

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ЕЛЕКТРОННОЇ ТЕХНІКИ ТА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

СХЕМОКОНСПЕКТ ЛЕКЦІЙ

з навчальної дисципліни

«Комп'ютерні технології та програмування. Частина 1»

для студентів денної та заочної форм навчання
галузі знань G Інженерія, виробництво та будівництво

Дрогобич, 2025

УДК 004.42(042.3)

С 92

Схемоконспект лекцій з навчальної дисципліни «Комп'ютерні технології та програмування. Частина 1» для студентів денної та заочної форм навчання галузі знань G Інженерія, виробництво та будівництво/ уклад. Т.В. Алтухова, Е.А. Петелін. – Дрогобич: ДонНТУ, 2025. – 274 с.

Навчальна дисципліна «Комп'ютерні технології та програмування. Частина 1» є актуальною і затребуваною в контексті підготовки висококваліфікованих фахівців у галузі знань G Інженерія, виробництво та будівництво. Схемоконспект лекцій охоплює основні питання в галузі комп'ютерних технологій та дозволить забезпечити студентам необхідну допомогу під час вивчення ними ключових положень базових тем курсу. Схематична форма конспекту акцентує увагу студентів на закріпленні основних фахових знань, формуванні відповідних умінь та навичок і в цілому отриманні ними кваліфікаційних компетентностей.

Укладачі: Тетяна АЛТУХОВА, канд. техн. наук, доцент кафедри ПМІ ДВНЗ «ДонНТУ»

Едуард ПЕТЕЛІН, канд. техн. наук, доцент, доцент кафедри ЕТКІ ДВНЗ «ДонНТУ»

Рецензент: Валерій ПОЦЕПАЄВ, канд. техн. наук, доцент, завідувач кафедри АТ ДВНЗ «ДонНТУ»

Відповідальний за випуск: Сергій КОВАЛЬОВ, канд. техн. наук, доцент, завідувач кафедри ЕТКІ ДВНЗ «ДонНТУ»

Затверджено навчально-методичним відділом ДонНТУ, протокол №6 від 29.04.2025 р.

Розглянуто на засіданні кафедри ЕТКІ ДВНЗ «ДонНТУ» протокол №9 від 03.04.2025 р.

© ДВНЗ «ДонНТУ», 2025

ЗМІСТ

ВСТУП

Тема №1. КЛАСИФІКАЦІЯ ТА АРХІТЕКТУРА ЕОМ

1.1 Класифікація ЕОМ

1.2 Архітектура ЕОМ

Тема №2. ПОДАННЯ ІНФОРМАЦІЇ В ЕОМ

2.1 Поняття про системи числення

2.2 Переведення чисел з однієї системи числення в іншу

2.3 Інформація в ПЕОМ

2.4 Форми подання чисел з фіксованою і плаваючою комою

Тема №3. АЛГОРИТМ ТА ЙОГО ВЛАСТИВОСТІ. СПОСОБИ ПОДАННЯ

3.1 Поняття алгоритма та його властивості

3.2 Форми запису алгоритмів

Тема №4. ОСНОВНІ СТРУКТУРИ АЛГОРИТМІВ

4.1 Визначення основних структур алгоритмів

4.2 Загальні правила виконання алгоритмів

Тема № 5. МОВИ І СИСТЕМИ ПРОГРАМУВАННЯ. МЕТОДИ ТА ПРИНЦИПИ ПРОГРАМУВАННЯ

5.1 Загальні визначення та поняття мови програмування. Класифікація мов програмування

5.2 Елементи мови програмування

6

7

8

16

19

20

23

29

35

41

42

44

61

62

69

76

77

83

5.3 Системи програмування	87
5.4 Методи програмування	91
5.5 Види програмного забезпечення. Загальні принципи розробки програмного забезпечення	93
5.6 Життєвий цикл програмного забезпечення	96
Тема №6. БАЗОВІ ЕЛЕМЕНТИ МОВИ ПРОГРАМУВАННЯ C/C++	98
6.1 Загальні відомості щодо мови програмування C/C++	99
6.2 Етапи написання ПЗ на C/C++	101
6.3 Базова структура програми на C/C++	102
6.4 Загальні поняття щодо бібліотечних функцій та заголовних файлів	104
6.5 Визначення основних характерних рис мови програмування C/C++	106
Тема №7. ТИПИ ДАНИХ	120
7.1 Загальна класифікація типів даних та операції над ними	121
7.2 Прості структури даних	123
7.3 Статичні структури даних	127
7.4 Напівстатичні структури даних	136
7.5 Динамічні структури даних	142
7.6 Нелінійні структури даних	149
Тема №8. ВИРАЗИ ТА ОПЕРАЦІЇ	158
8.1 Визначення базових виразів та операцій в мові програмування C/C++	159
8.2 Аналіз виконання операцій	164

8.3 Узгодження та перетворення типів операндів у виразах	181
Тема №9. ОПЕРАТОРИ МОВИ СІ	185
9.1 Умовні оператори: if, switch	186
9.2 Оператори циклу: while, do-while, for	207
9.3 Оператор переходу goto	221
Тема №10. ФУНКЦІЇ	227
10.1 Базові поняття функцій та їх правила написання на мові програмування C/C++	228
10.2 Виконання викликів функцій та способи передачі параметрів	233
10.3 Рекурсивні функції	238
10.4 Створення власної бібліотеки	240
10.5 Області оголошення та доступу до імен. Специфікатори та макроси	241
Тема №11. БАЗОВІ ПОНЯТТЯ ПРО ВКАЗІВНИКИ	245
11.1 Оголошення вказівників	246
11.2 Операції над вказівниками	249
11.3 Вказівник void та типізація вказівників	253
11.4 Використання вказівників для обміну даними між функціями	254
Тема №12. ПРЕПРОЦЕСОР	255
12.1 Базове визначення препроцесора. Визначення стадії препроцесорної обробки	256
12.2 Дерективи препроцесора та їх обробка	258
12.3 Умовна компіляція, мультирозгалуження та нумерація рядків	267
12.4 Операції defined, # та ##	271
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ	273

ВСТУП

Сучасний стан інформаційних технологій (ІТ) характеризується швидким розвитком та трансформацією багатьох галузей, тому питання застосування комп'ютерних технологій є на даний час досить актуальним. З огляду на це, **метою** викладання даної дисципліни є **формування базових теоретичних знань та практичних навичок в сфері комп'ютерних технологій та програмування на мові програмування C/C++**.

В результаті **опанування** навчальної дисципліни «Комп'ютерні технології та програмування. Частина 1» студент повинен отримати:

Знання: основні принципи алгоритмізації; основні поняття і методи (технології) програмування; етапи розробки програм; програмні середовища Dev-C++, Visual Studio, Microsoft Visual C++, Xcode та онлайн компілятори C/C++; конструкції мови Cі.

Уміння: працювати в середовищі операційної системи Windows для практичного використання систем програмування та наявних прикладних пакетів; здійснювати декомпозицію рішення задачі розв'язування типових математичних та прикладних задач і складати алгоритми окремих її частин відповідно до сучасної технології програмування; на підставі розроблених алгоритмів засобами Dev-C++, Visual Studio, Microsoft Visual C++, Xcode або онлайн компіляторів C/C++ розробляти та налагоджувати програмні коди мовою програмування C/C++.

При вивченні курсу «Комп'ютерні технології та програмування. Частина 1» передбачено опанування студентами 12 тем, що поділені на декілька змістовних тематичних питань. Стисле, але вичерпне подання матеріалу забезпечить студентам можливість зосередитися на основних ключових поняттях в галузі комп'ютерних технологій та програмування та в цілому сприятиме формуванню в них необхідних для

Тема №1. КЛАСИФІКАЦІЯ ТА АРХІТЕКТУРА ЕОМ

Тематичний план

- 1.1 Класифікація ЕОМ*
- 1.2 Архітектура ЕОМ*

1.1 КЛАСИФІКАЦІЯ ЕОМ

Електронна обчислювальна машина, комп'ютер - комплекс технічних засобів, призначених для автоматичної обробки інформації в процесі вирішення обчислювальних та інформаційних задач.

*Класифікація за
елементною базою, на якій
будуються ЕОМ*

**Перше покоління ЕОМ на базі електронних ламп
(1943 р.)**

**Друге покоління ЕОМ на базі напівпровідникових
елементів (транзисторах) (50-60 р.р.)**

**Третє покоління ЕОМ на базі інтегральних схем
малої і середньої інтеграції (60-70 р.р.)**

**Четверте покоління ЕОМ на базі великих
інтегральних схемах (мікропроцесори) (70-80 р.р.)**

**П'яте покоління ЕОМ на базі надвеликих
інтегральних схемах (90-ті роки)**

**Шосте покоління ЕОМ – оптоелектронні ЕОМ з
нейронною структурою**

Рисунок 1.1

Перше покоління

- Поява електронно-вакуумних ламп (40-ві роки ХХ ст.)
- Відмова від релейних машин
- Поява електронної обчислювальної машини (ЕОМ). Розширилося коло розв'язувальних завдань: розрахунки інженерних споруд, розрахунки руху планет, балістичні розрахунки тощо.



Рисунок 1.2 Радіолампа



*Рисунок 1.3 Програмісти біля пульта керування
ЕОМ МЕМС 1951 р.*

1943 - 1946 pp. у США **ЕНІАК** (ENIAC-Electronic Numerical Integrator and Calculator - електронно-числовий інтегратор і обчислювач)

1951 р. в СРСР була створена «МЗСМ» (мала електронно-рахувальна машина). Інститут електродинаміки (м. Київ). Конструктор С.О. Лебедєв.

Характерні риси:

- застосування електронних ламп у цифрових схемах
- великі габарити
- трудомісткий процес програмування

Друге покоління

- Поява напівпровідникових приладів (середина 50-х років)
- Розподілення обчислювальних машин на великі, середні й малі

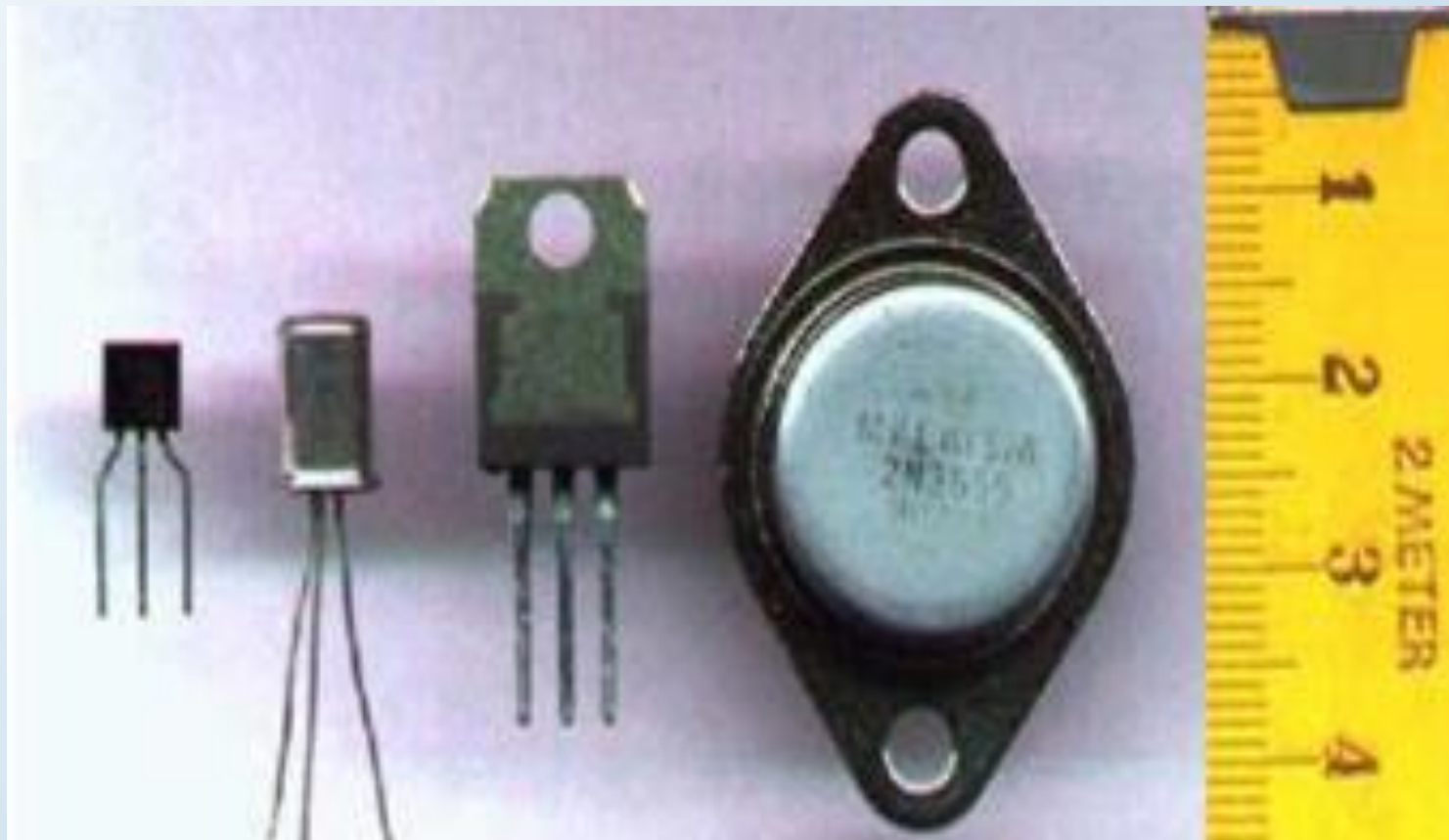


Рисунок 1.4 Транзистори, на базі яких створено ЕОМ

«Днепр-1», завод «Арсенал» (м. Київ).

«**Мир-1**» (модифікація **Мир-2**), 1965 р. Інститут кібернетики (керівник В.М. Глушков).

Малі ЕОМ «**Раздан**» і «**Наірі**». Середні ЕОМ серій «**Урал**», «**М-20**» і «**Мінск**».

ЕОМ «**БЗСМ-6**» під керівництвом академіка С. О.Лебедєва.

«**Елліот**» (Англія), «**Сіменс**» (ФРН), «**Стретч**» (США).

Характерні риси:

- застосування напівпровідникових елементів
- використання алгоритмічних мов програмування

Третє покоління

- Поява інтегральних схем (кінець 60-х років)
- Поява великих та середніх серії ЕОМ
 - ✓ ІВМ 360/370, США;
 - ✓ ЄС ЕОМ (Єдина Система ЕОМ, машини великі і середні), СМ ЕОМ (Система Малих ЕОМ) і «Електроника» (система мікро-ЕОМ), Радянський Союз

Характерні риси:

- застосування інтегральних схем
- можливість використання розвинутих мов програмування (мов високого рівня)



Рисунок 1.5 Мікросхеми

Четверте покоління

- Поява великих інтегральних схем (ВІС) - по н / ч.
- Поява першого мікропроцесора (4-розрядний Intel 4004, США, 1971 р.)
- Поява персонального комп'ютера (ПК)

Характерні риси: застосування мікропроцесорів; зник термін «ЕОМ», а його місце зайняло слово «комп'ютер»

П'яте покоління - поява супермасштабних ВІС (середина 90-х років)

Характерні риси: використання штучного інтелекту і природних мов спілкування

Шосте покоління ЕОМ - поява оптоелектронних ЕОМ з масовим паралелізмом і нейронної структурою

Характерні риси: нейронна структура з розподіленою ступенем великого числа нескладних мікропроцесорів, моделюючих архітектуру нейронних біологічних систем.


```
graph LR; A[Система класифікації ЕОМ] --- B[За продуктивністю і швидкодією]; A --- C[За призначенням]; A --- D[За рівнем спеціалізації]; A --- E[За типом процесора, що використовується]; A --- F[За особливостями архітектури]; A --- G[За розмірами];
```

Система класифікації ЕОМ

За продуктивністю і швидкодією

За призначенням

За рівнем спеціалізації

За типом процесора, що використовується

За особливостями архітектури

За розмірами

Рисунок 1.6

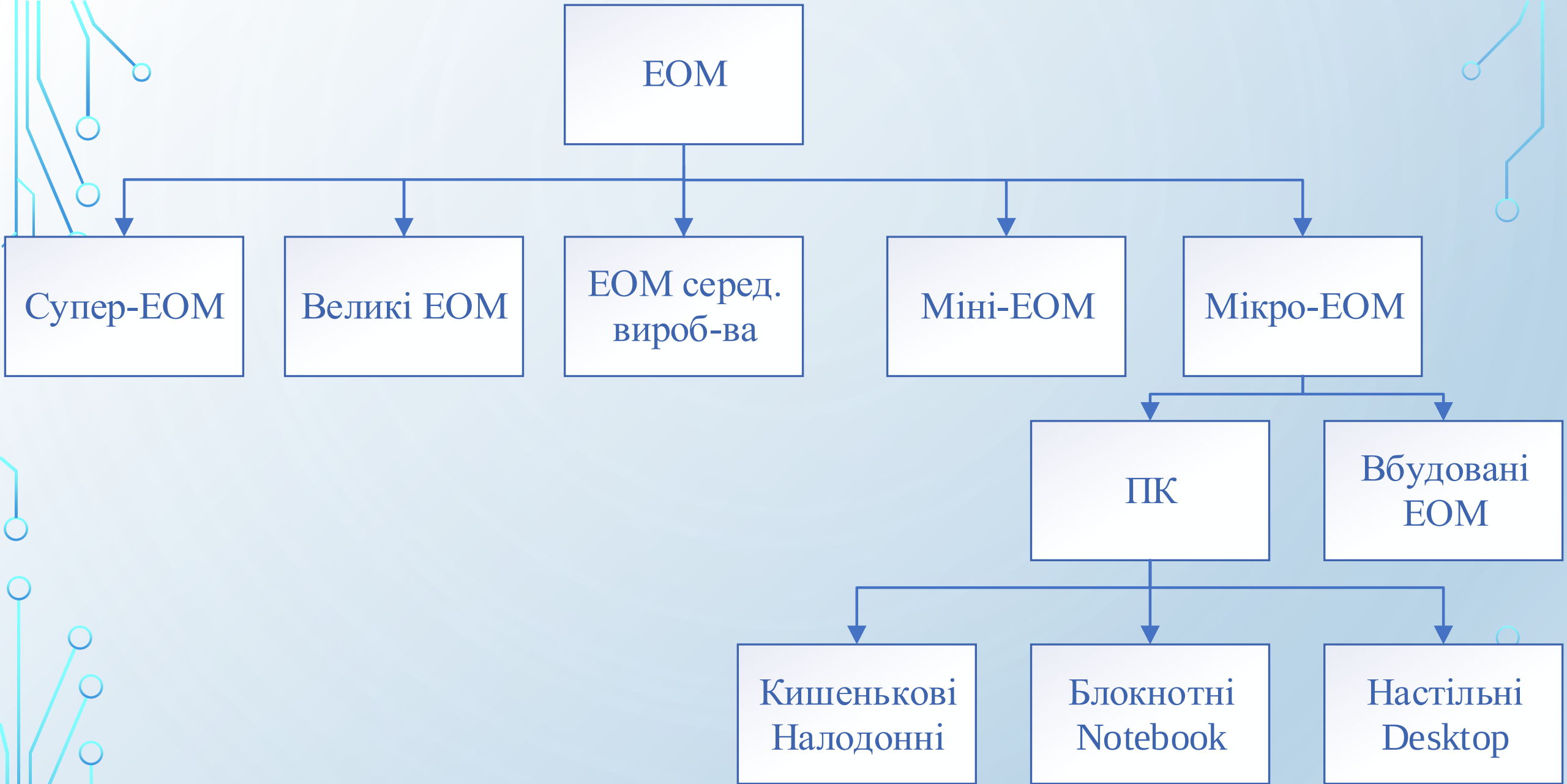


Рисунок 1.7 Класифікація за обчислювальною потужністю та габаритам

1.2 АРХІТЕКТУРА ЕОМ

Архітектура ЕОМ включає в себе як структуру, яка відображатиме склад ПК, так і програмно - математичне забезпечення.

Структура ЕОМ – сукупність елементів і зв'язків між ними. Основним принципом побудови всіх сучасних ЕОМ є програмне керування.

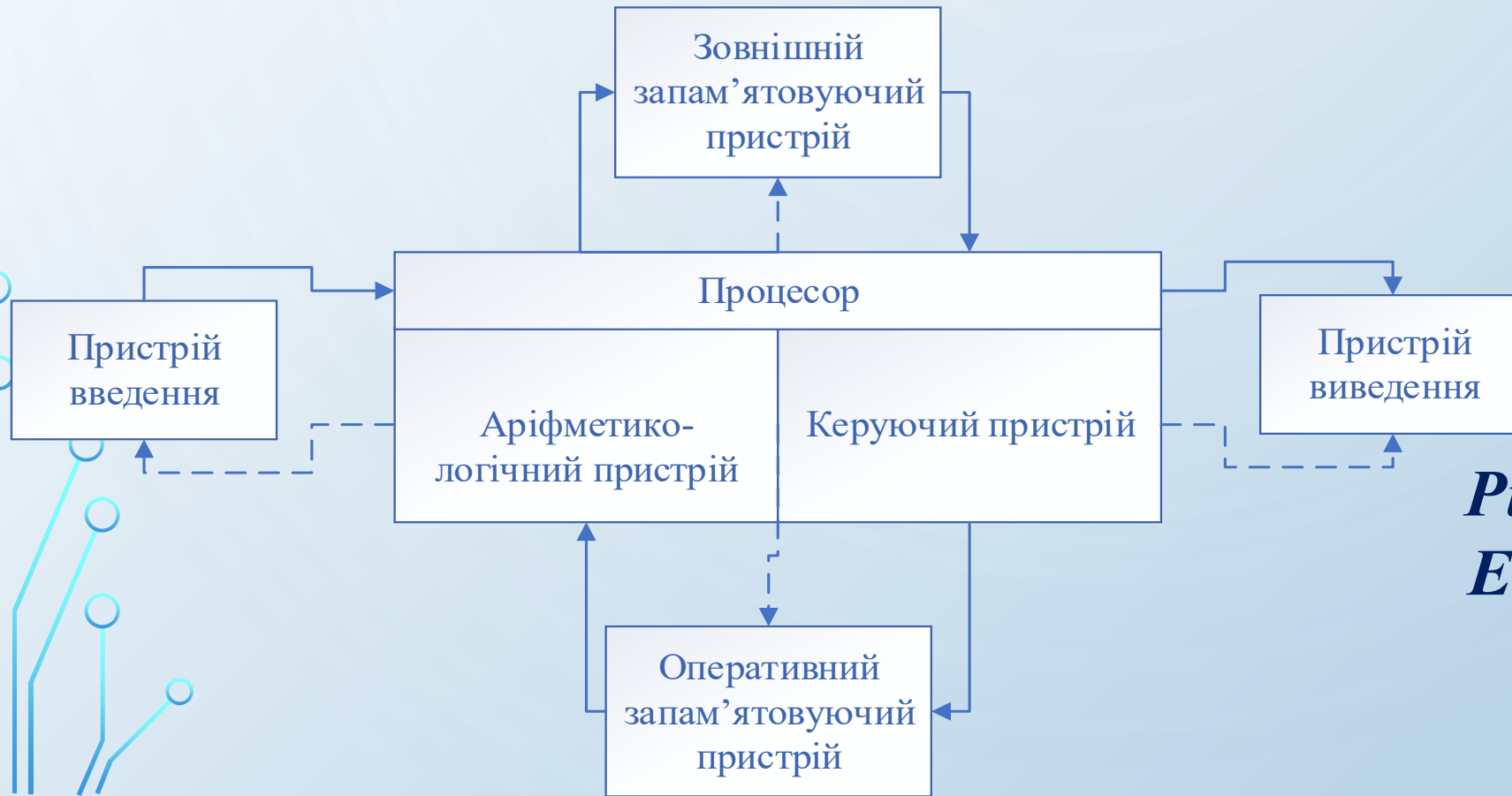
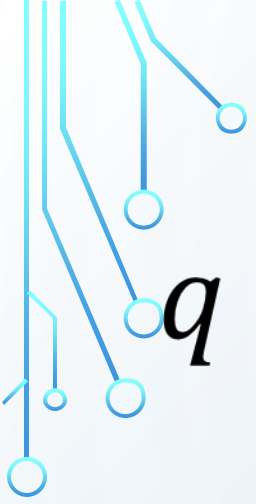


Рисунок 1.8 Структура ЕОМ за фон Нейманом



$$q = \frac{ax + bx}{c - d}$$

Умовно-машинна мова

1)	(-)	< c >	< d >	R1
2)	(*)	< a >	< x >	R2
3)	(*)	< b >	< y >	R3
4)	(+)	R2	R3	R2
5)	(:)	R2	R1	< q >

Машинна мова

Адреса команди

Машинна команда

10100011

0101011111001101110011100101000000

10100100

010110001001100110011101101000001

10100101

01011000100110101001111010100010

10100110

0101011010100000110100010101000001

10100111

0101100110100000110100000010011111



Тема №2. ПОДАННЯ ІНФОРМАЦІЇ В ЕОМ

Тематичний план

2.1 Поняття про системи числення

2.2 Переведення чисел з однієї системи числення в іншу

2.3 Інформація в ПЕОМ

2.4 Форми подання чисел з фіксованою і плаваючою

2.1 ПОНЯТТЯ ПРО СИСТЕМИ ЧИСЛЕННЯ

Система числення - це спосіб запису чисел за допомогою обмеженого набору спеціальних знаків (цифр). Як правило, в якості цифрових знаків використовуються арабські і римські цифри.

Системи числення поділяються на **позиційні** і **непозиційні**.

Римська система

1 - I, 2 - II, 3 - III, 4 - IV, 5 - V, 6 - VI, 7 - VII, 8 - VIII, 9 - IX, 10 - X, 50 - L, 100 - C, 500 - D, 1000 - M і т.д.

1983 - M C M X C V I I

$$1000 - 100 + 1000 - 10 + 100 - 5 - 1 - 1 = 1983$$

Давньослов'янська нумерація

333 - Т Л Г,

де Т = 300, Л = 30, Г = 3

$$83887: 80000 + 3000 + 800 + 80 + 7$$

$$a_n a_{n-1} a_{n-2} \dots a_2 a_1 a_0, b_1 b_2 \dots b_m$$

a_i — i -та цифра цілої частини,

b_j — j -та цифра дробової частини числа

$$a_n \cdot u^n + a_{n-1} \cdot u^{n-1} + \dots + a_1 \cdot u^1 + a_0 \cdot u^0 + \\ + b_1 \cdot u^{-1} + b_2 \cdot u^{-2} + \dots + b_m \cdot u^m$$

u — основа системи числення

Основа позиційної системи числення – кількість різних знаків або символів, які використовуються для зображення цифр в даній системі.

$$83887,47 = 8 \cdot 10^4 + 3 \cdot 10^3 + 8 \cdot 10^2 + 8 \cdot 10^1 + 7 \cdot 10^0 + 4 \cdot 10^{-1} + 7 \cdot 10^{-2}$$

Таблиця 2.1 – Системи числення

<i>Основа системи числення</i>	<i>Назва</i>	<i>Символи, що використовуються в системі числення</i>
10	десяткова	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
2	двійкова	0, 1
8	вісімкова	0, 1, 2, 3, 4, 5, 6, 7
16	шістнадцяткова	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, А, В, С, D, E, F

2.2 ПЕРЕВЕДЕННЯ ЧИСЕЛ З ОДНІЄЇ СИСТЕМИ ЧИСЛЕННЯ В ІНШУ

Двійкова система числення

2 → 10

$$11010,101_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + \\ + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 16 + 8 + 2 + \frac{1}{2} + \frac{1}{8} = 26 \cdot \frac{5}{8} = 26,625_{10}$$

~~10~~ → 2

$$43,375_{10} = 101011,011_2$$

43 | 2

$$\begin{array}{r} 42 \\ \hline \end{array} \quad \begin{array}{r} 21 \\ \hline \end{array} \quad \begin{array}{r} | \\ \hline \end{array} \begin{array}{r} 2 \\ \hline \end{array}$$

$$1 \quad \underline{20} \quad 10 \quad \underline{2}$$

$$1 \quad \underline{10} \quad 5 \quad \underline{2}$$

$$0 \quad \underline{4} \quad 2 \quad \underline{2}$$

1 2 1

0

$$0, \mid 4 \cdot 2$$

$0, \quad | \quad 8 \cdot 2$

1, | 6 · 2

1, | 2 · 2

0, | 4 · 2

0, | 8.2

1, 16.2

$$0,4_{10} = 0,011001100110..._2 = 0,(0110)_2$$

$$0, \mid 375 \cdot 2$$

$$0, \mid 75 \cdot 2$$

$$1, \mid 5 \cdot 2$$

1, | 0

Вісімкова система числення

8 → 10

$$\begin{aligned} 35174,6_8 &= 3 \cdot 8^4 + 5 \cdot 8^3 + 1 \cdot 8^2 + 7 \cdot 8^1 + 4 \cdot 8^0 + \\ &+ 6 \cdot 8^{-1} = 3 \cdot 4096 + 5 \cdot 512 + 1 \cdot 64 + 7 \cdot 8 + 4 + \frac{6}{8} = \\ &= 12288 + 2560 + 64 + 56 + 4 + \frac{3}{4} = 14972,75_{10} \end{aligned}$$

10 → 8

$$\begin{array}{r} 397 \quad \underline{8} \\ 32 \quad 49 \quad \underline{8} \\ 77 \quad \underline{48} \quad 6 \\ 72 \quad 1 \\ 5 \end{array}$$

$$397,2_{10} = 615,15_8$$

$$0, \mid 2 \cdot 8$$

$$1, \mid 6 \cdot 8$$

$$4, \mid 8 \cdot 8$$

$$6, \mid 4 \cdot 8$$

$$3, \mid 2 \cdot 8$$

$$1, \mid 6 \cdot 8$$

.....

$$0,2_{10} = 0,146314631... = 0,(1463)_8$$

Правило округлення: щоб округлити дробове число до m цифр, потрібно до $(m+1)$ -ій цифрі додати половину ціни розряду для даної системи числення, після чого відкинути всі дробові цифри, починаючи з $(m+1)$ -ої.

Для 8 с/ч половина ціни розряду дорівнює 4, для 2 с/ч - 1, для 16 с/ч - 8.

Для **$m=2$** :

$$\begin{array}{r} 0,14631463 \\ + \quad \quad 4 \\ \hline |0,15|231463 \end{array}$$

8 → 2

Таблиця 2.2 – Перетворення з 8 в 2 с/с

<i>Вісімкове число</i>	<i>Двійкове число</i>	<i>Двійкова тріада</i>
0	0	000
1	1	001
2	10	010
3	11	011
4	100	100
5	101	101
6	110	110
7	111	111

$$3763,24_8 = 011111110011,010100_2$$

2 → 8

$1|011|110|111, 101|01_2 = 001|011|110|111, 101|010 == 1367,552_8$

Таблиця 2.3 – Шістнадцяткова система числення

<i>Двійкова система числення</i>	<i>Шістнадцяткова система числення</i>	<i>Двійкова система числення</i>	<i>Шістнадцяткова система числення</i>
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

2.3 ІНФОРМАЦІЯ В ПЕОМ

біт, від слів **B**inary **dig**it - двійкова цифра

Нумерація: 0, 1, 2, 3, 4, 5, 6, 7

$$0000\ 0000 = 00$$

$$0000\ 0001 = 01$$

$$0000\ 0010 = 02$$

.....

$$1111\ 1111 = FF_{16} = 255_{10}.$$

ASCII

Цифра **6** - 00110110 (36)

Буква **К** - 01001011 (4B)

$$1\text{ Кбайт} = 2^{10}\text{ байт} = 1024\text{ байта};$$

$$1\text{ Мбайт} = 2^{10}\text{ Кбайт} = 2^{20}\text{ байт};$$

$$1\text{ Гбайт} = 2^{10}\text{ Мбайт} = 2^{30}\text{ байт}.$$

Дані:

➤ **Цілі числа (числа з фіксованою комою)**

Int – 2 байти. Нумерація бітів: 0, 1, ..., 15

"0" - це знак "+", "1" - знак "-"

$3104_{10} = C20_{16}$ в форматі int має вигляд
 $0C20 = 0000\ 1100\ 0010\ 0000$

➤ **Дійсні числа (числа з плаваючою комою)**

Float – 4 байти. Нумерація бітів: 0, 1, 2, ..., 31

3 байта займає мантиса числа, 1 байт – характеристика (порядок) та знак.

➤ **Логічні дані**

“сніг білий”, “цього дня - п'ятниця”, “ $x > 0$ ”, “ $a + b < z$ ”

0 (вислів хибний) або **1** (вислів істинно)
16 логічних операцій

➤ *Заперечення (операція НЕ). Таблиця операції:*

$$\overline{0} = 1$$

$$\overline{1} = 0$$

Читається: "Не нуль є одиниця".

➤ *Логічне множення (кон'юнкція, операція І). Таблиця операції:*

$$0 \wedge 0 = 0$$

$$0 \wedge 1 = 0$$

$$1 \wedge 0 = 0$$

$$1 \wedge 1 = 1$$

Читається: "Нуль і нуль є нуль".

➤ *Логічне додавання (диз'юнкція, операція АБО). Таблиця операції:*

$$0 \vee 0 = 0$$

$$0 \vee 1 = 1$$

$$1 \vee 0 = 1$$

$$1 \vee 1 = 1$$

Читається: "Нуль або нуль є нуль".

Логічний тип (тип bool).

Логічне значення – одне з двох значень істинності, які позначають іменами false і true (false еквівалентно значенням 0, true - значенням 1 в алгебрі логіці; використання імен false і true викликано прагненням відрізнити в програмі логічні значення від числових значень 0 і 1).

$x + 1 < y$ при $x = 0, y = 0$ дає **false**; при $x = 0, y = 10$ - значення **true**.

Для змінних логічного типу визначені чотири логічні операції:

- 1) заперечення **not**;
- 2) логічне множення **and**;
- 3) логічне додавання **or**;
- 4) виключне АБО (заперечення рівнозначності, додавання за модулем 2) **xor**.

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

(a or b) and (not a or not b).

$$a \oplus b = (a \vee b) \wedge (\bar{a} \vee \bar{b})$$

$((x + 1 < y) \text{ and not } (x > 5)) \text{ or } ((y > 0) \text{ and true})$

при $x = 5, y = 5$

$((6 < 1) \text{ and not } (5 > 5)) \text{ or } ((5 > 0) \text{ and true}) \Rightarrow (\text{false and not false}) \text{ or}$

$(\text{true and true}) \Rightarrow (\text{false and true}) \text{ or } (\text{true and true}) \Rightarrow \text{false or true} \Rightarrow \text{true} .$

У символіці алгебри логіки:

$$(0 \wedge \bar{0}) \vee (1 \wedge 1) = (0 \wedge 1) \vee (1 \wedge 1) = 0 \vee 1 = 1$$

2.4 ФОРМИ ПОДАННЯ ЧИСЕЛ З ФІКСОВАНОЮ І ПЛАВАЮЧОЮ КОМОЮ

<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>
-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	------------------

MAX = +999999,9999

MIN = 0,0001

Діапазон значень: 0,0001 .. 1000000

Способи представлення чисел з фіксованою комою:

- кома фіксована перед старшим розрядом
- кома фіксована після молодшого розряду

Числа з плаваючою комою:

$$q \cdot 10^p$$

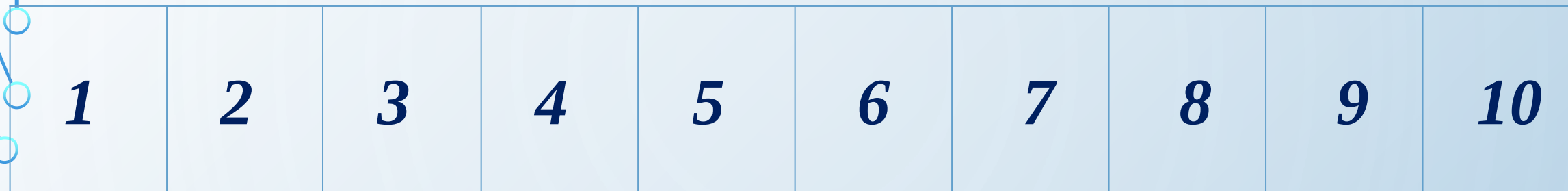
q – мантиса,

p – порядок,

10 – основа системи числення, що записано в цій ж системі

$$358,5 = 0,3585 \cdot 10^3 = 358,5 \cdot 10^0 = 3585 \cdot 10^{-1} = 0,003585 \cdot 10^5$$

Обмеження мантиси: $0,1 \leq q < 1$ (в даній с/ч).



8 розрядів – мантиса , 2 розряда – порядок

MAX мантиси = 0,99999999 ~ 1

MAX порядку = 99

MAX Число = $1 \cdot 10^{99}$

Додавання чисел з фіксованою та плаваючою комою

$$x_1 = 78535, \quad x_2 = 416.$$

У форматі цілих чисел:

$$x_1 = 0000078535, \quad x_2 = 0000000416.$$

Сума

$$\begin{array}{r} 0000078535 \\ + 0000000416 \\ \hline 0000078951 \end{array}$$

У форматі з плаваючою комою: $x_1 = 0,78535 \cdot 10^5$, $x_2 = 0,416 \cdot 10^3$

або в машинному поданні: $x_1 = 7853500005$, $x_2 = 4160000003$.

Алгоритм додавання:

- Крок 1. Визначається різниця порядків доданків $\Delta p = p_1 - p_2$, в даному випадку $\Delta p = 2$
- Крок 2. Якщо $\Delta p > 0$, то q_2 зсувається вправо на Δp розрядів. Якщо $\Delta p < 0$, то q_1 зсувається вправо на Δp розрядів. Якщо $\Delta p = 0$, то зсув відсутній. В даному випадку після зсуву $q_1 = 0,78535000$, $q_2 = 0,00416000$
- Крок 3. Виконується додавання мантий:

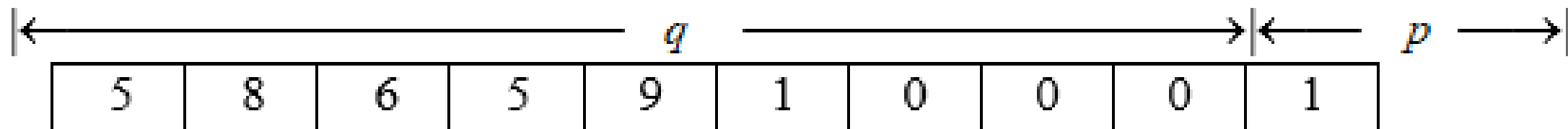
$$q = q_1 + q_2 = 0,78951000$$

- Результату приписується максимальний з порядків доданків

$$p = \max(p_1 - p_2) = 5$$

Результат $y = x_1 + x_2 = 789510005$

Походження назви «число з плаваючою комою»



$$x = 0,586591 \cdot 10^1 = 5,86591$$

Якщо розглядати його як число з фіксованою комою, то кома, що розділяє, повинна бути поставлена після першого розряду ($p=1$).

5	8	6	5	9	1	0	0	0	2
---	---	---	---	---	---	---	---	---	---

Для даного кома буде встановлена після другого розряду ($x = 58,6591$; $p=2$).

5	8	6	5	9	1	0	0	0	3
---	---	---	---	---	---	---	---	---	---

Для числа кома встановлюється після третього розряду ($x = 586,591$) і т.д.

Тема №3. АЛГОРИТМ ТА ЙОГО ВЛАСТИВОСТІ. СПОСОБИ ПОДАННЯ

Тематичний план

3.1 Поняття алгоритма та його властивості

3.2 Форми запису алгоритмів

3.1 ПОНЯТТЯ АЛГОРИТМА ТА ЙОГО ВЛАСТИВОСТІ

Алгоритмізація - складання покрокового опису процесу рішення задачі.

Алгоритм - система чітких однозначних вказівок, яка визначає послідовність дій над деякими об'єктами і після кінцевого числа кроків призводить до отримання необхідного результату.

Слово **«алгоритм»** від перського математика Аль-Хорезми (в **IX ст.**).
Правила чотирьох арифметичних дій.

XX ст. – поява галузі теорія алгоритмів.

Чисельні алгоритми - алгоритми, відповідно до яких рішення поставлених завдань зводиться до арифметичних дій.

Логічні алгоритми - алгоритми, відповідно до яких рішення поставлених завдань зводиться до логічних дій.

Властивості алгоритму:

- **Дискретність** - поділ виконання рішення задачі на окремі операції.
- **Детермінованість (визначеність)** - кожна команда алгоритму повинна однозначно визначати дії виконавця.
- **Результативність (кінцівка)** - завершення роботи алгоритму за кінцеве число кроків (при цьому кількість кроків може бути заздалегідь невідомим і різним для різних вихідних даних).
- **Масовість (універсальність)** - алгоритм рішення задачі розробляється в загальному вигляді, тобто можливість вирішення класу задач, що розрізняються лише вихідними даними.
- **Чіткість** - зміст допустимого набору команд, зрозумілого конкретному виконавцю.

3.2 ФОРМИ ЗАПИСУ АЛГОРИТМІВ

Графічний спосіб представлення алгоритмів має ряд переваг завдяки візуальності і явного відображення процесу виконання завдання. Алгоритми, представлені графічними засобами, отримали назву блок-схем.

Текстовий опис алгоритму є досить компактним і може бути реалізовано на *природній мові* або *спеціальній (алгоритмічній) мові* у вигляді програми.

Опис алгоритмів природною мовою

2, 3, +

1-ша операція - зчитування в стек значення

2-га операція - зчитування другого значення перше значення при цьому зсувається в другий осередок пам'яті.

3-тя операція - обчислення суми двох лічених значень (вони називаються **операндами**).

Результат цієї операції - значення 5 - **записується в перший осередок стека.**

Алгоритм

Крок 1. Зчитати число a.

Крок 2. Зчитати число b.

Крок 3. Виконати додавання $c := a + b$.

Крок 4. Вивести число c

Конструкція в програмі

$$a = a + 1$$

Рівняння в математиці

$$a = a + 1$$

$$0=1$$

Приклад 3.1 Скласти алгоритм вирішення наступного завдання. Нехай задані два значення x і y . Необхідно порівняти ці значення і надрукувати ім'я більшої змінної.

Алгоритм

Крок 1. Ввести значення x .

Крок 2. Ввести значення y .

Крок 3. Якщо $x < y$, то надрукувати « y », інакше надрукувати « x ».

Приклад 3.2 Знайти $y = \max(a, b, c)$

Алгоритм

Крок 1. Покласти, що $y = a$.

Крок 2. Якщо $b > y$, то покласти, що $y = b$.

Крок 3. Якщо $c > y$, то покласти, що $y = c$. Завершення алгоритму.

Приклад 3.3 Знайти найбільший спільний дільник двох цілих чисел.

Наприклад, для $m=420$ та $n=90$

$$420 = 2 \cdot 2 \cdot 3 \cdot 5 \cdot 7 \quad 90 = 2 \cdot 3 \cdot 3 \cdot 5$$

Найбільший спільний дільник в цьому випадку дорівнює $2 \cdot 3 \cdot 5 = 30$.

Алгоритм Евкліда

$$d(m, n), \quad d(m, 0) = m$$

Крок 1. Якщо $n = 0$, то прийняти $d = m$ та завершити обчислення, або перейти до кроку 2.

Крок 2. Обчислити $q = [m/n]$,
 $r = m - qn$

Крок 3. Замінити значення m на n значення, а значення n на значення r .
Перейти до кроку 1.

q — ціла частина від ділення m на n , r — залишок від ділення.

