2 Алгоритмічні структури

Повторення

1. Що таке алгоритм?
2. Які способи подання алгоритмів ви знаєте?
3. Хто може бути виконавцем алгоритму?
4. Опишіть принцип роботи сучасної прикладної програми.

У попередньому розділі йшлося про те, що діям програм притаманна певна логічність, що програми можуть аналізувати дані та залежно від ситуації приймати рішення щодо виконання тих чи інших дій. Проте які дії не виконували б програми, вони завжди працюють за чітко визначеним алгоритмом. Таким чином, програмна логіка має бути «прошита» в самих алгоритмах. Сьогодні ми з'ясуємо, як саме алгоритми допомагають моделювати елементи логічного мислення.

Лінійні алгоритми та розгалуження

Розглянемо та порівняємо алгоритми відкриття та збереження текстового документа у програмі Блокнот. Текстові описи алгоритмів наведено на рис. 2.1, а графічні подання — на рис. 2.2.

Структурна відмінність між цими алгоритмами очевидна: в алгоритмі відкриття документа завжди виконуються всі кроки, а чи будуть виконані кроки 4–6 в алгоритмі збереження, залежить від того, чи вперше зберігають документ. Алгоритм, усі інструкції якого виконуються безумовно, називають *лінійним* (див. рис. 2.1, *а* та рис. 2.2, *а*). Алгоритм, у якому виконання чи невиконання певних інструкцій залежить від того, чи справджується деяка умова, отримав назву *алгоритм із розгалуженням*. Чому вживають саме такі назви, зрозуміло з графічного подання (рис. 2.2): всі інструкції лінійного алгоритму можна «витягнути в лінію», розташувати їх одна за одною, а в алгоритмі з розгалуженням є дві чи більше «гілок» і виконання алгоритму кожного разу відбувається за однією з них. На рис. 2.2 ці гілки відходять від інструкції, розміщеної в ромбі; початок однієї гілки позначено словом «так», початок іншої — словом «ні».

1. Вибрати меню Файл.
2. Виконати команду Відкрити.
3. У вікні Відкрити вибрати файл, що відкриватиметься.
4. Клацнути кнопку Відкрити.

а

1. Вибрати меню Файл.
2. Виконати команду Зберегти.
3. Якщо файл ще не було збережено, то виконати кроки 4–6.
4. Вибрати папку, де зберігатиметься файл.
5. Увести ім'я файлу.
6. Клацнути кнопку Зберегти.

б

Рис. 2.1. Текстові описи алгоритмів роботи з програмою Блокнот: *а* — відкриття документа; *б* — збереження документа

