МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Курс лекцій
з основ інФормаційних технологій
і програмування
для студентів фізико-металургійного факультету
(частина 1. Алгоритмізація
і основи програмування в Delphi)

Затверджено
на засіданні кафедри ОМіП
протокол № 4 від 24.11.2009 р.

Затверджено на засіданні
навчально-видавничої ради ДОННТУ
протокол № 4 від 21.12.2009 р.

2009

УДК 681.3.06 (071)

Курс лекцій з основ інформаційних технологій і програмування для студентів фізико-металургійного факультету (частина 1. Алгоритмізація і основи програмування в Delphi) / О.М.Копитова. – Донецьк: ДОННТУ, 2009. – 72 с.

Дано основні відомості про алгоритми, блок-схеми і мову Object Pascal. Описано базові алгоритми і розглянуто способи їх програмування в середовищі Delphi. Наведено приклади розв’язання типових задач і дано методичні вказівки по їх програмуванню.

Автор: О.М. Копитова, доц.

Відп. за видання В.М. Павлиш, проф.

ЗМІСТ

ЛЕКЦИЯ 1. АЛГОРИТМ І ФОРМИ ЙОГО ПРЕДСТАВЛЕННЯ. ОСНОВНІ СТРУКТУРИ АЛГОРИТМІВ

1.1 Алгоритм і форми його представлення.

Під алгоритмом розумітимемо послідовний процес перетворення початкових даних в результат, що володіє наступними властивостями:

1) дискретність – алгоритм має бути представлений як послідовне виконання простих або раніше визначених кроків;

2) детермінована – застосування алгоритму до одних і тих же початкових даних повинне приводити до однакових результатів;

3) результативність – алгоритм повинен приводити до вирішення задачі за кінцевий час;

4) масовість – алгоритм повинен дозволяти отримувати результат при різних початкових даних в досить широких межах.

В результаті побудови алгоритму математичне формулювання завдання перетвориться у процедуру її вирішення. Ця процедура є послідовністю арифметичних операцій і логічними зв'язками між ними.

Основні форми представлення алгоритмів:

- словесний опис алгоритму;
- графічне представлення алгоритму (блок-схема);
- представлення алгоритму на мові програмування (програма).

Приклад. Скласти словесний опис алгоритму обчислення об'єму (V) прямокутного циліндра по радіусу (r) підстави і висоті (h).

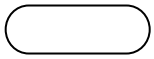
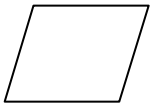
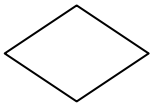
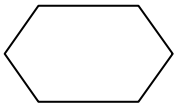
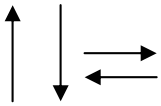
1. Початок алгоритму.
2. Ввести початкові дані: r, h.
3. Обчислити об'єм за формулою: $V=\pi \cdot r^2 h$.
4. Вивести V.
5. Кінець алгоритму.

1.2 Блок-схеми

Блок-схема – наочне графічне зображення алгоритму. Складається із замкнутих фігур стандартної форми (блоків) і стрілок, що сполучають їх. У блоках записуються команди або назви дій, а стрілки указують на лад їх виконання.

Умовне позначення стандартних блоків, їх призначення і найменування приведені в таблиці 1. Блок-схема читається зверху вниз, і в цьому випадку стрілки на лініях потоку можна не указувати. Якщо блок-схема вимагає продовження, то в лівому верхньому кутку кожного блоку ставиться його номер, і за допомогою з'єднувачів указується, з якого блоку і на якій слід передати управління.

Таблиця 1.- Умовні позначення в блок-схемах

Найменування блоку	**Позначення блоку**	**Опис блоку**
Пуск/останов		Зачало або завершення алгоритму
Введення/вивід		Введення/вивід даних
Процес		Виконання арифметичних операцій
Вибір		Перевірка умови
Модифікація		Заголовок циклу
Лінії потоку		Зображення зв'язків між блоками
Внутрішньосторінковий з'єднувач		Вказівка зв'язку між перерваними лініями потоку в межах однієї сторінки
Міжсторінковий з'єднувач		Вказівка зв'язку між частками блок-схеми, розташованими на різних сторінках

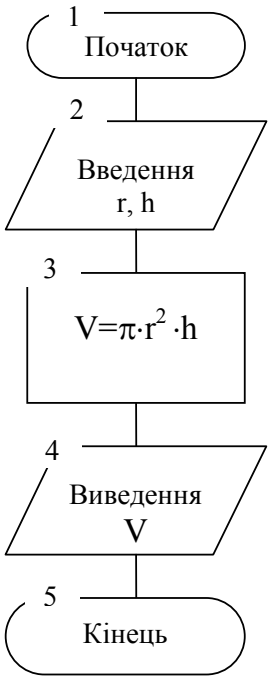
1.3 Основні структури алгоритмів

В процесі розробки алгоритму рекомендується так званий **структурний підхід**, при якому використовуються лише три основні алгоритмічні структури – лінійний алгоритм, розгалужений і циклічний. Математично доведено, що будь-який алгоритм може бути представлений у вигляді комбінації цих основних структур. Особливість основних структур – кожна така структура має рівно один вхід і один вихід. Розгледимо основні структури алгоритмів.

1.4 Лінійний алгоритм

Простим прикладом алгоритму є алгоритм лінійної структури. Він описує обчислювальний процес, в якому операції виконуються послідовно один за одним.

Приклад лінійного алгоритму – алгоритм обчислення об'єму (V) прямокутного циліндра по радіусу (r) підстави і висоті (h). Блок-схема показана на малюнку 1.1. Алгоритм лінійної структури реалізується таким чином. Початок обробки даних – блок 1. У блоці 2 здійснюється введення початкових даних (значень r і h), необхідних для обчислень. У блоці 3 обчислюється об'єм циліндра V. Після обчислень здійснюється виведення результату (блок 4) і останов (блок 5).

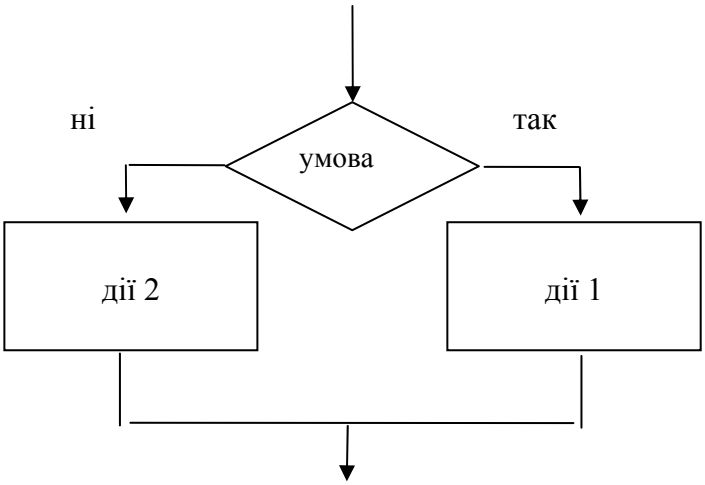


Малюнок 1.1. - Приклад лінійного алгоритму

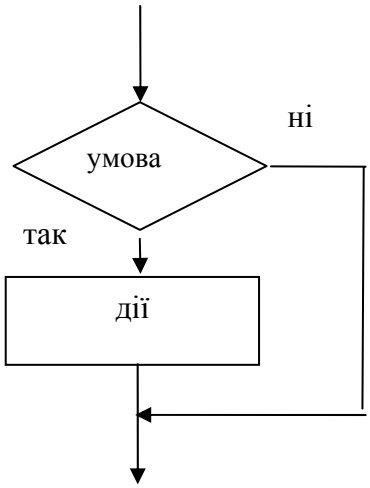
Проте, вирішення абсолютної більшості інженерних завдань неможливо звести до лінійних алгоритмів.

1.5 Алгоритм, що розгалужується

Як правило, обчислювальний процес передбачає декілька можливих шляхів вирішення завдання, реалізація яких залежить від виконання певних умов. Алгоритм (або просто розгалуження), що розгалужується, застосовується в тих випадках, коли залежно від умови необхідно виконати одну або іншу групу дій. На малюнку 1.2 показана блок-схема алгоритму, що розгалужується. Окремий випадок розгалуження – обхід, коли по вітці «ні» ніяких дій виконувати не треба (блок-схема обходу – на малюнку 1.3).



Малюнок 1.2. – Блок-схема розгалуження



Малюнок 1.3. – Блок-схема обходу

Приклад. Обчислити значення f по одній з трьох формул – залежно від значення x:

$$f = \begin{cases} a \cdot x^2 - 2 \cdot x + 7, & x < -1 \\ \dfrac{x \cdot \sin(x)}{(x+5)^2}, & -1 \le x \le 5 \\ \dfrac{\sqrt[3]{x} - 1}{\sqrt[3]{x} + 2}, & x > 5 \end{cases}$$

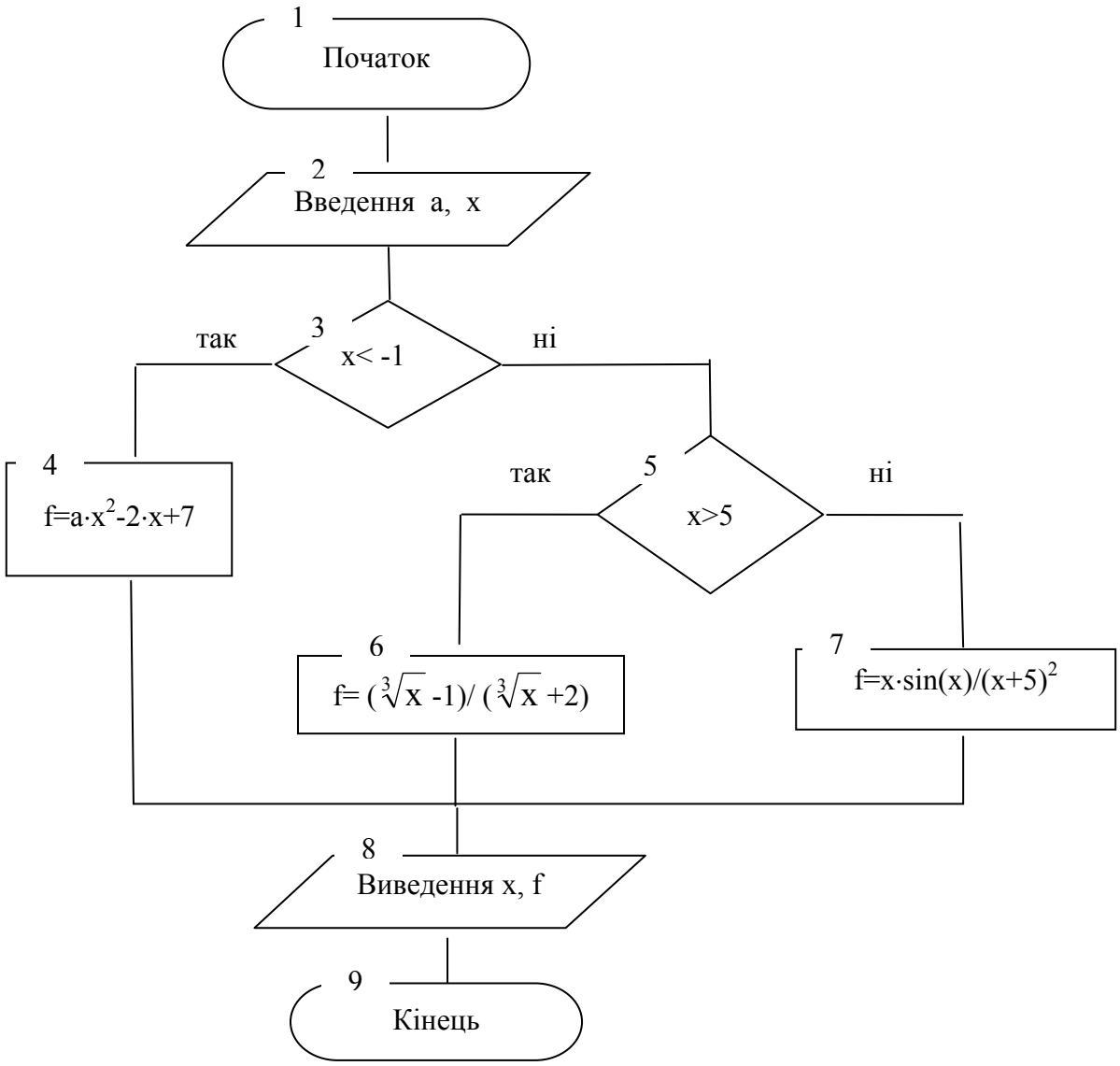
Блок-схема алгоритму даного завдання приведена на малюнку 1.4.

Для обчислення значення f потрібно перевірити два з трьох взаємовиключних умов (для x<-1 і x>5). Після введення початкових даних (блок 2) перевіряється перша умова x<-1 (блок 3). Якщо воно виконується, то значення f визначається по першій гілці формули (блок 4). Інакше перевіряється будь-яка з умов, що залишилися (вони взаємовиключні). В даному випадку в блоці 5 перевіряється умова x>5. Якщо воно виконується, то значення f визначається по третій гілці формули (блок 6), інакше - по другій гілці в блоці 7. У блоці 8 здійснюється виведення результату.

1.6 Циклічні алгоритми

Алгоритм, окремі дії в якому багато разів повторюються, називається **циклічним** (або просто циклом).

Багато разів повторювані дії алгоритму називаються **тілом** циклу. Очевидно, повторювати окремі обчислення доцільно при різних значеннях змінних. Одна з таких змінних називається змінній циклу, що управляє. Значення змінної, що управляє, визначає, буде цикл продовжуватися або він буде завершений.



Малюнок 1.4. Приклад розгалуженого алгоритму

Перед виконанням циклу необхідно привласнити початкові значення змінної циклу, що управляє, і тим змінним, які обчислюватимуться в циклі. Цей етап називається **підготовкою циклу**. Потім необхідно перевірити умову продовження циклу і задати правило зміни змінній, що управляє, для повторного виконання циклу.

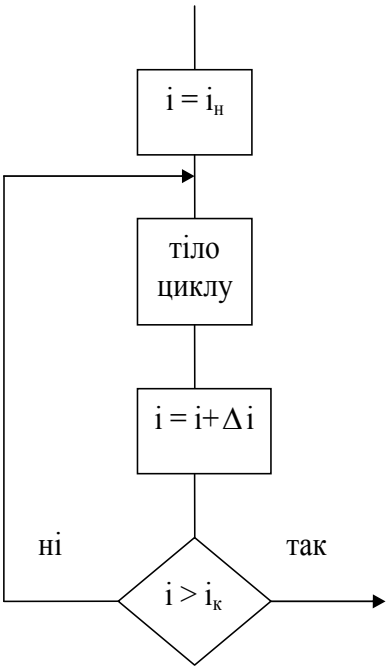
По числу повторень цикли діляться на цикли **з відомим числом повторень** і цикли **з невідомим числом повторень.**

1.7 Цикли з відомим числом повторень

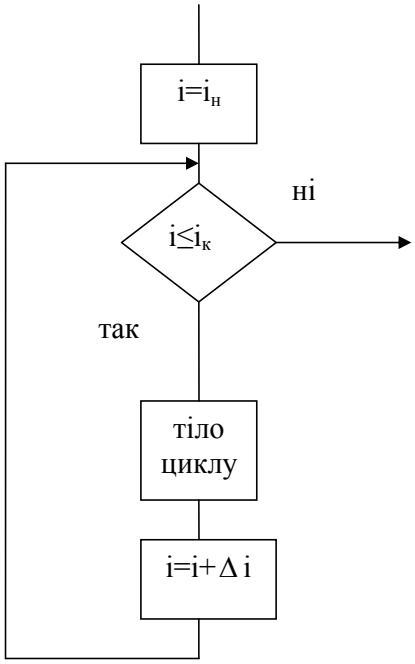
Це цикли, в яких змінна, що управляє, змінюється у відомих межах по відомому закону. Простий випадок – коли змінна і, що управляє, змінюється від свого початкового значення ін до кінцевого значення ік з кроком `Di`. Трійка величин (ін, ік, `Di`) називається параметрами циклу.

На малюнках 1.5–1.7 представлені різні варіанти організації такого циклічного процесу. На малюнку 1.5 показана організація циклу з умовою поста; на малюнку 1.6 – циклу з передумовою; на малюнку 1.7 – циклу з блоком “модифікація”. Останні дві

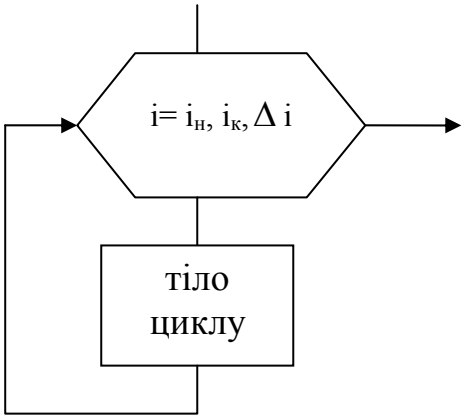
блок-схеми еквівалентні в тому сенсі, що реалізують один і той же обчислювальний процес. Тому, щоб зрозуміти, як працює блок модифікації 1.7, досить звернутися до циклу з передумовою 1.6.



Малюнок 1.5. – Цикл з постумовою



Малюнок 1.6. – Цикл з передумовою

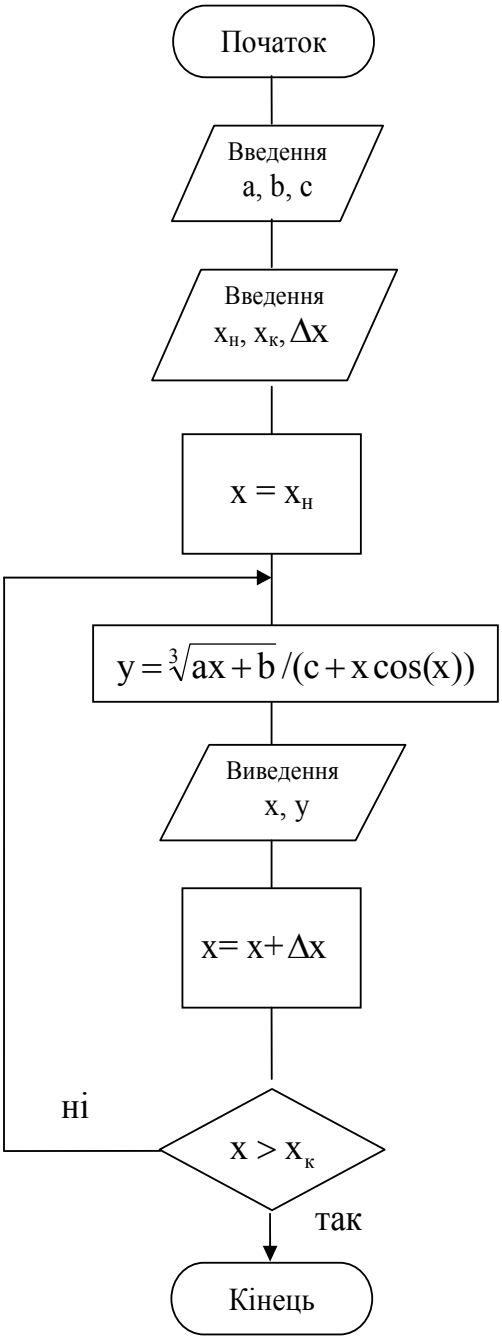


Малюнок 1.7. – Цикл з блоком “модифікація”

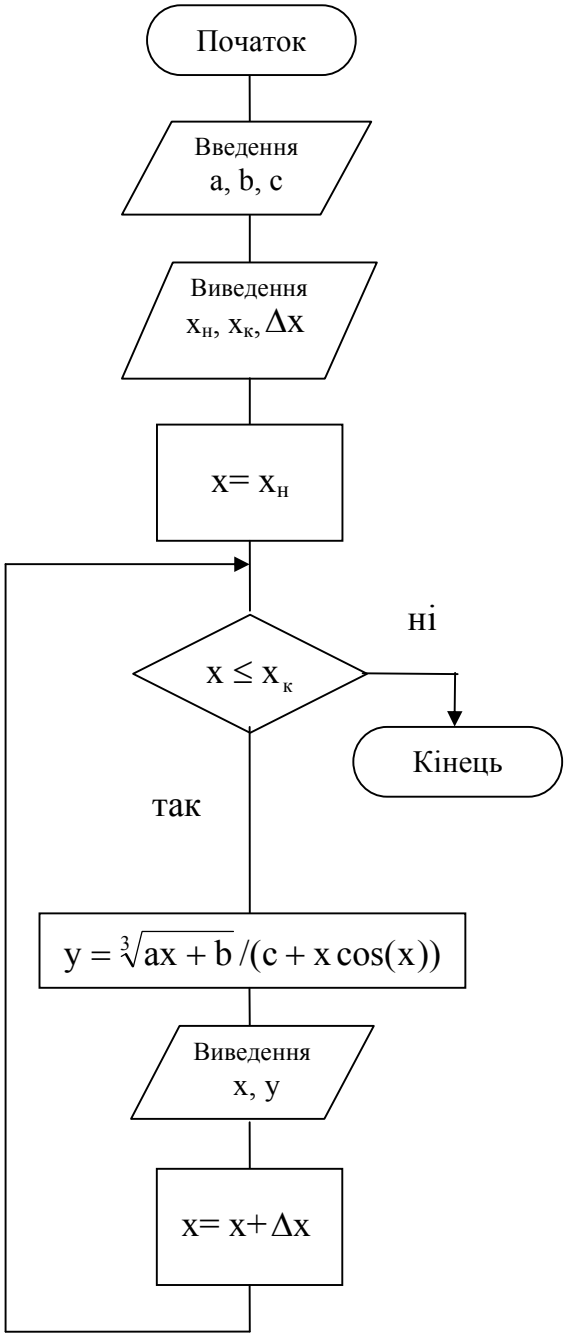
Якщо Δi=1, то в блоці “модифікація” значення Δi не вказують. Кількість m повторень (ітерацій) циклу обчислюється по формулі: $m = [(i_к - i_н)/\Delta i + 1]$, де $[(i_к - i_н)/\Delta i + 1]$ - ціла частина величини $(i_к - i_н)/\Delta i$.

Приклад. Обчислити значення функції $y = \sqrt[3]{ax + b}/(c + x\cos(x))$ $2 \le x \le 8$ $\Delta x = 0{,}4$. Значення a, b, із задані.

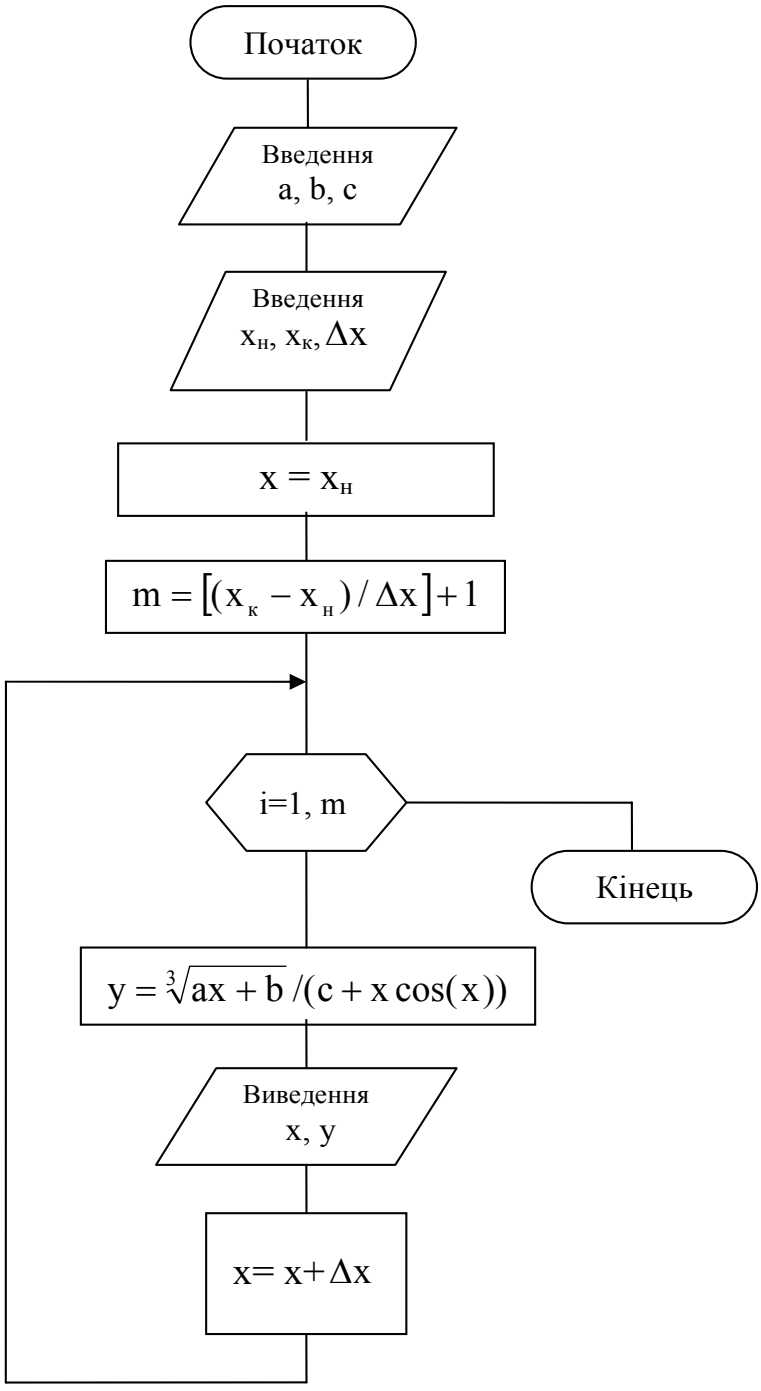
Для вирішення цього завдання потрібно в циклі перебрати всі значення x від xн=2 до xк=8 з кроком $\Delta x = 0{,}4$ і для кожного з них набути значення. Різні варіанти реалізації циклічного процесу для даного завдання показані на малюнках 1.8 - 1.10.