Приклад 3.4 $m=420$ та $n=90$

Крок 1. $n = 90 \neq 0$

Крок 2. $q = [420/90] = 4$,
 $r = 420 - 4 \cdot 90 = 60$

Крок 3. $m=90$ та $n=60$

Крок 1. $n = 60 \neq 0$

Крок 2. $q = [90/60] = 1$,
 $r = 90 - 1 \cdot 60 = 30$

Крок 3. $m=60$ та $n=30$

Крок 1. $n = 30 \neq 0$

Крок 2. $q = [60/30] = 2$,
 $r = 60 - 2 \cdot 30 = 0$

Крок 3. $m=30$ та $n=0$

Крок 1. $n = 0 \rightarrow d = 30$

Опис алгоритмів за допомогою блок-схем

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

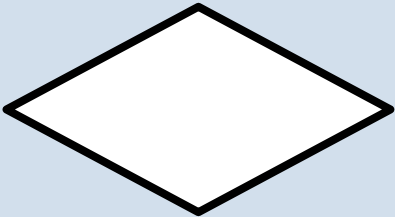
ДСТУ ISO 5807:2016 Обробляння інформації. Символи та угоди щодо документації стосовно даних, програм та системних блок-схем, схем мережевих програм та схем системних ресурсів (ISO 5807:1985, IDT) [1]

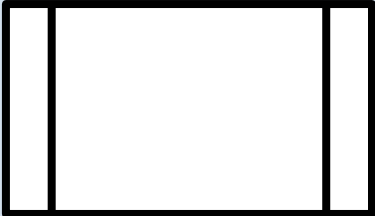
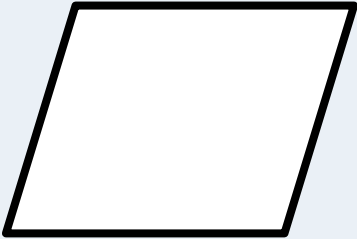
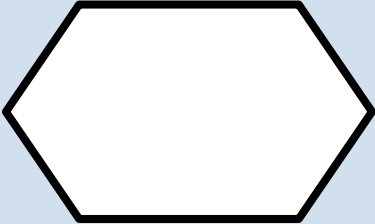
Основні *правила виконання блок-схем*:

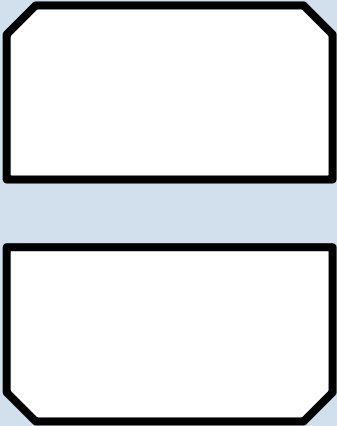
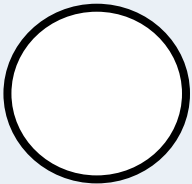
- Для зображення основних алгоритмічних структур і блоків на блок-схемах використовують спеціальні графічні символи.
- При виконанні блок-схем всередині кожного блоку вказується пояснювальна інформація, яка характеризує дії, що виконуються цим блоком.
- Потoki даних в схемах показуються лініями. Напрямок потоку зліва направо і зверху вниз вважається стандартним. У випадках, коли необхідно внести велику ясність в схему або потік має напрямок відмінне від стандартного, на лініях використовуються стрілки, що вказують цей напрямок.
- У схемах слід уникати перетину ліній. Пересічні лінії не мають логічного зв'язку між собою, тому зміни напрямку в точках перетину не допускаються. Якщо дві або більше вхідних лінії об'єднуються в одну вихідну лінію, то місце об'єднання ліній зміщується.
- Кількість вхідних ліній не обмежена, лінія, що виходить з блоку повинна бути одна, за винятком логічного блоку.

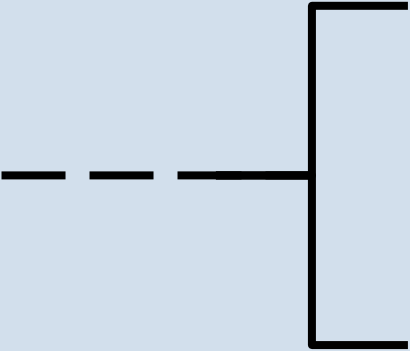
Таблиця 3.1 – Базові елементи

№ зп	Найменування	Позначення	Функціональні можливості
1	Початок (завершення)		Елемент відображає вхід у зовнішнє середовище або вихід з нього (найчастіше застосування - початок і кінець програми). Всередині фігури записується відповідна дія.
2	Процес		Елемент відображає одну або кількох операцій, обробку даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

№ зп	Найменування	Позначення	Функціональні можливості
3	Умова		Елемент відображає обробку умови, рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елемента. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елемента. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижньої). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижньої) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

№ зп	Найменування	Позначення	Функціональні можливості
4	Функція (процедура)		Елемент відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.
5	Ввід / вивід		Елемент відображає перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).
6	Цикл з параметром		Елемент відображає заголовок циклу з параметром. У ньому через крапку з комою вказуються ім'я змінної (параметра) з початковим значенням, граничне значення параметра (або умова виконання циклу), крок зміни параметра.

№ зп	Найменування	Позначення	Функціональні можливості
7	Межа циклу		Елемент складається з двох частин - відповідно, початок і кінець циклу - операції, що виконуються всередині циклу, розміщуються між ними. Умови циклу і збільшення записуються всередині символу початку або кінця циклу - в залежності від типу організації циклу. Часто для зображення на блок-схемі циклу замість цього символу використовують символ рішення, вказуючи в ньому умову, а одну з ліній виходу замикають вище в блок-схемі (перед операціями циклу).
8	З'єднувач		Елемент відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

№ зп	Найменування	Позначення	Функціональні можливості
9	Коментар		Елемент використовується для детальнішої інформації про кроки, процесу або групи процесів. Опис поміщається з боку квадратної дужки і охоплюється нею по всій висоті. Пунктирна лінія йде до описуваного елементу, або групи елементів (при цьому група виділяється замкнутою пунктирною лінією). Також символ коментаря слід використовувати в тих випадках, коли обсяг тексту в будь-якому іншому символі (наприклад, символ процесу, символ даних та ін.) перевищує його обсяг.

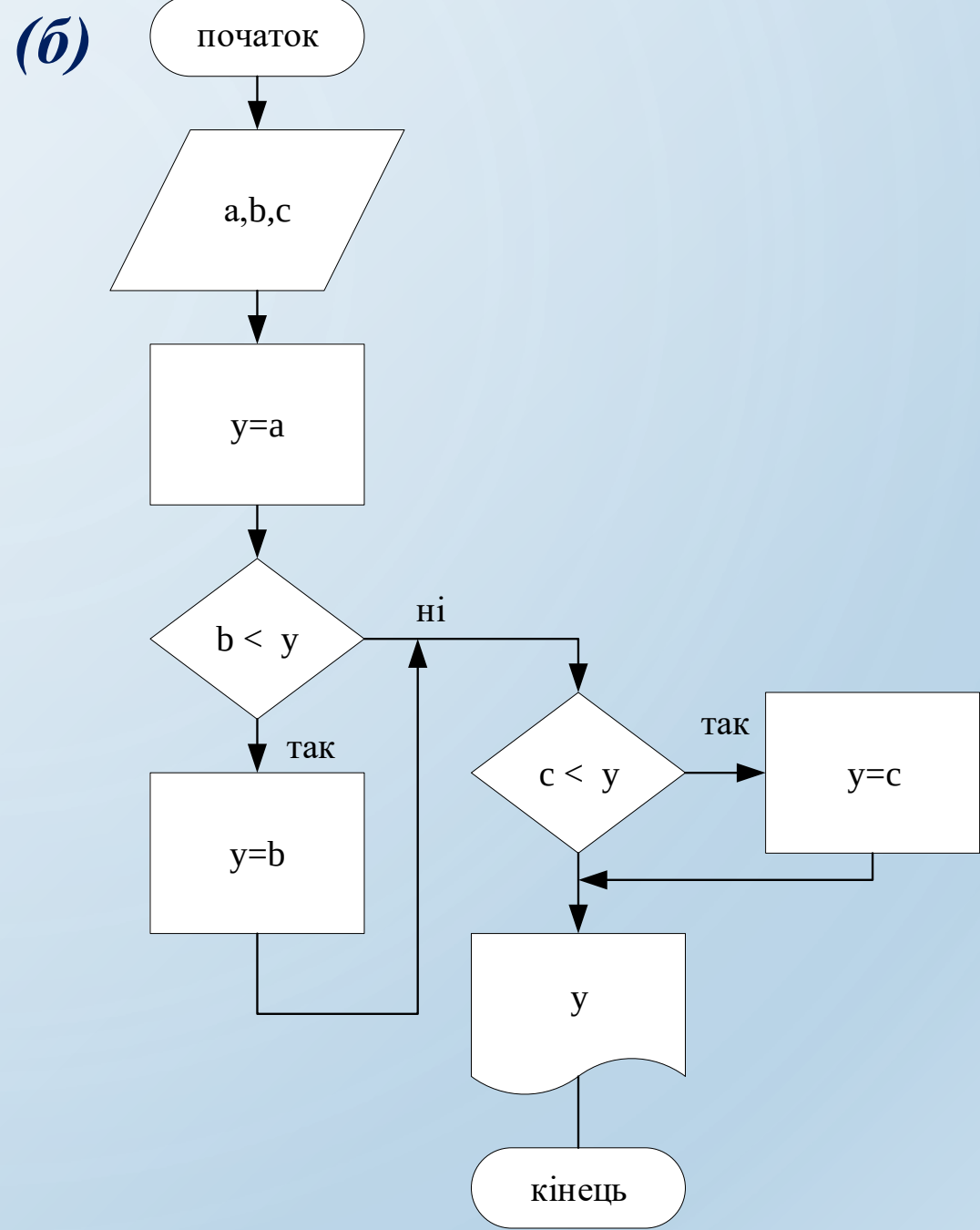
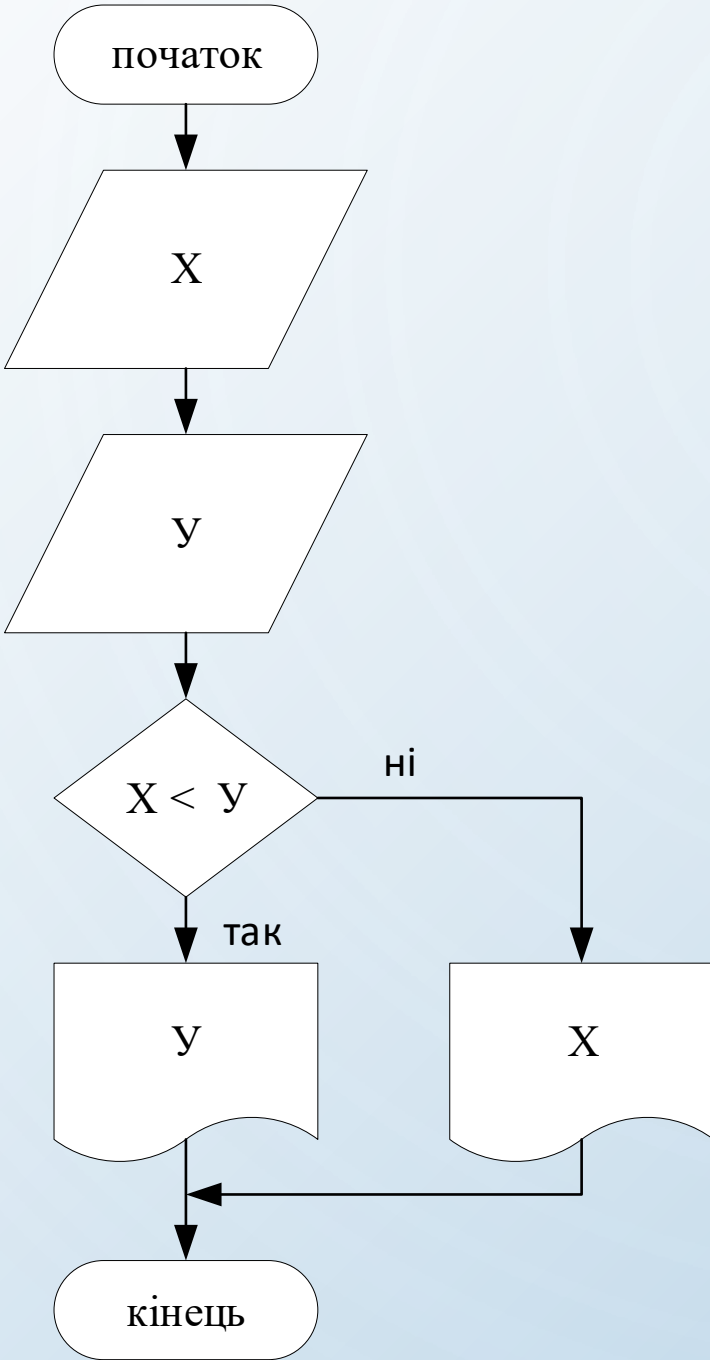


Рисунок 3.1 Блок-схеми прикладів 3.1 (а) та 3.2 (б)

Приклад 3.5 Рішення квадратного рівняння:

$$ax^2 + bx + c = 0$$

Умови розв'язку рівняння:

➤ Якщо $D > 0$, то наявні два дійсні корені:

$$x_1 = \frac{-b - \sqrt{D}}{2a}$$

$$x_2 = \frac{-b + \sqrt{D}}{2a}$$

➤ Якщо $D = 0$, то існує один єдиний корінь.

➤ Якщо $D < 0$, то дійсні рівняння відсутні

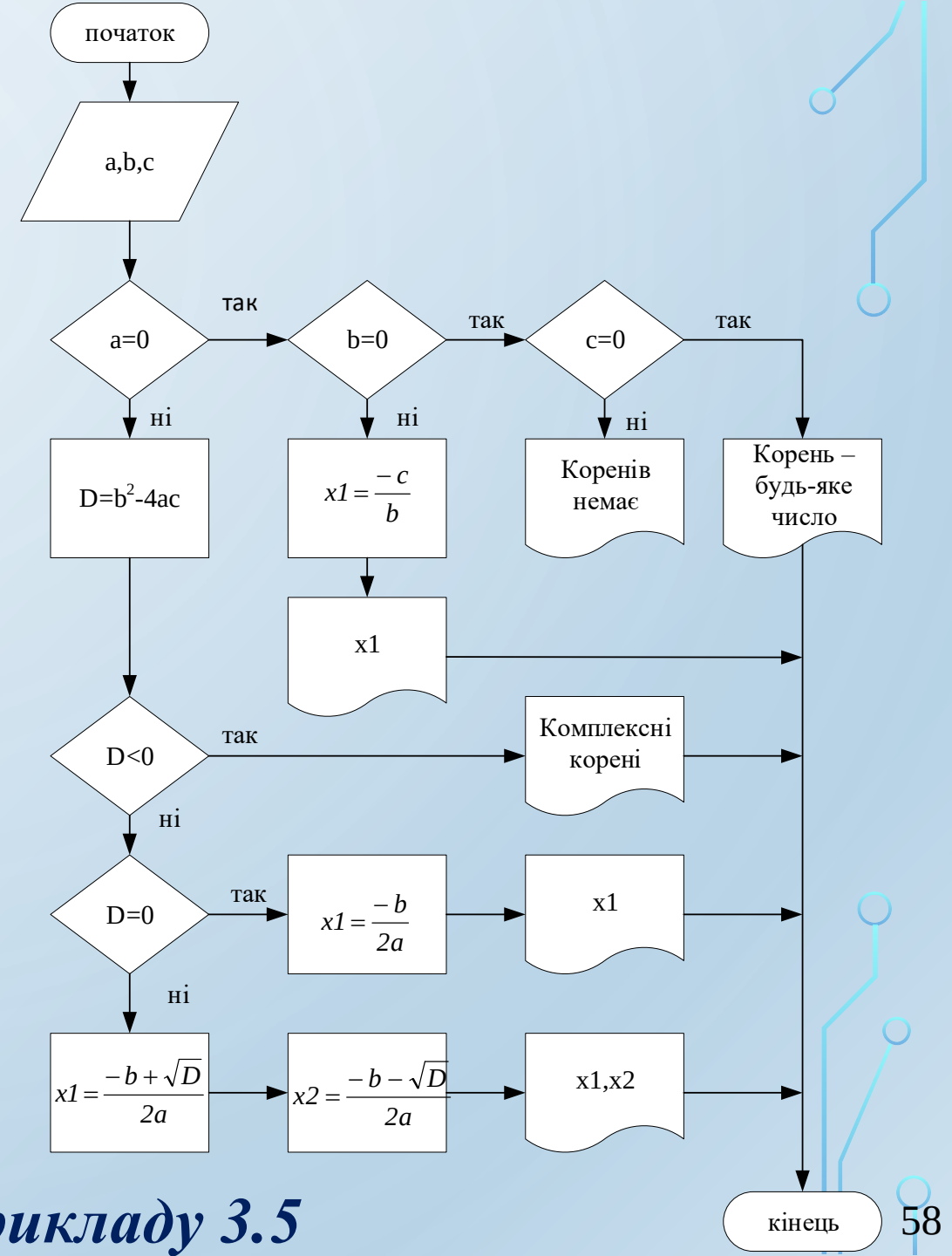
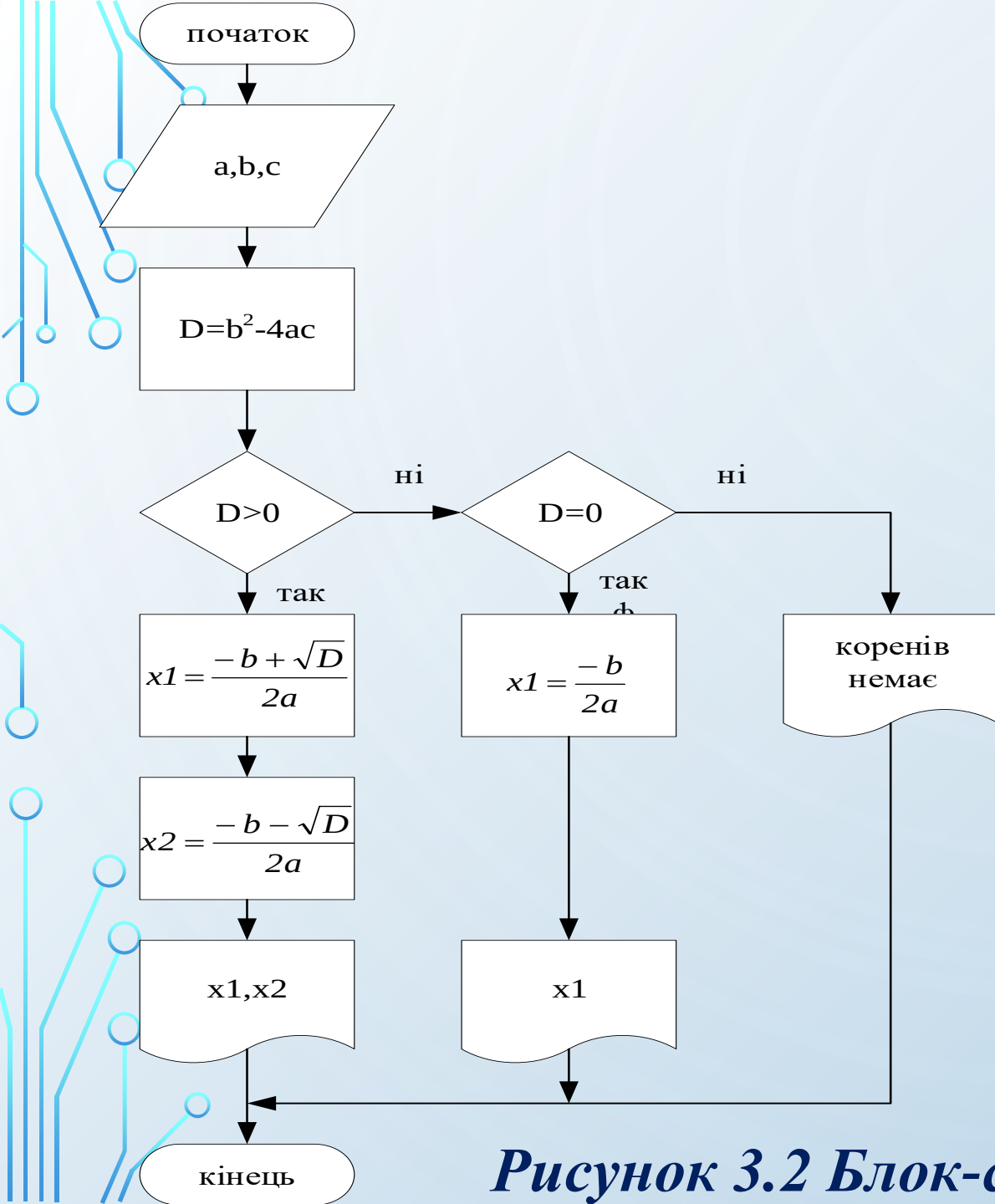


Рисунок 3.2 Блок-схеми прикладу 3.5

Метод Герона

Приклад 3.6 Складемо алгоритм обчислення квадратного кореня з довільного позитивного дійсного числа x методом Герона і запишемо його природною мовою, а також у вигляді блок-схеми. Метод заснований на багаторазовому застосуванні формули:

Алгоритм методу

$$z_{n+1} = \frac{1}{2} \left(z_n + \frac{x}{z_n} \right), \quad \text{при } z_0 = 1$$

Крок 1. Ввести x .

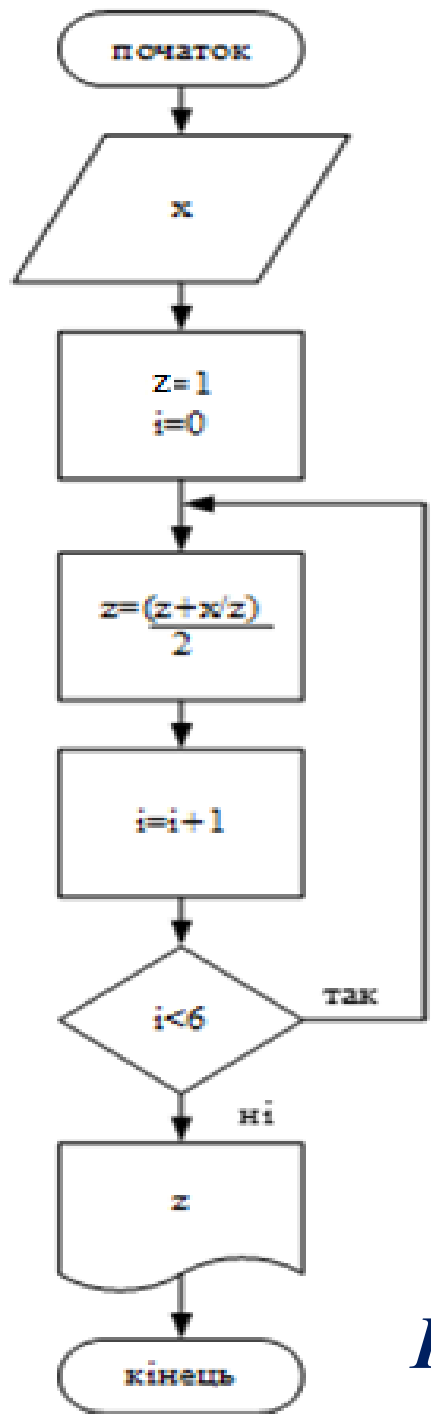
Крок 2. Присвоїти $z = 1$

Крок 3. Присвоїти $i = 0$

Крок 4. Присвоїти $z = \frac{z + \frac{x}{z}}{2}$

Крок 5. Присвоїти $i = i + 1$

Крок 6. Якщо $i < 6$, то перейти до кроку 4, або надрукувати значення z



i	z
0	1,00000
1	1,50000
2	1,41666
3	1,41421
4	1,41421
5	1,41421

Відповідь: 1,414213

Рисунок 3.3

Тема №4. ОСНОВНІ СТРУКТУРИ АЛГОРИТМІВ

Тематичний план

4.1 Визначення основних структур алгоритмів

4.2 Загальні правила виконання алгоритмів

4.1 ВИЗНАЧЕННЯ ОСНОВНИХ СТРУКТУР АЛГОРИТМІВ

Основні структури алгоритмів – це обмежений набір блоків і стандартних способів їх з'єднання для виконання типових послідовностей дій.

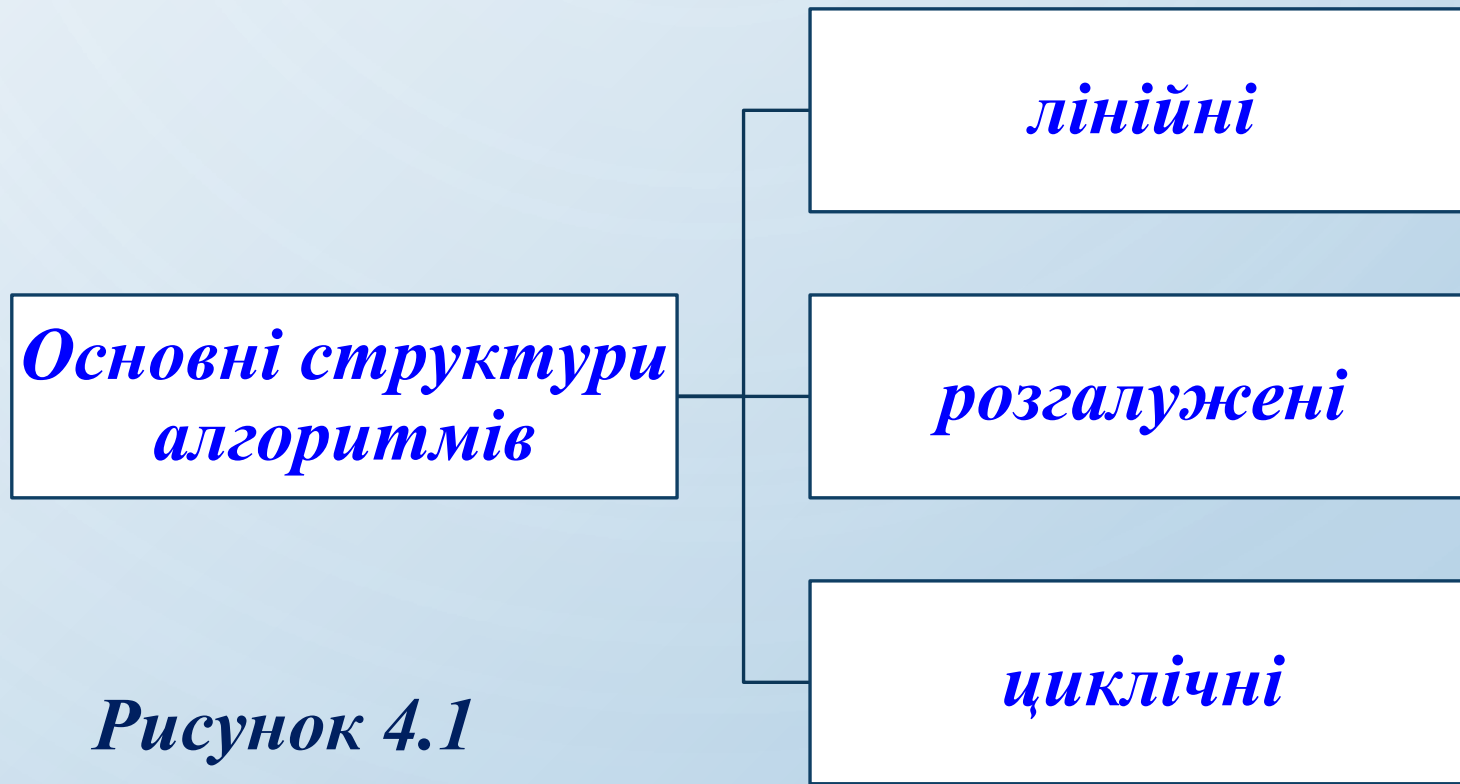


Рисунок 4.1

Лінійна або **послідовна** без будь-яких розгалужень конфігурація алгоритму, що нагадує форму ланцюжка

Приклад 4.1 Розробити блок-схему алгоритму обчислення площі та периметра прямокутника за двома заданими сторонам **a** і **b**.

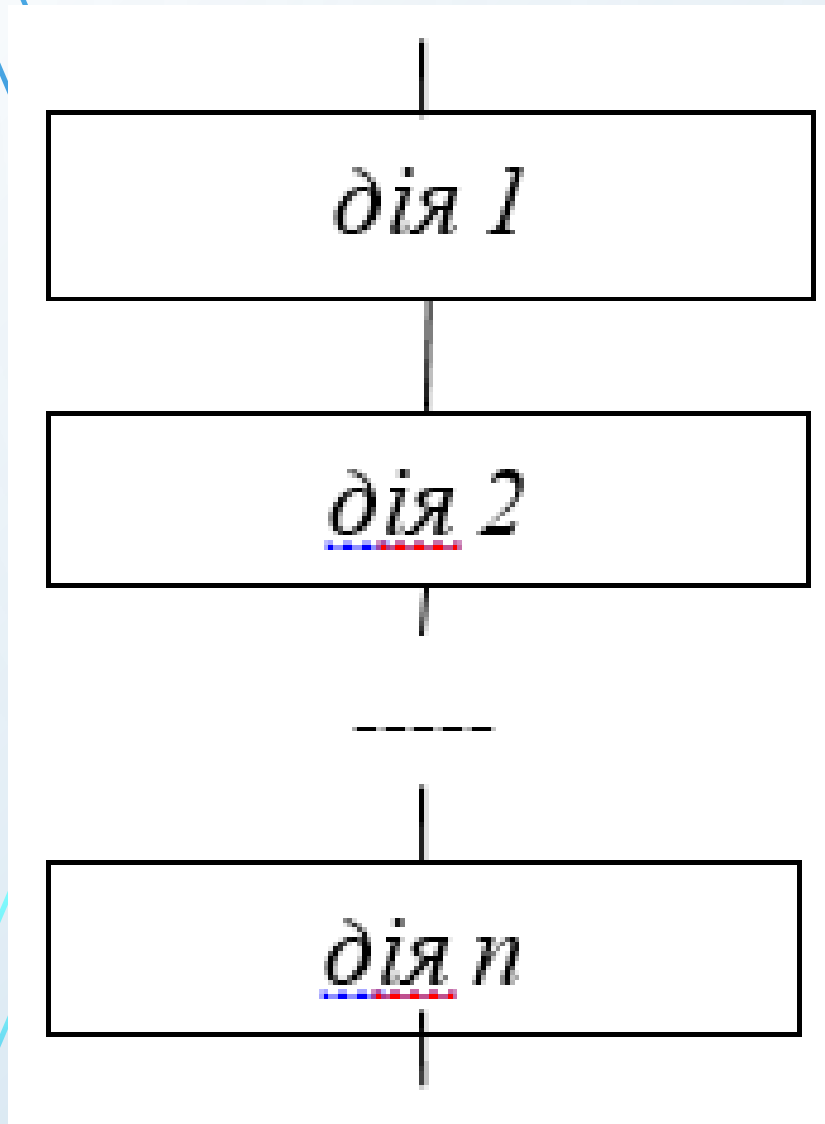


Рисунок 4.2

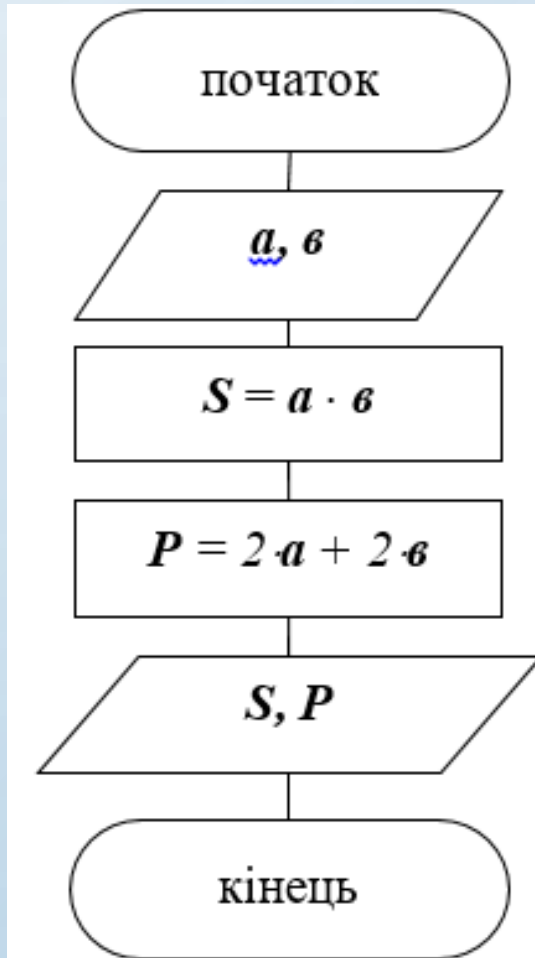


Рисунок 4.3

Розгалужена конфігурація алгоритму, що містить в собі як послідовності, так і розпаралелення послідовностей

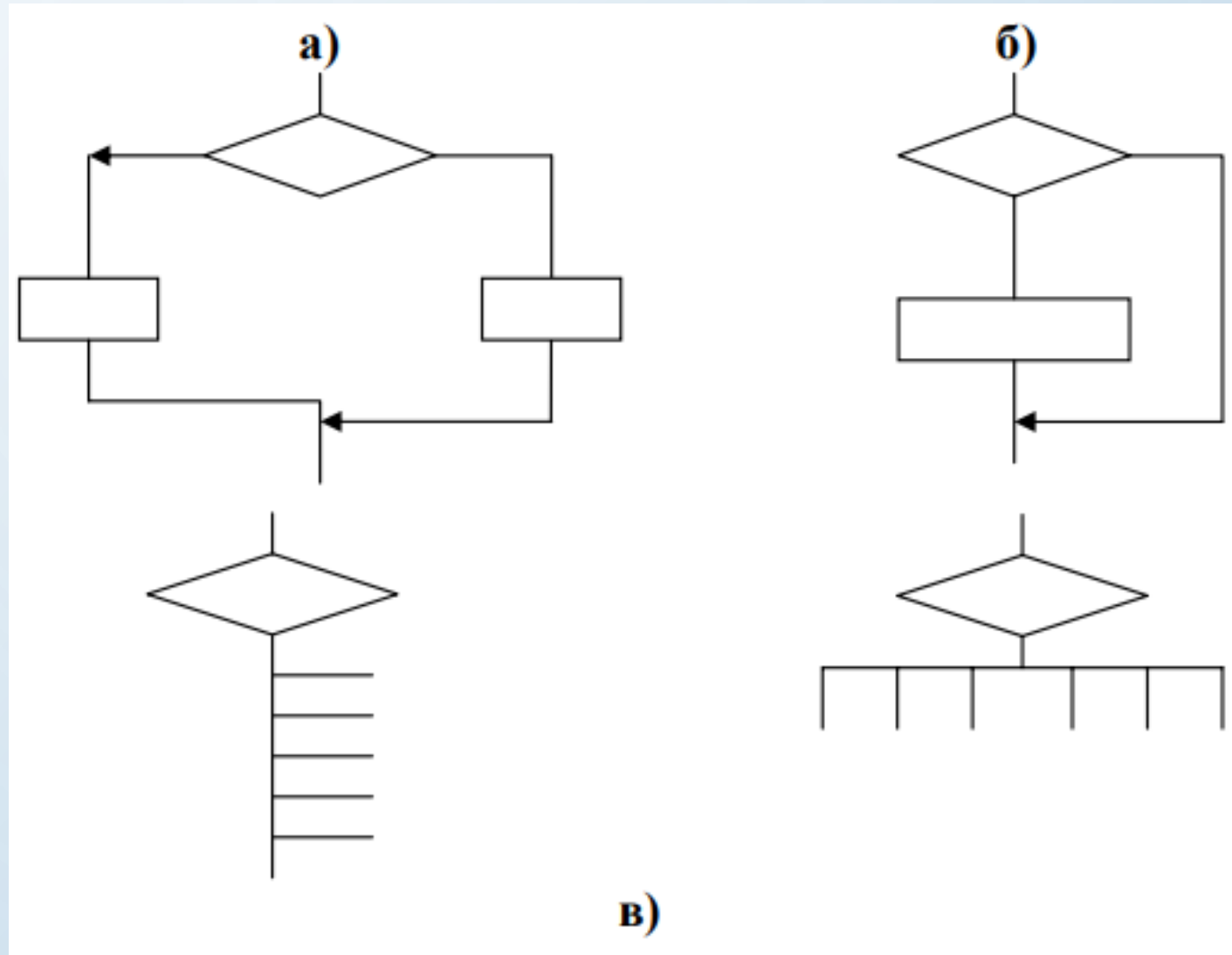
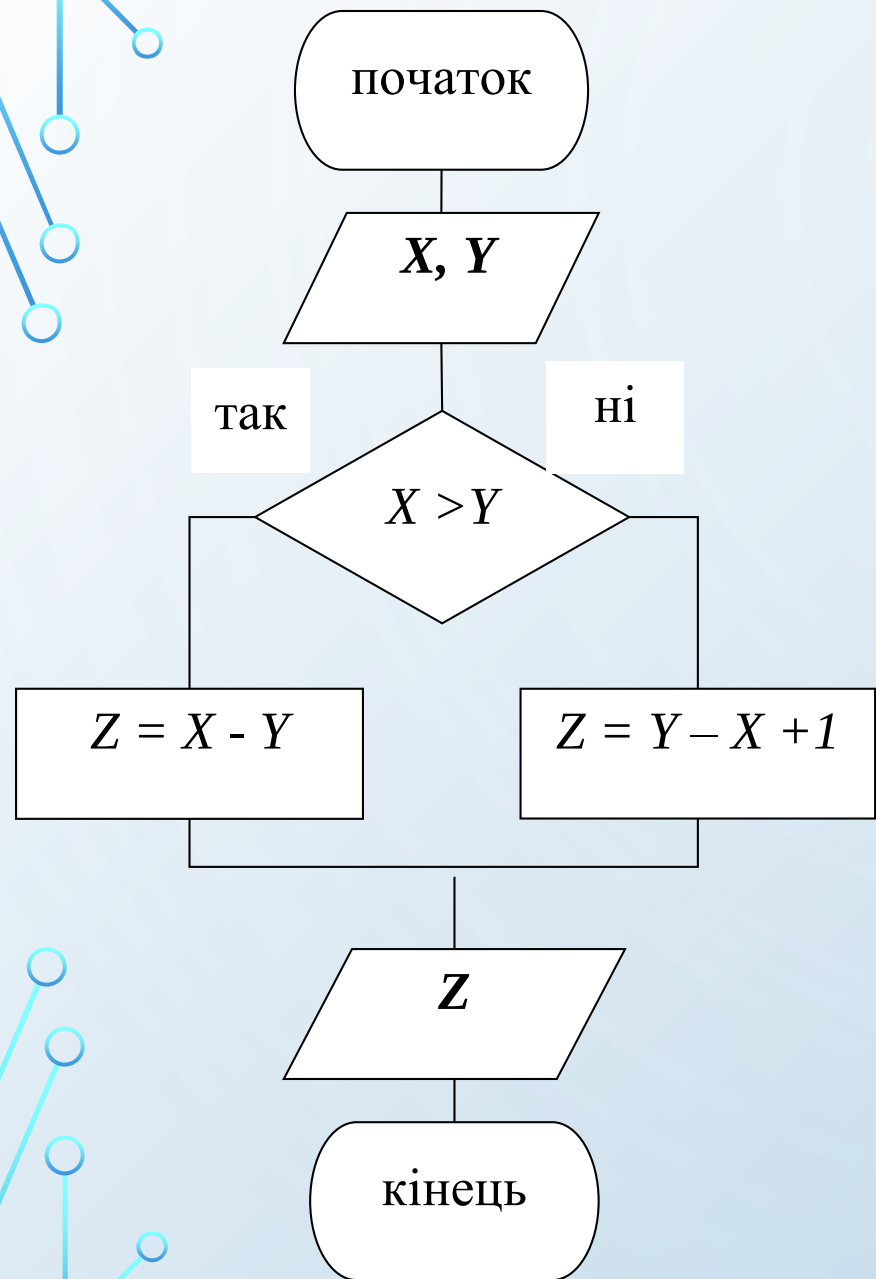


Рисунок 4.4



Приклад 4.2 Розробити блок-схему алгоритму обчислення **Z**, якщо дані два дійсних числа **x** і **y**.

$$Z = \begin{cases} X - Y, \text{ якщо } X > Y \\ Y - X + 1, \text{ якщо } X \leq Y \end{cases}$$

Рисунок 4.5

Циклічним називається алгоритм, в якому деяка частина операцій виконується багаторазово.

Цикл – послідовність дій, що виконуються багаторазово, кожен раз при нових значеннях параметра.

Для **організації** циклу необхідно:

1. задати перед циклом початкове значення змінної, що змінюється в циклі;
2. змінювати змінну перед кожним новим повторенням циклу;
3. перевіряти умову завершення або повторення циклу;
4. управляти циклом, тобто переходити до його початку, якщо він незакінчений, або виходити з нього після закінчення.

Залежно від способу організації числа повторень розрізняють **3 типи циклів**:

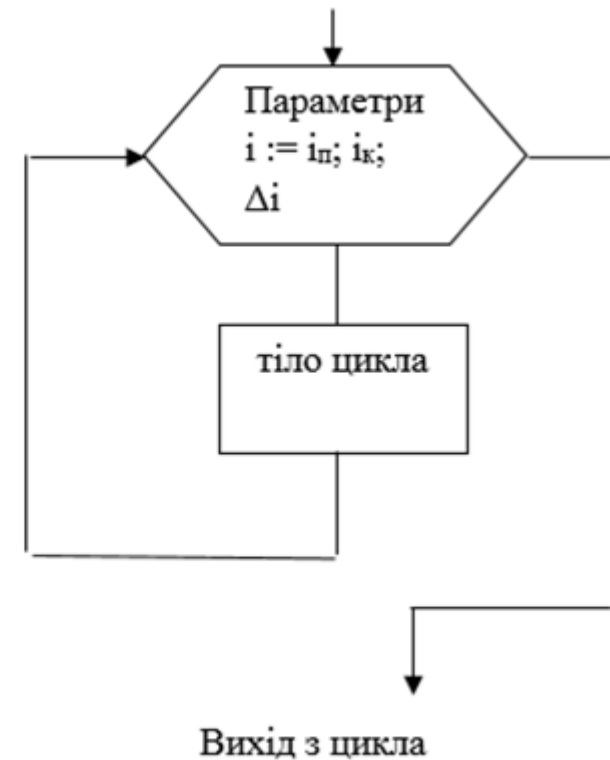
- цикл з передумовою (рис. 4.6, а);
- цикл з наступною умовою (рис. 4.6, б)
- цикл з параметром (рис. 4.6, в).



а

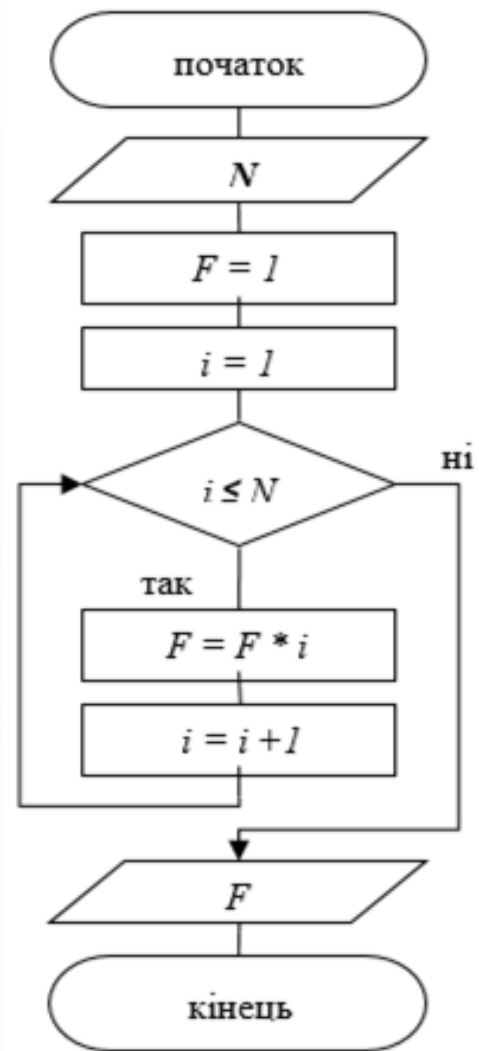


б

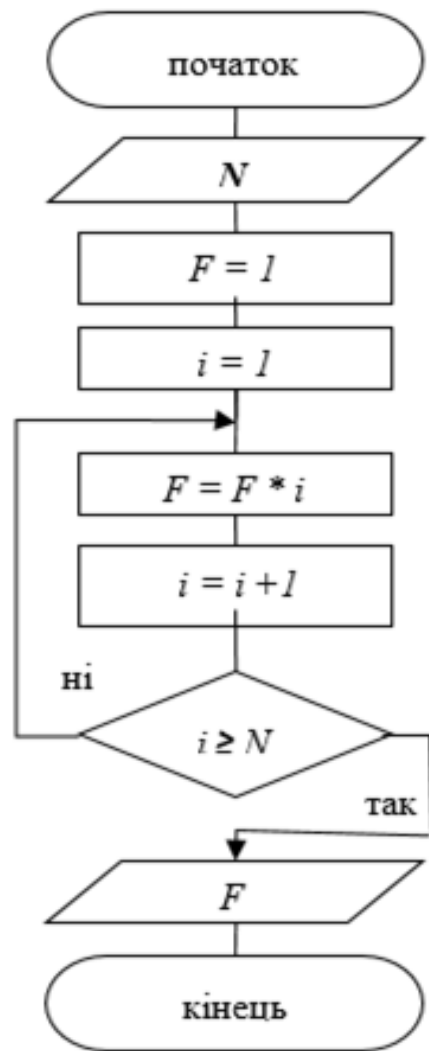


в

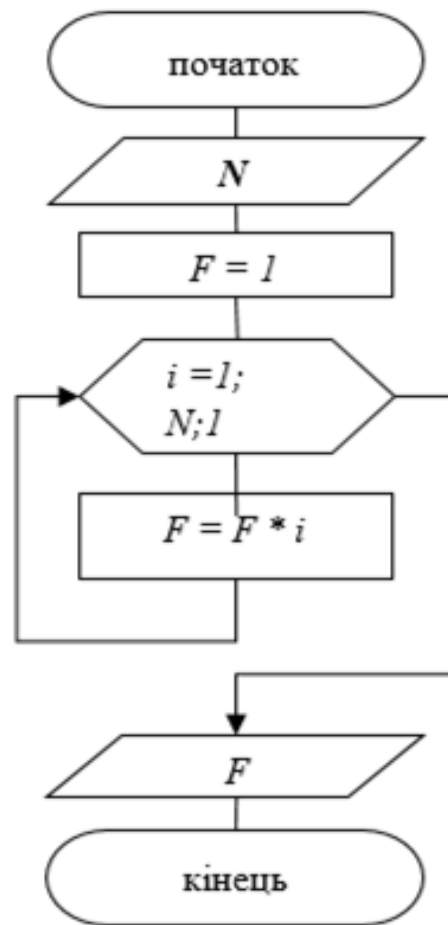
Рисунок 4.6



а) цикл с
передумовою



б) цикл з
наступною
умовою



в) цикл з
параметром

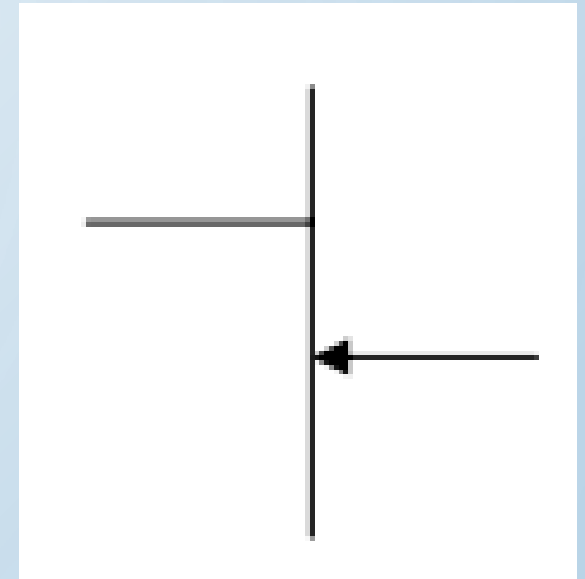
Приклад 4.3 Розробити блок-схему алгоритму обчислення факторіала (F) натурального числа N. Факторіал числа (!) - це множення всіх натуральних чисел від 1 до N.

$$N! = 1 * 2 * 3 * \dots * N, \quad (0! = 1)$$

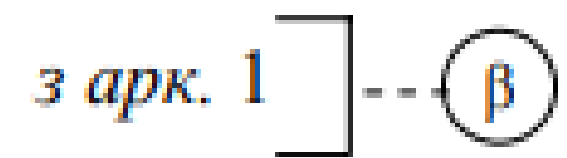
4.2 ЗАГАЛЬНІ ПРАВИЛА ВИКОНАННЯ АЛГОРИТМІВ

Вимоги до побудови схем

Якщо дві чи більше ліній зливаються в одну, то місце з'єднання повинно бути зміщене.

