

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"



Звіт
з лабораторної роботи № 1
З дисципліни «Програмування , частина 2»

Виконав:
Ст. гр. ТР-13
Матейчук С. І.
Прийняв:
Асистент. Каф. ТК
Гордійчук-Бублівська О.В.

Львів - 2023

Мета роботи: ознайомитися із загальними принципами розробки алгоритмів, навчитися виконувати задачі щодо побудови різних типів алгоритмів.

Теоретичні відомості

Поняття алгоритму інтуїтивно зрозуміло та часто використовується в математиці та комп'ютерних науках. Говорячи неформально, алгоритм - це довільна коректно визначена обчислювальна процедура, на вхід якої подається деяка величина або набір величин, а результатом виконання якої є вихідна величина або набір значень.

Таким чином, алгоритм є послідовністю обчислювальних кроків, які перетворюють вхідні величини у вихідні. Алгоритм можна також розглядати як інструмент, який призначений для вирішення коректно поставленої обчислювальної задачі. У постановці задачі в загальних рисах визначаються відношення між входом та виходом.

В алгоритмі описується конкретна обчислювальна процедура, за допомогою якої можна досягнути виконання вказаних відношень. Можна навести загальні риси алгоритму:

- а. Дискретність інформації. Кожний алгоритм працює із даними: вхідними, проміжними, вихідними. Ці дані представляються у вигляді скінченних слів деякого алфавіту.
- б. Дискретність роботи алгоритму. Алгоритм виконується по кроках та при цьому на кожному кроці виконується тільки одна операція.
- с. Детермінованість алгоритму. Система величин, які отримуються в кожний (не початковий) момент часу, однозначно визначається системою величини, які були отримані в попередні моменти часу.
- d. Елементарність кроків алгоритму. Закон отримання наступної системи величин з попередньої повинен бути простим та локальним.
- е. Виконуваність операцій. В алгоритмі не має бути не виконуваних операцій. Наприклад, неможна в програмі призначити значення змінній «нескінченність», така операція була би не

виконуваною. Кожна операція опрацьовує певну ділянку у слові, яке обробляється.

- f. Скінченність алгоритму. Опис алгоритму повинен бути скінченним.
- g. Спрямованість алгоритму. Якщо спосіб отримання наступної величини з деякої заданої величини не дає результату, то має бути вказано, що треба вважати результатом алгоритму.
- h. Масовість алгоритму. Початкова система величин може обиратись з деякої потенційно нескінченної множини. Розглянемо для прикладу задачу сортування послідовності чисел у зростаючому порядку. Ця задача часто виникає на практиці і, фактично, буде центральною проблемою першого розділу даного курсу.

Задача сортування визначається формально наступним чином. Вхід: послідовність n чисел $(a_1; a_2; \dots a_n)$. Вихід: перестановка $(a'_1; a'_2; \dots a'_n)$ вхідної послідовності таким чином, що для всіх її членів виконується співвідношення $a'_1 < a'_2 < \dots < a'_n$. Наприклад, якщо на вхід подається послідовність $\langle 31, 41, 59, 26, 11, 58 \rangle$,

то вивід алгоритму сортування повинен бути таким: $\langle 11, 26, 31, 41, 58, 59 \rangle$.

Подібна вихідна послідовність називається екземпляром задачі сортування. Взагалі, екземпляр задачі складається із входу, який необхідний для розв'язання задачі та який задовольняє усім обмеженням, які присутні в постановці задачі.

В комп'ютерних науках сортування є основною операцією (у багатьох програмах вона використовується в якості проміжного кроку), в результаті чого з'явилося багато якісних алгоритмів сортування.

Вибір найбільш адекватного алгоритму залежить від багатьох факторів, в тому числі й від кількості елементів для сортування, від їх порядку у вхідній послідовності, від можливих обмежень, які накладаються на членів послідовності. Кажуть, що алгоритм є коректним, якщо для кожного входу результатом його роботи є коректний вивід. Тоді коректний алгоритм розв'язує дану обчислювальну задачу.

Якщо алгоритм некоректний, то для деяких входів він може взагалі не завершити свою роботу або видати відповідь, яка відрізняється від очікуваної.

Для чого вивчати алгоритми? По-перше, алгоритми є життєво необхідними складовими для рішення будь-яких задач з різноманітних напрямків комп'ютерних наук. Алгоритми відіграють ключову роль у сучасному розвитку технологій. Тут можна згадати такі розповсюджені задачі, як:

- розв'язання математичних рівнянь різної складності, знаходження добутку матриць, обернених матриць;
- знаходження оптимальних шляхів транспортування товарів та людей;
- знаходження оптимальних варіантів розподілення ресурсів між різними вузлами (виробниками, верстатами, працівниками, процесорами тощо);
- знаходження в геномі послідовностей, які співпадають;
- пошук інформації в глобальній мережі Інтернет; - прийняття фінансових рішень в електронній комерції; - обробка та аналіз аудіо та відео інформації.

