

1.3. Приклади найпростіших типових алгоритмів

Розглянемо деякі типові прийоми алгоритмізації, які на практиці використовуються або у вигляді окремих алгоритмів, або входять до складу більш складних алгоритмів. При цьому слід мати на увазі, що та ж сама задача може бути розв'язана різними способами.

Основними методами побудови алгоритмів є розробка розгалужених алгоритмів, організація простих і вкладених циклів, обчислення суми, добутку, кількості, обробка одновимірних та багатовимірних масивів: визначення максимального та мінімального значень масивів, сортування масивів за зростанням чи за спаданням тощо.

Реальні задачі мають суперпозицію або композицію цих типових засобів. Практично будь-який алгоритм містить один чи декілька циклів, тобто ділянок, що виконуються багаторазово. У простому циклі змінюється один параметр від заданого початкового до кінцевого значення з постійним кроком (залежно від вимог задачі параметр циклу може збільшуватися або зменшуватися). При кожному значенні параметра виконуються оператори, що знаходяться всередині циклу, тобто тіло циклу. Вихід з циклу здійснюється після досягнення параметром циклу свого кінцевого значення.

Приклад 1.1. Обчислити значення функції $y = ax^2 - \sin x$, якщо $x \in [-1; 2]$; $h_x = 0,5$; $a = 10,5$, та знайти кількість додатних значень функції.

У цьому прикладі **проста змінна** x є аргументом функції, який змінюється з кроком h_x .

На *рис. 1.1* представлено схему алгоритму розв'язання поставленої задачі, яка включає:

- блок введення значень коефіцієнта a та кроку зміни аргументу h_x .
- блоки присвоювання початкових значень змінній kol ($kol = 0$) та аргументу x ($x = -1$); змінна kol використовується для підрахунку кількості додатних значень функції;
- реалізацію багаторазового обчислення значень функції y при різних значеннях аргументу x , яка здійснюється циклічною ділянкою алгоритму, що виконує такі дії, як обчислення значення функції, виведення значень x і y ;
- розгалуження, що міститься в тілі циклу за параметром x , реалізоване блоком перевірки умови $y > 0$. У випадку, коли y додатний, **відбувається підрахунок кількості додатних значень функції** — $kol = kol + 1$;
- якщо додатний елемент функції підраховано або y від'ємний, значення аргументу x збільшується на крок — $x = x + h_x$;
- логічний блок, що керує циклом, містить перевірку нерівності $x < 2$ і, якщо x перевищить значення 2, забезпечує вихід з циклу.

Розроблений алгоритм більш наочно можна зобразити з використанням блоків початку і кінця циклу (див. *рис. 1.1, б*).