Малюнок 1.8. - Обчислення y (цикл ПОКИ)



Малюнок 1.9. - Обчислення y (цикл ДО)

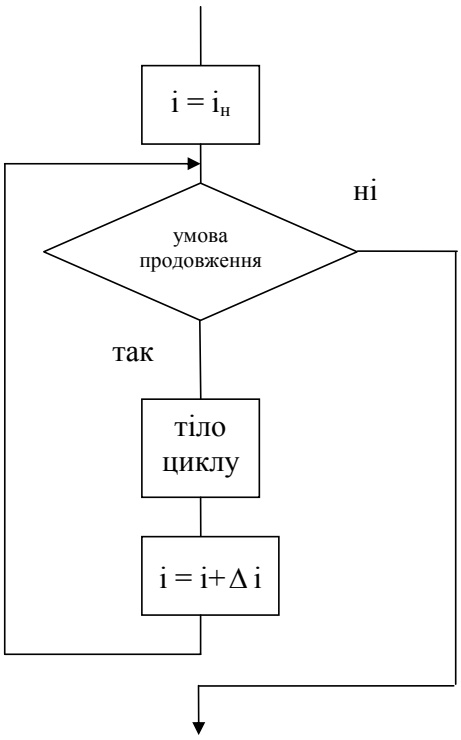


Малюнок 1.10. – Обчислення y (цикл з блоком “модифікація”)

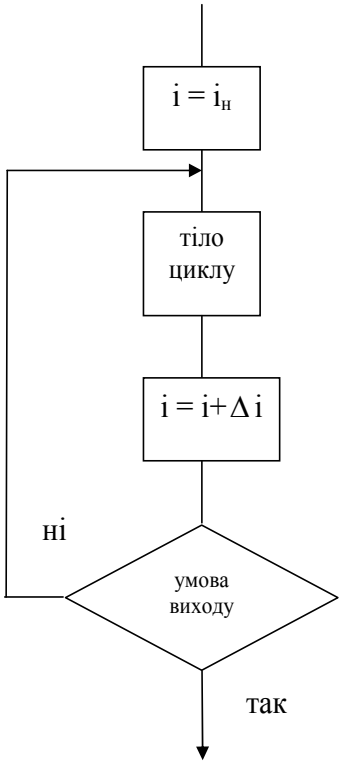
1.8 Цикли з невідомим числом повторень

У таких циклах число повторень циклу заздалегідь не визначене, а обчислювальний процес завершується при настанні деякої умови. Щоб підрахувати кількість повторень циклу, необхідно організувати лічильник, який слід обнулити до початку циклу.

Цикли з невідомим числом повторень можуть бути двох типів – з передумовою (їх також називають циклами ПОКИ) і з умовою поста (цикли ДО). Блок-схеми цих циклів показані на малюнках 1.11 і 1.12.



Малюнок 1.11. – Цикл ПОКИ (з передумовою)



Малюнок 1.12. – Цикл ДО (з постумовою)

Тут x – змінна циклу, що управляє, входить в умови продовження і виходу з циклу. Помітимо, що умови, які перевіряються в цих циклах, взаємо протилежні: у циклі ПОКИ перевіряється умова продовження циклу, а в циклі ДО – умова виходу з циклу.

Особливість циклу ПОКИ: якщо при першій перевірці умова продовження порушується, те тіло циклу не буде виконано жодного разу.

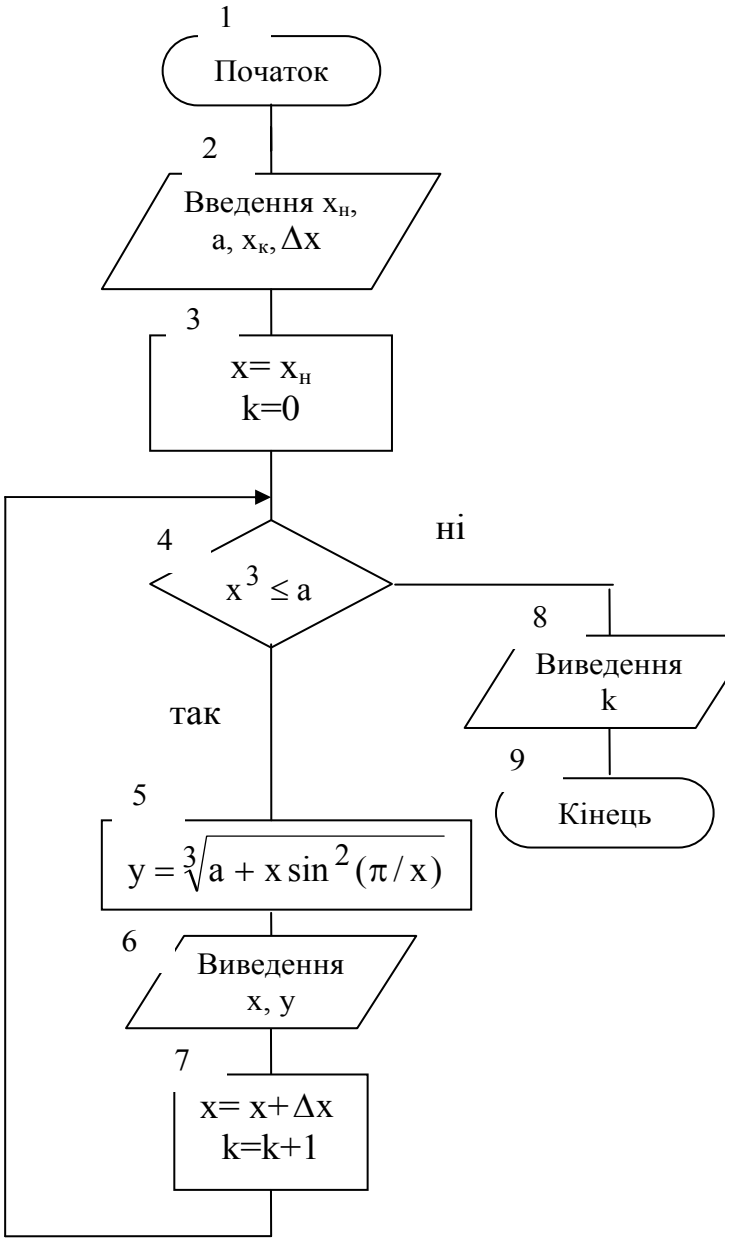
Особливість циклу ДО: тіло циклу завжди виконується хоча б один раз.

В обчислювальному плані ці цикли еквівалентні, тобто в алгоритмі завжди можна замінити цикл ПОКИ циклом ДО і навпаки

Приклад. Обчислити значення y по формулі $y = \sqrt[3]{a + x\sin^2(\pi/x)}$ для $x \geq 1$, $\Delta x = 0{,}8$. Обчислення y виконувати доти, поки значення x^3 стане більше a. Визначити кількість обчислених значень y. Вивести всі значення y і їхню кількість.

Особливістю цієї задачі є те, що для організації циклу неможливо використовувати блок модифікації, тому що невідомо кінцеве значення параметра x, при якому буде досягнута умова $x^3 > a$. Блок-схема розв'язання задачі представлена на малюнку 4.13.

У блоці 2 відбувається введення значень a, $x_н$, Δx. У блоці 3 присвоюються початкові значення змінній x, лічильникові кількості повторень циклу k. У блоці 4 перевіряється умова продовження циклу: поки умова виконується, у блоці 5 обчислюються значення y, у блоці 6 виводяться значення змінних x і y, підготовляються значення змінних x і k для наступного проходження циклу. Як тільки змінна x досягне такого значення, що $x^3 > a$, виводиться значення лічильника k (блок 8) і алгоритм завершується.



Малюнок 1.13. – Приклад циклу з невідомим числом повторень

ЛЕКЦІЯ 2. БАЗОВІ АЛГОРИТМІЧНІ ЗАВДАННЯ

2.1 Вкладені цикли

Цикл, до складу якого не входять інші цикли, називається **простим.**

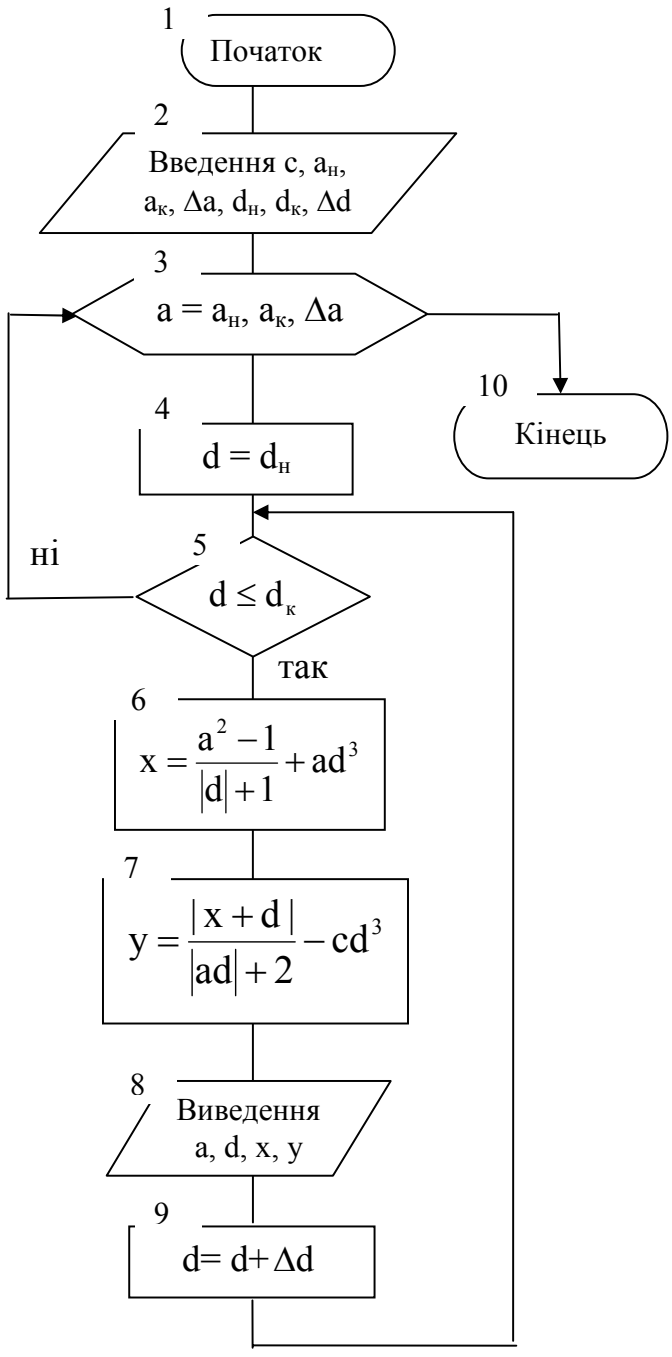
При розв'язанні задач може виникнути необхідність організувати цикл усередині циклу. Якщо до складу циклу входить інший цикл, то говорять про пару **вкладених циклів**. При цьому перший цикл називається **зовнішнім,** а вкладений у нього – **внутрішнім**. Кожний з пари вкладених циклів має свою керуючу змінну і свої параметри. При виконанні вкладених циклів діє правило: **у першу чергу завжди**

виконується самий внутрішній цикл. Таким чином, для кожного значення керуючої змінної зовнішнього циклу керуюча змінна внутрішнього циклу послідовно пробігає усі свої значення.

Усередині вкладеного циклу може знаходитися ще один вкладений цикл і т.д. Той самий цикл може бути зовнішнім стосовно одного і внутрішнім стосовно іншого циклу. Границі внутрішнього циклу не можуть виходити за границі зовнішнього циклу.

Приклад. Обчислити значення x і біля: $x = \frac{a^2 - 1}{|d| + 1} + ad^3, y = \frac{|x + d|}{|ad| + 2} - cd^3$, де c=2,8; $2 \le a \le 9$; $\Delta a = 1$; $0,6 \le d \le 3,5$; $\Delta d = 0,2$. Вивести всі пари x, біля.

Блок-схема обчислення значень x і біля показана на малюнку 2.1.



Малюнок 2.1. - Блок-схема обчислення значень x і y (вкладений цикл)

Після введення вхідних даних (блок 2) організовано два цикли для обчислення значень x і y. Зовнішній цикл організований за допомогою блоку “модифікація”, а внутрішній цикл організований як цикл з передумовою. Зовнішній цикл починається з блоку 3 модифікації, у якому задається закон зміни керуючої змінної a: від $a_н$ до $a_к$ із кроком Δa (параметри $a_н$, $a_к$, Δa при введенні одержали свої конкретні значення – відповідно 2, 9 і 1). Кожному значенню змінної a відповідають 15 значень змінної d, що змінюється в межах від $d_н$=0,6 до $d_к$=3,5 із кроком Δd=0,2. Тому наступний блок 4 є підготовкою до внутрішнього циклу з керуючою змінною d. У блоці 4 керуючій змінній d присвоюється її початкове значення. У блоці 5 – перевіряється умова продовження циклу. У блоках 6 і 7 обчислюються значення x і y. У блоці 8 здійснюється виведення значень a, d, x і y. У блоці 9 керуюча змінна d внутрішнього циклу змінюється на крок Δd і відбувається повернення на початок циклу по d.

Після закінчення внутрішнього циклу керування передається на заголовок зовнішнього циклу (блок 3). Тут відбувається перехід до наступного кроку зовнішнього циклу. При цьому значення змінної a збільшується на крок Δa і перевіряється умова продовження циклу ($a \le a_к$?) – саме так працює блок модифікації (див. мал. 1.7 і мал. 1.6). Коли значення змінної, що управляє, а стає більше ак, зовнішній цикл закінчується і управління передається блоку 10.

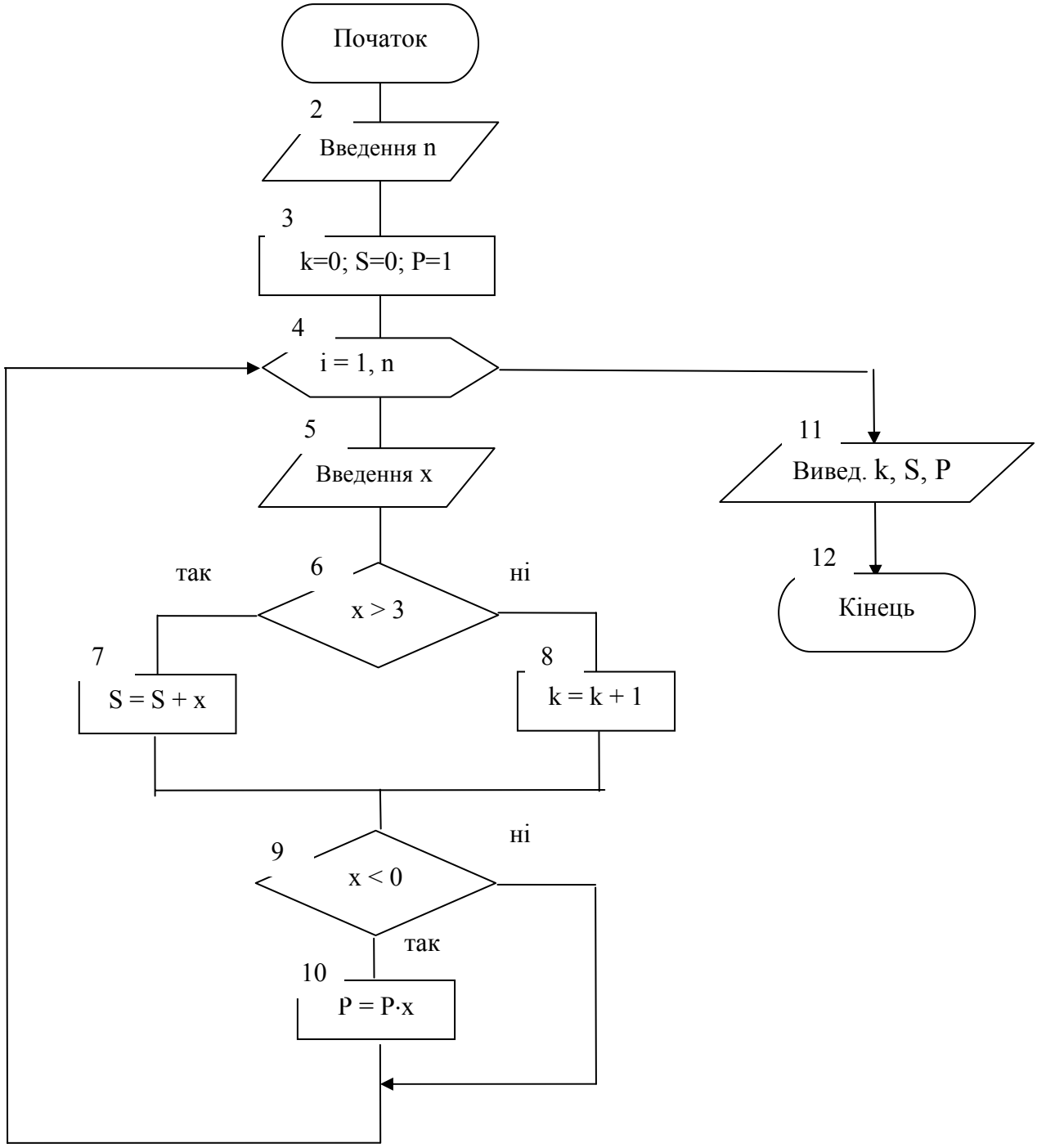
2.2 Обчислення суми/добутку послідовності чисел

При обчисленні суми/добутку послідовності чисел застосовується **принцип накопичення**. Виділяється змінна, в якій накопичуватиметься результат, і в неї послідовно заноситься сума/добуток одного, два, три і так далі чисел. Заздалегідь цією змінною привласнюється початкове значення: для суми – 0, для добутку – 1.

Приклад. Ввести n чисел. Обчислити: S – суму тих з них, які більше 3, P – добуток тих, які менше 0 і до, – кількість чисел, менших або рівніших 3.

Блок-схема вирішення представлена на малюнку 2.2.

Початкові дані в цьому завданні: n – кількість чисел і самі числа. Значення n слід ввести **до циклу**, а самі числа вводитимемо в змінну x в циклі. У блоці 2 вводиться n. Блок 3 привласнює змінним до, S, P їх початкові значення – це підготовка до циклу. Блок 4 – початок циклу по i, де i – номер числа в послідовності. Блоки 5–10 складають тіло циклу. У блоці 5 чергове число вводиться в змінну x. У блоках 6–8 методом накопичення обчислюються до і S. Оскільки умови x>3 і x3≤, які відповідно треба перевірити для обчислення S і до, взаємно протилежні, то досить організувати одне розгалуження. У блоках 9,10 методом накопичення обчислюється добуток P. Для цього організовується розгалуження, в якому перевіряється умова x < 0. Відмітимо, що це розгалуження є обходом. Після закінчення циклу в блоці 11 виводиться результат – знайдені значення до, S і P.



Малюнок 2.2. – Приклад обчислення суми/добутку послідовності чисел

2.3 Пошук мінімального/максимального значення послідовності чисел

Пошук мінімального (максимального) значення здійснюється в циклі. Для зберігання мінімального (максимального) значення виділяється змінна, наприклад, min (max). Спочатку, до циклу, в неї заноситься перше число послідовності. Потім, в циклі, чергове число x послідовності порівнюється з min (max). Якщо виявилось, що x<min (x>max), то в змінну min (max) заноситься значення x. Перше число послідовності само з собою порівнювати не треба, тому цикл починається з другого елементу. Таким чином, в змінну min (max) послідовно заноситься мінімальне (максимальне) зі всіх попередніх чисел.

Приклад. Для -2 x 6; Δx=0,4 .обчислити Z обчислити за формулою: $z = 1/7 + e^x \ln|\pi - x|$. Знайти максимальний Z серед тих, які менше 3 (результат занести в змінну max) і мінімальний серед всіх Z (результат – в змінну min).

Блок-схема вирішення задачі представлена на малюнку 2.3.

У блоці 2 здійснюється введення початкових даних. У блоці 3 змінною x привласнюється її початкове значення. У блоці 4 обчислюється кількість повторень циклу. Блок 5 організовує цикл по змінній i. Блоки 6 і 7 забезпечують формування і виведення чергового значення Z. У блоках 8 і 9 шукається найбільше значення Z, менше 3. Аналогічно, в блоках 10 і 11 здійснюється пошук найменшого значення Z. У блоці 12 здійснюється перехід до чергового значення x. Блок 13 виводить результати: максимальне і мінімальне значення порахованої величини Z.

ЛЕКЦІЯ 3. АЛГОРИТМИ З ВИКОРИСТАННЯМ МАСИВІВ

3.1 Обробка одновимірних масивів

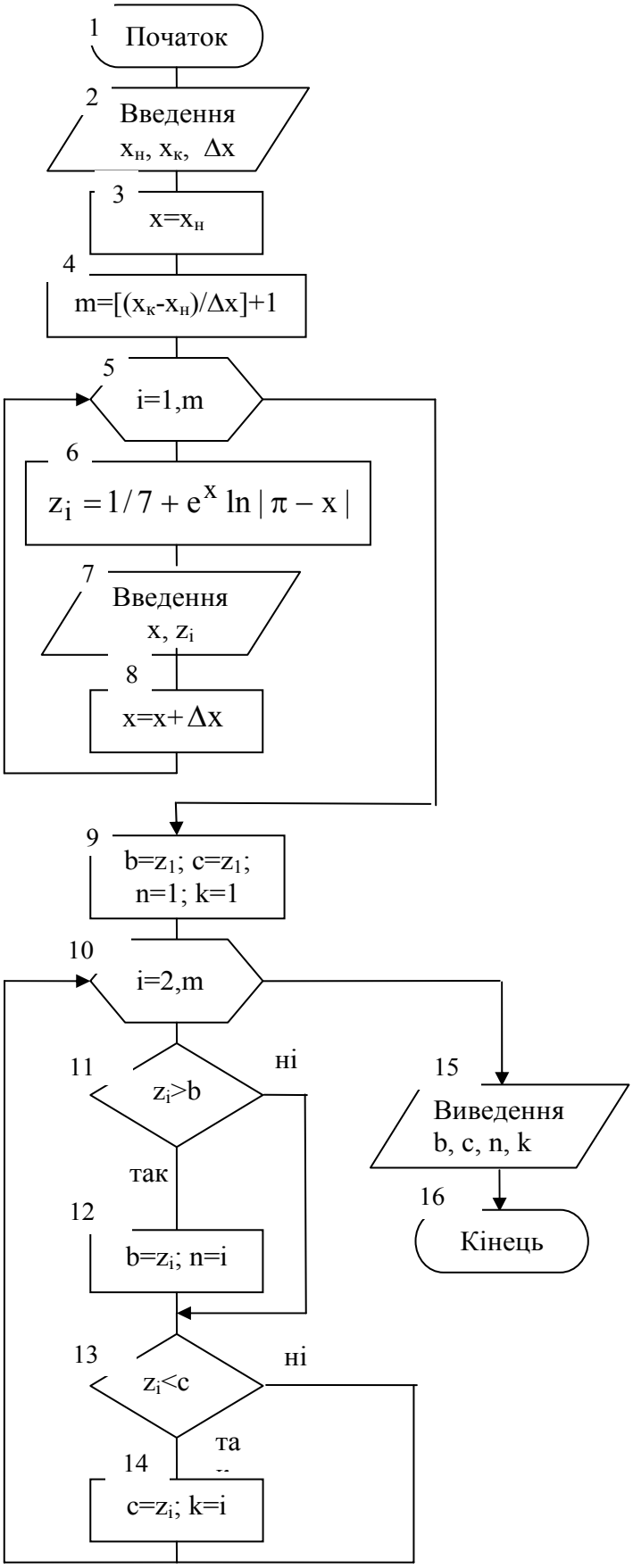
Масив – це **упорядкована** послідовність **однотипних** величин, що позначається **одним ім'ям**. Окремий елемент масиву визначається списком індексів. **Список індексів** – цілі числа, що однозначно визначають місце розташування елемента в масиві. Для одномірного масиву місце розташування його елемента визначається єдиним індексом. Наприклад, x_{10} – десятий елемент масиву $X=(x_1,x_2,\ldots,x_{20})$.

Кількість елементів у масиві називається його **розмірністю.** Так, розмірність масиву $Y=(y_1,y_2,\ldots,y_m)$ дорівнює m. Для роботи з масивом необхідно організувати цикл, у якому здійснити обробку кожного елемента масиву. Введення/виведення масивів здійснюється окремо для кожного елементу. Для введення масиву треба спочатку (до циклу) увести його розмірність, а потім (у циклі) увести кожен елемент масиву. Аналогічно, виведення масиву здійснюється в циклі.