Цей список можна продовжувати й продовжувати і, власне кажучи, майже неможливо знайти таку галузь комп'ютерних наук та інформатики, де б не використовувались ті або інші алгоритми.

По-друге, якісні та ефективні алгоритми можуть бути каталізаторами проривів у галузях, які є на перший погляд далекими від комп'ютерних наук (квантова механіка, економіка та фінанси, теорія еволюції).

І, по-третє, вивчення алгоритмів це також цікавий процес, який розвиває математичні здібності та логічне мислення.

Ефективність алгоритмів. Припустимо, швидкодія комп'ютера та об'єм його пам'яті можна збільшувати до нескінченності. Чи була би тоді необхідність у вивченні алгоритмів? Так, але тільки для того, щоб продемонструвати, що метод розв'язку має скінченний час роботи і що він дає правильну відповідь. Якщо б комп'ютери були необмежено швидкими, підійшов би довільний коректний метод рішення задачі. Звісно, тоді найчастіше обирався би метод, який найлегше реалізувати.

Сьогодні є дуже потужні комп'ютери, але їх швидкодія не є нескінченно великою, як і пам'ять.

Таким чином, час обчислення - це обмежений ресурс, як і об'єм необхідної пам'яті. Цими ресурсами слід користуватись розумно, чому й сприяє застосування алгоритмів, які ефективні в плані використання ресурсів часу та пам'яті.

Алгоритми, які розроблені для розв'язання однієї та тієї самої задачі, часто можуть дуже сильно відрізнятись за ефективністю. Ці відмінності можуть бути набагато більше помітними, чим ті, які викликані застосуванням різного апаратного та програмного забезпечення.

Перший алгоритм, який буде розглядатись - сортування включенням, для своєї роботи вимагає часу, кількість якого оцінюється як $c_1 n^2$, де n - розмір вхідних даних (кількість елементів у послідовності для сортування), c - деяка стала.

Цей вираз вказує на те, як залежить час роботи алгоритму від об'єму вхідних даних. У випадку сортування включенням ця залежність є квадратичною.

Використання коду, час роботи якого зростає повільніше, навіть при поганому комп'ютері та поганому компіляторі потребує на порядок менше процесорного часу. Для сортування 10 мільйонів чисел перевага сортування злиттям стає ще більш відчутною: якщо сортування включенням потребує для такої задачі приблизно 2,3 дня, то для сортування злиттям - менше 20 хвилин.

Загальне правило таке: чим більша кількість елементів для сортування, тим помітніша перевага сортування злиттям. Алгоритми, як і програмне забезпечення комп'ютера, являють собою технологію.

Загальна продуктивність системи настільки ж залежить від ефективності алгоритму, як і від потужності апаратних засобів.

Написанню програми на будь-якій мові програмування передують складання алгоритму, який показує послідовність виконання операцій, розгалуженість обчислювального процесу, логіку виконання всього ланцюгу програми від блоку введення вхідних даних до отримання кінцевого результату.

Існує чотири способи написання алгоритмів:

- вербальний (словесний);
- алгебраїчний (за допомогою літерно-цифрових позначень виконуваних дій); - графічний;
- з допомогою алгоритмічних мов програмування.

Словесна форма запису алгоритмів використовується в різних інструкціях, призначених для виконання їх людиною.

Алгебраїчна форма найчастіше використовується у теоретичних дослідженнях фундаментальних властивостей алгоритмів.

Графічна форма відповідно до державних стандартів (ГОСТ) на оформлення документації прийнята як основна для опису алгоритмів.

Алгоритм записаний за допомогою алгоритмічної мови програмування називається програмою. Алгоритм у такій формі може бути введений у ЕОМ і після відповідного оброблення виконаний з метою отримання шуканого результату.

Блок-схеми – найбільш зручний спосіб візуального представлення алгоритмів. Для того, щоб не заплутатися в численних подробицях, є сенс складати блок-схему

алгоритму в декілька ітерацій: почати з найбільш загального і поступово його уточнювати.

Блок-схема – наочне графічне зображення алгоритму, коли окремі його дії (етапи) зображуються за допомогою різних геометричних фігур (блоків), а зв'язки між етапами указуються за допомогою стрілок, що сполучають ці фігури.

