоретичний матеріал

2. Завдання до лабораторної роботи №6

1. Обчислити суму та/або добуток

$$- \sum_{c=1}^n (\sqrt{b} + c)$$

$$- \prod_{n=1}^m (a^2 + \ln(nc))$$

$$- \sum_{n=1}^k (n \cdot \sin^2(n^2 a))$$

$$- \prod_{a=2}^{10} \sum_{b=1}^{10} \cos(a + b)$$

$$- \sum_{c=1}^{10} \prod_{b=2}^8 (\sqrt{b} + c)$$

2. Обчислити суму членів ряду

$$- s = -\frac{(2x)^2}{2!} + \frac{(2x)^4}{4!} + \dots + (-1)^n \frac{(2x)^{2n}}{(2n)!} + \dots \quad \text{з точністю } 0.00001;$$

$$- s = x - \frac{x^3}{3} + \dots + (-1)^{n-1} \frac{x^{2n-1}}{2n-1} + \dots \quad \text{з точністю } 0.00005;$$

$$- s = 1 + \frac{x}{1!} \cos \cos \left(\frac{\pi}{4} \right) + \dots + \frac{x^n}{n!} \cos \cos \left(\frac{\pi}{4} n \right) + \dots \quad \text{з точністю } 0.0001;$$

Побудувати 2 блок-схеми для одного завдання з кожного пункту.

Лабораторна робота № 7. Одновимірні масиви

Мета роботи: ознайомитися з одновимірними масивами в мові програмування C#. Навчитися створювати, ініціалізувати, обробляти та виводити елементи одновимірного масиву.

1. Теоретичний матеріал

2. Завдання до лабораторної роботи №7

Написати програми для вирішення наступних задач:

- Порахувати суму всіх елементів, виключаючи третій елемент у одновимірному масиві.
- Порахувати добуток усіх парних елементів масиву, значення яких менше числа В. Число В задається користувачем.
- Порахувати кількість елементів позитивних та негативних в масиві.
- Вивести перший та останній ненульовий елемент у масиві.

Лабораторна робота № 8. Створення власних функцій

Мета роботи: ознайомитись з функціями у мові C#. Навчитися створювати, оголошувати та викликати власні функції.

1. Теоретичний матеріал

2. Завдання до лабораторної роботи №8

Написати програму, яка міститиме у собі завдання з попередньої лабораторної роботи у вигляді окремих функцій. У функції main викликати ці функції по черзі

Лабораторна робота № 9. Двовимірні масиви

Мета роботи: ознайомитися з. Навчитися.

1. Теоретичний матеріал

Текст

2. Завдання до лабораторної роботи №9

Текст

Лабораторна робота № 10. Робота зі строками

Мета роботи: ознайомитися з. Навчитися.

1. Теоретичний матеріал

Текст

2. Завдання до лабораторної роботи №10

Текст

Лабораторна робота № 11. Створення віконних додатків

Мета роботи: ознайомитися з. Навчитися.

1. Теоретичний матеріал

Текст

2. Завдання до лабораторної роботи №11

Текст