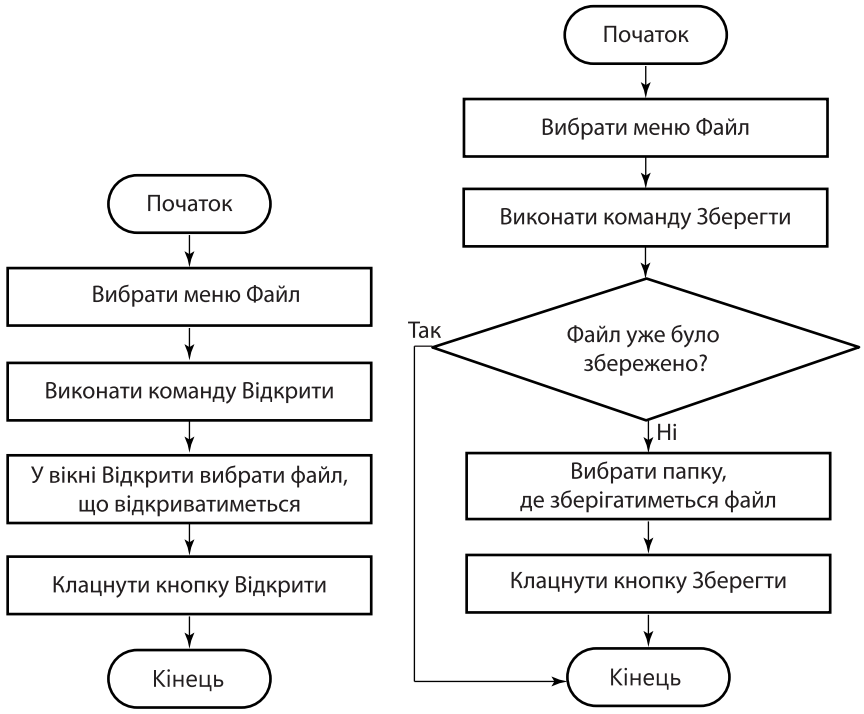


Рис. 2.2. Графічне подання алгоритмів роботи з програмою Блокнот: *а* — відкриття документа; *б* — збереження документа

Блок-схема алгоритму

Графічні подання алгоритмів, подібні до зображених на рис. 1.3, *б* та рис. 2.2, називають *блок-схемами*. Кожна блок-схема складається з блоків, що позначаються різними геометричними фігурами і з'єднуються стрілками. В усіх блоках, крім блоків **Початок** і **Кінець**, записують інструкції алгоритму, по одній у кожному, а стрілками вказують порядок виконання інструкцій. Призначення основних блоків описано в табл. 2.1.

Таблиця 2.1. Основні блоки алгоритму

Назва	Позначення	Призначення
Початок алгоритму	Початок	*Точка входу* в алгоритм. Вказує на інструкцію, що має виконуватися першою
Кінець алгоритму	Кінець	*Точка виходу* з алгоритму — блок, що завершує його виконання
Лінійний блок	Інструкція	Будь-яка інструкція, що не потребує перевірки умов та введення/виведення даних
Розгалуження	Так Умова Ні	Інструкція, що визначає хід виконання алгоритму залежно від істинності деякої умови

Назва	Позначення	Призначення
Введення/ Виведення		Інструкція з уведення даних в інформаційну систему чи виведення даних з неї

З усіх блоків, крім розгалуження та блока кінця, може виходити тільки одна стрілка, адже коли на порядок обчислень не впливає ніяка умова, виконавець має чітко знати, яку інструкцію виконувати наступною. Однак вказувати на будь-який блок, крім блока **Початок**, може довільна кількість стрілок. Наприклад, на рис. 2.2, *б* на блок **Кінець** вказують дві стрілки.

У будь-якій блок-схемі має бути по одному блоку **Початок** та **Кінець**. З очевидних причин на блок **Початок** не може вказувати жодна стрілка, а з блоку **Кінець** жодна стрілка не може виходити. Якщо від деякого блока потрібно повернутися на початок алгоритму, стрілка має вказувати на блок, наступний після блока **Початок**.

Зазначимо, що в табл. 2.1 наведено лише найбільш поширений варіант блока розгалуження. У загальному випадку з нього можуть виходити дві або більше стрілок, над якими можна записувати різноманітні повідомлення, а не тільки слова «так» і «ні». Як приклад на рис. 2.3 подано фрагмент алгоритму, що пояснює дії водія під час проїзду через регульоване перехрестя. У блоці розгалуження перевіряється сигнал на світлофорі, а отже, можливі три варіанти розвитку подальших дій. Розглянемо приклади блок-схем алгоритмів.

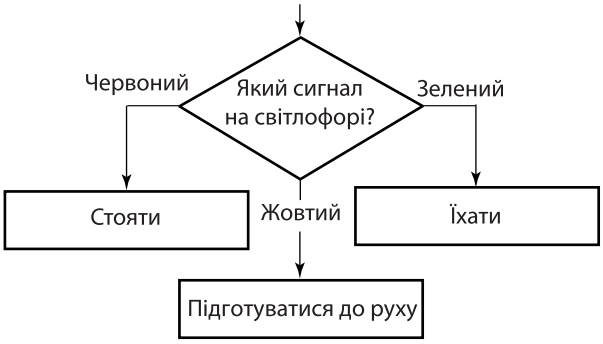


Рис. 2.3. Фрагмент блок-схеми алгоритму дій водія на перехресті

Приклад 2.1

Повернемося до прикладу 1.1 і побудуємо блок-схеми алгоритму обробника події клацання кнопки **/**, а також алгоритму роботи користувача з програмою Калькулятор у цілому.