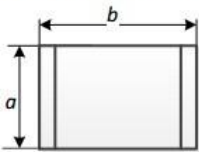
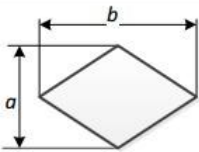
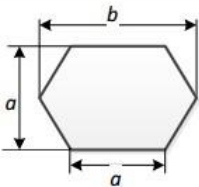
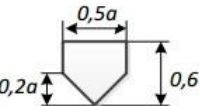
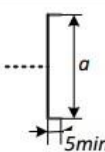
Блок-схеми відображають кроки, які повинні виконуватися комп'ютером, і послідовність їх виконання.

Умовні графічні позначення у блок-схемах алгоритмів наведені у табл. 1.

Таблиця 1

Умовні графічні позначення у блок-схемах алгоритмів

№	Назва	Графічний блок	Призначення
1	Блок початок-кінець		Вхід-вихід з програми, початок-кінець функції
2	Блок вводу-виводу даних		Уведення даних з клавіатури або виведення на екран результату
3	Обчислювальний блок		Обчислення та послідовність обчислень

№	Назва	Графічний блок	Призначення
4	Визначений процес		Виконання підпрограми (функції)
5	Логічний блок (блок умови)		Перевірка умови
6	Цикл		Початок циклу
7	Перехід		З'єднання між двома сторінками
8	Коментар		Пояснення

Для використанням блок-схем алгоритмів застосовують основні правила їх оформлення, що чітко регламентуються ГОСТ 19.701-90 (ИСО 5807-85).

Розміри символів розраховуються відповідно до наступних правил:

- менший геометричний розмір символу слід обирати з ряду 10, 15, 20, ... мм (тобто $a = \{10, 15, 20, \dots\}$ мм;
- співвідношення більшого та меншого розмірів має становити 1.5 (тобто $b = 1.5 a$).

Початок і кінець алгоритму зображуються за допомогою овалів (табл. 1., підпункт 1). У середині овалу записується "початок" або "кінець".

Введення початкових даних і виведення результатів зображуються паралелограмом (табл. 1., підпункт 2). У середині нього пишеться слово "введення" або "виведення" і перераховуються змінні, що підлягають введенню або виведенню.

Виконання операцій зображується за допомогою прямокутників (табл. 1., підпункт 3) в яких записано вираз/операцію. Для кожної окремої операції використовується окремий блок. Раніше створені і окремо описані функції та підпрограми зображуються у вигляді прямокутника з бічними лініями (табл. 1., підпункт 4). У середині такого "подвійного" прямокутника указуються ім'я функції (підпрограми), параметри, при яких вона повинна бути виконана. Блок вибору, що

визначає шлях, по якому підуть ці дії (наприклад, обчислення) далі, залежно від результату аналізу даних, зображується у вигляді ромбу (табл. 1., підпункт 5).

Сама умова записується усередині ромба. Якщо умова, що перевіряється, виконується, тобто має значення "істина", то наступним виконується етап по стрілці "так". Якщо умова не виконується ("хибність"), то здійснюється перехід по стрілці "ні". Стрілки повинні бути підписані.

Шестикутник (табл. 1, підпункт 6) використовують для зміни параметра змінної, яка керує виконанням циклічного алгоритму, також зазначаються умови завершення циклу. Стрілками зображуються можливі шляхи алгоритму, а малими п'ятикутниками (табл. 1., підпункт 7) – розриви цих шляхів потоку з переходом на наступну сторінку.

Коментарі використовуються в тих випадках, коли пояснення не поміщається усередині блока (табл. 1., підпункт 8).

Існують такі правила графічного запису алгоритмів:

- блоки алгоритмів з'єднуються лініями потоків інформації;
- лінії потоків не повинні перетинатися;
- будь-який алгоритм може мати лише один блок початку і один блок кінця.

Будь-який, навіть найскладніший, алгоритм може бути поданий у вигляді комбінацій кількох елементарних фрагментів, які називають базовими структурами.

До них належать:

- послідовне проходження (рис. 1);
- розгалуження "якщо-то" (рис. 2);
- розгалуження "якщо-то-інакше" (рис. 3);
- обирання варіанта за ключем (рис. 4);
- цикл з параметром (рис. 5);
- цикл з передумовою (рис. 6); – цикл з післяумовою (рис. 7).
- використання коментарю (рис. 8).