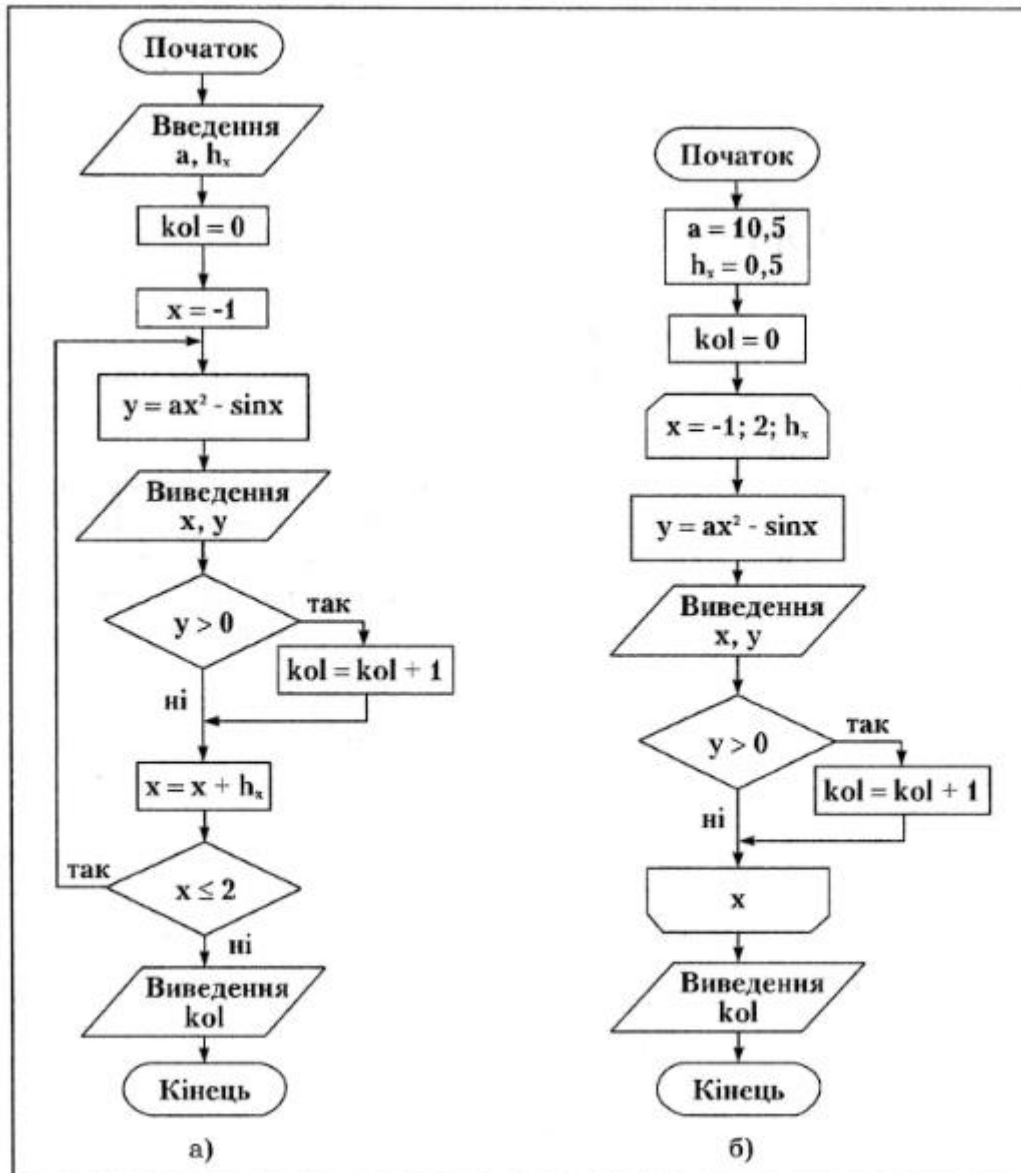


Рис. 1.1. Схема алгоритму прикладу 1.1

Приклад 1.2. Скласти алгоритм обчислення значень функції y , аргументи якої розташовано в масиві x_i ($i = 0..n-1$), $n = 20$ і $a = 0,75$; $b = 1,19$; $c = -2,5$.

$$y = \begin{cases} ax_i + b \cos x_i + c, & \text{якщо } x_i \leq 0,5 \\ bx_i^2 + c \sin 2x_i, & \text{для всіх інших значень } x_i \end{cases}$$

Знайти добуток значень функції, що перевищують величину **1,75** та суму всіх інших значень функції. Схему алгоритму розв'язання даної задачі наведено на рис. 1.2.

Алгоритм розв'язання складається з таких етапів:

- спочатку відбувається послідовне введення в пам'ять аргументів функції — елементів одновимірного масиву x_i ($i = 0..n-1$), $n = 20$, тобто блок введення передбачає для цього використання циклу за індексною змінною i ;
- для організації доступу до потрібного елемента масиву та виконання необхідних обчислень використовується новий цикл за параметром i ;
- згідно з умовою задачі в алгоритмі передбачено два розгалуження;