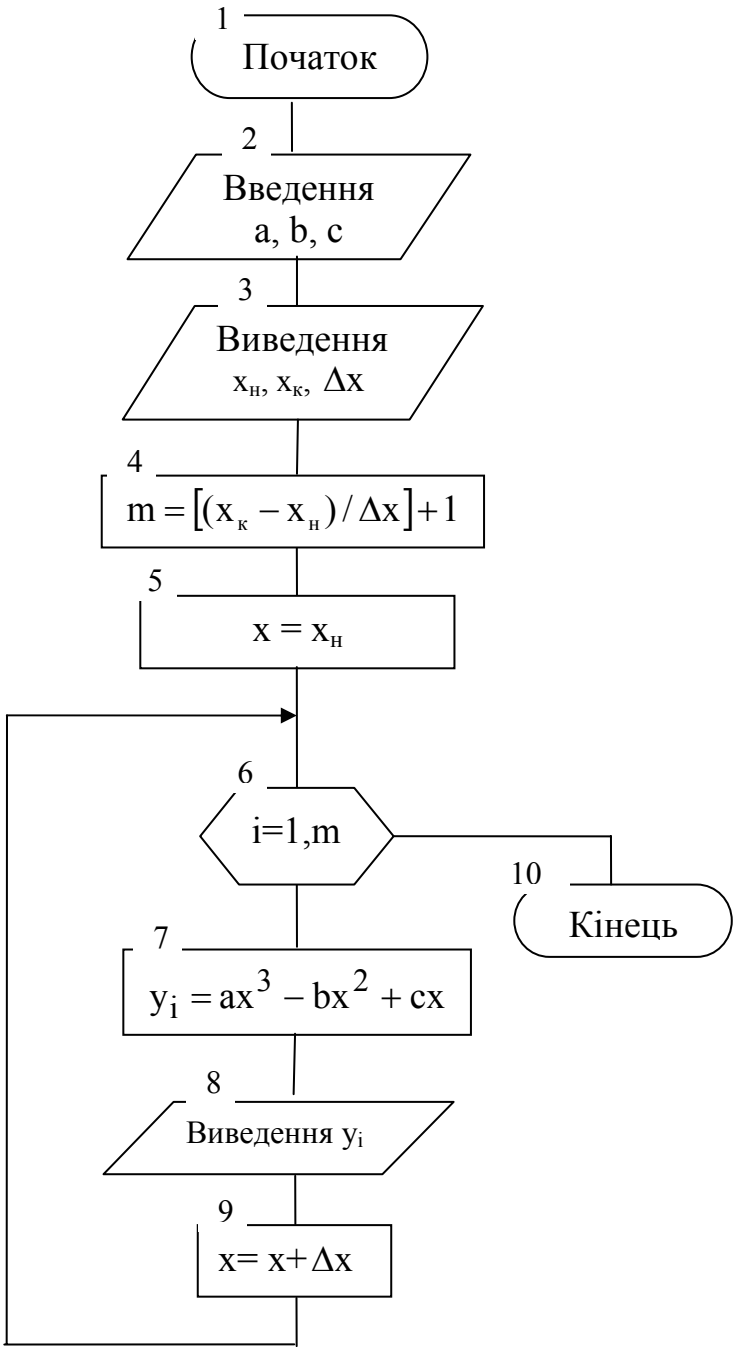
Приклад. Обчислити значення елементів масиву $Y=(y_1,y_2,\ldots,y_m)$ по формулі $y_i = ax^3 - bx^2 + cx$ для $0 \le x \le 10$, $\Delta x = 2{,}5$. Значення a, b, c задані.

Блок-схема алгоритму цієї задачі приведена на малюнку 3.1.

Після введення початкових даних (блоки 2 і 3) визначається m – число повторень циклу, рівне кількості значень x (блок 4). Далі змінною x привласнюється початкове значення (блок 5). У блоках 6–8 організований циклічний процес для обчислення елементів масиву Y. У циклі формується масив Y з m елементів. Змінна i, що управляє, змінюється від 1 до m з кроком 1, що визначене блоком модифікації (блок 6). При цьому значення y1 обчислюється для x=xн (x=0, блок 7), після чого x збільшується на крок і управління передається на заголовок циклу (блок 6). Значення y2 обчислюється для x=2,5, і операції в циклі повторюються в тому ж ладі, що і для y1. Значення y3 обчислюється для x=5, значення y4 обчислюється для x=7,5 і значення y5 обчислюється для x=10. На кожному кроці циклу після обчислення елементу масиву Y здійснюється виведення його значення. Таким чином, після закінчення циклу будуть виведені всі елементи масиву Y.



Малюнок 2.3. – Приклад пошуку мінімального і максимального значень



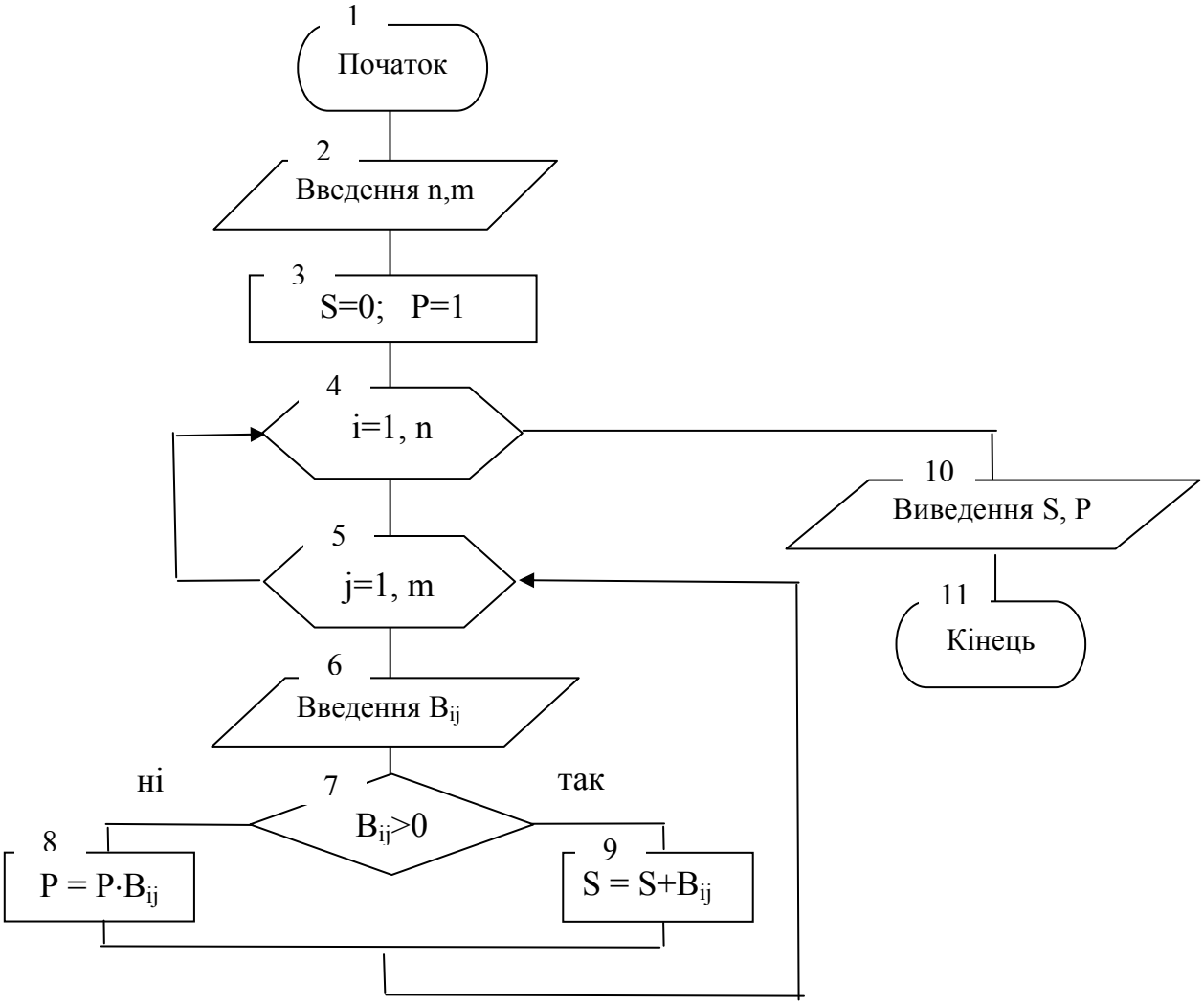
Малюнок 3.1. – Приклад обробки одновимірного масиву

3.2 Обробка двовимірних масивів

Двовимірний масив – це таблиця (матриця) $A=(a_{ij})_{n,m}$ з числом рядків **n** і числом стовпців **m**. На перетинанні рядків і стовпців знаходяться однотипні однойменні упорядковані елементи a_{ij}. Кожен елемент визначається двома **індексами (i,j)**, що вказують номер рядка (i) і номер стовпця (j). Про матрицю А говорять, що вона має **розмірність nxm**.

Приклад. Для масиву цілих чисел $B=(b_{ij})_{n,m}$ знайти добуток (P) негативних і суму (S) позитивних елементів.

Блок-схема алгоритму приведена на малюнку 3.2.



Малюнок 3.2. - Блок-схема обробки двовимірного масиву

Початкова матриця містить n рядків і m стовпців. Розв'язання цієї задачі зводиться до перевірки в циклі знаку чергового елементу і накопичення добутку, якщо цей елемент негативний, і накопиченню суми, якщо цей елемент позитивний. Можна організувати проглядання масиву по рядках (зовнішній цикл по першому індексу, внутрішній цикл по другому індексу) або по стовпцях (навпаки).

У блоці 2 вводиться розмірність матриці – n і m. У блоці 3 привласнюються початкові значення сумі і добутку. Блоки 4,5 організують пару вкладених циклів: зовнішній цикл – по рядках (індекс i), внутрішній – по стовпцях (індекс j). Блок 6 здійснює введення елементів матриці B по рядках. У блоці 7 перевіряється знак чергового елементу. У блоці 8 накопичується добуток негативних елементів матриці, а в блоці 9 накопичується сума її позитивних і нульових елементів. Блок 10 виводить результати розрахунку.

3.3 Таблиця трасування

Перевірка правильності алгоритму уручну здійснюється за допомогою таблиці трасування. У верхній рядок таблиці виписуються імена всіх змінних, присутніх в блок-схемі. Потім алгоритм прораховується послідовно, блок за блоком, зачинаючи

від блоку «зачало» і закінчуючи блоком «кінець», і кожне пораховане значення заноситься у відповідний стовпець.

Приклад. Перевіримо правильність роботи попереднього алгоритму для матриці $B = \begin{vmatrix} 2 & 4 & -6 \\ 0 & -2 & 5 \\ -3 & 8 & 1 \end{vmatrix}$, розмірності 3 ×3. Складемо таблицю трасування:

n	**m**	**i**	**j**	**B**	**S**	**P**	**№блока**
3	3				0	1	2,3
		1	1	B11=2	0+2=2		4,5,6,7,9
			2	B12=4	2+4=6		5,6,7,9
			3	B13=-6		1*(-6)=-6	5,6,7,8
			4				5
		2	1	B21=0	6+0=6		4,5,6,7,9
			2	B22=-2		(-6)*(-2)=12	5,6,7,8
			3	B23=5	6+5=11		5,6,7,9
			4				5
		3	1	B31=-3		12*(-3)=-36	4,5,6,7,8
			2	B32=8	11+8=19		5,6,7,9
			3	B33=1	19+1=20		5,6,7,9
			4				5
		4					4
					20	-36	10

Як бачимо, остаточні значення порахованих величин є: S=20, P=-36, тобто алгоритм відпрацював правильно.

ЛЕКЦІЯ 4. БАЗОВІ ВІДОМОСТІ ПРО МОВУ PASCAL

4.1 Структура процедури

Результатом програмування алгоритму є процедура, що має наступну структуру:

```
procedure   имя_процедуры(список_параметров);   | заголовок процедури
РОЗДІЛ ОГОЛОШЕНЬ
begin          початок тіла процедури
          ТІЛО ПРОЦЕДУРИ
end;           кінець тіла процедури
```

4.2 РОЗДІЛ ОГОЛОШЕНЬ

Всі об'єкти, які використовувано в процедурі (константи, змінні, масиви, вкладені функції і процедури .) мають бути описані в розділі оголошень **до** їх першого використання. Жирним виділено службове слово, яке використовується в мові для опису конкретного об'єкту. Порядок оголошення об'єктів наступний:

№	об'єкт	приклад	коментар
1.	мітки	***label*** *MET;*	
2.	константи	***const*** *DLIN=30;* *e=2.718;* *MAX1=100;* *MAX2=20;* *str='##';*	числові константи строкова константа (у апострофах) константи не можуть бути змінені у програмі
3.	типи	***type*** *matr=array[**1..MAX1,1..MAX2**]* ***of real**;*	типа matr описує двовимірний масив (матрицю) з дійсних чисел
4.	змінні цілі речові логічні символьні строкові масивові	***var*** *i,j:* ***integer**;* *a,b:* ***real**;* *p,q:* ***boolean;*** *simvol**: char;*** *stroka:* ***string[DLIN];*** *name:* ***string[15];*** *TAB:array[1..MAX1,1..MAX2]* *of real;* *T1,T2: matr;*	

	файлові (текстових файлів)	*InFile, OutFile:* ***TextFile;***	
5.	процедури і функції		

4.3 Оператор привласнення (:=)

Використовується для привласнення змінній значення якого-небудь вираження і записується так:

имя_змінної:=вираження;

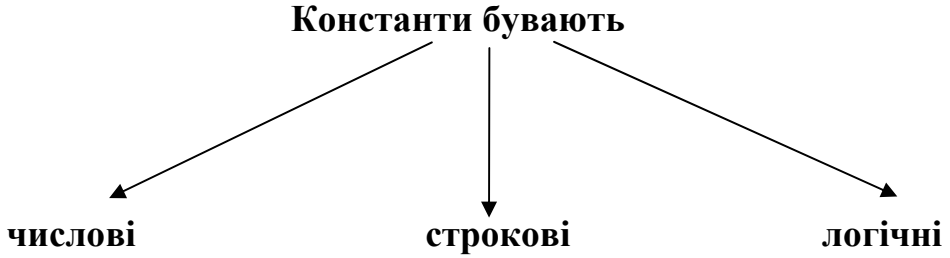
4.4 Імена

Ім'я будь-якого об'єкту в програмі складається з латинських букв, цифр і символу підкреслення і починається з букви. Російські і грецькі букви, пропуски і індекси в іменах не допускаються.

Правильні імена	Неправильні імена
delta_x S dlina x1 x_1 Edit_a	Dx сума xi 1x x 1 Edit a

Заголовні і рядкові букви для Паськаля не відрізнимі.

4.5 Константи



-10 9.5 1E10 означає (1010) 4e3 (4103·)·	строкова константа - будь-яка послідовність символів в апострофах, наприклад: ’London’ ’ДОННТУ’	0 (False) 1 (True)

4.6 Стандартні арифметичні функції

у математиці	на Паськале
sin x	sin(x)
cos x	cos(x)
arctg x	arctan(x)
ln x	ln(x)
ex	exp(x)
|x|	abs(x)
x2	sqr(x)
$\sqrt{x}$	sqrt(x)

Аргумент функції завжди указується в дужках; для тригонометричних функцій – в радіанах.

Для перетворення дійсного числа **x** в ціле користуватимемося функціями: **trunc(x)** і **round(x)**. Фукция **trunc(x)** відкидає дробову частку числа **x**, а фукция **round(x)** округлює **x** до найближчого цілого по правилах арифметики.

Наприклад, хай змінні a, i, j оголошені так:

```
var a: real;
    i,j: integer;
```

і хай a=-7.7.

Тоді в результаті обчислень по формулах

```
i:=trunc(a);
j:=round(a);
```

змінна i набуде значення -7, а змінна j стане рівною -8.

4.7 Арифметичні вирази

Записуються в рядок; черговість операцій – як в математиці. Якщо пріоритет операцій однаковий, то операції виконуються зліва направо.

операція	у Паськалю	коментар
складання	a+b	
віднімання	a-b	
множення	a*b	
ділення		
речове	a/b	
без остачі	a div b	дорівнює цілій частці від ділення a на b. Наприклад, 17

		div 5 дорівнює 3.
залишок по модулю	a mod b	дорівнює залишку від ділення a на b. Наприклад, 17 mod 5 дорівнює 2.

піднесення до ступеня: xy
 	
t= xy
 	
t= xy
	t:=exp(y*ln(x));
	
	Uses Math;
	
	t:=POWER(x,y);
		для цієї операції немає спеціального символу; користуємося одним з 2-х способів:
		1спосіб: представляємо $x^y = e^{\ln(x^y)}$
		2спосіб: у пропозиції Uses (на початку тексту модуля) додаємо бібліотеку Math до списку підключених до модуля биб-к. Після цього можна користуватися функцією POWER з цієї биб-ки, яка обчислює xy.

4.8 Логічні вирази

Включають **логічні константи** (1 і 0), **логічні змінні** і **логічні операції**. Константи 1 і 0 інтерпретуються відповідно як ІСТИНА (TRUE) і ХИБНІСТЬ (FALSE).

Логічною називається змінна, яка може приймати одне з двох значень: 1 ТА ВКЛ ІСТИНА Т і так далі
0 НЕМАЄ ВИКЛ БРЕХНЯ F

Над логічними змінними і виразами виконуються логічні операції І (AND), АБО (OR), НЕ (NOT). Результат цих операцій задається за допомогою **таблиць істинності:**

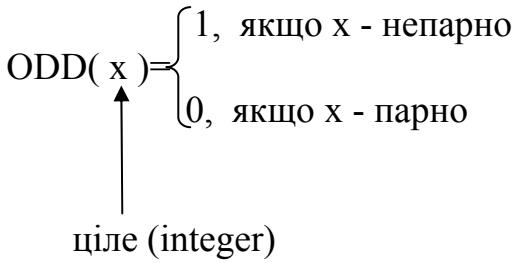
a a	b b	a AND b b
0	0	0
0	1	0
1	0	0
1	1	1

a a	b b	a OR b b
0	0	0
0	1	1
1	0	1
1	1	1

a a	NOT a a
0	1
1	0

Результатом перевірки будь-якої умови є Істина або Брехня, тому умова є логічним вираженням.

Існує стандартна **логічна функція**, що повертає ІСТИНУ, якщо її аргумент є непарне число, і ХИБНІСТЬ – інакше:



Старшинство операторів (арифметичних, логічних, стосунків)

Пріоритет	**Оператори**
0	дужки
1	одномісний мінус (наприклад -5); NOT
2	* / mod div AND
3	+ - OR
4 у Паськале **у математиці**	< <= > >= = <> < ≤ > ≥ = ≠

Приклад. Обчислити значення логічного вираження:

(a*2>b) OR NOT(c=6) AND (d-1<=e) при a=2, b=4, c=7, d=4, e=3

(2*2>4) OR NOT(7=6) AND (4-1<=3) *спочатку - дужки*

(F) OR NOT(F) AND (T) *потім NOT*

(F) OR (T) AND (T) *потім AND*

(F) OR (T) *потім OR*

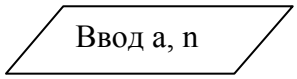
T

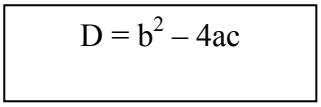
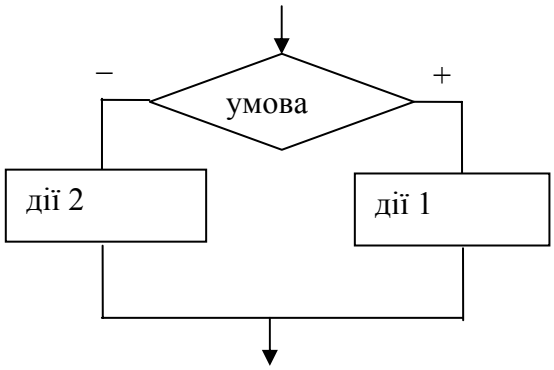
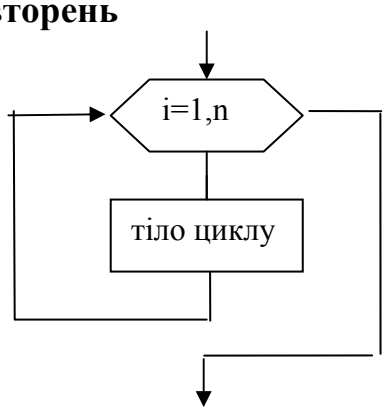
Відповідь: ІСТИНА (T).

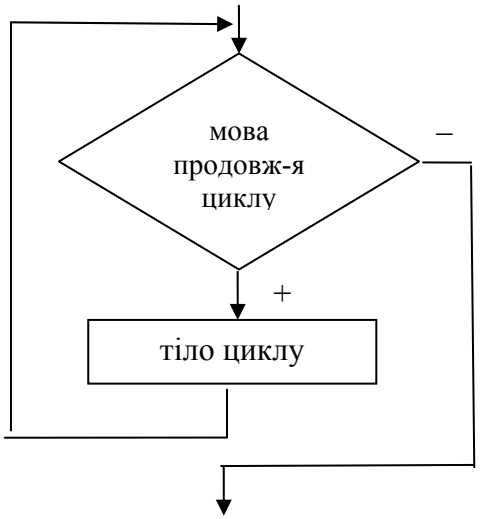
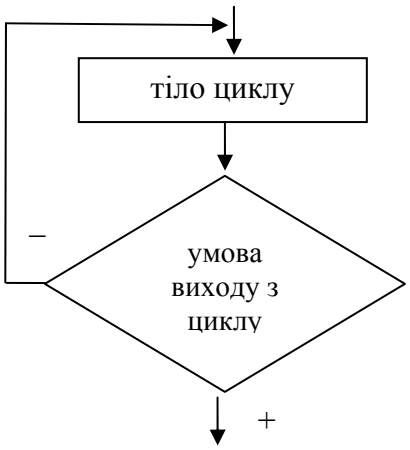
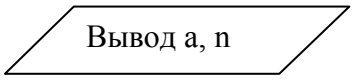
4.9 Відповідність між блоками в блок-схемі і операторами мови Pascal в програмі

Пояснення до операторів буде дано пізніше.

	Фрагмент блок-схеми	Оператор Pascal

1.	**Введення даних**
	(a – речове, тобто типа real
	n – ціле, тобто типа integer)
	 
		Введення здійснюється за допомогою оператора привласнення:
		
		a:=StrToFloat(Edit_a.Text);
		n:=StrToInt(Rdit_n.Text);

2	**Блок обчислень** 	**Оператор привласнення** D:= b*b – 4*a*c;
3	**Розгалуження** 	**Умовний оператор** **if** (умова) **then** оператор1 **else** оператор2; Тут оператор1 відповідає «діям 1», а оператор2 – «діям 2»
4	**Цикл з відомим числом повторень** 	**Оператор циклу for** **for** i:=1 to n do оператор; Тут оператор відповідає «тілу циклу».

5	**Цикл з передумовою (ПОКИ)**  	**Оператор циклу while** **while** (умова) do оператор;
6	**Цикл з умовою поста (ДО)**  	**Оператор циклу repeat – until** **repeat** оператори тіла циклу через «;» **until** (умова виходу);
7	**Виведення даних** (a – дійсне, тобто типа real n – ціле, тобто типа integer)  	1)вивід у вікно редагування (Edit): Edit_a.Text:=FloatToStr (a); Edit_n.Text:=IntToStr(n); 2)вивід в мітку (Label): Label5.Caption:=’a=’+ FloatToStr (a) + ’ n=’ + IntToStr(n); 3)виведення повідомлення в діалогове вікно: ShowMessage(‘Помилка: введіть a(0’);

Зауваження:

1. Оператори розділяються крапкою з комою. Після **begin** і перед end крапку з комою можна не ставити.

2. Після **then, else** і do по правилах Паськаля може стояти тільки один оператор. Якщо необхідно виконати групу операторів, то цю групу беруть в операторних дужок «begin . end», утворюючи одного складеного оператора. Таким чином, складений оператор має вигляд:

```
begin
    оператор перший;
    оператор другий;
    . . . . . ;
    оператор n–ый
end;
```

ЛЕКЦІЯ 5. ВВЕДЕННЯ В DELPHI. ПЕРШИЙ ПРОЕКТ

5.1 Введення

Програма, написанныая для роботи під Windows, називається **додатком.**

Borland Delphi – це система швидкої розробки додатків під Windows. У основі систем швидкої розробки (RAD-систем, Rapid Application Development — середа швидкої розробки додатків) лежить технологія візуального проектування і подієвого програмування. Її суть полягає в тому, що середа розробки бере на себе велику частку генерації коди програми, залишаючи програмістові роботу по конструюванню діалогових вікон і функцій обробки подій. Швидкість програмування при використанні RAD-систем різко зростає.

Delphi – середа швидкої розробки на мові **Object Pascal**. У основі ідеології Delphi лежить технологія візуального проектування і методологія об'єктно-орієнтованого подієвого програмування.

Візуальне проектування означає, що користувач будує своє застосування за допомогою миші і на екрані спостерігає результат своєї роботи.

Об'єктно-орієнтована програма – це система, визначувана сукупністю об'єктів і способів їх взаємодії.

Об'єкт – це сукупність властивостей, методів і подій, на які він може реагувати.

5.2 Основні поняття об'єктно-орієнтованого програмування

Основними поняттями Delphi є: класи, спадкоємство і інкапсуляція.

Класи. Кожен об'єкт відноситься до деякого класу об'єктів. Класи бувають **стандартні** і призначені для користувача. До стандартних відносяться: кнопки, мітки, вікна . . Наприклад, конкретні кнопки Button1, Button2, Button3 є екземплярами класу кнопок, який називається TButton.

Спадкоємство. Класи і їх об'єкти можуть утворювати ієрархічну структуру у вигляді дерева. Верхній рівень дерева називається **батьківським (предком)** по відношенню до нижнему, який називається дочерним (нащадком). За умовчанням **Форма** є батьком всіх встановлених на ній компонентів. У об'єктно-орієнтованому

програмуванні нащадки успадковують властивості предків і, крім того, можуть володіти своїми особливими властивостями. Тому у нащадка може бути більше можливостей, чим у його предка.

Інкапсуляція – це вбудовування в об'єкт його властивостей, методів і подій.

5.3 Програмування в Delphi

Для будь-якого об'єкту в Delphi визначені події, на які він може реагувати. Реакція на подію визначається процедурою, яка називається **оброблювачем події**. Програмування в Delphi включає 3 етапи:

1) компоновка **Форми** – це майбутнє вікно додатка;
2) налаштування властивостей об'єктів **Форми;**
3) написання оброблювачів подій над об'єктами **Форми**.

5.4 Склад проекту

Результатом роботи Delphi є **проект**, що складається з декількох файлів:

1) *.dpr – файл проекту (наприклад, Project1.dpr); текстовий файл, зберігає інформацію про форми і модулі;
2) *.pas – файл модуля (наприклад, Unit1.pas); кожній **Формі** відповідає файл модуля з текстом програми на Паськалe;
3) *.dfm – двійковий (у Delphi 3) або текстовий (у пізніших версіях) файл з інформацією про **Форму**; ім'я файлу збігається з ім'ям модуля (наприклад, Unit1.dfm);
4) *.res – двійковий файл ресурсів; зберігає інформацію про піктограми, зображення курсора і інші ресурси, використовувані в проекті. Створюється автоматично при створенні проекту, і його ім'я збігається з ім'ям проекту (наприклад, Project.res);
5) *.exe – виконуваний файл створюваного застосування; ім'я збігається з ім'ям проекту (наприклад, Project1.exe);

і ін.

Для запуску проекту необхідні перші 4 файли. Останні створюються автоматично системою Delphi.

Після запуску Delphi автоматично створює текст проекту на мові Object Pascal (файл Project1.dpr) і заготівку для модуля (файл Unit1.pas), відповідного формі Form1.