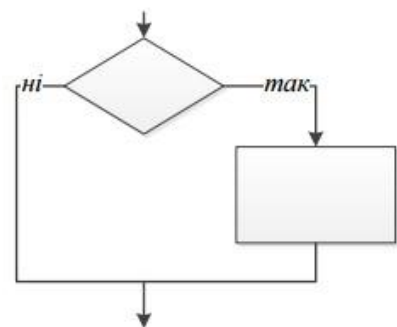
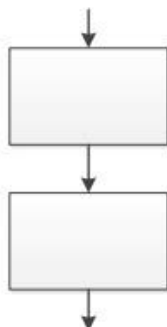


Рис. 1. Послідовне проходження

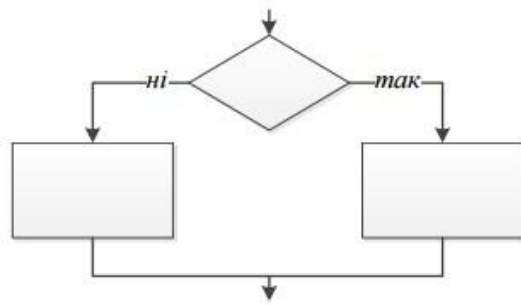


Рис. 2. Розгалуження « якщо – то »

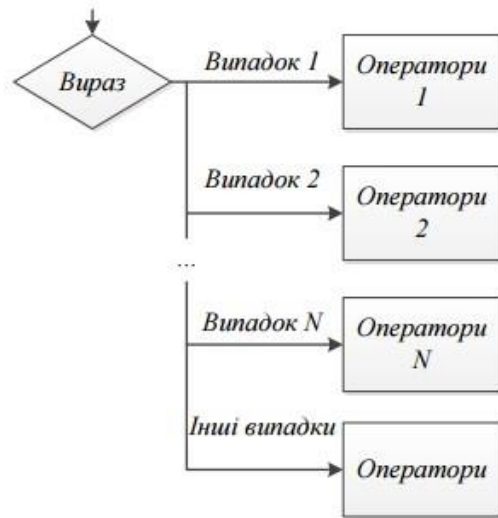


Рис. 3. Розгалуження "якщо-то- інакше"

ключем

Рис. 4. Обирання варіанта за

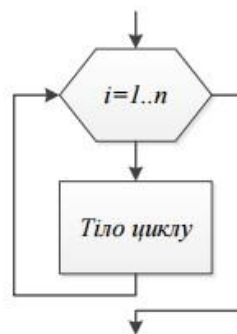


Рис. 5. Цикл з параметром

Рис. 6. Цикл з передумовою

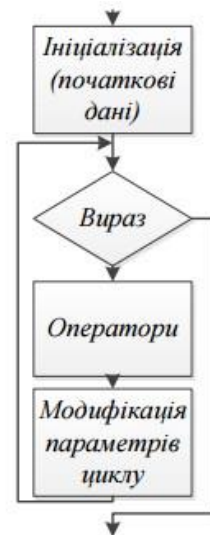


Рис. 7. Цикл з післяумовою

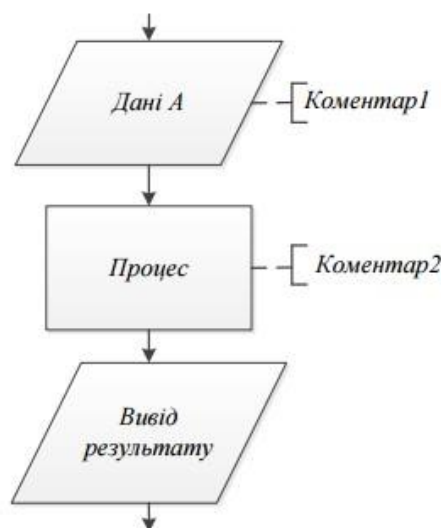


Рис. 8. Використання коментарю Блок

«Коментар» застосовують у тому випадку, коли усередині блоку не вдається

розмістити (написати) всю необхідну інформацію. У цьому випадку до лінії, що сполучає блоки, перед блоком, для якого необхідно написати додаткову інформацію, за допомогою пунктирної лінії приєднують (справа або в межах-зліва) блок «Коментар». Інформацію можна розміщувати за висотою висоти даного блоку, або за шириною – до краю сторінки.

Окремі блоки у алгоритмі можуть нумеруватись всі або деякі, це визначається складністю та розгалуженістю процесу. Нумерація блоків завжди полегшує читання блок-схеми. Блоки між собою з'єднуються стрілками, які показують напрямок обчислювального процесу.

Проектування схем алгоритмів як правило виконують із дотриманням наступної послідовності:

- 1) постановка задачі;
- 2) математична формалізація задачі;
- 3) вибір методу розв'язування задачі;
- 4) побудова схеми алгоритму; 5) перевірка алгоритму.