- у тілі циклу перша ділянка розгалуження починається з перевірки умови $x_i \leq 0,5$, оскільки область визначення функції поділяється на дві частини, в кожній з яких значення y обчислюється за окремою формулою;
- всередині циклу відбувається виведення поточних значень y та x_i ;
- друга ділянка розгалуження починається з перевірки умови $y > 1,75$ та здійснює **обчислення добутку і суми**;
- якщо умова $y > 1,75$ не виконується, реалізується обчислення суми: **початкове значення змінної суми sum дорівнює нулю ($sum = 0$)** і ця інструкція розташована ще до початку циклу, а далі у циклі багаторазово виконується накопичення суми $sum = sum + y$;
- у випадку виконання умови $y > 1,75$, аналогічно здійснюються створення добутку **pr : початкове значення добутку дорівнює одиниці ($pr = 1$)**, а подальше значення обчислюється як $pr = pr * y$;
- після завершення циклу за параметром i , тобто коли всі аргументи x_i вичерпано, здійснюється виведення одержаних значень суми sum і добутку pr .

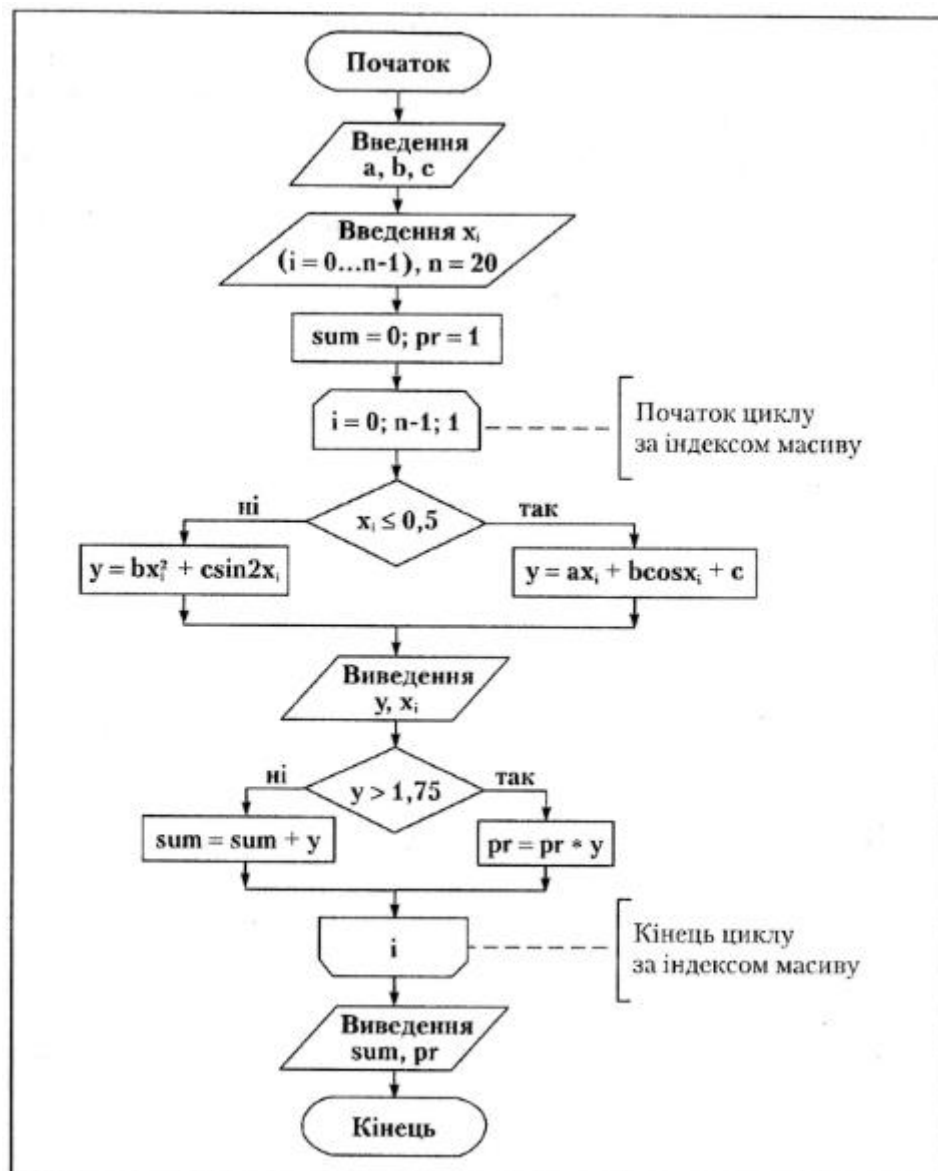


Рис. 1.2. Схема алгоритму прикладу 1.2

Приклад 1.3. За один перегляд масиву c_i ($i = 0 \dots n-1$), $n = 15$ визначити значення і положення максимального та мінімального його елементів і поміняти їх місцями. Схему алгоритму розв’язання задачі зображено на рис. 1.3.