5.5 Спільний вигляд і налаштування вікна Delphi

Запускається Delphi звичайним способом:

Пуск → Програми → Borland Delphi → Delphi 3.

Вид екрану показано на мал. 5.1. Після запуску на екрані з'являються 4 вікна:

- головне вікно Delphi;
- вікно стартової **Форми** (Form1);
- вікно редактора властивостей об'єктів (Object Inspector (Інспектор об'єктів));
- вікно редактора коди головного модуля (Unitl.pas).

Вікно редактора коди майже повністю закрите вікном стартової форми. Для перемикання між вікном редактора коди і **Формою** – клавіша F12.

У головному вікні знаходиться меню команд, панелі інструментів і палітра компонентів.

Вікно стартової форми – це заготівка головного вікна разрабатываемого застосування.

Вікно Object Inspector (мал. 5.2) включає дві вкладки: «**Properties» (**Властивості) «**Events**» (Події). Вкладка «Властивості» призначена для редагування значень властивостей об'єктів. У термінології візуального проектування об'єкти – це діалогові вікна і елементи управління (поля введення і виводу, командні кнопки, перемикачі і ін.). Властивості об'єкту – це характеристики, що визначають вигляд, положення і поведінку об'єкту. Наприклад, властивості Width і Height задають розмір (ширину і висоту) форми, властивість caption – текст заголовка. Вкладка «Події» призначена для призначення подіям, на які об'єкт може реагувати, відповідних ним процедур-оброблювачів подій.

5.6 Створення проекту. Приклад програмування лінійного алгоритму

Візуальне проектування полягає в компоновці Форми і налаштуванні властивостей її об'єктів**.**

Компоновка Форми

1) вибрати клацанням миші потрібну палітру компонентів (наприклад, Standard) і клацнути в ній на потрібному компоненті (наприклад, на кнопці, або мітці, і тому подібне);

2) клацнути мишею на Формі в тому місці, куди треба вставити вибраний компонент.

Налаштування властивостей об'єктів

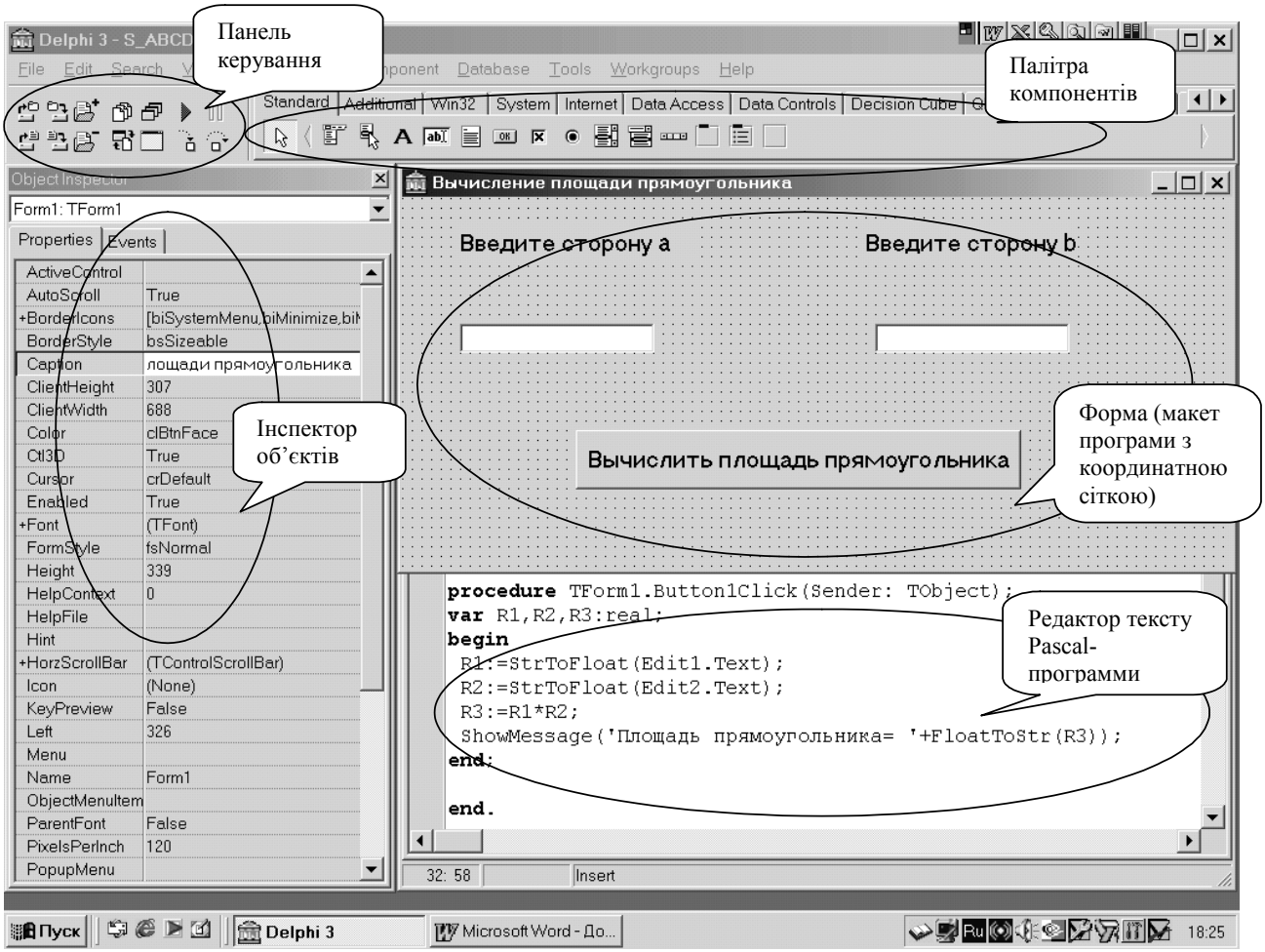
1) виділити об'єкт на Формі. Для цього:
- або клацнути мишею по цьому об'єкту на Формі
- або вибрати цей об'єкт із списку об'єктів Інспектора об'єктів;

2) перейти в Інспектор об'єктів і на вкладці Властивості («Properties») задати значення потрібних властивостей даного об'єкту.

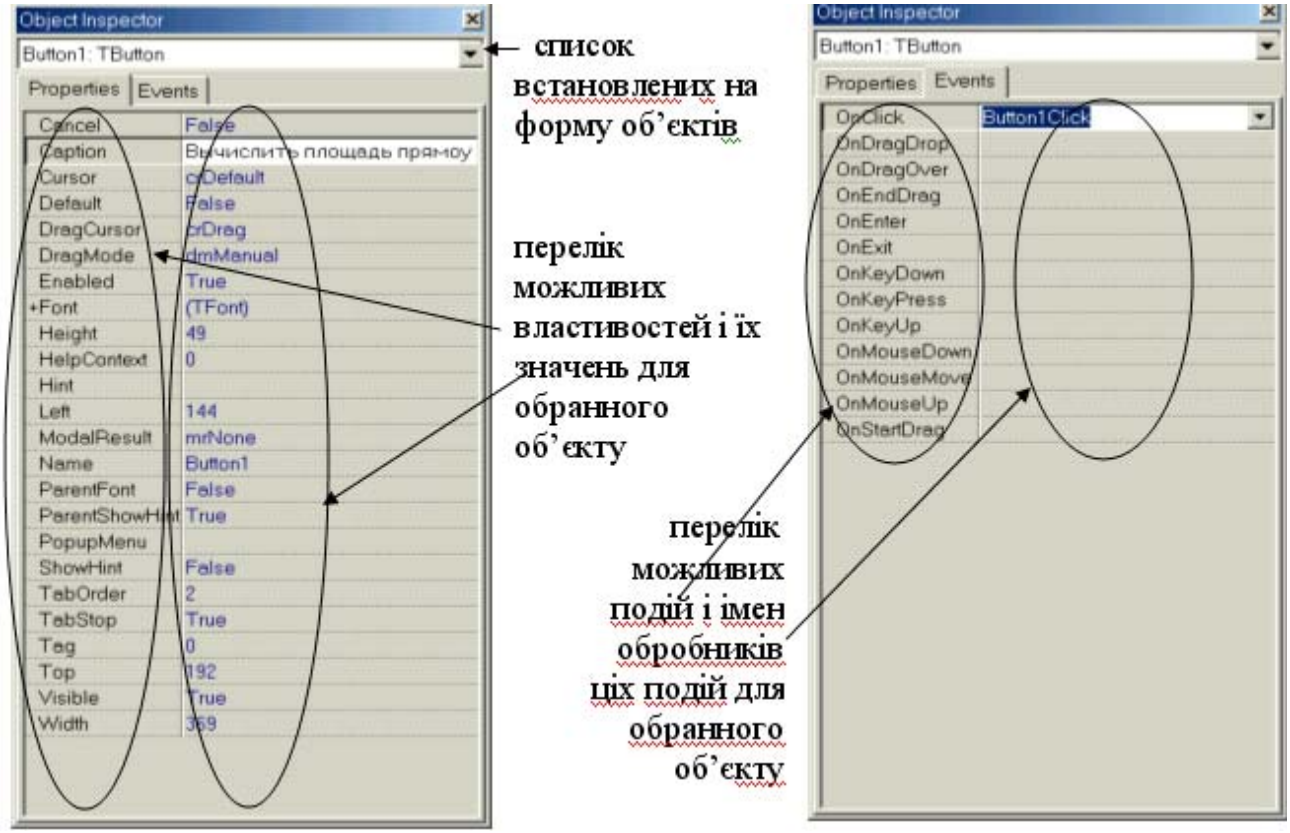
Перелік всіх властивостей і подій для виділеного об'єкту можна отримати через контекстну довідку F1. Приведемо найбільш часто використовувані властивості різних компонентів (див. таб. 5.1).

Після того, як **Форма** налагоджена, залишається написати оброблювачі подій.

Опишемо послідовність дій при створенні нового проекту на прикладі. *Цій послідовності рекомендується дотримуватися при виконанні лабораторних робіт.*



Малюнок 1. Вигляд екрану у середовищі Delphi



Малюнок 6.2 – Ліворуч – сторінка «Властивості» (Properties), праворуч - сторінка «Події» (Events)

Таблиця 5.1. Часто використовувані властивості компонентів

Властивість	**Опис**
Name	Ім'я **Форми** або компоненту **Форми**
Align	Вибір способу вирівнювання компоненту щодо вікна **Форми**
Alignment	Вибір способу вирівнювання тексту усередині об'єкту
Border . . .	Налаштування кордонів об'єкту
BorderStyle	Вид кордону Форми або деяких об'єктів. Кордон може бути звичайним (bsSizeable), тонким (bsSingle) або бути відсутнім (bsNone). Якщо біля вікна звичайна границя, то під час роботи програми користувач може при помощи миші змінити розмір вікна. Змінити розмір вікна з тонким кордоном не можна. Якщо кордон відсутній, то на екран під час роботи програми буде виведено вікно без заголовка. Положення і розмір такого вікна під час роботи програми змінити не можна
Caption	Текст заголовка Форми або текст напису на компоненті **Форми**
Color	Колір фону. Колір можна задати, вказавши назву кольору (наприклад, clAqua – блакитний) або привязку до поточної колірної схеми операційної системи (наприклад, clButoonFace – колір фону кнопки). У другому випадку колір визначається поточною колірною схемою, вибраним компонентом прив'язки (кнопка) і міняється при зміні колірної схеми операційної системи
Enable	Доступність об'єкту (=True/False) – показує, чи повинен об'єкт реагувати на мишу, клавіатуру, час
+Font	Шрифт. Подвійне клацання миші на знаку «+» розкриває детальні характеристики шрифту (назва, розмір, колір). За умовчанням зміна властивості Font форми приводить до автоматичної зміни властивості Font компоненту, встановленого на **Формі**. Тобто компоненти успадковують властивість Font від **Форми** (є можливість заборонити спадкоємство – властивість ParentFont встановити в False)
Heigh	Висота об'єкту в пікселях
Hint	Текст спливаючої підказки
Items	Елементи об'єкту, якщо вони є
Left	Для Форми – це відстань від лівого кордону Форми до лівого кордону екрану;для об'єкту – це відстань в пікселях від лівого кордону об'єкту до лівого кордону батька
Parent . . .	Узяти або не брати властивість з батьківського компоненту
ShowHint	Показувати або ні підказку Hint
Top	Для **Форми** – це відстань в пікселях від верхнього кордону Форми до верхнього кордону екрану;для об'єкту – це відстань в пікселях від верхнього кордону об'єкту до верхнього кордону батька
Visible	Видимий або невидимий об'єкт (=True/False)
Width	Ширіна об'єкту в пікселях

Завдання. Скласти блок-схему алгоритму і написати програму обчислення площі і периметра прямокутника по двох його сторонам.

Початкові дані: довжини сторін a і b.
Вихідні дані: площа S, периметр P.

1. **Оскільки проектом є набір файлів, то рекомендується для кожного проекту створювати окрему теку.**
   Засобами WINDOWS створюємо окрему папку для нового проекту Delphi, наприклад, папку ***C:\Students\ Тпе-09в\ Іванов\lab1.***

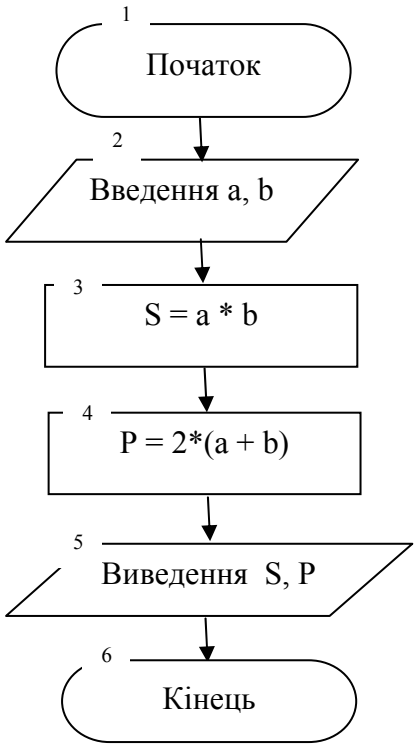
2. **Входимо в Delphi**:

   Пуск → Програми Borland Delphi 3 Delphi 3 (або кнопка ).
   При цьому відкриється порожня **Форма**.

3. **Відразу після відкриття порожньої Форми зберігаємо новий проект в папці Lab1 командою**:

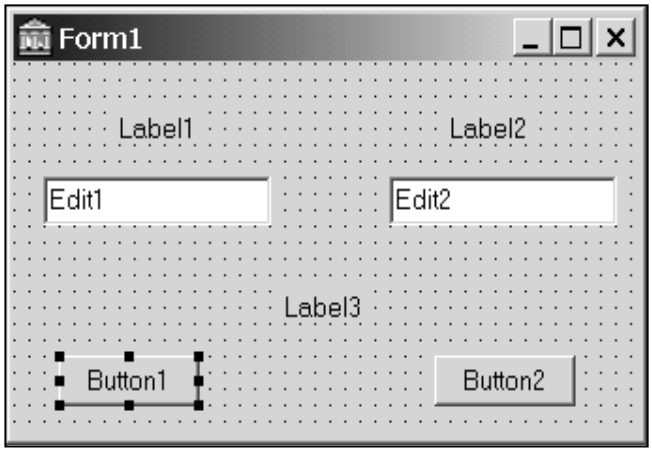
   File → Save All ( ***) → указуємо шлях до теки Lab1→ →*** *при першому збереженні Delphi запитає ім'я модуля (за умовчанням, unit1.pas), що зберігається, а потім – ім'я проекту (project1.dpr). Змінимо ім'я модуля на* ***U_lab1.pas,*** *а ім'я проекту – на* ***P_lab1.dpr.***

   Зауваження. Якщо необхідно створити новий проект, то слід скористатися командою*:* ***File → New Application.***



Малюнок 5.3 – Блок-схема лінійного алгоритму

4. <u>Компонуємо наступну форму</u>.



Для цього в палітрі компонентів знаходимо сторінку **Standard** і переносимо з неї на форму наступні об'єкти (компоненти):

A - Label - три мітки,

ab - Edit – два рядки редагування,

OK - Button – дві кнопки.

Для переносу спочатку клацаємо на потрібному об'єкті сторінки **Standard** (наприклад, на мітці A), а потім клацаємо в потрібному місці форми (мітка вставляється).

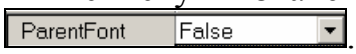
Зберігаємо проект командою: **File → Save All** ( ).

5. <u>Задаємо властивості об'єктів Форми</u> за допомогою Інспектора Об'єктів. Спочатку виділяємо потрібний об'єкт, а потім налагоджуємо його властивості. Виділити об'єкт можна двома способами: клацнувши один раз у формі на цьому об'єкті або вибравши потрібний об'єкт зі списку, що розкривається, всіх об'єктів форми.

5.1 <u>Властивості самої **Форми**</u>

Змінимо текст у заголовку форми на **Обчислення площі і периметру прямокутника**, колір її фону на **голубий** і шрифт написів на всіх об'єктах форми, крім кнопок, на **напівжирний, курсив**.

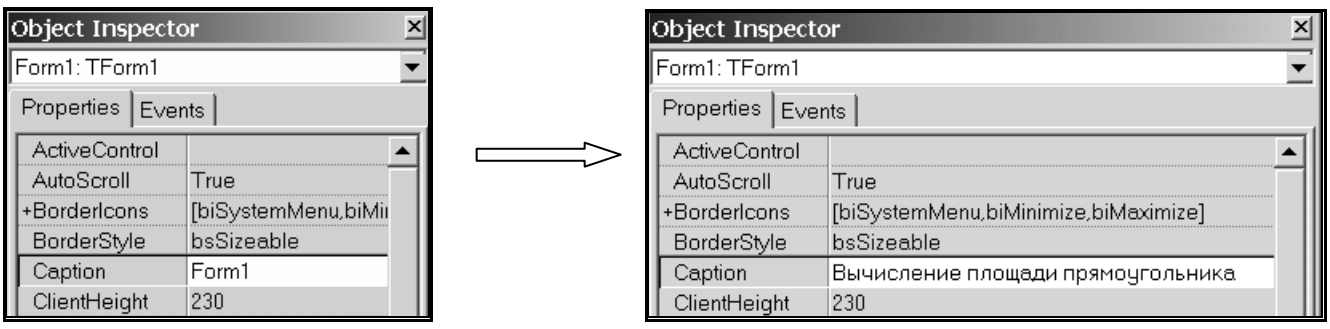
Щоб кнопки не успадковували властивостей шрифту свого батька (форми), задамо для них властивість **ParentFont** рівною **False.** Для цього виділимо клацанням кнопку **Button1,** виберемо в **Інспекторі об'єктів** властивість **ParentFont** і в списку значень, що розкривається, праворуч **виберемо False:**



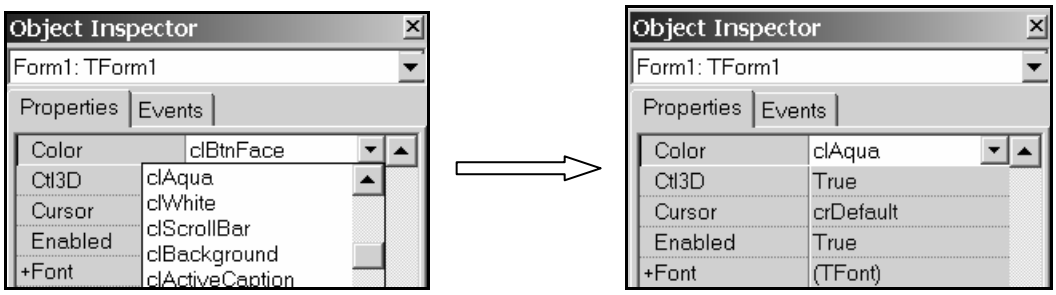
. Аналогічне налагодження зробимо для кнопки **Button2.**

Після цього **клацнемо на Формі** і задамо її властивості.

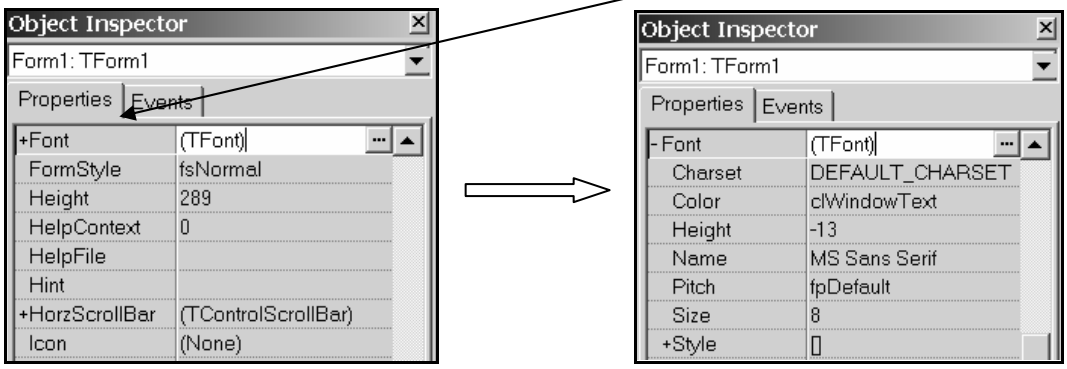
5.1.а) Caption – напис у заголовку форми. На вкладці **Properties** замінимо значення, що умовчується, **Form**1 властивості **Caption** на **Обчислення площі і периметру прямокутника**, увівши його з клавіатури.



5.1.b) Color – колір фону. Замінимо значення **clBtnFace**, що умовчується, тобто колір, що збігається з фоном кнопки, на голубий - **clAqua.**

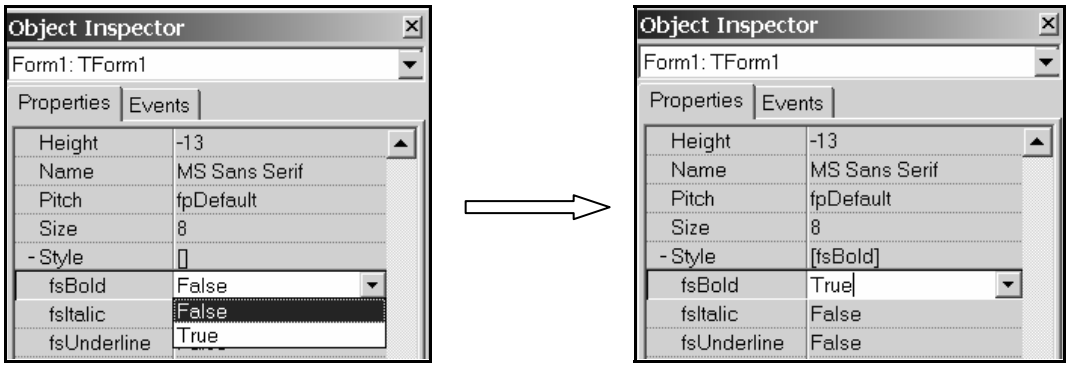


5.1.c) Font – шрифт. Подвійне клацання на властивості **+Font** відкриває таблицю властивостей шрифту:



.

Аналогічно, подвійне клацання на властивості **+Style** відкриває таблицю стилів шрифту, у якій для властивостей **fsBold** (напівжирний) і **fsItalic** (курсив) вибираємо зі списку, що розкривається, значення True:



Надалі присвоєння нового значення властивості об'єкта будемо записувати за допомогою символу крапки:

об'єкт.властивість:=значення.

Наприклад, перші два призначення властивостей a) і b) будуть виглядати так:

Form1.Caption:='**Обчислення площі і периметру прямокутника'** і

Form1.Color:=**clAqua.**

При цьому текст напису набирається на клавіатурі без лапок.

5.2 Властивості міток

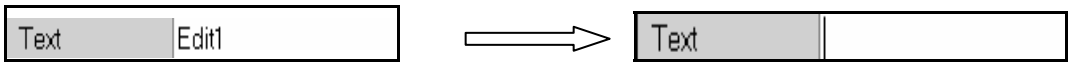
5.2.a) Label1.Caption := **'Введіть сторону a '**

5.2.b) Label2.Caption := **'Введіть сторону b'**

5.2.c) Label3.Caption := **'Площа та периметр = '**

5.3 Властивості рядків редагування

5.3.a) Edit1.Text :=' ' (**порожньо**, тобто стираємо слово Edit1 у властивості Text):



5.3.b) Edit2.Text :=**' '** (**порожньо**)
5.3.c) Edit1.Name := **Edit_a**
5.3.d) Edit2.Name := **Edit_b**

5.4 Властивості кнопок

5.4.a) Button1.Caption :=**'Обчислення**
5.4.b) Button2.Caption := **'Вихід'**
5.4.c) Button1.Name := **Button_calc**
5.4.d) Button2.Name := **Button_close**

При необхідності змінюємо розміри об'єкта, потягнувши мишкою за його границю. Після виконаних налагоджень **Форма** приймає наступний вигляд:

Зберігаємо проект командою: **File → Save All** ( ).

Якщо Форма містить декілька однакових компонентів, то при переносі компонента на форму його ім'я **(властивість Name)** ***варто зробити осмисленим – для зручності використання в програмі.*** Так, рядки редагування **Edit1** і **Edit2** ми перейменували відповідно в **Edit_a** і **Edit_b**.

6. Задаємо процедури – оброблювачі подій.