Постановка задачі – це чітке формулювання задачі, визначення вхідних даних для її розв'язування і точні вказівки відносно того, які результати і в якому вигляді повинні бути отримані. Перед виконанням наступних кроків роботи необхідно:

- уважно усвідомити задачу до чіткого розуміння її суті і вимог;
- визначити, які дані є вхідними, тобто такими, які задаються

користувачем алгоритму;

- визначити, які дані є вихідними, тобто такими, які треба отримати

в результаті розв'язання задачі.

Математична формалізація задачі – це опис задачі у вигляді формул, рівнянь, співвідношень, обмежень. Цей крок є найважливішим при виконанні даної роботи.

Більша частина задач потребують математичної формалізації.

Математична формалізація вимагає певного рівня знань, вмінь та навичок в області, до якої належить поставлена задача.

Вибір методу розв'язування задачі полягає у виборі сукупності способів та підходів до розв'язання задачі. Вибір методу залежить як від самої задачі, так і від можливостей комп'ютера.

При оцінюванні якості розв'язку задачі враховуються наступні показники:

- оригінальність розв'язку;
- об'єм пам'яті, який займає і використовує алгоритм (програма);
- трудомісткість обчислень, тобто ефективність алгоритму;
- лаконічність і наочність алгоритму.

Побудова схеми алгоритму – це графічний запис алгоритму на основі вибраного методу. Перед формуванням кінцевого варіанту схеми бажано розглянути і проаналізувати декілька її варіантів.

Розглянемо в деталях процес виконання алгоритму, заданого блок-схемою на рис. 9.

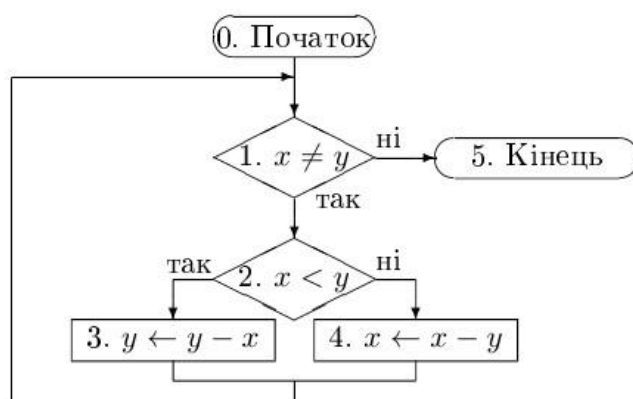


Рис. 9. Блок-схема алгоритму Евкліда

Алгоритм являє собою циклічну конструкцію з умовою в блоці 1, тілом якої є умовний оператор, у якого умова міститься в блоці 2, а гілками є оператори 3 та 4.

Нехай початковий стан пам'яті обчислювальної машини складається з двох змінних, значення яких відповідно $x = 15$, $y = 9$. Після запуску алгоритму управління передається на оператор розгалуження під номером 1. Перевірка умови $x \neq y$ при підстановці значень змінних з поточного стану пам'яті дає $15 \neq 9$ – істинне твердження.

Оператор розгалуження лише перевіряє умову, не присвоюючи нових значень, тому стан пам'яті залишається незмінним: $x = 15$, $y = 9$. За стрілкою, поміченою словом «так», управління передаються на оператор під номером 2, теж оператор розгалуження. Підстановка значень змінних дає співвідношення $(15 < 9)$, ця умова хибна, тому за стрілкою з позначкою «ні» управління передається на оператор 4, стан пам'яті поки що залишається незмінним.

Виконання оператору 4 полягає в тому, що обчислюється значення виразу при підстановці значень змінних: $x - y = 15 - 9 = 6$. Оператор присвоювання змінює стан пам'яті: значенням змінної x відтепер стає число 6, а значення змінної y залишається

таким, як було раніше, тобто 9. За стрілкою приходимо знов до умовного оператору 1 - тіло циклу виконалося вже один раз , і тепер машина перевіряє, чи потрібно продовжувати виконання циклу.

Виконання оператору 1 в новому стані пам'яті встановлює, що умова $x \leq y$ істинна ($6 \leq 9$) , це призводить до продовження циклу. Відбувається перехід до оператору 2. Він, в свою чергу, передає управління на оператор 3, оскільки істинною виявляється умова $x < y$ ($6 < 9$) .

Результатом виконання оператору 3 стає новий стан пам'яті, в якому $x = 6$ (значення не змінилося) , $y = 3$ (нове значення) . Управління передається на оператор 1 .