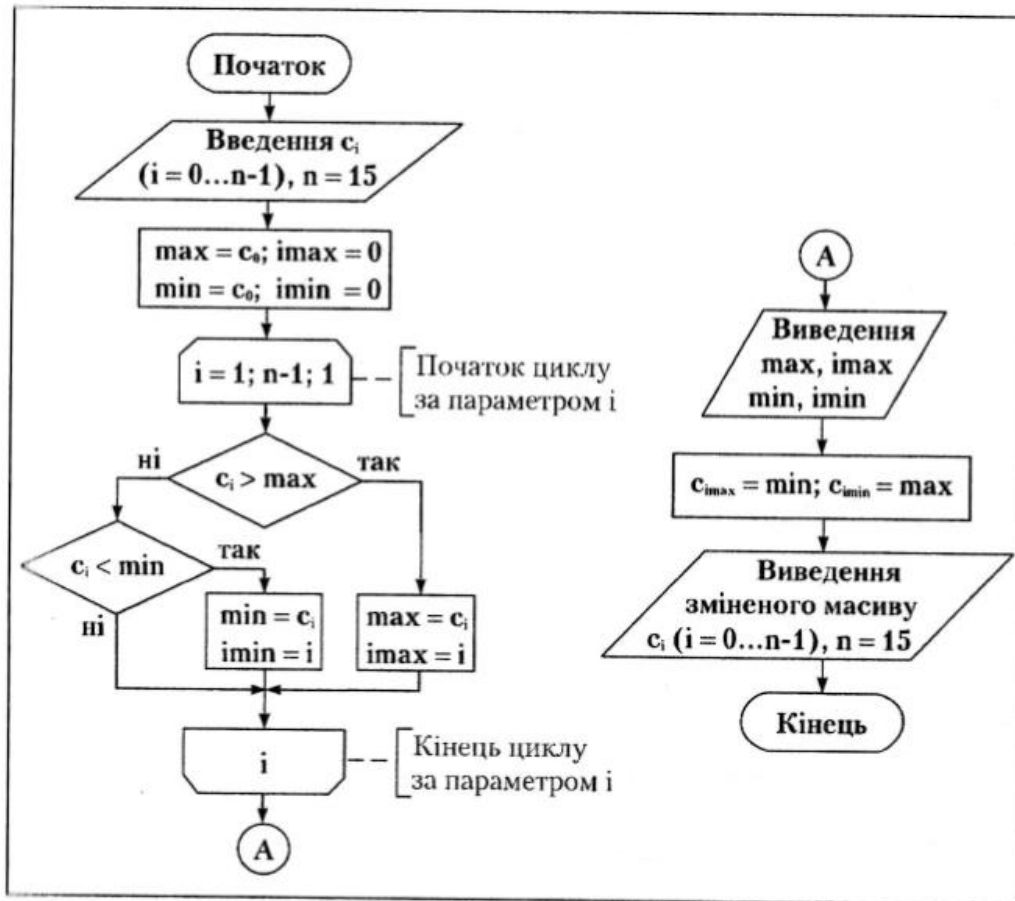


Рис. 1.3. Схема алгоритму прикладу 1.3

Для знаходження максимального та мінімального елементів заданого одновимірного масиву скористаємося типовим прийомом — введенням допоміжної (робочої) змінної. Максимальний елемент позначимо через **max**, а його позицію — **imax**. Відповідно для мінімального елемента та його позиції використовуються змінні **min** та **imin**. Перед переглядом масиву як в **max**, так і в **min** записується перший елемент масиву, тобто елемент, розташований в позиції $i = 0$ (**max** = c_0 та **min** = c_0), а його номер (0) запам'ятовується змінними **imax** та **imin** (**imax** = 0; **imin** = 0).

У процесі порівняння, наприклад, **max** з c_i , якщо змінна **max** буде більше за елемент c_i , то значення **max** зберігається, у протилежному випадку **max** набуде значення c_i . Отже, в будь-якому разі змінна **max** матиме максимальне значення серед c_0 та c_i . Аналогічно аналізуються значення c_2, \dots, c_{n-1} .

Таким чином, здійснюючи пошук максимального елемента серед сукупності чисел, необхідно послідовно їх переглянути і порівняти між собою. Для цього дії над елементами масиву слід реалізувати циклом, параметром якого є індекс i елемента масиву. В тілі циклу зміна значень параметрів **max** та **imax** (тобто **max** = $c[i]$; **imax** = i) відбувається у випадку, коли при перегляді зустрічається елемент c_i , більший за **max** ($c_i > \text{max}$). Значення **min** та **imin** змінюються на нові, якщо $c_i < \text{min}$.

По закінченні циклу одержимо значення та номери максимального і мінімального елементів. Для перестановки цих елементів місцями достатньо виконати дві дії, а саме: **c_{imax}** = **min** та **c_{imin}** = **max**.

Приклад 1.4. Упорядкувати за зростанням масив x_i ($i = 0 \dots n-1$), $n = 12$, з використанням обмінного сортування (за методом «пухирця»).

