ЛЕКЦІЯ 1. АЛГОРИТМ І ЙОГО ВЛАСТИВОСТІ

План

1. Поняття алгоритму
2. Властивості алгоритму
3. Базові структури алгоритму
4. Способи запису алгоритму

Поняття алгоритму

Поняття алгоритму в інформатиці є фундаментальним, тобто таким, котре не визначається через інші ще більш прості поняття (для порівняння у фізиці - поняття простору і часу, у математиці - крапка).

Слово «алгоритм» походить від «algorithmi» — латинської форми написання імені великого математика аль-Хорезмі, який сформулював правила виконання арифметичних дій. Тому спочатку під алгоритмом розуміли тільки правила виконання чотирьох арифметичних дій над багатоцифровими числами в десятковій системі числення. Зараз він є одним із фундаментальних понять інформатики. **Алгоритм** може являти собою деяку послідовність обчислень, а може — послідовність дій нематематичного характеру. Але, у кожному разі, перед його складанням повинні бути чітко визначені початкові умови й те, що має бути одержано.

Алгоритм – послідовність дій, спрямованих на розв’язання поставленої задачі.

Алгоритм — кінцева послідовність кроків у рішенні завдання, що приводить від вихідних даних до необхідного результату.

Алгоритмом називається зрозуміле і точне розпорядження виконавцю виконати послідовність дій, спрямованих на досягнення зазначеної мети чи на розв'язання поставленої задачі.

В останньому означенні використовується поняття "виконавець". Що це означає? Під виконавцем алгоритму ми розуміємо будь-яку істоту (живу чи неживу), яка спроможна виконати алгоритм. Все залежить від того, якої мети ми намагаємося досягнути. Наприклад: риття ями (виконавці - людина або екскаватор), покупка деяких товарів (один із членів родини), розв'язування математичної задачі (учень або комп'ютер) тощо.

Виконавець алгоритму

Кожний алгоритм створюється з розрахунку на конкретного виконавця, тому можна сказати, що **алгоритм** — це точні розпорядження (указівки, команди,

операції, інструкції) виконавцеві здійснити послідовність дій, спрямованих на розв'язання поставленої задачі.

Алгоритм складається із команд — окремих указівок виконавцеві виконати деякі конкретні дії. Команди алгоритму виконуються одна за одною, і на кожному кроці відомо, яка команда повинна виконуватися. Почергове виконання команд за кінцеве число кроків приводить до розв'язання задачі. Для того щоб виконавець міг розв'язати задачу за заданим алгоритмом, він повинен уміти виконувати кожну з дій, що вказується командами алгоритму.

Виконавцями алгоритмів можуть бути людина, тварини, автомати, тобто ті, хто розуміє та може виконати вказівки алгоритму.

Система команд виконавця — сукупність команд, які можуть бути виконані виконавцем; кожна команда алгоритму входить до системи команд виконавця.

В основі роботи автоматичних пристроїв лежить положення, що найпростіші операції, на які розпадається процес розв'язання задачі, може виконати машина, яка спеціально створена для виконання окремих команд алгоритму і виконує їх у послідовності, вказаній в алгоритмі.

Розробляти алгоритми може тільки людина. Виконують алгоритми люди й усілякі пристрої — комп'ютери, роботи, верстати, супутники, складна побутова техніка й навіть деякі дитячі іграшки.

Будь-який виконавець (і комп'ютер зокрема) може виконувати тільки обмежений набір операцій (екскаватор копає яму, вчитель вчить, комп'ютер виконує арифметичні дії). Тому алгоритми повинні мати наступні **властивості**.

Властивості алгоритму

Виконуючи алгоритм, виконавець може не вникати в зміст того, що він робить, і разом із тим отримати потрібний результат, тобто виконавець діє формально. Тому для правильної побудови алгоритму необхідно знати систему команд виконавця, бути впевненим, що виконання алгоритму завершиться за кінцеве число кроків. Тому кажуть про деякі *загальні властивості алгоритмів*.