Оператор 1 передає управління на оператор 2 (виконання циклу продовжується) , оскільки твердження $x \leq y$ істинне. Оператор 2 з'ясовує що співвідношення $x < y$ хибне та передає управління на оператор 4. Той, в свою чергу, присвоює змінній x нове значення, яке обчислюється з виразу $x - y$ при підстановці наявних в поточному стані значень змінних, це значення дорівнює 3. Управління знов передається на перевірку умови циклу, блок 1 .

Цього разу умова $x \leq y$ виявляється хибною, і управління передається на оператор 5, який є завершальним. Отже, алгоритм завершує свою роботу, його результатом є прикінцевий стан пам'яті : $x = 3, y = 3$.

Весь процес виконання алгоритму зручно зобразити у вигляді таблиці. Перша колонка таблиці - номер кроку процесу виконання, друга - номер блоку, який на цьому кроці виконується.

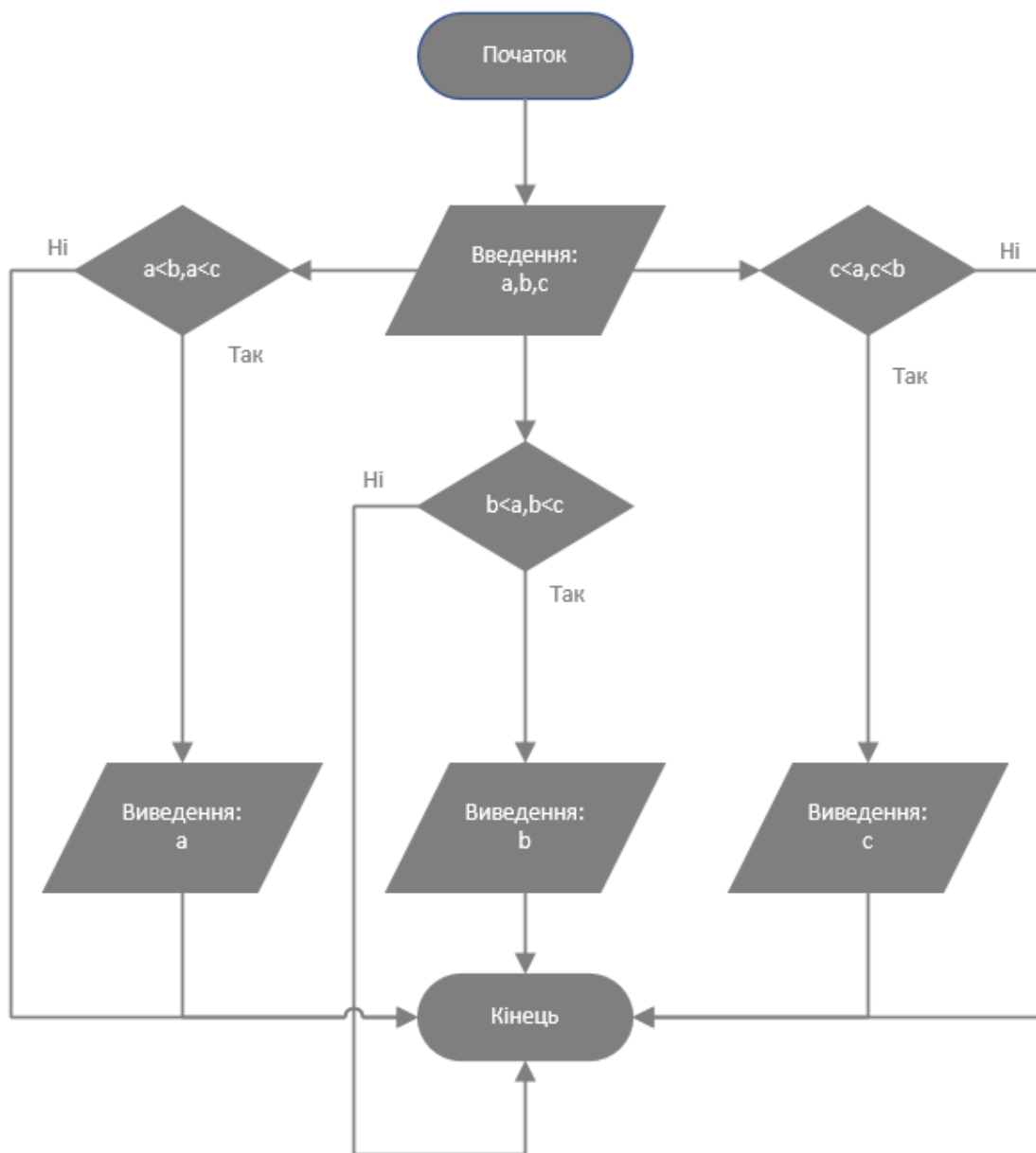
В наступних колонках наведено значення змінних, які складають стан пам'яті після виконання даного кроку. Якщо на даному кроці значення деякої змінної змінилося, то будемо позначати нове значення жирним шрифтом. В останню колонку будемо записувати істинність чи хибність умов в розгалуженнях та циклах, а також номер блоку, на який здійснюється перехід. Таким чином, описаному вище процесу застосування алгоритму до заданого початкового стану пам'яті відповідає таблиця 2.