Сортування — це упорядкування масиву за будь-якою ознакою. Існують різні методи сортування, серед них обмінне сортування (метод «пухирця») — найбільш простий, але і найменш ефективний

метод сортування, його доцільно застосовувати для сортування невеликих масивів. Простота алгоритму (див. рис. 1.4) і програмної реалізації роблять метод досить популярним.

Метод «пухирця» полягає в послідовному порівнянні пар чисел і перестановці їх за необхідності. В процесі упорядкування вихідний масив проглядається зліва направо. Наприклад, на першому кроці сортування за зростанням, якщо елементи задовольняють умові: $x_i > x_{i+1}$, то вони міняються місцями. При цьому i -й елемент (крім першого й останнього) бере участь у двох порівняннях. Таким чином, найбільший елемент масиву буде переміщатися («спливати» як пухирець) до правого кінця масиву і виявиться останнім.



Рис. 1.4. Схема алгоритму прикладу 1.4

На другому кроці сортування масиву виявиться наступний за значенням найбільший елемент, що «спливе» на передостаннє місце. Ці дії повинні повторюватися доти, поки залишиться менше двох невідсортованих елементів масиву. Для масиву з n елементів за один крок сортування виконується $n-1$ порівнянь. На кожному новому кроці сортування масиву кількість порівнянь зменшується на одиницю.

Таким чином, окрім блоків введення та виведення елементів масиву, алгоритм сортування методом «пухирця» має два вкладених цикли:

- зовнішній цикл за параметром k – цикл кроків сортування, який керує багаторазовим виконанням внутрішнього циклу;
- внутрішній цикл за параметром i — цикл порівняння елементів та їх перестановки.

Розглянутий приклад наочно ілюструє застосування одного з типових прийомів алгоритмізації — *робочої змінної (a), яка необхідна для перестановки місцями двох елементів.*

Приклад 1.5. Задана матриця V_{ij} ($i = 0 \dots n-1, j = 0 \dots m-1$), $n = 4, m = 5$. Необхідно для кожного рядка матриці знайти кількість від'ємних елементів та записати їх в одновимірний масив.