6.1 Процедура обробки клацання (клацання – це подія OnClick) на кнопці «Вихід» (ім'я кнопки Button_close)

Двічі клацаємо на кнопці **«Вихід»** і додаємо в Редакторі Коду до тексту процедури оператор закриття форми (**close**):

```
// Завершення роботи програми при натисканні кнопки "Вихід"
procedure TForm1.Button_closeClick(Sender: TObject);
begin
  close; // закрити форму
end;
```

6.2 Процедура обробки клацання на кнопці «Обчислення» (кнопка Button_calc)

Щоб написати процедуру обробки події OnClick на кнопці Button_calc потрібно потрапити у вікно Редактора Коду даної процедури. Для цього треба двічі клацнути на кнопці **«Обчислення»** безпосередньо у **Формі**. У вікні Редактора Коду з'явиться заготівля оброблювача події:

Змінюємо цю процедуру в такий спосіб:

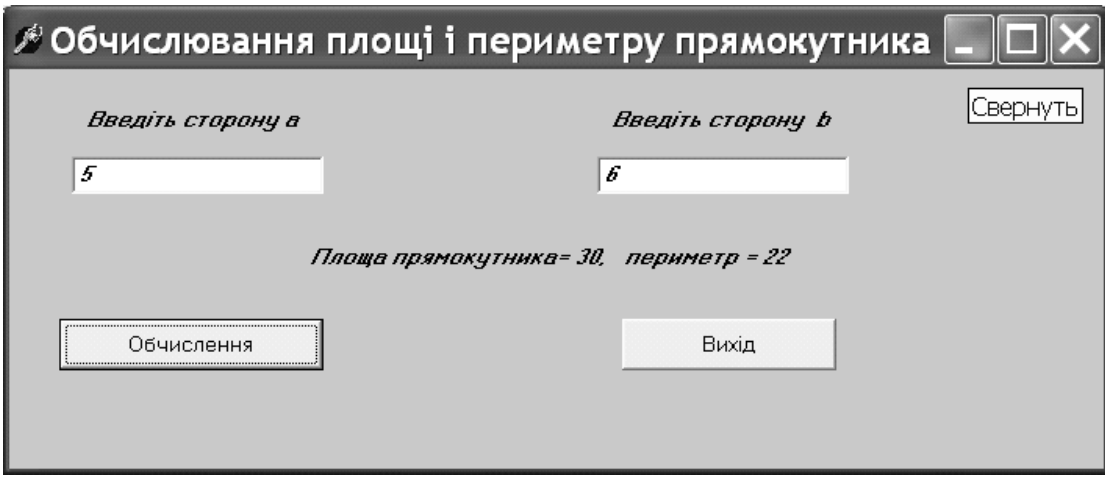
```
//клацання на кнопці Обчислення
procedure TForm1.Button_calcClick(Sender: TObject);
var a,b,S,P:real;  // оголошення сторон и площи
begin
 a:=StrToFloat(Edit1.Text);      // введення сторони a
 b:=StrToFloat(Edit2.Text);      // введення сторони b
 S:=a*b;                         // обчислювання площи
 P:=2*(a+b);                     // обчислювання периметру
 // виведення результата в метку
 Label3.Caption :=' Площа прямокутника= '+FloatToStr(S)
 +',   периметр = '+FloatToStr(P);;
 end;
```

Зберігаємо проект командою: **File → Save All (** ).

7. <u>Запускаємо проект на виконання</u> одним з 3-х способів:

- клацання на кнопці  (Run) у панелі керування;
- клавіша **F9**;
- команда **Run → Run.**

Результат роботи проекту показаний на малюнку 5.4.



Малюнок 5.4 – Результат роботи програми

ПОЯСНЕННЯ

Програма

В результаті натиснення кнопки **Обчислення** запускається процедура **Button_calcClick**, яка виконує розрахунок майдану і периметра прямокутника і виводить результат розрахунку в мітку **Label3**. Початкові дані вводяться з рядків

редагування **Edit_a** і **Edit_b**, точніше з властивості **Text** цих рядків. Властивість **Text** містить рядок символів, який під час роботи програми введе користувач. Для правильної роботи програми рядок повинен містити тільки цифри. Для перетворення рядка в число в програмі використовується функція StrToFloat. Функція StrToFloat перевіряє, чи допустимі символи рядка, переданого їй як параметр (Edit_a.Text – це вміст поля Edit_a), і, якщо всі символи вірні, повертає відповідне число (число може бути дробом). Це число записується в змінну a**.** Аналогічно вводиться змінна b**.**

Після того, як початкові дані будуть поміщені в змінні **a** і **b**, виконується розрахунок майдану (змінна **S**) і периметра (змінна **P**). Обчислені значення виводяться в мітку **Label3** шляхом надання значення властивості **Caption**. Для перетворення чисел в рядки використовується функція **FloatToStr**. Операція «+» виконує зчеплення рядків, тобто приписування одного рядка до іншої.

В результаті натиснення кнопки **Вихід** програма повинна завершити роботу. Щоб це сталося, треба закрити **Форму**. Робиться це за допомогою методу **close**.

Запуск на виконання

Команда **Run → Run** спочатку компілює програму, а потім виконує її. Компіляція – це процес перетворення початкової програми у виконувану. Процес компіляції складається з двох етапів. На першому етапі виконується перевірка тексту програми на відсутність помилок, на другому – генерується виконувана програма, тобто exe-файл. Компілятор генерує виконувану програму лише в тому випадку, якщо початковий текст не містить синтаксичних помилок.

Синтаксичні помилки (Errors)

Помилки виправляються послідовно. Спочатку усуваються найбільш очевидні помилки, наприклад, оголошуються не оголошені змінні. Після чергового виправлення тексту програми повторно виконується команда **Run → Run.** Компілятор не завжди може точно локалізувати помилку. Тому, аналізуючи помилковий фрагмент програми, потрібно звертати увагу не лише на той фрагмент коду, на який компілятор встановив курсор, але і на той, який знаходиться в попередньому рядку. У таблиці. 5.2 перераховані найбільш типові помилки і відповідні ним повідомлення компілятора.

Таблиця 5.2. Помилки компиляції

Помилка	**Повідомлення про помилку**	**Переклад повідомлення про помилку на українську мову і ймовірна причина помилки**	**Правильний варіант**
var a, S: real;	Undeclared Identifier: ‘b’	Неоголошений ідентифікатор ‘b’. Забули оголосити змінну b	var a, **b,** S: real;

a: = StrToFloat(Edit1.Text);	Undeclared Identifier: ‘Edit1’	Неоголошений ідентифікатор ‘Edit1’. На **Формі** відсутній компонент Edit1.	a: = StrToFloat(**Edit_a**.Text);
b: = StrToFloa(Edit_b.Text);	Undeclared Identifier: ‘StrToFloa’	Неоголошений ідентифікатор ‘StrToFloa’. Помилка в записі імені функції	b: = **StrToFloat**(Edit_b.Text);
b:=StrToFloat(Edit2.Text) S:=a*b;	Missing operator or semicolon	Пропущений оператор або крапка з комою	b:=StrToFloat(Edit2.Text)**;** S:=a*b;
S=a*b;	‘:=’ expexted, but ‘=’ found	Очікувалося ‘:=’, а виявлено ‘=’	S:=a*b;
	’’ is not a valid floating point value	’’ (порожньо) – невірне значення з плаваючою крапкою	в процесі обчислювань поле введення опинилося порожнім; ввести в поле введення дійсне число
’ площа= +FloatToStr(S)	Unterminated string	Незавершений рядок. При записі строкової константи, наприклад, повідомлення, не поставлена завершуюча лапка	’ площа= +FloatToStr(S)’
var n:integer; a:real; . . . . . . n:=a;	Incompatible types ...and ...	Несумісні типи. У оператора привласнення тип вираження не відповідає або не може бути приведений до типа змінної, одержуючої значення вираження	Уточнити спосіб перетворення речової величини a в цілу n, наприклад: n:=trunc(a);

Попередження (Warnings) і підказки (Hints)

При виявленні в програмі неточностей, які не є помилками, компілятор виводить підказки (Hints) і попередження (Warnings). У таблиці. 5.3 приведені попередження, що найчастіше виводяться компілятором.

Таблиця 5.3. Попередження компіляції

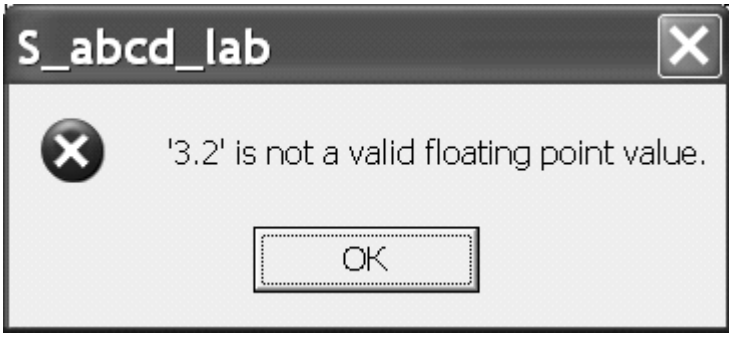
var n:integer;	Variable n is declared but never used in ...	Змінна n оголошена, але не використовується	Навіщо оголошувати невживану змінну?
	Variable n might not have been initialized	У програмі немає оператора, який привласнює змінною **n** початкове значення (ймовірно, використовується змінна, що не ініціалізувала)	Перевірити, чи не пропущена ініціалізація змінної n

Помилки часу виконання (run-time errors, exceptions)

Під час роботи застосування можуть виникати помилки, які називаються помилками часу виконання (run-time errors) або виключеннями (exceptions). В більшості випадків причинами виключень є невірні початкові дані. Наприклад, якщо під час роботи програми обчислення майдану прямокутника в полі а **ввести** 3.20, тобто для відділення дробової частки числа від цілої використовувати крапку, то в результаті натиснення кнопки Обчислення **на** екрані з'явиться вікно з повідомленням про помилку (мал. 5.5).

Малюнок 5.5 – Приклад помилки часу виконання (запуск з Delphi)

Причина помилки – в тому, що роздільником цілої і дробової частки числа має бути кома. Який з цих двох символів є допустимим, залежить від налаштування Windows. Якщо програма запущена з Windows, а не у середовищі Delphi, то ця ж помилка виглядатиме так (див. мал. 5.6):



Малюнок 5.6 – Приклад помилки часу виконання (запуск з Windows)

ЛЕКЦІЯ 6. ЕТАПИ РОЗРОБКИ ПРОГРАМИ. ПРОГРАМУВАННЯ РОЗГАЛУЖЕНИХ АЛГОРИТМІВ

6.1 Етапи розробки програми

Програмування – це процес створення (розробки) програми, який може бути представлений послідовністю наступних кроків:

1) визначення вимог до програми;
2) розробка алгоритму;
3) написання програми (кодування на алгоритмічній мові);
4) відладка;
5) тестування.

6.1.1 Визначення вимог до програми

На цьому етапі детально описується початкова інформація і формулюються вимоги до результату. Крім того, описується поведінка програми в особливих випадках.

Якщо програма працюватиме в Windows, то потрібно розробити діалогові вікна, що забезпечують взаємодію користувача і програми.

6.1.2 Розробка алгоритму

На етапі розробки алгоритму необхідно визначити послідовність дій, які треба виконати для отримання результату. Якщо завдання можна вирішити декількома способами, то необхідно вибрати деякий критерій, наприклад швидкість роботи алгоритму, і на його основі вибрати найбільш відповідне рішення. Результатом етапу розробки алгоритму є докладний словесний опис алгоритму або його блок-схема.

6.1.3 Кодування

Після того, як визначені вимоги до програми і складений алгоритм вирішення, алгоритм записується на вибраній мові програмування. В результаті виходить початкова програма.

6.1.4 Відладка

Відладка – процес пошуку і усунення помилок. Помилки в програмі розділяють на дві групи: синтаксичні (помилки в тексті) і логічні. Синтаксичні помилки – найбільш помилки, що легко усуваються. Логічні помилки виявити важче. Етап відладки можна вважати за закінчений, якщо програма правильно працює на одном-двух наборах вхідних даних.

6.1.5 Тестування

Етап тестування особливо важливий, якщо ви передбачаєте, що вашою програмою користуватимуться інші. На цьому етапі слід перевірити, як поводиться програма на як можна більшій кількості вхідних наборів даних, у тому числі і на свідомо невірних.

6.2 Умовний оператор if

Вибір в точці розгалуження алгоритму чергового кроку програми може бути реалізований за допомогою операторів **if** і case. Умовний оператор if дозволяє вибрати один з двох можливих варіантів, оператор множинного вибору case – один з декількох. Ми обмежимося оператором **if.**

6.3 Приклад програмування розгалуженого алгоритму

Завдання. Скласти блок-схему алгоритму і програму розрахунку на Object Pascal значень:

$$X = 10 \cdot (5{,}28 - \sqrt{A})$$

$$Y = \begin{cases} \sqrt{B^2 + 2B - 3}, & \text{если } A^2 < 25; \\ \ln|X|, & \text{если } A^2 \geq 25. \end{cases}$$

Вхідні дані: A, B. **Вихідні дані**: X, Y.

Проаналізуємо вираження для X і Y.

Область визначення X – будь-які ненегативні значення A. У противному випадку на екран видається повідомлення про невизначеність X.

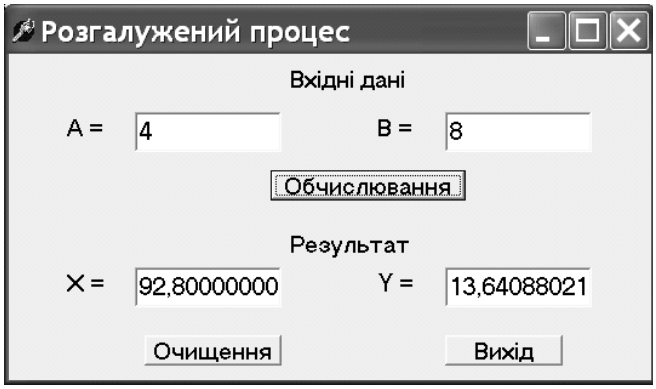
При обчисленні Y по верхній формулі припустимими є ті значення B, для яких $B^2 + 2B - 3 >= 0$. При обчисленні Y по нижній формулі припустимі значення X ≠ 0 за умови, що X визначений. У противному випадку на екран видається повідомлення про невизначеність Y. Оскільки дійсні числа представляються в пам'яті комп'ютера не точно, те будемо вважати, що X дорівнює нулеві, якщо його модуль менше 10^{-10}.

Введемо логічну змінну PX – ознаку визначеності X. На початку алгоритму присвоїмо PX значення TRUE. У випадку, коли X не визначений, ознаці PX присвоїмо значення FALSE. При обчисленні Y по нижній формулі спочатку перевіримо, чи визначений X, і якщо це так, то потім перевіримо, чи не дорівнює він нулеві.

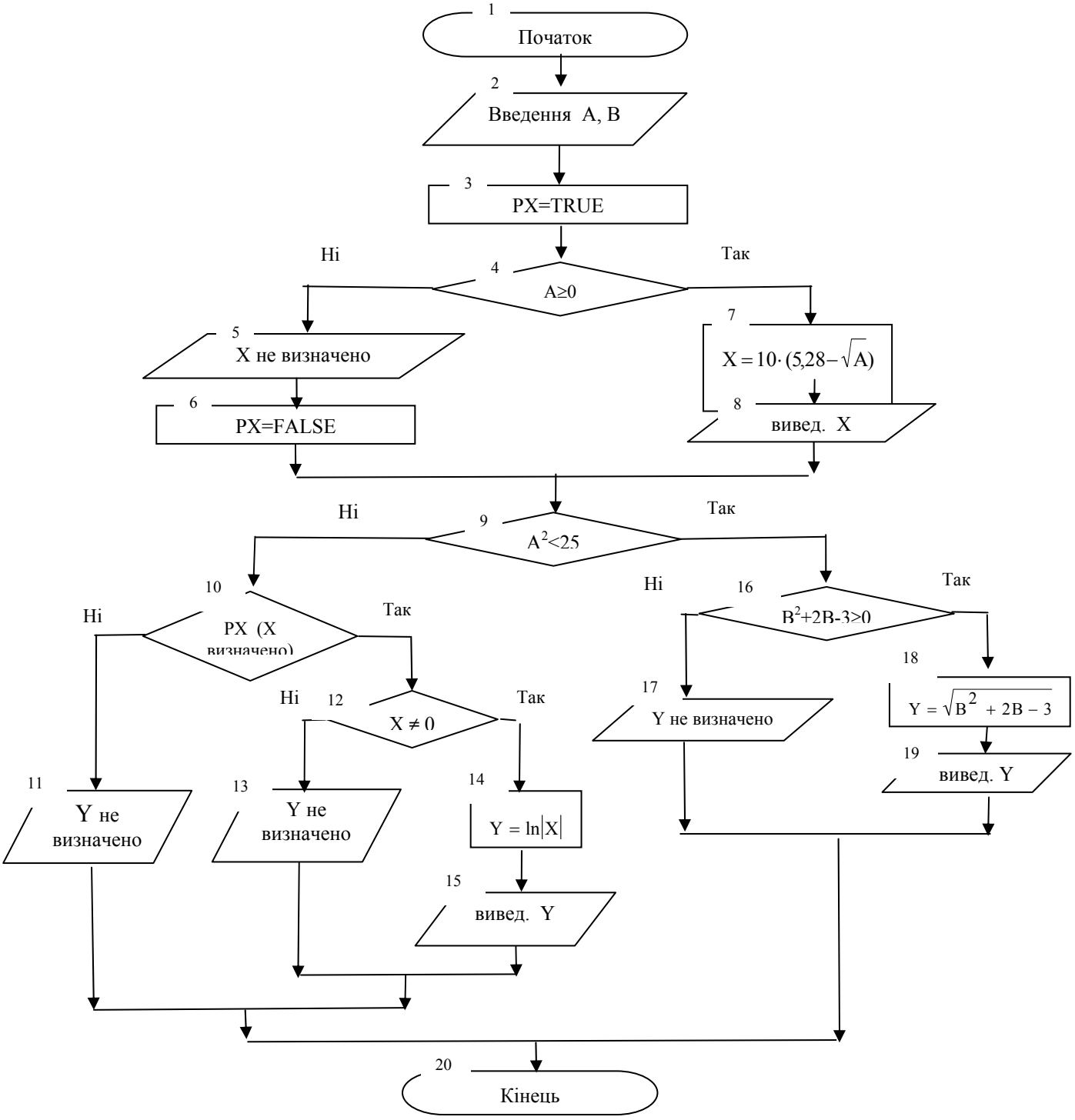
Блок-схема алгоритму приведена на малюнку 6.5.

Скомпонуємо наступну форму і налагодимо її властивості.

Вихідна **Форма** після перейменування об'єктів



Форма після налагодження властивостей об'єктів і виконання розрахунків



Малюнок 6.1. – Блок-схема розгалуженого алгоритму

1. Процедура обробки клацання на кнопці «Вихід» (кн. Button_close)

```
procedure TForm1.Button_closeClick(Sender: TObject);
begin
   close
end;
```

2. Процедура обробки клацання на кнопці «Очищення» (кн. Button_clear)

```
// Очищення всіх текстових полів при натисканні кнопки "Очищення"
procedure TForm1.Button_clearClick(Sender: TObject);
begin
  Edit_a.Clear;
  Edit_b.Clear;
  Edit_x.Clear;
  Edit_y.Clear
end;
```

3. Процедура обробки клацання на кнопці «Обчислення» (кн. Button_calc)

```
procedure TForm1.Button_calcClick(Sender: TObject);
var A,B,X,Y:real;   //оголошення
    PX:boolean;     //ознака визначеності X
begin
    A:=StrToFloat(Edit_a.Text);    // введення A
    B:=StrToFloat(Edit_b.Text);    // введення B
    PX:=True;       //X визначений
    if (A>=0) then
    begin
        X:=10*(5.28-sqrt(A));
        Edit_x.Text:=FloatToStr(X)
    end
    else begin
        Edit_x.Text:='x не визначений';
        PX:=False
    end;
    if (A*A<25) then
       if(B*B+2*B-3>=0)then
          begin
             Y:=sqrt(B*B+2*B+3);
             Edit_y.Text:=FloatToStr(Y)
          end
       else  Edit_y.Text:='Y не визначений'
    else if (PX) then            // X визначений
            if (abs(X)>1E-10) then       // X не дорівнює 0
            begin
               Y:=ln(abs(X));
               Edit_y.Text:=FloatToStr(Y)
            end
            else Edit_y.Text:='Y не визначений'
         else Edit_y.Text:='Y не визначений'
end;
```

ЛЕКЦІЯ 7. ПРОГРАМУВАННЯ ЦИКЛІВ З ВІДОМИМ ЧИСЛОМ ПОВТОРЕНЬ

7.1 Оператор for

Оператор **for** використовується в тому випадку, якщо тіло циклу, тобто деяку послідовність дій (операторів програми) треба виконати кілька разів, причому число повторень заздалегідь відоме._

У спільному вигляді оператор **for** записується таким чином:

for *счетчик:= нач_знач* **to** *кон_знач* **do**
begin
// тут оператори тіла циклу через крапку з комою
end;

де:

лічильник – змінна-лічильник числа повторень циклу; нач_знач – вираження, що визначає початкове значення лічильника циклу; кон_знач – вираження, що визначає кінцеве значення лічильника циклу.

Якщо між словами **begin** і end знаходиться тільки один оператор, то слова begin і end можна не писати. Змінна *лічильник*, вирази нач_знач і кон_знач мають бути цілого типа.

Кількість повторень циклу можна обчислити за формулою:

(кон_знач - нач_знач +1).

7.2 Пример программирования цикла for

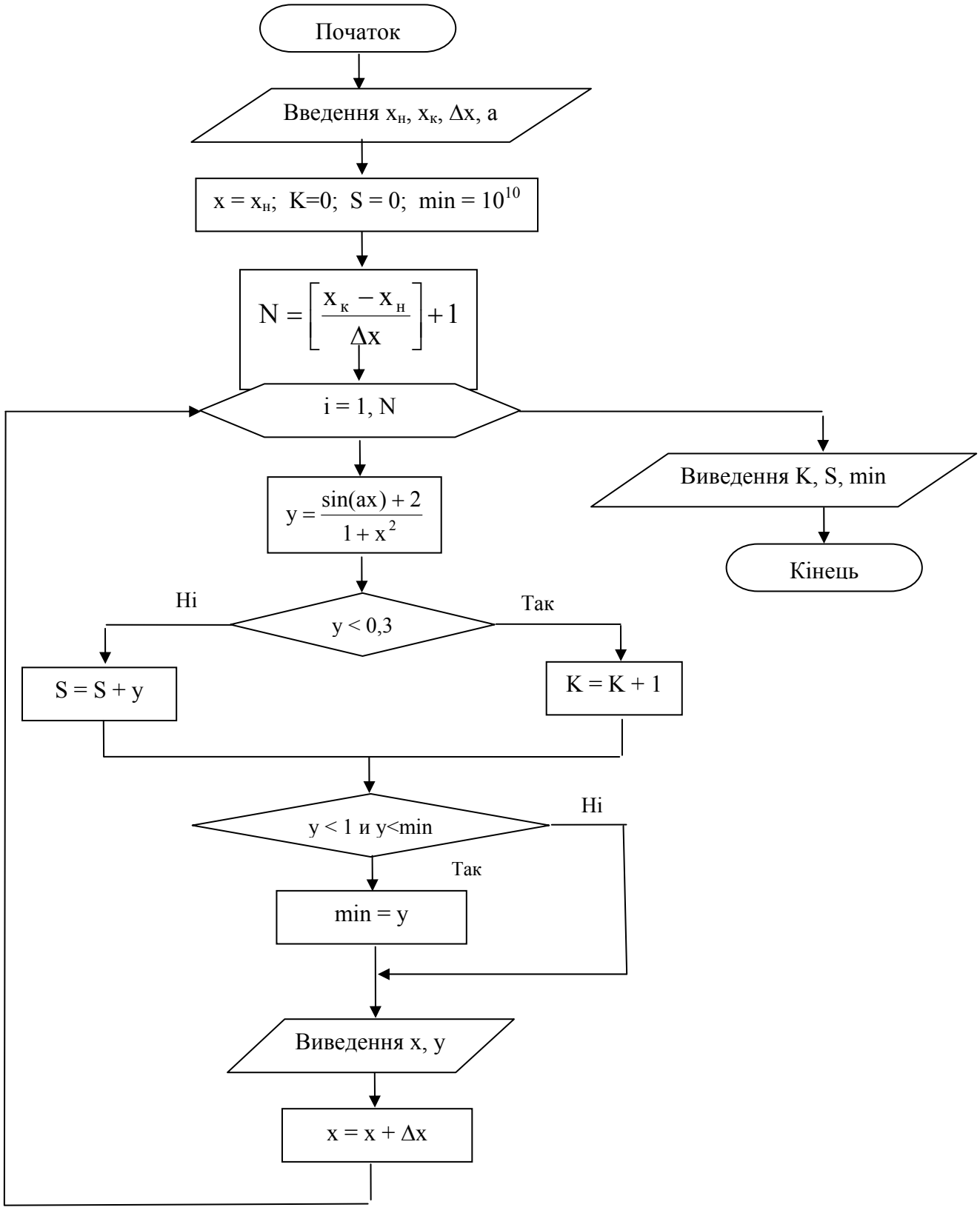
Завдання. Скласти алгоритм і програму, що для $-3 \le x \le 6$ із кроком 0,5 обчислює $y = \frac{\sin(ax)+2}{1+x^2}$, де a = 1,2. Визначити K - кількість $y < 0{,}3$; $S = \sum_{y \ge 0{,}3} y$ и. $\min\{y \mid y < 1\}$

Вхідні дані: $x_н$, $x_к$, Δx, a.
Вихідні дані: K, S, $\min\{y \mid y < 1\}$ і всі значення x и y.

Проаналізуємо вираження для y. Областю визначення y є всі дійсні числа x и a. Обчислення значень y будемо здійснювати в циклі.

Спочатку (до циклу) знайдемо число повторень циклу N і надамо початкові значення змінній x і змінним K, S, min.

Потім у циклі для кожного значення *x* будемо обчислювати відповідне значення *y* і порівнювати його з 0,3. Якщо виявиться, що *y*<0,3 , то збільшимо на одиницю змінну K, інакше – наростимо на величину *y* суму S. Далі перевіримо умову (*y*<1 і *y*<min). Якщо воно виконується, то значення *y* збережемо в змінній min. Виведемо пари значень *x, y* і перейдемо до наступного значення змінної *x*, збільшивши її на крок: $x = x+\Delta x$. На цьому тіло циклу закінчиться. Виведення K, S, min виконаємо після виходу з циклу. Блок-схема алгоритму приведена на малюнку 7.1.



Малюнок 7.1 – Блок-схема циклічного алгоритму з відомим числом повторень

Виведення K, S, min здійснюватимемо у відповідні компоненти **Edit**. Для виведення ж пар **X, Біля** використовуватимемо компонент **Memo –** багаторядкове вікно редагування (6-а кнопка зліва на сторінці **Standard**).