1. ***Зрозумілість***. Щоб виконавець міг досягти поставленої перед ним мети, використовуючи даний алгоритм, він повинен уміти виконувати кожну його вказівку, тобто розуміти кожну з команд, що входять до алгоритму. *Зрозумілість* - це властивість алгоритму, що полягає в тім, що кожен алгоритм повинен бути написаний у командах, зрозумілих даному виконавцю.
2. **Дискретність.** Алгоритм розв'язання задачі повинен складатися з послідовності окремих кроків — відокремлених одна від одної команд (указівок), кожна з яких виконується за кінцевий час. Тільки закінчивши виконання однієї команди, виконавець переходить до виконання іншої.
3. **Визначеність** (однозначність). Кожна команда алгоритму однозначно визначає дії виконавця і не припускає подвійного тлумачення. Суворо визначеним є й порядок виконання команд.
4. **Формальність.** Будь-який виконавець, який володіє заданою системою команд, може виконати заданий алгоритм, не вникаючи в суть задачі.

5. **Скінченність алгоритму.** Послідовність команд, які потрібно виконати, має бути скінченною.

6. **Результативність.** Виконання алгоритму не може закінчуватися невизначеною ситуацією або зовсім не закінчуватися. Будь-який алгоритм передбачає, що його виконання при допустимих початкових даних за кінцеве число кроків приведе до очікуваного результату.

7. **Масовість.** Алгоритм має передбачати можливість зміни початкових (вхідних) даних у деяких допустимих межах і можливість використання його для розв'язання задач одного класу (універсальність алгоритму). Отож, під масовістю алгоритму мається на увазі можливість його застосування для вирішення великої кількості однотипних завдань.

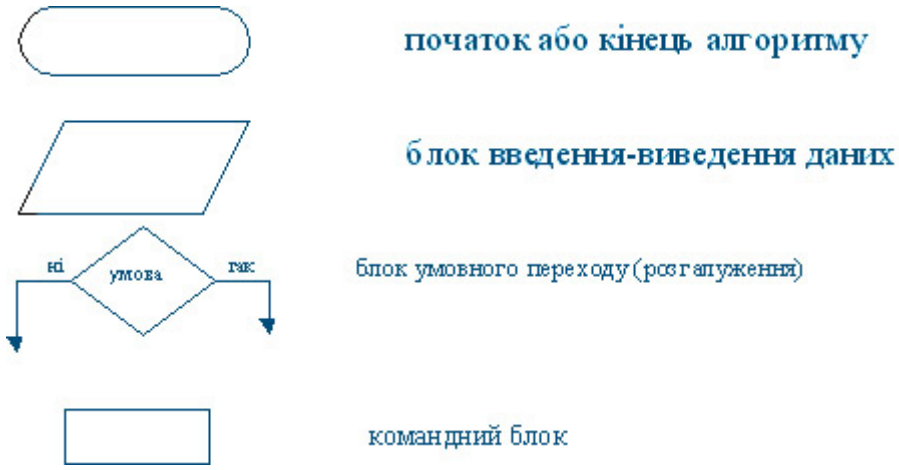
✍Саме через ці властивості часто дається визначення поняття **алгоритму** як скінченної однозначно визначеної послідовності операцій, формальне виконання якої приводить до розв'язання певної задачі за кінцеве число кроків.

Тепер залишається з'ясувати, яким чином можна подати алгоритм виконавцю. Існує кілька методів запису алгоритмів, вибір яких залежить від виконавця та того, хто його задає.