Головна умова правильного розв'язання задач з матрицями полягає в розумінні порядку змінювання індексів її елементів. Перший індекс (i) завжди визначає номер рядка, другий (j) — номер стовпця.

Алгоритм розв'язання поставленої задачі (див. *рис. 1.5*) виконується так:

- спочатку елементи масиву V_{ij} ($i = 0 \dots 3, j = 0 \dots 4$) послідовно вводяться в пам'ять комп'ютера, тобто блок введення передбачає використання двох циклів — «за рядками» (за параметром i) та «за стовпцями» (за параметром j). *Двовимірні масиви вводяться «за рядками»*, тобто цикл за параметром i є зовнішнім, а цикл за параметром j — внутрішнім;
- стосовно організації циклічного процесу блок виведення матриці реалізується подібно до блока введення;
- оскільки підраховується кількість від'ємних елементів для кожного рядка матриці, то зовнішній цикл передбачено за параметром i ;
- підрахунок кількості від'ємних елементів виконує параметр **kol** (його початкове значення **kol = 0**);
- для здійснення процесу підрахунку кількості від'ємних елементів рядка слід перебирати всі його елементи, тобто використати цикл за параметром j , що буде внутрішнім відносно циклу за параметром i ;
- для організації лічильника **kol** необхідно скористатися типовим прийомом алгоритмізації (див. *приклад 1.1*), тобто **kol = kol + 1**;
- для запам'ятовування результатів аналізу кожного рядка (**rab_i = kol**) необхідно скористатись типовим прийомом алгоритмізації — формуванням робочого масиву **rab_i** ($i = 0 \dots n-1$), виведення елементів якого здійснюється після перегляду всіх рядків матриці.

У процесі організації вкладених циклів рекомендується дотримуватися такого правила: *параметр циклу, що визначений останнім, повинний мінятися першим* (так, параметр j визначається після i , тому в першу чергу змінюється параметр j).

Доцільно зауважити, що у розглянутому прикладі здійснюється перегляд матриці «за рядками». При перегляді матриці «за стовпцями» порядок зміни індексів протилежний, тобто для кожного j індекс i змінюється потрібну кількість разів.