У таблиці. 7.1 перераховані деякі властивості і метод компоненту Memo.

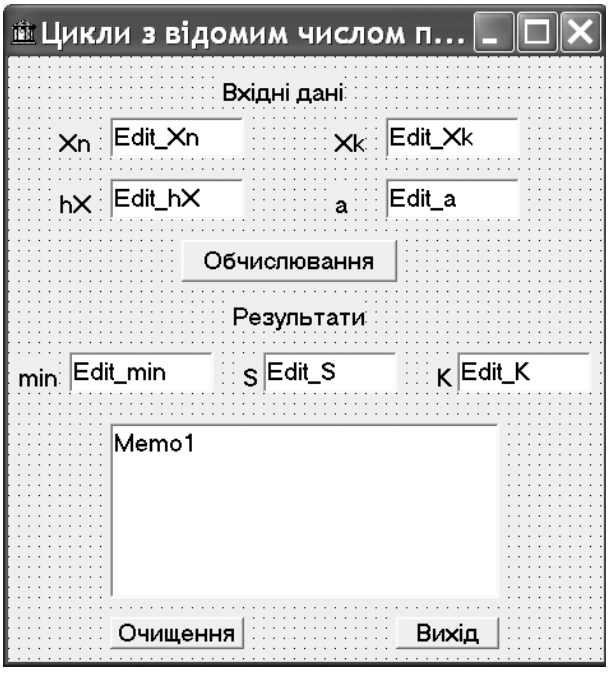
Таблиця 7.1. Властивості і метод компоненту Memo

Властивість	**Визначає**
Text	Текст, що знаходиться в полі **Memo.** Розглядується як єдине ціле
Lines	Текст, що знаходиться в полі **Memo.** Розглядується як сукупність рядків. Доступ до рядка здійснюється по її номеру. Рядки нумеруються від нуля.
Lines.Count	Кількість рядків тексту в полі **Memo.**
Add	Додавання рядка в кінець списку рядків **Lines.**

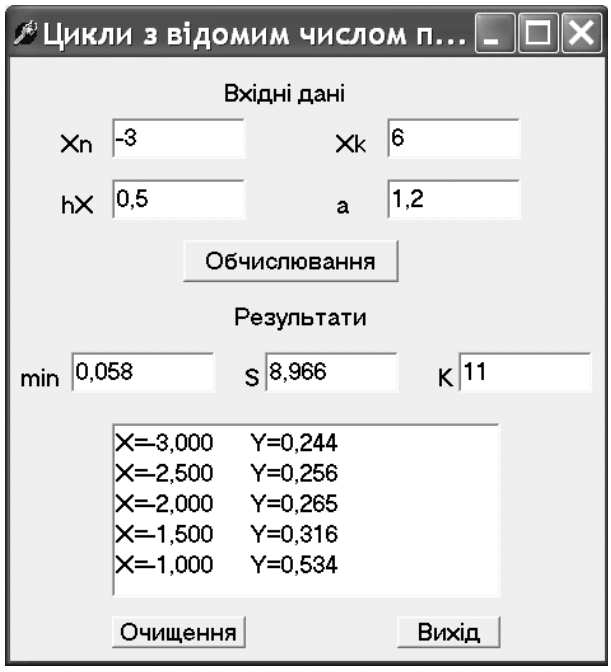
Додати рядок в кінець списку рядків компоненту Memo можна, застосувавши метод **Add**, наприклад:

Memo1.Lines.Add(добавляемая_строка)

Скомпонуємо наступну форму і налагодимо її властивості.



Вхідна **Форма**



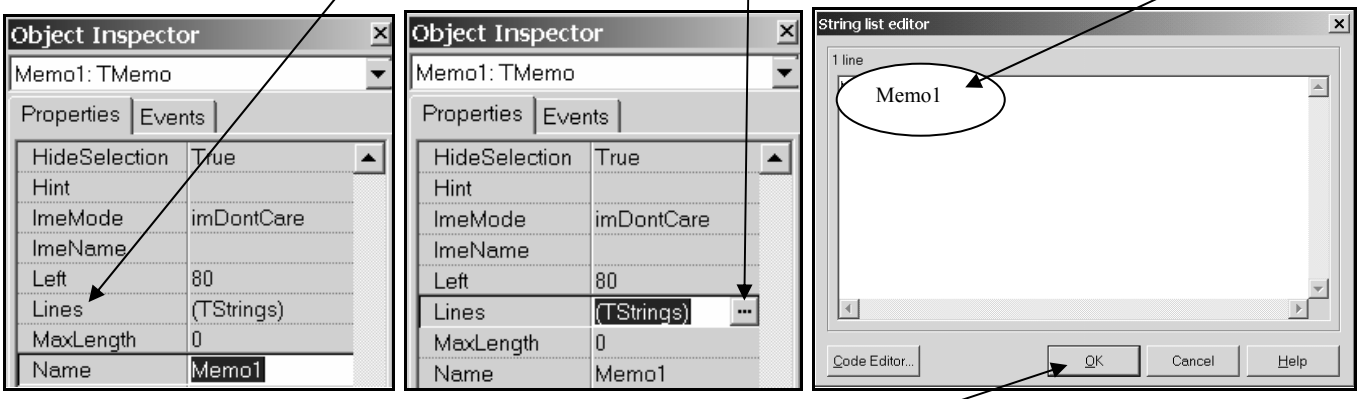
Форма з результатами

У даній **Формі**, у порівнянні з двома попередніми прикладами, є присутнім новий

компонент **Memo1**. Це **багаторядкове** вікно редагування (6-я кнопка ліворуч на сторінці **Standard**), що будемо використовувати для виведення пар значень X и Y. Усі кнопки на формі, а також об'єкти Edit, зв'язані з введенням/виведенням даних, перейменовані зрозумілим чином (тобто змінена їхня властивість **Name**).

Властивості компонента Memo1.

Клацаємо на властивості Lines, потім на кнопці , у вікні, що відкрилося, стираємо текст.



Закінчуємо налагодження натисканням кнопки ОК.

Процедури - оброблювачі подій

Процедура обробки клацання на кнопці **Вихід** є однаковою у всіх додатках (див. приклад п.5.6), тому надалі її не приводитиме.

1. Процедура обробки клацання на кнопці «Очищення» (кн. Button_clear)

```
procedure TForm1.Button_ClearClick(Sender: TObject);
begin
    Edit_Xn.Clear;
    Edit_Xk.Clear;
    Edit_hX.Clear;
    Edit_a.Clear;
    Edit_min.Clear;
    Edit_S.Clear;
    Edit_K.Clear;
    Memo1.Clear;
end;
```

2. Процедура обробки клацання на кнопці «Обчислення» (кн. Button_calc)

```
procedure TForm1.Button_CalcClick(Sender: TObject);
var  Xn,Xk,hX,X,Y,a,min,S:real;
     N,K,i:integer;
begin
   Xn:=StrToFloat(Edit_Xn.Text);
   Xk:=StrToFloat(Edit_Xk.Text);
```

```
   hX:=StrToFloat(Edit_hX.Text);
   a:=StrToFloat(Edit_a.Text);
   N:=trunc((Xk-Xn)/hX)+1;
   X:=Xn; K:=0; S:=0; min:=1E10;
   for i:=1 to N do
   begin
      Y:=(sin(a*X)+2)/(1+x*x);
      if (Y<0.3) then K:=K+1
                 else S:=S+Y;
      if (Y<1) and (Y<min) then min:=Y;
      // додавання в Memo1 чергового рядка
      Memo1.Lines.Add('X = '+FormatFloat('##0.000',X)+
      '      Y = '+FormatFloat('##0.000',Y));
      X:=X+hX;
   end;
   Edit_K.Text:=IntToStr(K);
   Edit_S.Text:=FormatFloat('##0.000',S);
   Edit_min.Text:=FormatFloat('##0.000',min);
end;
```

ЛЕКЦІЯ 8. ПРОГРАМУВАННЯ ЦИКЛІВ З НЕВІДОМИМ ЧИСЛОМ ПОВТОРЕНЬ

8.1 Оператор while

Оператор (цикл) **while** використовується в тому випадку, якщо деяку послідовність дій (операторів програми) треба виконати кілька разів, причому необхідне число повторень під час розробки програми невідоме і може бути визначено тільки під час роботи програми.

Типовими прикладами використання циклу **while** є обчислення із заданою точністю, пошук в масиві або у файлі.

У спільному вигляді оператор **while** записується таким чином:

while (*умова*) **do**
begin
  // тут оператори тіла циклу через крапку з комою
end

Тут *умова* – це вираження логічного типа, що визначає умову продовження циклу.

Оператор **while** виконується таким чином.

Спочатку обчислюється значення вираження *умова.*

Якщо *умова* дорівнює **False** (*умова* не виконується),то на цьому виконання циклу **while** завершується.

Якщо *умова* дорівнює **True** (*умова* виконується), то виконуються оператори тіла циклу, які розташовані між **begin** і **end**. Після цього знову перевіряється виконання *умови*. Якщо *умова* виконується, то оператори тіла циклу виконуються ще раз. І так до тих пір, поки *умова* не стане помилковою (**False**).

Алгоритм, відповідний операторові **while**, представлений на мал. 1.11.

8.2 Приклад програмування циклу while

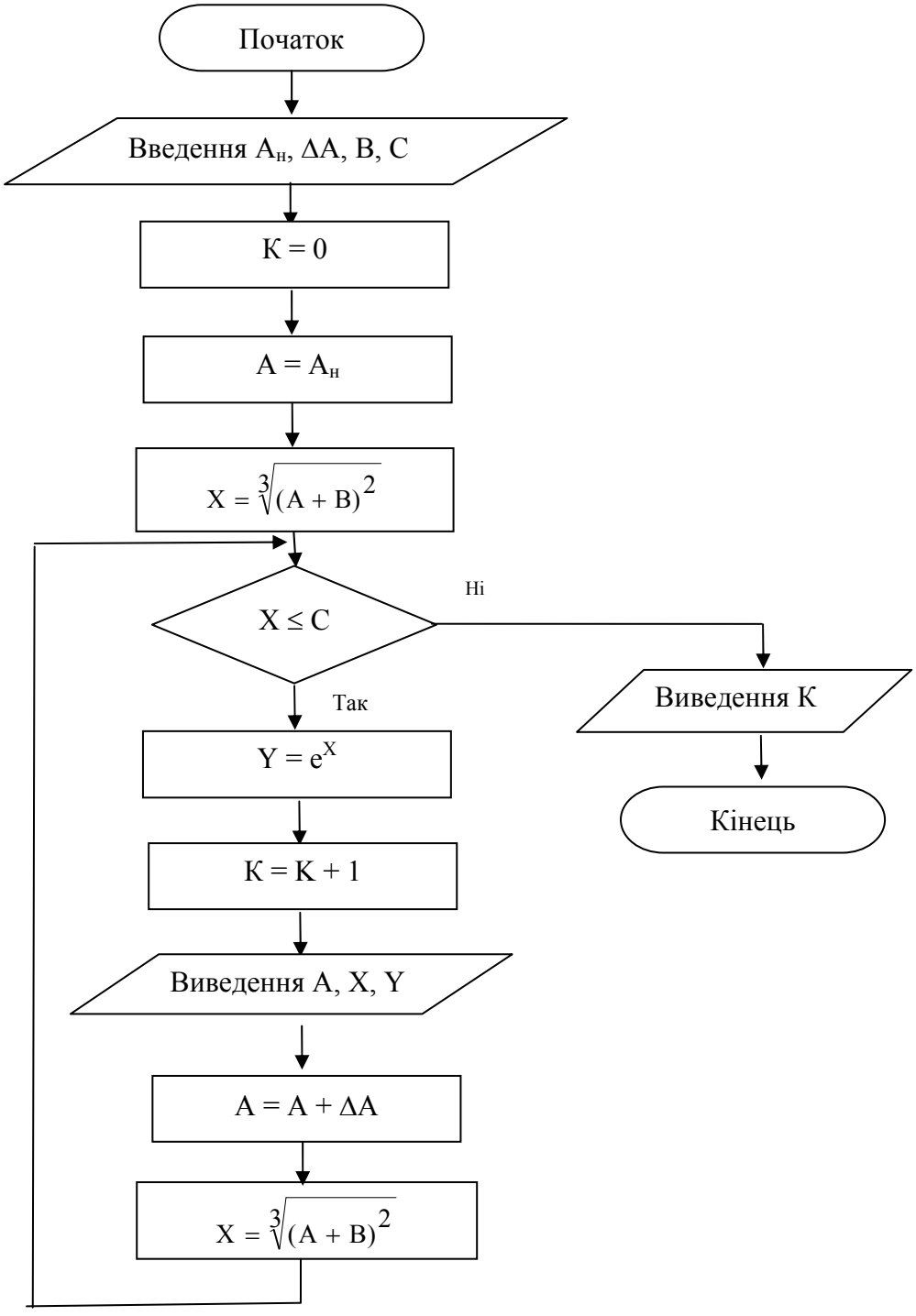
Завдання. Скласти блок-схему алгоритму і програму розрахунку на Object Pascal наступних значень:

$$Y = e^X, \text{ где } X = \sqrt[3]{(A+B)^2}.$$

Обчислення робити доти, поки X не стане більше C. Визначити K - кількість обчислених значень Y.

Вхідні дані: A = 2,5; ΔA = 0,1; B = 3; C = 5.

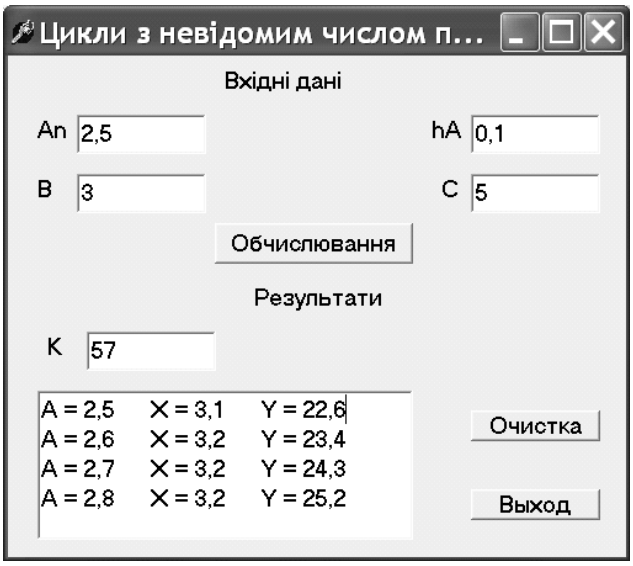
Вихідні дані: K и всі значення A, X, Y.



Малюнок 8.1 – Блок-схема циклічного алгоритму з невідомим числом повторень

Скомпонуємо форму, показану нижче. Так само, як у попередньому прикладі 6.3, для виведення трійки значень A, X, Y будемо використовувати багаторядкове вікно редагування Memo1 (компонент  на сторінці **Standard).**

Вхідна **Форма**



Форма з результатами

1. Процедура обробки клацання на кнопці «Обчислення» (кн. Button_calc)

```
procedure TForm1.Button_CalcClick(Sender: TObject);
      var An,hA,X,Y,A,b,c:real;
       K:integer;
begin
   An:=StrToFloat(Edit_An.Text);
   hA:=StrToFloat(Edit_hA.Text);
   B:=StrToFloat(Edit_b.Text);
   C:=StrToFloat(Edit_c.Text);
   K:=0;    A:=An;
   X:=exp(ln(sqr(A+B))/3);
while (X<=C) do
begin
      Y:=exp(X);
      K:=K+1;
      Memo1.Lines.Add('A = '+FormatFloat('##0.0',A)+
      '      X = '+FormatFloat('##0.0',X)+
      '      Y = '+FormatFloat('##0.0',Y));
      A:=A+hA;
      X:=exp(ln(sqr(A+B))/3);
end;
    Edit_K.Text:=IntToStr(K);
end;
```

2. Процедура обробки клацання на кнопці «Очищення» (кн. Button_clear)

```
procedure TForm1.Button_ClearClick(Sender: TObject);
begin
    Edit_An.Clear;
    Edit_hA.Clear;
    Edit_b.Clear;
    Edit_c.Clear;
    Edit_K.Clear;
    Memo1.Clear;
end;
```

ЛЕКЦІЯ 9. ПРОГРАМУВАННЯ ВКЛАДЕНИХ ЦИКЛІВ

9.1 Переривання циклів

Перервати цикл до його завершення, можна 4-мя способами:

1) за допомогою процедури **Continue**, яка перериває поточний крок циклу і передає управління наступному кроку цього ж циклу;
2) за допомогою оператора **Break,** який перериває цикл і передає управління операторові, наступному за циклом;
3) за допомогою оператора переходу на мітку **Goto;**
4) використовуючи логічну змінну (*прапора закінчення циклу*).

Перервати виконання програми можна процедурою **Exit.**

У наступному прикладі ми скористаємося процедурою **Continue –** для переривання поточного кроку циклу.

9.2 Приклад програмування вкладених циклів

Завдання. Скласти блок-схему алгоритму і програму розрахунку на Object Pascal наступних значень:

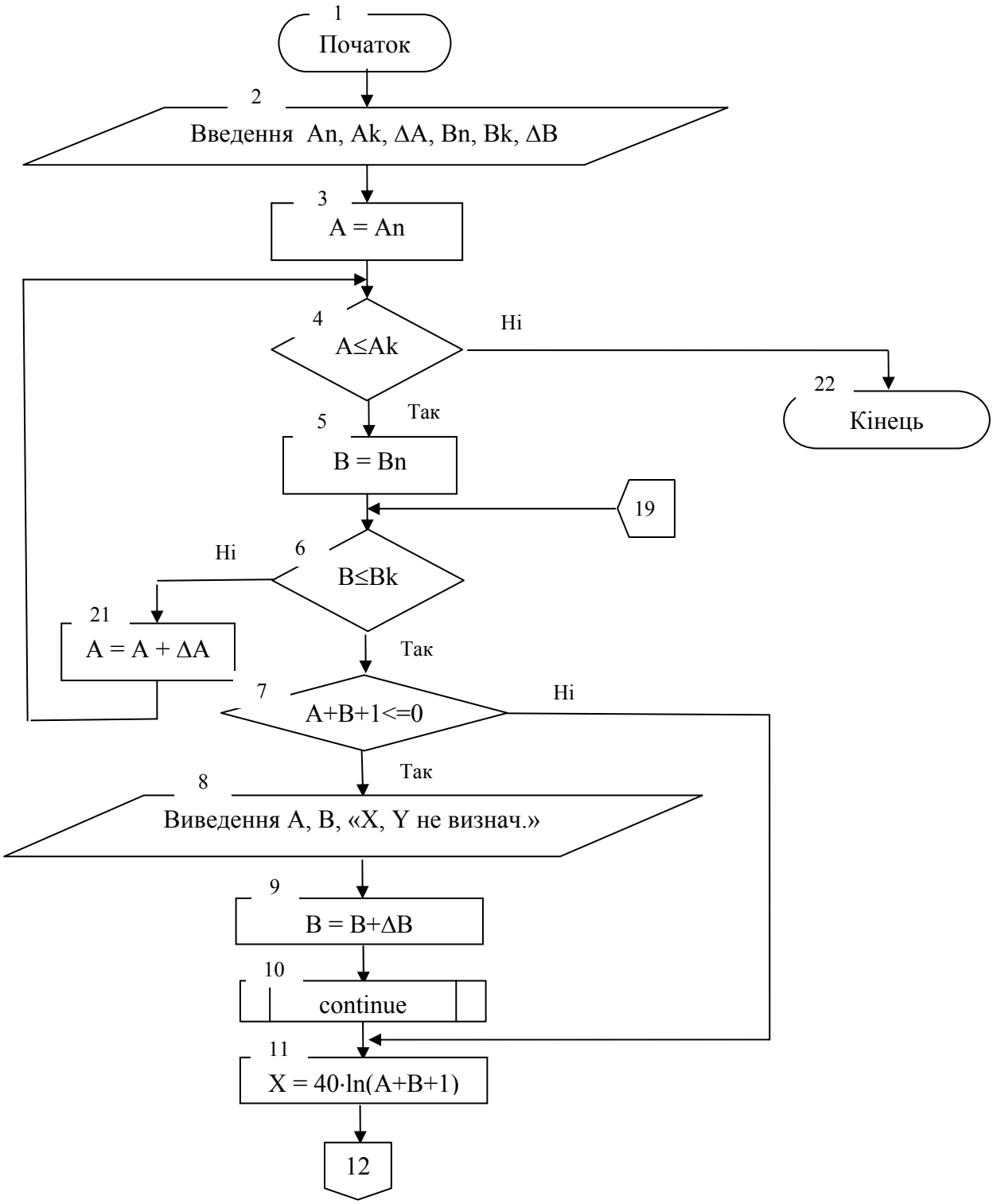
$$X = 40 \cdot \ln(A + B + 1); \qquad Y = \begin{cases} \dfrac{X-5}{B} + \sqrt{A^2 + X^2}, & \text{если } X \geq 5; \\ \dfrac{X - A}{\sqrt{X^2 + 1}}, & \text{если } X < 5. \end{cases}$$

Вхідні дані: $-1 \leq A \leq 8$; $\Delta A = 3$; $-1 \leq B \leq 2$; $\Delta B = 0{,}5$.
Вихідні дані: A, B, X, Y.

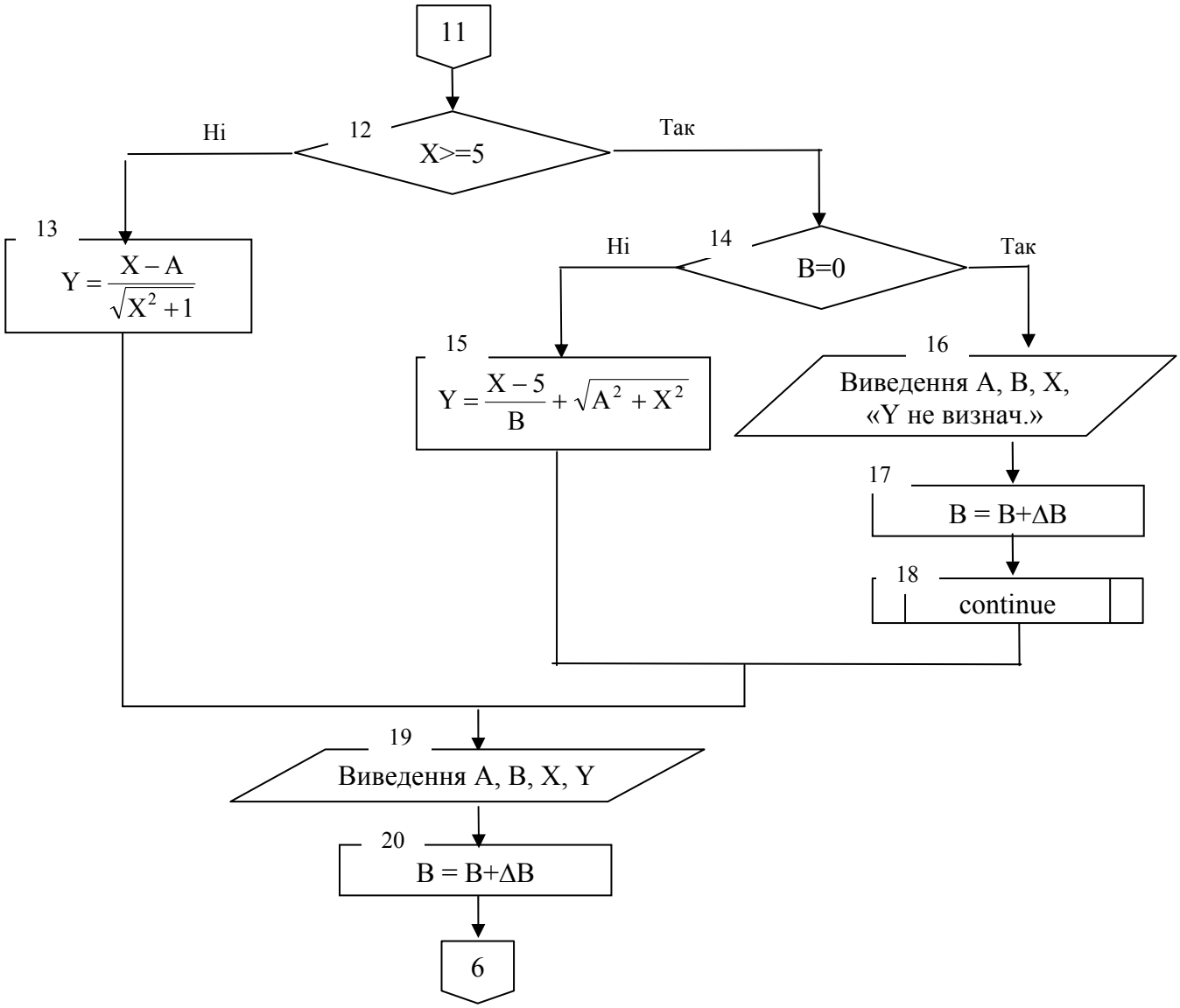
Проаналізуємо значення, що обчислюються. Кожна з змінних A и B змінюється у своїх межах зі своїм кроком. Тому організуємо два незалежних цикли – по A и по B відповідно. Один з них, наприклад, цикл по A зробимо зовнішнім, а інший, цикл по B – внутрішнім.

Значення X є визначеним для тих пар A и B, що задовольняють співвідношенню A+B+1>0. Якщо ця умова не виконується, то X и Y є невизначеними. В внутрішньому циклі перевіряємо цю умову. Якщо вона не виконується, то виводимо відповідне повідомлення, перериваємо поточний крок внутрішнього циклу і переходимо до наступного кроку, змінивши попереднє значення B на ΔB. У програмі на Паскале переривання поточного кроку циклу і перехід до наступного виконує процедура **continue**. У блок-схемі також звертаємося до цієї процедури. Якщо значення X визначене і X≥5, то перевіряємо знаменник на нуль. Якщо B=0, то виводимо повідомлення про невизначеність Y і переходимо до наступного кроку циклу по B – аналогічно тому, як тільки що було описано.