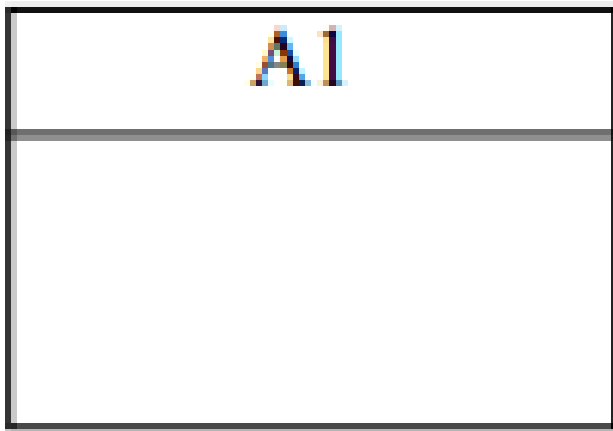


При необхідності розриву лінії (для уникнення зайвих перетинів чи надто довгих ліній) використовують символ “з'єднувач”, який у випадку продовження схеми на кількох аркушах подають разом із символом “коментар”.



У схемах можна використовувати детальне представлення, яке позначається символом процесу або даних з горизонтальною смужкою у верхній частині.

Символ зі смужкою



Детальне представлення

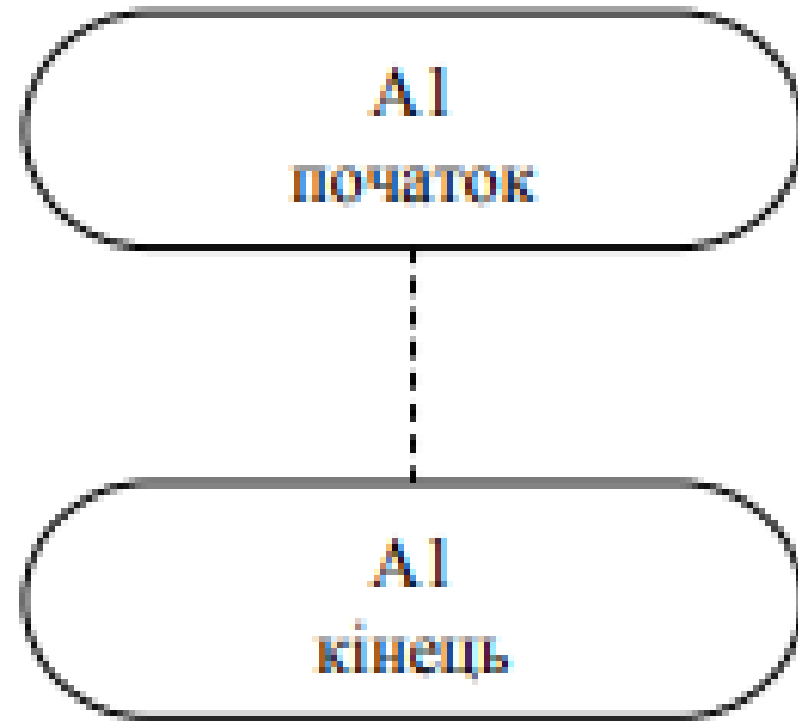


Рисунок 4.8

На схемах для зображення використання або формування кількох носіїв даних чи файлів, множини копій друкованих документів тощо замість одного символа з відповідним текстом можуть бути використані кілька символів з перекриттям зображення, які містять відповідний текст



Рисунок 4.9

Схема даних

Схема даних складається із символів даних, символів процесу, символів ліній та спеціальних символів.

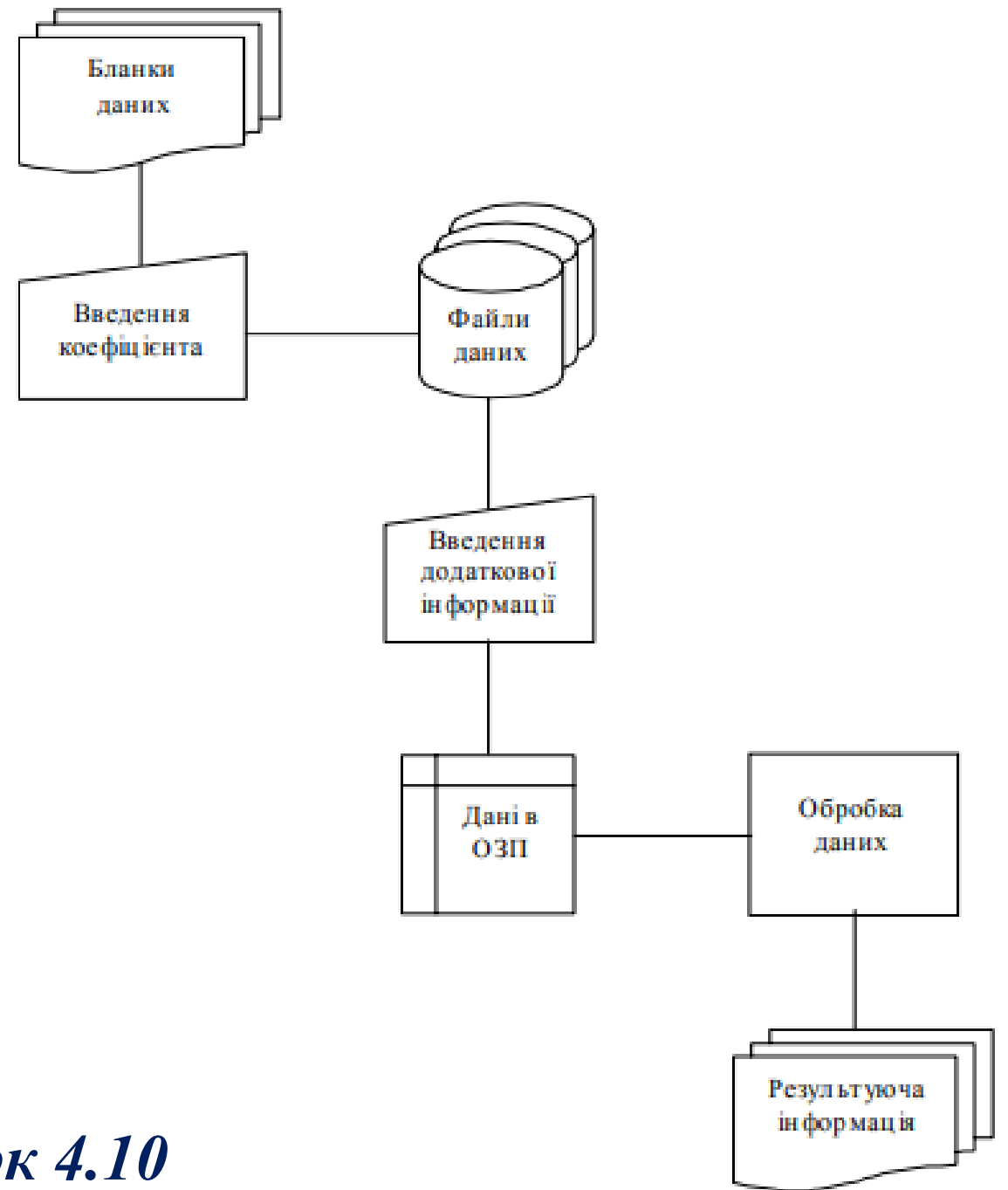


Рисунок 4.10

Схема програми

Схема програми складається із символів процесу, символів ліній та спеціальних символів.

Рисунок 4.11

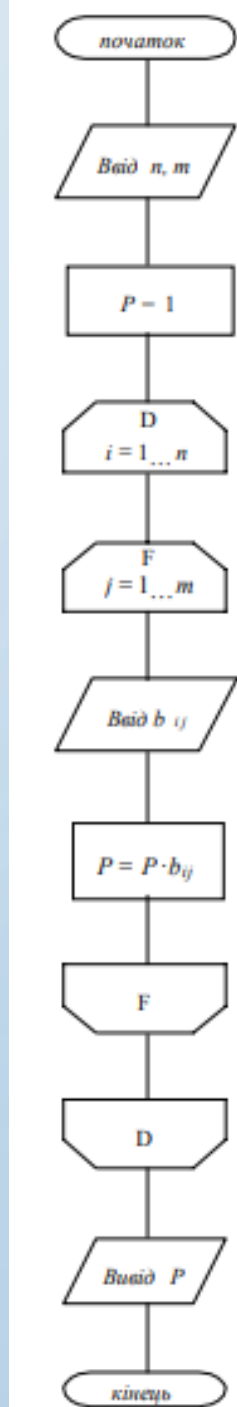
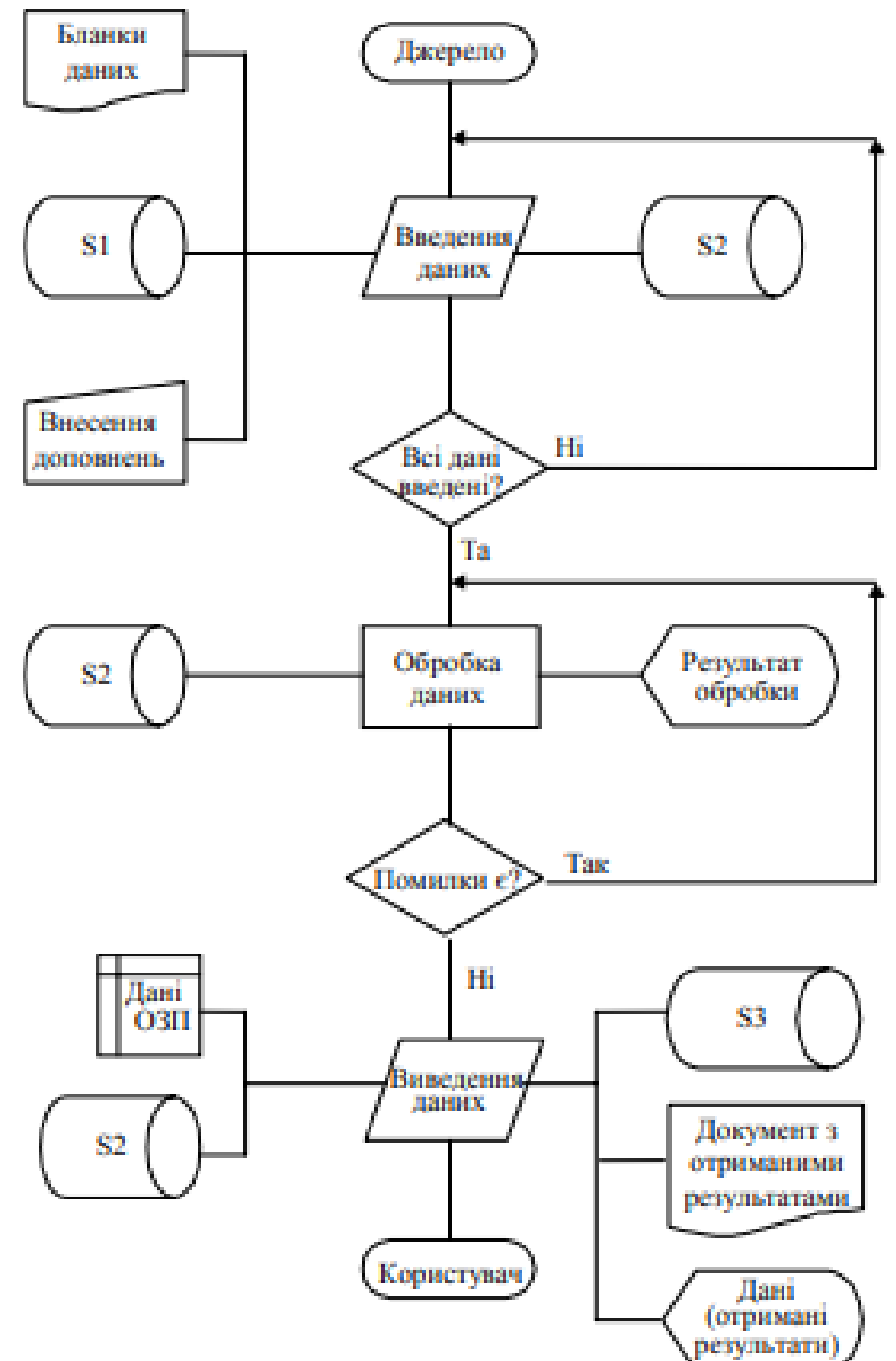


Схема роботи системи

Схеми складаються із символів даних, процесу, ліній та спеціальних символів.

Рисунок 4.12



Схем

програма
відповід



Рисунок 4.13

Тема № 5. МОВИ І СИСТЕМИ ПРОГРАМУВАННЯ. МЕТОДИ ТА ПРИНЦИПИ ПРОГРАМУВАННЯ

Тематичний план

- 5.1 Загальні визначення та поняття мови програмування.
Класифікація мов програмування*
- 5.2 Елементи мови програмування*
- 5.3 Системи програмування*
- 5.4 Методи програмування*
- 5.5 Види програмного забезпечення. Загальні принципи розробки
програмного забезпечення*
- 5.6 Життєвий цикл програмного забезпечення*

5.1 ЗАГАЛЬНІ ВИЗНАЧЕННЯ ТА ПОНЯТТЯ МОВИ ПРОГРАМУВАННЯ. КЛАСИФІКАЦІЯ МОВ ПРОГРАМУВАННЯ

Мова програмування (англ. Programming language) – це штучна мова, створена для передачі команд машинам, зокрема комп'ютерам.

Мова програмування – це система позначень для опису алгоритмів та структур даних, певна штучна формальна система, засобами якої можна виражати алгоритми.

Мову програмування визначає набір лексичних, синтаксичних і семантичних правил, що задають зовнішній вигляд програми і дії, які виконує виконавець (комп'ютер) під її управлінням

Програма - впорядкований список команд для вирішення деякої задачі

Таблиця 5.1 – Класифікація мов програмування та їх характеристика

Покоління	Мови програмування	Характеристика
I	Машинні	Набір машинних команд в двійковім або вісімковому форматі, який визначається архітектурою конкретної ЕОМ
II	Процедурні низького рівня (Асемблер)	Машинно-залежні мови, що використовують символні позначення
III	Процедурні високого рівня (Fortran, Pascal, Basic, C)	Машинно-незалежні мобільні мови, орієнтовані на структурне програмування
IV	Непроцедурні мови (Prolog)	Мінімальна кількість синтаксичних правил, що використовуються для задач штучного інтелекту, орієнтовані на непрофесійного користувача
	Об'єктно-орієнтовані мови (C++, Delphi, Visual Basic)	Засновані на понятті програмного об'єкта
	Мови запитів SQL	Отримання інформації з баз даних
	Мови паралельного програмування (FP)	Орієнтовані на створення програмного забезпечення ЕОМ з паралельною архітектурою
V	Мови штучного інтелекту, експертних систем, баз знань (LISP,HAL)	Орієнтовані на підвищення інтелектуального рівня ЕОМ і інтерфейсу з мовами



Рисунок 5.1 Класифікація мов програмування

Процедурна мова – тип комп'ютерної мови програмування, який визначає низку добре структурованих кроків та процедур у контексті програмування для складання програми.

Непроцедурні мови – мови обробки символічної інформації, що описують тільки що, а не як треба зробити. До непроцедурного програмування відносяться **функціональні** і **логічні** мови.

Мови низького рівня (машинно-залежні або машинно-орієнтовані) – мови, які реалізують набір машинних команд, закладених в центральний процесор конкретного комп'ютера

Мови високого рівня (машинно-незалежні) – мови програмування, що імітують природні, що володіють укрупненими командами, орієнтовані «на людину».

Мови надвисокого рівня – це мова з високим рівнем абстракції.

Мови загального призначення (універсальні) – мова, що призначена для написання програмного забезпечення у широкому різноманітті доменів застосувань.

Спеціалізовані мови – для розв’язування завдань певного виду.

Спеціалізовані мови розробника використовують для створення конкретних типів програмного забезпечення.

Спеціалізовані мови користувача зазвичай є частиною професійних середовищ, характеризуються вузькою спрямованістю і розробниками програмного забезпечення не використовуються.

Переклад програм, записаних на будь-якій мові програмування в машинні коди, здійснюється спеціальною службовою програмою, що називається **транслятором**.

Відомі два види трансляторів - **компілятор** і **інтерпретатор**, які відрізняються способом виконання перекладу.

Компілятор читає всю програму цілком, робить її переклад і задає остаточний варіант на машинній мові, який потім завантажується і виконується комп'ютером.

Інтерпретатор перекладає і виконує програму рядок за рядком.

Вибір мови програмування визначається **трьома факторами**:

- мова повинна бути зручною для програміста;
- мова повинна бути придатна для даного комп'ютера;
- мова повинна бути придатна для вирішення даного завдання

5.2 ЕЛЕМЕНТИ МОВИ ПРОГРАМУВАННЯ



Рисунок 5.2

Алфавіт — це сукупність дозволених в мові символів чи груп символів, що розглядаються як єдине ціле.

Константа — це число, символ або строка символів. Константи використовують для визначення постійних величин. В С розрізняють чотири типи констант: *цілі, з плаваючою крапкою, символьні константи й символьні рядки.*

Ідентифікатори — це імена змінних, функцій і міток, які використовують в програмі. Ідентифікатор вводять при визначенні змінної або функції, або як мітка оператора.

Ключові слова — це слова, які в мові програмування мають спеціальне значення. Ключові слова не можна використовувати як ідентифікатори, вони можуть бути використані лише так, як це передбачено мовою програмування.

Список ключових слів:

auto	do	for	return	switch
break	double	goto	short	typedef
case	else	if	signed	union
char	enum	int	sizeof	unsigned
continue	extern	long	static	void
default	float	register	struct	while

Коментарі — це будь-яка послідовність символів, що міститься між парами символів «/*» та «*/»

Недопустимими є вкладені коментарі:

```
/* Коментар /* вкладений коментар не допускається, це — помилка */  
продовження коментарю */
```

```
/* Коментар */
```

```
/* Багаторядковий  
коментар */
```

Оператори – це основні елементи програми. Власне програма складається з послідовності операторів. Оператори керують процесом виконання програми.

Лексема – це одиниця тексту програми, яка має самостійний зміст для компілятора мови С. Лексема не може містити інших лексем.

Лексика - сукупність правил створення ланцюжків символів (лексем), що утворюють ідентифікатори, оператори, операції та інші компоненти мови.

Синтаксис - сукупність правил творення слів і речень мов програмування.

Семантика - смисловий зміст слів і речень мов програмування.

5.3 СИСТЕМИ ПРОГРАМУВАННЯ

Система програмування – це сукупність засобів розробки програм, що забезпечують автоматизацію складання і налагодження програм користувача.

Система програмування містить:

- **текстовий редактор**, який здійснює функції запису і редагування текстів програми;
- **транслятор**, що здійснює переведення вихідного тексту програми в машинні коди з діагностикою синтаксичних і логічних помилок;
- **завантажувач програм**, що дозволяє вибрати з директорії потрібний текстовий файл програми;

- **запускач програм**, здійснює процес виконання програм;
- **диспетчер файлів**, що надає можливість виконувати операції з файлами;
- **відладчик**, що виконує сервісні функції з налагодження й тестування програми;
- **компоновник** (редактор зв'язків), що здійснює об'єднання об'єктних і завантажувальних модулів в єдиний модуль з подальшим записом в файл;
- **бібліотека підпрограм**, тобто організований відповідним чином набір підпрограм, що використовується при вирішенні різних завдань.

Системи програмування класифікують за **3 ознаками:**

1. за набором вихідних мов:

- одномовні;
- багатомовні, в яких окремі частини програм можуть бути підготовлені на різних мовах і об'єднані в єдиний модуль під час або перед виконанням програм.

2. за можливістю розширення:

- замкнуті;
- відкриті, в які можна ввести нову вхідну мову з транслятором, не вимагаючи змін в системі.

3. за способом трансляції:

- компіляція;
- інтерпретація.

Структура багатомовної, відкритої системи програмування, що компілює



Рисунок 5.3

5.4 МЕТОДИ ПРОГРАМУВАННЯ

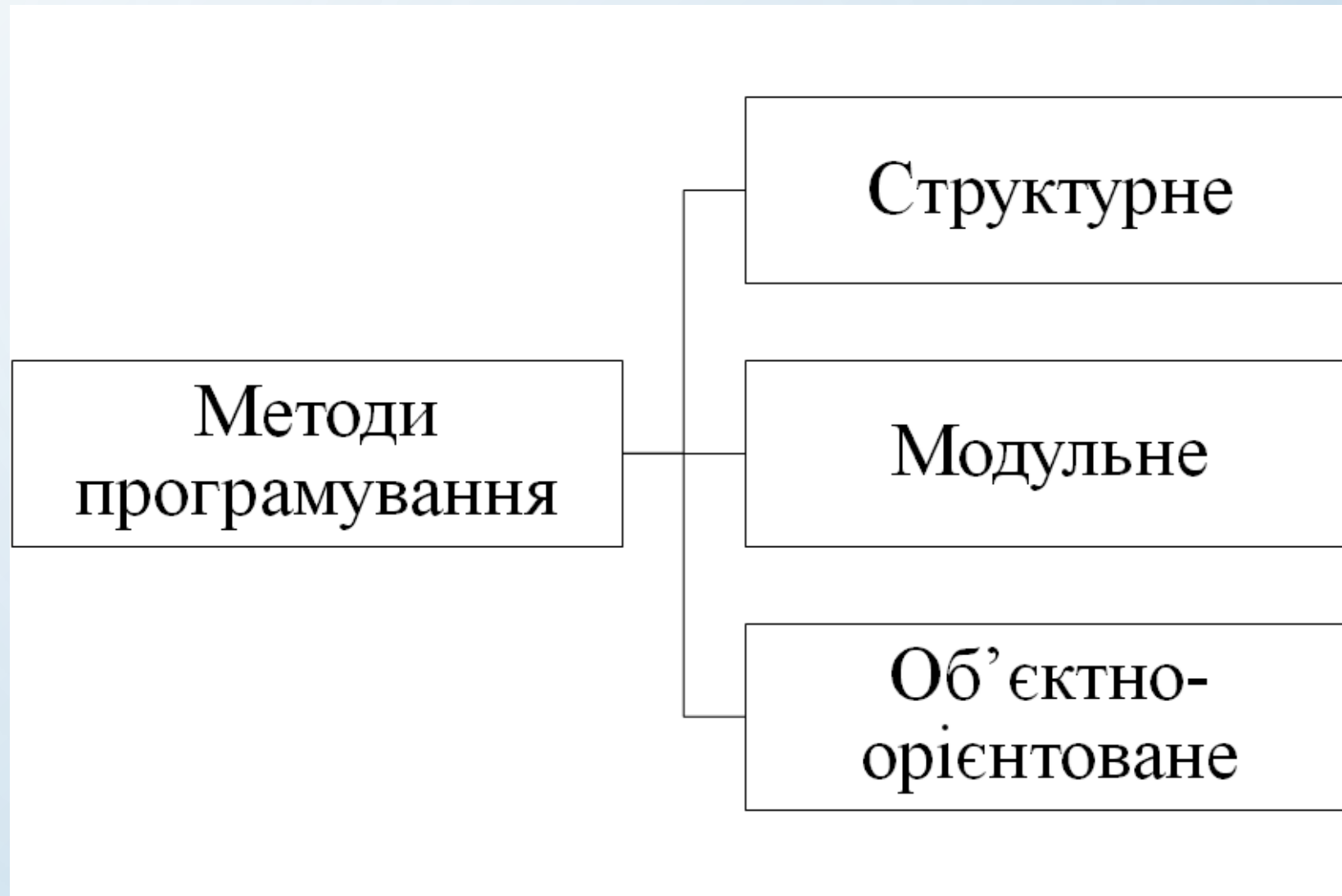


Рисунок 5.4

Структурне програмування - описується кожна дія алгоритму для досягнення кінцевого результату, використовуючи процедурний стиль програмування і послідовну декомпозицію алгоритму розв'язання задачі зверху вниз.

Модульне програмування - проектування нової програмної системи на базі розроблених і налагоджених раніше модулів.

Об'єктно-орієнтоване програмування - описує програмні системи у вигляді взаємодії об'єктів.

Переваги об'єктно-орієнтоване програмування:

- спрощення процесу проектування програмних систем,
- легкість їх супроводу і модифікації,
- мінімізація часу розробки за рахунок багаторазового використання готових модулів.

Компоненти при ООП бувають **візуальні** та **невізуальні**.

5.5 ВИДИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

ЗАГАЛЬНІ ПРИНЦИПИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Програмне забезпечення (ПЗ) — це набір інструкцій, даних або програм, що використовуються для роботи з комп'ютером з метою виконання конкретних завдань.



Рисунок 5.5

Загальні принципи при розробці ПЗ:

- **частотний принцип** - виділення дій і даних за частотою використання;
- **принцип модульності** - відокремлення складових частин ПЗ в окремі модулі за функціональною ознакою;
- **принцип функціональної вибірковості** - при проектуванні програмного забезпечення, обсяг якого перевищує обсяг оперативної пам'яті, частина програмного забезпечення, яка постійно використовується, зберігається в оперативній пам'яті, інша частина - на зовнішніх запам'ятовуючих пристроях;
- **принцип генерування** - здійснення налаштування на конкретну конфігурацію технічних засобів, коло вирішуваних проблем, умови роботи користувача;
- **принцип функціональної надмірності** - можливість проведення однієї і тієї ж роботи (функції) різними засобами;
- **принцип «за замовчуванням»** - зберігання в системі базових описів структур, модулів, конфігурацій обладнання та даних, які визначають умови роботи з ПЗ.

Загальносистемні принципи при створенні та розвитку ПЗ:

- **принцип включення** - вимоги до створення та функціонування ПЗ визначаються системою, що включає його в себе;
- **принцип системної єдності** - забезпечення цілісності між підсистемами;
- **принцип інформаційної єдності** - використання єдиних термінів, символів, умовних позначень і способів подання;
- **принцип розвитку** - можливість нарощування і вдосконалення компонентів ПЗ і зв'язків між ними;
- **принцип комплексності** - забезпечення зв'язності обробки інформації;
- **принцип сумісності** - забезпечення спільного функціонування всіх підсистем ПЗ;
- **принцип інваріантності** - підсистеми і компоненти ПЗ універсальні по відношенню до оброблюваної інформації.

5.6 ЖИТТЄВИЙ ЦИКЛ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

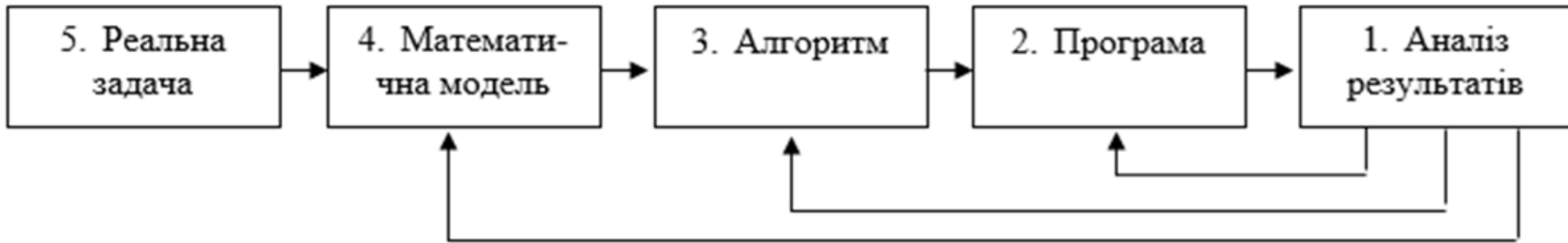


Рисунок 5.6 Технологічний ланцюжок рішення задач на ЕОМ

Життєвий цикл ПЗ - це процес від початку створення ПО до повного вилучення з експлуатації.

Життєвий цикл ПЗ складається зі стадій: **визначення вимог і специфікацій; проєктування ПЗ; програмування; налагодження ПЗ та супровід ПЗ**

Визначення вимог і специфікацій - встановлюються загальні вимоги щодо надійності, технологічності, універсальності, ефективності, інформаційної узгодженості, розробляються вхідні та проміжні мови, форми вихідної інформації.

Проектування ПЗ - формується структура ПО, розробляються алгоритми, встановлюється склад модулів і інтерфейси.

Програмування - проектні рішення реалізуються у вигляді програм.

Налагодження ПЗ - перевірка виконання всіх вимог, всіх структурних елементів системи, працездатності ПЗ, виявлення та виправлення помилок.

Супровід ПЗ - процес координації всіх елементів системи відповідно до вимог користувача, внесення всіх необхідних йому виправлень і змін, подальше вдосконалення системи під час експлуатації.

Тема №6. БАЗОВІ ЕЛЕМЕНТИ МОВИ ПРОГРАМУВАННЯ C/C++

Тематичний план

- 6.1 Загальні відомості щодо мови програмування C/C++*
- 6.2 Етапи написання ПЗ на C/C++*
- 6.3 Базова структура програми на C/C++*
- 6.4 Загальні поняття щодо бібліотечних функцій та заголовних файлів*
- 6.5 Визначення основних характерних рис мови програмування C/C++*

6.1 ЗАГАЛЬНІ ВІДОМОСТІ ЩОДО МОВИ ПРОГРАМУВАННЯ C/C++

1972-1974 рр. – фірма **Bell Laboratories** Б. Керніган, Д. Рітчі. ОС Unix.

Компілятори мови Cі працюють в **операційних систем** UNIX, MS-DOS, Mac OS, OS / 2, Windows, Windows NT, Solaris і ін.

Стандарти: ANSI C, ISO C (International Standard Organization C)

Файл з розширенням - **.c** або **.cpp**

Переваги

- **багатоплатформність** – для всіх використовуваних в даний час платформ існують компілятори з мови C і C ++ (став **основою багатьох сучасних мов** (C++, C#, Javascript, Java, ActionScript, PHP));
- **наявність операторів**, що реалізують основні структурні алгоритмічні конструкції (умовну обробку, всі види циклів);
- можливість **програмування на низькому** (системному) **рівні** з використанням адрес оперативної пам'яті;
- **величезні бібліотеки підпрограм і класів**;
- **висока швидкість** роботи програм;

Недоліки

- висока вірогідність зробити помилку, що не виявиться автоматично.

6.2 ЕТАПИ НАПИСАННЯ ПЗ НА C/C++



Рисунок 6.1

6.3 БАЗОВА СТРУКТУРА ПРОГРАМИ НА C/C++

```
# Директиви препроцесора
int main() // початок програми
{ // початок функції
.....;
.....; // інструкції (оператори, функції)
.....;
return 0;
} //кінець функції і кінець програми
```

```
#include <stdio.h>

int main()
{
printf ("Hello, world! \n"); /*
виведення на екран */
getch(); /* чекати натискання
клавiши */
return 0;
}
```

Директиви препроцесора

Препроцесор – це спеціальна програма, що опрацьовує текстовий файл програми на Сі перед компіляцією. Він входить як обов'язковий компонент до складу компілятора. Результатом роботи препроцесора є готовий до компіляції текст програми, в якому реалізовані всі задані директиви (інструкції) попереднього опрацювання програми.

Дії при виконанні препроцесорної обробки:

- включення до складу програми текстів заданих файлів;
- вилучення з тексту програми тих частин, які не повинні компілюватись у даній реалізації (умовна компіляція);
- заміна в програмі макроімен на задані вирази підстановки;
- реалізація макropідстановок (макросів) з параметрами.

6.4 ЗАГАЛЬНІ ПОНЯТТЯ ЩОДО БІБЛІОТЕЧНИХ ФУНКЦІЙ ТА ЗАГОЛОВНИХ ФАЙЛІВ

Директива `#include`

Форми запису:

- `#include < ім'я_файлу >` // ім'я в кутових дужках
- `#include "ім'я_файлу"` // ім'я в лапках

Таблиця 6.1 – Заголовні файли та їх призначення

Заголовний файл	Призначення
<code>assert.h</code>	Діагностика програми
<code>ctype.h</code>	Перетворення і перевірка символів
<code>conio.h</code>	Перевірка помилок

Заголовний файл	Призначення
float.h	Робота з числами з плаваючою комою
limits.h	Визначення розмірів цілочисельних типів
locale.h	Підтримка інтернаціональної середовища
math.h	Математичні бібліотеки
setjmp.h	Можливості нелокальних переходів
signal.h	Відпрацювання сигналів
stdarg.h	Підтримка функцій з невизначеним числом аргументів
stddef.h	Різне
stdio.h	Бібліотека стандартів введення / виведення
stdlib.h	Бібліотека загального призначення
string.h	Функції роботи з рядками символів
time.h	Функції роботи з датами і часом
dos.h	Підключення Dos

6.5 ВИЗНАЧЕННЯ ОСНОВНИХ ХАРАКТЕРНИХ РИС МОВИ ПРОГРАМУВАННЯ C/C++

«Сі — інструмент, гострий, як бритва: за його допомогою можна створити і елегантну програму, і кроваве м'ясо»

(Brian W. Kernighan)

Складовими частинами мови є:

- *символи (алфавіт мови)* - це основні неподільні знаки, за допомогою яких пишуться всі тексти на мові програмування;
- *лексеми (слова)* - мінімальні одиниці мови, які мають самостійний зміст;
- *вирази* - задають правило обчислення деякого значення;
- *оператори (інструкції)* задають опис деякої дії.

Алфавіт мови

- великі та малі латинські літери: A-Z, a-z.
- арабські цифри;
- символи: графічні та ескейп-послідовності (символи табуляції, символ переходу на наступний рядок тощо);
- символи , . ; : ? ' ! | / \ ~ () [] { } < > # % ^ & - + * =

Лексеми — це слова мови. До лексем відносять ключові слова, ідентифікатори та константи.

Ключові слова

<i>auto</i>	<i>double</i>	<i>int</i>	<i>struct</i>
<i>break</i>	<i>else</i>	<i>long</i>	<i>switch</i>
<i>case</i>	<i>enum</i>	<i>register</i>	<i>typedef</i>
<i>char</i>	<i>extern</i>	<i>return</i>	<i>union</i>
<i>const</i>	<i>float</i>	<i>short</i>	<i>unsigned</i>
<i>continue</i>	<i>for</i>	<i>signed</i>	<i>void</i>
<i>default</i>	<i>goto</i>	<i>sizeof</i>	<i>volatile</i>
<i>do</i>	<i>if</i>	<i>static</i>	<i>while</i>

Константи

1. Константи цілочислові

Десяткові: 12345-17 +1098

Вісімкові: 0110 077 01234567

Шістнадцяткові: 0x1230xF3a2E

2. Дійсні константи

З фіксованою крапкою: 1.2345 -0.017 +1098.2345

З плаваючою крапкою: 123.45e-2 1.0982345E3

3. Символьні константи

Прості символи: 'A' 'b' 'щ' '[' '\'

Ескейп-послідовності '\n' '\a' '\"' '\t'

4. Рядки символів

"Привіт першокурснику!\n"

"Ми вивчаємо курс \"Програмування на C\"!"

5. Знаки операцій, роздільники, коментарі

/* Односимвольні операції:*/

+ * < ^ / ...

// Багатосимвольні операції:

++ || >> += <= <<=

// Роздільники:

() [] { } , ; : #

Таблиця 6.2 – Операції введення-виведення		
Ім'я функції	Опис	Файл, що містить прототип
printf()	Здійснює форматований вивід даних в stdout	stdio.h
puts()	Здійснює виведення рядка символів в stdout	stdio.h
putchar()	Здійснює виведення символу в stdout	stdio.h
cprintf()	Здійснює форматований вивід на екран	conio.h
cputs()	Здійснює виведення рядка на екран	conio.h
putch()	Здійснює виведення символу на екран	conio.h

Функції виведення

printf("РядокФорматів", об'єкт 1, об'єкт 2, ..., об'єкт n);

Рядок Форматів складається з наступних елементів:

- керуючих символів;
- тексту, представленого для безпосереднього виведення;
- форматів, призначених для виведення значень змінних різних типів.

Як **об'єкт** виведення використовуються константи, змінні, висловлюння, покажчики функцій. Об'єкти можуть бути відсутні.

Основні *керуючі символи*:

'\n' - новий рядок;

'\t' - горизонтальна табуляція;

'\v' - вертикальна табуляція;

'\b' - повернення на символ;

'\r' - повернення на початок
рядка;

'\a' - звуковий сигнал.

Формат (Специфі- катор)

Типи вихідної інформації

%d

Десяткове ціле число

%i

Десяткове ціле число зі знаком

%c

Символ

%s

Рядок символів

%f

Число з плаваючою точкою

%u

Десяткове ціле число без знака

%ld

Длине ціле

%p

Ціле покажчика

%o

Вісімкове ціле без знака

%x

Шістнадцяткове ціле без знака

% -10s; % 6d; % 8.3f; % ld.