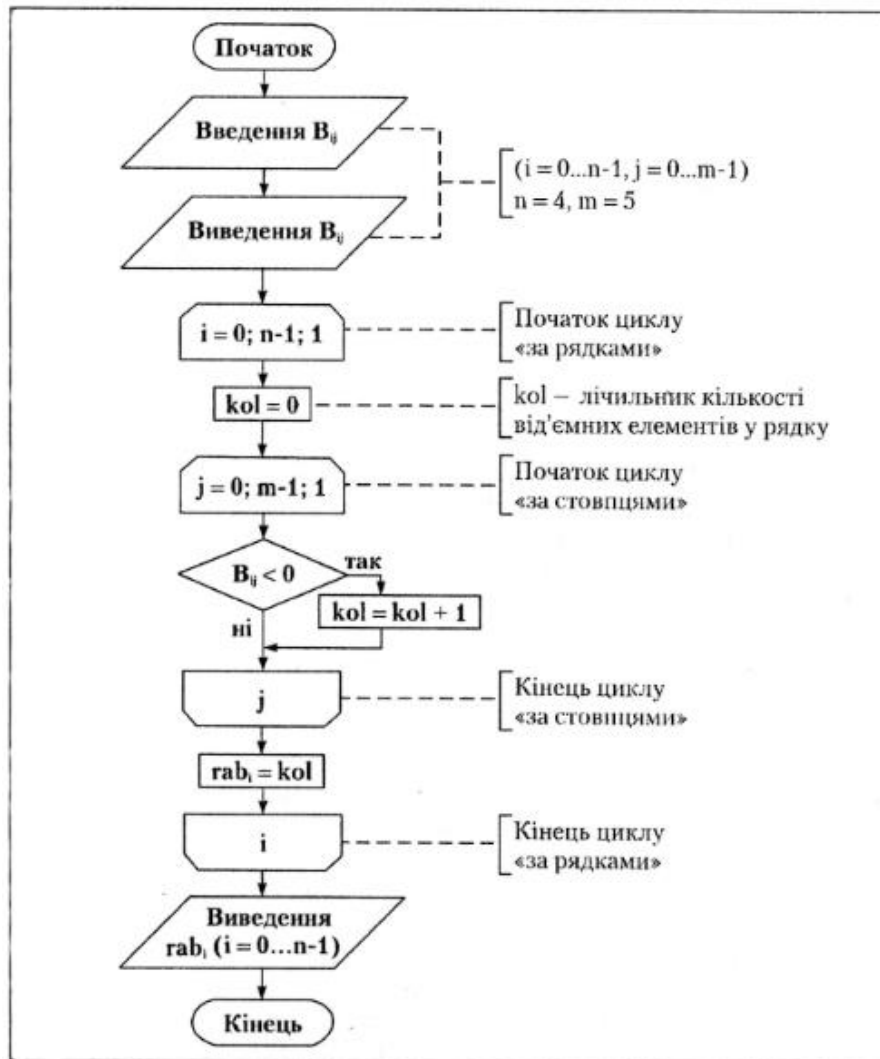


Рис. 1.5. Схема алгоритму прикладу 1.5

Приклад 1.6. Парні елементи заданої матриці a_{ij} ($i = 0 \dots n-1, j = 0 \dots m-1$), $n = 3, m = 4$ переписати до масиву \mathbf{b} , а непарні — до масиву \mathbf{c} . Схему алгоритму розв'язання поставленої задачі зображено на рис. 1.6.

Організація введення-виведення елементів заданої матриці a_{ij} ($i = 0 \dots n-1, j = 0 \dots m-1$), $n = 3, m = 4$, здійснюється аналогічно до розглянутої у **прикладі 1.5**.

Для формування двох одновимірних масивів, в які будуть заноситися відповідно парні та непарні елементи, здійснюється перегляд усіх елементів вихідної матриці. З цією метою реалізуються два вкладених цикли — «за рядками» (за параметром \mathbf{i}) та «за стовпцями» (за параметром \mathbf{j}).

Алгоритм використовує два лічильники: \mathbf{kc} та \mathbf{kn} відповідно для підрахунку кількості парних і непарних елементів (початкові значення $\mathbf{kc} = 0$ і $\mathbf{kn} = 0$). Зрозуміло, що змінні \mathbf{kc} і \mathbf{kn} одночасно являють собою індекси масивів, які створюються. Організація лічильників \mathbf{kc} і \mathbf{kn} здійснюється за типовим прийомом алгоритмізації (див. **приклад 1.1** і **приклад 1.5**).

Після перевірки умови на парність відбувається запис елементів матриці у відповідні масиви: $\mathbf{b}_{kc} = a_{ij}$ та $\mathbf{c}_{kn} = a_{ij}$. Етап формування масивів парних елементів \mathbf{b}_i ($i = 0 \dots kc-1$) та непарних елементів \mathbf{c}_i ($i = 0 \dots kn-1$) завершується після виконання вкладених циклів за параметрами \mathbf{i} та \mathbf{j} .

Виведення одержаних масивів виконується послідовно і передбачає використання циклу за індексною змінною \mathbf{i} . З цією метою можна було б застосувати іншу змінну, тобто змінну з другим ім'ям. Але в подібних випадках краще використовувати змінні з програми, які до потрібного етапу

її виконання закінчили своє функціонування. У даному прикладі параметр i виконав свої функції у циклі визначення парних та непарних елементів матриці і тому може брати участь у процесі виведення елементів масивів.