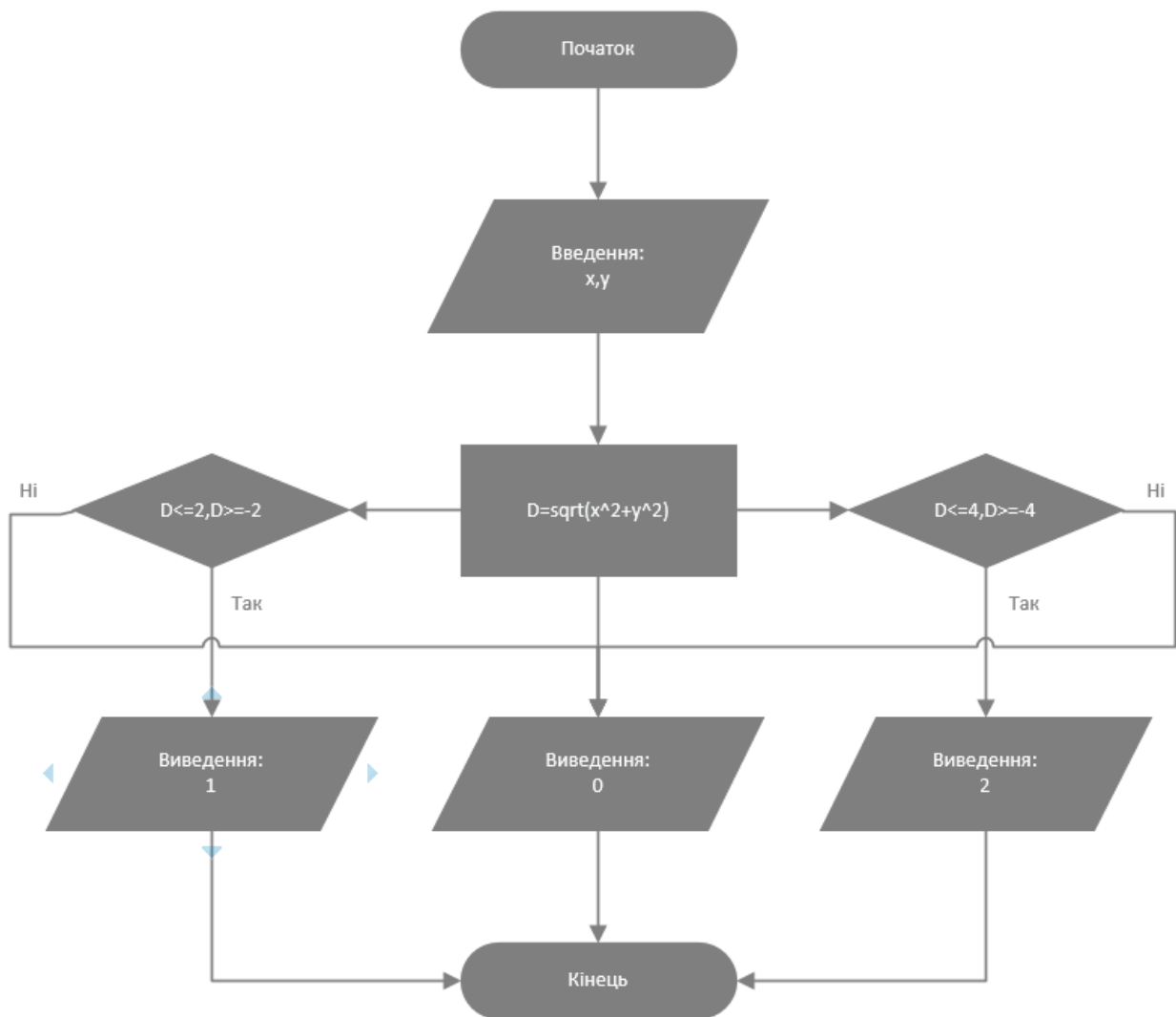
Щодо даного прикладу , то алгоритм, який тут розглядався, є алгоритмом Евкліда для обчислення найбільшого спільного дільника двох чисел.