Приклад 6.1

```
int y; // оголошення цілочисельної змінної
```

```
int x = 5; // оголошення та ініціалізація змінної
```

```
...
```

```
y = x + 20; // операція присвоювання
```

```
printf ( "отримано число %d \n", y); // виведення числа
```

```
printf ( "отримано число %d \n", x + 20); // виведення значення виразу
```

```
printf ( "отримано числа %d%d \n", x, y); // виведення двох об'єктів
```

Функції виведення *puts ()* і *cputs ()*

Приклад 6.2 Вивести "Hello,world!"

```
# include <stdio.h> //підключення функцій введення/виведення
# include <conio.h >
void main ()      // головна функція
{
    printf("Hello,world! \n"); // друк рядка
    puts("Hello,world! ");     // друк рядка
    cputs("Hello,world! \n\r"); // друк рядка
}
```

Таблиця 6.3 – Функції введення

Ім'я функції	Опис	Файл, що містить прототип
Scanf()	Виконує форматний введення з потоку stdin	stdio.h
Gets()	Отримує рядок символів з потоку stdin	stdio.h
Getchar()	Вводить символ з потоку stdin	stdio.h
Cscanf()	Виконує форматний введення з клавіатури	conio.h
Cgets()	Зчитує рядок символів з клавіатури	conio.h
Getch()	Вводить символ з клавіатури без луна-друку	conio.h

Функції *scanf()* і *cscanf()*

(c)scanf ("РядокФорматів", адреса1, адреса2,...);

РядокФорматів: % d, % f, % c, % s
адреса = & об'єкт

Приклад 6.3

```
/ * Введення двох цілих чисел в  
осередки пам'яті " a " і " b " * /  
(c)scanf ( "% d% d", & a, & b);  
де &a, &b- адреси операндів " a " і  
" b ".
```

Приклад 6.4

```
/ * Введення рядка символів, представлених  
масивом * /  
#include <conio.h>  
void main ()  
{  
    clrscr ();  
    char im [10];  
    cprintf ( "Введіть ім'я:");  
    cscanf ( "% s", im);  
    cprintf ( "\ n \ r Ваше ім'я:% s \ n \ r", im);  
}
```

Функції *gets()* і *cgets()*

'\0' - термінатор

Приклад 6.5

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

void main ()
{
    char string [40];
    printf ("Введіть рядок : ");
    gets (string);
    printf ("Рядок = %s \n", string);
    getch ( );
}
```


Функції `getchar()` і `getch()`

Приклад 6.6

```
#include <stdio.h>

int main (void)
{
    char c;
    while ((c=getchar())!='\n')
        printf ("%c", c);
    return 0;
}
```

Приклад 6.7

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int I;
    int J;
    int C;
    printf(«Введіть число у метрах: \n");
    scanf("%d",&I);
    J=100*I;
    printf(" В %d метрах міститься %d\n", I,J);
    C=getch();
}
```

Тема №7. ТИПИ ДАНИХ

Тематичний план

- 7.1 Загальна класифікація типів даних та операції над ними*
- 7.2 Прості структури даних*
- 7.3 Статичні структури даних*
- 7.4 Напівстатичні структури даних*
- 7.5 Динамічні структури даних*
- 7.6 Нелінійні структури даних*

7.1 ЗАГАЛЬНА КЛАСИФІКАЦІЯ ТИПІВ ДАНИХ ТА ОПЕРАЦІЇ НАД НИМИ

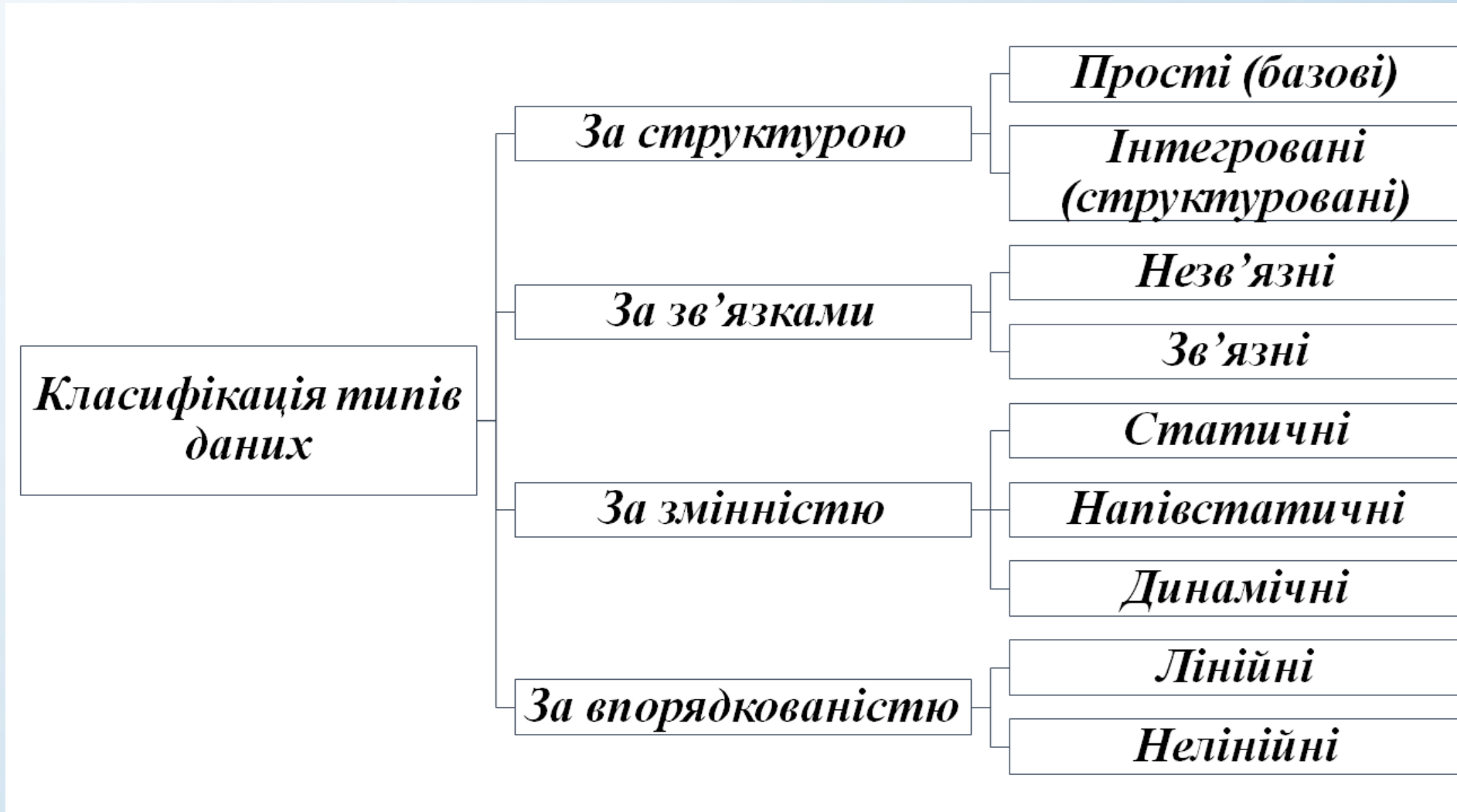


Рисунок 7.1

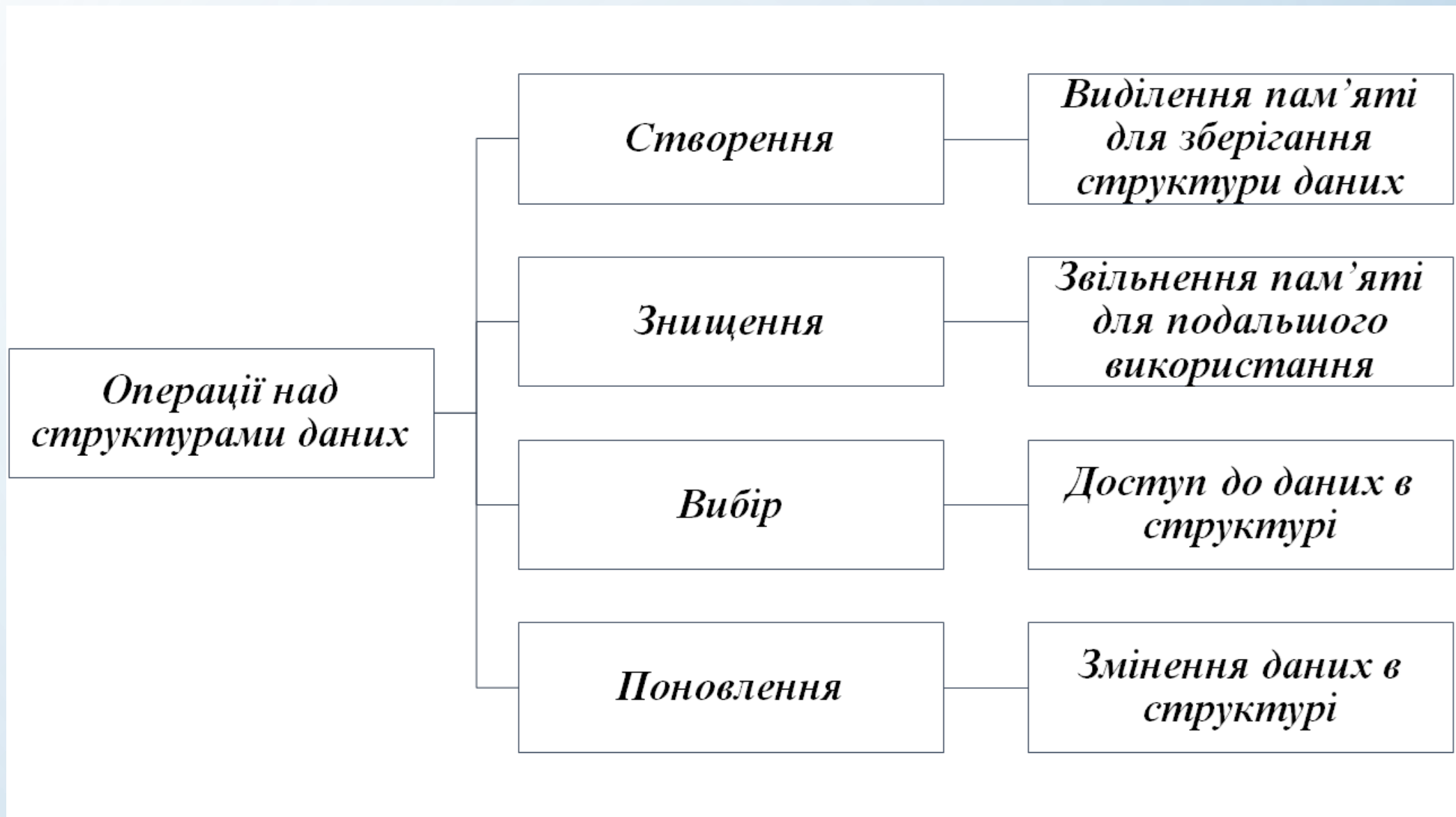


Рисунок 7.2

7.2 ПРОСТІ СТРУКТУРИ ДАНИХ

Арифметичні типи даних - це дані, які використовуються в арифметичних операціях.

Для опису основних типів визначено ключові слова: **цілий (int), символний (char), логічний (true, false), дійсний (float, double)**.

Внутрішнє представлення величин цілого типу – ціле число в бінарному коді.

Внутрішнє представлення дійсного типу складається з двох частин – мантиси і порядку.

Результати **логічного типу** отримуються при порівнянні даних будь-яких типів. Величини логічного типу можуть приймати тільки значення **true** і **false**.

Значенням символного типу є символи з деякої наперед визначеної множини (256 символів).

Над арифметичними типами здійснюються *чотири основних операції*: створення, знищення, вибір, поновлення.

Специфічні операції над числовими типами — додавання, віднімання, множення і ділення.

Операції порівняння: „рівно”, „не рівно”, „більше”, „менше” і т.п.

Особливість: виконання порівнянь на рівність/нерівність дійсних чисел. Оскільки ці числа представляються в пам'яті з деякою точністю, порівняння їх не завжди може бути абсолютно достовірним.

Перерахований тип представляє собою впорядкований тип даних, який визначається програмістом.

Діапазон значень перелічень визначається кількістю біт, які необхідні для представлення всіх його значень.

При відсутності ініціалізації перша константа приймає **нульове значення**, а кожній наступній присвоюється на **одиницю більше значення від попереднього**.

На фізичному рівні над змінними **перерахованого типу** визначені операції **створення, знищення, вибору, поновлення**. При цьому виконується визначення порядкового номера ідентифікатора за його значенням і, навпаки, за номером ідентифікатора його значення.

При виконанні **арифметичних операцій перелічення перетворюються у ціле**. Оскільки перелічення є типом, який визначається користувачем, для нього можна вводити власні операції.

Показчик – це адреса комірки пам'яті.

Випадки використання показчиків:

1. При необхідності представити одну й ту ж ділянку пам'яті, а значить, одні й ті ж фізичні дані, як дані різної логічної структури.
2. При роботі з динамічними структурами даних

В C++ розрізняють три види показчиків – **показчик на об'єкт, на функцію і на пустий тип**, які відрізняються властивостями і набором допустимих операцій.

У програмах на мовах високого рівня показчики можуть бути **типізованими** і **нетипізованими**.

Основними операціями, в яких беруть участь показчики є **присвоєння, отримання адреси, вибір**.

7.3 СТАТИЧНІ СТРУКТУРИ ДАНИХ

Статичні структури даних – це структурована множина примітивних, базових, структур.

Оскільки статичні структури відрізняються відсутністю змінності, пам'ять для них виділяється автоматично – на етапі компіляції, або при виконанні – в момент активізації того програмного блоку, в якому вони описані.

Статичні структури в мовах програмування зв'язані із структурованими типами.

Структуровані типи в мовах програмування – це засоби інтеграції, які дозволяють будувати структури даних будь-якої складності. До таких типів відносяться **масиви, структури** та **їхні похідні типи**.

Масив – це впорядкована сукупність елементів деякого типу, які адресуються за допомогою одного або декількох індексів.

Масиви **класифікуються** за кількістю розмірностей масиву масиви поділяються на одновимірні масиви (вектори), двохвимірні (матриці) і багатовимірні (трьох, чотирьох і більше).

Масив – це така структура даних, яка характеризується:

- фіксованим набором елементів одного і того ж типу;
- кожний елемент має унікальний набір значень індексів;
- кількість індексів визначають мірність масиву;
- звернення до елемента масиву виконується за ім'ям масиву і значенням індексів для даного елемента.

Фізична структура масиву – це спосіб розміщення елементів масиву в пам'яті комп'ютера.

Операції над масивами: **сортування масиву, пошук елемента за ключем.**

Розріджений масив – це масив, більшість елементів якого рівні між собою, так що зберігати в пам'яті достатньо лише невелику кількість значень відмінних від основного (фонового) значення інших елементів.

Розрізняють ***два типи розріджених масивів***:

- масиви, в яких розташування елементів із значеннями відмінними від фонового, можуть бути описані математично;
- масиви з випадковим розташуванням елементів.

До масивів з математичним описом розташування елементів відносяться масиви, в яких існує закономірності в розташуванні елементів із значеннями відмінними від фонового.

До масивів з випадковим розташуванням елементів відносяться масиви, в яких не існує закономірностей у розташуванні елементів із значеннями відмінними від фонового.

Множина – така структура, яка є набором даних одного і того ж типу, що не повторюються. Кожен окремий елемент є членом множини, якщо він входить до складу елементів множини.

Над множинами визначені наступні *специфічні операції*:

1. **Об'єднання множин.** Результатом є множина, що містить елементи початкових множин.
2. **Перетин множин.** Результатом є множина, що містить спільні елементи початкових множин.
3. **Різниця множин.** Результатом є множина, яка містить елементи першої множини, які не входять в другу множину.
4. **Симетрична різниця.** Результатом є множина, яка містить елементи, які входять до складу однієї або другої множини (але не обох).
5. **Перевірка на входження елемента в множину.** Результатом цієї операції є значення логічного типу, що вказує чи входить елемент в множину.

Структура може містити елементи різних типів.

Елементи структури називаються полями структури і можуть мати довільний тип, крім типу цієї ж структури, але можуть бути покажчиками на неї.

Звернення до окремих полів структури замінюються на їхні адреси ще на етапі компіляції.

Самою найважливішою **операцією** для структури є операція доступу до вибраного поля структури – **операція кваліфікації**.

Більшість мов програмування підтримує деякі операції, які працюють із структурою, як з єдиним цілим, а не з окремими її полями. Це операція присвоєння значення одного запису іншому однотипному запису, при цьому відбувається по елементне копіювання

Об'єднання – це частковий випадок структури, усі поля якої розміщуються за однією ж і тою ж адресою.

Об'єднання **застосовуються** для економії пам'яті в тих випадках, коли відомо, що більше одного поля одночасно не потрібно, а також для різної інтерпретації одного і того ж бітового представлення.

У загальному випадку змінна структура буде складатися з трьох частин: **набір спільних компонентів, мітки активного компоненту і частини зі змінними компонентами.**

Бітові типи – це набір бітів, які упаковані в байти або слова, і логічно не зв'язаних один з одним.

Бітові поля – це особливий вид полів структури.

Вони використовуються для компактного розміщення даних, наприклад, прапорців типу „так/ні”. Мінімально адресована комірка пам'яті – 1 байт, а для зберігання прапорця достатньо одного біта. Бітові поля описуються за допомогою структурного типу.

Бітові поля можуть бути довільного цілого типу. Ім'я поля може бути відсутнім, такі поля використовуються для вирівнювання на апаратну межу. Доступ до поля здійснюється звичайним способом – за іменем.

Над бітовими типами можливі три групи специфічних операцій: **операції алгебри логіки, операції зсуву, операції порівняння.**

Таблиця є вектором, елементами якого є структури.

Характерною особливістю таблиць є те, що доступ до елементів таблиці проводиться не за номером (індексом), а за ключем – значення однієї з властивостей об'єкту, який описується структурою-елементом таблиці.

Ключ – це властивість, що ідентифікує дану структуру в множині однотипних структур і є, як правило, унікальним в даній таблиці. Ключ може включатися до складу структури і бути одним з його полів, але може і не включатися в структуру, а обчислюватися за деякими її властивостями. Таблиця може мати один або декілька ключів.

Основною операцією при роботі з таблицями є операція доступу до структури за ключем. Вона реалізовується **процедурою пошуку**. Оскільки пошук може бути значне більш ефективним в таблицях, впорядкованих за значеннями ключів, досить часто над таблицями необхідно виконувати **операції сортування**.

Таблиці з *фіксованою* і із *змінною довжиною структури*.

Таблиці, які об'єднують структури ідентичних типів, будуть мати *фіксовані довжини структур*.

Таблиці із *структурами змінної довжини* обробляються тільки послідовно — *в порядку зростання номерів структур*. Доступ до елемента такої таблиці звичайно здійснюється в *два кроки*:

- На першому кроці вибирається постійна частина структури, в якій міститься, — в явному чи неявному вигляді — довжина структури.
- На другому кроці вибирається змінна частина структури у відповідності з її довжиною. Додавши до адреси поточної структури її довжину, одержують адресу наступної структури.

7.4 НАПІВСТАТИЧНІ СТРУКТУРИ ДАНИХ

Характерні особливості напівстатичних структур:

- вони мають змінну довжину і прості процедури її зміни;
- зміна довжини структури відбувається в певних межах, не перевищуючи якогось максимального (граничного) значення.

Фізичне представлення напівстатичних структур даних в пам'яті – це звичайно послідовність комірок в пам'яті, де кожний наступний елемент розташований в пам'яті в наступній комірці.

Фізичне представлення може мати також вид одно-направленого зв'язного списку (ланцюжки), де кожний наступний елемент адресується покажчиком, який знаходиться в поточному елементі.

Стеком називається множина деякої змінної кількості даних, над якою виконуються наступні операції:

- Поповнення стеку новими даними;
- Перевірка, яка визначає чи стек пустий;
- Перегляд останніх добавлених даних;
- Знищення останніх добавлених даних.

Стек – це такий послідовний список із змінній довжиною, включення і виключення елементів з якого виконуються тільки з одного боку списку.

Основні операції над стеком – **включення нового елемента** (push – заштовхувати) і **виключення елемента зі стеку** (pop – вискакувати).

Допоміжні операції:

- визначення поточної кількості елементів в стеку;
- очищення стеку;
- „неруйнуюче” читання елемента з вершини стека, яке може бути реалізоване, як комбінація основних операцій – виключити елемент зі стеку та включити його знову в стек

Чергою називається множина змінної кількості даних, над якою можна виконувати наступні операції:

- Поповнення черги новими даними;
- Перевірка, яка визначає чи пуста черга;
- Перегляд перших добавлених даних;
- Знищення самих перших добавлених даних.

Чергою FIFO (First – In – First – Out – „першим прийшов – першим вийшов”). називається такий послідовний список із змінній довжиною, в якому включення елементів виконується тільки з одного боку списку (хвіст черги), а виключення – з другого боку (голова черги).

Основні операції над чергою – ті ж, що і над стеком – **включення, виключення, визначення розміру, очищення, „неруйнуюче” читання**

Дек (deq – double ended queue, тобто черга з двома кінцями) – це такий послідовний список, в якому як включення, так і виключення елементів, може здійснюватися з будь-якого з двох кінців списку.

Окремий випадок деку – **дек з обмеженим входом і дек з обмеженим виходом**.

Операції над Деком:

- включення елемента праворуч;
- включення елемента ліворуч;
- виключення елемента з права;
- виключення елемента з ліва;
- визначення розміру;
- очищення.

Фізична структура деку в статичній пам'яті ідентична структурі кільцевої черги.

Лінійний список – це скінчена послідовність однотипних елементів (вузлів), можливо, з повторенням. Кількість елементів у послідовності називається довжиною списку.

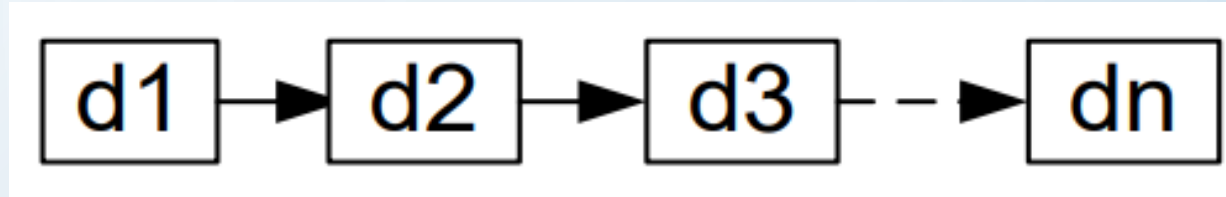


Рисунок 7.3

Важливою властивістю лінійного списку є те, що його елементи можна лінійно впорядкувати у відповідності з їх позицією в списку.

Операції:

- Знайти елемент із заданою властивістю;
- Визначити i -й елемент у лінійному списку;
- Внести додатковий елемент до або після вказаного вузла;
- Вилучити певний елемент списку;
- Впорядкувати вузли лінійного списку в певному порядку.

Допустимі операції: вставки, локалізації, вибірки елементу зі списку, вилучення, вибірки попереднього і наступного елемента

Мультисписок – це багато-зв’язковий список або, кожний елемент якого може входити одночасно в декілька однозв’язних списків.

До переваг мультисписків крім економії пам’яті слід віднести також цілісність даних – в тому сенсі, що всі підзадачі працюють з однією і тією ж версією інформаційної частини і зміни в даних, зроблені однієї підзадачею негайно стають доступними для іншої підзадачі.

Стрічка – це лінійно впорядкована послідовність символів, які належать до скінченної множини символів, яка називається алфавітом.

Властивості стрічки: їхня довжина, як правило, змінна, хоч алфавіт фіксований; звичайне звернення до символів стрічки йде з будь-якого одного боку послідовності; метою доступу до стрічки є на окремий її елемент, а ланцюжок символів.

Операції: визначення довжини стрічки; присвоєння стрічки; конкатенація (зчеплення) стрічок; виділення підстрічки; пошук входження.

7.5 ДИНАМІЧНІ СТРУКТУРИ ДАНИХ

Динамічні структури даних – це структури даних, пам'ять під які виділяється і звільняється в міру необхідності.

Динамічні структури даних в процесі існування в пам'яті *можуть змінювати не тільки число складових їх елементів, а й характер зв'язків між елементами.*

Динамічна структура даних характеризується тим що:

- вона не має імені;
- їй виділяється пам'ять в процесі виконання програми;
- кількість елементів структури може не фіксуватися;
- розмірність структури може змінюватися в процесі виконання програми;
- в процесі виконання програми може змінюватися характер взаємозв'язку між елементами структури.

Кожній динамічній структурі даних *зіставляється статична змінна типу вказівник* (її значення - адреса цього об'єкта), за допомогою якої здійснюється доступ до динамічної структури.

Необхідність в динамічних структурах даних зазвичай виникає в наступних випадках:

- Використовуються змінні, що мають досить великий розмір (наприклад, масиви великої розмірності), необхідні в одних частинах програми і абсолютно не потрібні в інших.
- В процесі роботи програми потрібен масив, список або інша структура, розмір якої змінюється в широких межах і важко передбачуваний.
- Коли розмір даних, що обробляються в програмі, перевищує обсяг сегмента даних.

Динамічні структури, за визначенням, *характеризуються відсутністю фізичної суміжності елементів структури в пам'яті, непостійністю і непередбачуваністю розміру* (числа елементів) структури в процесі її обробки.

Встановлення явних зв'язків між елементами динамічної структури є *зв'язним* представлення даних в пам'яті.

Переваги:

- можливість забезпечення значної змінності структур;
- розмір структури обмежується тільки доступним об'ємом машинної пам'яті;
- при зміні логічної послідовності елементів структури потрібно виконати не переміщення даних в пам'яті, а тільки корекцію покажчиків.

Недоліки:

- робота з покажчиками вимагає більш високої кваліфікації від програміста;
- на поля зв'язку витрачається додаткова пам'ять;
- доступ до елементів зв'язної структури може бути менш ефективним за часом.

Порядок роботи з динамічними структурами даних наступний:

- створити (відвести місце в динамічній пам'яті);
- працювати за допомогою вказівника;
- видалити (звільнити зайняте структурою місце).

Класифікація динамічних структур даних:

- односпрямовані (одинзв'язні) списки;
- двонаправлені (двусвязного) списки;
- циклічні списки;
- стек;
- дек;
- черга;
- бінарне дерева.

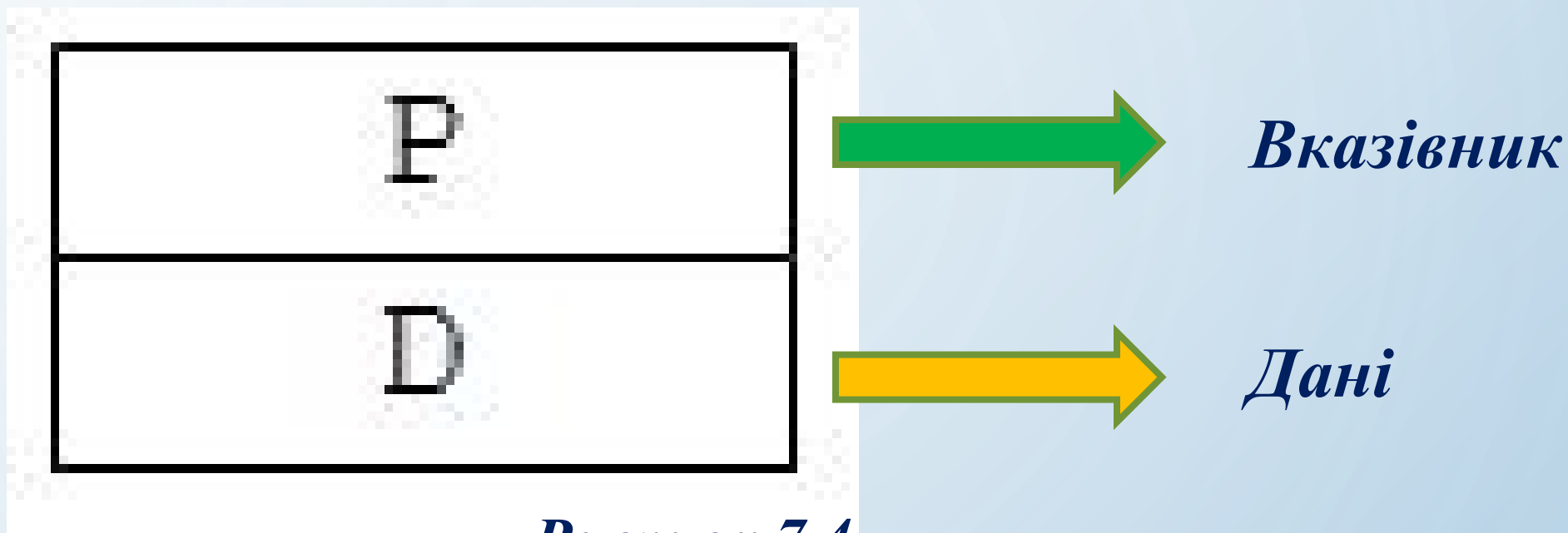


Рисунок 7.4

Елемент динамічної структури складається з двох полів:

- інформаційного поля або поля даних, в якому містяться ті дані, заради яких і створюється структура;
- поле зв'язку, в якому міститься один або декілька покажчиків, які зв'язують даний елемент з іншими елементами структури

Оголошення динамічних структур даних:

```
struct ім'я_типу
```

```
{
```

```
    інформаційне поле;
```

```
    адресне поле;
```

```
};
```

*Інформаційних і адресних полів
може бути як одне, так і
декілька.*

Приклад 7.1:

```
struct TNode
```

```
{
```

```
    int Data;
```

```
    // інформаційне поле
```

```
    TNode * Next;
```

```
    // адресне поле
```

```
};
```

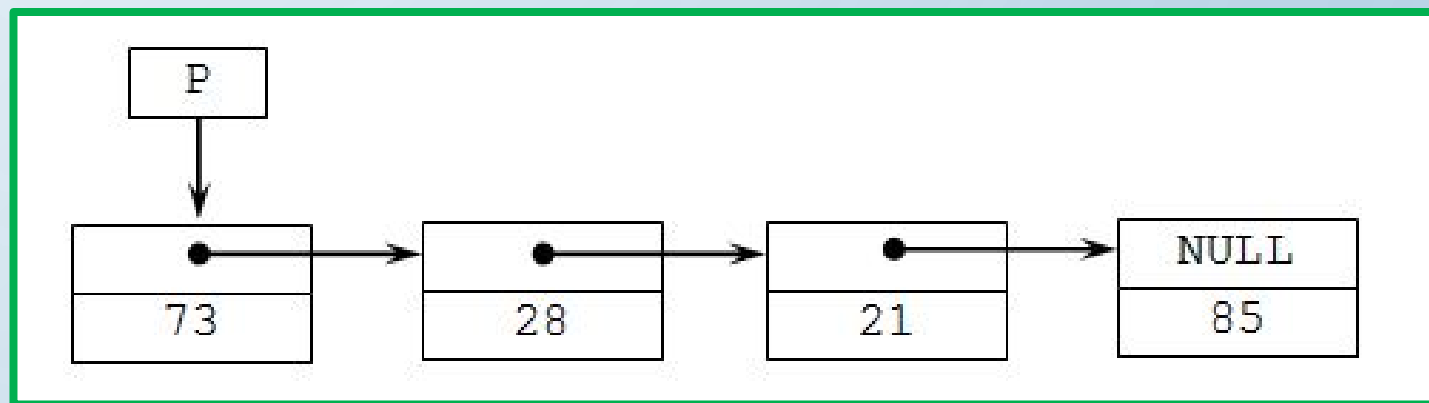


Рисунок 7.5

Доступ до даних в динамічних структурах

Вказівник містить адресу певного об'єкта в динамічній пам'яті. Адреса формується з двох слів: адреса сегмента і зміщення. Сам вказівник є статичним об'єктом і розташований в сегменті даних.

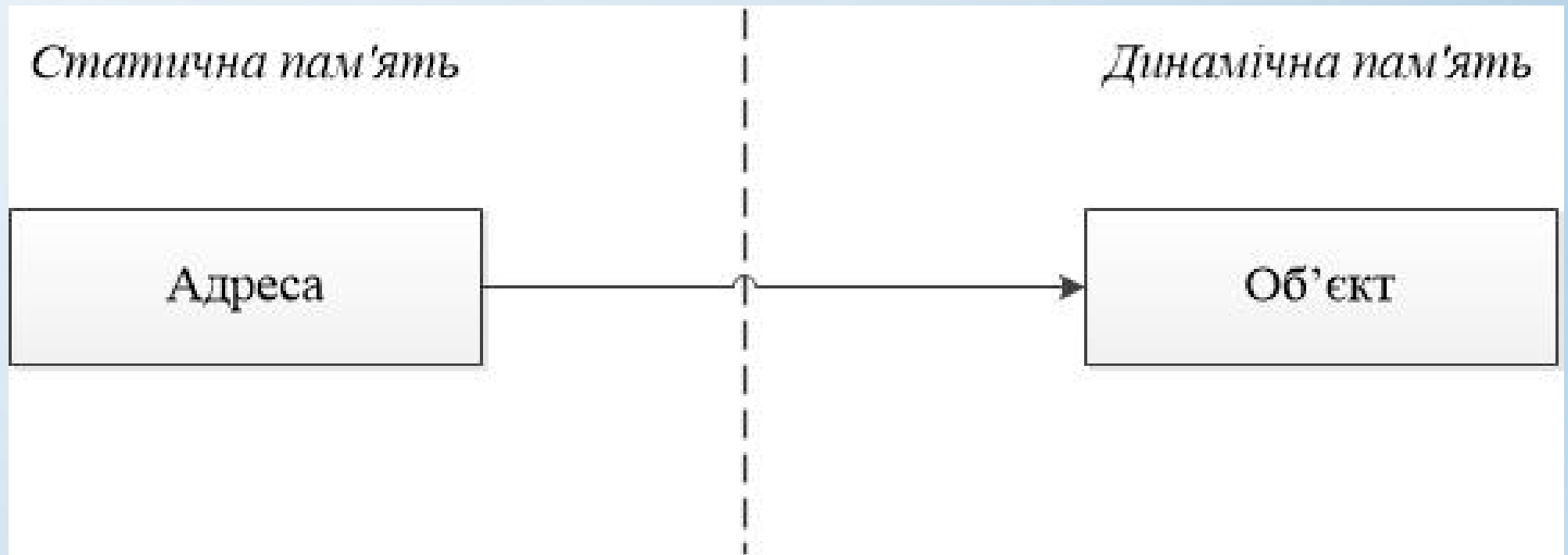


Рисунок 7.6

7.6 НЕЛІНІЙНІ СТРУКТУРИ ДАНИХ

Нелінійні структури – багатозв'язні списки, дерева, графи.

Граф – це складна нелінійна багато-зв'язна динамічна структура, що відображає властивості і зв'язки складного об'єкту.

Властивості графів:

- на кожний елемент (вузол, вершину) може бути довільна кількість посилань;
- кожний елемент може мати зв'язок з будь-якою кількістю інших елементів;
- кожний зв'язок (ребро, дуга) може мати напрям і вагу.

Існує **два основні методи представлення графів** в пам'яті комп'ютера: матричний і зв'язними нелінійними списками.

Дерево – це граф, який характеризується наступними властивостями:

- Існує єдиний елемент (вузол або вершина), на який не посилається ніякий інший елемент, – він називається коренем.
- Починаючи з кореня і слідуючи по певному ланцюжку покажчиків, що містяться в елементах, можна здійснити доступ до будь-якого елемента структури.
- На кожний елемент, крім кореня, є єдине посилення, тобто кожний елемент адресується єдиним покажчиком.

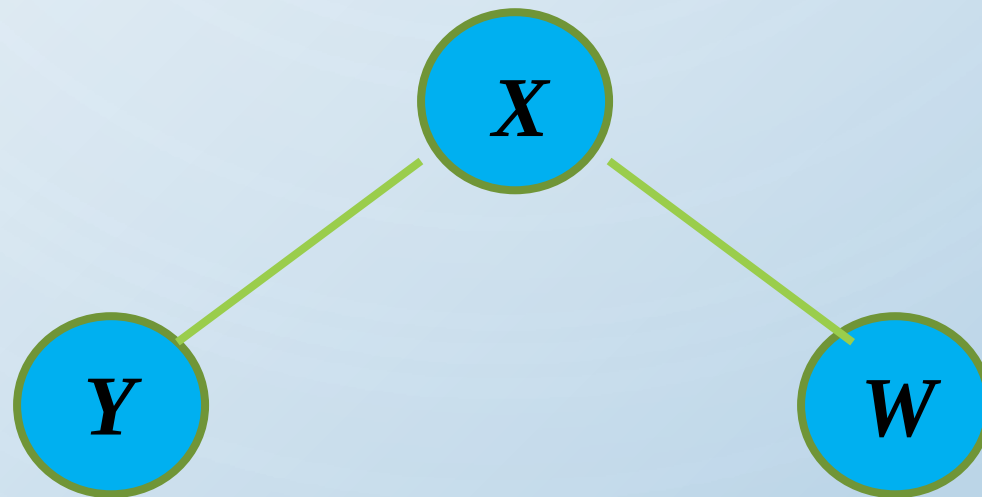


Рисунок 7.7

Способи графічного зображення дерев:

- *перший спосіб полягає у використуванні для зображення піддерев відомого методу діаграм Венна,*
- *другий – методу дужок, що вкладаються одна в одну,*
- *третій спосіб – це спосіб, який використовується при складанні змісту книг.*

Правило побудови бінарного дерева з будь-якого дерева:

- В кожному вузлі залишити тільки гілку до старшого сина;
- З'єднати горизонтальними ребрами всіх братів одного батька;
- Таким чином перебудувати дерево за правилом: лівий син – вершина, розташована під даною; правий син – вершина, розташована праворуч від даної (тобто на одному ярусі з нею).
- Розвернути дерево так, щоб усі вертикальні гілки відображали лівих синів, а горизонтальні – правих

В бінарному дереві є два покажчики, тому зручно вузол представити у вигляді структури в якій

left – покажчик на ліве піддерево,

right – покажчик на праве піддерево,

inf – містить інформацію, яка зв'язана з вершиною,

data – визначений тип.

Над деревами визначені наступні основні операції:

- Пошук вузла із заданим ключем.
- Додавання нового вузла.
- Видалення вузла (піддерева).
- Обхід дерева в певному порядку.

Пошук в бінарному дереві

Нехай побудовано деяке дерево і вимагається знайти вузол з ключем X . Спочатку порівнюємо з X ключ, що знаходиться в корені дерева. У разі рівності пошук закінчений і потрібно повернути покажчик на корінь в якості результату пошуку. Інакше переходимо до розгляду вершини, яка знаходиться зліва внизу, якщо ключ X менший тільки що розглянутого, або справа внизу, якщо ключ X більший тільки що розглянутого. Порівнюємо ключ X з ключем, що міститься в цій вершині, і т.д.

Процес завершується в одному з двох випадків:

- знайдена вершина, що містить ключ, рівний ключу X ;
- в дереві відсутня вершина, до якої потрібно перейти для виконання чергового кроку пошуку.

Бінарне дерево можна обходити трьома основними способами: *низхідним, змішаним і висхідним* (можливі також зворотний низхідний, зворотний змішаний і зворотний висхідний обходи).

Прийняті назви методів обходу зв'язані з часом обробки кореневої вершини: *До того як оброблено обидва його піддерева, після того, як оброблено ліве піддерево, але до того як оброблено праве, після того, як оброблено обидва піддерева.*

Використовувані назви методів відображають напрям обходу в дереві: *від кореневої вершини вниз до листя – низхідний обхід; від листя вгору до кореня – висхідний обхід, і змішаний обхід – від найлівішого листка дерева через корінь до найправішого листка.*

Алгоритм обходу бінарного дерева відповідно до низхідного способу:

1. В якості чергової вершини взяти корінь дерева. Перейти до пункту 2.
2. Провести обробку чергової вершини відповідно до вимог задачі. Перейти до пункту 3.
3. а) Якщо чергова вершина має обидві гілки, то в якості нової вершини вибрати ту вершину, на яку посилається ліва гілка, а вершину, на яку посилається права гілка, занести в стек; перейти до пункту 2; б) якщо чергова вершина є кінцевою, то вибрати в якості нової чергової вершини вершину із стека, якщо він не порожній, і перейти до пункту 2; якщо ж стек порожній, то це означає, що обхід всього дерева закінчений, перейти до пункту 4; в) якщо чергова вершина має тільки одну гілку, то в якості чергової вершини вибрати ту вершину, на яку ця гілка вказує, перейти до пункту 2.
4. Кінець алгоритму.

Низхідний обхід (алгоритм):

- 1) Обробка кореневої вершини;
- 2) Низхідний обхід лівого піддерева;
- 3) Низхідний обхід правого піддерева.

Змішаний обхід (алгоритм):

- 1) Спуститися по лівій гілці із запам'ятовуванням вершин в стеку;
- 2) Якщо стек порожній те перейти до п.5;
- 3) Вибрати вершину із стеку і обробити дані вершини;
- 4) Якщо вершина має правого сина, то перейти до нього; перейти до п.1.
- 5) Кінець алгоритму.

Рекурсивний змішаний обхід (алгоритм):

- 1) Змішаний обхід лівого піддерева;
- 2) Обробка кореневої вершини;
- 3) Змішаний обхід правого піддерева.

Алгоритм висхідного обходу:

- 1) Спуститися по лівій гілці із запам'ятовуванням вершини в стеку як 1-й вид стекових записів;
- 2) Якщо стек порожній, то перейти до п.5;
- 3) Вибрати вершину із стека, якщо це перший вид стекових записів, то повернути його в стек як 2-й вид стекових записів; перейти до правого сина; перейти до п.1, інакше перейти до п.4;
- 4) Обробити дані вершини і перейти до п.2;
- 5) Кінець алгоритму.

Рекурсивний висхідний обхід (алгоритм):

- 1) Висхідний обхід лівого піддерева;
- 2) Висхідний обхід правого піддерева;
- 3) Обробка кореневої вершини.

Тема №8. ВИРАЗИ ТА ОПЕРАЦІЇ

Тематичний план

8.1 Визначення базових виразів та операцій в мові програмування C/C++

8.2 Аналіз виконання операцій

8.3 Узгодження та перетворення типів операндів у виразах

8.1 ВИЗНАЧЕННЯ БАЗОВИХ ВИРАЗІВ ТА ОПЕРАЦІЙ В МОВІ ПРОГРАМУВАННЯ C/C++

Будь-який вираз мови складається з операндів (змінних, констант і ін.), з'єднаних знаками *операцій*.

Знак операції - це символ або група символів, які повідомляють компілятору про необхідність виконання певних арифметичних, логічних або інших дій.

Таблиця 8.1 – Оператори

Пріоритет	Оператор	Опис	Асоціативність
1	()	Виклик функції	Зліва направо
	[]	Індексація в масиві	
	.	Вибір елемента структури за посиланням	
	->	Вибір елемента структури за вказівником	

Пріоритет	Оператор	Опис	Асоціативність
2	++	Інкремент	Справа наліво
	--	Декремент	
	+	Унарний плюс	
	-	Унарний мінус	
	!	Логічне заперечення (NOT)	
	~	Побітне заперечення (Bitwise NOT, доповнення до одиниці)	
	(type)	Перетворення типу	
	*	Вибірка значення за посиланням	
	&	Операція взяття адреси (отримання покажчика)	
	sizeof	Отримання розміру об'єкта зазначеного типу (структури, змінної і т. п.)	

Пріоритет	Оператор	Опис	Асоціативність
3	*	Множення	Зліва направо
	/	Ділення	
	%	Взяття по модулю числа	
4	+	Складання	Зліва направо
	-	Віднімання	
5	<<	Побітний зсув вліво	Зліва направо
	>>	Побітний зсув вправо	
6	<	Менше ніж...	Зліва направо
	<=	Менше або дорівнює...	
	>	Більше ніж...	
	>=	Більше або дорівнює...	

Продовження таблиці 8.1 –

Пріоритет	Оператор	Опис	Асоціативність
7	==	Оператор визначення рівності (Equal to).	Зліва направо
	!=	Оператор визначення нерівності (Not equal to)	
8	&	Побітна операція І (Bitwise AND)	Зліва направо
9	^	Побітна операція Виключне або (Bitwise XOR, exclusive or).	Зліва направо
10		Побітна операція АБО (Bitwise OR, inclusive or).	Зліва направо
11	&&	Логічна операція І (AND).	Зліва направо
12		Логічна операція АБО (OR).	Зліва направо
13	?:	Теренарний оператор	Справа наліво

Пріоритет	Оператор	Опис	Асоціативність
14	=	Пряме присвоювання	Справа наліво
	+=	Присвоєння зі складанням	
	-=	Присвоєння з відніманням	
	*=	Присвоєння з множенням	
	/=	Присвоєння з діленням	
	%=	Присвоєння з взяттям залишку від ділення	
	<<=	Присвоєння зі побітним зсувом вліво	
	>>=	Присвоєння зі побітним зсувом вправо	
	&=	Присвоєння зі побітним AND.	
	^=	Присвоєння зі побітним XOR.	
	=	Присвоєння зі побітним OR.	
15 найнижчий	,	Кома	Зліва направо

8.2 АНАЛІЗ ВИКОНАННЯ ОПЕРАЦІЙ

Проста операція присвоєння

<змінна> = <вираз>;

<змінна> - ім'я змінної, значення якої змінюється в результаті виконання інструкції присвоювання;

<вираз> – вираз, значення якого присвоюється змінній, ім'я якої вказане ліворуч від символу присвоювання;

= знак операції присвоювання.

Приклад 8.1

```
a=5;  
x=a+20;  
y=(a+x)*(x-a);  
z=a=x;
```

Приклад 8.2

```
a + (b = c+d)
```

Що не правильно?

Приклад 8.3

```
void main  
{  
  int a,b;  
  float x,y;  
  a=5;  
  10=x;  
  y=7,8;  
  b=2.5;  
  x=2*(a+y);  
  a=b+x;  
}
```

Таблиця 8.2 –

Повний запис	Скорочений запис
$a=a+1$	$a++$
$a=a+b$	$a+=b$
$a=a-1$	$a--$
$a=a-b$	$a-=b$
$a=a*b$	$a*=b$
$a=a/b$	$a/=b$
$a=a\%b$	$a\%=b$

Унарні присвоєння

number++ number--

++ number --number

Приклад 8.4

// оператори інкремента
(++) та декремента (--)

```
int a, b;  
a = 10;  
b = a++; // b = 10; a = 11  
a = 10;  
b = ++a; // b = 11; a = 11  
a = 10;  
b = a--; // b = 10; a = 9  
a = 10;  
b = --a; // b = 9; a = 9
```

Приклад 8.5

```
int t=1, z;  
z=(t++)*5;
```

Приклад 8.6

```
int s=2, f;  
f=(++s)/3;
```

Таблиця 8.3 –

Операція	Опис
- унарний	Зміна знаку операнда
*	Множення
/	Ділення
%	Залишок від цілочислового ділення
+	Додавання
-	Віднімання
++	Інкремент
--	Декремент

Арифметичний вираз може включати:

- константи (постійні);
- імена змінних;
- знаки арифметичних операцій (множення, ділення, залишок від ділення, додавання, віднімання);
- виклики функцій;
- круглі дужки.

Приклад 8.7

$$z = \frac{5 \cdot a \cdot c + 3 \cdot (c - d)}{a \cdot b} \cdot (b - c)$$

Вираз на мові програмування Cі: $z = (5^2 * a^3 * c^5 + 3^4 * (c^1 - d^7)) / a^8 * (b^6 - c^9) / b$

Таблиця 8.4 –

Операція	Опис
==	Дорівнює
>	Більше
<	Менше
>=	Більше або дорівнює
<=	Менше або дорівнює
!=	Не дорівнює

Приклад 8.8

$a > b$; $a \geq b$; $a \leq b$; $a = b$; $a \neq b$

Логічні операції

Таблиця 8.5 –

Операція	Опис
$\parallel$	Або
$\&\&$	І
$!$	Не
$(a \parallel b) \&\& !(a \&\& b)$	Виключне або (xor)
$b \geq 4 \&\& b < 5$	Дужки не потрібні

Приклад 8.9

$a \&\& b$ - операція логічне “І”,
 $a \parallel b$ - операція логічне “АБО”,
 $! a$ - операція логічне “НЕ”.