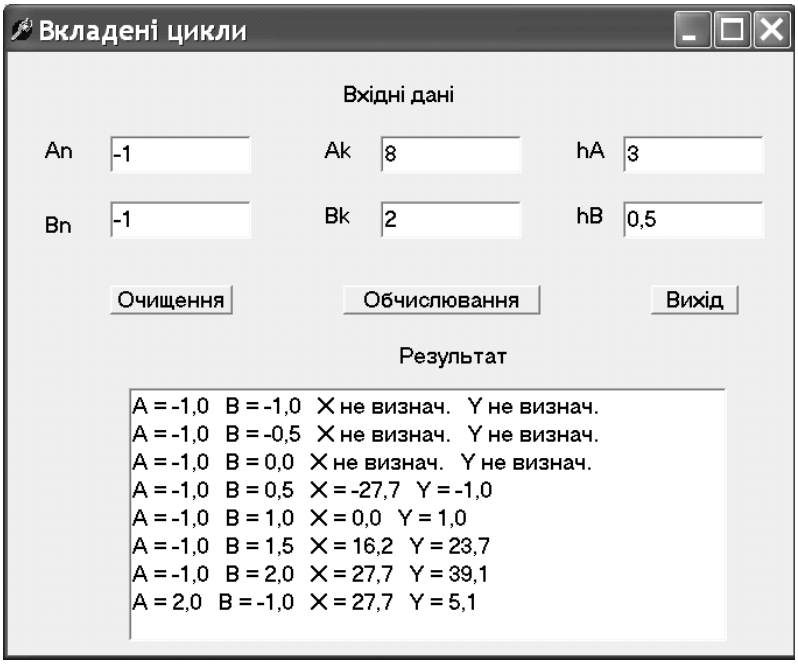
Блок-схема алгоритму показана на малюнку 9.1.



Малюнок 9.1 – Блок-схема алгоритму з вкладеними циклами



Малюнок 9.1 (продовження)



Властивості об'єктів форми налагоджуються так само, як у прикладах п.5.6 и п.7.2.

Для виведення результатів використовуємо багаторядкове вікно редагування Memo (див. приклад п.7.2.).

Процедури обробки клацань на кнопках **Очищення** і **Вихід** аналогічні процедурам з попередніх прикладів.

Процедура обробки клацання на кнопці «Обчислення» (кн. Button_calc)

```
procedure TForm1.Button_CalcClick(Sender: TObject);
var A,An,Ak,h,B,Bn,Bk,h,X,Y:real;
begin
    //  Уведення вихідних даних
    An:=StrToFloat(Edit_An.Text);
    Ak:=StrToFloat(Edit_Ak.Text);
    hA:=StrToFloat(Edit_hA.Text);
    Bn:=StrToFloat(Edit_Bn.Text);
    Bk:=StrToFloat(Edit_Bk.Text);
    hB:=StrToFloat(Edit_hB.Text);
    A:=An;
    while (A<=Ak)do           // початок циклу по A
    begin
       B:=Bn;
       while (B<=Bk) do       // початок циклу по B
       begin
          if (A+B+1<=1E-10) then   //перевірка умови A+B+1=0
          begin
            Memo1.lines.Add('A = '+FormatFloat('##0.0',A)+
                            '   B = '+FormatFloat('##0.0',B)+
                            '   X не визнач.'+
                            '   Y не визнач.');
              B:=B+hB;
              CONTINUE
          end;
          X:=40*LN(A+B+1);
          if (X>=5) then
             if (abs(B)<1E-10) then   // перевірка B=0
             begin
                 Memo1.lines.Add('A = '+FormatFloat('##0.0',A)+
                                 '   B = '+FormatFloat('##0.0',B)+
                                 '   X = '+FormatFloat('##0.0',X)+
                                 '   Y не визнач.');
                 B:=B+hB;
                 CONTINUE
             end
             else Y:=(X-5)/B+sqrt(A*A+X*X)
          else Y:=(X-A)/sqrt(X*X+1);
          Memo1.lines.Add('A = '+FormatFloat('##0.0',A)+
                          '   B = '+FormatFloat('##0.0',B)+
                          '   X = '+FormatFloat('##0.0',X)+
                          '   Y = '+FormatFloat('##0.0',Y));
          B:=B+hB;
       end;           // кінець циклу по B
       A:=A+hA;
    end;              // кінець циклу по A
end;
```

ЛЕКЦІЯ 10. ОБРОБКА ОДНОВИМІРНИХ МАСИВІВ

10.1 Оголошення одновимірного масиву

Масив, як і будь-яка змінна програми, перед використанням має бути оголошений в розділі оголошення змінних. У спільному вигляді оголошення масиву виглядає таким чином:

имя:array[нижний_индекс..верхний_индекс] of тип

де:

ім'я – ім'я масиву;

array – службове слово мови Object Pascal, що позначає «масив»;

нижний_индекс і верхний_индекс – цілі константи, що визначають діапазон зміни індексу елементів масиву і, неявно, кількість елементів (розмір) масиву;

тип – тип елементів масиву.

Приклади оголошення масивів:

```
temper:array[1..31] of real;     // масив, що містить максимум 31 дійсне число
koef:array[0..2] of integer;     // масив, що містить максимум 3 цілих числа
name:array[1..30]  of string[25]; // масив, що містить максимум 30 рядків довжини 25.
```

При оголошенні масиву зручно використовувати **іменовані константи і типів**. Іменована константа оголошується в розділі оголошення констант, який зазвичай розташовують перед розділом оголошення змінних. Зачинається розділ оголошення констант словом **const.** Після констант оголошують типів (службове слово **type**).

Наприклад, наступні три оголошення описують один і той же масив A, що полягає не більш, ніж з 20 цілих чисел:

```
a)  const  n_max=20;
    type  vect=array[1..n_max] of integer;
    var A:vect;
b)  type  vect=array[1..20] of integer;
    var A:vect;
c)  A: array[1..20] of integer;
```

10.2 Доступ до елементу масиву

Для того, щоб в програмі використовувати елемент масиву, треба вказати ім'я масиву і номер елементу (індекс), уклавши індекс в квадратні дужки. Як індекс можна використовувати константу або вираження цілого типа, наприклад:

```
name[l]:= 'Шахтар';
d := koef[1]*koef[l]-4*koef[2]*koef[1];
ShowMessage(name[n+1]);
temper[i]:= StrToFloat(Editl.text);
```

10.3 Локальні і глобальні змінні

Якщо змінна (масив.) оголошена в процедурі обробки події, то вона називається **локальною**. Локальна змінна відома тільки усередині даної процедури. У момент запуску цієї процедури для змінної виділяється пам'ять, а у момент завершення роботи процедури пам'ять звільняється, і змінна втрачає своє значення.

Якщо змінна оголошена усередині розділу **Implementation (**тобто після слова **Implementation),** то вона називається **глобальній** змінній даного модуля. Така змінна відома у всіх процедурах даного модуля. Пам'ять під цю змінну виділяється у момент запуску модуля і не звільняється до завершення модуля. Якщо локальна і глобальна змінні мають однакові імена, то сильнішим є локальне оголошення.

10.4 Ініціалізація глобальних масивів

При оголошенні глобального масиву (у розділі **Implementation)** одночасно з оголошенням масиву можна виконати його ініціалізацію, тобто привласнити початкові значення елементам масиву. Наприклад:

Implementation
a: **array**[10] **of integer** = (0,0,0,0,0,0,0,0,0,0);
Team: **array**[1..4] **of String**[10]=
('Шахтер','дінамо','спартак', 'Зеніт');

Звернете увагу, що **кількість елементів списку ініціалізації повинна відповідати розмірності масиву.** Якщо це буде не так, то компілятор виведе повідомлення про помилку: Number ***of elements differs from declaration*** (кількість елементів не відповідає вказаному в оголошенні).

При спробі ініціалізувати локальний масив, компілятор виводить повідомлення про помилку: ***Cannot initialize local variables*** (локальна змінна не може ініціалізувати).

10.5 Введення і виведення масивів

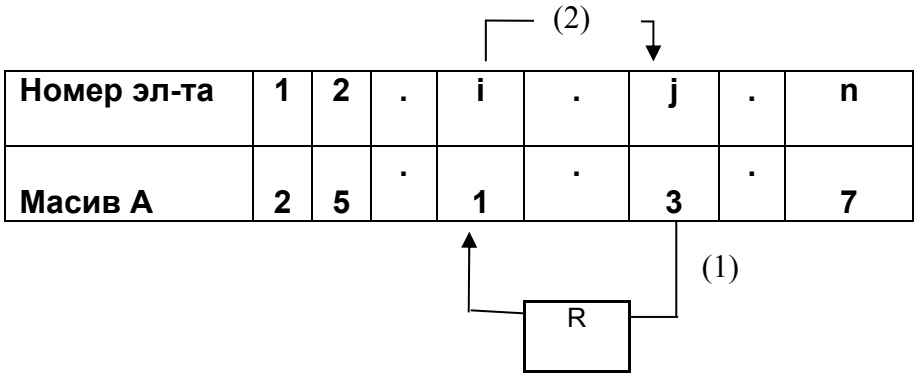
Під введенням масиву розуміється процес отримання від користувача (або з файлу) під час роботи програми значень елементів масиву.

Для введення і виведення одновимірного масиву можна використовувати компоненти **Memo** або **StringGrid**. У наступному прикладі ми скористаємося вже знайомим компонентом **Memo.**

10.6 Стандартні операції над одновимірними масивами

1. Перестановка двох елементів

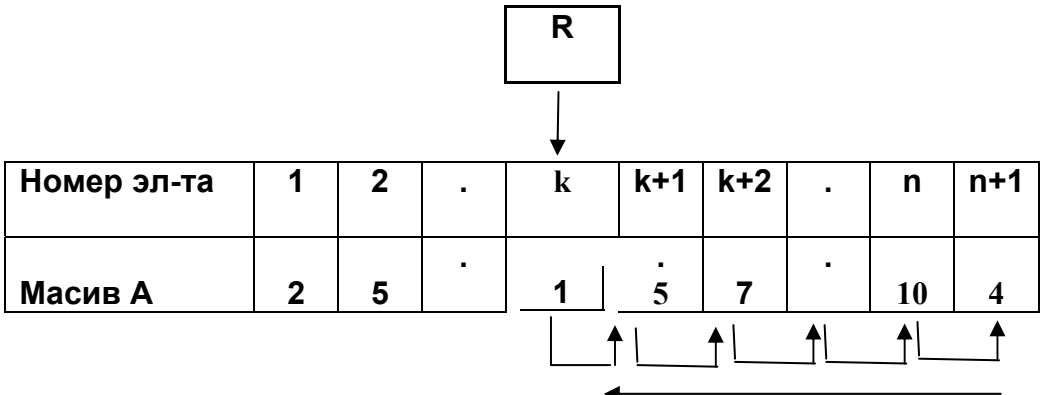
Завдання. Поміняти місцями i-й i j-й елементи масиву А.




```
R:=A[j];
A[j]:=A[i];
A[i]:=R;
```

2. Вставка нового елементу у вказане місце масиву

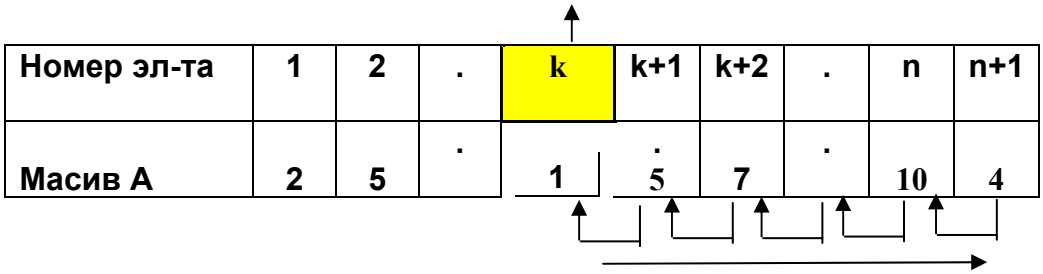
Завдання. Вставити елемент R в к-ую позицію масиву А.




```
//   зсуваємо хвіст масиву, зачинаючи з n-го елементу
//   до к-го управо на 1 позицію
for i:=1 to n downto до do
A[i+1]:=A[i];
n:=n+1;        // збільшуємо розмірність масиву А на 1
//вставляємо елемент в к-ю позицію
A[k]:=R;
```

3. Видалення елементу з вказаного місця масиву

Завдання. Видалити елемент з к-ой позиції масиву А.




```
//зрушуємо хвіст масиву, зачинаючи з n-го елементу, управо на 1 позицію
     for i:=k to n-1 do
     A[i]:=A[i+1];
     n:=n-1;        // зменшуємо розмірність масиву А на 1
```

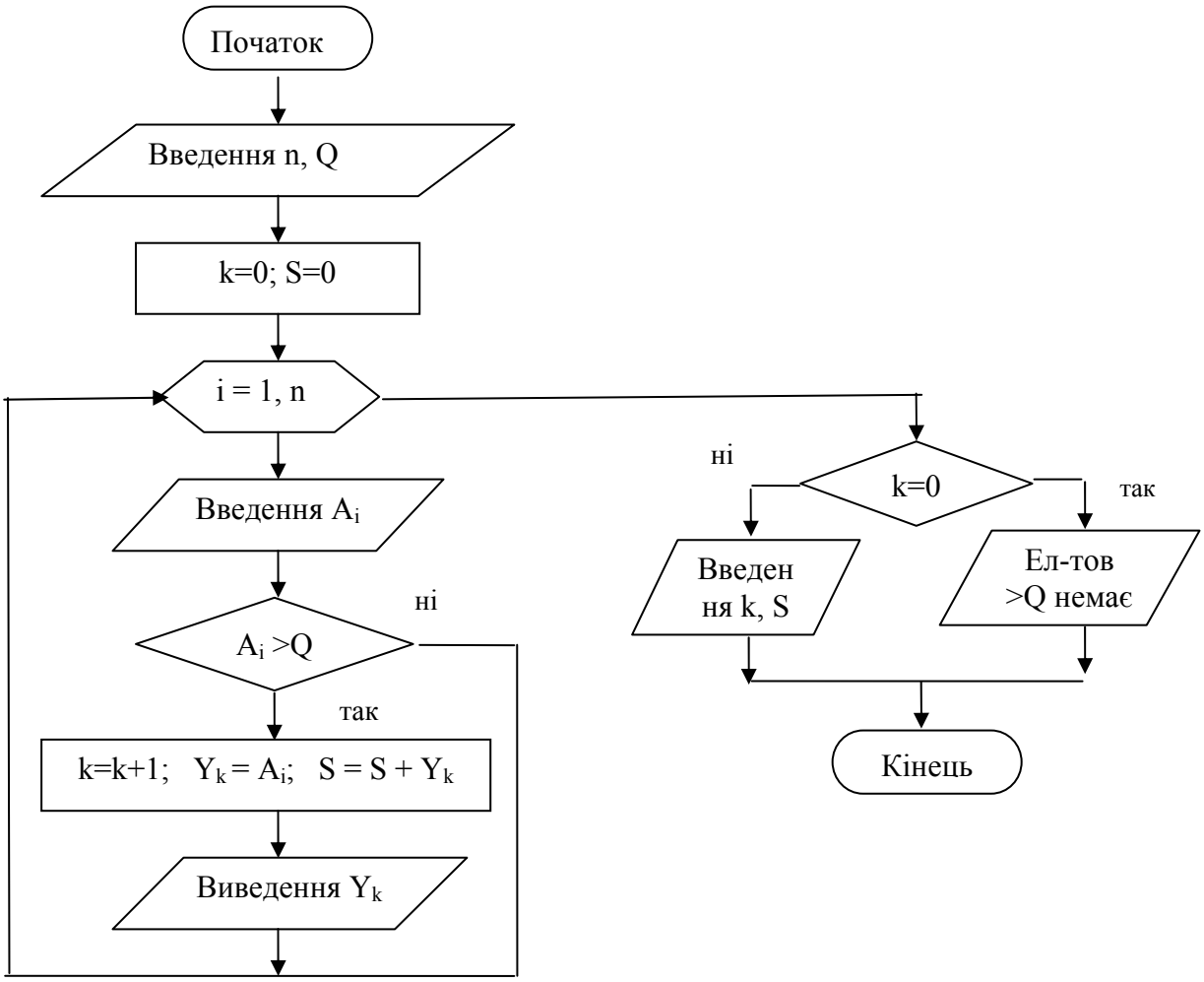
10.7 Приклад обробки одновимірного масиву

Завдання. Записати всі елементи масиву A=(A1, A2 ..., An), що задовольняють умові Ai>Q, підряд в масив B. Знайти кількість (k) і суму (S) таких елементів. Якщо таких елементів немає, то видати відповідне повідомлення.

Початкові дані: масив A, Q.

Вихідні дані: масив B, до, S.

Блок-схема алгоритму представлена на малюнку 10.1.



Малюнок 10.1 – Блок-схема алгоритму обробки одномірних масивів

Алгоритм включає наступні дії:

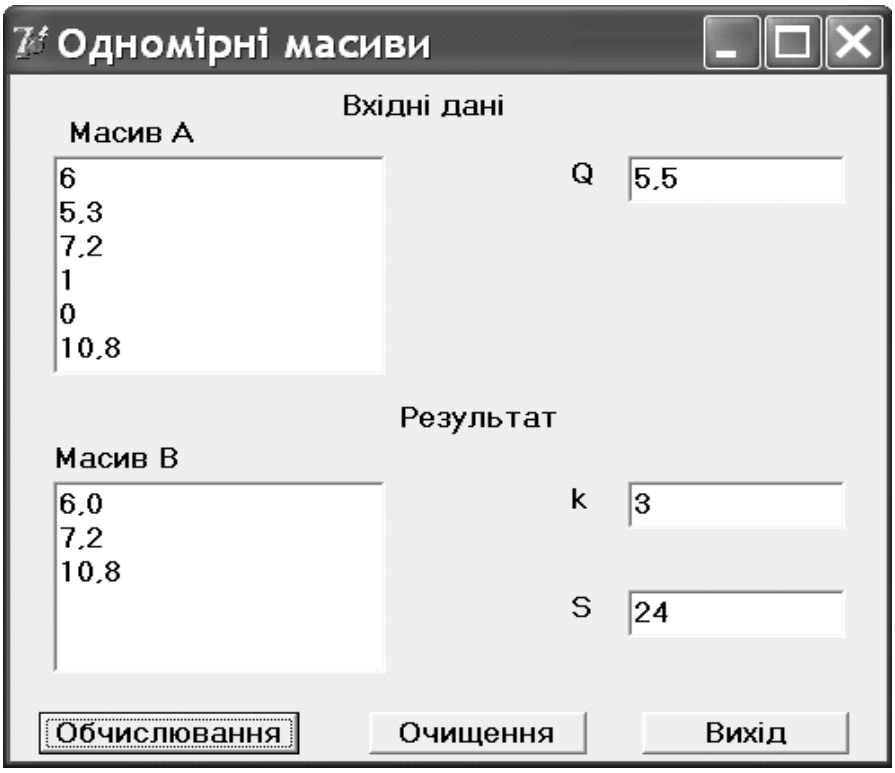
- введення масиву A;
- пошук елементів A_i>Q і підрахунок суми S і кількості k таких елементів;

Опишемо алгоритм докладніше.

Введення масиву. Для роботи з одномірним масивом A спочатку введемо його розмірність (n), тобто кількість елементів масиву. Потім для введення всіх елементів масиву організуємо цикл по номері елемента i. У цьому циклі введемо черговий елемент A_i.

Пошук елементів A_i>Q**, підрахунок S, k.** Для підрахунку зазначених величин використовуємо цикл по індексі елементів масиву A, організований для введення масиву. Перед циклом величинам **S, k** присвоїмо початкове значення 0. Усередині циклу перевіримо умову: A_i>Q. Якщо умова виконується, то методом накопичення обчислимо **S** і **k.**

Блок-схема алгоритму представлена на малюнку 10.1.



Скомпонуємо форму, показану ліворуч. Для введення масиву A будемо використовувати багато-рядкове вікно редагування Memo_A, а для виведення масиву B – Memo_B (компонент  на сторінці **Standard).** Усі кнопки на **Формі**, а також об'єкти Edit, пов'язані з введенням або виведенням даних, перейменовано зрозумілим чином (тобто змінена їхня властивість **Name**). Процедура обробки клацання на кнопці **Очищення** аналогічна попереднім прикладам, і ми її пропускаємо.

Процедура обробки клацання на кнопці «Обчислення» (кн. Button_calc)

```
procedure TForm1.Button_calcClick(Sender: TObject);
type mas=array[1..10] of real;
var  A,B:mas;
     Q,S:real;
     i,k,N:integer;
begin
     Q:=StrToFloat(Edit_Q.Text);
     N:=Memo_A.Lines.Count;     // розмірність A
     k:=0; S:=0;
     for i:=1 to N do
     begin
        A[i]:=StrToFloat(Memo_A.Lines[i-1]);
        if A[i]>Q then
        begin
           k:=k+1;
           B[k]:=A[i];
           S:=S+B[k];
           Memo_B.Lines.Add(FormatFloat('##0.0',B[k]))
        end;
     end;
     If k=0 then ShowMessage('Масив A не містить елементів >'
```

```
                                +FloatToStr(Q))
      else
      begin
        Edit_k.Text:=IntToStr(k);
        Edit_S.Text:=FloatToStr(S)
      end
end;
```

ЛЕКЦІЯ 11. ОБРОБКА ДВОВИМІРНИХ МАСИВІВ

11.1 Оголошення двовимірного масиву

У спільному вигляді оголошення двовимірного масиву виглядає так:

ім'я:**array**[НижняГраниця1..ВерхняГраниця1,
НижняГраниця2..ВерхняГраниця2] of Тип

де:

ім'я – ім'я масиву;

array – службове слово мови Pascal, вказуюче, що оголошуваний елемент даних є масивом;

НижняГраниця1, ВерхняГраниця1, НижняГраниця2, ВерхняГраниця2 – цілі константи, що визначають діапазон зміни індексів;

Тип – тип елементів масиву.

Приклади оголошення масивів:

a)
```
const  n_max=10;
       m_max=5;
type  matr=array[1..n_max,1..m_max] of integer;
var B:matr;
```

b)
```
type  matr=array[1..n_max,1..m_max] of integer;
var B:matr;
```

c)
```
B: array[1..10,1..5] of integer;
```

Це три різні оголошення одного і того ж двовимірного масиву, що складається максимум з 10 рядків і максимум з 5 стовпців і кожен елемент якого є ціле число.

11.2 Доступ до елементу масиву

Щоб використовувати елемент масиву, потрібно вказати ім'я масиву і індекси елементу. Перший індекс зазвичай відповідає номеру рядка таблиці, другий – номеру стовпця. Наприклад, елемент B[2,j] указує на елемент масиву B, що стоїть на перетині 2-го рядка і j-го стовпця.

11.3 Введення і виведення масивів

Для введення початкових даних і відображення результату використовується компонент StringGrid (сітка, таблиця), властивості якого приведені в таблиці. 11.1.

Таблиця 11.1. Найбільш важливі властивості StringGrid

Властивість	**Значення**

ColCount	Число стовпців таблиці За умовчанням=5
RowCount	Число рядків таблиці За умовчанням=5
FixedCols	Число фіксованих (заголовних) стовпців За умовчанням=1
FixedRows	Число фіксованих (заголовних) рядків За умовчанням=1
Options.goEditing	Дозвіл на редагування вічок. За умовчанням заборонено (False)
Options.goTab	Дозвіл користуватися клавішею табуляції для переходів між елементами таблиці За умовчанням заборонено (False)
DefaultColWidth	Замовчувана ширина стовпця в пікселях(=64)
DefaultRowHeight	Замовчувана висота рядка в пікселях (=24)
GridLineWidth	Товщина лінії таблиці в пікселях (За умовчанням=1)
Cells[0..ColCount-1, 0..RowCount-1]	Двовимірний масив вічок строкового типа. 1-й індекс – номер стовпця, 2-ий – номер рядка. Рядки і стовпці нумеруються, починаючи від нуля

11.4 Приклад обробки двовимірного масиву

Завдання. Дано цілочислову матрицю A = $||a_{i,j}||_{n,m}$. Сформувати вектор B=(b_1, b_2, … b_n), кожен елемент якого дорівнює сумі непарних елементів відповідного рядка матриці A. Знайти індекси (i0, j0) непарного елемента матриці A, розташованого в самому нижньому її рядку і самому правому стовпці. Якщо матриця не містить непарних елементів, видати відповідне повідомлення. Передбачити можливість введення матриці уручну і заповнення її випадковими числами.

Вхідні дані: матриця A = $||a_{i,j}||_{n,m}$.