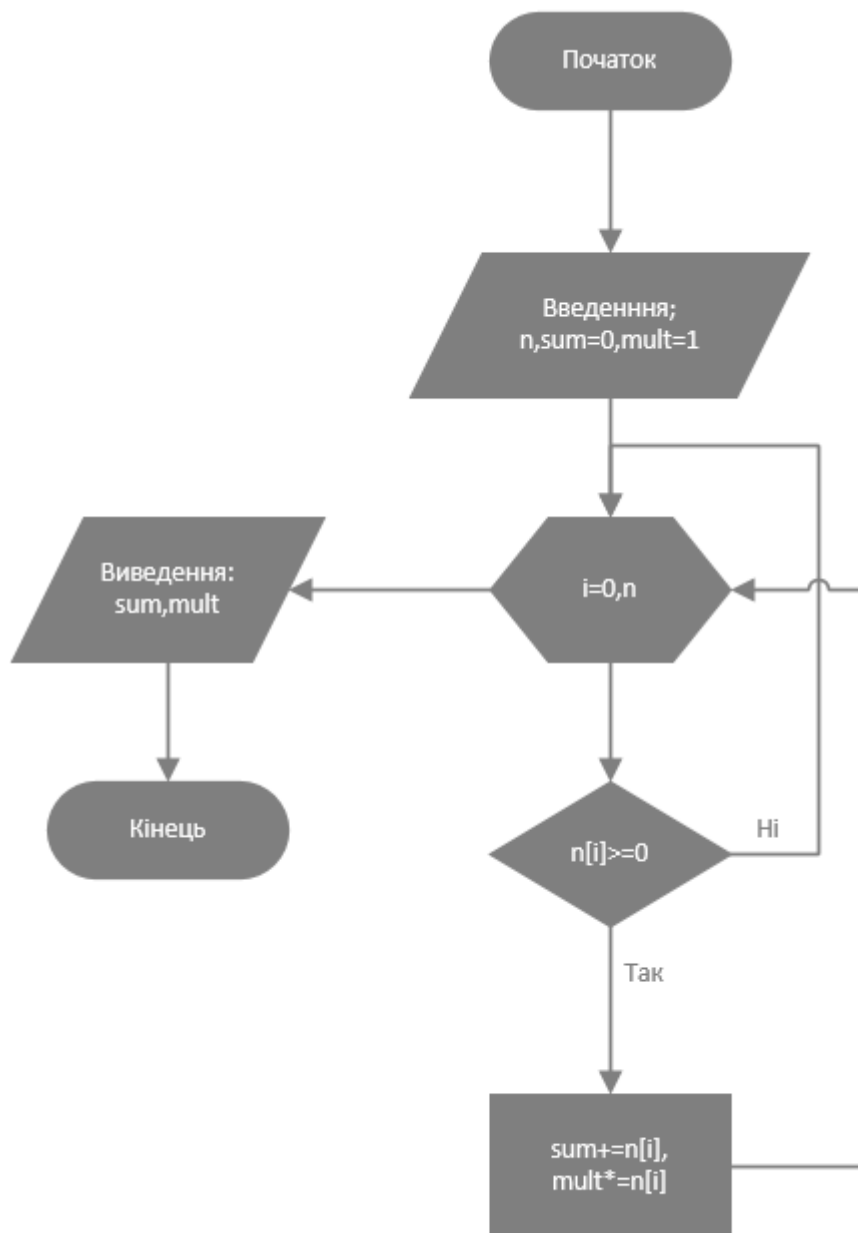
Таблиця 2 Протокол процесу виконання алгоритму Евкліда

Крок	Блок	x	y	Примітка
0	0	15	9	Початок. Перехід до 1
1	1	15	9	$x \neq y$ істинне, перехід до 2
2	2	15	9	$x < y$ хибне, перехід до 4
3	4	6	9	перехід до 1
4	1	6	9	$x \neq y$ істинне, перехід до 2
5	2	6	9	$x < y$ істинне, перехід до 3
6	3	6	3	перехід до 1
7	1	6	3	$x \neq y$ істинне, перехід до 2
8	2	6	3	$x < y$ хибне, перехід до 4
9	4	3	3	перехід до 1
10	1	3	3	$x \neq y$ хибне, перехід до 5
11	5	3	3	Алгоритм завершено

Хід роботи:







Висновки: при виконанні даної лабораторної роботи ми ознайомилися із загальними принципами розробки алгоритмів, навчилися виконувати задачі щодо побудови різних типів алгоритмів