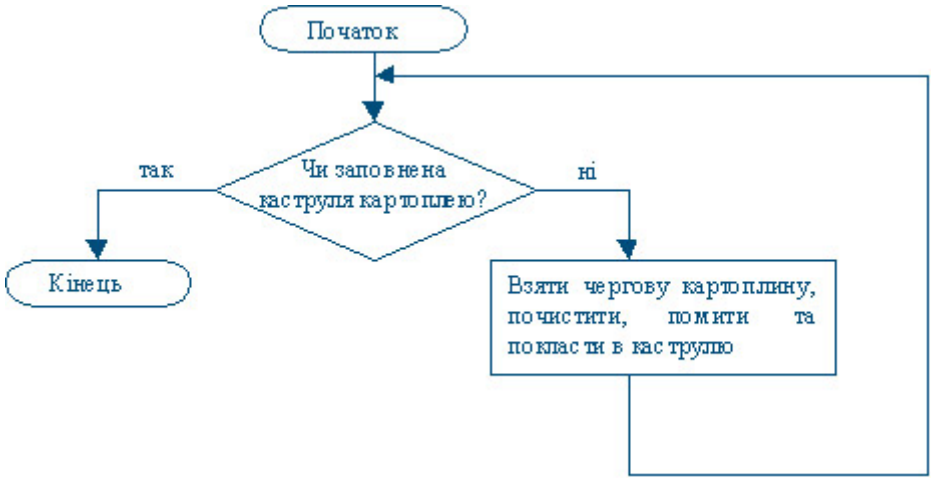
1. Перший спосіб - це ***словесний опис*** алгоритму. Це, по суті, звичайна мова, але з ретельним відбором слів і фраз, що не допускають зайвих слів, двозначностей і повторень. Доповнюється мова звичайними математичними позначеннями й деякими спеціальними відношеннями. Алгоритм описується у вигляді послідовності кроків. На кожному кроці визначається склад виконуваних дій і напрямок подальших обчислень. При цьому, якщо на поточному кроці не вказується який крок повинен виконуватися наступним, то здійснюється перехід до наступного кроку.

2. Другий спосіб - це подача алгоритму у вигляді ***таблиць, формул, схем, малюнків*** тощо. Наприклад, всіх вас вчили в дитячому садочку правилам поведінки на дорозі. І найкраще діти, вочевидь, сприймають алгоритм, що поданий у вигляді схематичних малюнків. Дивлячись на них, дитина, а потім і доросла людина, відпрацьовує ту лінію поведінки, що їй пропонується. Аналогічно можна навести приклади алгоритмів, що записані у вигляді умовних позначок на купленому товарі, щодо його користування (заварювання чаю, прання білизни тощо). У математиці наявність формул дозволяє розв'язати задачу, навіть "не використовуючи слів".

3. Третій спосіб - запис алгоритмів за допомогою ***блок-схеми***. Цей метод був запропонований в інформатиці для наочності представлення алгоритму за допомогою набору спеціальних блоків. Основні з цих блоків наступні:



Використовуючи дані блоки, можна подати, наприклад, алгоритм чищення картоплі в такому вигляді:



Примітка: У цьому алгоритмі, як правило, можна знайти неточності (наприклад, не відомо, що значить "взяти" і де "взяти", що значить "почистити" і таке інше). Тому степінь деталізації алгоритму залежить від виконавця. Ми, наприклад, передбачаємо, що наш виконавець в своїй системі команд має (тобто їх розуміє) команди "взяти чергову картоплю", "почистити картоплю", "помити" тощо.

4. Четвертий спосіб - ***навчальні алгоритмічні мови*** (псевдокоди). Ці мови мають жорстко визначений синтаксис і вже максимально наближені до машинної мови (мови програмування). Але створені вони з навчальною метою, тому мають зрозумілий для людей вигляд. Таких псевдокодів зараз існує велика кількість, починаючи з графічних середовищ "Алгоритміка", "Роботландія", "Лого-світи", "Черепашка" тощо і закінчуючи текстовими "національними" реалізаціями алгоритмічних мов, подібних до Бейсіка, Паскаля. Ці псевдокоди мають програмну реалізацію і дуже широко застосовуються на етапі навчання основам програмування.

5. П'ятий спосіб максимально наближений до комп'ютера - це ***мови програмування***. Справа в тому, що найчастіше у практиці виконавцем створеного людиною алгоритму являється машина (комп’ютер) і тому він повинен бути написаний мовою, зрозумілою для комп'ютера, тобто мовою програмування.