Порозрядні логічні операції

Ціле число – 18

а) один байт (тип char)

7	6	5	4	3	2	1	0
0	0	0	1	0	0	1	0

б) два байти (тип short int)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

в) чотири байти (тип int)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

Таблиця 8.6 –

Операція	Опис
	Або
&	І
!	Не
^	Виключне або (xor)
~	Заперечення

Застосування бітових операцій:

- перевірка, скидання і установка окремих бітів в складі цілого числа
- економія пам'яті (упаковка декількох логічних значень в одну цілу змінну)
- підвищення продуктивності програм (бітові операції можуть виконуватися одночасно над усіма розрядами на відміну від арифметичних операцій, де потрібно враховувати перенесення)
- робота з регістрами апаратури

Операція «виключне АБО»

x	y	$x \wedge y$
0	0	0
0	1	1
1	0	1
1	1	0

Операція «І»

x	y	$x \& y$
0	0	0
0	1	0
1	0	0
1	1	1

Операція «АБО»

x	y	$x y$
0	0	0
0	1	1
1	0	1
1	1	1

Операція «НЕ»

x	$y = \sim x$
0	1
1	0

Приклад 8.10 Побітова операція І

Змінна	Значення змінної	Двійкове представлення
x	14	0000 1110
y	11	0000 1011
z=x&y	10	0000 1010

```
int x=14, y=11, z;
```

```
z=x&y;
```

```
printf("z=%d",z); //10
```

Приклад 8.11 Побітова операція АБО

Змінна	Значення змінної	Двійкове представлення
x	10	0000 1010
y	9	0000 1001
z=x y	11	0000 1011

```
int x=10,y=9,z;
```

```
z=x|y;
```

```
printf("z=%d",z); //11
```

Приклад 8.11 Побітова операція НЕ

Змінна	Значення змінної	Двійкове представлення
x	10	0000 1010
y= $\sim$ x	245	1111 0101

тип **unsigned char** = **245**

тип **char** = **-11**

	7	6	5	4	3	2	1	0		
	0	0	0	0	1	0	1	1		11
									+	
	1	1	1	1	0	1	0	1		$\sim$ 10 -11
1	0	0	0	0	0	0	0	0		0

Приклад 8.12

	7	6	5	4	3	2	1	0				
	0	0	0	0	1	0	1	0		10	x	
									+			
	1	1	1	1	0	1	0	1		~10	~x	
									+			
	0	0	0	0	0	0	0	1		1	1	
1	0	0	0	0	0	0	0	0		0	0	

$$-x = 1 + \sim x$$

Операції бітового зсуву

Приклад 8.13 Побітовий зсув вліво

Змінна	Значення змінної	Двійкове представлення
x	10	0 0 0 0 1 0 1 0
y=x<<1	20	0 0 0 0 1 0 1 0 0
y=x<<2	40	0 0 0 0 1 0 1 0 0 0
y=x<<3	80	0 0 0 0 1 0 1 0 0 0 0
y=x<<4	160	0 0 0 0 1 0 1 0 0 0 0 0

```
int x=10, y;
```

```
y=x<<1;
```

```
printf("y=%d",y); //20
```

$x \ll n$ еквівалентно множенню x на 2^n

Приклад 8.14 Побітовий зсув вправо

Змінна	Значення змінної	Двійкове представлення
x	10	0 0 0 0 1 0 1 0
y=x>>1	5	0 0 0 0 0 1 0 1 0
y=x>>2	2	0 0 0 0 0 0 1 0 1 0
y=x>>3	1	0 0 0 0 0 0 0 1 0 1 0
y=x>>5	0	0 0 0 0 0 0 0 0 0 1 0 1 0

```
int x=10, y;  
y=x>>1;  
printf("y=%d",y); //5
```

x >> n еквівалентно поділу на 2^n з відкиданням дробової частини результату

Умовна операція (теренарний оператор)

вираз_1 ? вираз_2 : вираз_3;

Приклад 8.15

1. Знаходження *max* із двох чисел:

$\text{max} = (a > b) ? a : b;$

2. Знаходження *абсолютної величини* числа x :

$\text{abs} = (x > 0) ? x : -x;$

8.3 УЗГОДЖЕННЯ ТА ПЕРЕТВОРЕННЯ ТИПІВ ОПЕРАНДІВ У ВИРАЗАХ

Приклад 8.16

Арифметичні перетворення

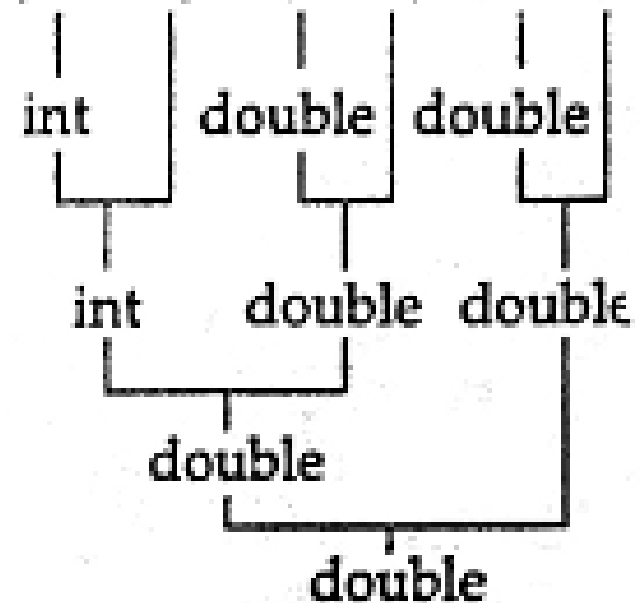
char, short → int

Принцип "найбільшого операнду",
відповідно до ряду:

*long double, double, float, unsigned
long, long, unsigned int, int*

```
char ch;  
int i;  
float f;  
double d;
```

```
result = (ch / i) + (f * d) - (f + i);
```



Перетворення типів в операціях присвоєння

Правила:

- якщо тип змінної ліворуч знаку операції присвоєння вище або дорівнює типу результату обчислення правої частини, то результат обчислення перетворюється за принципом «найбільшого операнду»;

```
float a = 10;
```

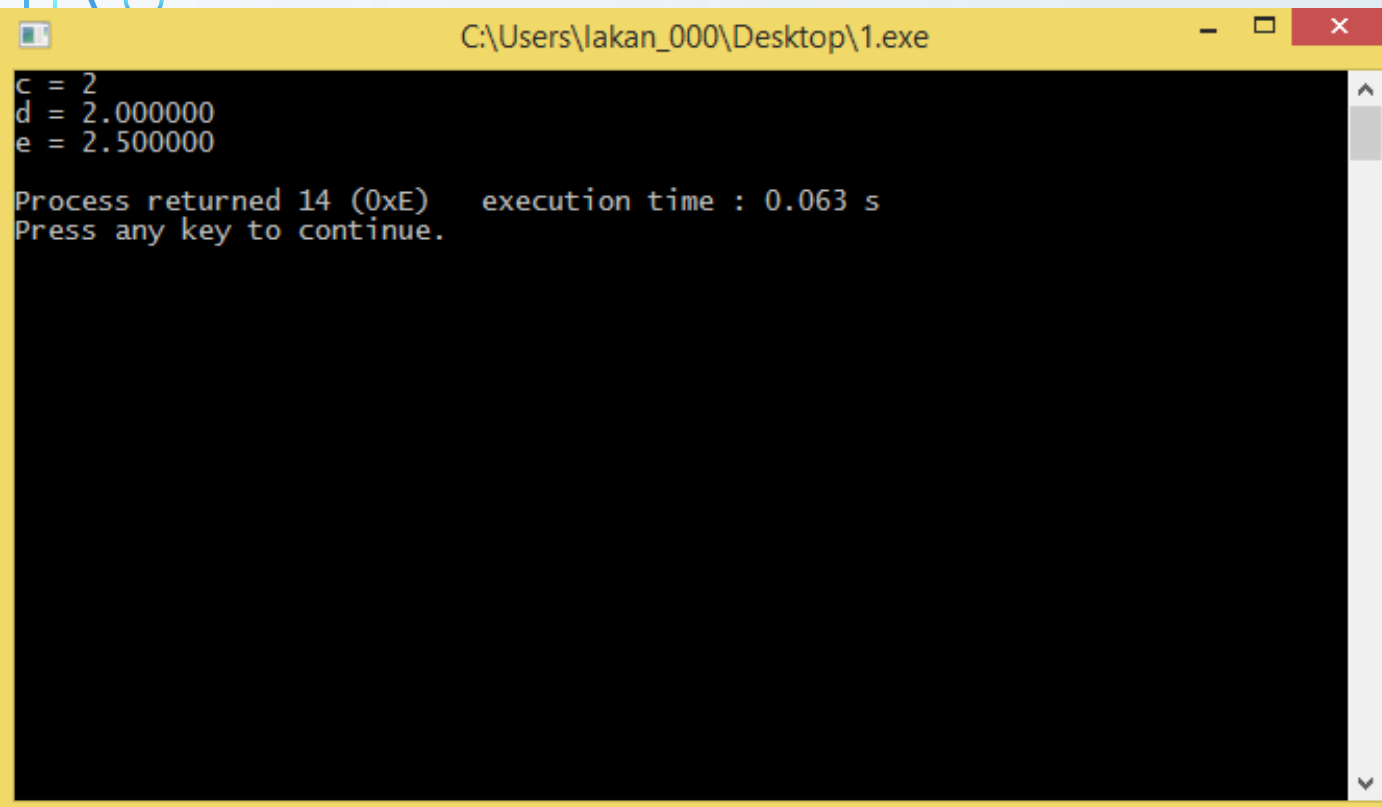
```
float b = 4;
```

```
int c = a/b; // 2
```

- якщо тип змінної ліворуч знаку операції присвоєння нижче типу результату обчислення правої частини, то результат обчислення обтинається. Це полягає у тому, що може бути зменшено число розрядів мантиси (**double ->float**), або відкинуто дробову частину (**float ->int**). Але внаслідок таких перетворень можливо і спотворення результату. Наприклад, перетворення **long -> int** може призвести до втрати старших біт результату.

Конструкція: *(тип) вираз*

Приклад 8.17



```
C:\Users\lakan_000\Desktop\1.exe
c = 2
d = 2.000000
e = 2.500000

Process returned 14 (0xE)   execution time : 0.063 s
Press any key to continue.
```

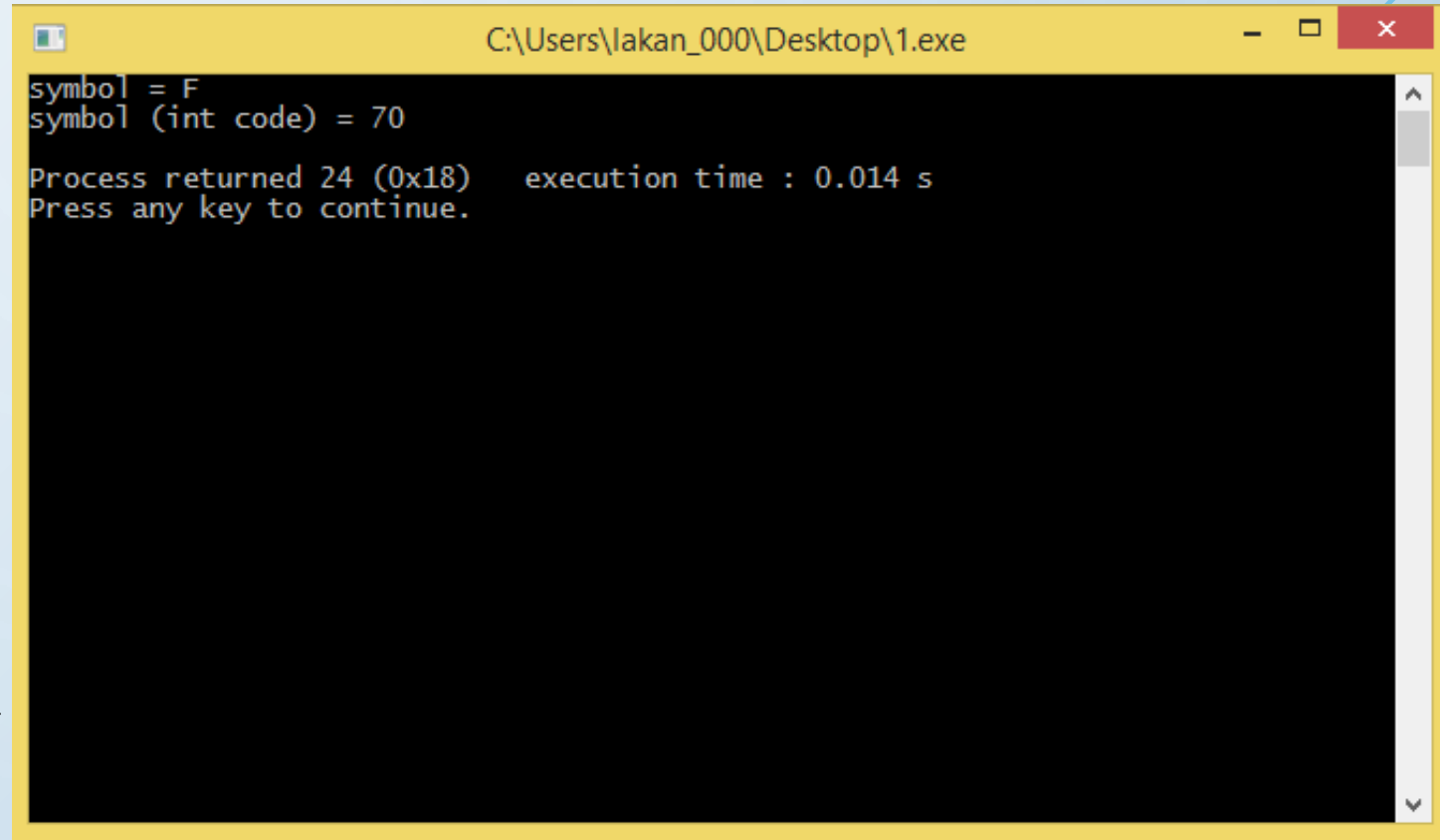
```
int a = 10;
int b = 4;
int c = a / b;           // 2
double d = a / b;        // 2.000000
double e = (double)a / (double)b; // 2.500000

printf("c = %d \n", c);
printf("d = %lf \n", d);
printf("e = %lf \n", e);
```

Рисунок 8.1

Приклад 8.18

```
int number = 70;  
  
char symbol = (char) number;  
printf("symbol = %c \n", symbol);  
// F  
  
printf("symbol (int code) = %d  
\n", symbol); // 70
```



```
C:\Users\lakan_000\Desktop\1.exe  
symbol = F  
symbol (int code) = 70  
Process returned 24 (0x18)    execution time : 0.014 s  
Press any key to continue.
```

Рисунок 8.2

Тема №9. ОПЕРАТОРИ МОВИ СІ

Тематичний план

- 9.1 Умовні оператори: if, switch*
- 9.2 Оператори циклу: while, do-while, for*
- 9.3 Оператор переходу goto*

9.1 УМОВНІ ОПЕРАТОРИ: IF,

SWITCH

Програма – це опис засобами мови програмування алгоритму розв'язання задачі.

Складові алгоритму: слідування, розгалуження і цикли.

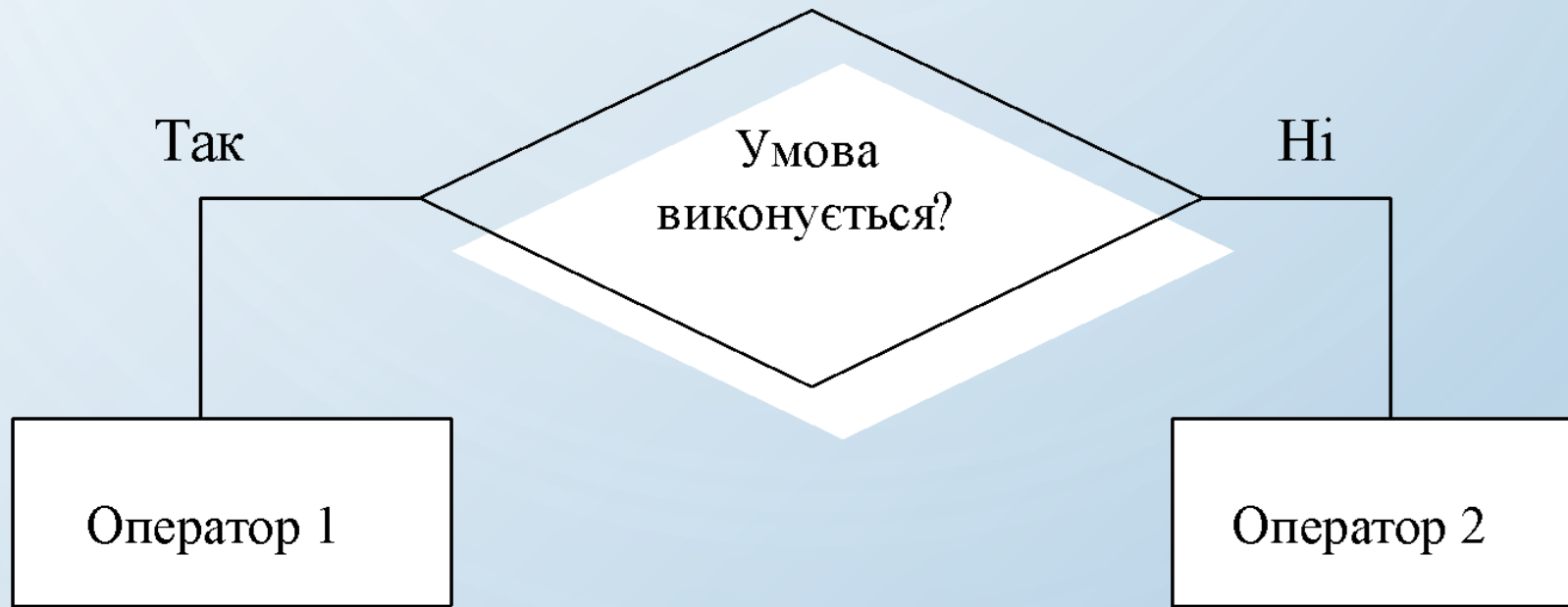


Рисунок 9.1

Оператор розгалуження *if...else*

Повна форма оператора

```
if(<умова>)  
    <оператор 1>  
else <оператор 2>
```

Скорочена форма оператора

```
if(<умова>)  
    <оператор>
```

```
if(<умова>)  
{  
    <оператор 1>  
    <оператор 2>  
    ...  
    <оператор n>  
}
```

```
else  
{  
    <оператор 1>  
    <оператор 2>  
    ...  
    <оператор n>  
}
```

Приклад 9.1 Визначення максимального з двох чисел.

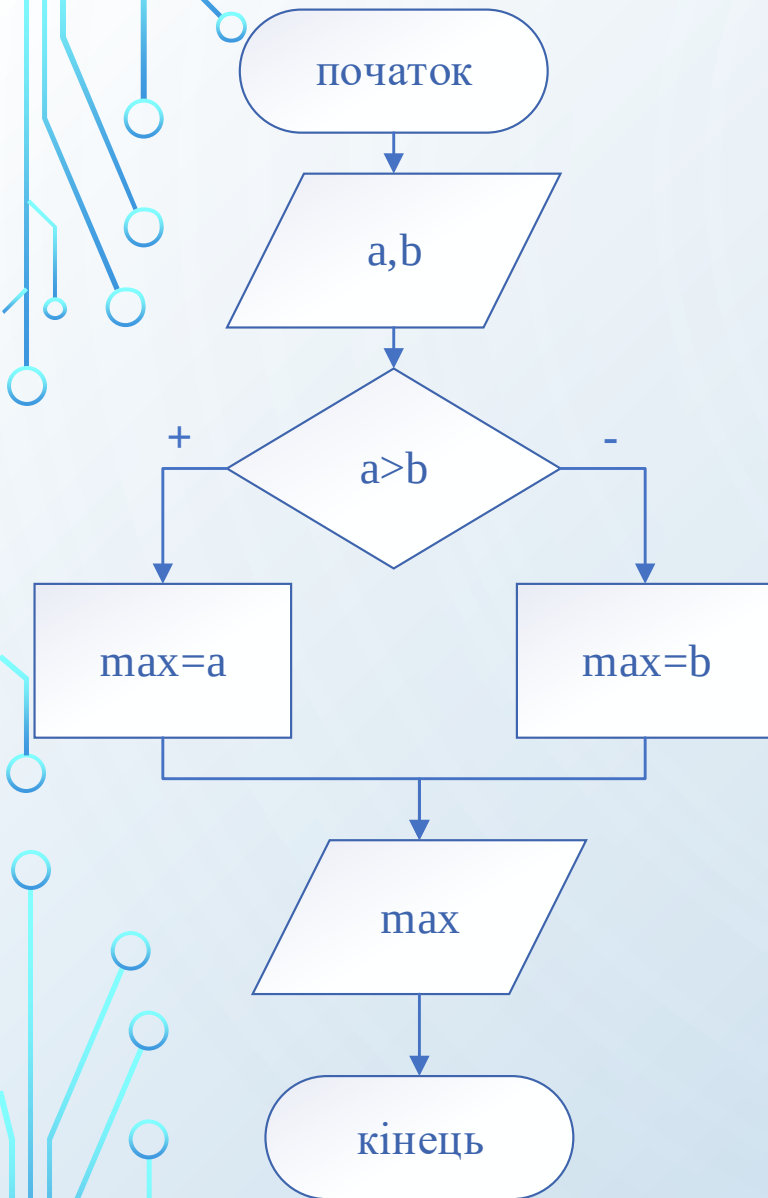


Рисунок 9.2

```
int main ()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    int a,b,max;
    printf("Введіть два цілих числа: \n");
    scanf("%d%d",&a,&b);
```

```
    if(a>b)
        max=a;
    else
        max=b;
```

```
    printf("Найбільше число - %d", max);
    return 0;
}
```

1 варіант

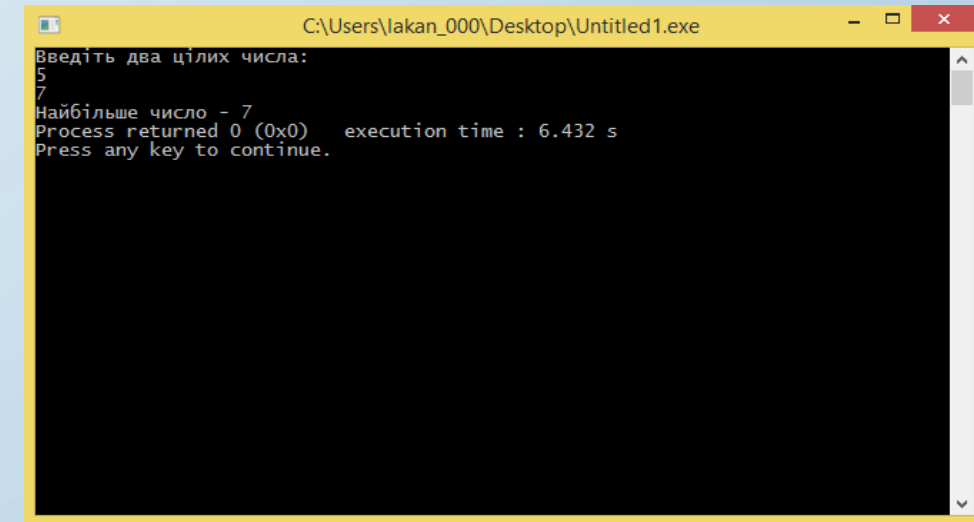


Рисунок 9.3

Якщо a=b ???

2 варіант

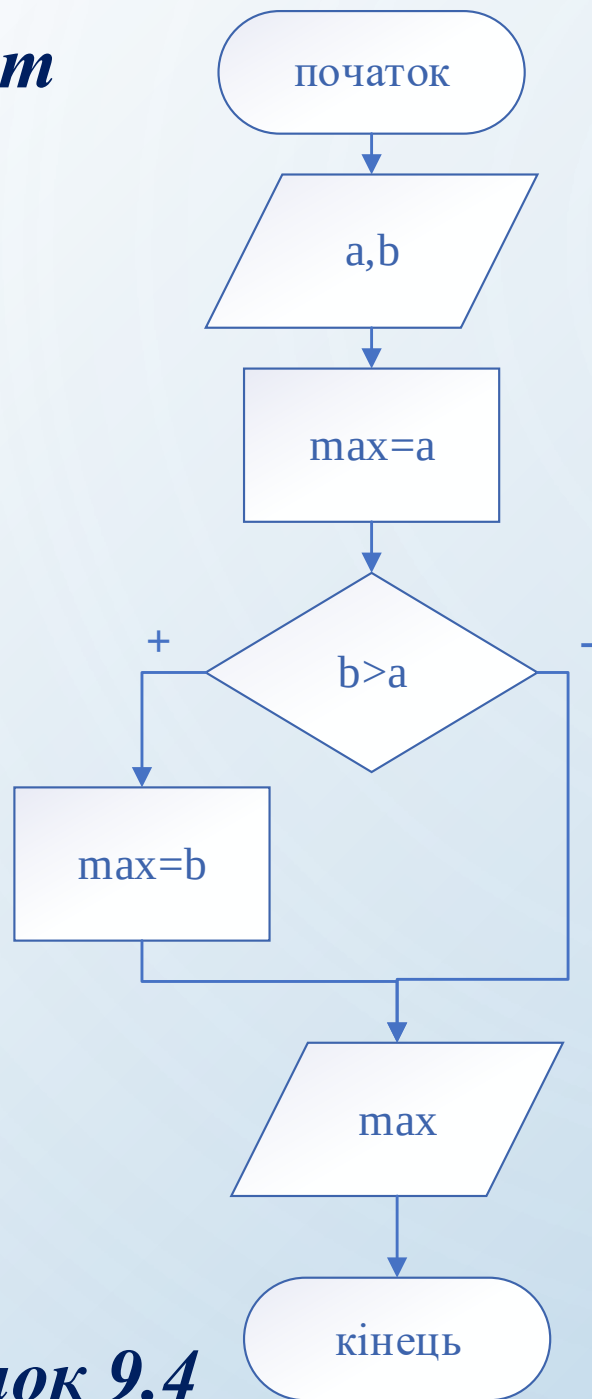


Рисунок 9.4

```
#include <stdio.h>
#include <windows.h>
int main ()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    int a,b,max;
    printf("Введіть два цілих числа: \n");
    scanf("%d%d",&a,&b);
    max=a;
    if(b>a)
    max=b;
    printf("Найбільше число - %d", max);
    return 0;
}
```

Приклад 9.2 Знайти max(a,b,c).

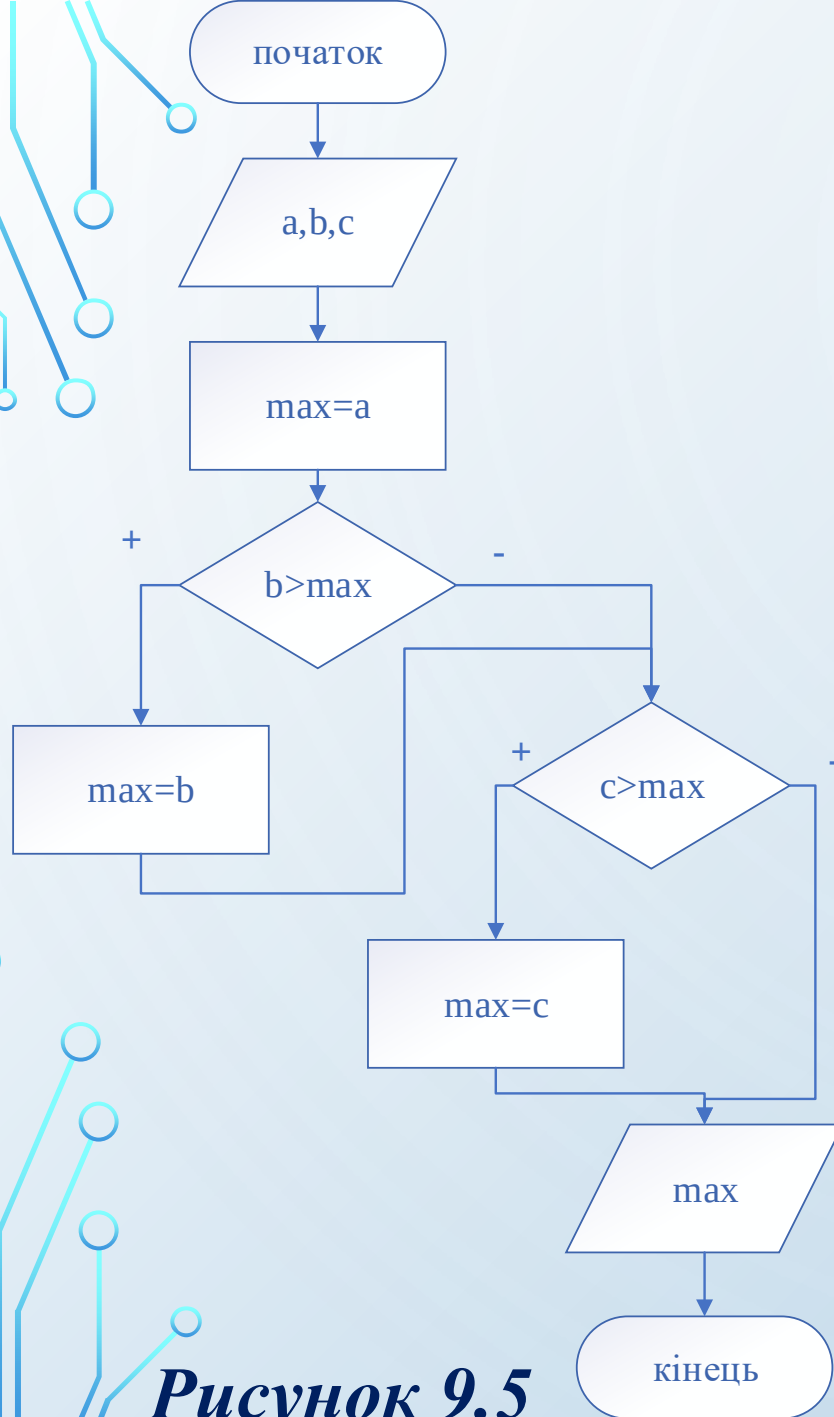


Рисунок 9.5

```
#include <stdio.h>
#include <windows.h>
int main ()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    double a,b,c,max;
    printf("a = ");
    scanf("%lf",&a);
    printf("b = ");
    scanf("%lf",&b);
    printf("c = ");
    scanf("%lf",&c);
    max=a;
    if(b>max)
        max=b;
    if(c>max)
        max=c;
    printf("Найбільше число - %lf",
        max);
    return 0;
}
```

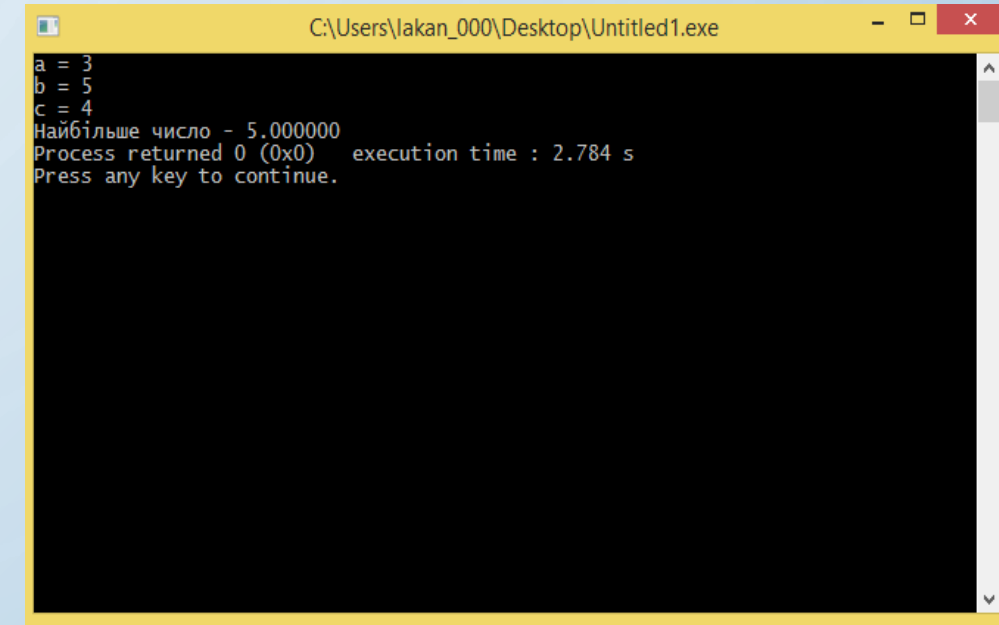


Рисунок 9.6

Вкладений оператор if

```
if <умова1>
{
    <оператор 1.1>;
    if <умова2>
    {
        <оператор 2.1>;
    }
    else // для умови 2
    {
        <оператор 2.2>;
    }
    <оператор 1.2>;
    else
}
else // для умови 1
{
    <оператор 1.3>;
}
```

Приклад 9.3 Дана точка А з координатами x і y . Визначити який координатній чверті належить А.

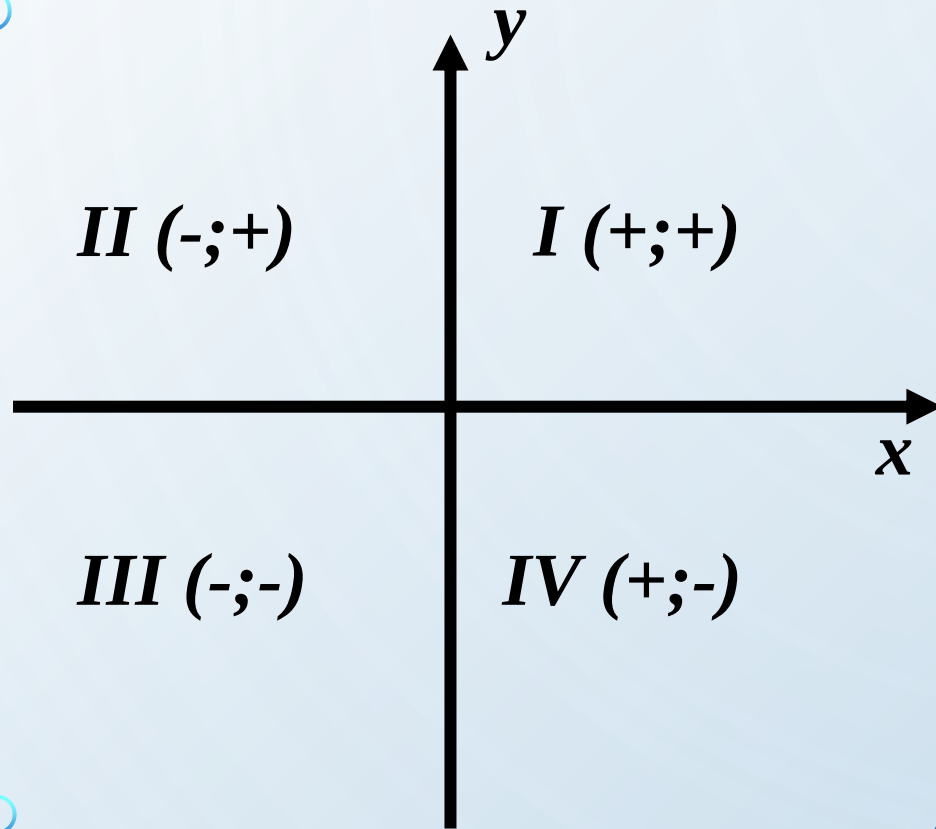


Рисунок 9.7

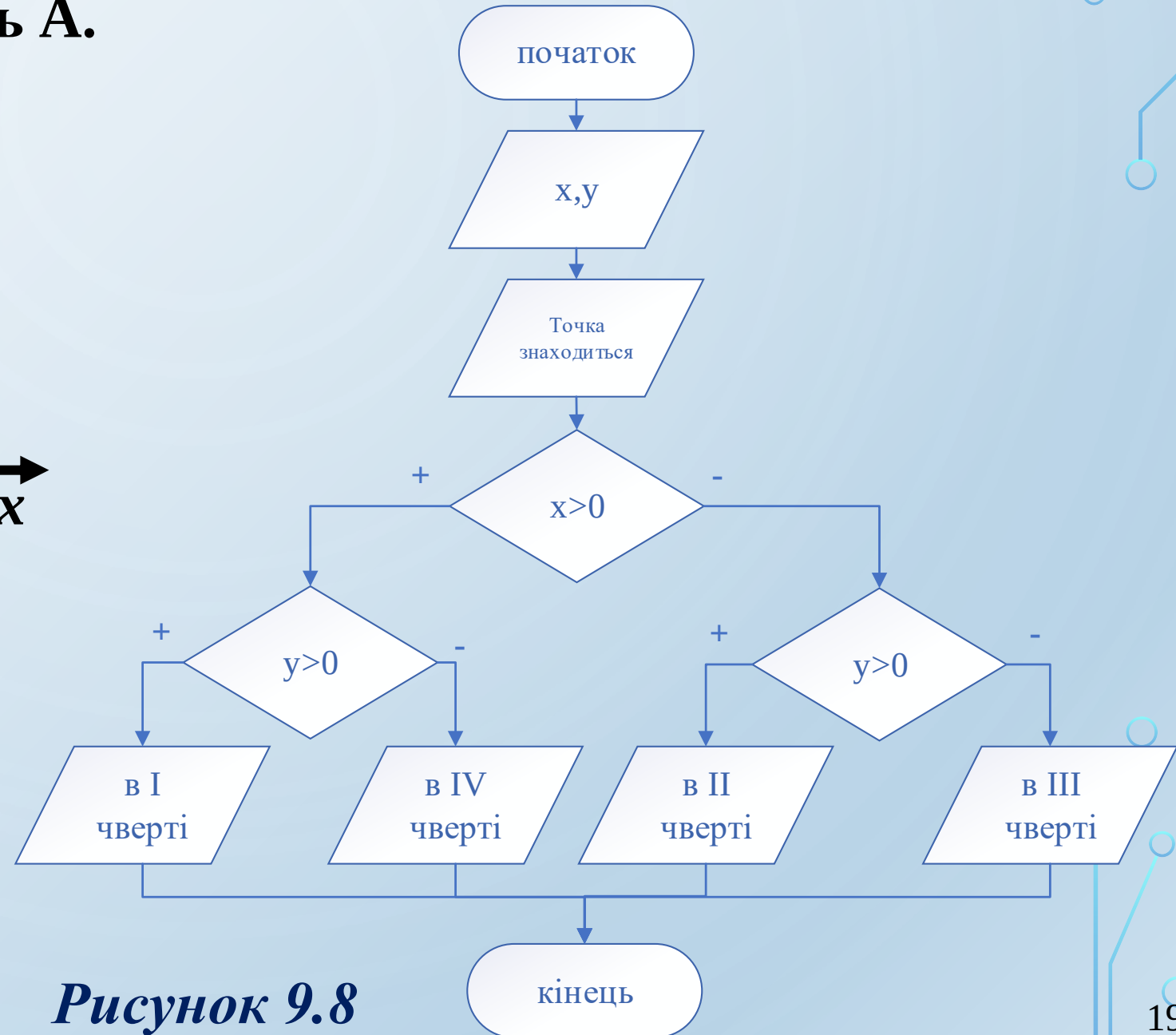
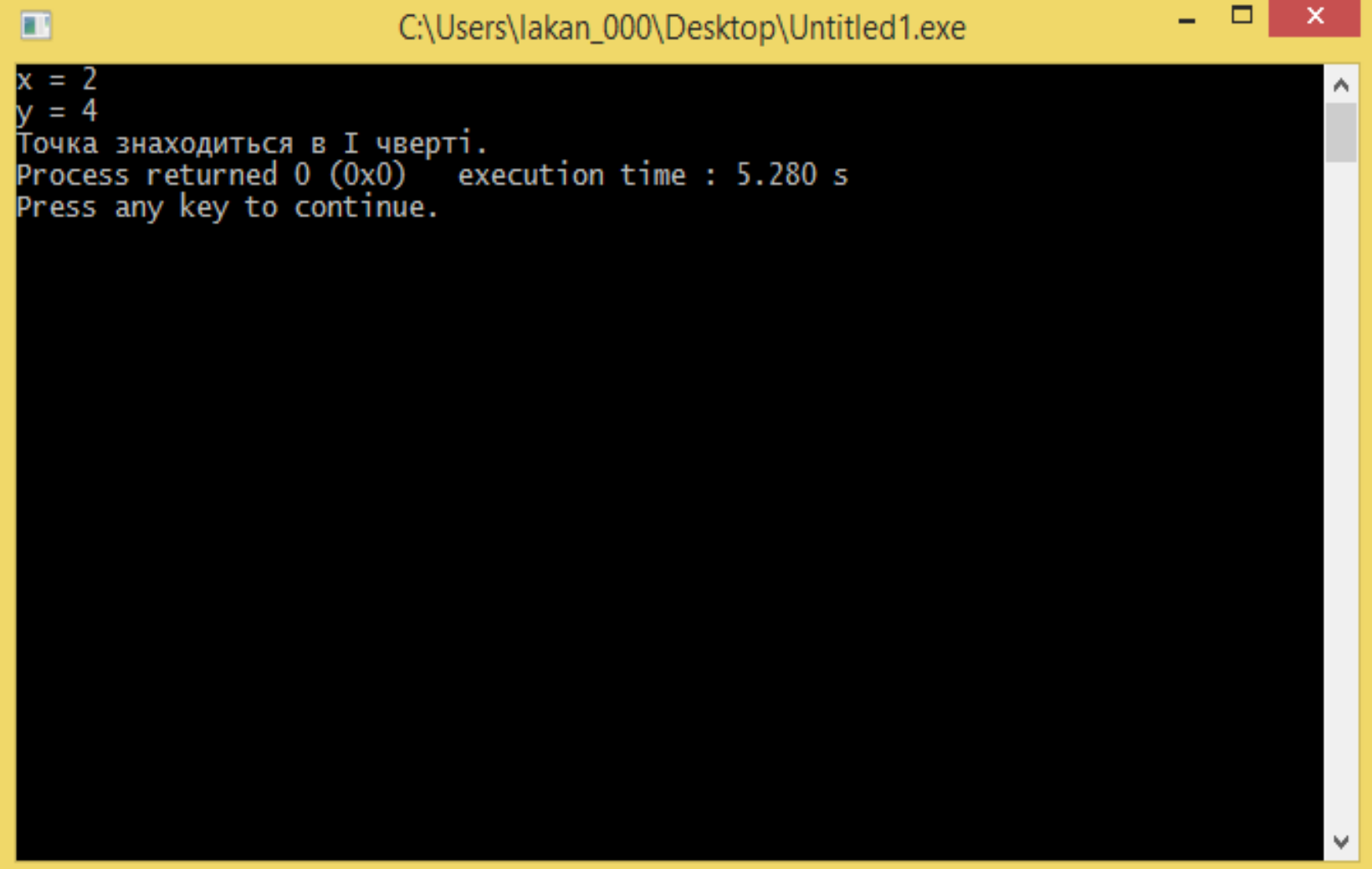


Рисунок 9.8

```
#include <stdio.h>
#include <windows.h>
int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    double x, y;
    printf("x = ");
    scanf("%lf",&x);
    printf("y = ");
    scanf("%lf",&y);
    printf("Точка знаходиться ");
    if (x > 0)
    {
        if (y > 0)
            printf("в I чверті.");
        else
            printf("в IV чверті.");
    }
    else
    {
        if (y > 0)
            printf("в II чверті.");
        else
            printf("в III чверті.");
    }
    return 0;
}
```



```
x = 2
y = 4
Точка знаходиться в I чверті.
Process returned 0 (0x0) execution time : 5.280 s
Press any key to continue.
```

Рисунок 9.9

Ланцюжок повних операторів if

```
if <умова1>  
    <оператор 1>  
else if <умова2>  
    <оператор 2>  
else if <умова3>  
    <оператор 2>  
  
.....  
  
else <оператор N>
```

Приклад 9.4 Обчислити значення функції $y(x)$ в залежності від введеного значення x .