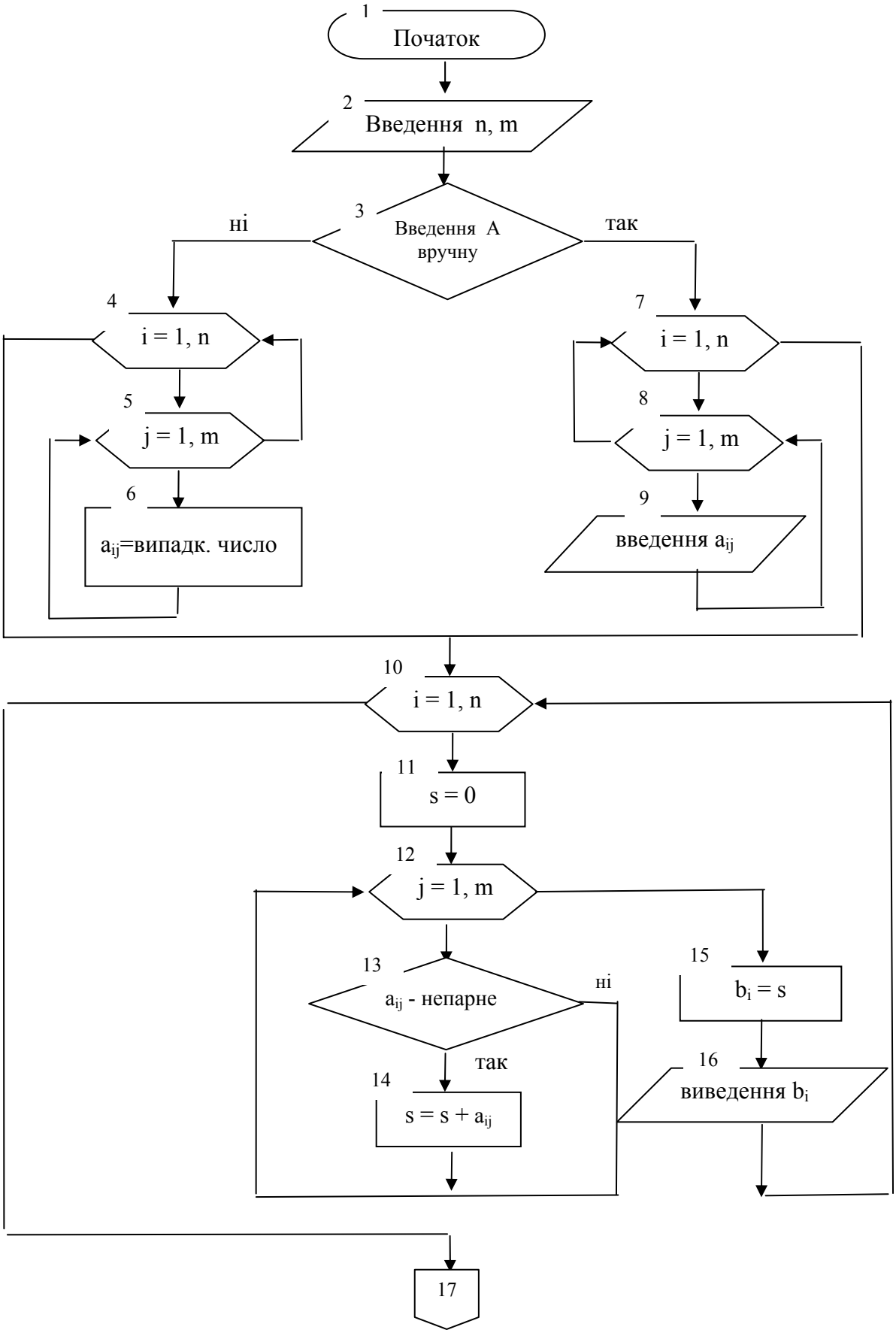
Вихідні дані: вектор B, i0, j0.

Блок-схема алгоритму показана на малюнку 11.1. Алгоритм складається з введення матриці A, формування вектора B, пошуку індексів (i0,j0) непарного елемента, розташованого в самій нижній правій позиції матриці A.

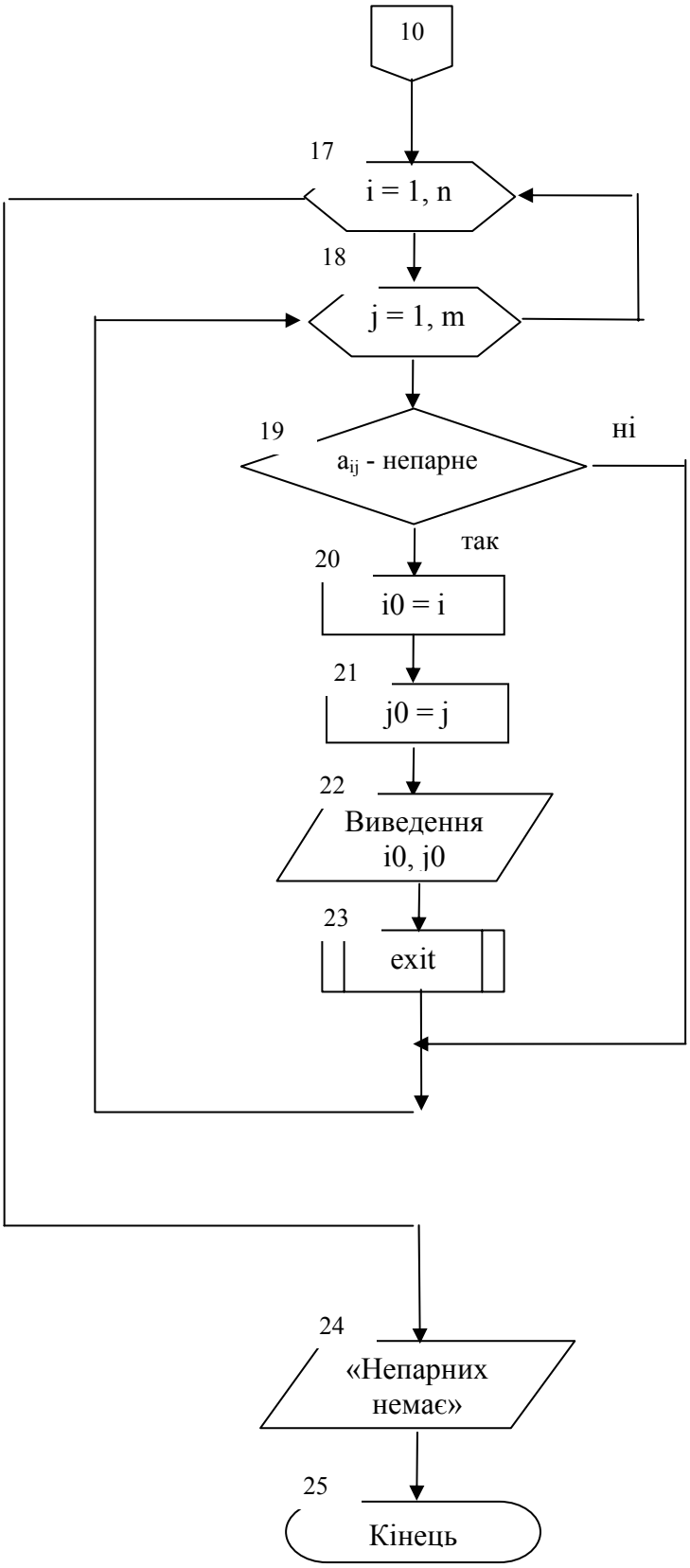
Введення матриці. При роботі з багатомірними масивами спочатку вводиться розмірність матриці, тобто кількість її рядків (n) і стовпців (m). Потім для введення елементів матриці $a_{i,j}$ організується пара вкладених циклів – по рядках (i) і по стовпцях (j). У нашому випадку необхідно додати перевірку, як вводити матрицю – уручну або випадковим чином.

Формування вектора B. Вектор B містить n елементів – по одному для кожного рядка матриці A. Для формування елемента b_i (i=1,2,…,n) організуємо зовнішній цикл по рядках, усередині якого методом накопичення обчислимо суму (s) непарних елементів i-ої рядка. Накопичення суми здійснимо у внутрішньому циклі по стовпцях (j). Після закінчення циклу по j занесемо обчислену суму s в елемент b_i.

Пошук індексів (i0,j0) непарного елемента, розташованого в самій нижній правій позиції матриці A. Перед початком пошуку індексові i0 привласнимо нульове значення – ознака того, що елемент поки ще не знайдений. Потім організуємо пари циклів: зовнішній – по рядках (i), оскільки в першу чергу потрібно визначити номер рядка, і внутрішній – по стовпцях (j). Обидва цикли – по убуванню індексу, тому що нас цікавить номер самого нижнього рядка і номер самого правого стовпця. Як тільки непарний елемент знайдений, його індекси виводяться на екран і програма зупиняється (процедура exit). Якщо елемент не знайдений (тобто i0 залишилося рівним нулеві), видаємо повідомлення про відсутність елемента.

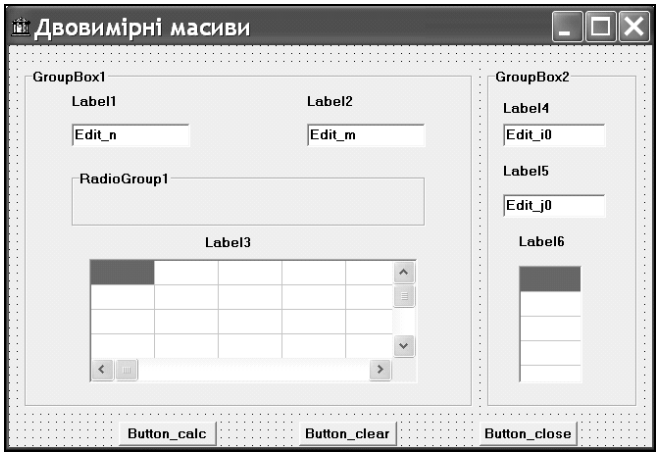


Малюнок 11.1 – Блок-схема алгоритму обробки двовимірних масивів

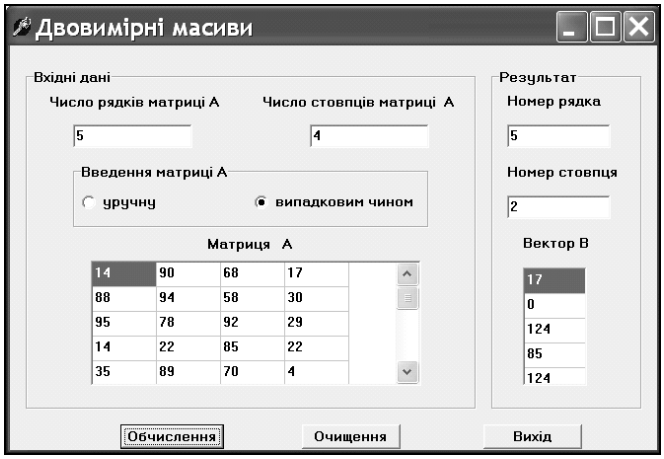


Малюнок 11.1 (продовження) – Блок-схема алгоритму обробки двовимірних масивів

Скомпонуємо наступну форму і налагодимо її властивості.



Вхідна **Форма** після перейменування об'єктів

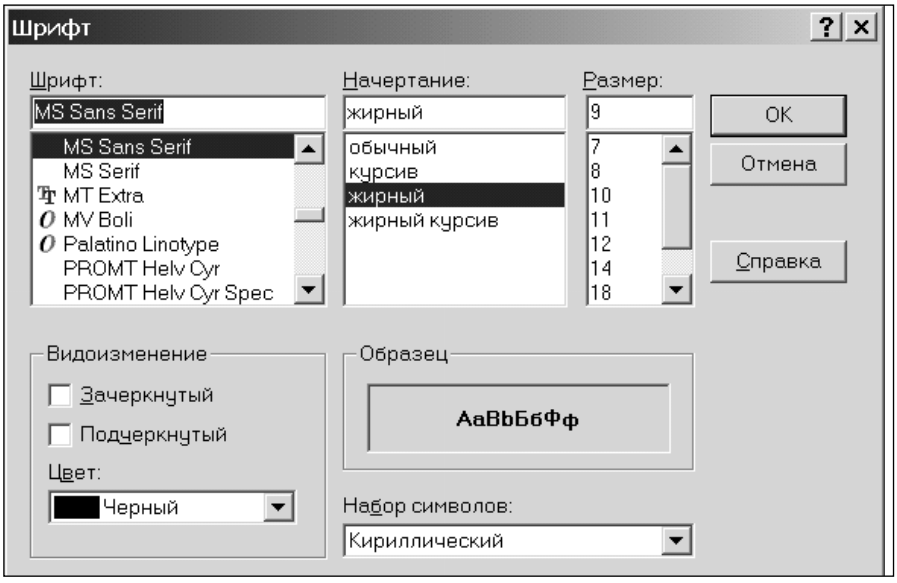


Форма після налагодження властивостей об'єктів

У даній **Формі** застосуємо нові компоненти: GroupBox (3-я кнопка праворуч на сторінці Standard) , RadioGroup (2-я кнопка праворуч на сторінці Standard), StringGrid (4-я кнопка ліворуч на сторінці Additional). Панель GroupBox1 використаємо для зорового об'єднання компонентів, зв'язаних із введенням вхідних даних, а на панелі GroupBox2 розташуємо компоненти, у які виводяться результати. Групу перемикачів (радіокнопок) RadioGroup1 використаємо для вибору способу завдання матриці A – вручну або випадковим чином. Для введення матриці A и виведення вектора B скористаємося компонентами StringGrid1 і StringGrid2 відповідно. Налагодимо властивості цих компонентів. Усі кнопки на формі, а також об'єкти Edit, зв'язані з введенням/виведенням даних, перейменовані зрозумілим чином (тобто змінена їхня властивість **Name**).

Властивості Form1

1. Form1.Caption:=Двовимірні масиви
2. Задамо жирний шрифт розміром 9 пунктів для всіх написів на формі. Для цього клацнемо на формі – для її виділення, потім клацнемо в Інспекторі об'єктів на рядку +Font (TFont). Праворуч від TFont з'явиться багатокрапка: +Font (TFont) ... Клацнемо на цій багатокрапці й установимо параметри, показані в діалоговому вікні «Шрифт»: накреслення «жирний», розмір 9.



Властивості компонентів GroupBox

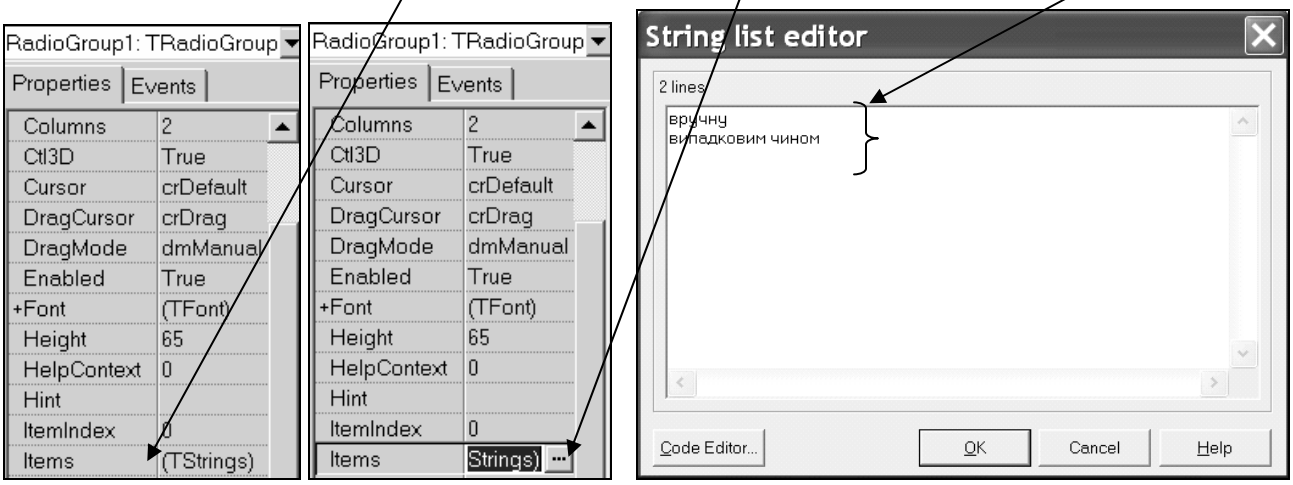
Задамо написи в лівому верхньому куті панелей:

1. GroupBox1.Caption:=Вихідні дані
2. GroupBox2.Caption:=Результат

Властивості компонента RadioGroup

1. RadioGroup1.Caption:=Введення матриці A
2. Задамо написи для 2-х радиокнопок.

Клацаємо на властивості Items, потім на багатокрапці. У вікні редактора рядків, що відкрилося, вводимо написи, що хочемо бачити поруч з радіокнопками, по одному в рядку, після чого натискаємо кнопку ОК..



3. RadioGroup1.Columns:=2 – число стовпців, у яких розташовуються радіокнопки.
4. RadioGroup1.ItemIndex:=0 – буде обрана кнопка «вручну» (за замовчуванням ItemIndex:=-1, тобто жодна радіокнопка не обрана).

Властивості компонента StringGrid_A

Для введення матриці A використовуємо компонент StringGrid1. Перейменуємо його в StringGrid_A:

1. StringGrid1.Name:= StringGrid_A

За замовчуванням, це таблиця, що складається з 5-ти рядків і 5-ти стовпців. 1-й рядок і 1-ий стовпець фіксовані (нерухомі) і виділені обсягом і кольором. Це заголовки таблиці. Тому що нам заголовки не потрібні, то ми обнуляємо число фіксованих стовпців і рядків:

2. StringGrid_A.FixedCols:=0 - число заголовних стовпців
3. StringGrid_A.FixedRows:=0 - число заголовних рядків

Число рядків і стовпців таблиці зберігається відповідно у властивостях RowCount і ColCount. Ці властивості одержать свої значення в програмі після введення розмірності матриці з компонентів Edit_n і Edit_m.

Щоб мати можливість редагувати текст в комірках таблиці і переміщатися по комірках таблиці за допомогою клавіші табуляції, треба двічі клацнути на властивості +Options й установити наступні два режими:

4. goEditing:=True (за замовчуванням установлене goEditing:=False, тобто редагування заборонене);
5. goTabs:=True (за замовчуванням установлене goTabs:=False)

Властивості компонента StringGrid_B

Для виведення вектора B використовуємо компонент StringGrid2. Перейменуємо його в StringGrid_B, видалимо заголовні рядок і стовпець і задамо число стовпців рівним одиниці, оскільки це вектор:

1. StringGrid2.Name:= StringGrid_B
2. StringGrid_B.FixedCols:=0
3. StringGrid_B.FixedRows:=0
4. StringGrid_B.ColCount:=1

1. Процедура обробки клацання на кнопці «Обчислення» (кн. Button_calc)

```
procedure TForm1.Button_calcClick(Sender: TObject);
  const n_max=20;   // максим. число рядків
        m_max=10;   // максим. число стовпчиків
  type  matr=array[1..n_max,1..m_max] of integer;
  type  vect=array[1..n_max] of integer;
  var   n,m:integer;  //n-число рядків, m-число стовпчиків
  var   a:matr;
        b:vect;
        i,j,i0,j0,s:integer;
 begin
     //введення розмірності матриці і налагодження компонентів
StringGrid
     n:=StrToInt(Edit_n.Text);
     m:=StrToInt(Edit_m.Text);
     StringGrid_a.RowCount:=n;
     StringGrid_a.ColCount:=m;
     StringGrid_b.RowCount:=n;
     StringGrid_b.ColCount:=1;
     // введення самої матриці
     if (RadioGroup1.ItemIndex =0)     // вручну
     then for i:=1 to n do
              for j:=1 to m do
                  a[i,j]:=StrToInt(StringGrid_a.Cells[j-1,i-1])
     else  for i:=1 to n do         // заповнення випадковими числами
              for j:=1 to m do
              begin
                 a[i,j]:=Random(101);  // числа від 0 до 100
                 StringGrid_a.Cells[j-1,i-1]:=IntToStr(a[i,j])
              end;
     // заповнення масиву B
     for i:=1 to n do
```

```
     begin
        s:=0;
        for j:=1 to m do
           if (a[i,j] mod 2 <>0 ) then s:=s+a[i,j];
        b[i]:=s;
        StringGrid_b.Cells[0,i-1]:=IntToStr(s)
     end;
     // пошук непарного елемента в нижній правій позиції
        for i:=n downto 1 do
        for j:=m downto 1 do
           if  (a[i,j] mod 2 <>0 ) then
           begin
              i0:=i;
              j0:=j;
              Edit_j0.Text:=IntToStr(j0);
              Edit_i0.Text:=IntToStr(i0);
              exit    // вихід із процедури
           end;
       ShowMessage('Матриця не містить непарних елементів')
 end;
```

2. Процедура обробки клацання на кнопці «Очищення» (кн. Button_clear)

```
procedure TForm1.Button_clearClick(Sender: TObject);
var i,j:integer;
begin
  Edit_n.Clear ;
  Edit_m.Clear ;
  Edit_i0.Clear ;
  Edit_j0.Clear ;
  with StringGrid_a do
  for i:=0 to  RowCount-1 do
      for j:=0 to ColCount-1 do
         Cells[j,i]:='';
  with StringGrid_b do
  for i:=0 to RowCount-1 do
         Cells[0,i]:='';
end;
```

3. Процедура обробки події OnCreate (створення) Форми

(для входу в редактор коду процедури треба двічі клацнути на формі)

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Randomize  //для генерації нових випадкових чисел при запуску додатка
end;
```

ПОЯСНЕННЯ

Компоненти GroupBox і RadioGroup.

Панель **GroupBox** (3-а кнопка  справа на сторінці **Standard**) використовується для зорового об'єднання функціонально зв'язаних між собою елементів. Наприклад, початкові дані ми розмістили на **GroupBox1,** а результати обчислень **–** на **GroupBox2.** Тепер можна по-різному набудовувати властивості вхідних і вихідних даних – за допомогою відповідного **GroupBox.** Властивість **Caption –** напис в лівому верхньому кутку панелі.

RadioGroup (2-а кнопка  справа на сторінці Standard) – панель перемикачів (у Delphi вони називаються радіокнопками). Використовується для об'єднання групи радіокнопок, які можна мати в своєму розпорядженні стовпці або рядки. У таблиці 11.2 перераховані найбільш важливі властивості компоненту.

Таблиця 11.2. Найбільш важливі властивості RadioGroup

Властивість	**Опис**
Caption	Напис в лівому верхньому кутку панелі
Items	Список написів кнопок, що мають строкового типа (аналогічно, списку Lines компоненту Memo). Клацнувши на багатокрапці праворуч від списку Items, потрапляємо в редактор списку рядків. Сюди заносимо написи, які хочемо бачити біля кнопок, по одній в рядку. Скільки запишемо строчок, стільки буде і кнопок. Це масив рядків типа **string.** Індекси починаються від нуля. **Items[0]** – це 1-й рядок списку
Text	Містить всі рядки списку у вигляді одного рядка
Columns	Число стовпців, в яких розташовуються радіокнопки. За умовчанням Columns=1, тобто радіокнопки розміщуються один під одним. Якщо задати, наприклад, Columns=2, то кнопки розташовуватимуться в 2 стовпці
Count	Містить число рядків списку
Capacity	Число рядків, яке може містити список
ItemIndex	Показує індекс вибраної кнопки. Перша кнопка має індекс = 0. За умовчанням ItemIndex = -1, тобто жодна кнопка не вибрана

Компонент **RadioGroup** використовується, якщо написи кнопок мають приблизно однакову довжину і число кнопок в кожному стовпці однаково. Якщо бажане нерегулярне розташування кнопок, то слід використовувати компоненти **RadioButton**, згруповані панеллю **GroupBox.**

Оператор with. Використовується для скорочення запису при зверненні до властивостей і методів об'єкту.

Синтаксис оператору:
with об'єкт **do** оператор;

Приклад. Замість наступного фрагмента програми:

```
Form1.GroupBox1.StringGrid1.FixedCols:=0;
Form1.GroupBox1.StringGrid1.FixedRows:=0;
Form1.GroupBox1.StringGrid1.Cells[0,1]:=’Дата’;
```

можна написати коротше, уникаючи повторні заслання на об'єкт:

```
with Form1.GroupBox1.StringGrid1 do
begin
    FixedCols:=0;
    FixedRows:=0;
    Cells[0,1]:=’Дата’;
end;
```

Генерація випадкових чисел для заповнення масиву

Комп'ютер генерує не випадкові числа, а псевдовипадкові. При цьому задається деяке початкове число, до якого застосовуються перетворення. Псевдовипадкові числа відрізняються від випадкових наступними двома властивостями:

1. при однаковому початковому числі кожного разу генерується одна і та ж послідовність чисел;
2. псевдовипадкові числа повторюватимуться з деяким періодом (хоча і дуже великим). Якщо кількість випадкових чисел більше періоду, то вони зачнуть повторюватися, а такі числа не можна вважати за випадкові.

У модулі **System**, підключеному в пропозиції **uses,** оголошена функція **Random**, яка генерує псевдовипадкові числа, рівномірно розподілені в деякому діапазоні.

Існує два способи звернення до функції **Random**:

без параметра	з параметром Range типа integer
функція повертає випадкові дійсні числа, рівномірно розподілені в інтервалі $0 \le x < 1$	функція повертає випадкові цілі числа, рівномірно розподілені в інтервалі $0 \le x < Range$

Приклад 1. Заповнити масив A випадковими дійсними числами від 0 до 1.

```
for i:=Low(A) to High(A) do
    A[i]:=Random;
```

Тут функції Low(A) і High(A) повертають відповідно нижній і верхній кордони індексу масиву A.

Приклад 2. Заповнити масив A випадковими дійсними числами від 50 до 150. Для цього досить зрушити початкове значення чисел на 50 і помножити числа, що генеруються, на довжину інтервалу: (150-50):

```
var A1,A2:real;
. . . . . .
A1:=50;
A2:=150;
for i:=Low(A)  to  High(A) do
        A[i]:=A1 + (A2-a1)*Random;
 //масив А заповнюється випадковими числами  50≤x<150.
```

Приклад 3. Заповнити масив В випадковими цілими числами від 0 до 100.

```
for i:=Low(B)  to  High(B) do
        У[i]:=Random(101);
```

Приклад 4. Заповнити масив В випадковими цілими числами, рівномірно розподіленими від 100 до 200.

```
for i:=Low(B)  to  High(B) do
        B[i]:=100+Random(101);
```

Оскільки генеруються псевдовипадкові числа, то при черговому запуску програми вироблятиметься одна і та ж послідовність. Це зручно тільки при відладці. Щоб цього не відбувалося, потрібне генератору випадкових чисел задавати нове випадкове число. Це робить функція **Randomize**, яку досить вставити де-небудь в програмі, наприклад, в процедуру обробки події **OnCreate Форми.** Для цього треба двічі клацнути на **Формі** і записати:

```
procedure Tform1.FormCreate(   );
begin
  Randomize
end;
```

СПИСОК ЛІТЕРАТУРИ

1. Архангельский А.Я. Язык Pascal и основы программирования в Delphi. Учебное пособие – М.:ООО "Бином-Пресс", 2004 г. – 496 с.
2. Культин Н. Программирование в Turbo Pascal 7.0 и Delphi. – СПб.: БХВ-Петербург. – 2003. – 416 с.
3. Культин Н.Б. Delphi в задачах и примерах. – СПб.:БХИ-Петербург, 2004. – 288с.
4. Ремнев А.А., Федотова С.В. Курс Delphi для начинающих. Полигон нестандартных задач. – м.:СОЛОН-Прес, 2006. – 360с.
5. Методические указания и задания к лабораторным работам по алгоритмизации и программированию в среде Delphi (для студентов всех специальностей) / О.М.Копытова, Н. К. Шатохина. - Донецк: ДонНТУ, 2007 .- 84 с.