При побудові алгоритму часто виникає необхідність пояснити виконавцю деякі складні дії, якщо їх виконання не входить в систему команд виконавця. Наприклад, перший раз даючи дитині завдання пришити ґудзик до плаття, їй треба пояснити, як необхідно підбирати нитки для шиття, як вдягати нитку в голку, як тримати голку та ґудзик при роботі, яка різниця між пришиванням ґудзика до тоненької сорочки та товстої куртки (в другому випадку ґудзик робиться на "ніжці"). В подальшому такі пояснення будуть вже зайві, бо алгоритм "пришивання ґудзика" стає вже командою в системі команд виконавця "дитина".

Взагалі кажучи, кожна дія людини (якщо вона її може виконати) може вважатися командою її "системи команд", хоча колись, на етапі навчання, учитель або хтось інший ретельно пояснював, яку треба виконати послідовність дій, щоб досягти поставленої мети.

Узагальнюючи сказане, можна сказати, що кінець кінцем кожну задачу можна вважати окремою командою виконавцю, якщо його навчено виконувати поставлене завдання. Якщо ж виконавець не знає, як розв'язувати запропоновану задачу, виникає потреба розкласти її на такі *підзадачі*, що являються "посильними" для виконання, тобто входять до системи команд виконавця. Продовжуючи цей процес, остаточно отримують алгоритм, що складається з простих команд, зрозумілих виконавцю, або остаточно переконуються, що дана задача непосильна для вибраного виконавця, тому що в його системі команд не існує необхідних для цього команд. Наприклад, як би ми не деталізували алгоритм побудови багатоповерхової будівлі для дитини, задача кінець кінцем являється для неї непосильною.

Запропонований підхід до конструювання алгоритмів називається ***методом покрокової деталізації зверху вниз***. Вочевидь, що при такому підході кожна операція остаточно буде подана у вигляді лише одного з трьох типів базових структур алгоритмів - *лінійної* (в літературі часто ця структура називається *слідування*), *розгалуження* або *повторення* (циклу). Степінь деталізації алгоритму при цьому сильно залежить від того, на якого виконавця його орієнтовано.

Алгоритми, що складаються для розв'язування окремих підзадач основної задачі, називаються ***допоміжними***. Вони створюються при поділі складної задачі на прості або при необхідності багаторазового використання одного ж того набору дій в одному або різних алгоритмах.

Допоміжний алгоритм повинен мати тільки один вхід та один вихід, причому того, хто користується ним, зовсім не цікавить, як реалізований цей алгоритм. Головне, щоб всі команди, що входять до складу допоміжного алгоритму входили до системи команд обраного виконавця. Зверніть увагу ще на те, що в реальному житті допоміжні алгоритми можуть виконувати, навіть, зовсім інші виконавці. Наприклад, якщо батьки вдома вирішили зробити ремонт, це не значить, що вони самостійно повинні зробити собі шпалери та клей. Алгоритми виробництва матеріалів існують і їх хтось виконує, а ми тільки користуємось результатами їх роботи.

Таким чином, можна вважати допоміжний алгоритм своєрідним "чорним ящиком", на вхід якого подаються деякі вхідні дані, а на виході ми отримуємо очікуваний результат. Головне чітко домовитись про правила оформлення вхідних

даних та вигляд результату. Невиконання домовленостей може привести до збою у виконанні допоміжного алгоритму або до отримання неочікуваного результату.

Описаний метод послідовної деталізації лежить в основі технології структурного програмування і широко застосовується при використанні таких мов програмування, як Бейсік, Паскаль, С, С++ та інших.

При описуванні програми для комп'ютера мовами високого рівня допоміжні алгоритми реалізовуються у вигляді підпрограм. Правила опису, звернення до них та повернення в точку виклику визначаються конкретною мовою програмування. Для зручності часто використовувані підпрограми можна об'єднувати в бібліотечні модулі і при необхідності підключати їх в свої програми.