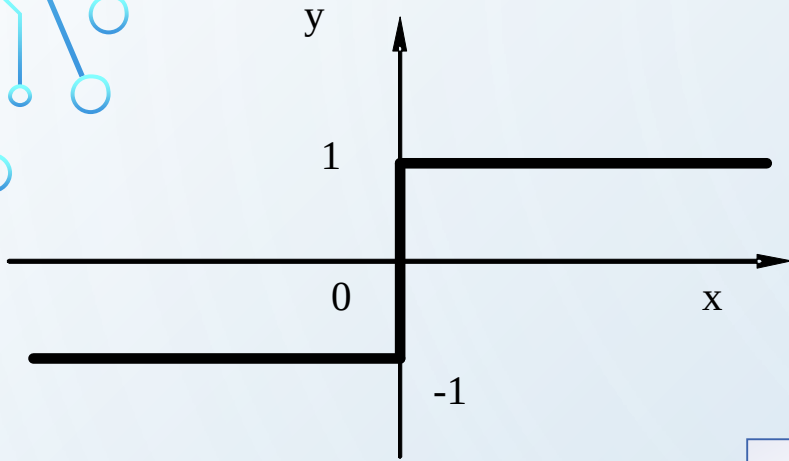


Рисунок 9.10

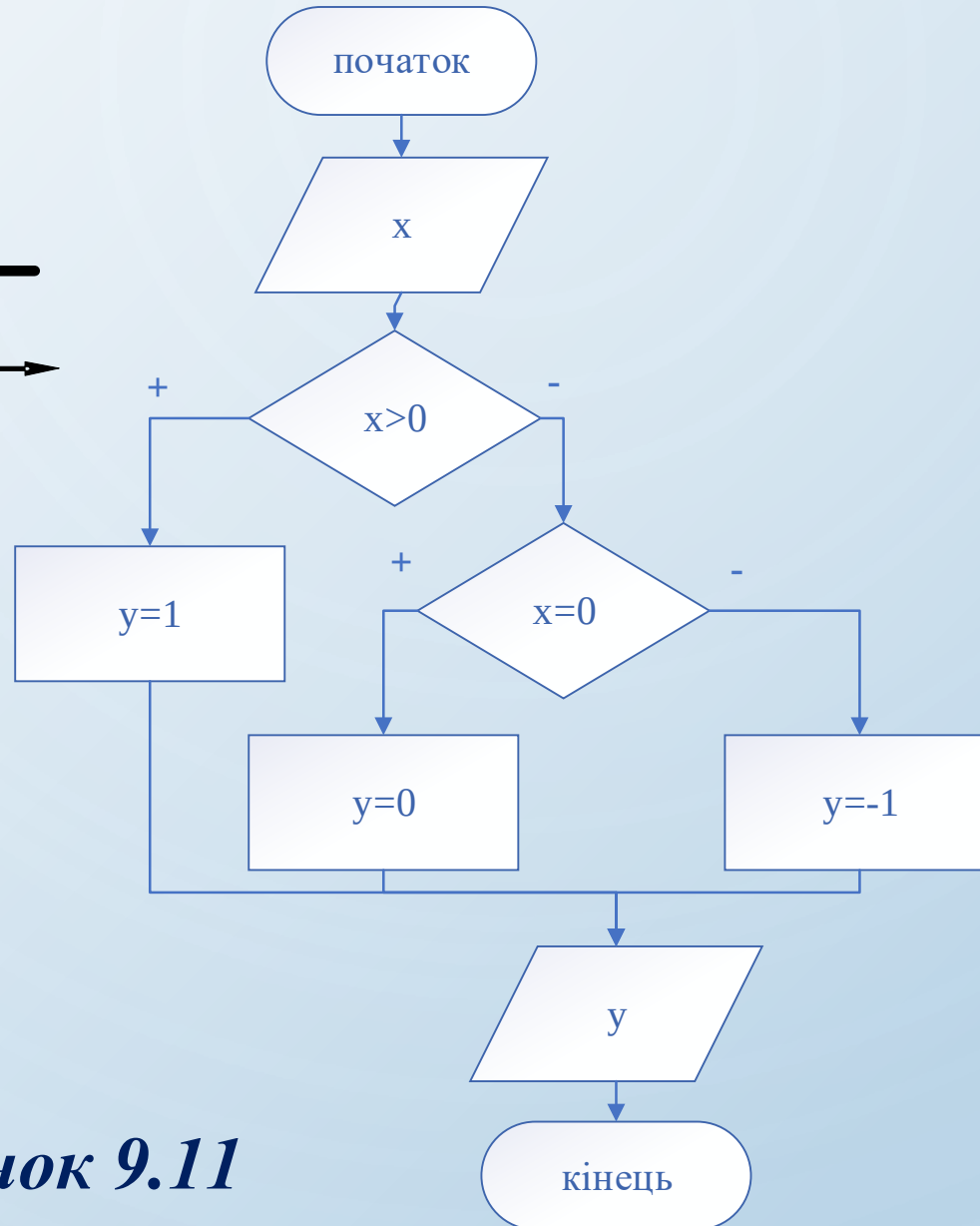


Рисунок 9.11

```
#include <stdio.h>
int main ()
{
    int x,y;
    printf("x = ");
    scanf("%d",&x);
    if(x>0)
        y=1;
    else
        if(x==0)
            y=0;
        else
            y=-1;
    printf("y=%d", y);
    return 0;
}
```

Складна умова – це умова, що складається з декілька простих умов, що пов'язані між собою логічними операціями:

! – НЕ (інверсія, заперечення);

&& - І (кон'юнкція, логічне множення);

|| - АБО (диз'юнкція, логічне складання).

Прості умови: <, <=, >, >=, ==, !=

Порядок виконання складних умов:

➤ вираз в дужках

➤ !

➤ <, <=, >, >=

➤ ==, !=

➤ &&

➤ ||

Приклад 9.5

```
      2      1      6      3      5      4
if (  !(a > b) || c != d && b == a)
{
    ...
}
```

Істинно або помилково при $a=2, b=3, c=4, d=5$

$!(a > b)$	1
$a < b \ \&\& \ b < c$	1
$!(a \geq b) \ \ c == d$	1
$a < c \ \ b < c \ \&\& \ b < a$	1
$a > b \ \ !(b < c)$	0

Для яких значень x істинно умови:

$x < 6 \ \&\& \ x < 10$
$x < 6 \ \&\& \ x > 10$
$x > 6 \ \&\& \ x < 10$
$x > 6 \ \&\& \ x > 10$
$x < 6 \ \ x < 10$
$x < 6 \ \ x > 10$
$x > 6 \ \ x < 10$
$x > 6 \ \ x > 10$

$(-\infty, 6)$	$x < 6$
$\emptyset$	
$(6, 10)$	
$(10, \infty)$	$x > 10$
$(-\infty, 10)$	$x < 10$
$(-\infty, 6) \cup (10, \infty)$	
$(-\infty, \infty)$	
$(6, \infty)$	$x > 6$

Приклад 9.6 Фірма набирає співробітників від 25 років до 40 років включно. Ввести вік людини і визначити підходить він фірмі чи ні (вівести відповідь "підходить" "не підходить").

Особливість: потрібно перевірити чи виконуються 2 умови одночасно.

Варіант 1

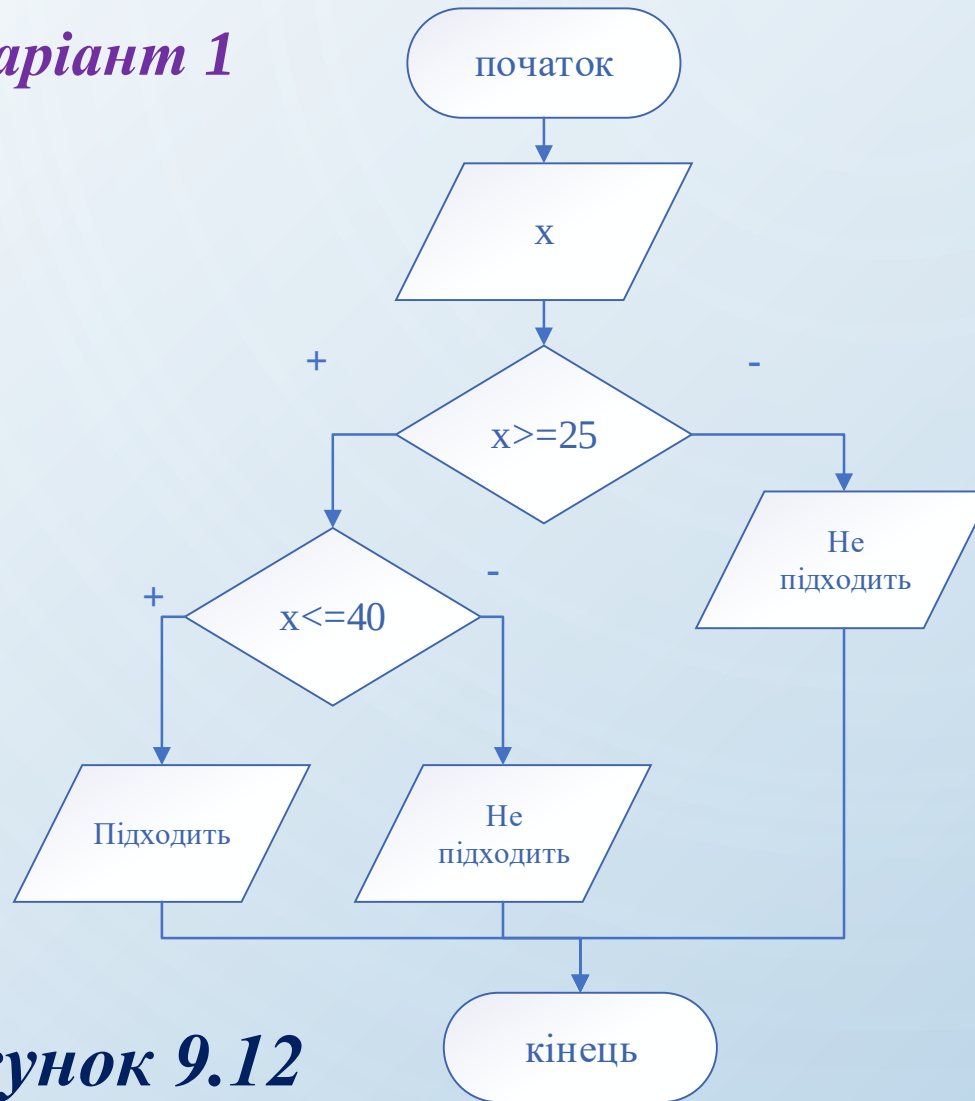
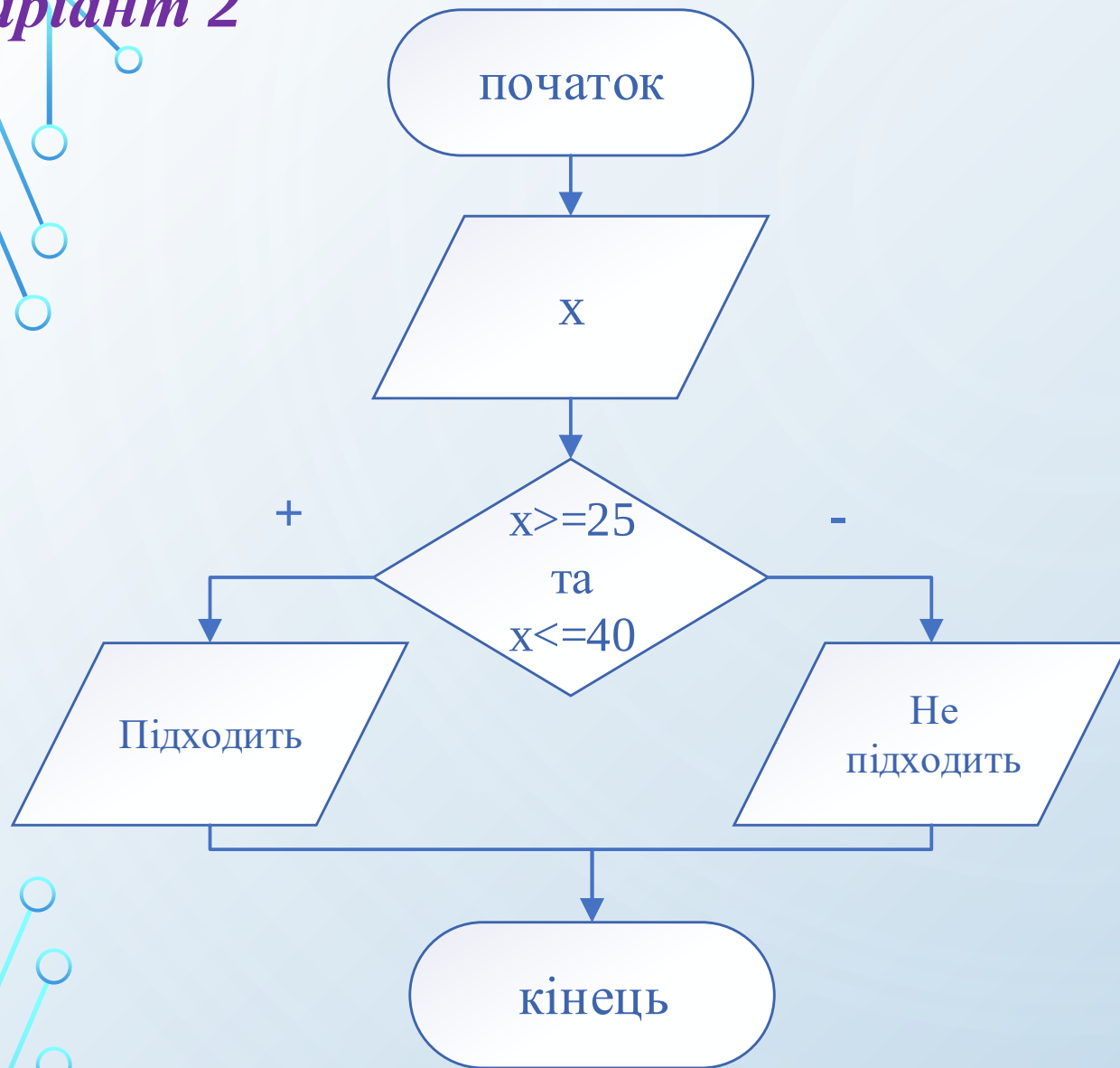


Рисунок 9.12

```
#include <windows.h>
#include <stdio.h>
int main ()
{
    int x;
    SetConsoleOutputCP(1251);
    printf("x = ");
    scanf("%d",&x);
    if(x >= 25)
    if(x <= 40)
        printf("Підходить");
    else
        printf("Не підходить");
    else
        printf("Не підходить");
    return 0;
}
```

```
#include <windows.h>
#include <stdio.h>
int main ()
{
    int x;
    SetConsoleOutputCP(1251);
    printf("x = ");
    scanf("%d",&x);
    if(x >= 25 && x <= 40)
        printf("Підходить");
    else
        printf("Не підходить");
    return 0;
}
```

Рисунок 9.13

Приклад 9.7 Дана точка А з координатами x і y . Визначити якій координатній чверті належить А.

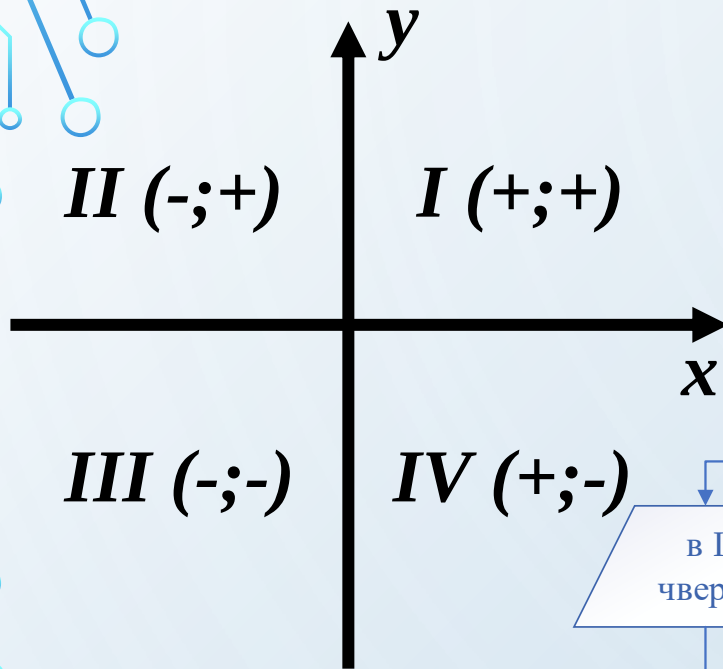


Рисунок 9.14

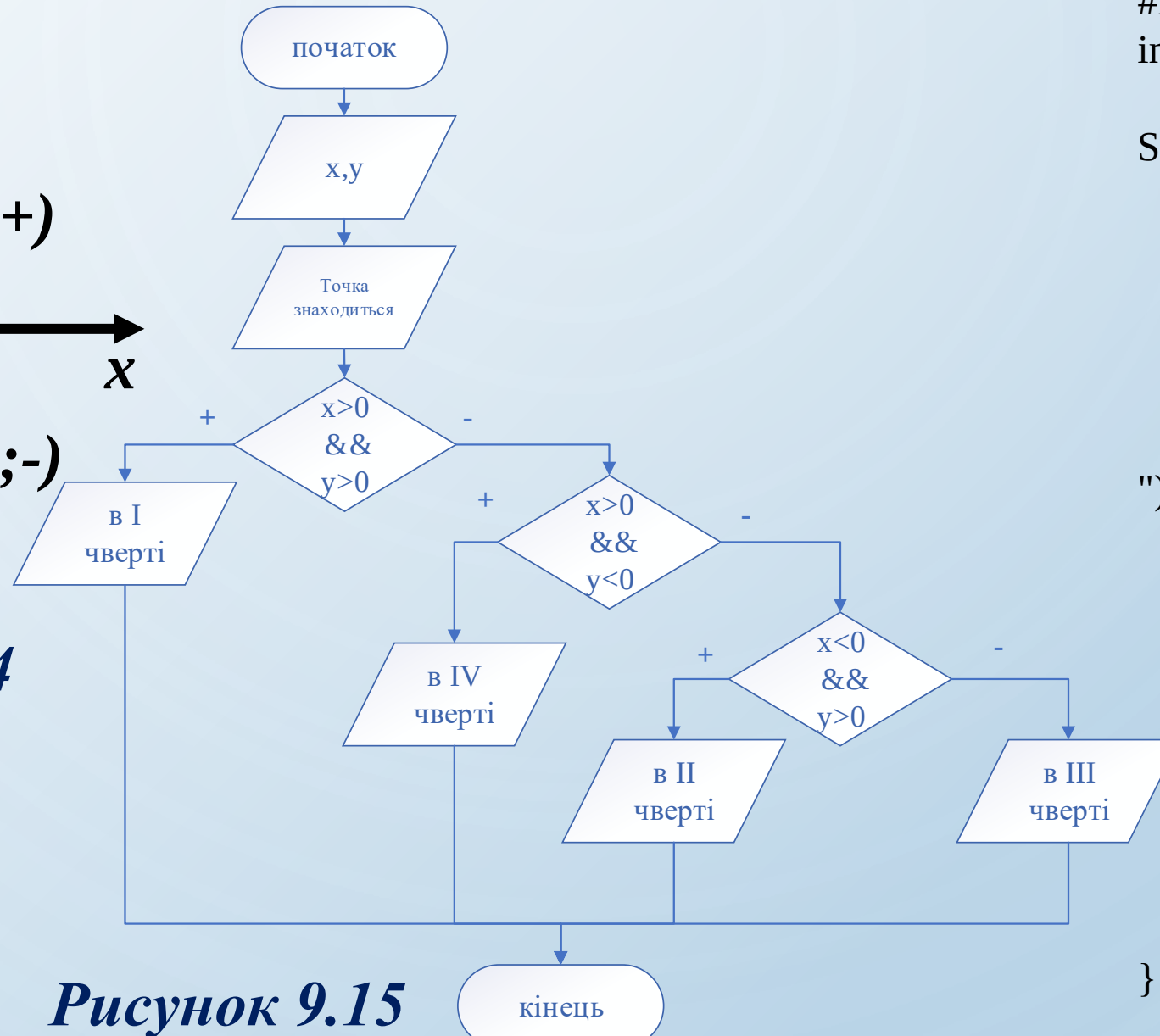


Рисунок 9.15

```
#include <windows.h>
#include <stdio.h>
int main() {
```

```
SetConsoleOutputCP(1251);
double x, y;
printf( "x = ");
scanf("%lf",&x);
printf( "y = ");
scanf("%lf",&y);
printf( "Точка знаходиться
");
```

```
if (x > 0 && y > 0)
    printf( "в I чверті.");
else if (x > 0 && y < 0)
    printf( "в IV чверті.");
else if (x < 0 && y > 0)
    printf("в II чверті.");
else
    printf( "в III чверті.");
return 0;
}
```

Оператор вибору switch

```
switch (<вираз>)  
{  
    case<константа1>:  
        <оператор1>; break;  
    case<константа2>:  
        <оператор2>; break;  
    .....  
    case< константа n>:  
        <оператор n>; break;  
default:  
    <оператор n+1>;  
}
```

Порядок виконання оператора switch:

- спочатку обчислюється значення виразу;
- далі, отримане значення послідовно порівнюється зі значеннями констант;
- якщо значення виразу збігається з однією із міток, то виконується оператор цього варіанту;

Опція **break**; здійснює вихід з оператора **switch** і перехід до наступного за ним оператора. При відсутності опції **break** будуть виконуватися всі оператори, починаючи з позначеного даної міткою і закінчуючи оператором в опції **default**.

- якщо значення виразу не збігається з жодною із міток, то виконується оператор після слова default, якщо ця частина присутня.

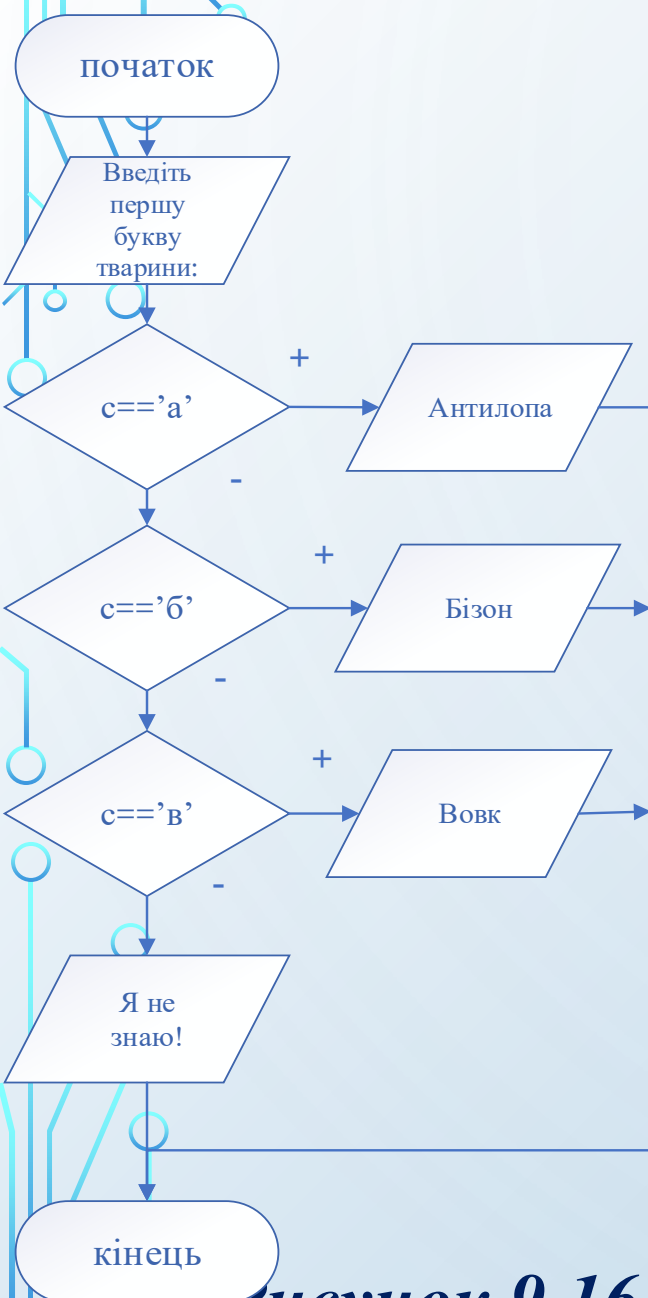
Особливості:

- об'єкт порівняння з міткою може бути: ім'я змінної, константа, арифметичний вираз, покажчик функції цілого типу, символьного типу або типу **enum**;
- константи можуть мати лише типи **int, char, enum**;
- кілька "міток" **case** з різними значеннями констант можуть позначати один і той же оператор.

case 1: case 3: case 5: оператор;

case 2: case 4: case 6: оператор;

Приклад 9.8 Ввести букву і вивести назву тварини на цю букву.



```
#include <windows.h>
#include <stdio.h>
int main ()
{
    char c;
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    printf("Введіть першу букву назви тварини:");
    scanf("%c",&c);
    switch(c)
    {
        case 'а': printf("Антилопа"); break;
        case 'б': printf("Бізон"); break;
        case 'в': printf("Вовк"); break;
        default: printf("Я не знаю!");
    }
    return 0;
}
```

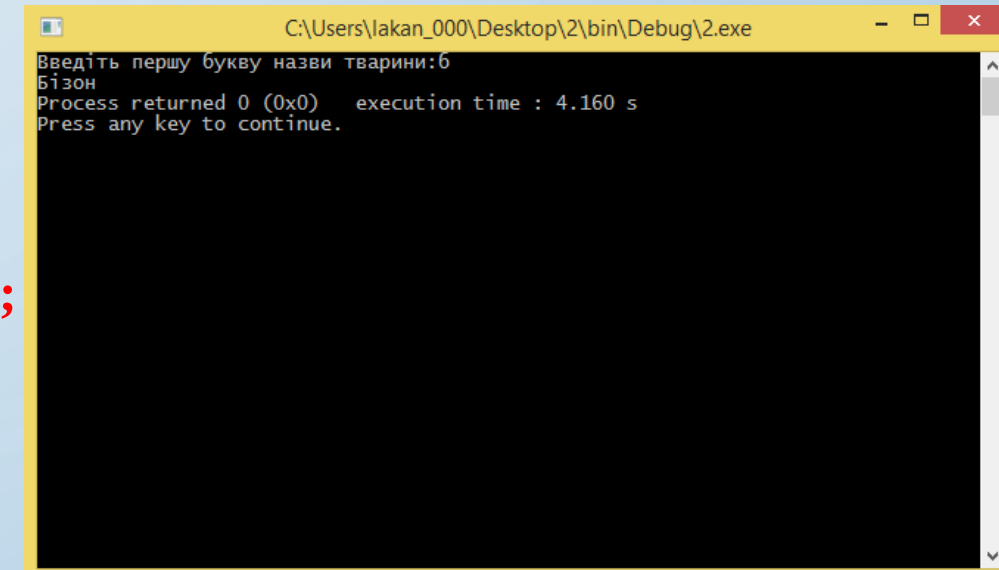


Рисунок 9.17

Приклад 9.9 Ввести номер місяця і вивести кількість днів у цьому місяці.

Кількість днів в місяцях:

28 днів – 2 (лютий).

30 днів – 4 (квітень), 6 (червень), 9 (вересень), 11 (листопад).

31 день – 1 (січень), 3 (березень), 5 (травень), 7 (липень), 8 (серпень), 10 (жовтень), 12 (грудень).

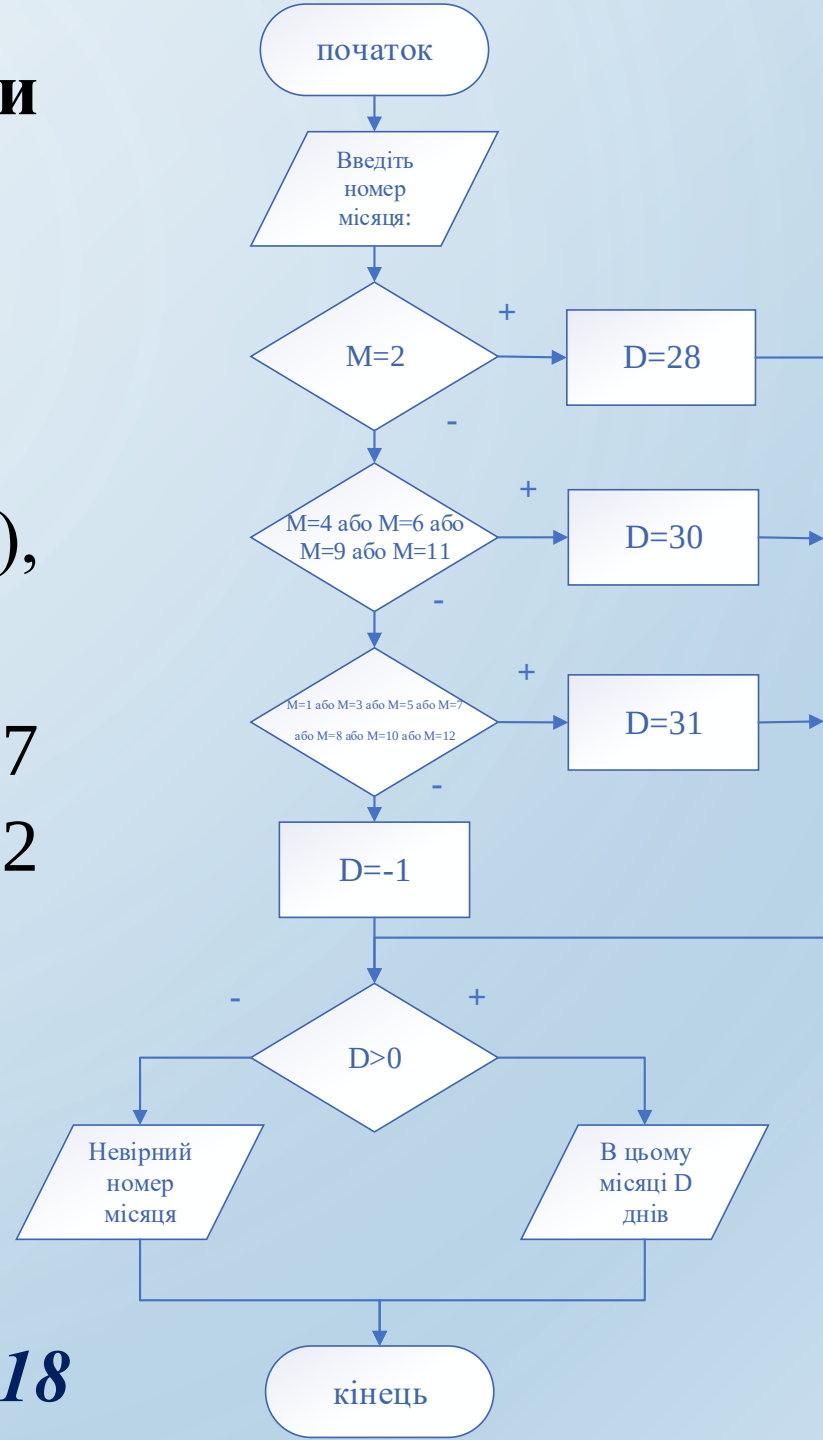
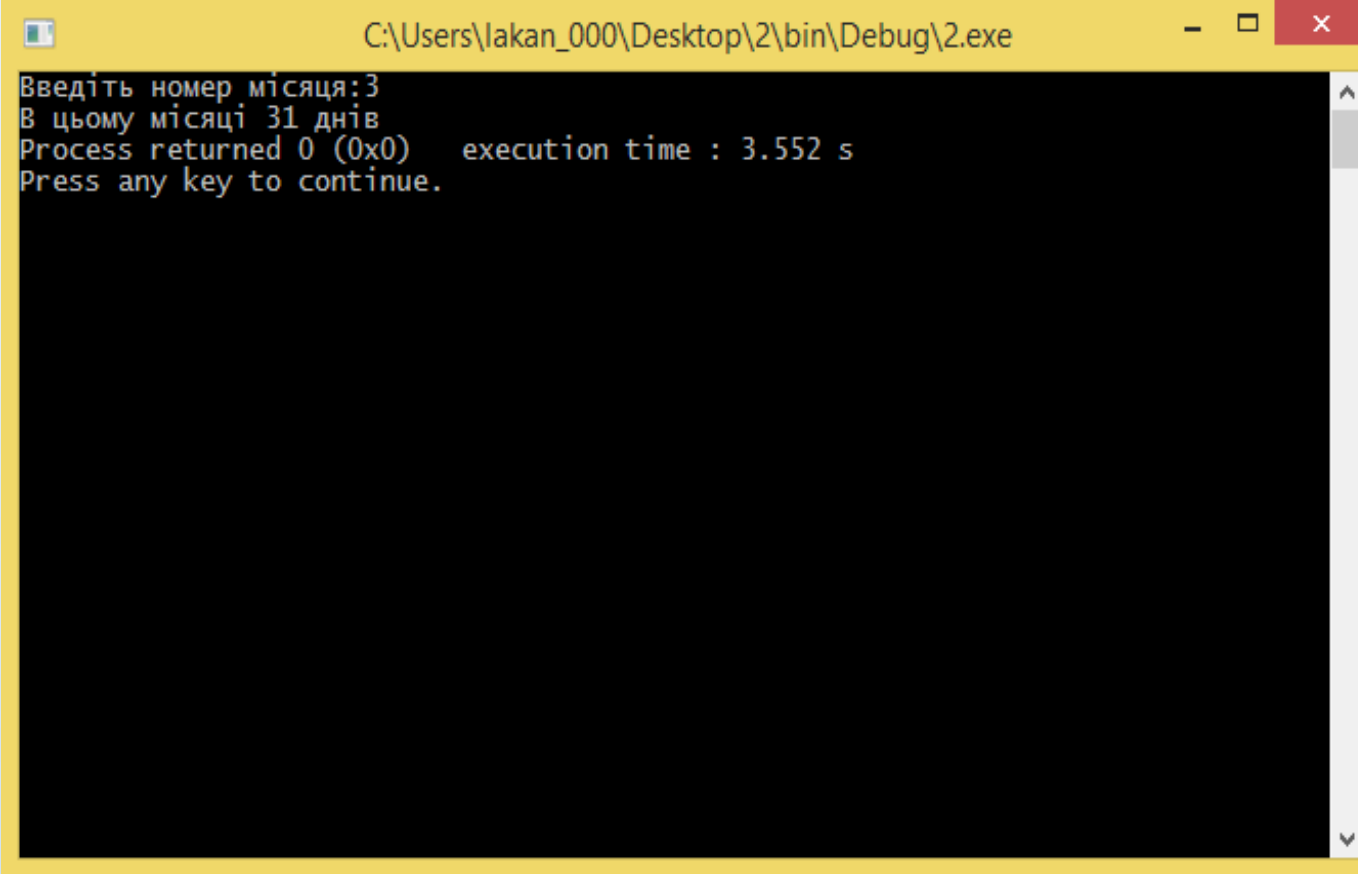


Рисунок 9.18

```
#include <windows.h>
#include <stdio.h>
int main ()
{
    int M,D;
    SetConsoleOutputCP(1251);
    printf("Введіть номер місяця:");
    scanf("%d",&M);
    switch(M)
    {
        case 2: D=28; break;
        case 4: case 6: case 9: case 11: D=30;
        break;
        case 1: case 3: case 5: case 7: case 8: case
        10:
        case 12: D=31; break;
        default: D=-1;
    }
    if(D>0)
        printf("В цьому місяці %d днів", D);
    else
        printf("Невірний номер місяця");
    return 0;
}
```



```
C:\Users\lakan_000\Desktop\2\bin\Debug\2.exe
Введіть номер місяця:3
В цьому місяці 31 днів
Process returned 0 (0x0) execution time : 3.552 s
Press any key to continue.
```

Рисунок 9.19

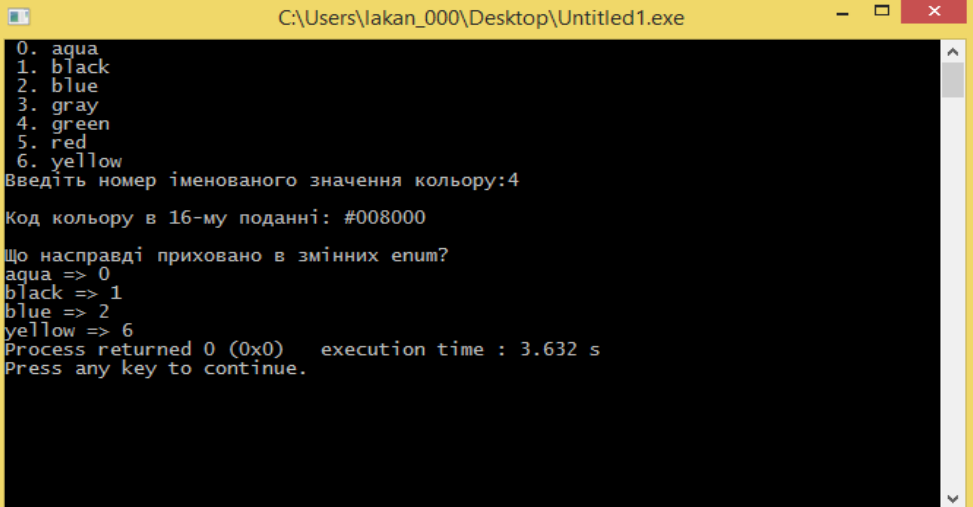
Приклад 9.10 Реалізувати виведення коду кольору в шістнадцятковому поданні.

```
#include <stdio.h>
#include <windows.h>
enum Color {aqua, black, blue, gray, green, red,
yellow};
int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    int r;
    printf (" 0. aqua\n 1. black\n 2. blue\n 3. gray\n 4.
green\n 5. red\n 6. yellow\n");
    printf ("Введіть номер іменованого значення
кольору:");
    scanf("%d",&r);
    printf ("\nКод кольору в 16-му поданні: ");
    switch(r)
    {
        case aqua   : printf ("#00FFFF"); break;
        case black  : printf ("#000000"); break;
        case blue   : printf ("#0000FF"); break;
        case gray   : printf ("#808080"); break;
```

```
        case green : printf ("#008000"); break;
        case red    : printf ("#FF0000"); break;
        case yellow : printf ("#008000"); break;
        default : printf ("\nУ наборі немає
кольору з таким ім'ям!");
    }
    printf ( "\n\nЩо насправді приховано в
змінних enum?\n");
    printf ("aqua => %d ",aqua);
    printf ( "\nblack => %d ",black);
    printf ("\nblue => %d ",blue);
    printf ("\nyellow => %d ", yellow);

    return 0;
}
```

Рисунок 9.20



```
C:\Users\lakan_000\Desktop\Untitled1.exe
0. aqua
1. black
2. blue
3. gray
4. green
5. red
6. yellow
Введіть номер іменованого значення кольору:4

Код кольору в 16-му поданні: #008000

Що насправді приховано в змінних enum?
aqua => 0
black => 1
blue => 2
yellow => 6
Process returned 0 (0x0)   execution time : 3.632 s
Press any key to continue.
```

9.2 ОПЕРАТОРИ ЦИКЛУ: WHILE, DO-WHILE, FOR

Алгоритм, в якому є послідовність операцій (група операторів), яка повинна бути виконана декілька разів, називається **циклічним**, а сама послідовність операцій іменується **тілом циклу**.

Оператори: **while**, **do...while** і **for**

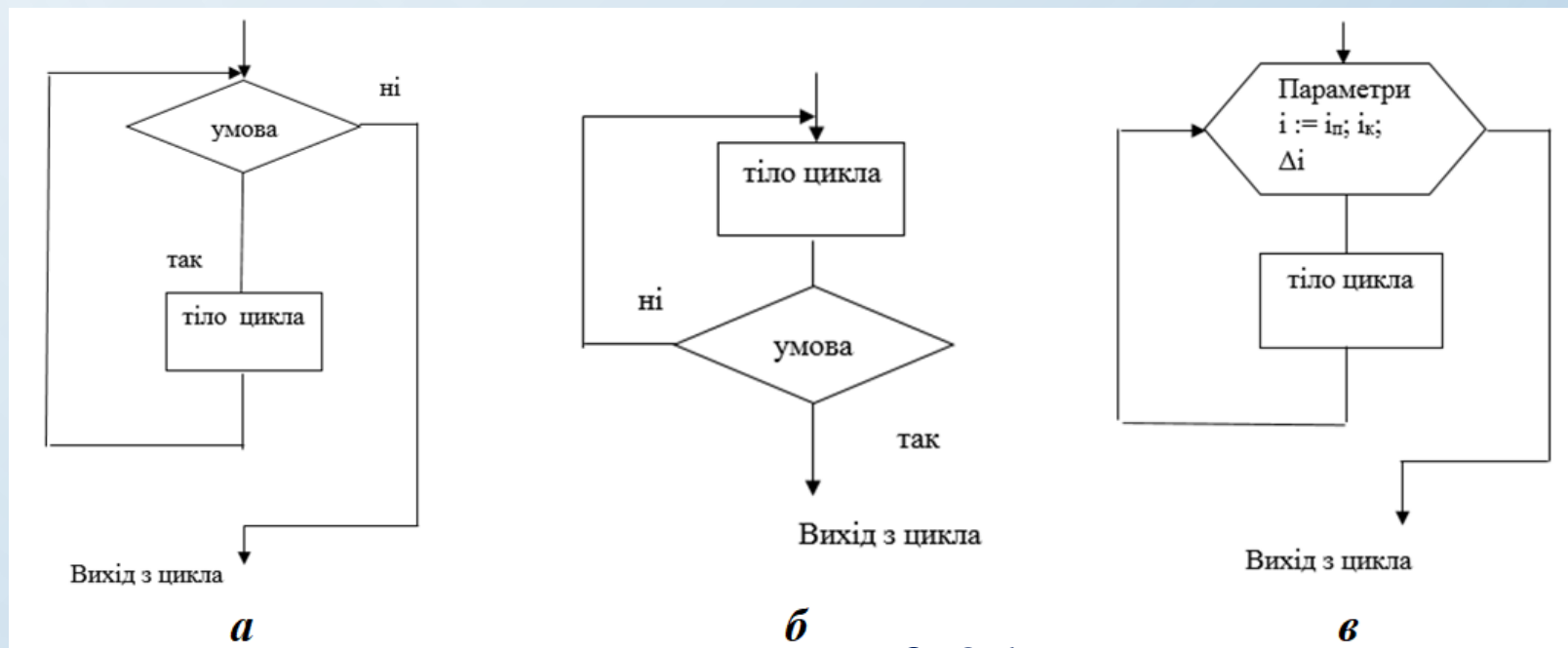


Рисунок 9.21

Оператор *while*

```
while(умова виконання)  
{  
    оператори тіла циклу;  
}
```

На рис. 9.22 <умова виконання> - це вираз логічного типу, що визначає умову за якої виконуються <оператора тіла циклу>.

Загалом, оператор **while** виконується у такий **спосіб**:

- обчислюється значення виразу <умова виконання>;
- якщо значення виразу <умова виконання > дорівнює **false** або 0, тобто умова не виконується, виконання <операторів тіла циклу > припиняється;.
- якщо значення виразу <умова виконання> дорівнює **true** або не 0 (умова виконується), то виконуються <оператора тіла циклу >, розташовані між фігурними дужками;
- після цього знову все повторюється.

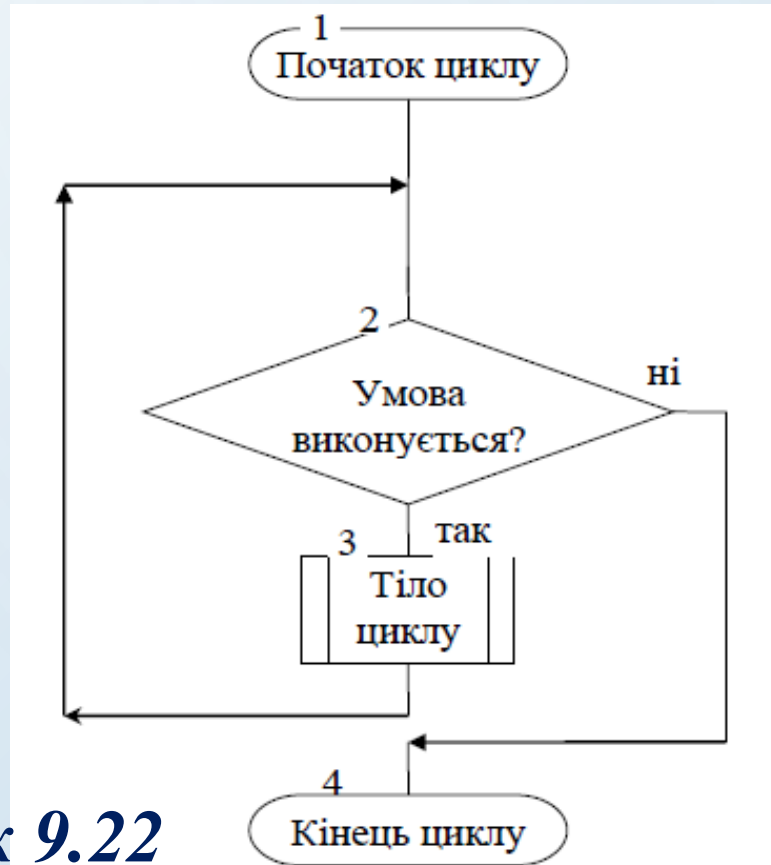


Рисунок 9.22

Приклад 9.11 Порахувати суму чисел від 1 до введеного k.