Логіка обробника клацання кнопки ділення проста: якщо значення в полі **Число 2** дорівнює 0, відобразити повідомлення про помилку, інакше відобразити в полі **Результат** частку **Число 1 / Число 2** (рис. 2.4).

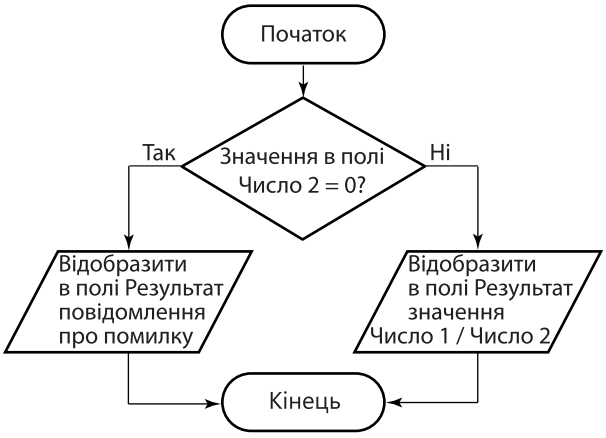


Рис. 2.4. Блок-схема алгоритму, за яким працює обробник події клацання кнопки ділення

Рис. 2.5. Блок-схема алгоритму роботи користувача з програмою Калькулятор

Алгоритм роботи користувача з програмою Калькулятор буде зовсім іншим, оскільки розрахований на іншого виконавця (на користувача, а не на комп'ютер) і охоплює роботу з програмою в цілому, а не з однією кнопкою. Його блок-схему зображено на рис. 2.5.

Завдання 2.1. Побудуйте блок-схему алгоритму дій, які необхідно виконати користувачу ОС Windows для копіювання файлу з однієї папки в іншу. Якщо в цільовій папці вже є файл з таким самим ім'ям, як у того, що копіюється, цей файл потрібно замінити.

Алгоритмічна структура повторення

Як видно з рис. 2.5, у разі отримання у програмі Калькулятор повідомлення про помилку ми маємо повернутися на вже пройдений крок алгоритму, тобто повторити дії. Подібні конструкції в алгоритмах називають *алгоритмічною структурою повторення* або *циклами*. У загальному випадку ця структура має такий вигляд, як показано на рис. 2.6. На них кружечками позначено довільні блоки, а трьома крапками — певні послідовності інструкцій.

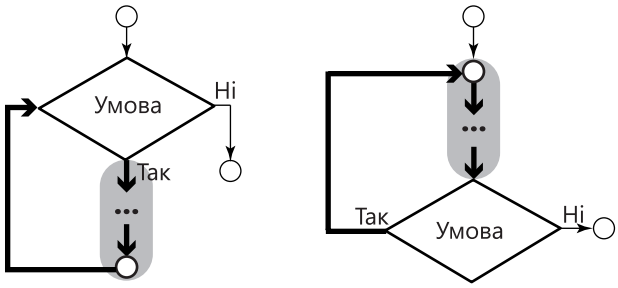


Рис. 2.6. Алгоритмічна структура повторення: *а* — цикл з передумовою; *б* — цикл з постумовою

На рис. 2.6, *а* зображено *цикл з передумовою*, а на рис. 2.6, *б* — *цикл з постумовою*. Цикл з передумовою працює так: спочатку перевіряється умова і, якщо вона істинна, виконуються деякі інструкції та знову перевіряється ця умова. Такі дії повторюються доти, доки умова не стане хибною — тоді здійснюється перехід за стрілкою «ні» до інструкцій, які вже не приведуть до повторної перевірки умови, тобто відбувається *вихід з циклу*. Отже, поки умова виконується, ми «крутитимемось» тим маршрутом, який на рис. 2.6 позначено жирними стрілками. Інструкції, розташовані на цьому маршруті, мають назву *тіло циклу* (на рис. 2.6 воно позначене сірим кольором), а умова — назву *умова продовження циклу*. Одне виконання тіла циклу називають *ітерацією*.