Базові структури алгоритму

✍**Базові структури алгоритму** — це структури, за допомогою яких створюється алгоритм для розв'язання певної задачі. Існують три основні (базові) алгоритмічні структури, або три основні типи алгоритмів: *лінійний, розгалужений* та *циклічний.*

1. Лінійний алгоритм (послідовне виконання, структура слідування) — це алгоритм, який забезпечує отримання результату шляхом одноразового виконання послідовності дій, незалежно від вхідних даних і проміжних результатів. Дії в таких алгоритмах виконуються послідовно, одна за однією, тобто лінійно.

Алгоритм ***РАНОК***

1.Встати о 6.30 годині.
2.Виконати гімн. вправи.
3.Умитися.
4.Поснідати.
5.Вийти з дому о 7.30 годині.

2. Розгалужений алгоритм (умова, структура вибору) — у класичному варіанті ця структура розглядається як вибір дій у разі виконання або невиконання заданої умови. Галуження бувають повними і неповними.

Повне галуження — це галуження, в якому певні дії визначені й у разі виконання, і в разі невиконання умови. Неповне галуження — це розгалуження, в якому дії визначені тільки у разі виконання (або у разі невиконання) умови.

Якщо *логічний вираз*, **то** *команда 1*, **інакше** *команда 2*.
Серія команд – це декілька команд.

Алгоритм ***ВЕЧІР***

1.Повернутися з коледжу додому після занять.
2.Пообідати.
3.Якщо погода хороша, то попрацювати в саду, інакше піти в бібліотеку, взяти книжку, повернутися додому.
4.Зробити домашнє завдання.
5.Повечеряти.
6.Якщо є цікава телепередача, то подивитися телевізор, інакше

почитати книжку.
7.Лягти спати.

3. Циклічний алгоритм (цикл, структура повторення) — це алгоритм, у якому передбачено повторення деякої серії команд. За допомогою цієї структури описуються однотипні дії, що повторюються декілька разів. Такі алгоритми забезпечують виконання довгої послідовності дій, записаних порівняно короткою послідовністю команд. Саме використання циклів дозволяє у повній мірі реалізувати швидкодію комп'ютерів.

Циклом називають процес повторення дій. Циклічні алгоритми забезпечують повторне виконання деяких команд скінчену кількість разів.

Доки *логічний вираз*, **виконати** *команди*

Алгоритм ***КОЛЕДЖ***

1.Іти на першу пару.
2.Доки не закінчилися заняття іти на наступну пару.
3.Іти додому.

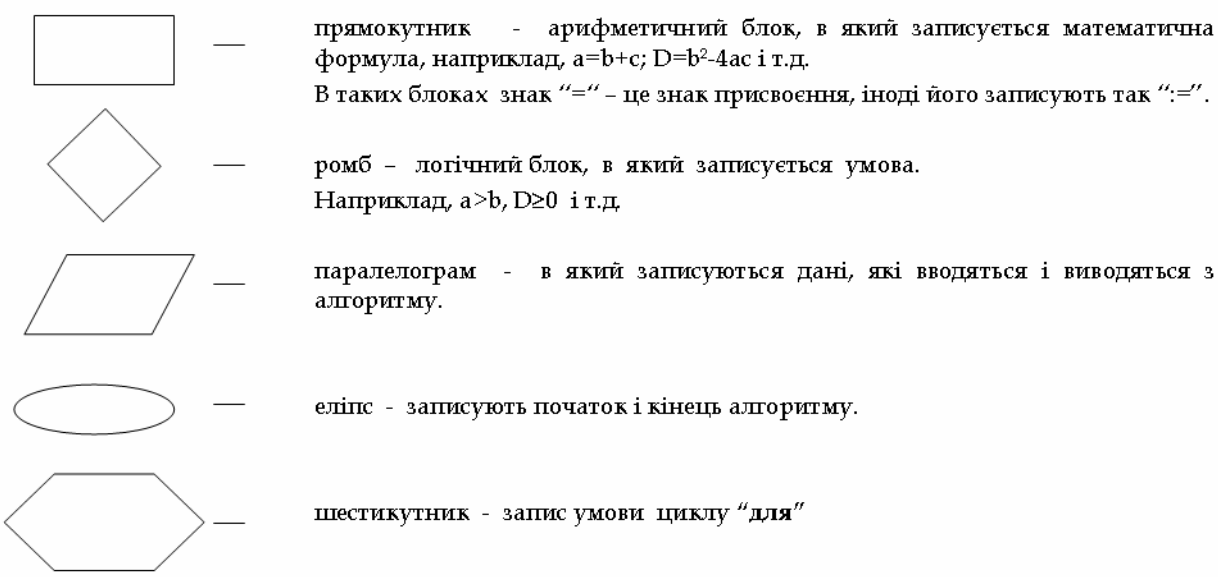
Основна особливість базових алгоритмічних структур — це їх повнота, тобто цих структур достатньо для створення найскладнішого алгоритму.