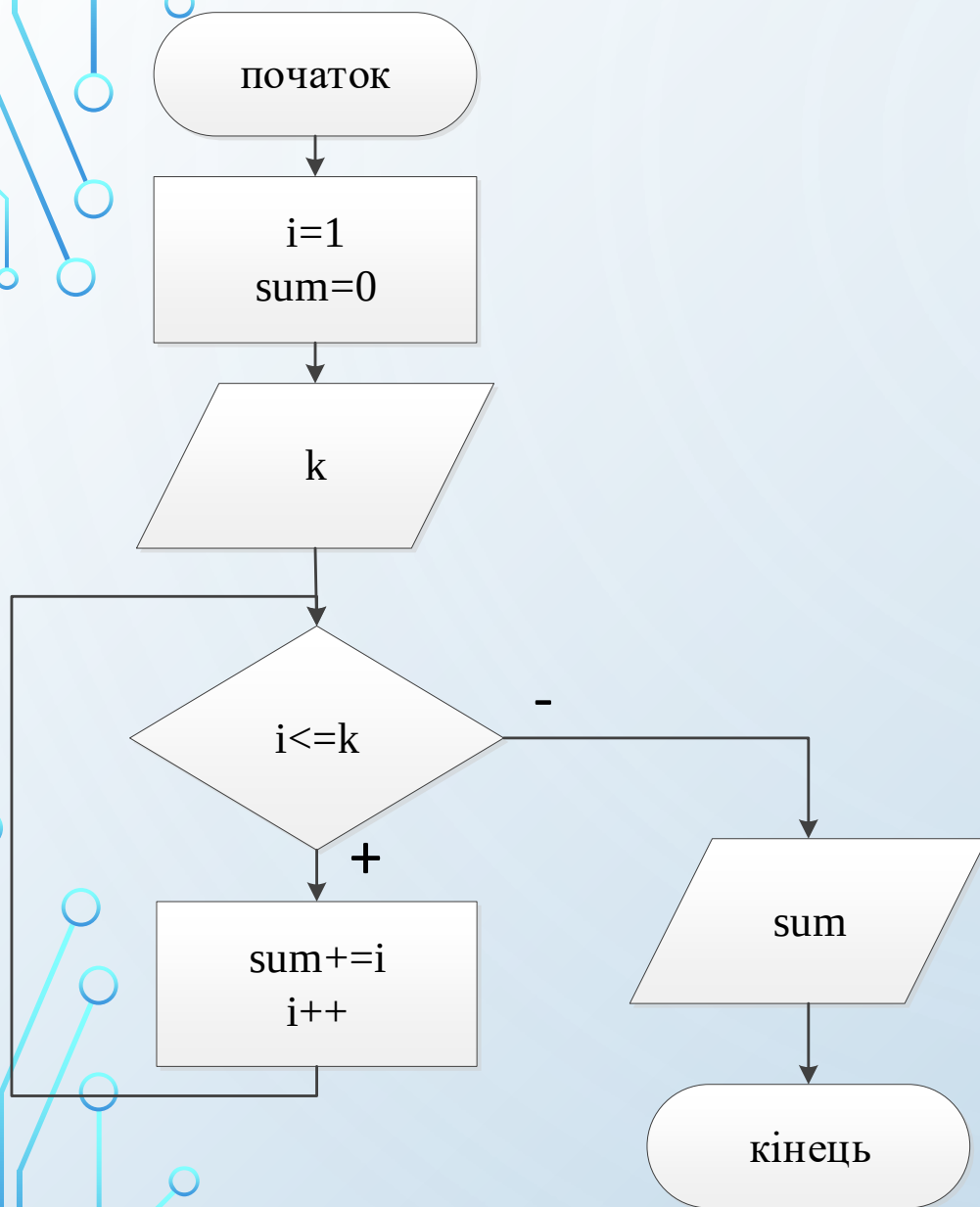


Рисунок 9.23

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

```
int main()
```

```
{
```

```
    int k; // оголошуємо цілу змінну key
```

```
    int i = 1;
```

```
    int sum = 0; // початкове значення суми дорівнює 0
```

```
    printf("k = ");
```

```
    scanf("%d", &k); // вводимо значення змінної k
```

```
    while (i <= k) // поки i менше або дорівнює k
```

```
    {
```

```
        sum += i; // додаємо значення i до суми
```

```
        i++; // збільшуємо на 1
```

```
    }
```

```
    printf("sum = %d\n", sum); // виведення значення суми
```

```
    return 0;
```

```
}
```

The screenshot shows a Windows command prompt window titled "C:\Users\lakan_000\Desktop\1.exe". The output of the program is displayed as follows:

```
k = 10
sum = 55

Process returned 0 (0x0)   execution time : 1.039 s
Press any key to continue.
```

Рисунок 9.24

Приклад 9.12 Знайдіть найбільшу цифру даного натурального числа.

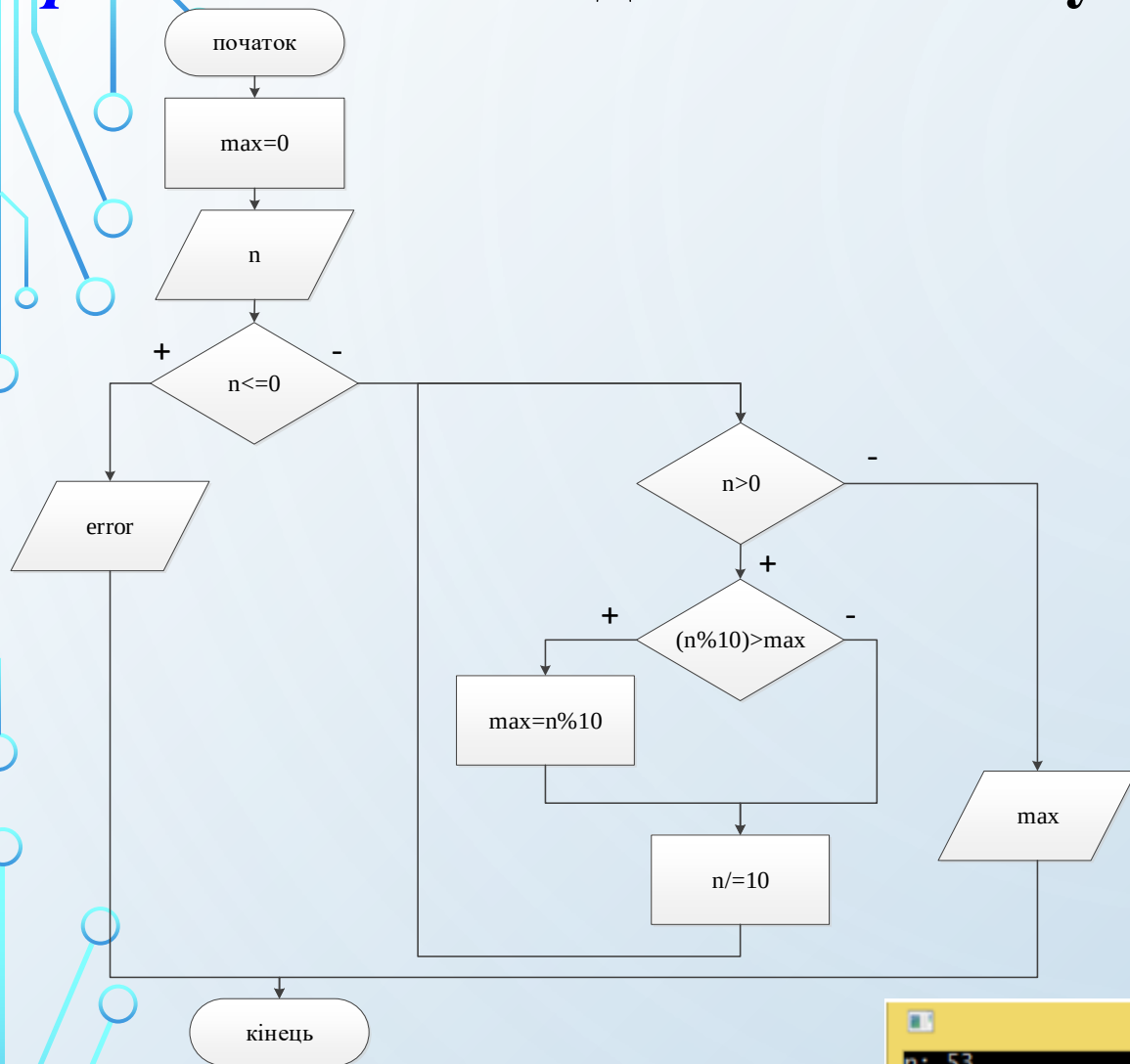


Рисунок 9.25

```
1
#include <stdio.h>
int main()
{
    int n;
    int max=0;
    printf("n: ");
    scanf ("%d",&n);
    if (n<=0)
        printf("error n");
```

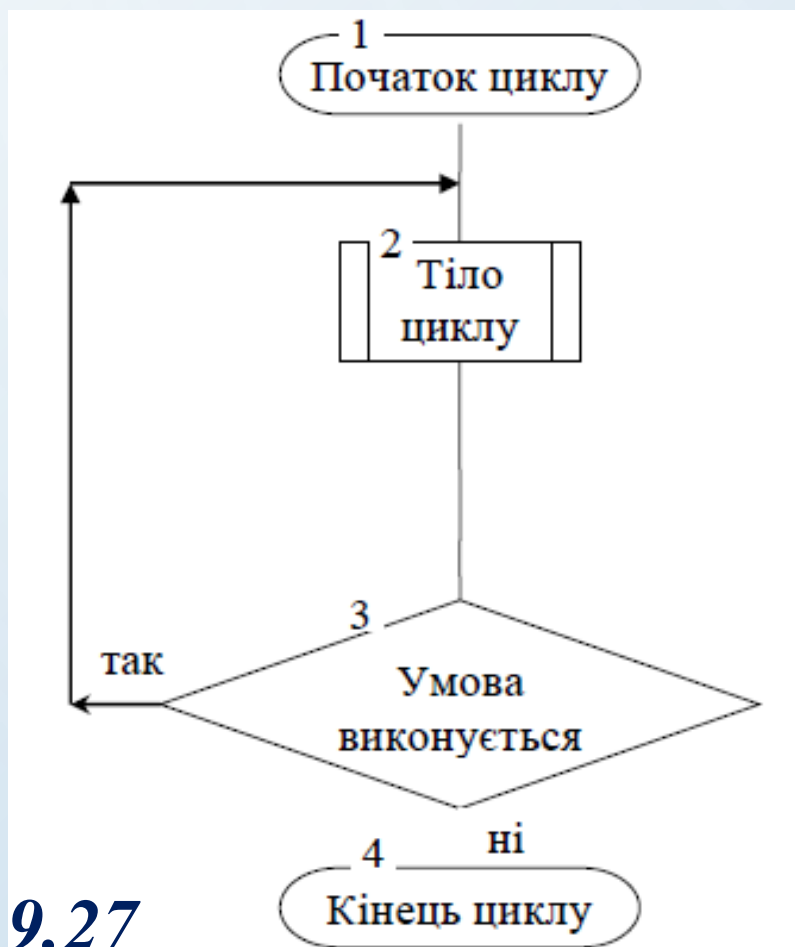
```
2
else
{
    while (n>0)
    {
        if ((n%10)>max)
            max=n%10;
        n/=10;
    }
    printf("max:%d",max);
    return 0;
}
```

C:\Users\lakan_000\Desktop\1.exe
n: 53
max:5
Process returned 0 (0x0) execution time : 9.589 s
Press any key to continue.

Рисунок 9.26

Оператор *do...while*

```
do{  
    оператори тіла циклу  
} while(умова виконання);
```



На рис. 9.27 *<умова повтору тіла циклу>* - це вираз логічного типу, що визначає умову за якої будуть знов виконані *<оператора тіла циклу>*.

Цикл виконується таким чином:

- спочатку виконуються *<оператори тіла циклу>*, розташовані між фігурними дужками;
- потім обчислюється значення виразу *<умова повтору тіла циклу>*.
- якщо значення виразу *<умова повтору тіла циклу>* дорівнює **true** або не 0 (умова виконується), то знову виконуються *<оператори тіла циклу>*, розташовані між фігурними дужками;
- якщо значення виразу *<умова виконання>* дорівнює **false** або 0, тобто умова не виконується, виконання *<операторів тіла циклу>* припиняється.

Приклад 9.13 Перевірка, що користувач ввів число від 0 до 10.



Рисунок 9.28

C:\Users\lakan_000\Desktop\1.exe
Введіть число від 0 до 10: 34
Введіть число від 0 до 10: 12
Введіть число від 0 до 10: 4
Ведено число 4
Process returned 0 (0x0) execution time : 5.816 s
Press any key to continue.

Рисунок 9.29

```
#include <stdio.h>
#include <stdlib.h> // для використання функції system
()
int main()
{
    int num;          // оголошуємо цілу змінну для числа
    system("chcp 1251"); // переходимо на кирилицю в
консолі
    system("cls");     // очищаємо екран
    do {
        printf("Введіть число від 0 до 10: "); //
запрошення користувачеві
        scanf("%d", &num); // введення числа
    } while ((num < 0) || (num > 10)); // повторюємо цикл
поки num<0 або num>10
    printf("Ведено число %d", num); // виводимо введенне
значення num - от 0 до 10
    return 0;
}
```

Приклад 9.14 Вивести на екран всі цілі ступені числа 2 від 0 до n.

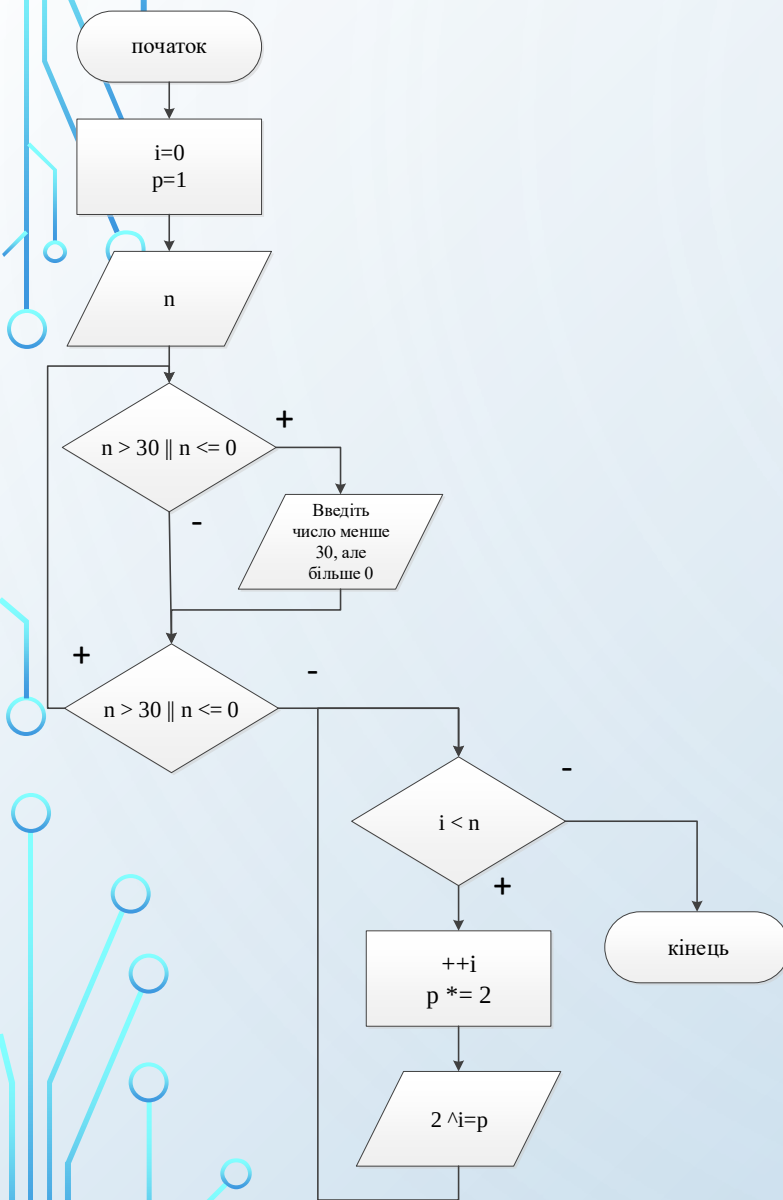


Рисунок 9.30

```
#include <stdio.h>
#include <stdlib.h> // для
використання функції system
()
int main()
{
system("chcp 1251");
int n, p = 1, i = 0;
```

```
do {
printf("\n n = ");
scanf("%d",&n);
if (n > 30 || n <= 0)
printf("Введіть
число менше 20, але більше
0");
} while (n > 30 || n <= 0);
```

```
while (i < n)
{
++i;
p *= 2;
printf("2 ^%d =
%d\n",i,p);
}
return 0;
}
```

C:\Users\lakan_000\Desktop\44

Active code page: 1251

n= 45
Введіть число менше 20, але більше 0
n= 60
Введіть число менше 20, але більше 0
n= 3
2 ^1 = 2
2 ^2 = 4
2 ^3 = 8

Process returned 0 (0x0) execution time : 6.930 s
Press any key to continue.

Рисунок 9.31

Оператор циклу *for*

```
for(<вираз1>; <вираз2>; <вираз3>)  
    <тіло циклу>;
```

< *вираз1* > - це вираз ініціалізації, який встановлює початкові значення змінних циклу; <*вираз2*> - це вираз умови, що задає умову виконання тіла циклу; <*вираз3*> - це вираз ітерації, який виконує зміну значень змінних циклу; <*тіло циклу*> - це оператор або блок, який задає дії, що мають повторюватися.

Оператор циклу *for* виконується наступним чином:

- Обчислюється < вираз1 >, внаслідок чого змінні циклу приймають початкові значення. Цей вираз обчислюється тільки один раз на початку роботи циклу.
- Обчислюється <вираз2>, що є умовою продовження виконання циклу.
- Якщо умова хибна, то цикл закінчується.
- Виконується тіло циклу.
- Обчислюється < вираз3 >, який використовують для зміни параметрів циклу.
- Виконання продовжується з пункту 2.

Приклад 9.15 Складіть програму, що виводить на екран квадрати чисел від 2 до 5 включно.

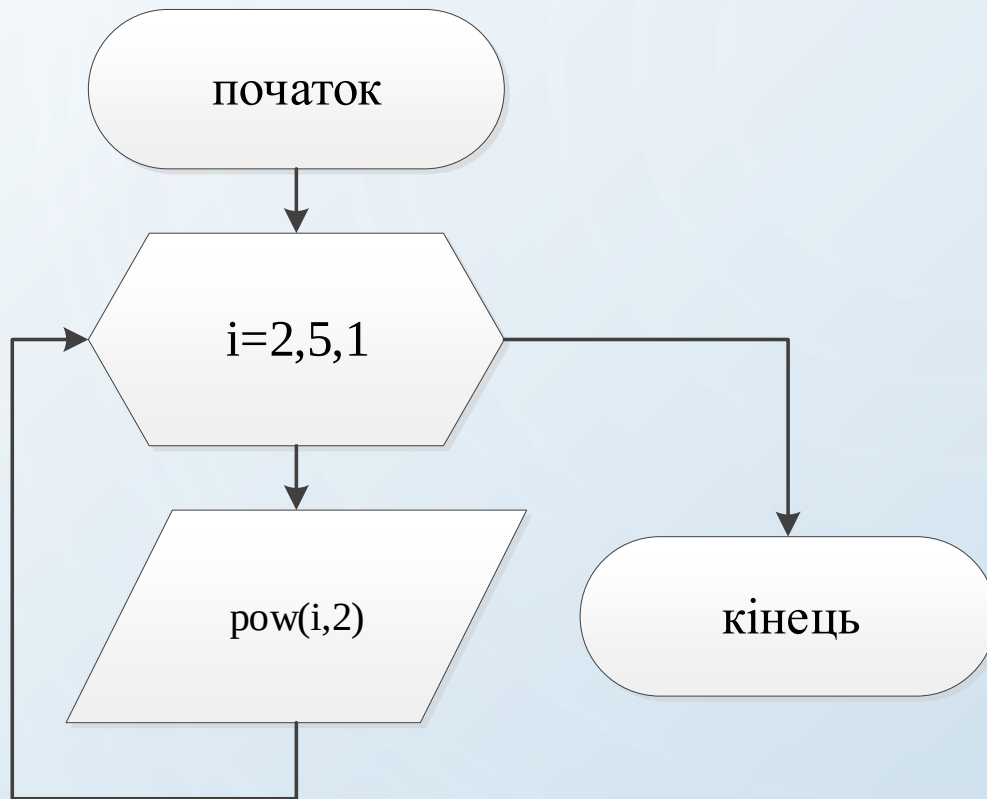


Рисунок 9.32

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

```
int main()
{
    double i;
    for (i=2; i<=5; i++)
        printf("\n%lf",pow(i,2));
    return 0;
}
```

The screenshot shows a Windows command prompt window with a yellow title bar. The title bar text is 'C:\Users\lakan_000\Desktop\1.exe'. The black command prompt area displays the following output: '4.000000', '9.000000', '16.000000', '25.000000', followed by 'Process returned 0 (0x0) execution time : 0.010 s' and 'Press any key to continue.' on the final line.

Рисунок 9.33

Приклад 9.16 Скласти програму, що запитує у користувача два числа A і B і виводить суму всіх чисел між ними (включно).

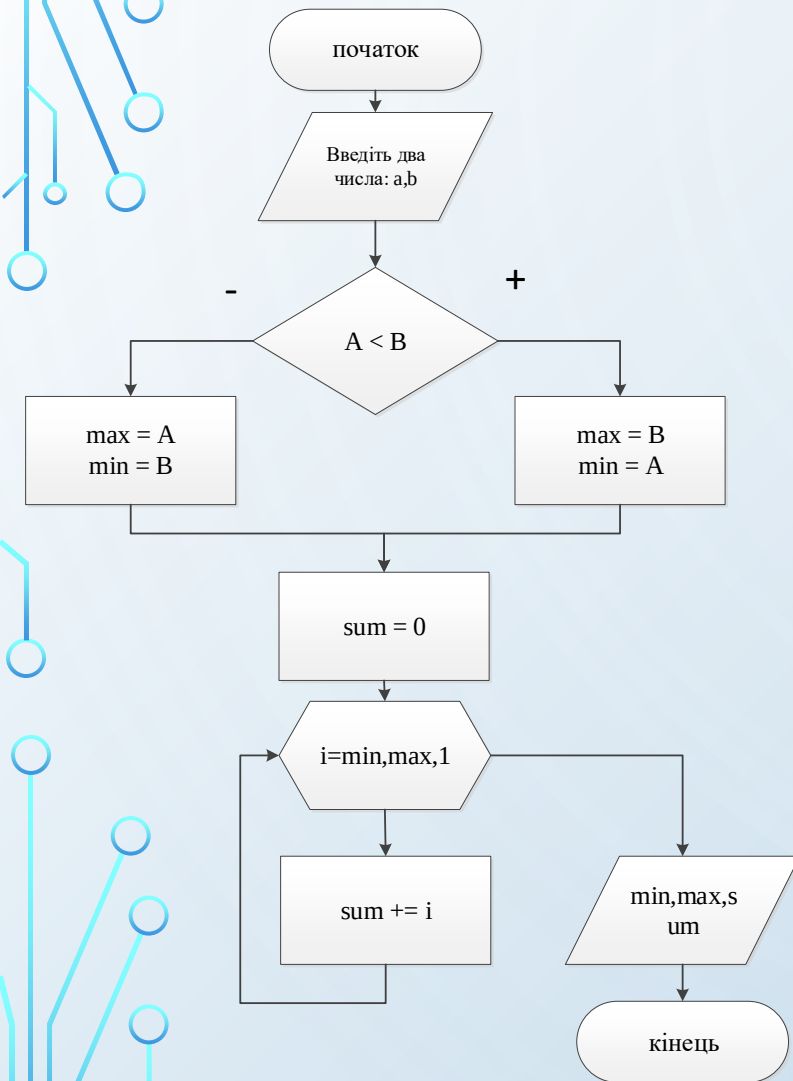


Рисунок 9.34

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    system("chcp 1251");
    int A, B, sum;
    int min, max;
    printf("Введіть два числа:\n ");
    scanf("%d %d", &A, &B);
    if (A < B)
    {
        max = B; min = A;
    }
```

```
else
{
    max = A; min = B;
}
sum = 0;
for (int i = min; i <= max; i++)
sum += i;
printf("Сума всіх чисел від %d до %d = %d\n", min, max, sum);
return 0;
}
```

```
C:\Users\lakan_000\Desktop\1.exe
Active code page: 1251
Введіть два числа:
4
2
Сума всіх чисел від 2 до 4 = 9
Process returned 0 (0x0) execution time : 6.717 s
Press any key to continue.
```

Рисунок 9.35

Особливості використання циклу for

безкінечний цикл

```
int fact(n)
{
    int f=1;
    int i = 1;
    for( ; ; )
    {
        f*=i;
        if(i>=n) return f;
        i++;
    }
}
```

цикл має тільки вираз перевірки умови

```
int x;
printf ("Введіть ціле число\n ");
scanf("%d",&x);
int sum=0;
for( ; x!=0; )
{
    sum+=x%10;
    x/=10;
}
printf ("/nСума цифр у числі
дорівнює %d",sum");
```

декілька параметрів циклу

```
int fact1 (n)
{
    int f=1;
    for( int i = 1; i<=n; f*=i, i++ );
    return f;
}
```

Вкладені цикли

```
for (i = 0; i<n; i++) // зовнішній цикл - Цикл1
{
    for (j = 0; j<n; j++) // вкладений цикл - Цикл2
    {
        ; // блок операцій Цикла2
    }
    ; // блок операцій Цикла1
}
```


Приклад 9.17 Вивести числа від 0 до 99, по 10 в кожному рядку.

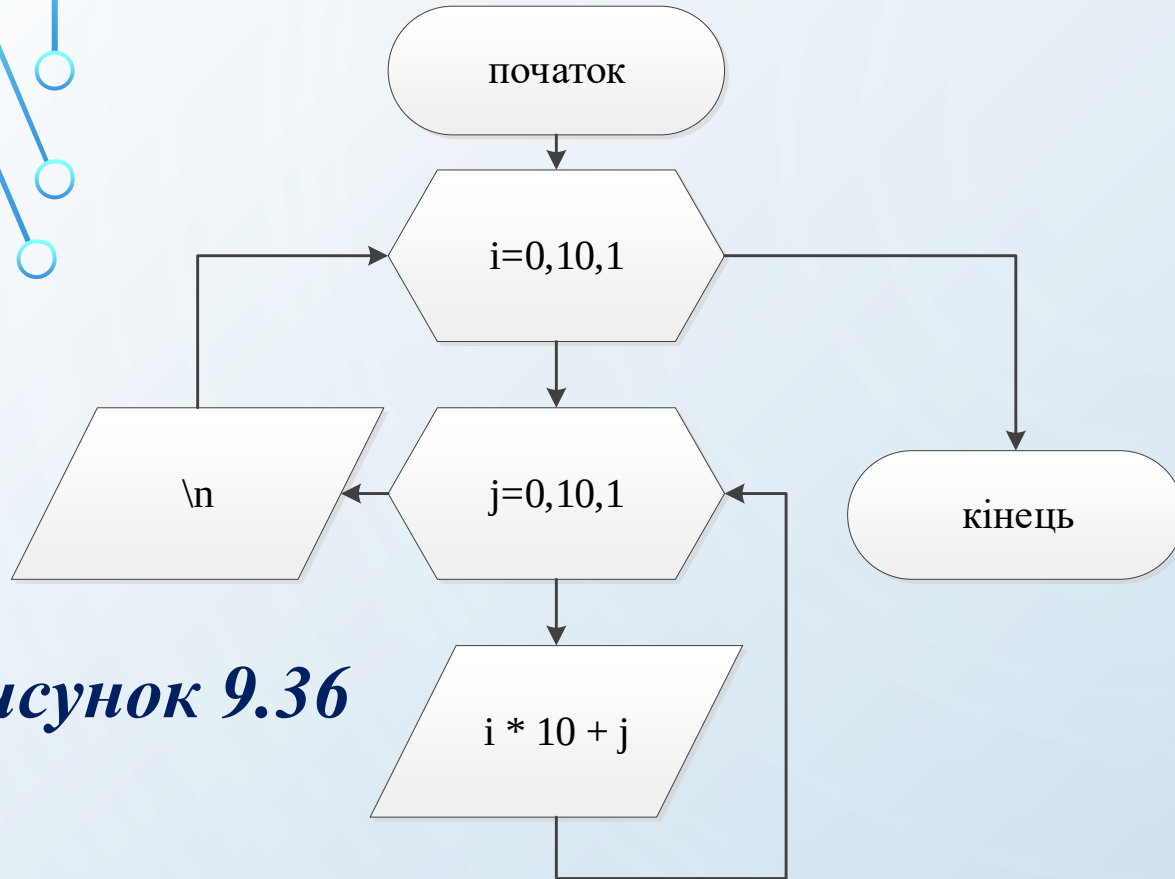


Рисунок 9.36

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

```
int main()
```

```
{
```

```
    for(int i=0; i<10; i++) // цикл для  
    десятків
```

```
    {
```

```
        for (int j = 0; j < 10; j++) // цикл для  
        одиниц
```

```
        {
```

```
            printf("%2d ", i * 10 + j); // виводимо  
            розраховано число
```

```
        }
```

```
        printf("\n"); // в зовнішньому циклі  
        переводимо рядок
```

```
    }
```

```
    return 0;
```

```
}
```

The screenshot shows a Windows command prompt window with the title 'C:\Users\lakan_000\Desktop\1.exe'. The output displays numbers from 0 to 99 arranged in 10 rows of 10 numbers each. At the bottom, it shows 'Process returned 0 (0x0) execution time : 0.032 s' and 'Press any key to continue.'

Рисунок 9.37

Приклад 9.18 Дано ціле число. Розкласти число на прості множники.

```
#include <stdio.h>
int main()
{
    int n;
    printf("n = ");
    scanf("%d",&n);
    int k = n;
    for (int i = 2; i <= n / 2; i++)
    {
        while (k % i == 0)
        {
            k /= i;
            printf(" %d ",i);
        }
    }
    return 0;
}
```

A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Users\lakan_000\Desktop\1.exe'. The window has a black background with white text. The text displayed is: 'n = 34' on the first line, '2 17' on the second line, 'Process returned 0 (0x0) execution time : 3.940 s' on the third line, and 'Press any key to continue.' on the fourth line. The cursor is positioned at the end of the fourth line.

Рисунок 9.38

Рекомендації з вибору циклу

- При виборі циклу необхідно оцінити необхідність перевірки умови при вході в цикл або по завершенні проходження циклу.
 - ❖ Цикл з післяумовою зручно застосовувати у випадках, коли для перевірки умови потрібно обчислити значення виразу, яке потім буде розміщено в тілі циклу.
 - ❖ Цикл с передумовою використовується в разі якщо всі змінні, які беруть участь у виразі, що перевіряє умову, проініціалізовані заздалегідь, але точне число повторень циклу невідомо чи передбачається складна модифікація змінних, що беруть участь у формуванні умови повторення циклу.
- Якщо цикл орієнтований на роботу з параметром, для якого заздалегідь відомо кількість повторень і крок зміни, то більш привабливим є параметричний цикл. Дуже зручно використовувати параметричний цикл при роботі з масивами для перебору елементів.

9.3 ОПЕРАТОР ПЕРЕХОДУ GOTO

```
goto <мітка>;  
.....  
<мітка>: <оператор>
```

Особливості:

- Оператор goto передає управління безпосередньо оператору, що позначено міткою.
- Область дії мітки обмежується функцією, в якій вона визначена, тому не можна передати управління оператору, що знаходиться в іншій функції.
- Ім'я мітки повинно бути унікальним.

Випадки використання:

- в тілі програми (деякої функції) для організації переходу між операторами;
- в операторі switch для переходу до однієї з гілок case.

Приклад 9.19 Організувати циклічний процес обчислення суми елементів послідовності за допомогою оператора goto. Кінець послідовності - число 0.

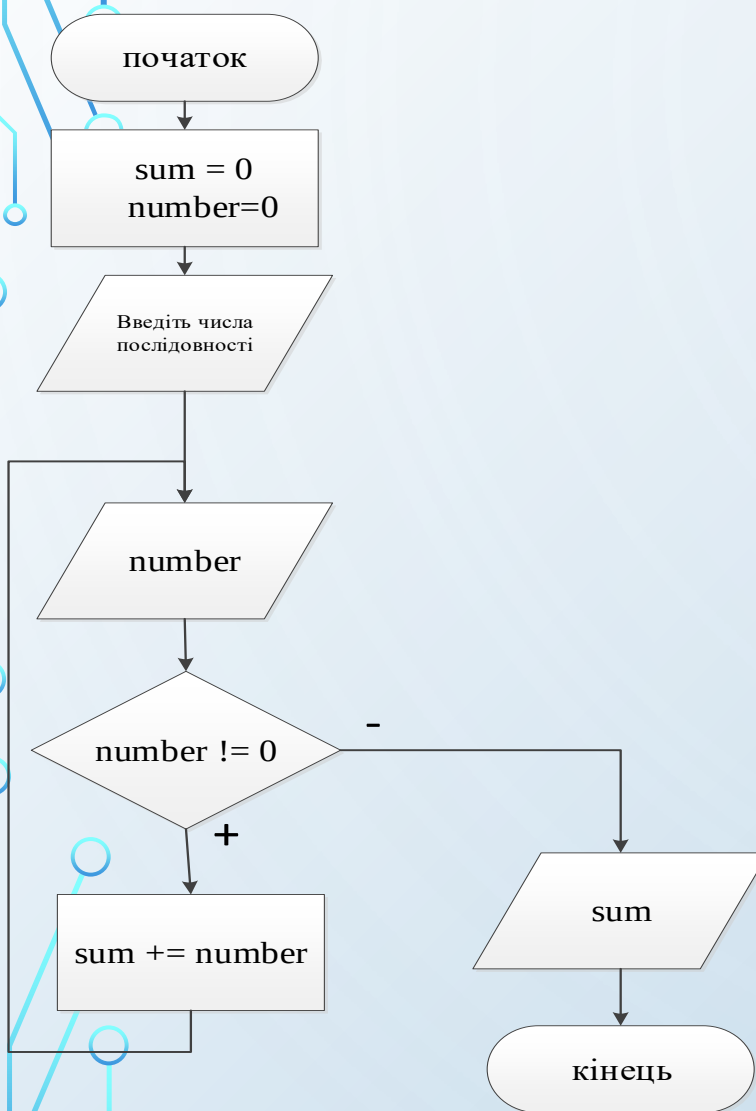


Рисунок 9.39

```
1
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    system("chcp 1251");
    double sum = 0;
    double number=0;
    printf ("Введіть числа
послідовності:\n");
```

```
C:\Users\Iakan_000\Desktop\1.exe
Active code page: 1251
Введіть числа послідовності:
2
4
7
0
sum = 13.000000
Process returned 0 (0x0)   execution time : 45.639 s
Press any key to continue.
```

Рисунок 9.40

```
2
// мітка, символізує початок
блоку, що повторюється
М:
    scanf(" %lf",&number);
    if (number != 0)
    {
        sum += number;
        goto М; // перехід до
мітки М
    }
    printf("sum = %lf", sum);
    return 0;
}
```


Переривання циклу

Оператор **break** призначений для штучного переривання виконання:

- послідовності ітерацій в операторах циклу for, while або do-while;
- послідовності операторів в операторі вибору switch.

Оператор **continue** призначений для переходу до виконання наступної ітерації циклу. Якщо в тілі циклу зустрічається оператор continue, то:

- виконання поточної ітерації зупиняється;
- відбувається перехід до наступної ітерації циклу.

Оператор continue може бути застосований у всіх видах циклів: for, while, do-while.

Приклад 9.20 Дано натуральне число n. Визначити, чи є воно простим.

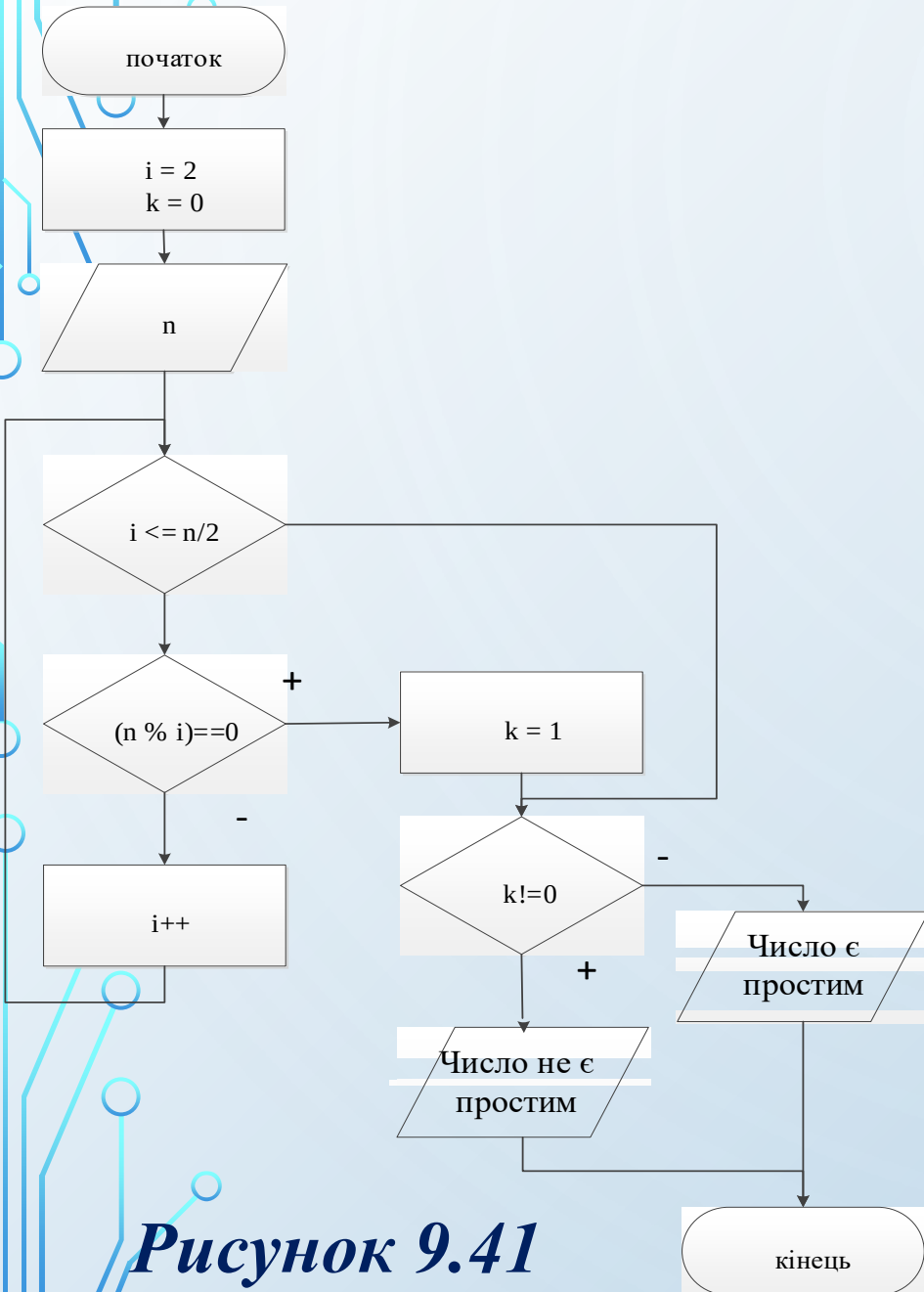


Рисунок 9.41

```
#include <stdio.h>
#include <windows.h>
int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    int n, i = 2, k = 0;
    printf("n = ");
    scanf("%d",&n);
    while (i <= n/2)
    {
        if ((n % i) == 0)
        {
            k = 1;
            break;
        }
        i++;
    }
    printf("Число %d - %s",n,(k? "не є простим":"просте"));
    return 0;
}
```

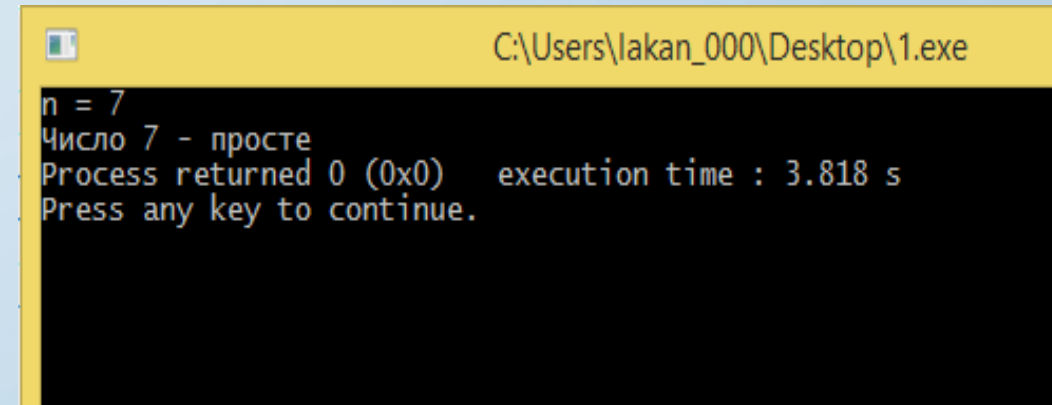
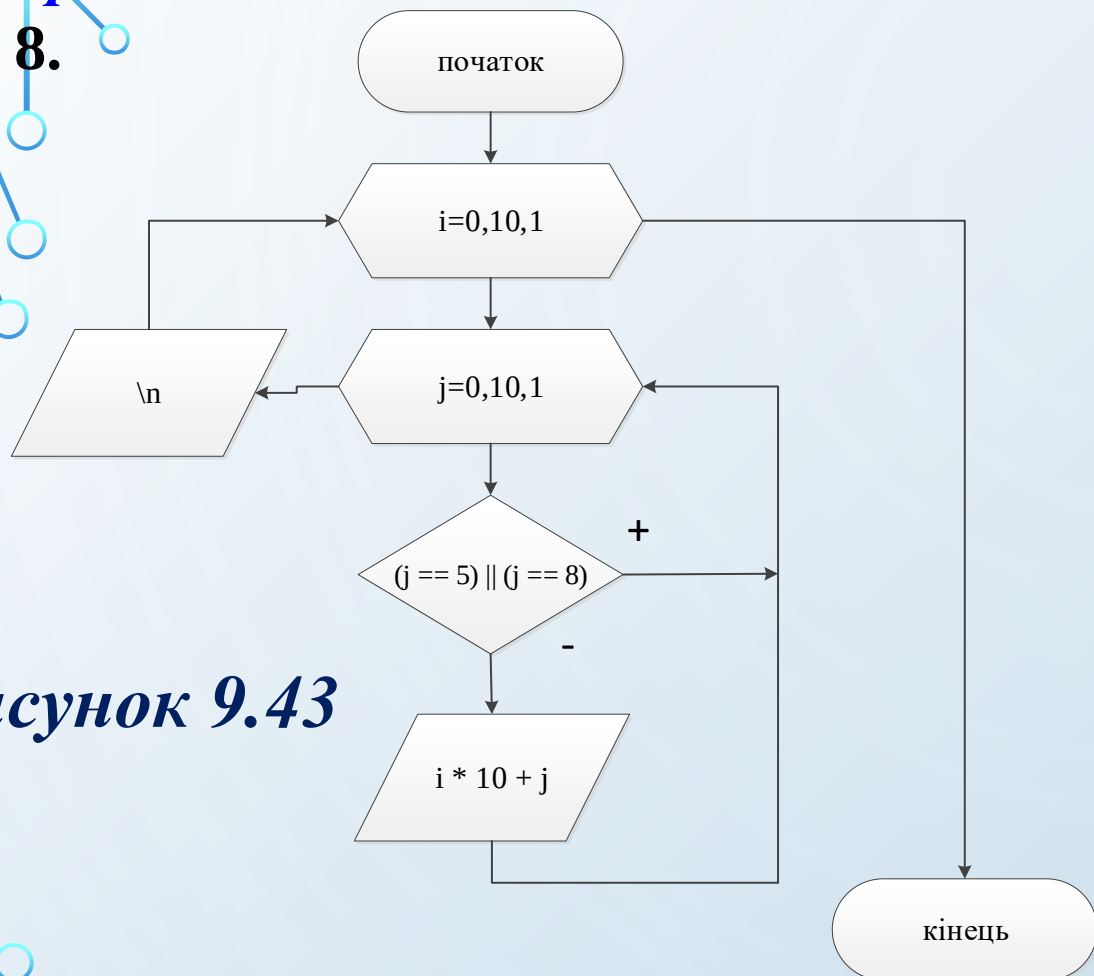


Рисунок 9.42

Приклад 9.21 Вивести числа від 0 до 99 виключаючи числа, що закінчуються на 5 або 8.