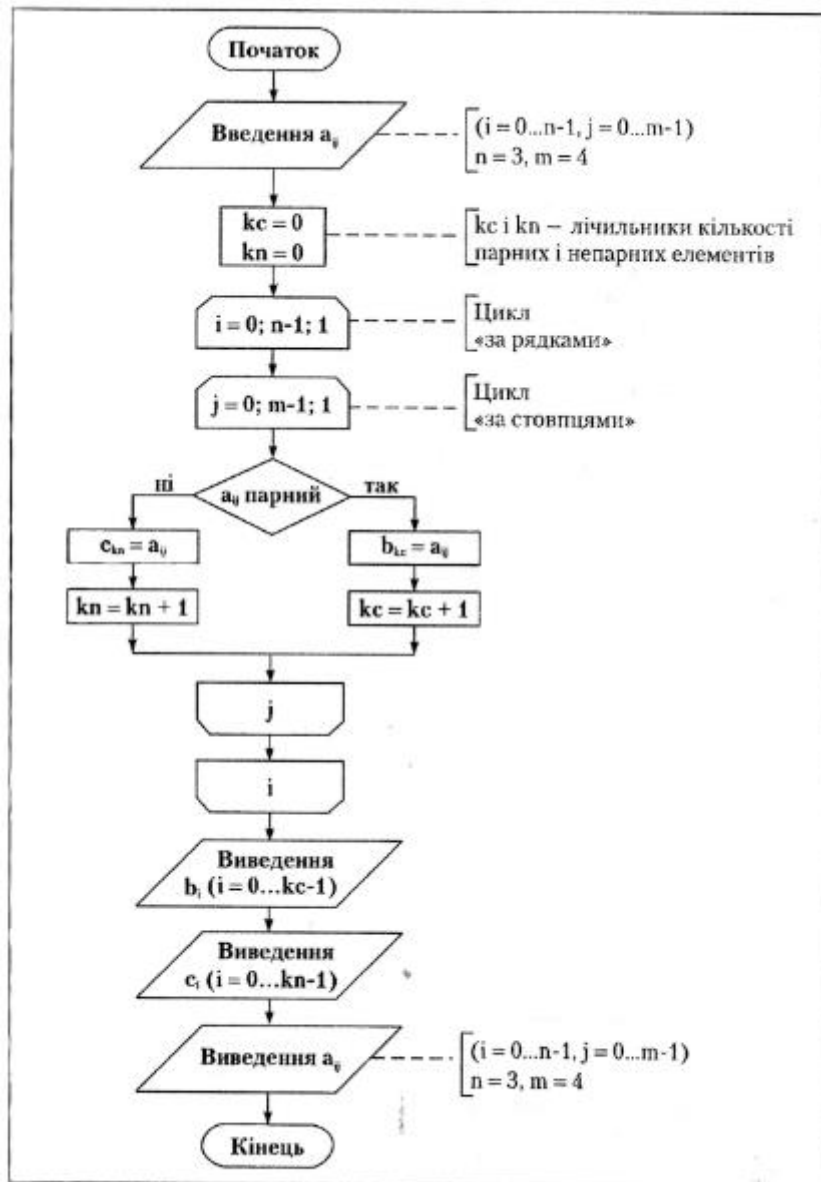


Рис. 1.6. Схема алгоритму прикладу 1.6

Підводячи підсумок, зазначимо, що у розділі наведено реалізації типових структур алгоритмів, а саме: лінійних, розгалужень та циклів. У процесі розробки алгоритму в першу чергу звертається увага на спосіб завдання початкових даних. Розглянуті приклади ілюструють два головних способи завдання даних:

- у вигляді простої змінної, яка змінює своє значення від початкової до кінцевої величини з деяким кроком;
- у вигляді масиву чисел.

Слід зауважити, що у межах посібника схеми алгоритмів наведено з урахуванням важливої особливості мови C++: при роботі з масивами нумерація елементів починається з нуля (0), а не з одиниці (1).