Способи запису алгоритму

Процес алгоритмізації — це визначення елементарних дій та порядку їх виконання для розв'язання поставленого завдання. Існують різні способи запису алгоритмів (словесний, формульно-словесний, метод блок-схем, програмний та ін.), які застосовуються для представлення алгоритму у вигляді, що однозначно розуміється і розробником, і виконавцем алгоритму.

Для опису алгоритмів людина часто користується природною мовою, але для запису багатьох алгоритмів природна мова виявилась незручною, тому виникла необхідність у створенні штучних мов, наприклад мови математичних формул, хімічних процесів тощо. Існує спеціальна навчальна алгоритмічна мова, яка була створена для запису алгоритмів на папері; вона використовує слова природної мови, але має більш жорстку структуру. Найбільше поширення для запису логічної структури алгоритмів отримали графічні (структурні) схеми, які спрощують складання та аналіз алгоритму, полегшують перехід від запису алгоритму до написання програми.

Блок-схема алгоритму — це графічне зображення алгоритму у вигляді спеціальних блоків з необхідними словесними поясненнями. Кожний етап алгоритму представляється у вигляді геометричної фігури (блоку), що має певну форму в залежності від характеру операції. Блоки на схемі з'єднуються стрілками (лініями зв'язку), які визначають послідовність виконання операцій та утворюють логічну структуру алгоритму.



прямокутник - арифметичний блок, в який записується математична формула, наприклад, a=b+c; $D=b^2-4ac$ і т.д.
В таких блоках знак "=" – це знак присвоєння, іноді його записують так ":=".

ромб – логічний блок, в який записується умова.
Наприклад, a>b, D≥0 і т.д.

паралелограм - в який записуються дані, які вводяться і виводяться з алгоритму.

еліпс - записують початок і кінець алгоритму.

шестикутник - запис умови циклу "**для**"

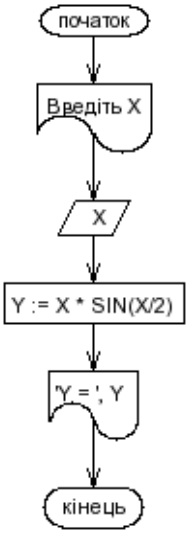
Важливою особливістю базових структур алгоритмів є те, що вони мають один вхід і один вихід, що дозволяє при відносній незалежності конструювати окремі блоки алгоритмів, а потім окремо розроблені структури з'єднувати між собою (вихід однієї базової структури сполучається із входом іншої). Весь алгоритм представляє лінійну послідовність базових структур.

Розглянемо приклади блок-схем.

Приклад 1. Створити блок-схему для обчислення функції X*SIN(X/2) по введеному значенню аргументу X

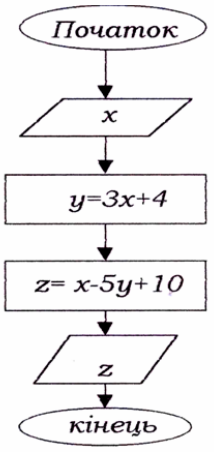
В цій задачі треба послідовно виконати ряд кроків, однакових для всіх вхідних даних, отже це буде **лінійний алгоритм**.