```
#include <stdio.h>
int main()
{
    for(int i=0; i<10; i++) // цикл для десятків
    {
        for (int j = 0; j < 10; j++) // цикл для одиниць
        {
            if ((j == 5) || (j == 8)) // якщо число
            одиниць в числі дорівнює 5 або 8
            continue; // переходимо до наступної
            ітерації цикла
            printf("%2d ", i * 10 + j); // виводимо
            обчислене число (2 знакомісця) та пробіл
        }
        printf("\n"); // в зовнішньому циклі
        переводимо рядок
    }
    return 0;
}
```

Рисунок 9.43

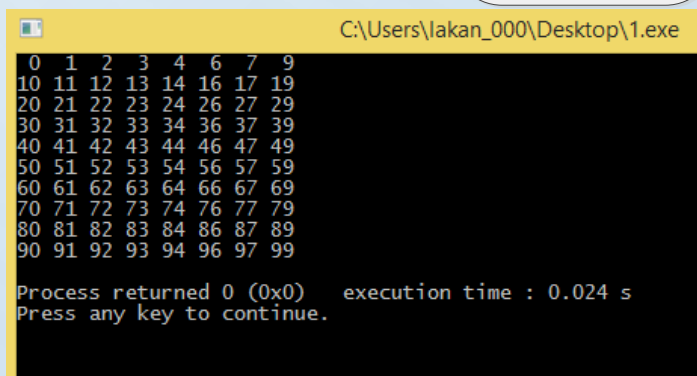


Рисунок 9.44

Тема №10. ФУНКЦІЇ

Тематичний план

- 10.1 Базові поняття функцій та їх правила написання на мові програмування C/C++*
- 10.2 Виконання викликів функцій та способи передачі параметрів*
- 10.3 Рекурсивні функції*
- 10.4 Створення власної бібліотеки*
- 10.5 Області оголошення та доступу до імен. Специфікатори та макроси*

10.1 БАЗОВІ ПОНЯТТЯ ФУНКЦІЙ ТА ЇХ ПРАВИЛА НАПИСАННЯ НА МОВІ ПРОГРАМУВАННЯ C/C++

Функція - це сукупність оголошень та операторів, що призначена для вирішення певного завдання.

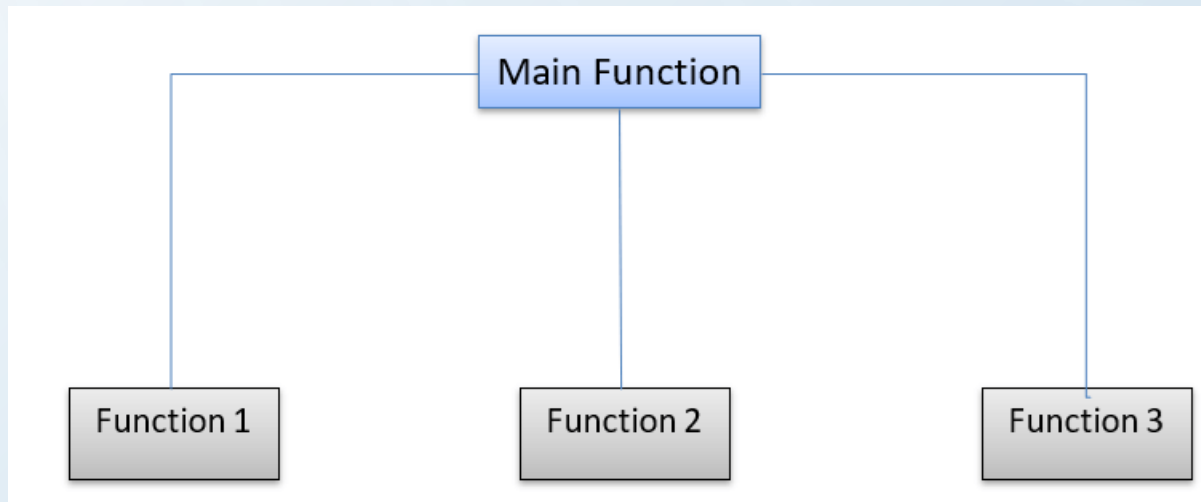


Рисунок 10.1

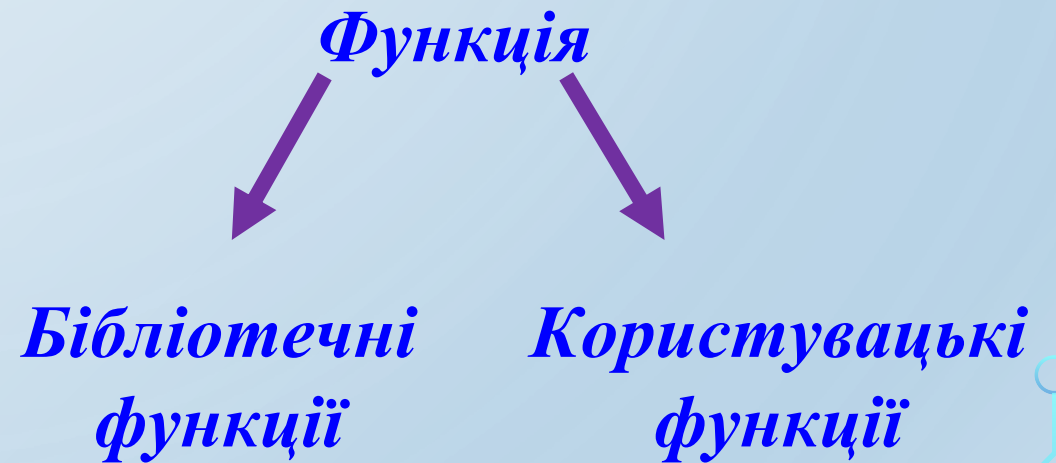


Рисунок 10.2

Структура та правила написання функцій

тип_значення_що_повертається

ім'я_функції(список_формальних_параметрів)

{

тіло_функції

}

return вираз;

Приклад 10.1

```
float sumTwoNumber (float a,  
float b)  
{  
    float z;  
  
    z=a+b;  
    return z;  
}
```

Приклад 10.2

```
float sumTwoNumber (float a, float b)  
{  
    return a+b;  
}
```

Як створити свою функцію?

ПІДКЛЮЧЕННЯ ЗАГОЛОВНИХ ФАЙЛІВ

ОПИС КОРИСТУВАЦЬКИХ ФУНКЦІЙ

ФУНКЦІЯ `main( )`

ОПИС КОРИСТУВАЦЬКИХ ФУНКЦІЙ

Рисунок 10.3

Приклад 10.1

```
#include <stdio.h>
// оголошуємо користувацьку функцію
max_num
// вхід: два цілочисельних параметра
з іменами a і b
// вихід: максимальне з двох
аргументів
int max_num(int a, int b){
    int max = b;
    if (a > b)
        max = a;
    return max;
}
//основна програма
int main(void) {
    int x = 0, y = 0;
    int m = 0;
    scanf("%d %d", &x, &y);
    m = max_num(x, y);
    printf("max(%d,%d) = %d\n", x, y, m);
    return 0;
}
```

```
#include <stdio.h>
int max_num(int, int);
int main(void)
{
    int x = 0, y = 0; int m = 0;
    scanf("%d %d", &x, &y);
    m = max_num(x,y);
    printf("max(%d,%d) =
%d\n",x,y,m);
    return 0;
}
int max_num(int a, int b)
{
    int max = b; if (a > b)
        max = a;
    return max;
}
```

Прототип функції

тип_значення_що_повертається *ім'я_функції*(список_формальних_параметрів);

```
float sumTwoNumber (float a, float b);
```

або

```
float sumTwoNumber (float, float);
```

10.2 ВИКОНАННЯ ВИКЛИКІВ ФУНКЦІЙ ТА СПОСОБИ ПЕРЕДАЧІ ПАРАМЕТРІВ

ім'я_функції (список_фактичних_параметрів)

Приклад 10.2

```
...  
float x=2.3;  
float y=sumTwoNumber(4.2, 5.4);  
printf (“ %f ”,3+sumToNumber(x, y/3));
```


Виклик функції зі змінною кількістю параметрів

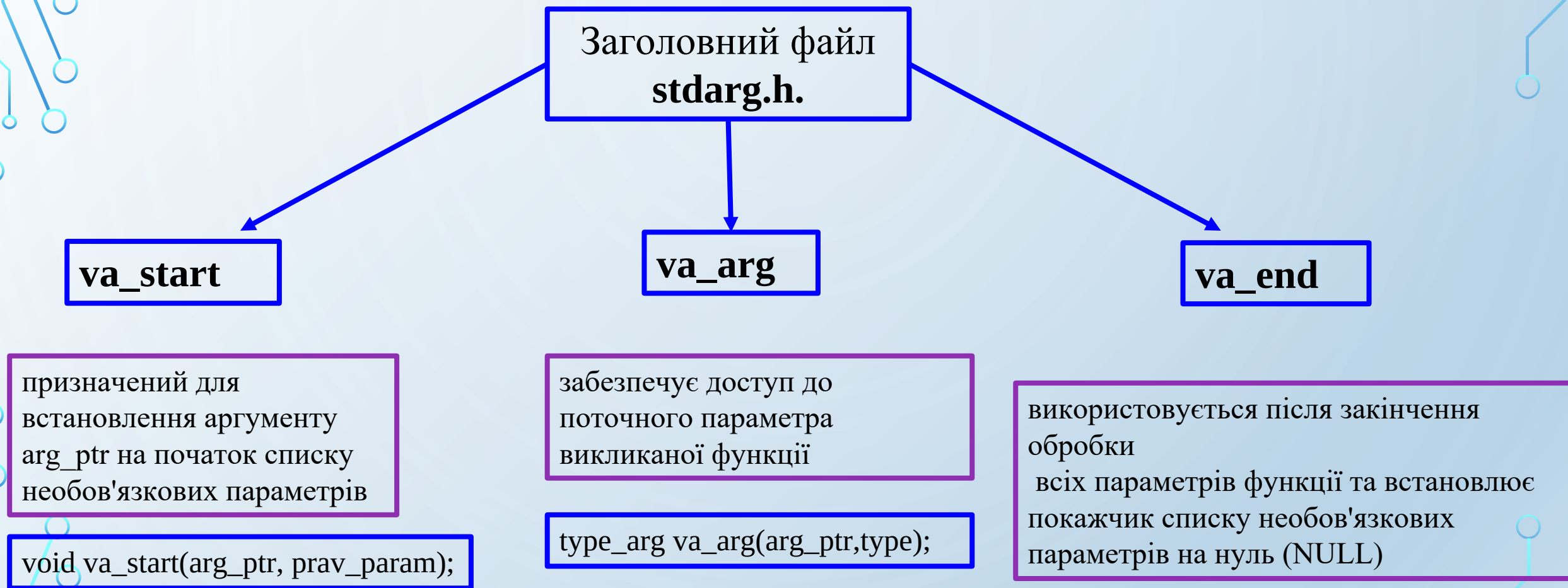


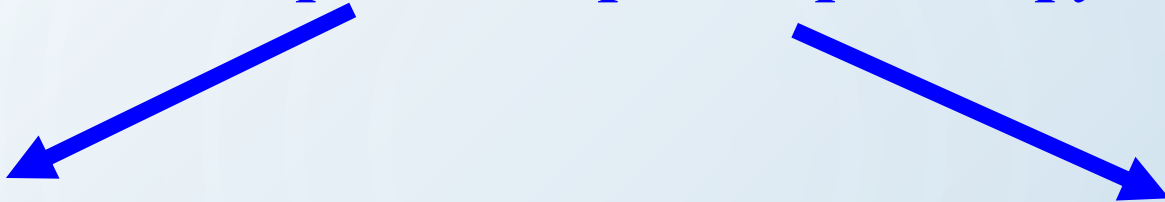
Рисунок 10.4

```
#include <stdio.h>
int sred_znach(int,...);
int main()
{
    int n;
    n=sred_znach(2,3,4,-1); /* виклик з 4-ма параметрами */
    printf("n=%d",n);
    n=sred_znach(5,6,7,8,9,-1); /* виклик з 6-ма параметрами */ printf("n=%d",n);
    return (0);
}

int sred_znach(int x,...);
{
    int i=0, j=0, sum=0;
    va_list uk_arg; va_start(uk_arg,x); /* встановлення uk_arg на перший необов'язковий
    параметр */
    if (x!=-1) sum=x; /* перевірка на пустоту списку */
    else return (0);
    j++;
    while ( (i=va_arg(uk_arg,int))!=-1) /* вибірка чергового */
    { /* параметра и перевірка */
        sum+=i; /* на кінець списку */
        j++;
    }
    va_end(uk_arg); /* закриття списку параметрів */
    return (sum/j);
}
```

Приклад 10.2 Функція, що обчислює середнє значення довільної послідовності цілих чисел. Кінець списку значення, що дорівнює -1

Способи передачі параметрів до функцій



Передача параметрів за значенням (by value)

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

void change(int a) {
    a = 100;
    printf("%d\n", a);
}

void main() {
    int d = 200;
    printf("%d\n", d);
    change(d);
    printf("%d", d);
    getch();
}
```

Передача параметрів через посилання (by reference)

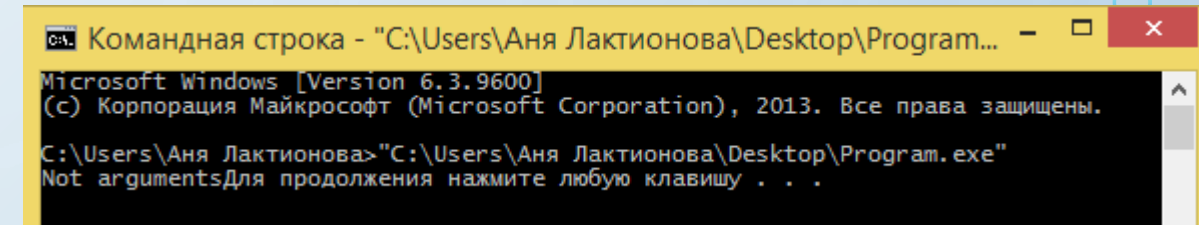
```
void swap(float &a, float &b)
{
    float temp= a;
    a = b;
    b = temp;
}
```

Передача параметрів функції main

Заголовок функції main :

```
int main (int argc, char *argv[], char *argp[])
```

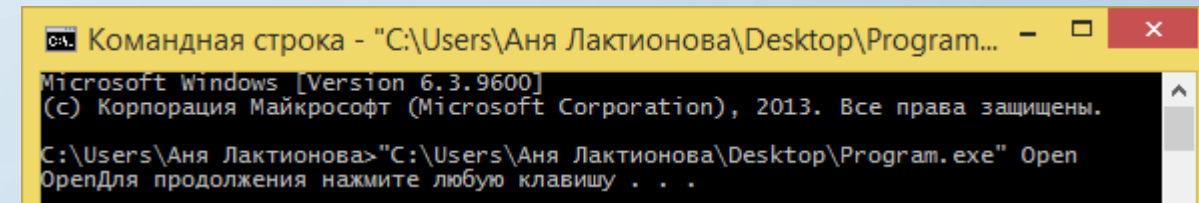
```
int main(int argc, char* argv[])
{
    if (argc > 1)// якщо передаємо аргументи, то argc
буде більше 1 (залежно від кількості аргументів)
    {
        printf ("%s",argv[1]);// виведення другого
рядка з масиву покажчиків на рядки (нумерація у
рядках починається з 0)
    } else
    {
        printf ("Not arguments");
    }
    system("pause");
    return 0;
}
```



Командная строка - "C:\Users\Аня Лактионова\Desktop\Program..."

Microsoft Windows [Version 6.3.9600]
(c) Корпорация Майкрософт (Microsoft Corporation), 2013. Все права защищены.

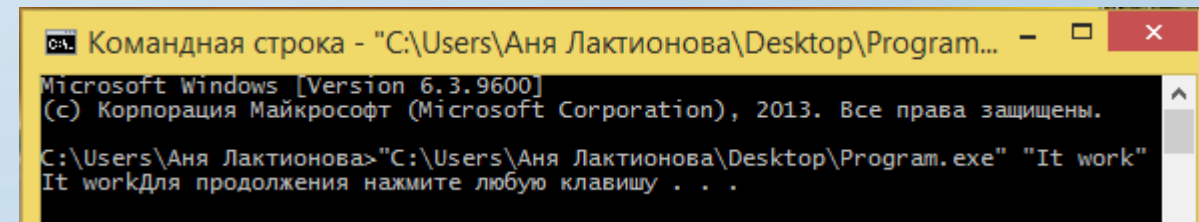
C:\Users\Аня Лактионова>"C:\Users\Аня Лактионова\Desktop\Program.exe"
Not argumentsДля продолжения нажмите любую клавишу . . .



Командная строка - "C:\Users\Аня Лактионова\Desktop\Program..."

Microsoft Windows [Version 6.3.9600]
(c) Корпорация Майкрософт (Microsoft Corporation), 2013. Все права защищены.

C:\Users\Аня Лактионова>"C:\Users\Аня Лактионова\Desktop\Program.exe" Open
OpenДля продолжения нажмите любую клавишу . . .



Командная строка - "C:\Users\Аня Лактионова\Desktop\Program..."

Microsoft Windows [Version 6.3.9600]
(c) Корпорация Майкрософт (Microsoft Corporation), 2013. Все права защищены.

C:\Users\Аня Лактионова>"C:\Users\Аня Лактионова\Desktop\Program.exe" "It work"
It workДля продолжения нажмите любую клавишу . . .

Рисунок 10.5

10.3 РЕКУРСИВНІ ФУНКЦІЇ

Рекурсія - полягає у визначенні, описі, зображенні будь-якого об'єкта або процесу всередині самого цього об'єкта або процесу. Це ситуація, коли об'єкт є частиною самого себе.

числа Фіббоначчі

$f(n) = f(n-1) + f(n-2)$,
причому $f(0) = 0$, а $f(1) = 1$.

```
#include <stdio.h>
#include <stdlib.h>
int fibonachi(int);
int main()
{
    int n;
    for(int i = 0; i < 10;
i++)
    {
        n = fibonachi(i);
        printf("%d\t",n);
    }
    return 0;
}
```

```
int fibonachi(int n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return fibonachi(n - 1) + fibonachi(n - 2);
}
```

n=4

fibonachi(3) + fibonachi(2) => 2 + 1 = 3

fibonachi(3) => fibonachi(2) + fibonachi(1) => 1 + 1 = 2

fibonachi(2) => fibonachi(1) + fibonachi(0) => 1 + 0 = 1

Рисунок 10.6

Рекурентні співвідношення

$$a_n = f(n, a_{n-1}, a_{n-2}, \dots, a_{n-p})$$

Рекурсія або ітерації

```
#include <stdio.h>
int fact(int num)
{
    int rez = 1;
    for (int i = 1; i <= num; i++)
        rez *= i;
    return rez;
}
int main()
{
    printf( "%d",fact(4));
    return 0;
}
```

```
#include <stdio.h>
int fact(int num)
{
    if(num==1) return 1;
    return  num*fact(num -
1);
}
int main()
{
    printf( "%d",fact(4));
    return 0;
}
```

4* factorial (3)

factorial (3) => 3* factorial (2)

factorial (2) => 2* factorial (1)

1*2*3*4 = 24

10.4 СТВОРЕННЯ ВЛАСНОЇ БІБЛІОТЕКИ

```
#include <conio.h>
#include <stdio.h>
//Прототипи функцій. Імена аргументів
можна не писати
int odd(int);
int even(int);
void main()
{
printf("if %d odd? %d\n", 11, odd(11));
printf("if %d odd? %d\n", 10, odd(10));
getch();
}
//Визначення функцій
int even(int a)
{
if (a)
{ odd(--a); }
else { return 1; }
}
int odd(int a)
{
if (a)
{ even(--a); }
else { return 0; } }
```

Заголовний файл File1.h

```
#ifndef _FILE1_H_
#define _FILE1_H_
int odd(int);
int even(int);
#endif
```

Вміст файла вихідного кода File1.c

```
#include "File1.h"
int even(int a) {
    if (a) {
        odd(--a);
    } else {
        return 1;
    }
}
int odd(int a) {
    if (a) {
        even(--a);
    } else {
        return 0;
    }
}
```

Оголошення функції в .h файл
Визначення функцій в .c файл

```
#pragma once
```

```
int odd(int);
int even(int);
```

Функція main

```
#include <conio.h>
#include <stdio.h>
#include "File1.h"

void main() {
    printf("if %d odd?
%d\n", 11, odd(11));
    printf("if %d odd?
%d\n", 10, odd(10));
    getch();
}
```

10.5 ОБЛАСТІ ОГОЛОШЕННЯ ТА ДОСТУПУ ДО ІМЕН. СПЕЦИФІКАТОРИ ТА МАКРОСИ

Область оголошення – це частина програми, в якій можуть здійснюватися оголошення імен змінних, функцій, тощо.

Область доступу до імені – це частина програми, у межах якої це ім'я доступно програмісту.

Змінні та константи, що оголошені у файлі, але за межами будь-якої функції називаються **глобальними**. Глобальні змінні автоматично ініціалізуються нульовим значенням. Глобальні змінні існують поки виконується програма.

Змінні, що оголошені у функції називають **локальними**. Локальні змінні автоматично не ініціалізуються. Локальні змінні створюються тільки після виклику функції та існують тільки до завершення роботи функції, або блоку (автоматичні змінні).

Приклад 10.3

```
#include <stdio.h>
int global = 1348;
void test();
int main()
{
    printf("from the main function : global =%d \n", global);
    test ();
    return 0;
}
void test ()
{ printf("from user defined function : global =%d \n",
global);
}
```

Результат:

```
from the main function : global =1348
from user defined function : global =1348
```

Специфікація static

```
1
#include <stdio.h>
void foo()
{
    int a = 10;
    static int sa = 10;
    a += 5;
    sa += 5;
    printf("a = %d, sa = %d\n", a, sa);
}
```

```
2
int main()
{
    int i;
    for (i = 0; i < 10; ++i)
        foo();
}
```

```
#include <time.h>
#include <stdio.h>
// Функція генерації
// випадкових чисел
unsigned long amkm()
{
    unsigned long a =
0xBL, b = 0x5DEECE66DL;
    static unsigned
long x = time(NULL);
    return x=a + b * x;
}
```

// Тестування функції

amkm

```
int main()
```

```
{
```

```
    m:
```

```
    printf("%lu \n", amkm());
```

```
    goto m;
```

```
    return 0;
```

```
}
```

Рисунок 10.7

```
C:\Users\lakan_000\Desktop\33.exe
a = 15, sa = 15
a = 15, sa = 20
a = 15, sa = 25
a = 15, sa = 30
a = 15, sa = 35
a = 15, sa = 40
a = 15, sa = 45
a = 15, sa = 50
a = 15, sa = 55
a = 15, sa = 60
Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```


Макроси з параметрами

`#define ім'я_макросу(список_параметрів) (вираз_для_заміни)`

Макрос піднесення числа a до ступеня b :

`#define POW(a,b) ( exp( (b) * log(a) ) )`

Звернення до цього макросу:

`printf(" %d \n ", POW( 3 + x , 4.5 ));`

Тема №11. БАЗОВІ ПОНЯТТЯ ПРО ВКАЗІВНИКИ

Тематичний план

- 11.1 Оголошення вказівників*
- 11.2 Операції над вказівниками*
- 11.3 Вказівник void та типізація вказівників*
- 11.4 Використання вказівників для обміну даними між функціями*

11.1 ОГОЛОШЕННЯ ВКАЗІВНИКІВ

Вказівник – це особливий тип даних, значенням якого є адреса певного байту оперативної пам'яті.

<тип даних> *<Ім'я вказівника>;

```
int *px ;
```

```
int *pw, *pz, pv ;
```

```
double *pd = (double*)1024;
```

```
double z, *pz = &z;
```

```
int *pw = NULL ;
```

символічна константа **NULL**

Ім'я покажчика – це ідентифікатор написаний за правилами запису імен у мові С.

***** – це ознака того, що наступне ім'я є вказівником.

1 варіант

```
const int z = 100;
```

```
const int *ptc = &z;
```

2 варіант

```
int *const cpt = &z;
```

3 варіант

```
const int z = 100;
```

```
const int *const ptc = &z;
```

Звертання до даних через вказівники

```
double z, *pz = &z;
```

```
z = 12.37; z += 2.5; z *= 2;
```

або

```
*pz = 12.37; *pz += 2.5; *pz *= 2;
```

Для чого потрібні показчики?

```
# include <stdio.h>
int main ()
{ float x = 10.1, y ;
  float * pf ;
  pf = &x ;
  y = *pf ;
  printf("x = %f y= %f", x, y); // Результат: x=10.1;
  y=10.1; pf=FFF2
  pf ++;
  printf ("x = %f y = %f", x, y); // Результат: x=10.1;
  y=10.1 ; pf=FFF6
  y =1+ *pf * y;
  printf ("x = %f y = %f", x, y); // Результат: x=10.1;
  y=1 ; pf=FFF6
  return 0;
}
```


11.2 ОПЕРАЦІЇ НАД ВКАЗІВНИКАМИ

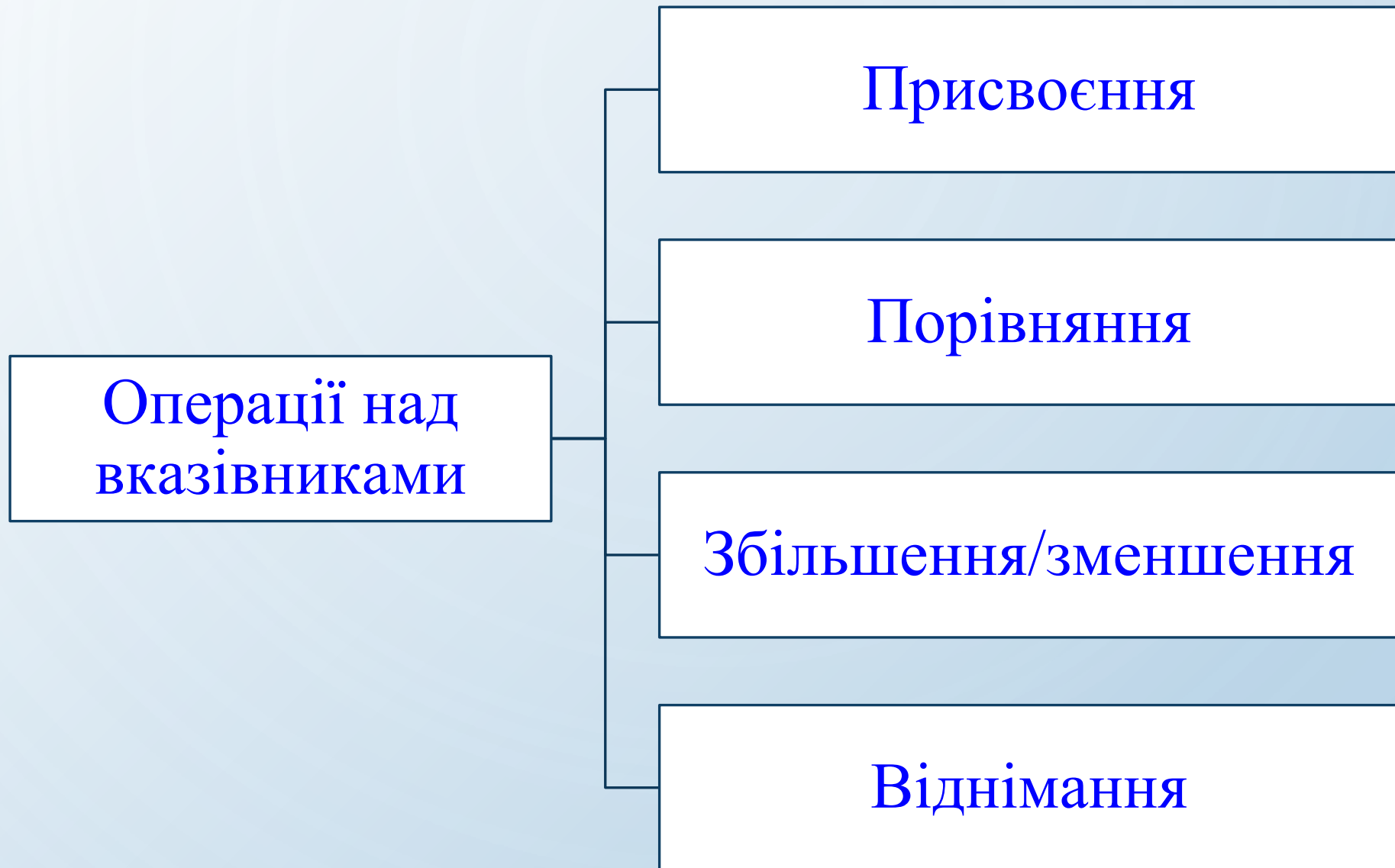


Рисунок 11.1

Операції порівняння

Приклад 11.1

```
int x = 10;
int y = 10;
int* xptr = &x;
int* yptr = &y; // порівнюємо покажчики
if (xptr == yptr)
printf( "\n Покажчики дорівнюють один одному"); else
printf( "\n Покажчики не дорівнюють один одному ");
// порівнюємо значення, на які вказують покажчики
if (*xptr == *yptr)
printf( "\n Значення дорівнюють один одному ");
else
printf( "\n Значення не дорівнюють один одному ");
```

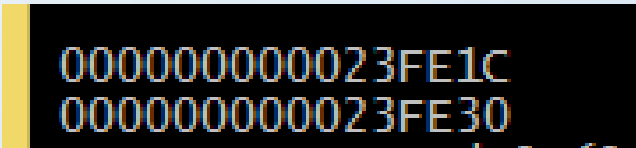
Операції додавання/віднімання

$\langle p \rangle = \langle p \rangle + n * \langle \text{кількість байт пам'яті базового типу покажчика} \rangle$

Приклад 11.2

```
int z1 = 100;  
int *ptn, *ptz = &z1;  
ptn = ptz+5;  
printf ("\n\n %p", ptz);  
printf ("\n %p", ptn);
```

2358812 2358832



```
000000000023FE1C  
000000000023FE30
```

Рисунок 11.2

Приклад 11.3

```
int x, *xptr;  
xptr=&x;  
  
printf("\n%p",xptr);  
xptr++;
```

printf("\n%p",xptr);

Результат – FFF2, FFF4

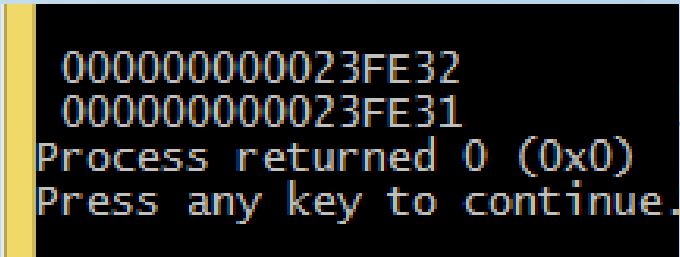
```
long int x, *xptr;  
xptr=&x;  
printf("\n%p",xptr);
```

```
xptr++;  
printf("\n%p",xptr);
```

Результат – FFF2, FFF6

Приклад 11.4

```
main()  
{  
char *p;  
char x;  
p = &x;  
printf ("%p", p);  
printf ("%p", --p);  
}
```



```
000000000023FE32  
000000000023FE31  
Process returned 0 (0x0)  
Press any key to continue.
```

Рисунок 11.3

Приклад 11.5

```
float a=4.0, *u, z;  
u=&z;  
*u=5;  
a=a+*u+1;  
// a = 10, u не змінилося, z =5.
```

a / * u

a / (* u)

"*", "++", "--" зправа наліво

11.3 ВКАЗІВНИК *VOID* ТА ТИПІЗАЦІЯ ВКАЗІВНИКІВ

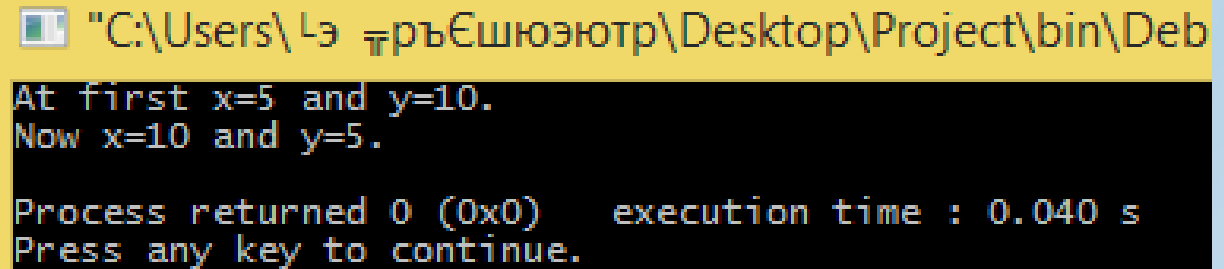
void – це безтиповий вказівник, означає просто адресу пам'яті.

```
void *pz;
```

```
long number = 0x12AB34DC;  
int part1, part2;  
void *p= &number;  
part1=*(int*)p;  
part2=*((int*)p+1);
```


11.4 ВИКОРИСТАННЯ ВКАЗІВНИКІВ ДЛЯ ОБМІНУ ДАНИМИ МІЖ ФУНКЦІЯМИ

```
#include <stdio.h>
void interchange (int * u, int * v) ;
int main(void)
{
    int x = 5, y = 10;
    printf ("At first x=%d and y=%d.\n", x, y);
    interchange (&x, &y); // передача адрес в функцію
    printf("Now x=%d and y=%d.\n", x, y);
    return 0;
}
void interchange(int * u, int * v)
{
    int temp;
    temp = *u; // temp отримує значення, на яке вказує
    u
    *u = *v; // ідентично x=y
    *v = temp;
}
```



```
"C:\Users\Ілля Шенюх\Desktop\Project\bin\Deb
At first x=5 and y=10.
Now x=10 and y=5.

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

Рисунок 11.4

Тема №12. ПРЕПРОЦЕСОР

Тематичний план

- 12.1 Базове визначення препроцесора. Визначення стадії препроцесорної обробки*
- 12.2 Дерективи препроцесора та їх обробка*
- 12.3 Умовна компіляція, мультирозгалуження та нумерація рядків*
- 12.4 Операції defined, # та ##*

12.1 БАЗОВЕ ВИЗНАЧЕННЯ ПРЕПРОЦЕСОРА. ВИЗНАЧЕННЯ СТАДІЇ ПРЕПРОЦЕСОРНОЇ ОБРОБКИ

Препроцесор - це спеціальна програма, яка є частиною компілятора мови Сі. Препроцесор **призначений** для попередньої обробки тексту програми. Препроцесор **дозволяє** включати в текст програми файли і вводити макровизначення.

Текст до
препроцесора
(вихідний текст
програми)

```
#define PI 3.141593
#define ZERO 0.0
if (r > ZERO) /* Порівняння з константою ZERO */
D=2*PI*r;      /* Довжина кола радіуса r:*/
```

Текст після
препроцесора

```
if (r > 0.0) /* Порівняння з константою ZERO */
D=2*3.141593*r; /* Довжина кола радіуса r:*/
```

Стадії препроцесорної обробки

- всі системно залежні позначення (наприклад, системно залежний індикатор кінця рядка) перекодуються в стандартні коди;
- кожна пара із символів '\' та “кінець рядка” разом із пробілами між ними забирається, і цим наступний рядок вихідного тексту приєднується до рядка, в якому була ця пара символів;
- у тексті кожного окремого рядка розпізнаються директиви та лексеми препроцесора, а кожен коментар замінюється одним символом порожнього проміжку;
- виконуються директиви препроцесора і здійснюються макро-підстановки;
- ескейп-послідовності у символьних константах та символьних рядках, наприклад '\n' або '\r', замінюються на їх еквіваленти(на відповідні числові коди);
- суміжні символьні рядки (рядкові константи) конкатенуються, тобто. з'єднуються в один рядок;
- кожна препроцесорна лексема перетворюється на текст мовою Cі.

12.2 ДЕРЕКТИВИ ПРЕПРОЦЕСОРА ТА ЇХ ОБРОБКА

Обробка директив препроцесора

- заміна ідентифікаторів (позначень) заздалегідь підготовленими послідовностями символів;
- включення в програму текстів із зазначених файлів;
- виключення із програми окремих частин її тексту (умовна компіляція);
- макропідставка, тобто заміна позначення параметризованим текстом,
- формується препроцесором з урахуванням конкретних параметрів.

Таблиця 12.1 – Дерективи

<i>Визначення</i>	<i>Призначення</i>
#define	Визначення макросу
#undef	Скасування визначення макросу
#include	Включення об'єкта-заголовка
#if	Компіляція, якщо вираз істинний
#ifdef	Компіляція, якщо макрос визначений
#ifndef	Компіляція, якщо макрос не визначений
#else	Компіляція, якщо вираз в if помилково
#elif	Складова директива else / if
#endif	Закінчення групи компіляції за умовою
#line	Заміна номера поточного рядка або імені вихідного файлу
#error	Формування помилок трансляції
#pragma	Дія, що залежить від конкретної реалізації компілятора
#	Null- директива

Операції

defined - перевірка істинності операнда;

- Конкатенація препроцесорних лексем;

- Перетворення операнда в рядок символів.

Підключення файлів

`#include < ім'я файла >`

або

`#include "ім'я файла"`

Приклад 12.1

`#include <stdio.h>`

`#include "func.c"`

Макровизначення

Макровизначення без параметрів

#define ІМ'Я_МАКРОСУ послідовність_символів

#define ІМ'Я_МАКРОСУ послідовність_символів1\
послідовність_символів2

Приклад 12.1

```
#define MAX 100
```

```
#define NAME "Привіт"
```

```
#define EULER 2.718282  
double mix = EULER; d = alfa*EULER;  
double mix = 2.718282; d = alfa*2.718282;
```

Макровизначення з параметрами

```
#define MIN(a, b) ((a)<(b)) ? (a) : (b)  
.....  
printf("Мінімум із x та y " % d, MIN(x ,y));  
printf("Мінімум із a та b " % d, MIN(n ,m));  
.....
```

```
printf("Мінімум " % d, MIN(x*x, x));
```

Ланцюжок підстановок

```
#include <limits. h>
#define RANGE ((INT_MAX) - < INT_MIN)+1)
/* RANGE - діапазон значень для int */
..... int RANGE_T = RANGE/8; .....
```

```
#define INT_MAX 32767 #define INT_MIN -32768
#define RANGE ((INT_MAX)-(INT_MIN)+1).
/* RANGE - діапазон значень для int*/
int RANGE_T = RANGE/8;
```

```
#define RANGE ((32767)-(-32768)+1)
/* RANGE - діапазон значень для int*/
int RANGE_T = RANGE/8;
```

```
/* RANGE - діапазон значень для int*/
int RANGE_T = ((32767) - (-32768)+1)/8;
```

Вихідний текст

#define K 40

```
void main( )  
{  
intM[K][K] ;  
float A[2*K+1],  
float B[K+3][K-3]  
.....
```

Результат препроцесорної обробки

```
void main ( )  
{  
intM[40][40] ;  
float A[2*40+1],  
float B[40+3][40-3];
```

```
#define PK printf("\n Номер  
елемента=%d.",N) ;
```

```
int N = 4; PK;
```

Результат

Номер элемента = 4.

#undef ім'я

```
.....  
A = 10; / Основний текст* /  
.....  
#define A X  
.....  
A = 5; / Включений текст * /  
.....  
#undef A  
.....  
B = A; / Основний текст * /  
.....
```

Директива *#error*

error повідомлення_про_помилку

#define NAME 5

```
#if (NAME != 5)
```

```
#error NAME должно быть равно 5 !
```

Сообщение будет выглядеть так:

Error<имя_файла><номер_строки>:

Error directive: NAME должно быть равно 5 !

В інтегрованому середовищі:

Fatal <имя_файла><номер_строки>:

Error directive: NAME должно быть равно 5!

`# pragma` лексеми

`#pragma once` - препроцесор запам'ятає цей файл і не включатиме його ще раз в ту ж одиницю трансляції (аналог `include guards`, саме для `gcc` анітрохи не ефективніше);

`#pragma message` `<рядок>` - виводить повідомлення при компіляції;

`#pragma redefine_extname` `<старе ім'я>` `<нове ім'я>` — перейменовує функцію таблиці символів об'єктного файла;

`#pragma GCC optimize` (`<рядки>`) — встановлює вказаний рівень оптимізації для наступного коду.

10.3 УМОВНА КОМПІЛЯЦІЯ, МУЛЬТИРОЗГАЛУЖЕННЯ ТА НУМЕРАЦІЯ РЯДКІВ

Умовна компіляція

```
#if цілочисельний_константний_вираз
#ifdef ідентифікатор
#ifndef ідентифікатор
# else
#endif
#elif
```

Структура застосування

```
#if...  текст_1
#else  текст_2
#endif
```

```
#define DEBUG
#ifdef DEBUG printf ("Друк налагодження");
#endif
```

Схема захисту від повторного включення:

```
/* Файл з іменем filename */
/* Перевірка визначеності _FILE_NAME */
#ifndef _FILE_NAME.../* Текст файла filename, що включається
*/ /* Визначення _FILE_NAME */
#define _FILE_NAME
#endif
```

Таблиця 12.2 –

#if, #else, #elif, #endif	# ifdef і #ifndef
<pre># include <stdio.h> # define MAX 100 void main() { # if MAX>99 printf(" MAX равно %d \n", MAX); # endif return 0; }</pre>	<pre>#if <вираз> послідовність операторів #elif <вираз 1> послідовність операторів #elif <вираз 2> послідовність операторів # endif</pre>
	<pre>#ifdef ІМ'Я_МАКРОСУ послідовність операторів # endif і відповідно #ifndef ІМ'Я_МАКРОСУ послідовність операторів # endif</pre>

#elif цілісний константний вираз

```
#if условие  
текст для if  
#elif выражение_1  
текст_1  
#elif выражение _2  
текст_2  
#else текст для випадку else  
#endif
```

Нумерація рядків

#line

#line 100

#line 100 "file1.c"

Нехай препроцесор отримує для обробки файл "kkk.c" з таким текстом:

```
#define N 3 /* Определение константы */
```

```
void main()  
{  
#line 23 "file.c"  
double z[3*N];  
}
```

Після препроцесора у файлі з ім'ям "kkk.i" буде отримано набір рядків:

```
kkk.c 1:  
kkk.c 2: void main()  
kkk.c: 3 {  
kkk.c 4:  
file.c 23: double z[3*3]  
file.c 24 }
```

12.4 ОПЕРАЦІЇ DEFINED, # TA

Операція defined

#defined операнд

Операнд – або ідентифікатор, або ув'язнений у дужки ідентифікатор або звернення до макросу

Вираз

#if defined операнд

еквівалентно виразу

#ifdef операнд

0L або 1L

defined Y &&
!definedN
Важливий_текст
#endif

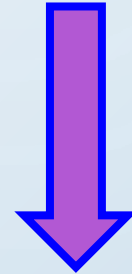
=

#ifdef Y
#ifndef N
важливий_текст
#endif
#endif

Onepauii # ma ##

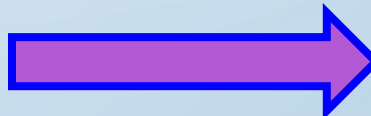
```
#define HELLO(x) printf("Hello, " #x "\n");
```

HELLO(JOHN)



printf ("Hello, " "John" "\n"); ~ printf ("Hello, John \n");

```
#define TOKENCONCAT(x,y) x ## y
```

TOKENCONCAT(O,K)  OK

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. ДСТУ ISO 5807:2016 Обробляння інформації. Символи та угоди щодо документації стосовно даних, програм та системних блок-схем, схем мережевих програм та схем системних ресурсів (ISO 5807:1985, IDT) [Електронний ресурс]. - Режим доступу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=67202. - Назва з екрана.
2. Жуковський, С.С. Об'єктно-орієнтоване програмування мовою С++: навч.-метод. посіб. для студ. напряму 6.040302 Інформатика / С.С. Жуковський, Т.А. Вакалюк. – Житомир: Вид-во ЖДУ, 2016. – 100 с.
3. Карпено Н.В. Розробка програм на мові С у сучасних середовищах: навч.-метод. посіб. / Н.В. Карпенко – Дніпро : ЛІРА, 2016. – 144 с.
4. Лабораторний практикум з дисципліни “Алгоритмізація та програмування”: навчальний посібник / Н. Б. Яворський, У. Б. Марікуца, М. І. Андрійчук, І. В. Фармага. – Львів: Видавництво Львівської політехніки, 2018. – 191с.
5. Алгоритмізація та програмування: практикум [Електронний ресурс]: навч. посіб. для здобувачів ступеня бакалавра за спеціальністю 122 “Комп’ютерні науки” / Л. І. Кублій; КПІ ім. Ігоря Сікорського. - Київ: КПІ ім. Ігоря Сікорського, 2019. - 209 с.

6. Трофименко О.Г. С++. Алгоритмізація та програмування: підручник / О.Г. Трофименко, Ю.В. Прокоп, Н.І. Логінова, О.В. Задерейко. 2-ге вид. перероб. і допов. – Одеса : Фенікс, 2019. – 477 с.
7. Комп'ютерні технології. Посібник для студентів спеціальності 172 «Телекомунікації та радіотехніка» факультету радіофізики електроніки та комп'ютерних систем / М.В. Кононов. – Київ: ФРЕКС Київського національного університету імені Тараса Шевченка, 2019. – 281 с.
8. Обчислювальна техніка та програмування: Алгоритми та їх реалізація. Комп'ютерний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 141 Електроенергетика, електротехніка та електромеханіка та 184 Гірництво / КПІ ім. Ігоря Сікорського; уклад. : Д. В. Філянін, В. П. Опришко, В. О. Броницький та ін. – Київ : КПІ ім. Ігоря Сікорського, 2022. – 95 с.
9. Основи алгоритмізації та програмування: навч. посіб. / Л.М. Бандоріна, Т.О. Климкович, К.О. Удачина. – Дніпро : УДУНТ, 2022. – 158 с.
10. Методичні вказівки до виконання практичних занять з дисципліни «Комп'ютерні технології та програмування. Частина 1» для студентів денної та заочної форм навчання всіх спеціальностей [Електронний ресурс] / уклад. Т. В. Алтухова, Е. А. Петелін. – Луцьк: ДВНЗ «ДонНТУ», 2023. – 39 с. - Режим доступу: <http://ea.donntu.edu.ua/jspui/handle/123456789/34575>. - Назва з екрана.