У циклі з постумовою спочатку виконується тіло циклу, а потім перевіряється умова. Якщо умова істинна, ми повертаємося на початок тіла циклу, інакше з циклу виходимо (див. рис. 2.5).

Фактично відмінність між циклами з передумовою та постумовою полягає в тому, що в останньому тіло циклу виконується принаймні один раз, а в першому може не виконуватися жодного разу, якщо умова продовження циклу буде відразу хибною.

Якщо на рис. 2.6 поміняти місцями слова «так» і «ні», то умова продовження циклу перетвориться на *умову завершення циклу*. Такі цикли теж широко використовуються.

Приклад 2.2

Побудуємо алгоритм створення у векторному графічному редакторі, інтегрованому в програми Microsoft Office, такого зображення гусениці, як на рис. 2.7.

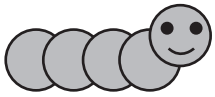


Рис. 2.7. Зображення гусениці

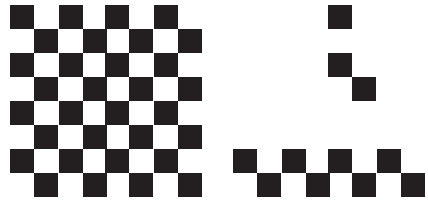
Спочатку наведемо текстовий опис алгоритму.

1. Намалювати коло з чорним контуром і зеленою заливкою.
2. Виділити коло.
3. Утримуючи клавішу **Ctrl**, перетягти коло вправо.
4. Повторювати крок 3, поки не буде намальовано чотири кола.
5. Утримуючи клавішу **Ctrl**, перетягти коло вправо вгору.
6. На колі, що є головою гусениці, намалювати маленьке чорне коло лівого ока.
7. Утримуючи клавішу **Ctrl**, перетягти коло ока вправо.
8. На колі голови намалювати рот інструментом **Дуга**.

Тепер можна побудувати блок-схему алгоритму (рис. 2.8).

Завдання 2.2. У векторному графічному редакторі, вбудованому у програми пакета Microsoft Office, розробіть блок-схему алгоритму креслення шахівниці (рис. 2.9, *а*). Щоб квадрати точно дотикалися кутами, потрібно встановити режим прив'язки одних об'єктів до інших (меню **Малювання ▸ Сітка**). На рис. 2.9, *б* показано проміжні етапи побудови зображення. Зауважте, що алгоритм креслення шахівниці повинен містити два цикли: один для отримання з двох чорних квадратів восьми, а другий — для отримання з восьми чорних квадратів готового зображення.

Рис. 2.8. Блок-схема алгоритму створення зображення гусениці



а *б*

Рис. 2.9. Шахівниця: *а* — готове зображення; *б* — етапи побудови

Змінні та присвоєння

У математиці ви часто стикалися зі змінними. Наприклад, розв'язати рівняння $ax+b=0$ — означає знайти таке значення змінної x, за якого наведений вираз перетворюється на рівність. Змінні широко використовуються і в програмуванні, зокрема під час побудови обчислювальних алгоритмів. Сформулюємо означення змінної з точки зору розробника алгоритму.

Змінна — це найменована величина, яка під час виконання алгоритму може набувати різних значень.

Звичайно, змінна набуває тих чи інших значень не сама по собі, а лише внаслідок виконання відповідних алгоритмічних інструкцій. Такі інструкції називають *присвоєннями*, оскільки вони вказують на необхідність присвоїти змінній значення. У загальному випадку інструкція присвоєння має такий вигляд:

ім'я змінної ← *вираз*

Знак «←» у цій інструкції читають як «присвоїти», а взагалі присвоєння виконується так, як описано далі.

1. Обчислюється значення виразу, вказаного праворуч від знаку «←».
2. Обчислене на кроці 1 значення надається змінній, ім'я якої вказане зліва від знаку «←».

Наведемо приклади присвоєнь:

1. $x \leftarrow 5$ 2. $x \leftarrow y$ 3. $d \leftarrow b^2-4ac$

Завдання 2.3. Визначте, яких значень набудуть змінні x та y після виконання такого алгоритму.

1. $x \leftarrow 5$; 2. $y \leftarrow 7$; 3. $x \leftarrow y$; 4. $y \leftarrow x$

Завдання 2.4. Запишіть алгоритм обміну значеннями між змінними x та y.

Вкладені алгоритмічні конструкції

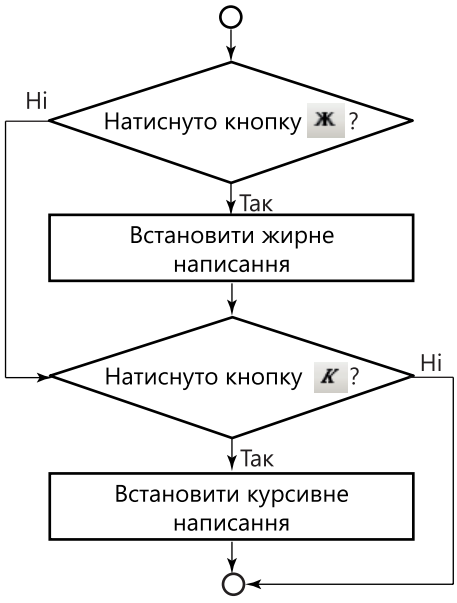
Один алгоритм може містити кілька конструкцій розгалуження. Ці конструкції можуть бути розташовані послідовно, як показано на рис. 2.10, *а*, або вкладені одна в одну — саме таку ви бачите на рис. 2.10, *б*. На рис. 2.10, *а* зображено фрагмент алгоритму, за яким у

програмі Microsoft Word визначається написання символу, що вводиться. Словесно цей фрагмент можна описати так.

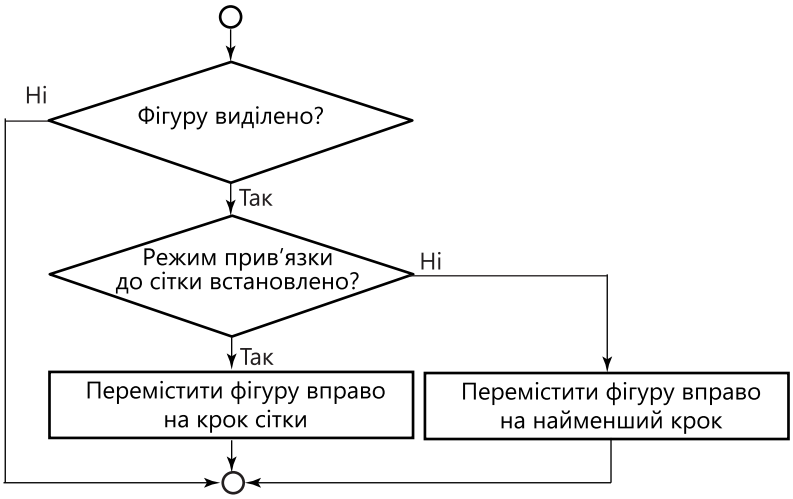
1. Якщо натиснуто кнопку **Ж**, встановити жирне написання.
2. Якщо натиснуто кнопку *К*, встановити курсивне написання.

На рис. 2.10, *б* зображено фрагмент алгоритму, за яким може працювати обробник події натискання клавіші → під час роботи у векторному графічному редакторі, вбудованому в Microsoft Word. Ось як його можна записати у текстовому вигляді.

1. Якщо фігуру виділено, то виконати крок 2.
2. Якщо встановлено режим прив'язки до сітки, перемістити фігуру вправо на горизонтальний крок сітки, інакше перемістити фігуру на мінімальний крок.



а



б

Рис. 2.10. Алгоритми з розгалуженнями: *а* — послідовні розгалуження; *б* — вкладені розгалуження

Завдання 2.5. Побудуйте блок-схему алгоритму розв'язання квадратного рівняння. Випадки, коли рівняння має один чи два корені або взагалі їх не має, повинні оброблятися порізно.

Зазначимо, що розгалуження можуть вкладатися не тільки в інші розгалуження, а й у структури повторення. Можливі й інші комбінації алгоритмічних структур: цикли, вкладені в цикли, та цикли, вкладені в розгалуження. При цьому кількість рівнів вкладеності нічим не обмежується.

Рекурентні обчислення

Побудуємо алгоритм обчислення факторіала натурального числа. Нагадаємо, що факторіалом числа n називають величину $n!$, що дорівнює добутку всіх чисел від 1 до n: $n! = 1 \times 2 \times ... \times (n-1) \times n$. Спробуйте обчислити $5! = 1 \times 2 \times 3 \times 4 \times 5$ і крок за кроком описати свої дії. Скоріш за все, ви отримаєте такий алгоритм:

1. Обчислюємо $2! = 1 \times 2 = 2$.
2. Обчислюємо $3! = 2! \times 3 = 6$.
3. Обчислюємо $4! = 3! \times 4 = 24$.
4. Обчислюємо $5! = 4! \times 5 = 120$.

Ви багато разів застосовували одну й ту саму обчислювальну процедуру: брали результат попереднього кроку обчислень і множили його на чергове число — інакше кажучи, використовували формулу $k! = (k-1)! \times k$. Цю формулу прийнято називати *рекурентною*, оскільки вона визначає, як отримати результат n-го кроку деякої обчислювальної процедури з результату $n-1$-го кроку. Саме обчислення факторіала за описаною схемою також називають *рекурентним*, оскільки в ньому застосовано рекурентну формулу.

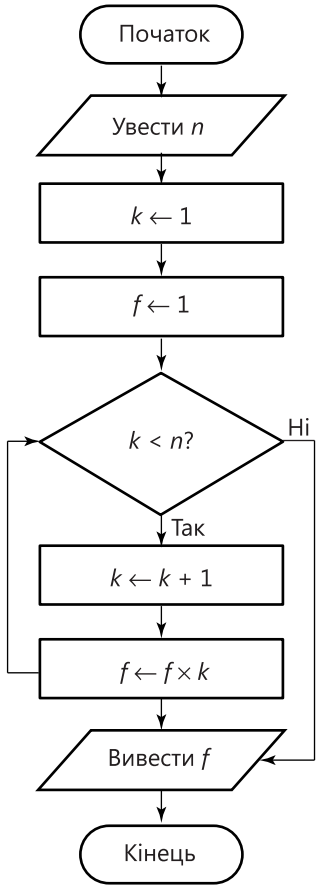


Рис. 2.11. Блок-схема алгоритму обчислення факторіала

Формула, що визначає, як отримати результат n-го кроку деякої обчислювальної процедури з результатів попередніх кроків, називається рекурентною. Обчислення, у якому багаторазово застосовується рекурентна формула, також називають рекурентним.

Оскільки рекурентні формули завжди застосовують багаторазово, це означає, що алгоритм, за яким виконують рекурентне обчислення, завжди містить цикл. Як приклад розглянемо зображену на рис. 2.11 блок-схему циклічного алгоритму, за яким комп'ютер може обчислювати факторіал. Зауважте, що рекурентна формула $k! = (k-1)! \times k$ в ньому набула вигляду $f \leftarrow f \times k$, де f — поточне значення факторіала, а k — номер кроку. Річ у тім, що комп'ютер може оперувати тільки зі змінними, а позначення на кшталт $(k-1)!$ він «не розуміє». Вираз $f \leftarrow f \times k$ можна прочитати так: «новим значенням факторіала f вважати попереднє значення, помножене на k». Зверніть також увагу на допоміжну змінну k, значенням якої є номер кроку, і тому вона має збільшуватися на одиницю кожної ітерації циклу. Це забезпечується завдяки виконанню в тілі циклу інструкції $k \leftarrow k + 1$, яка являє собою рекурентну формулу.

Для допитливих. Винайдення та програмування рекурентних формул — одне з основних завдань програмістів. Без таких формул багато алгоритмів перетворилися б на нескінченні послідовності інструкцій, а отже, описати їх було б неможливо. Наприклад, якщо не використовувати формули $k! = (k–1)! \times k$ та інших рекурентних формул, значення $n!$ доведеться обчислювати за допомогою n різних формул, опис яких матиме невизначений обсяг, оскільки n — нефіксована величина.

Завдання 2.6. Побудуйте блок-схему алгоритму обчислення величини a^n за заданими користувачем значеннями a та n.

Висновки

- Алгоритм, всі інструкції якого виконуються безумовно, називають лінійним.
- Алгоритм, у якому виконання чи невиконання певних інструкцій залежить від того, чи справджується деяка умова, називають алгоритмом з розгалуженням.
- Блок-схема — це графічне зображення алгоритму. Вона складається з блоків, що позначаються різними геометричними фігурами і з'єднуються стрілками. В усіх блоках, крім **Початок** і **Кінець**, записують по одній інструкції алгоритму.
- Алгоритмічну структуру, завдяки якій виконання деяких кроків алгоритму може повторюватися, називають структурою повторення або циклом.
- Тіло циклу — це набір інструкцій, які під час реалізації алгоритму можуть виконуватися багаторазово. Одне виконання тіла циклу називають ітерацією.
- Під змінною розуміють найменовану величину, яка під час виконання алгоритму може набувати різних значень.
- Загальний вигляд інструкції присвоєння такий: *ім'я змінної* ← *вираз*. Під час її виконання спочатку обчислюється значення виразу, вказаного справа від знака «←», а потім це значення надається змінній, ім'я якої вказане зліва від знака «←».
- Формулу, що визначає, як отримати результат n-го кроку деякої обчислювальної процедури з результатів попередніх кроків, а також обчислення, у якому багаторазово застосовується така формула, називають рекурентними.

Завдання для самостійного виконання

1. Побудуйте блок-схему алгоритму, який за координатами вершин чотирикутника визначає його тип: квадрат, прямокутник, ромб, паралелограм чи трапеція. Якщо чотирикутник належить відразу до кількох класів фігур, наприклад є ромбом, а отже, паралелограмом і трапецією, мають виводитися назви всіх цих класів. Якщо чотирикутник

до жодного з перелічених вище класів фігур не належить, нічого не повинно виводитися.

2. Створений у попередньому завданні алгоритм модифікуйте так, щоб виводилася назва тільки одного, найвужчого класу фігур. Наприклад, якщо чотирикутник є квадратом, має виводитися тільки слово «квадрат». Якщо фігура не належить до жодного з перелічених класів, має відображатися текст «чотирикутник загального вигляду».
3. Побудуйте блок-схему алгоритму розв'язання лінійного рівняння $ax + b = 0$. Ви маєте окремо розглянути випадки, коли $a \neq 0$, коли $a = 0$, $b = 0$ і коли $a = 0$, $b \neq 0$.
4. Опишіть алгоритм, який за введеними користувачем значеннями a і b обчислюватиме значення $\frac{a^2 - b^2}{(a-b)^2 + (a+b)^2}$ з використанням лише семи арифметичних операцій (це можуть бути будь-які операції — додавання, віднімання, множення, ділення).
5. Опишіть алгоритм, який за введеним користувачем числом a обчислюватиме значення a^8 із використанням лише трьох операцій множення.
6*. Побудуйте блок-схему алгоритму, що обчислює n перших чисел Фібоначчі. Перші два числа Фібоначчі дорівнюють 1, а кожне наступне — сумі двох попередніх.

Питання для роздумів

1. Яка мінімальна кількість операцій множення потрібна для того, щоб за значенням a обчислити величину a^{15}?
2*. Визначте максимальну кількість способів, якими можна виконати алгоритм, де є n структур розгалуження і немає жодної структури повторення. (Ви маєте розглядати всі алгоритми такого типу.)
3. Визначте максимальну кількість способів, якими можна виконати алгоритм, де є n структур розгалуження та принаймні одна структура повторення. (Ви маєте розглядати всі алгоритми такого типу.)