Блок-схема лінійного алгоритму



Приклад 2. Знайдіть значення виразу z = x-5y+10, де y = 3x+4

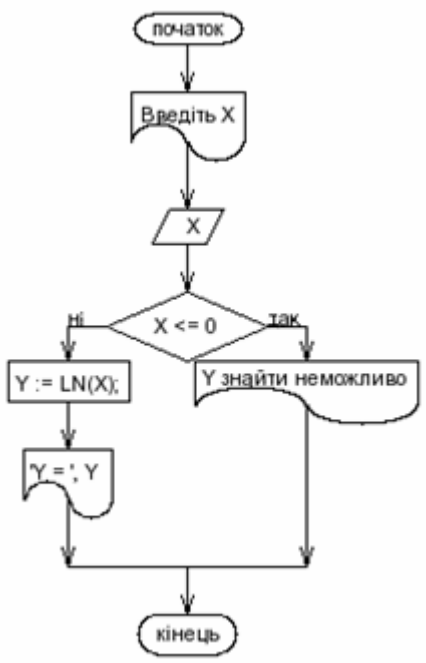
Блок-схема лінійного алгоритму має вигляд:



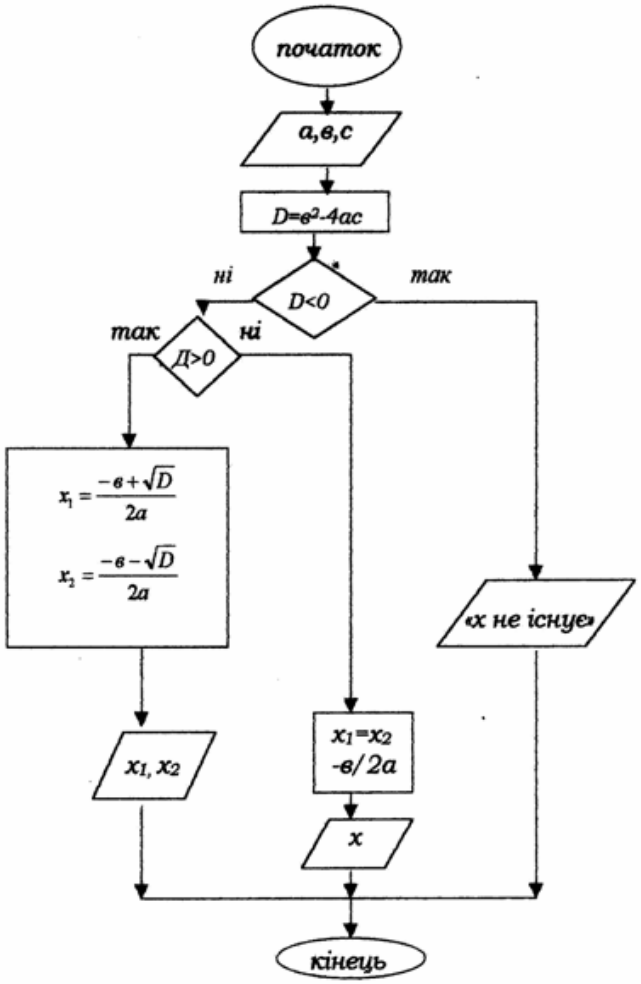
Приклад 3. Створити блок-схему для обчислення функції LN(X), яка перевіряє допустимість значення аргументу X

У цій задачі треба виконати ряд кроків, однакових для всіх вхідних даних, а деякі кроки будуть різними в залежності від значення аргументу, отже це буде **алгоритм з розгалуженням**.

У всіх випадках виконання починається з блоку ПОЧАТОК і закінчується у блоці КІНЕЦЬ. *Блок-схема алгоритму з розгалуженням:*



Приклад 4. Знайти корені квадратного рівняння $ax^2+bx+c=0$, де a, b, c > 0. *Блок схема представляє алгоритм з розгалуженням:*



Приклад 5. Знайти значення функції $y=x^2$ для x від -5 до 5 з кроком 0,5

Блок-схема циклічного алгоритму: