ОСНОВИ АЛГОРИТМІЗАЦІЇ І ПРОГРАМУВАННЯ

АЛГОРИТМИ

Досвід практичної діяльності дозволив людині виробити спосіб розв'язування складних задач, який називається *алгоритмічним.* Цей спосіб полягає в тому, що складний процес розв'язування задачі поділяється на декілька етапів, кожному з яких відповідає дія, виконання якої не становить труднощів. Якщо виконавець процесу розв'язування певної задачі (людина, комп'ютер, робот тощо) відомий, то всі дії, що відповідають етапам — складовим цього процесу, можна подати як конкретні команди. Виконавець обов'язково повинен їх правильно розуміти (у прямому чи переносному смислі) та вміти виконувати. У такому разі кажуть, що розв'язування задачі зводиться до виконання певного алгоритму.

Алгоритм — це скінченна послідовність вказівок (команд), формальне виконання яких дозволяє за обмежений час отримати розв'язок задачі.

Інакше кажучи, алгоритм — це певна інструкція для виконавця, яка може бути задана різними способами — словами, формулами, послідовністю обчислювальних операцій чи логічних дій тощо. Але не кожна інструкція може бути алгоритмом. Алгоритм повинен відповідати певним вимогам, мати такі властивості.

1. Масовість. Алгоритм має бути придатним для багатьох задач, що належать до певного класу.

2. Визначеність (детермінованість). Ця властивість означає, що кожна команда не повинна допускати двоякого тлумачення. Кожний крок алгоритму повинен бути точно визначеним.

3. Дискретність. Процес, який визначається алгоритмом, повинен мати дискретний (перервний) характер, тобто являти собою послідовність окремих завершених кроків — команд або дій.

4. Результативність. Результативність означає, що кожна

дія повинна приводити до цілком певного результату.

5. Формальність. Будь-який виконавець, здатний сприймати та виконувати вказівки алгоритму (навіть не розуміючи їх змісту), діючи за алгоритмом, може виконати поставлене завдання.

6. Скінченність. Діючи за алгоритмом, виконавець одержує розв'язок задачі за скінченну кількість кроків.

Досконалим виконавцем алгоритмів обробки інформації є комп'ютер, робота якого здійснюється під управлінням програми. Програма — це алгоритм, команди якого «зрозумілі» комп'ютеру і можуть бути ним виконані. Колений тип комп'ютера має свій власний набір операцій, із яких можна складати програми. Це так званий набір машинних команд комп'ютера, що визначає обмеження на розробку програм. Тому створення програм являє собою творчий процес, коли потрібно «навчити» комп'ютер розв'язувати різноманітні, як правило складні задачі, використовуючи обмежений набір простих операцій (команд). Разом з тим набір операцій кожного із сучасних комп'ютерів є алгоритмічне повним. Тобто, з цих простих комп'ютерних операцій, як із цеглинок, можна складати алгоритми для розв'язування задач будь-якої складності.

Алгоритм описується засобами мови, зрозумілої виконавцю. Для людини — це природна мова. Комп'ютер виконує операції з багаторозрядними числами у двійковій системі числення, тобто мовою комп'ютера є послідовність кодів із нулів і одиниць. Використання такої комп'ютерної мови для складання програм дуже неефективне. Записи програм, отримані таким чином, виявляються дуже докладними та громіздкими, процес розробки програм — програмування — складним і трудомістким. Тому для створення програм використовуються спеціальні мови — так звані мови програмування. Мова програмування дозволяє записувати команди в такій формі, щоб їх можна було замінити на машинні коди. Це перетворення автоматично здійснюється за допомогою спеціальних програм-перекладачів, які називаються трансляторами. Транслятори є складовою частиною системного програмного забезпечення

комп'ютера. Чим досконалішими стають комп'ютери, тим більше мови програмування за своїми властивостями наближаються до природної мови людини.

При розробці й поданні алгоритмів можуть бути застосовані різні способи їх запису в текстовій та графічній формі. Широкого розповсюдження набув найбільш наочний спосіб зображення алгоритмів у вигляді графічних схем (схем алгоритмів) .

Схема алгоритму складається з елементів двох типів. Перший тип — це графічні фігури (прямокутники, ромби тощо), кожна з яких відображає один з етапів процесу розв'язування задачі і містить у собі текст відповідної команди. При побудові схем алгоритмів здебільшого використовуються такі графічні позначення: овали — для початку й кінця алгоритму, паралелограми — для введення та виведення даних, прямокутники — для обчислень, ромби — для перевірки умов. У прямокутниках зображується також будь-яка результативна команда по перетворенню даних або ситуації. Другий тип елементів — це лінії зі стрілками, що вказують послідовність (порядок) виконання етапів.

У цілому такий спосіб запису алгоритмів можна розглядати як певну алгоритмічну мову — систему позначені і правил для однотипного запису алгоритмів та їх виконана·

Схема алгоритму ілюструється на рис. 34 на прикладі процесу розв'язування квадратного рівняння.

ЗАПИТАННЯ І ЗАВДАННЯ

1. Що таке алгоритм? Назвіть і поясніть властивості алгоритмів.

2. Зобразіть схему алгоритму ділення відрізка пополам за допомогою циркуля й лінійки.

3. Зобразіть схему алгоритму знаходження кількості додатних парних чисел, сума яких дорівнює 46.

4. Зобразіть у вигляді схеми алгоритми обробки текстів у текстовому редакторі.

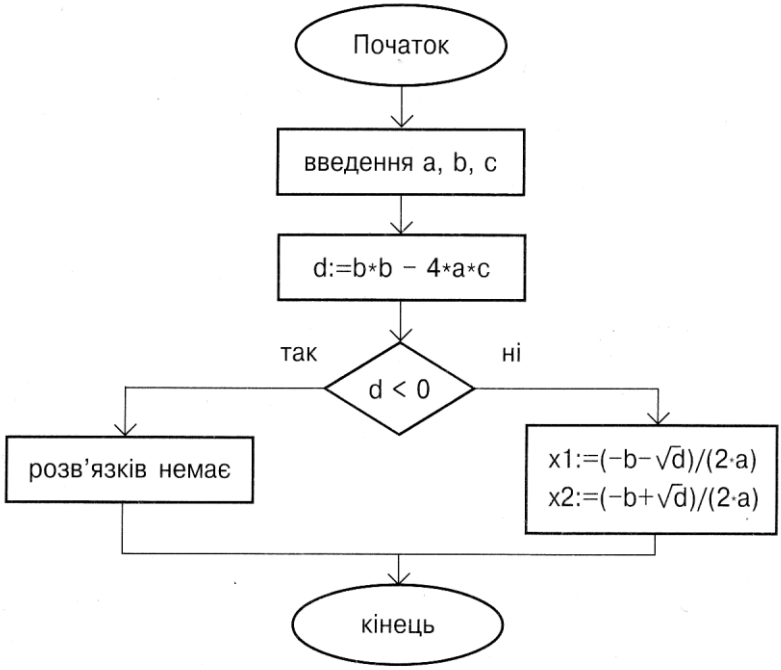


Рис. 34. Приклад схеми алгоритму

2.2. МОВА ПРОГРАМУВАННЯ ПАСКАЛЬ

Одна з найпопулярніших мов програмування — це мова Паскаль, яку створив у 1968р. швейцарський учений Ніклаус Вірт. Вона дозволяє записувати команди, завдяки яким комп'ютер може розв'язувати математичні задачі обробляти тексти, будувати зображення на екрані дисплея.

Як кожна мова, Паскаль має свій алфавіт. До нього входять латинські літери, цифри від 0 до 9, спеціальні знаки (+, — круглі, квадратні і фігурні дужки, крапка, кома та ін.), а також службові слова мови **(begin, end, for,** тощо). У текстах програм службові слова виділяються жирним шрифтом. Для зручності до текстів програм вносяться пояснення (коментарі), які в Паскалі записуються у фігурних дужках.

Одним із основних етапів розроблення програми для розв'язування задачі є присвоєння даним, які використовуються в цій задачі, імен. Ім'я в Паскалі — це слово, яке починається з літери і містить літери, цифри та знаки підкреслення. Як імена не можна використовувати службові слова Паскаля.

Кожне ім'я відповідає певній ділянці пам'яті, куди записуються значення даних. Оскільки на одну й ту саму ділянку можна записати різні значення, то ім'я також називають змінною або ім'ям змінної. Ділянка — поняття умовне: це послідовність різної кількості байтів пам'яті для різних даних. Для кожної змінної треба вказати її тип, щоб вказати транслятору (програмі, яка перекладає з мови програмування на мову машинних команд), скільки місця в пам'яті потрібно виділити для цієї змінної.

У Паскалі розрізняють цілі і дійсні числа, їм відповідають змінні цілого та дійсного типів. У Паскалі для цілих чисел в пам'яті комп'ютера відводиться два байти, а для дійсних — шість.

Для додатних чисел знак « + » можна не писати. При записі дійсних чисел ціла частина відокремлюється від дробової не комою, а крапкою. Дійсні числа можуть записуватись у двох формах: із фіксованою крапкою, наприклад 7.23, 897.5, — 0.11, та в експоненціальній (з плаваючою крапкою), наприклад 1.354E + 12. Літера Е означає множення на степінь 10. Таким чином, запис — 4.9876543234E —02 означає те саме, що й – 0,049876543234. Незалежно від форми запису дійсні числа зберігаються в пам'яті машини у формі з плаваючою крапкою.

Для роботи з числами використовуються шість операцій: « + » — додавання, «-» — віднімання, «/» — ділення, «*» — множення, «div» — ділення націло і «mod» — знаходження залишку від ділення. Останні дві операції застосовуються тільки до цілих чисел.

З імен змінних, чисел, знаків арифметичних операцій і функцій складаються арифметичні вирази. Щоб указати порядок дій, використовують тільки круглі дужки; їх може бути кілька, але кількість відкритих дужок повинна дорівнювати кількості

закритих.

До Паскаля вбудовані засоби розрахунку основних математичних функцій. Запис у Паскалі синуса, косинуса та натурального логарифма збігаються із загальноприйнятими: sin(x), cos(x), ln(x). Піднесення аргументу до квадрата позначається sqr(x), добування квадратного кореня sqrt(x), а модуль — abs(x). Аргументом функції може бути арифметичний вираз.

Приклад арифметичного виразу:

математичний запис: $\dfrac{x}{1+\dfrac{x^2}{5+x^3}}$

запис на Паскалі: x / (1 + sqr(x) / (5 + x*sqr(x)) Алгоритм перетворення даних на Паскалі складається з операторів, що є головними структурними елементами програм. Кожний оператор перетворюється транслятором у послідовність машинних команд.

Оператор присвоювання призначений для надання змінній нового значення. Загальний вигляд оператора присвоювання:

Ім'я змінної := арифметичний вираз;

Знак «: = » читається «присвоїти» (надати значення). У кінці запису оператора Паскаля ставиться крапка з комою.

При виконанні оператора присвоювання обчислюється значення арифметичного виразу, що стоїть у правій частині. При цьому з ділянок оперативної пам'яті (з відповідними іменами) зчитуються відповідні значення і над ними виконуються вказані дії. Одержаний результат записується до ділянки пам'яті, ім'я якої вказано зліва від знака присвоювання.

Приклади оператора присвоювання:

x := 3.14; змінній x присвоїти значення 3.14}

a := b + c; {з ділянок b і c зчитуються заздалегідь вміщені туди дані, обчислюється сума, результат записується до ділянки a}

i := i + 1; {значення змінної i збільшується на одиницю}

Для типів змінної ліворуч і арифметичного виразу праворуч від знака присвоювання існують обмеження:

1) якщо змінна ліворуч — дійсного типу, то арифметичний вираз може бути як цілого, так і дійсного типу, тобто містити або цілі змінні й допустимі для них операції, або дійсні, або і ті, й ті (тоді вираз перетворюється на вираз дійсного типу);

2) якщо змінна ліворуч — цілого типу, то арифметичний вираз може бути тільки цілого типу.

Це означає, що можна, наприклад, дійсній змінній присвоїти ціле значення. У пам'яті комп'ютера воно буде перетворено на дійсний тип. У фігурних дужках поряд з оператором подається коментар його дій.

ЗАПИТАННЯ І ЗАВДАННЯ

1. Хто створив мову програмування Паскаль?
2. Які символи входять до складу алфавіту Паскаля?
3. Для чого в Паскалі розрізняються цілі й дійсні числа?
4. Що таке арифметичний вираз і з чого він може складатися?
5. Як працює оператор присвоювання?
6. Які обмеження накладає Паскаль на типи даних при присвоюванні?
7. Що таке транслятор?
8. Запишіть у вигляді арифметичного виразу квадратний тричлен.

2.3. СТРУКТУРА ПРОГРАМИ НА ПАСКАЛІ. ВВЕДЕННЯ І ВИВЕДЕННЯ ДАНИХ

Програма на Паскалі починається із заголовка, далі розміщується описова частина, в якій визначаються дані, що використовуються в програмі, а після цього — тіло програми або програмний блок (блок операторів), що містить оператори для перетворення даних.

Загальний вигляд програми:

Program їм я програми;

label {список міток};
const {список сталих величин - констант};
type {описи нестандартних типів даних};
var {описи змінних, що використовуються в програмі};
begin {позначення початку програмного блоку}
{програма - послідовність операторів}
end. { кінець програми}

Ім'я програми складається не більше як з 8 символів. Воно починається з літери і містить літери, цифри й знаки підкреслення. Програма починається зі слова **program** і закінчується словом **end** із крапкою. Оператори, заголовок програми, описи типів та змінних закінчуються крапкою з комою.

До описової частини програми входять розділи міток **label,** констант **const,** нестандартних типів даних **type** і змінних **var**. їх використання буде розглянуто далі. Основним із них є розділ змінних **var**. У ньому вказуються імена змінних, що використовуються в програмі, та їх тип. Для числових даних використовуються основні описувачі типів **integer** (цілий) і **real** (дійсний). Наприклад, якщо у програмі використовуються дві цілочислові змінні *i, i* та одна дійсна *x,* то тоді розділ змінних може мати вигляд

var i, j: integer; x: real;

Імена змінних одного типу пишуться через кому, потім після двокрапки вказується їх тип. Опис кожного типу закінчується крапкою з комою. Коли при перекладі на мову машинних кодів транслятор зустрічає опис змінної, він відводить для цієї змінної ділянку пам'яті і ставить у відповідність до імені змінної адресу першого байта ділянки.

Програмний блок містить опис алгоритму розв'язування задачі.

Для введення даних у комп'ютер і виведення їх використовуються відповідно оператори введення і виведення.

Оператор введення поміщає значення змінної, яка вводиться, у відповідну ділянку пам'яті. Оператор введення має вигляд

read (список імен);

При виконанні оператора **read** (читати) програма зупиняється і чекає доти, доки користувач набере на клавіатурі число і натисне **Enter.** Якщо список введення містить кілька імен, то для кожного треба ввести своє значення. Числа, що вводяться, відокремлюються пропуском чи комою або після кожного натискується **Enter.** Наприклад, оператор

read (i, j);

потребує введення двох цілих чисел. Після роботи цього оператора курсор на екрані дисплея знаходиться в кінці останнього числа і не переходить на новий рядок. Щоб після введення даних курсор на екрані дисплея переходив на новий рядок, треба використовувати оператор

readln (список імен);

Для виведення результатів роботи програми на екран дисплея служить оператор

write (список виведення);

Оператор **write** (писати) виводить дані на екран дисплея. До списку виведення заносять через кому імена змінних або вирази. До списку виведення можуть також; входити взяті в апострофи тексти, наприклад,

write ('x= ', x);

При виконанні цього оператора на екрані буде надруковано текст, що міститься між апострофами і значення змінної *x.* Значення буде виведене у формі дійсного числа з плаваючою крапкою. Щоб число було виведене у формі з фіксованою крапкою, після імені змінної треба вказати два цілих числа, відокремивши їх від імені змінної та одне від одного двокрапками. Перше з цих чисел вказує, скільки позицій займає число (включаючи десяткову крапку і знак числа). Друге число вказує кількість цифр дробової частини числа. Наприклад, для друкування числа — 23.57 як значення змінної *x* оператор друкування слід подати у вигляді

write ('x= ', x:6:2);

На екран буде виведено: *x* = —23.57 Якщо після друкування треба перевести курсор на новий рядок, використовується оператор

writeln (список виведення);

Для переведення курсору на новий рядок використовується порожній оператор виведення

writeln;

Розглянемо приклад. Нехай потрібно обчислити суму, добуток і різницю двох даних чисел. Для кожного з чисел треба придумати ім'я змінної і вказати її тип. Потім ввести ці числа у відповідні ділянки пам’яті і, використовуючи можливість включення до оператора виведення арифметичних виразів, надрукувати результати:

```
program E1;
var a,b: real;
begin
write ('введіть два числа через пробіл, потім натисніть Enter');
readln(a,b);
writeln('a+b =', a+b, 'a*b =', a*b, 'a-b = ', a-b)
end.
```

Службові слова Паскаля виділені жирним шрифтом, на письмі вони звичайно, підкреслюються. При наборі тексту програми на клавіатурі вони ніяк не виділяються, їх розрізняє транслятор. Тому службові слова не можна використовувати як імена. Перший оператор програмного блоку виводить на екран підказ для користувача, що він повинен зробити. При введенні даних рекомендується робити подібні підкази. Слід зауважити, що перед службовим словом end крапку з комою можна не ставити.

При розв'язуванні задач імена присвоюються не тільки вхідним даним, а й результатам і проміжним значенням. Оскільки в розглянутому прикладі треба одержати три

результати, введемо для них імена *x, y, z*. У програмі цим змінним будуть присвоєні значення суми, добутку та різниці двох чисел, що вводяться. Програма має вигляд:

```
program E2;
var a,b,x,y,z: real;
begin
write (' введіть два числа через пробіл, потім натисніть Enter');
readln(a,b);
x:= a + b;
y:= a * b;
z:= a-b;
writeln('a + b =', x, ' a*b =', y, ' a - b =', z)
end.
```

ЗАПИТАННЯ І ЗАВДАННЯ

1. Для чого потрібно описувати дані в програмі?
2. Як описати змінні одного типу, наприклад дійсного?
3. Який оператор використовується для введення даних? Як він працює?
4. Куди потрапляють введені з клавіатури числа при роботі оператора введення?
5. Як перевести курсор на новий рядок після введення даних?
6. Як вивести результати роботи програми на екран дисплея?
7. Як зробити, щоб виведення даних здійснювалось з нового рядка?
8. Напишіть програму обчислення середнього арифметичного двох чисел.
9. Напишіть програму обчислення відстані між двома точками площини.
10. Напишіть програму обчислення площі трикутника за формулою Герона.
11. Напишіть програму обчислення площі бокової поверхні куба.

12. Напишіть програму обчислення площі та гіпотенузи прямокутного трикутника, якщо відомі його катети.

13. Напишіть програму обчислення суми модулів трьох дійсних чисел.

14. Напишіть програму обчислення площі круга, якщо відома довжина кола.

15. Напишіть програму обчислення площі рівностороннього трикутника.

16. Напишіть програму піднесення числа до четвертого степеня за дві операції.

17. Напишіть програму піднесення числа до сьомого степеня за чотири операції.

18. Напишіть програму визначення моменту зустрічі двох автомобілів, якщо відома відстань між пунктами, звідки вони вийшли назустріч один одному одночасно, та їх швидкості.

19. Напишіть програму обчислення суми членів арифметичної прогресії, якщо відомо її початковий член і різниця, а також кількість ЇЇ членів. Вказівка: при роботі на комп'ютері вкажіть різні формати виведення чисел з фіксованою крапкою, виконайте програму для різних даних кілька разів.

2.4. РОБОТА В СИСТЕМІ TURBO PASCAL

Система Turbo Pascal розроблена фірмою Borland для у і комп'ютерів IBM PC. Існує кілька версій цієї системи програмування, що містить транслятор, редактор, різні сервісні функції для роботи з файлами, а також бібліотеки, І які дозволяють будувати зображення, використовувати в \ програмі засоби операційної системи MS-DOS тощо.

Створення програми. Каталог, що містить систему Turbo Pascal, називається TP; після цих літер вказується версія системи, наприклад 5. У цьому каталозі треба знайти файл з іменем turbo.exe, підвести до нього курсор і натиснути **Enter.** При запуску системи з'являється вікно редактора текстів програм (цей редактор можна використовувати і як редактор текстів). Щоб увійти до меню, яке розташоване зверху екрана,

використовується клавіша F10. Переміщення курсора вздовж рядка меню здійснюється клавішами управління курсором. Якщо екран порожній, можна відразу набирати текст програми. При набиранні програм рекомендується робити відступи, як зроблено в цій книзі. Це полегшує читання текстів програм і пошук помилок.

Якщо на екрані після запуску системи знаходиться непотрібна програма, треба ввійти в пункт меню «File» і виконати команду «New». Екран очищається, і зверху з'являється ім'я програмного файла noname.pas (програма без імені). Набір кожного рядка програми завершується натисненням клавіші **Enter.** Курсор можна переміщувати по тексту за допомогою клавіш управління. На початок рядка можна перейти за допомогою клавіші **Home,** в кінець рядка — клавіші End.

Стерти непотрібний рядок можна натисненням комбінації клавіш **Ctrl + Y,** вставити — натисненням Enter (курсор при цьому повинен знаходитись у кінці рядка, після якого здійснюється вставляння). Якщо відбувся випадковий розрив рядка (натисненням клавіші **Enter** у середині рядка), то треба підвести курсор до кінця верхнього рядка і натиснути **Delete.**

Запуск програми. Для виконання програми треба ввійти до меню і в пункті «Run» виконати команду «Run». Система запускає спочатку транслятор, який перекладає програму з Паскаля на мову машинних кодів і шукає синтаксичні помилки в програмі. Якщо вони є, то програма не виконається і відбувається повернення до редактора. Над текстом програми з'являється червоне вікно з повідомленням типу помилки, після натиснення клавіші Esc вікно зникає, курсор встановлюється в рядок з помилкою. Для отримання докладної інформації про помилку треба натиснути **Ctrl + Fl.**

Щоб побачити результати, виведені на екран, потрібно в пункті меню «Windows» (вікна) вибрати підпункт «Output» (вихідне), в результаті чого відкриється вікно виведення. Розмір цього вікна, так само як і інших вікон у системі Turbo Pascal, можна змінювати.

Коли всі помилки виправлено, програма починає виконуватись. Якщо в системі вже є програма з ім'ям noname.pas, то видається вікно, де про це повідомляється. Якщо ім'я залишається без змін, натискується Enter і з'являється ще одне вікно, в якому запитується, чи буде програма з таким іменем записана поверх тієї, що вже є. Якщо користувач із цим погоджується, він натискує **Y** (yes — так). Якщо ім'я програми потрібно змінити, нове ім'я треба ввести в першому вікні, яке з'явиться, стерши noname.pas і записавши нове ім'я. Після цього програма починає виконуватись.

Програму можна модифікувати і виконувати скільки завгодно разів. Щоб помістити у вікно редактора програму, яка знаходиться на диску, треба виконати команду «Load» із пункту меню «File» (або натиснути F3). При цьому з'являється вікно, в якому набирається ім'я файла. Якщо замість цього натиснути **Enter,** то з'явиться список файлів з розширенням *.pas*, із якого можна вибрати потрібний файл.

ЗАПИТАННЯ І ЗАВДАННЯ

1. Що таке Turbo Pascal?
2. Яка послідовність дій при наборі програми на комп'ютері?
3. Як здійснити запуск програми на виконання?

2.5. Умовний оператор

Для запису алгоритмів здебільшого використовуються три типи команд: присвоювання, розгалуження і повторення. Команді розгалуження в Паскалі відповідає умовний оператор.

Умовному оператору відповідають дві структури, подані на рис. 2 і 3. На рис. 2 показано неповну форму умовного оператора, коли дія здійснюється тільки тоді, коли викопується записана в ромбі умова. У разі невиконання умови відбувається перехід до наступного оператора (вихід із структури).

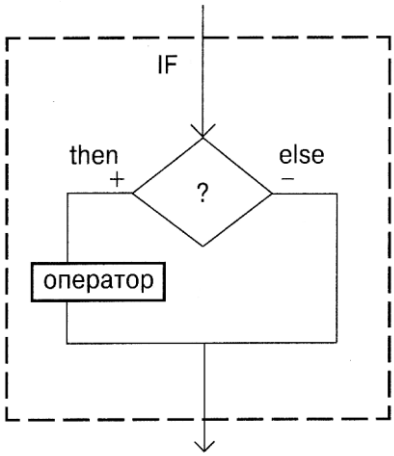


Рис. 2. Неповна форма умовного оператора

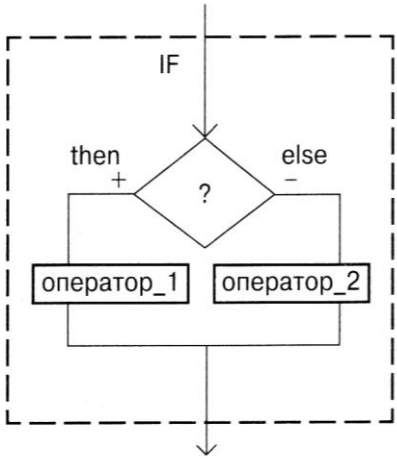


Рис. 3. Повна форма умовного оператора

На рис. 3 зображено повну форму умовного оператора: при виконанні умови (вихід « + ») виконується одна дія, при невиконанні (вихід « —») — інша дія. Кожна структура має один вхід і один вихід. Програму рекомендується будувати з послідовних, логічно завершених блоків. При цьому не допускається передача управління між блоками інакше як через вихід з одного блока та через вхід до іншого. У побудованій таким чином програмі буде менше помилок при розробці і її легше перевіряти на правильність виконання.

Неповний умовний оператор має вигляд

if умова **then** оператор;

Повний умовний оператор записується у вигляді

if умова **then** оператор_1 **else** оператор_2;

Якщо після слів **then i else** треба записати не один оператор, а кілька, тоді ці оператори беруться в так звані операторні дужки: відкриваюча дужка — **begin,** і закриваюча — end:

begin
{оператори}
end;

Перед словом **else** крапка з комою не ставиться. Кожну пару **begin...end** записують в одному стовпці: так легше перевірити наявність для кожної відкриваючої операторної дужки відповідної закриваючої.

Приклади умовного оператора:

```
if a < b then y := x;
if x < 0 then x := –x;   {зміна знака змінної x}
if a + b < c then
begin
z:=x;{обмін значеннями змінних x і y }
x:=y;
y:=z
```

В умовному операторі може бути інший умовний оператор. Наприклад:

```
if sqr(x) + sqr(y) > 1 then
if x > y then z := 0
else z := 1;
```

При такій формі запису, яка використовує зсув праворуч для кожної внутрішньої дії, легко зрозуміти, до якого з двох слів if відноситься слово **else.** Якщо цей оператор записати в один рядок, то .відповідь буде неоднозначною. Транслятор працює таким чином. Зустрівши складну конструкцію з вкладених умовних операторів, він аналізує її з кінця, приписуючи останнє знайдене **else** першому зустрінутому при перегляді справа наліво if.

Розглянемо приклад програми з використанням умовного оператора. Нехай для двох цілих чисел треба визначити, якими вони є — парними чи непарними. Для перевірки парності використовуємо умову: залишок від ділення парного числа на два дорівнює нулю. Програма матиме вигляд:

```
program E3;
var a,b: integer;
begin
writeln (' введіть два цілих числа');
readln(a,b);
if a mod 2 = 0 then writeln ('a - парне')
```

```
else writeln ('a - непарне');
if b mod 2 = 0 then writeln ('b - парне')
else writeln ('b - непарне')
end.
```

Логічні вирази. Подана на рис. 34 схема алгоритму розв'язування квадратного рівняння містить перевірку умови $d < 0$. Два значення d *і* О зв'язані відношенням < (менше). Якщо умова виконується, то кажуть, що відповідний вираз істинний; якщо умова не виконується, то вираз хибний. Тут ідеться про логічний вираз. Для побудови складних умов у Паскалі існують логічні операції **and** (і), or (або) і not (ні). Позначивши істинне значення 1 і хибне 0, побудуємо таблиці істинності для цих логічних операцій:

Таблиця 5

X	Y	X and Y
1	1	1
1	0	0
0	1	0
0	0	0

Таблиця 6

X	Y	X or Y
1	1	1
1	0	1
0	1	1
0	0	0

Таблиця 7

X	not X
1	0
0	1

Розглянемо приклади побудови складних логічних виразів.

1. Нехай потрібно визначити, чи належить точка з координатою x відрізку [a; b]. Якщо записати цю умову подвійною нерівністю, то ця нерівність читається так: x менше або дорівнює b і більше або дорівнює a ($a \le x \le b$). Відношення «менше або дорівнює» у Паскалі записується двома знаками.

Аналогічно записується і «більше або дорівнює». Однак у Паскалі не можна записувати подвійну нерівність. Використовуючи логічну операцію «і», запишемо:

```
(x >= a) and (x <= b)
```

Відношення, між якими стоїть логічна операція, беруться в круглі дужки.

2. Нехай є прямокутний отвір із сторонами a і b і цеглина з ребрами x, y, z. Потрібно скласти умову входження цеглини в

отвір (рис. 37).

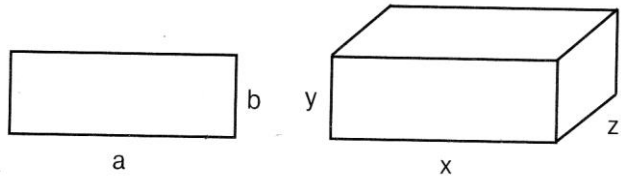


Рис. 37. «Отвір» і «цеглина»

Цеглина пройде в прямокутний отвір, якщо буде виконана така складна умова:

(x <= a) **and** (y <= b) **or**
(y <= a) **and** (x <= b) **or**
(x <= a) **and** (z <= b) **or**
(z <= a) **and** (x <= b) **or**
(y <= a) **and** (z <= b) **or**
(z <= a) **and** (y <= b)

Для трьох граней одержано шість умов тому, що кожну грань можна повернути на 90° і перевірити для кожної грані два випадки.

3. Нехай потрібно визначити, чи належить точка трикутнику *ABC* (рис. 38). Сторони трикутника є відрізками прямих, кожна з яких ділить площину на дві частини. Для кожної прямої визначимо півплощину, в якій знаходиться трикутник. Півплощина, яка знаходиться вище осі *x,* осі *x,* визначається нерівністю y>0. Півплощина, яка знаходиться справа від прямої і з'єднує точки (— 1,0) та (0,2), задається нерівністю y— *2x* — 2<0. Півплощина, яка знаходиться зліва від прямої і з'єднує точки (1,0) та (0,2), задається нерівністю *y+2x* — 2<0. Отже умова належності точки *(x,y)* фігурі має такий вигляд:

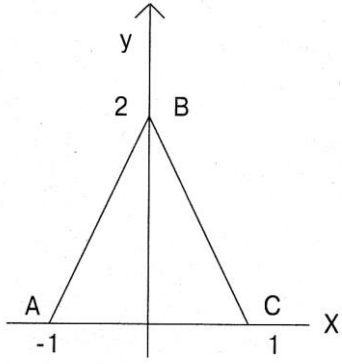


Рис. 38

(y > 0) and (y - 2*y - 2 < 0) and (y + 2*x - 2 < 0)

4. Наведемо приклад програми визначення існування трикутника із сторонами a, Ь і *c.* Умова існування трикутника відома з геометрії: сума двох будь-яких сторін повинна бути більша третьої. Отже, для всіх сторін умова «сума двох більше третьої» повинна виконуватись. Програма матиме вигляд:

```
program E4;
var a,b,c: real;
begin
writeln (‘введіть довжини трьох сторін трикутника ');
readln(a,b,c);
write('трикутник із сторонами ', a, b, c);
if (a + b > c) and (b + c > a) and (c + a > b)
then writeln (‘існує’)
else writeln (‘не існує’)
end.
```

ЗАПИТАННЯ І ЗАВДАННЯ

1. Як транслятор аналізує вкладені умовні оператори?
2. Як виконується неповний умовний оператор?
3. Як перевірити, чи є ціле число непарним?
4. Як виконуються логічні операції «і», «або», «ні»?
5. Напишіть програми на Паскалі для розв'язування таких задач:

а) Дано три числа a, b і c. З'ясуйте, чи має місце $a<b<c$. Відповідь подати у вигляді тексту: «так» чи «ні».

б) Дано додатні числа *a, b, c, x.* З'ясуйте, чи пройде цеглина з ребрами a, Ь, c у квадратний отвір із стороною x.

6. З'ясуйте, чи належать числа *a* і *b* проміжку (-1;1).
7. Присвоїти *z* значення більшого з чисел x і y у тому разі, якщо x < 0, і меншого, якщо x > O,
8. Дано три дійсних числа. Вибрати ті з них, які належать відрізку [1;3].
9. Присвоїти змінній a значення найбільшого з трьох заданих чисел.

10. Дано два числа. Вивести перше з них, якщо воно більше другого, і обидва числа, якщо це не так.

11. Перевірити, чи є серед трьох заданих чисел рівні.

12. Дано два дійсних числа. Менше з них замінити півсумою цих чисел, а більше - їх добутком.

13. Обчисліть найменше з трьох заданих чисел.

14. Знайти розв'язок рівняння ax + b = 0, якщо він існує.

15. Якщо дане число *x* менше нуля, то z присвоїти значення більшого з двох чисел x і y, в противному разі - z присвоїти значення півсуми цих чисел.

16. Дано три дійсних числа. Знайти найбільші значення їх попарних сум і добутків.

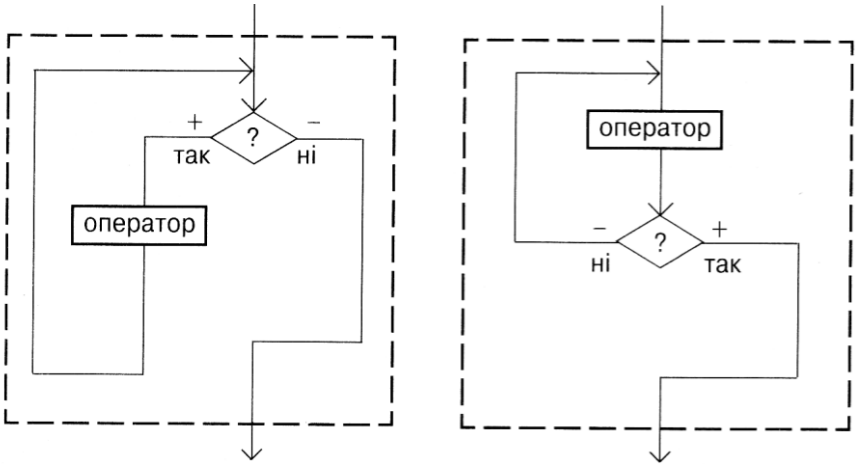
17. Дано дійсні числа a, *b* і *c*. Подвоїти ці числа, якщо вони впорядковані за зростанням.

2.6. ОРГАНІЗАЦІЯ ЦИКЛІВ

Циклом у програмуванні називається повторення одних і тих самих дій. Цикл повинен закінчуватись за якої-небудь умови. Перевіряти цю умову можна на початку кожного повторення, і в цьому разі цикл називається «доки». При перевірці умови в кінці коленого повторення цикл називається «до». Схеми цих двох типів циклів наведені на рис. 39.

У циклі «доки» спочатку перевіряється умова і якщо вона виконується, тобто логічний вираз істинний, то виконується оператор і знову перевіряється умова. Записана в циклі «доки» умова є умовою продовження циклу. Як тільки вона перестане виконуватись, цикл завершиться. На рис. 38 вихід із ромба « + » (або «так») означає виконання умови повторення циклу, « — » (або «ні») — невиконання. Цикл «доки» не виконається жодного разу, якщо умова при вході в структуру відразу виявиться хибною. Таким чином, цикл «доки» містить умову продовження повторення, а цикл «до» — умову закінчення повторення. Обидві структури мають один вхід і один вихід. Однак цикл «до» завжди виконується хоча б один раз, тому що умова перевіряється після виконання операторів, що входять до циклу.

Це ускладнює перевірку правильності програми, тому краще використовувати цикл «доки». Оператор у циклі може бути простим або складеним, тобто являти собою групу операторів, обмежених операторними дужками **begin...end.** Його називають тілом циклу. Цикли можна організовувати, використовуючи різні засоби мови Паскаль.



Цикл «доки» Цикл «до»

Рис. 38. Циклічні структури

Оператор безумовного переходу. Цей оператор дозволяє перейти без перевірки умови або на один з попередніх операторів, або на один із наступних, тобто змінити порядок виконання команд. Перед оператором, на який здійснюється безумовний перехід, потрібно поставити мітку — ціле число, що складається не більше ніж. із чотирьох цифр. Від оператора мітка відділяється двокрапкою.

Перехід до поміченого оператора здійснюється за допомогою оператора безумовного переходу, який має вигляд

goto n;

до *n* — мітка. Мітка повинна бути визначена в описовій частині програми в розділі **label.**

Організація циклів за допомогою операторів умовного і

безумовного переходів. Нехай потрібно знайти найбільший спільний дільник двох натуральних чисел *A і B,* Скористаємось алгоритмом Евкліда: будемо зменшувати кожний раз більше з чисел величину меншого доти, доки обидва значення не стануть рівними. У таблиці 8 подано алгоритм Евкліда.

Таблиця 8

Вхідні дані	Перший крок	Другий крок	Третій крок	НСД(A,B)=5
A = 25	A- 10	A= 10	A = 5	
B = 15	B = 15	B = 5	B = 5	

Програма матиме вигляд:

```
program E5;
label 1,2;
var a,b: integer;
begin
  write('введіть два натуральних числа');
  readln(a,b);
  1:if a = b then goto 2;
  if a > b then a := a - b
  else b := b - a;
  goto 1;
  2:writeln ('НСД=', a)
end.
```

Оператор циклу «доки». Як видно з попереднього прикладу, циклічний процес можна організувати без використання спеціальних операторів. Однак при складанні досить серйозних програм використовувати оператор безумовного переходу не рекомендується, оскільки можна швидко заплутатись при перевірці програми. Оператор циклу «доки» має вигляд:

while умова do оператор;

і виконується таким чином: оператор (тіло циклу) повторюється доти, доки виконується умова (істинний логічний вираз). Оператор може бути простим або складеним, вміщеним в операторні дужки **begin...end.**

Для алгоритму Евкліда програма набуде вигляду:

```
program E6;
var a,b:integer;
begin
  write('введіть два натуральних числа');
  readln(a.b);
  while a<>b do
  if a>b then a := a-b
  else b := b-a; writeln
  ('НСД=', a)
end.
```

Оператор циклу «до». Перевірка умови в циклі «до» здійснюється після виконання оператора. Якщо умова в циклі «доки» є умовою продовження повторень, то умова її циклі «до» — умовою виходу з циклу, його завершенням. Тому для тієї самої задачі ці умови протилежні.

Загальний вигляд оператора:

repeat оператор **until** умова;

Між словами **repeat** (повторити) і **until** (до того часу, поки) можна записати будь-яку кількість операторів без використання операторних дужок. Перед словом **until** не ставиться крапка з комою.

Програма знаходження найбільшого спільного дільника набуде такого вигляду:

```
program E7;
var a,b:integer;
begin
  write('введіть два натуральних числа');
  readln(a,b);
  repeat
```

```
  if a > b then a := a - b;
  if b > a then b := b - a until a = b;
  writeln('НСД=', a)
end.
```

Оператори циклів «перелік». При виконанні програм знаходження найбільшого спільного дільника кількість повторень неоднакова для різних даних. Коли відоме число повторень, зручно використовувати цикл «перелік». У Паскалі с, два оператори для організації циклів — прямий і зворотний «перелік». Прямий перелік іде від відомого меншого числа до відомого більшого, на кожному кроці додається одиниця (наприклад, від 120 до 140: 121, 122, 123, ..., 139, 140). Оператор прямого переліку

for i:=n1 **to** n2 **do** оператор;

читається як «для і починаючи з n1 до n2 виконати оператор».

Змінна і називається змінною циклу, яка при прямому переліку завжди змінюється від меншого значення до більшого. Треба звернути увагу, що для виконання оператора пі повинно бути не більше ніж n2 (n1 ≤ n2). При *i* = n! цикл виконується перший раз. Потім до значення змінної /додається одиниця і здійснюється перевірка, чи не стало отримане значення більшим ніж n2. Якщо і + 1 ≤ n2, то оператор виконується, а якщо і + 1 > n2, то відбувається вихід із циклу і виконується оператор програми, який слідує за оператором циклу. Оскільки оператор циклу for сам змінює значення змінної циклу, то її не можна змінювати іншим способом, наприклад присвоюванням їй якого-небудь значення в тілі циклу (вона не повинна з'явитися зліва від знака : =).

Оператор у циклі може бути простим або складеним, взятим в операторні дужки.

Розглянемо приклади використання операторів циклу.

1. Обчислення добутку a^n. Відомо, що для того щоб

отримати цілий степінь n числа a, потрібно помножити його самого на себе n разів. Цей добуток при виконанні програми буде зберігатися в ділянці пам'яті з іменем *p.* Щоразу, при черговому циклі, з цієї ділянки буде зчитуватись попередній результат, помножуватись на основу степеня *a i* знову записуватись у ділянку p. Основний оператор у тілі циклу повторюється л разів і має вигляд:

p := p*a;

При першому виконанні циклу в ділянці *p* повинно знаходитись число, що не впливає на множення, тобто перед циклом у цю ділянку треба записати одиницю.

Програма має вигляд:

```
program E8;
var a,p: real;  i,n: integer;
begin
  write('введіть a - основу степеня, a = ');
  readln (a);
  write('введіть ціле n - показник степеня, n = ');
  readln(n);
  p:=1;
  for i:=1 to n do
  p := p * a;
  writeln('p = ', p)
end.
```

У таблиці 9 відображено протокол виконання програми при піднесенні числа 2 до п'ятого степеня: a = 2, n = 5.

Таблиця 9

i		1	2	3	4	5
p	1	2	4	8	16	32

Протокол (таблиця), заповнений вручну, використовується для перевірки всіх етапів роботи програми — тестування. Для перевірки роботи програм, алгоритми яких реалізують методи розв'язування складних задач, використовують контрольні

приклади (тести). Програма виконується комп'ютером і звіряються всі проміжні дані, отримані при обчисленні, і кінцеві результати. Для розуміння роботи програми і виконання окремих операторів потрібно заповнювати подібні протоколи.

2. Обчислення p = n! (n факторіал).

За визначенням n! = 1*2*3*…n. Використовуючи попередню програму, обчислимо *p* як добуток чисел від 1 до n, тобто *p* кожний раз множиться не на одне й те саме число, а на значення змінної циклу. Програма матиме такий вигляд:

```
program E9;
var p,i,n:
integer;
begin
  write('введіть ціле n=');
  readin(n);
  p:=1;
  for i := 1 to n do
  p := p * i;
  writeln(n, '!=', p)
end.
```

3. Складання таблиці значень функції y=sin x. Нехай треба скласти таблицю значень функції y=sin *x* на відрізку [0; 3.14] з кроком 0.1. Щоб не визначати кількість повторень обчислень, можна скористатися циклом «доки». Використовуючи виведення дійсних чисел із фіксованою крапкою, визначимо, що кількість цифр після крапки в дробовій частині значення функції дорівнюватиме п'яти. Тоді подання всього числа, враховуючи область значень синуса, займе сім позицій (числа додатні, тому додається позиція для десяткової точки і цілої частини числа).

Програма матиме вигляд

```
program E10;
var x,y: real;
begin
```

```
  x:= 0;
  writeln('x':10,'sin x':10);
  while x <= 3.14 do
        begin
        y:= sin(x);
        writeln(x:10, '    ',y:7:5);
        x:= x + 0.1
        end
end.
```

При кожному виконанні циклу спочатку перевіряється умова x < = 3.14, потім обчислюється значення функції, друкується аргумент *x* (для нього відведено десять позицій, із них одна — для цифри дробової частини) і, через три пропуски — значення функції, і нарешті, для наступного кроку циклу обчислюється нове значення аргументу (x збільшиться на 0.1). Цикл «доки» дозволяє змінювати змінну циклу як завгодно, збільшуючи її або зменшуючи на будь-яке значення.

4. Додавання чисел. При додаванні кількох чисел необхідно накопичувати результат у певній ділянці пам'яті, щоразу зчитуючи з цієї ділянки попереднє значення суми і додаючи до нього наступний доданок. Нехай відомо, що будуть додаватися n чисел. У такому випадку треба n разів виконати дію

s := s + a;

де a — наступне число, що вводиться з клавіатури. Для першого виконання цього оператора присвоювання потрібно з ділянки з іменем *s* взяти таке число, яке не впливало б на результат додавання. Отже, перед початком циклу слід присвоїти змінній *s* значення нуль. Програма має вигляд

```
program E11;
var a,s: real; i,n: integer;
begin
  write ('введіть кількість доданків n=');
  readln(n); s:=0;
  for i:= 1 to n do
```

```
  begin
        write(i,'- е число =');
        readln(a);
        s := s + a
        end;
  writeln('сума s =',s)
end.
```

Якщо кількість чисел невідома, то можна задати число-обмежувач, наприклад нуль. У такому разі використовується цикл **while або repeat:**

```
s := 0;  readln (a);
while a <> 0 do
  begin s := s + a;
        readln(a)
end;
```

```
s:=0;
repeat
  readln (a);
    s := s + a
until a = 0;
```

Оператор циклу зворотного переліку працює аналогічно оператору циклу прямого переліку, але змінна циклу не збільшується на одиницю з кожним кроком, а зменшується. Цей оператор має вигляд

for i := n2 downto n1 **do** оператор;

Для цього оператора повинна також виконуватись умова n2 > n1.

При використанні в програмі операторів циклу необхідно дотримувати таких правил:

1) всередині циклу може знаходитись інший цикл, але цикли повинні мати різні змінні і внутрішній цикл повинен повністю знаходитись у тілі зовнішнього циклу;

2) не можна передавати управління всередину циклу, обминаючи заголовок (це означає, що якщо всередині циклу є мітка, то оператор **goto** переходу до неї також повинен знаходитись всередині цього циклу);

3) якщо необхідно обійти групу операторів у тілі циклу і

продовжити цикл, тобто виконати його наступний крок, то треба передати управління на кінець циклу;

4) можна достроково вийти з циклу, використовуючи або оператор **goto,** або змінивши параметр умови в операторах **while і repeat** так, щоб цикл більше не виконувався.

ЗАПИТАННЯ І ЗАВДАННЯ

1. Нехай тіло циклу в програмі E7 таке саме, як в програмі E6.

Як працюватиме програма, якщо ввести два однакових числа *a* і b?

2. Скільки разів буде виконуватися оператор циклу **repeat,** якщо умова після слова **until** істинна при входженні в цикл?

3. Поясніть, яка різниця між умовами, записаними після слів **while і repeat** для тієї самої задачі?

4. Напишіть програми обчислення сум:

а) для сорока доданків вигляду *n-i,* де *i* =1, 2, 3, ..., 40, а п - дане число;

б) для п доданків вигляду $x+i$, де x - задане число, a і змінюється від 1 до n;

в) для ста доданків, що мають вигляд дробу $(i+1)/(i+2)$;

г) для п доданків вигляду $(i+1)^2$, i =1, 2, ..., п;

д) для п доданків $\sin x + \sin x^2 + \sin x^3 + ... + \sin x^n$;

е) для п доданків $\sin x + \sin 2x + \sin 3x + ... + \sin nx$;

є) кубів n перших натуральних чисел.

5. Для різних цілих чисел, що вводяться з клавіатури, знайти суму додатних непарних.

6. Напишіть програми обчислення добутків:

а) $a(a+1)(a+2)-...-(a+n-1)$;

б) $a(a-n)(a-2n)-...-(a-n^2)$;

в) $(x-1)(x-2)(x-3)-...-(x-n)$;

г) $2\text{-}4\text{-}6\text{-}.,.\text{-}(2n)$;

д) $(1+\sin 0.1)(1+\sin 0.2) – ... – (1+\sin 10)$;

е) Усіх чисел від 1 до 100 кратних 3, але не кратних 6;

є) n співмножників вигляду $(x+i)^2$.

7. Для додатного числа E знайти серед чисел 1, 1+1/2,

1+1/3, ... перше, більше ніж *A*.

8. Ввівши числа з клавіатури без обмеження їх кількості (кінець введення - число нуль), знайти суму додатних і добуток від'ємних чисел.

2.7. КОМП'ЮТЕРНІ ОБЧИСЛЕННЯ

Числа, з якими ми маємо справу в житті, бувають двох типів. Одні точно дають істинну величину, інші — наближено. Перші називають точними, другі — наближеними. Досить часто використовуємо наближене число замість точного, тому що останнє буває необов'язковим. Але навіть якщо є потреба знати число точно, часто отримати його неможливо.

Якщо говорять про відстань між сусідніми селами, то цю відстань вказують у кілометрах. Вимірювати цю відстань можна також у метрах або навіть і в сантиметрах. Тобто відстань між селами може бути вказана з точністю до кілометра, метра або сантиметра. В усіх цих випадках числа, які характеризують відстань, є наближеними. Результат обчислень із використанням наближених чисел є наближеним числом.

Нескінченний неперіодичний десятковий дріб можна обчислити тільки наближено, із скінченною кількістю десяткових знаків, або, як прийнято говорити, — із певною точністю. Точність вказує на кількість десяткових знаків, які задовольняють користувача. Якщо число є результатом обчислень на комп'ютері, то кількість десяткових знаків обмежуються розміром пам'яті, яка виділяється для цього числа.

Дійсні числа, що використовуються в Паскалі, займають шість байтів у пам'яті комп'ютера. Тому точність подання дійсних чисел обмежена розміром даної пам'яті і не перевищує 11 десяткових знаків.

Комп'ютери дозволяють обчислювати значення таких чисел з різним ступенем точності, але ця точність не може бути вищою ніж комп'ютерна — не можна отримати більшу кількість цифр у числі, ніж уміщується у відведеній для числа пам'яті.

Розглянемо приклади обчислень, які виконуються на

комп'ютері, із заданою точністю.

1. Обчислення кореня квадратного з даного числа. Точному значенню кореня квадратного з числа A відповідає точка на числовій прямій. Візьмемо деяке наближене значення цього числа ліворуч від даної точки: нехай це буде x_0 (рис. 40). Тоді значення A/x_0 на числовій прямій відповідає точці, яка знаходиться праворуч від точного значення кореня.

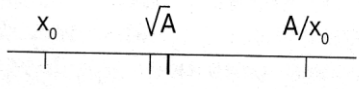


Рис. 40. Вибір наближеного значення кореня

За перше наближення ірраціонального числа можна взяти півсуму чисел x_0 і A/X_0

$$x_1 = 1/2\ (x_0 + A/x_0).$$

Продовжуючи міркування, дістанемо послідовність значень $x_1\ x_2, \ldots, x_n$. Загальний вигляд формули, за якою проводять обчислення, такий:

$$x = 1/2\ (x + A/x_n).$$

Обчислювальний процес, при якому кожне наступне значення визначається з використанням одного або кількох попередніх значень, називається ітераційним. Розглянутий алгоритм, за допомогою якого можна одержати наближене значення кореня квадратного з даного числа, був відомий ще грецькому математику Герону (близько I ст. н.е.), і тому він називається алгоритмом Герона.

Можна узагальнити наведену вище формулу для кореня n-го степеня:

$$x_{n+1} = 1/n\ ((n–1)\ x_n + A/x_n^{n–1}).$$

Як же визначити, коли буде досягнута потрібна точність обчислень? Для ітераційних процесів одна з ознак досягнення заданої точності може бути така: модуль різниці двох сусідніх

значень не повинен перевищувати задану похибку ε (читається «епсілон»):

$$\left|x_{n+1} - x_n\right| \le \varepsilon$$

Оскільки в Паскалі немає грецької літери є, позначимо похибку обчислень через eps. У програмі E12 використовуються всього дві змінні: *x*1 і *x*2, але обчислення можуть повторюватися досить велику кількість разів (залежно від заданої точності). При побудові послідовності уточнених значень невідомо, скільки її членів буде одержано для заданого значення похибки.

Отже, невідомо, який об'єм пам'яті потрібен для збереження всіх цих даних. Використовуючи той факт, що на кожному кроці обчислень необхідно знати значення тільки одного, попереднього, члена одержуваної послідовності, будемо щоразу нове значення записувати в ділянку з іменем *x*1, а попереднє брати з ділянки з іменем *x*0. Щоб на кожному кроці обчислень використовувалось щойно одержане значення, будемо заносити його щоразу до ділянки *x*1:

***x*1 := *x*0;**

Тоді, незалежно від числа повторень і отриманих при цьому членів послідовності, у програмі можна обмежитись двома змінними. За початкове значення кореня можна взяти будь-яке число (крім нуля), наприклад одиницю.

Програма наближеного обчислення кореня квадратного I числа A із заданою точністю eps може бути такою:

```
program E12;
var A, eps, xO, x1: real;
  begin
  write('введіть додатне число A =');
  readln (A);
  write('введіть точність обчислень-правильний десятковий
  дріб'); readln(eps);
  x0 := 1;
  x1 := 0.5*(x0 + A/x0);
  while abs(x1 – x0) > eps do
```

```
    begin
        x0 :=x1;
        x1 :=0.5*(x0 + A/x0)
end;
writeln('корінь квадратний з числа A, '=', x1)
end.
```

2. Обчислення значень тригонометричних функцій. З вищої математики відомо, що багато функцій, у тому числі й тригонометричних, можна подавати у вигляді нескінченної суми. Для того щоб обчислити значення такої суми із заданою точністю, потрібно взяти певну кількість доданків. Кількість доданків залежить від заданої точності обчислень. Наприклад, функцію sin *x* можна подати у вигляді

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} + \ldots, \quad n = 1, 2, 3, \ldots$$

Це представлення зручно використовувати для обчислення значень синуса при $0 \le x \le \pi/4$

Для обчислення одного за іншим доданків розглянутої суми, що включають степені і факторіали, використовуються залежності між послідовними значеннями. Це дозволяє на кожному кроці одержувати наступний доданок, знаючи попередній, для чого досить попередній доданок помножити на деяке число.

Для отримання залежності між двома сусідніми доданками запишемо два послідовних доданки в загальному вигляді і поділимо наступний на попередній. У результаті отримаємо:

$$a_n = (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}; \quad a_{n+1} = (-1)^{n+1}\frac{x^{2n+1}}{(2n+1)!} = -a_n\frac{x^2}{2n(2n+1)}.$$

Обчислення припиняються, коли модуль різниці двох послідовних сум буде меншим або дорівнюватиме заданій точності. Різниця двох послідовних сум дорівнює наступному доданку, тому умова закінчення обчислень запишеться так: $|a_n + 2| < \varepsilon$

Використаємо в програмі наближеного обчислення значення sin *x* одну змінну a для обчислення кожного доданка. У програмі зручно ввести змінну *n2 — 2n*— 1, яка на кожному кроці буде збільшуватися на 2. При цьому програма матиме вигляд:

```
program E13;
var x,a,s,eps: real;   n2: integer;
begin
  write("x=");
  readln(x);
  write("eps = ");
  readln(eps);
  a := x;
  s := x;
  n2 := 1;
  repeat
        a := -a * sqr(x)/((n2+1)*(n2+2));
        s := s + a;
        n2 := n2 + 2
until abs(a) < eps;
writeln("sin ", x, "= ", s) e
end.
```

3. Уточнення кореня рівняння методом ділення відрізка навпил. Нехай задано рівняння

$$x^2 - \cos x + 1/2 = 0.$$

У шкільному курсі математики розглядаються методи підшукання точних розв'язків деяких типів рівнянь: квадратних, тригонометричних, показникових тощо. Якщо ж алгебраїчне або трансцендентне рівняння досить складне (як у розглянутому випадку), то в більшості випадків знайти його точний розв'язок неможливо. Тому важливого значення набувають методи наближеного обчислення коренів рівняння. Одним із таких методів є метод ділення відрізка навпіл. Розглянемо рівняння

$$f(x) = 0,$$

де функція *f(x)* визначена і неперервна на деякому

інтервалі $a \le x \le b$

Будемо виходити з припущення, що рівняння $f(x) = 0$ має лише ізольовані корені, тобто для коленого кореня рівняння існує деякий відрізок на осі О*x,* який не містить інших коренів цього рівняння.

Наближене обчислення ізольованих дійсних коренів рівняння складається з двох етапів:

1) відокремлення коренів, тобто встановлення невеликих (по можливості менших) проміжків, колений з яких містить тільки один корінь рівняння;

2) уточнення наближених коренів, тобто проведення обчислень до досягнення наперед заданої точності.

Якщо функція *f(x)* набуває нульового значення, то це означає, що вона перетинає вісь О*x* у деякій точці, а знак функції змінюється або з мінуса на плюс (рис. 41), або з плюса на мінус.

Для відокремлення коренів можна використати графік функції *f(x).* Графік може бути побудовано досить наближено, щоб за ним можна визначити кінці інтервалу, який містить корінь. Потім треба переконатися, що на даному інтервалі функція дійсно змінює знак, тобто слід обчислити *f(a) і f(b)*Для уточнення кореня обчислимо значення функції *t(x)* у середній точці відрізка:

$$c := (a + b)/2;$$

Якщо в точці с функція *f(x)* має такий самий знак, як і в точці а, перенесемо точку а вправо в точку с (при цьому змінна а набуде значення с, тобто *a: — c).* Якщо в точці с функція має такий самий знак, як і в точці b, перенесемо точку b вліво в точку с (*b*: = с). Після перенесення точки новий відрізок ділиться навпіл і знову визначається знак функції у його середній точці.

Обчислюючи значення кореня за описаним методом, переносячи кінці відрізка в нові точки і тим самим поступово звужуючи відрізок, отримуємо послідовні наближення до точного значення шуканого кореня рівняння — координати точки перетину графіка функції *f(x)* з віссю абсцис. Обчислення продовжуються доти, доки не буде досягнута задана точність.

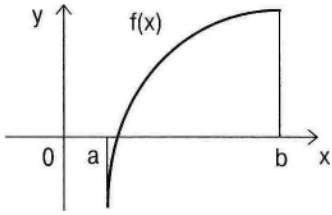


Рис. 41

Напишемо програму відшукання наближеного розв'язку рівняння *f(x)* =0 за методом ділення відрізка навпіл для конкретної функції

f(x) = x^2 — cos *x* + 1/2 на відрізку [0; 1]:

```
program E14;
var a, b, c, eps : real;
begin
  a := 0;  b := 1;  eps := 0.0001;
  while abs(b-a) > eps do
          begin c := ( a + b ) / 2;
          if (a*a-cos(a)+0.5)*(c*c-cos(c)+0.5)>0
          then a := c
          else b := c
      end;
   writeln(корінь f(x) на відрізку ", a, ",", b," дорівнює ", (a+b)/2)
end.
```

ЗАПИТАННЯ І ЗАВДАННЯ

1. Поясніть ідею методу обчислення наближеного значення кореня квадратного з числа *A,* використовуючи алгоритм Герона.

2. Напишіть програму обчислення кореня п'ятого степеня з числа *A.*

3. Як узнати кількість ітерацій при обчисленні sin x у програмі Е13?

4. Поясніть, чому достатньо двох величин для обчислення будь-якої кількості членів послідовності в програмі Е12?

5. Чому точність обчислень на комп'ютері не може бути довільною?

6. Чи можна вибрати відрізок для уточнення кореня, якщо

на кінцях відрізка функція має однакові знаки?

7. Чи можна сказати, що відрізок [–5; +5] для функції $f(x)=3x^2-5x-3$ містить ізольовані корені?

8. Напишіть програму обчислення кореня рівняння для функції f(x)=0 із прикладу E14 на відрізку [0; 1] методом ділення відрізка навпіл 20 разів. Визначіть, із якою точністю буде обчислений корінь рівняння.

9. Використовуючи програму E12, обчисліть корені квадратні з чисел 1.45, 1.678, 0.2, 131 з точністю 1E-6.

10. Використовуючи приклад E13, обчислити значення sin x при x = π/8 із точністю 1E-5. Число π задати як константу 3.141592.

2.8. МАСИВИ

У розглянутих раніше прикладах програм проводилась обробка одиничних даних — значень простих змінних. При розв'язуванні практичних задач часто доводиться аналізувати не окремі значення, а деякі сукупності таких значень (наприклад при табличному задані функцій). Такі дані об'єднуються в різні структури, найпростішими з яких є масиви. Масив — це впорядкований іменований набір із фіксованої кількості однотипних даних. Доступ до будь-якого елементу масиву здійснюється за його номером. У масиви можна об'єднати результати експериментів, списки прізвищ співробітників, різні складні структури даних. Наприклад, список учнів у класному журналі 10-А є масивом. У масиві дані розрізняються за своїми порядковими номерами (індексами). Якщо кожний елемент масиву визначається за допомогою одного номера, то такий масив називається одновимірним, якщо за двома — то двовимірним. Двовимірний масив — це таблиця з рядків і стовпців. У таблицях перший номер вказує на рядок, а другий — на положення елемента в рядку. Усі рядки таблиці мають однакову довжину.

Одновимірний масив може бути набором чисел, сукупністю символьних даних чи елементів іншої природи

(навіть масив масивів). Так само, як і в послідовності, в одновимірному масиві можна вказати елемент з конкретним номером, наприклад a_5 , або записати загальний вигляд елемента, використовуючи як індекс змінну *i*, вказуючи діапазон її зміни:

$$a[i], i = 1,2...,n$$

Задачі, в яких використовуються масиви, можуть мати різне формулювання. Наприклад, задача починається зі слів «Дано n чисел...» і далі вказується, що треба зробити з цими числами. При написанні програми мовою Паскаль для розв'язання подібної задачі слід виконати такі дії:

1) визначити, які числа задані: цілі чи дійсні; якщо про це конкретно не сказано, то краще вважати їх дійсними;

2) назвати весь масив одним ім'ям (це ім'я буде використовуватись для кожного елемента, тільки до імені масиву додаватиметься номер елемента — індекс);

3) описати масив у розділі змінних **var,** тим самим відвести місце в пам'яті для масиву;

4) ввести дані до пам'яті.

В описі масиву використовують спеціальне слово **array** (масив), після якого у квадратних дужках через дві крапки вказують діапазон змінювання номерів елементів, далі слово **of** (із) і тип даних масиву. Коли транслятор зустрічає опис масиву, він відводить для нього стільки послідовних ділянок пам'яті, скільки вказано в квадратних дужках, і такої довжини, яка відповідає типу даних у масиві. Здебільшого номери елементів змінюються від 1 до заданого числа n, яке є останнім значенням (верхньою межею) номера елемента масиву. Значення л можна задати в розділі констант **(const).**

Приклад опису масиву:

```
const n = 10;

var a: array [1..n] of real;
```

Цей опис означає, що для масиву а буде відведено десять ділянок оперативної пам'яті по шість байтів кожна. У Паскалі

імена цих ділянок записуються так:

a [1], a[2], a[3], a[4], a[5], a[6].

В описі після імені масиву a ставиться двокрапка, після якої вказується тип даного — масив. Якщо в програмі кілька масивів одного розміру і типу, то, як і для простих змінних, їх імена можна дати через кому, а потім, після двокрапки, вказати опис масиву.

Для введення даних у пам'ять необхідно організувати цикл. Оскільки введення описаного масиву a може мати вигляд:

for i := 1 **to** n **do** **read**(a[i]);

Значення, що вводяться, набираються на клавіатурі і відокремлюються одне від одного пропусками, після чого натискається **Enter.** Введення можна прокоментувати і вводити кожне дане в окремому рядку після підказу:

```
for i := 1 to n do
begin
  write(a [', i, '] = ');
  readln(a[i])
end;
```

При обробці масивів розв'язування багатьох задач базується на простіших задачах: обчислення суми (добутку) елементів масиву; знаходження найбільшого (найменшого) елемента; упорядкування елементів за зростанням або спаданням. Розглянемо ці базові задачі.

1. Обчислення суми елементів масиву. Знаходження суми елементів масиву нічим не відрізняється, по суті, від додавання значень простих змінних (програма E11). Розв'язування задачі поділяється на три основні етапи:

1) введення даних;

2) обчислення суми;

3) друкування результатів. Програма матиме вигляд:

```
program E15;
const n = 7;
```

```
var a : array [1..n] of real; s: real; i : integer;
begin
  write ('вводьте елементи масиву - ');
  write (n,' дійсних чисел через пропуск');
  for i := 1 to n do
          read(a[i]);
  s := 0;
  for i :=1 to n do
  s := s + a [i];
  writeln;
  writeln(' сума елементів масиву s = ',s)
end.
```

Результати виконання цієї програми наведено в таблиці 10.

Таблиця 10

Вхідні дані 3, –2, 7, 9, –1, 6, 1							
i	1	2	3	4	5	6	7
a[i]	3	–2	7	9	–1	6	1
S	3	1	8	17	16	22	23

2. Знаходження найбільшого елемента масиву. У попередньому прикладі значення змінної *s* змінювались у процесі розв'язування задачі. Для відшукання найбільшого числа серед сукупності чисел потрібно послідовно переглядати і порівнювати між собою числа, записані в пам'яті. Уявімо, що кожне число написане на окремій картці і картки складені купкою. Перше число запам'ятаємо відразу як найбільше і перевернемо картку. Тепер у нашому розпорядженні є два числа: одне бачимо, друге — пам'ятаємо. Порівнюючи їх між собою, запам'ятаємо більше, тобто якщо число, запам'ятоване раніше, більше, то запам'ятовувати нове число не треба, а необхідно дивитись наступну картку. Якщо друге число більше першого, перше далі пам'ятати не треба і запам'ятовуємо лише друге. Таким чином, на кожному етапі порівняння пам'ятатимемо більше з переглянутих чисел і наприкінці знайдемо найбільший

елемент, тобто розв'яжемо поставлену задачу. Записавши цей алгоритм дій у вигляді відповідного набору операторів, дістанемо програму 11 входження найбільшого значення. Проміжні значення-шуканої величини і остаточний результат містить змінна max. Програма має вигляд:

```
program E16;
const n=7;
var a: array [1..n] of integer; max, i : integer;
begin
    for i := 1 to n do
        begin
        write ('a[', i,']= ');
        readln (a[i]);
    end;
  max := a[1];
  for i := 2 to n do
  if max < a[i]
  then max := a[i];
  writeln('Hau6mbUMU елемент масиву max =', max)
end.
```

3. Упорядкування масиву за зростанням. Упорядкування масивів за будь-якою ознакою називається також: сортуванням. Існують різні методи сортування. Вони розрізняються переважно швидкістю отримання результату. Розглянемо один із них — метод «бульбашки». Нехай є послідовність чисел $a_1, a_2, ..., a_n$, яку необхідно впорядкувати за зростанням. Зафіксуємо перший елемент і будемо послідовно порівнювати його з елементами, що знаходяться справа від нього. Якщо якийсь з елементів справа виявиться меншим, ніж зафіксований, поміняємо місцями цей елемент із зафіксованим і продовжимо порівняння вже нового елемента, який знаходиться на першому місці, із тими елементами, що залишилися правіше. Якщо знову знайдеться елемент, менший зафіксованого, повторимо переставляння. У результаті першого перегляду послідовності на першому місці буде найменший з усіх елементів, тобто він як

«найлегший» наче «спливає» вгору. Звідси й назва методу — метод «бульбашки». Далі зафіксуємо другий елемент і повторимо процедуру, виконуючи при потребі переставляння елементів і т.д. З'ясувавши ідею розв'язування, зупинимося на двох питаннях: як фіксувати елементи і як здійснювати перестановку двох елементів. Для того щоб при перебиранні елементів, які знаходяться правіше від того, що перевіряється, не змінювався індекс останнього, необхідно, щоб індекси зафіксованого елемента і тих, що стоять правіше від нього, були різні: наприклад *i* та *j*. Значення індексу *i* змінюється від 1 до n — 1. Значення індексу *j* завжди більше *i* (воно змінюється від i+l до *n).* Для кожного значення z індекс *j* повинен послідовно набути всіх допустимих значень. Отже, конструкція програми, що відображає повний перебір усіх елементів та їх упорядкування за зростанням, являє собою подвійний цикл. При перестановці двох елементів використовується третя змінна. Обмін значеннями в пам'яті двох змінних *a* і *b* виглядає так:

1) *c* := a; 2) a:= b; 3) b := c.

Програма сортування методом «бульбашки» має вигляд:

```
program E17;
const n = 7;
var a: array [1..n] of real; ij: integer; c: real;
begin
  for i:=1 to n do
     begin
          write ('a[',i, ']= ');
          readln (a[i])
     end;
for i:=1 to n-1 do
   for j := i+1 to n do
   if a[i] > a[j] then
          begin
                c := a[i];
                a[j] := c
          end;
   writeln( 'упорядкований за зростанням масив');
```

```
    for i:=1 to n do
    writeln (a[i])
end.
```

4. Пошук елемента в масиві. Одна з важливих необчислювальних задач — пошук даного елемента серед елементів масиву. Такий пошук називається також, пошуком за ключем. На практиці пошук здійснюється в упорядкованому масиві, алгоритми пошуку бувають різні. Розглянемо приклад, де здійснимо пошук елемента масиву шляхом суцільного перебору. Якщо елемент знайдено, надрукуємо його номер; якщо — ні, то видамо відповідне повідомлення.

Суттєвим є те, який з однакових елементів масиву, що збігається з даним, нас цікавить: той, що трапився першим при пошуку чи останнім. У даному прикладі будемо шукати перший. Пошук здійснюється в циклі: як тільки елемент знайдено, треба припинити пошук інших елементів (вийти з циклу). Для дострокового виходу з циклу for використовуємо оператор **goto.** Якщо достроковий вихід не відбудеться, значить елемента, що дорівнює даному, в масиві немає. В такому разі повідомлення про відсутність елемента має бути згаданим відразу після закінчення пошуку. Для цього треба використовувати ще один оператор **goto** і ще одну мітку в програмі. Програма пошуку даного елемента в масиві має вигляд:

```
program E18;
label 1,2;
const n = 7;
var a: array [1.. n] of real; x: real; i: integer;
begin
  writeln('введіть елементи масиву');
  for i:= 1 to n do
          read(a[i]);
  writeln;
  write('введіть число для пошуку в масиві x =');
  readln (x);
  for i:=1 to n do
```

```
            if a[i] = x then goto 1;
  writeln('такого числа в масиві немає');
  goto 2;
1:writeln('номер елемента масиву, що дорівнює даному ', i)
2:end.
```

Якщо шукати не перший елемент, що дорівнює заданому, а останній, то зручно використати цикл зворотного переліку:

for i :- n down to 1 do

ЗАПИТАННЯ І ЗАВДАННЯ

1. Чим відрізняється масив від файла?
2. Для чого потрібний опис масиву?
3. Що треба зробити, щоб почати розв'язувати на комп'ютері задачу, формулювання якої починається зі слів: «Дано *n* чисел...»?
4. За якими ознаками дані об'єднуються в масиви?
5. Чи можна в прикладі програми Е15 обмежитися одним оператором циклу?
6. Що треба змінити в програмі Е16, щоб здійснився пошук не найбільшого, а найменшого елемента масиву?
7. Які зміни в програму Е16 треба внести, щоб одночасно із значенням найбільшого числа визначався його порядковий номер?
8. Поясніть роботу подвійного циклу в програмі Е17.
9. Змініть програму Е18 так, щоб замість циклу «перелік» при пошуку елемента використовувався цикл «доки». Застосуйте змінну-прапо-рець, яка до циклу мала б нульове значення, а у випадку знаходження необхідного елемента змінила б значення на 1. Як при цьому обійтись без операторів **goto?**
10. У заданій послідовності цілих чисел визначити кількість і суму елементів, кратних 10.
11. Дано n чисел. Знайти суму чисел, більших ніж задане число a.
12. У заданому масиві замінити найбільший елемент нулем.

13. Знайти суму квадратів невід'ємних елементів і кількість додатних чисел у заданому цілочисловому одновимірному масиві.

14. У послідовності n чисел поміняти місцями перший і найменший елементи.

15. Дано n чисел. Замінити всі від'ємні числа їх модулями.

16. Обчислити середнє арифметичне найбільшого і найменшого з л чисел.

2.9. АЛГОРИТМИ ОБРОБКИ ТАБЛИЦЬ

Двовимірний масив або прямокутна таблиця *B* з n рядків і *m* стовпців у загальному вигляді записується так:

$$\begin{matrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{matrix}$$

У мові Паскаль імена елементів двовимірного масиву записуються так: вказується ім'я масиву, після якого в квадратних дужках записуються індекси, відокремлені комою, наприклад b[1,1], b[1,2], ..., b[1,m], b[2,1], b[2,2], ... , b[2,m], ..., b[n,m]. Перший із цих індексів вказує номер рядка в таблиці, другий — номер елемента в рядку. У пам'яті комп'ютера елементи двовимірного масиву розташовані один за одним: після елементів першого рядка йдуть елементи другого рядки таблиці і т.д.

Якщо число рядків таблиці дорівнює числу стовпців, то така таблиця називається квадратною. Головна діагональ квадратної таблиці проходить із лівого верхнього кута до правого нижнього.

Розглянемо деякі задачі.

1. Обчислення суми елементів головної діагоналі квадратної таблиці. Для розв'язання цієї задачі треба виконати такі дії:

1) ввести таблицю до пам'яті;
2) знайти суму елементів головної діагоналі;
3) надрукувати результат.

Опис таблиці, як і опис одновимірного масиву, використовується для виділення пам'яті. В описі таблиці вказуються діапазони зміни для двох номерів — номера рядків і номера стовпців:

```
const n = 3;
var b : array [1..n, 1..n] of real; i,j : integer;
```

При обробці масивів у розділі змінних опису програми з'являються імена індексів елементів: для одновимірного масиву — однієї, для двовимірного — двох цілочислових змінних.

При обчисленні суми елементів діагоналі слід звернути увагу на імена елементів, що додаються: обидва індекси мають однакові значення, тобто в загальному вигляді ім'я елемента діагоналі — b[i, i]. Це означає, що сукупність діагональних елементів можна розглядати як одновимірний масив і використовувати один цикл для обчислень.

Програма має вигляд:

```
program E19;
const n = 3;
var b: array[1..n, 1..n] of real;  i,j: integer;  s : real;
begin
  writeln('введіть значення елементів таблиці порядкове);
  writeln('у кінці кожного рядка натискайте Enter');
  for i:= 1 to n do
          begin
                  for j := 1 to n do
                  read(b[i,j]);
          writeln;
    end;
  s:=0;
  for i := 1 to n do
  s := s + b[i,i];
```

writeln(' сума елементів діагоналі таблиці S =', s)
end.

Таблиця 11

Масив результатів		Задана таблиця		
a [1]	6	5	6	1
a [2]	15	4	12	15
a [3]	2	2	–3	0

1. Знаходження найбільших елементів кожного рядка таблиці. Кожний рядок таблиці 11 можна розглядати як одновимірний масив і використовувати ідею знаходження найбільшого значення в програмі Е16. Знайдені значення вміщатимемо в одновимірний масив у таблиці 11. У програмі Е20 для кожного рядка таблиці змінна a[i] відіграє таку саму роль, як змінна max у програмі Е16. Для кожного рядка (її задає змінна i) елемент a[i] набуває значення першого елемента рядка. Потім виконання внутрішнього циклу за *j* мінною *j* дозволяє переглянути всі елементи даного рядка і, якщо серед них трапиться елемент, значення якого більше, ніж запам'ятоване у a[i], то значення цього елемента присвоюється a[i]. Щоб надрукувати результати роботи програми — масив a — використовується цикл. Коментар, взятий у фігурні дужки, дозволяє при читанні програми виділити її окремі частини. Програма має вигляд:

```
program E20;
const n = 3;
var     b: array [1..n, 1..n] of integer;   i,j: integer;
        a: array [1..n] of integer;
begin
        writeln (‘введіть значення елементів таблиці порядкове’);
        writeln('у кінці кожного рядка натискуйте Enter');
        for i:=1 to n do
```

```
                begin
                        for j:=1 to n do
                        read(b[i,j]);
                writeln
        end;
```
{побудова масиву найбільших значень елементів рядків таблиці}
```
        for i:=1 to do
                begin
                a[i]:=b[i,1];
                for j := 2 to n do
                if a[i] < b[i,j]
                then a[i] := b[i,j];
        end;
  writeln( 'найбільші числа рядків таблиці');
  for i := 1 to n do
  writeln(a[i])
end.
```

Таблиця 12

Задана таблиця		
5	6	1
4	12	15
2	–3	0
Масив результатів		
a [1]	a [2]	a [3]
11	15	16

2. Знаходження сум елементів стовпців таблиці.

При обробці таблиць можна здійснювати операції як над рядками, так і над стовпцями. При знаходженні сум елементів стовпців можна використовувати алгоритм прикладу Е15. Введемо змінну S для обчислення суми, а потім для кожного стовпця запишемо отриманий результат в масив a, тобто присвоїмо змінній a[j], де *j* — поточний номер стовпців таблиці.

Приклад виконання програми Е21 наведено в таблиці 12. Програма має вигляд:

```
program E21;
const n = 3;
var b: array [1..n, 1.. n] of integer;  S, i, j: integer;
     a: array [1..n] of integer;
begin
  writeln('введіть значення елементів таблиці порядково');
  writeln('у кінці рядка натискайте Enter');
  for i:=1 to n do
          begin
                  for j:=1 to n do
                  read( b[i,j] ); writeln
          end;
{побудова масиву сум  елементів стовпців таблиці}
  for j:=1 to n do
          begin
                  S:=0;
                  for i:=1 to n do
                  S := S + b[i,j];
                  a[j] := S
          end;
  writeln('суми елементів стовпців таблиці');
  for i:=1 to n do
  writeln(a[i])
end.
```

4. Перестановка рядків таблиці. У прямокутній таблиці *b* із *n* рядків і *m* стовпців треба поміняти місцями два рядки. При розв'язуванні цієї задачі можна використовувати алгоритм обміну значеннями двох змінних із програми сортування (програма Е17).

Для цього достатньо організувати цикл за номером стовпця *j*, використовуючи проміжну змінну, міняти місцями кожну пару елементів, які стоять в одному стовпці в заданих рядках. При заданих номерах рядків *K і L* програма матиме вигляд:

```
program E22;
const n = 3; m = 4;
var b: array [1..n, 1..m] of real;  c: real;  i,j,K,L: integer;
begin
  write ('введіть номери рядків таблиці, що міняються місцями');
  readln (K.L);
{введення таблиці}
  for i:=1 to n do
        begin
              writeln(i, '-ий рядок таблиці');
              for j := 1 to m do
              read(b[i,j]);
        writeln
  end;
{перестановка рядків}
        for j:=1 to m do begin
        c := b[K,j];
        b[j] := b[LJ];
        b[L,j] := c
  end;
{друкування результатів}
  writeln;
  writeln('таблиця з переставленими рядками');
  for i:=1 to n do
  begin
        for j:=1 to m do
        write (b[i,j]);
        writeln end
 end.
```

ЗАПИТАННЯ І ЗАВДАННЯ

1. У квадратній таблиці, що не має від'ємних елементів, знайти корінь квадратний з добутку діагональних елементів.
2. Знайти найбільший елемент квадратної таблиці.
3. Знайти найменший елемент квадратної таблиці і

замінити його нулем.

4. У прямокутній таблиці замінити всі елементи їх квадратами.

5. У цілочисловій прямокутній таблиці збільшити на 0.5 всі від'ємні елементи.

6. У квадратній таблиці знайти найбільший елемент діагоналі.

7. Переставити місцями перший і останній рядки прямокутної таблиці.

8. Знайти добуток елементів рядків прямокутної таблиці.

2.10 Оператор варіанта

Умовний оператор дозволяє здійснювати розгалуження програми тільки за двома напрямами, один із яких відповідає виконанню перевіреної умови, а другий — невиконанню. Якщо для змінної необхідно здійснити ряд дій, які залежать від інших умов, то треба записувати або вкладені умовні оператори, або кілька таких операторів підряд. Для такої ситуації зручно використовувати оператор варіанта. Дана структура називається також перемикачем і виконується так: вхід у структуру містить обчислене або раніше отримане значення змінної (індексу варіанта). Це значення збігатися з міткою, яка стоїть перед оператором на з гілок перемикача. У такому разі виконується оператор із цією міткою і обробка структури закінчується. Оператор бути простим або складним, обмеженим операторними дужками **begin** ... **end;.** Якщо значення індексу варіарта не збіглося ні з однією із міток, то виконується оператор з номером *n*+ 1 із рядка else. Якщо оператор варіанта містить рядок else, то це — повна форма оператора; якщо такого рядка немає, то таку форму оператора варіанта називають скороченою.

Мітки оператора варіанта можуть бути константами будь-якого типу, їх тип повинен збігатися з типом змінної Індексу варіанта. Індекс варіанта може бути як ім'ям змінної, так і виразом відповідного типу.

У мові Паскаль оператор варіанта має вигляд:

```
case індекс варіанта of мітка
  мітка1:  оператор 1;
  мітка2:  оператор 2;
  ...
  мітка n:  оператор n;
else оператор n + 1
end;
```

Наведемо приклад програми, яка містить оператор варіанта. Поширеною задачею з розділу молекулярної фізики є задача, пов'язана з обчисленням кількості молекул в одиниці об'єму в тілі із заданою масою і в тілі з відомим об'ємом. Для розв'язування таких задач можна побудувати програму.

Надано молярну масу речовини *M,* густину даної речовини P, масу *R* або об'єм даного тіла *V.* Треба знайти число молекул *K:*

I) в одиниці маси речовини;

2) в тілі з заданою масою;

3) в одиниці об'єму речовини;

4) в тілі із заданим об'ємом. Для розв'язування задачі скористаємось формулою

$$K = N_A/M,$$

де N_A= 6.022-10^{23} г/моль — число Авогадро.

На основі цієї формули отримаємо розрахункові формули для програми:

1) $K=N_A/M$; 2) $K=N_A R/M$; 3) $K = N_A PV/M$; 4) $K = N_A P/M$.

Програма має вигляд:

```
program E23
const NA = 6.23E23;
var N: integer;   M, R, R V, K: real;
begin
  writeln('Якщо відоме число Авогадро, густина P');
  writeln('цієї речовини і її молярна маса M,');
  writeln('то можна знайти кількість молекул в:');
```

```
  writeln('1. В одиниці маси речовини');
  writeln('2. В тілі масою R');
  writeln('3. В одиниці об'єму');
  writeln('4. В тілі об'ємом V);
  write('Введіть номер задачі, що розв'язується ');
  readln(N);
  write('Введіть початкові дані : M=');
  readln(M);
  case N of
        1:K:=NA/M;
        2:begin
              write('R= ');
              readln(R);
              K := NA * R / M
          end;
        3:begin
              write('густина речовини P=');
              readln(P);
              writer (V= ');
              readln(V);
              K := NA * P * V / M
        end;
        4:begin
              write('густина речовини P=');
              readln(P);
              K:=NA*P/V
        end;
    end;
writeln('кількість молекул K =', K)
end.
```

ЗАПИТАННЯ І ЗАВДАННЯ

1. Для чого використовується оператор варіанта?

2. Навести приклади алгоритмів, які при написанні програм потребують використання операторів варіанта.

2.11 Підпрограми

При розробці програми іноді з'являються повторювані групи дій або виникає необхідність поділити програму на функціональні модулі, зробити її структуру ієрархічною. Для цього у всіх мовах програмування існують засоби організації підпрограм. При розв'язанні складної задачі рекомендується спочатку алгоритм, а потім і програму розроблювані «зверху вниз», від більш загального плану до докладного. В такому вигляді головна програма відповідає укрупненому плану розв'язання задачі, а її команди — виклику відповідної підзадачі, реалізованої у вигляді підпрограми. Ідучи 1.1 ким шляхом створення програми, можна окремі функції, що найбільш часто використовуються, реалізувати спочатку у вигляді невеликих підпрограм, перевірити їх на контрольних прикладах і, переконавшись у правильності, включити їх до складу основної програми. Це зручніше робити, миючи в розпорядженні мову, яка дозволяє повністю виділі Іти підпрограму з тексту основної програми у вигляді окремого модуля. Використання підпрограм дозволяє розробити програму частинами, доручати реалізацію великих проектів окремим групам розробників.

У Паскалі підпрограма є частиною основної програми, її опис дається між розділом **var** головної програми та ЇЇ програмним блоком (першим **begin).** Підпрограм може бути декілька, їх описи розміщуються у довільному порядку один за одним. Опис підпрограми можна порівняти із записуваною в математиці формулою «у загальному вигляді», в яку при розрахунках підставляються конкретні значення. Як і формулу, підпрограму можна використовувати для різних даних, які передаються з програми, що її викликає. Тому параметри, які використовуються в описі підпрограми, називають *формальними,* а параметри, над значеннями яких виконуються операції, вказані в підпрограмі, називають фактичними.

Підпрограма — це спеціально оформлений алгоритм, який можна використовувати багато разів при розв'язуванні задач.

У Паскалі розрізняють два види підпрограм: процедури і

функції. Основна різниця між ними полягає в тому, що процедура може мати будь-яку кількість вхідних і вихідних даних (параметрів), а функція — довільну кількість вхідних і тільки одне вихідне значення.

Підпрограми мають структуру, аналогічну до структури головної програми. Вони починаються з заголовка із спеціальним словом — ознакою підпрограми, далі вказується ім'я підпрограми і, при потребі, список формальних параметрів. Далі розміщуються всі розділи описів, які є в головній програмі: мітки, константи, типи і змінні. У цих розділах описуються дані, які використовуються тільки всередині підпрограми. Такі дані називаються локальними. У підпрограмі можуть використовуватися також змінні, описані у викликаючій програмі. Такі змінні називаються глобальними. Вони можуть використовуватися не лише програмами, які викликають, а й підпрограмами.

При роботі з підпрограмою завжди виділяється два етапи: опис підпрограми, тобто запис алгоритму розв'язання задачі в спеціальній формі, і виклик підпрограми — передача їй даних на обробку з викликаючої програми та отримання результатів.

Розглянемо способи організації підпрограм у Паскалі.

Підпрограми-процедури. Процедура — це підпрограма, яка має будь-яку кількість вхідних і вихідних даних. Процедура може бути описана без параметрів і з параметрами. Параметри в заголовку процедури використовуються для обміну інформацією між процедурою і програмою, яка її викликає. Вони визначають дані, які передаються на обробку до процедури, і дані, які отримуються у вигляді результатів.

Процедури без параметрів. Опис процедури має такий вигляд:

```
procedure ім'я;
{опис локальних змінних}
begin
{оператори}
 end;
```

Процедура без параметрів може реалізовувати будь-який алгоритм. Усі змінні, з якими проводять дії оператори процедури, визначаються у викликаючій програмі, їм присвоюються потрібні для виконання процедури значення. Для виклику процедури без параметрів просто вказується її ім'я.

Розглянемо приклад обчислення найменшого спільного кратного двох натуральних чисел НСК (Х,У), яке можна знайти, використовуючи найбільший спільний дільник (НСД) цих чисел, за формулою

$$НСК(Х, У) = X\text{- } У/ НСД (Х, У).$$

При складанні програми оформимо як процедуру без параметрів програму Е7 — обчислення НСД за алгоритмом Евкліда. Результат роботи процедури заносимо до ділянки з іменем *M.* Змінна *M* описується як глобальний параметр, вона використовується і у викликаючій програмі, і в процедурі. У програмі будемо обчислювати НСК кількох чисел, заносячи їх до масиву *C.* Цей масив формується в розділі констант головної програми. Якщо дані визначаються в розділі констант, то вони не потребують додаткового опису в розділі змінних (**var**). Змінна *X* спочатку набуває значення першого числа, а потім їй присвоюється значення, отримане в результаті виконання підпрограми — НСК двох перших чисел. Значення змінної Y є друге число з кожної пари, для якої обчислюється найменше спільне кратне. Таким чином, на кожному наступному кроці циклу обчислюється НКС двох чисел, перше з яких *X* містить результат попереднього кроку. Наочно цей процес подано в таблиці 13.

Програма має вигляд:

```
program E24;
const c:array[1 .. 5] of integer = ( 36,54,72,18,15 );
var x, y, i, m: integer;
procedure NSD; {заголовок процедури}
var a,b Integer; {опис локальних змінних}
```

Таблиця 13

X: = НСК (X,Y)	36	108	216	216	1080
Y: = C[i]	54	72	18	15	
M: = НСК (X, Y)	18	36	18	3	

```
begin
  a := x; b := y; {збереження початкових даних }
  while a <> b do
  if a > b then a := a - b
  else b := b - a;
  m := a {результат роботи процедури присвоюється глобальній змінній}
  end; {кінець процедури}
  begin { початок головної програми }
        x:=c[1];
        for i := 2 to 5 do
  begin
        y :=c [i];
        NSD; {виклик процедури без параметрів}
        x := x * y div m {div - ділення націло для цілочислових даних}
     end;
        writeln('НСК=',x)
  end.
```

Процедури з параметрами. Для передавання даних до процедури і отримання з неї результатів використовуються формальні і фактичні параметри.

Формальні параметри — це умовні позначення в описі процедури; вони описуються в її заголовку. При виклику процедури після її імені в дужках слід вказати список *фактичних параметрів,* які конкретизують значення, над якими будуть виконуватись операції в тілі процедури. Послідовність, кількість і тип формальних і фактичних параметрів повинні

збігатися. Формальні параметри, описані и заголовку процедури, більше ніде не описуються, їх опис схожий на опис даних у розділі змінних і може також містити слово **var**. Слово **var** у заголовку процедури ставиться перед тими параметрами, імена яких відповідають початковим даним. Фактичні параметри, які відповідають формальним, перед якими стоїть слово **var**, можуть бути тільки іменами змінних. Перед іменами формальних параметрів, чім с вхідними даними процедури, слово **var** можна не вказувати. Якщо перед формальним параметром у заголовку процедури немає слова **var**, то йому може відповідати формальний параметр, який має вигляд виразу відповідного типу. Якщо для вхідних даних процедури при описі формальних параметрів вказано слово **var**, то їм також відповідають фактичні параметри - імена змінних.

Наприклад, процедура NSD (знаходження найбільшого спільного дільника) із параметрами може мати заголовок:

procedure NSD **(**a, b**: integer; var** k**: integer);**

Виклик цієї процедури:

NSD (x, y, m);

або

NSD (36, 54, m);

Змінні *a, b, k y* заголовку процедури NSD — це формальні параметри, які заміняються при виконанні процедури па конкретні значення змінних *x* і у або числа 36 і 54. У заголовку процедури NSD описані формальні параметри:

a і *b* – вхідні дані, для яких знаходиться найбільший спільний дільник; *k* – результат роботи процедури. При виклику процедури змінна а набуде значення змінної *x,* а змінна *b* - значення змінної *y*. Результат виконання процедури надійде до ділянки пам'яті з іменем *m.*

Програма при використанні процедури з параметрами матиме вигляд:

```
program E25;
const c: array [ 1..5 ] of integer = ( 36,54,72,18,15 );
var x,y,i,m: integer;
procedure NSD (a,b : integer; var k: integer); {заголовок
```

процедури}

```
begin
  while a <> b do
  if a > b then a := a – b
  else b := b - a;
  k := a {значення змінної k - результат роботи процедури }
end; {кінець процедури}
  begin {початок Головної програми}
        x:=c[1];
        for i:=2 to 5 do
  begin
        y := c [i];
        NSD (x, y, т); {виклик процедури з фактичними
  параметрами}
        x := x * y div m
  end;
writeln('НСК=', x)
end.
```

Розглянемо ще один приклад використання процедури з параметрами. Знайдемо за допомогою процедури середнє арифметичне, найбільший і найменший елементи масиву.

Програма запишеться у вигляді:

```
program E26;
const n = 10;
type R = array [1 ..n] of real;
var Y: R; A,B,C: real; i: integer;
procedure Stat ( X:R; var S, min, max: real);
begin
  S := 0; min := x[1]; max := x [1];
  for i := 1 to n do begin
  S :=S + x [i];
  if x [i] < min then min := x [i];
  if x [i] > max then max := x [i]
  end;
  S := S/n
```

```
  end;
begin {головна програма}
  for i := 1 to n do
  read ( Y[i] );
Stat (Y, A, B, C); {виклик процедури}
  writeln;
  write (' середнє =', A, ' найменше =', B);
  writeln (' найбільше = ', C)
end.
```

У програмі Е26 з'явився новий розділ описів — розділ типів даних **type,** в якому можна описати новий тип даних через уже відомі типи. Тип даних *R* — це масиви з n дійсних чисел, *R* — ім'я типу. Далі цей тип дозволяє скоротити описи, він використовується в головній програмі при описі вихідного масиву *Y* і в заголовку процедури при описі формального параметра — масиву *X*.

Головна програма складається з трьох основних частин:

I) введення даних — масиву *Y;*

2) виклику процедури Stat з фактичними параметрами — масивом *Y* і отримуваними результатами, які надходять відповідно до ділянок *A* (середнє значення), *B* (найменше) і *C* (найбільше);

3) друкування результатів роботи програми.

Підпрограми-функції. Підпрограма, яка має єдиний результат, може бути оформлена як функція. Опис її має вигляд:

```
function  ім'я _ функції (вхідні дані): тип _ результату;
{опис локальних змінних}
begin
{оператори}
ім'я _ функції := результат;
end;
```

Після опису формальних параметрів, які є аргументами функції, в заголовку вказується тип результату, тобто тип самої функції. Цей опис відноситься до імені функції, якому необхідно присвоїти значення результату роботи підпрограми. Як і

процедура, функція може містити всі чотири розділи опису локальних змінних — **label, const, type** і **var**. Ім'я функції не можна використовувати для проміжних обчислень.

Функція викликається за допомогою вказівника. Вказівник — це ім'я функції, після якого в круглих дужках перелічені фактичні параметри — аргументи функції. Вказівник має вигляд:

ім'я функції (список фактичних параметрів)

Вказівник може з'явитися у виразі відповідного типу, в умовах операторів **if**, **while і repeat** після слова **until,** а також в операторі друку **write.** Прикладами виклику функцій г, арифметичні функції, наприклад **write** (**sin** (x));

Розглянемо третій варіант програми обчислень найменшого спільного кратного. Оскільки найбільший спільний дільник двох натуральних чисел — єдине число, то підпрограму для його обчислення можна оформити як функцію. Програма матиме вигляд:

```
program E27;
const c : array [ 1..5 ] of integer = ( 36,54,72,18,15 );
var x,y,i: integer;
function NSD (a,b: integer): integer; {заголовок функції}
begin
 while a <> b do
   if a > b then a := a - b
   else b := b - a;
 NSD := a {результат роботи функції присвоюється її
            імені}
end; {кінець опису функції}
begin {початок головної програми}
  x:=c[1];
  for i := 2 to 5 do
     begin
     y := c[i];     x := x * y div NSD (x, y)
       end;
writeln('НСД=',x)
```

end. {кінець головної програми}

Одна підпрограма може викликати іншу. Розглянемо програму уточнення кореня рівняння методом ділення відрізка навпіл (приклад Е14), яка включає дві підпрограми. Перша підпрограма — процедура, яка реалізує метод ділення відрізка навпіл, друга — підпрограма-функція, яка обчислює значення функції у потрібних точках. Програма при цьому матиме вигляд:

```
program E28;
var a1, b1, e, y : real; { a1, b1 - кінці відрізка,
                           e - точність, y – результат обчислень}
function f (x: real): real; {заголовок функції}
begin
  f := x * x - cos(x) + 0.5;
  end; {кінець опису функції}
procedure precise(a,b, eps: real; var c: real);
{процедура уточнення кореня методом ділення відрізка навпіл}
begin
  while abs(b - a) > eps do
begin c:=(
  if f(a) * f(c) > 0
  then a := c
  else b := c
  end;
  c := (a + b)/2
end; {кінець процедури}
begin {початок головної програми}
  write ('введіть значення кінців відрізка');
  readln (a1, b1);
  write ('введіть точність');
  readln (e);
  precise(a1, b1, e, y);
  writeln('корінь f(x) на відрізку ', a1, ',', b1, ' дорівнює ', y)
end. {кінець головної програми}
```

ЗАПИТАННЯ І ЗАВДАННЯ

1. Що таке підпрограма і для чого вона використовується?
2. Поясніть призначення локальних і глобальних змінних.
3. Що таке формальні і фактичні параметри?
4. До чого відноситься опис типу в кінці заголовка підпрограми-функції?
5. Чим відрізняється виклик функції від виклику процедури?
6. Як задати значення елементів масиву без використання оператора введення?
7. Оформити програми попередніх параграфів, крім розглянутих у даному, із використанням процедур.

2.12. Рекурсія

Якщо поставити два дзеркала одне проти одного і між ними помістити предмет, то вийде нескіченна кількість зображень, кожне з яких містить само себе. Кожне з цих зображень можна розглядати як рекурсивний об'єкт, тобто такий, який частково складається або визначається за допомогою самого себе. Рекурсивні об'єкти мають такі властивості: простоту побудови; несхожість кінцевого результату з початковими даними; внутрішню самоподібність.

У математиці трапляються рекурсивні визначення, які дозволяють описати об'єкти за допомогою самих себе. Одним з таких визначень є, наприклад, визначення натурального числа:

1) одиниця є натуральне число;

2) наступне за натуральним ціле число є натуральним числом.

Визначення, яке задає деякий об'єкт у термінах більш простого випадку цього ж об'єкта, називається рекурсивним. Як і цикл, рекурсивне визначення містить повторення, але це повторення є неявним.

Рекурсія — це спосіб опису функцій або процесів із використанням самих себе.

Процес може бути описано деяким алгоритмом, названим у даному разі рекурсивним. У таких алгоритмах виділяється два етапи виконання:

1) «занурення» алгоритму в себе, тобто використання визначення «в інший бік», доки не буде знайдене початкове визначення, яке не є рекурсивним;

2) послідовне виконання операцій від початкового визначення до визначення з введеним в алгоритм значенням.

Розглянемо приклади рекурсивних алгоритмів, які часто оформляються у вигляді процедур і функцій.

1. Найпоширенішим прикладом рекурсії є визначення факторіала (не рекурсивне обчислення факторіала наведено в прикладі Е9):

(a) 1! = 1,

(b) n>1 : n! = n(n-1)!

На основі цього визначення можна записати програму обчислення факторіала, яка використовує рекурсивну функцію:

```
program E29;
var n,y :integer;
function F (x: integer): integer; {опис рекурсивної функції}
begin
  if x= 1
  then F := 1 {виклик для початкового визначення}
  else F := x * F(x-1); {виклик для попереднього визначення}
end; {кінець опису функції}
begin {початок головної програми}
   write ('введіть натуральне число не більше 14 n=');
   readln (n);
   Y := F(n); {виклик функції у головній програмі}
   writeln(n,'!=', Y)
end. {кінець головної програми}
```

Виконаємо програму Е29 для n=4. Рекурсивна функція буде працювати таким чином (при виклику функції значення *n* буде присвоєно змінній *x).* Спочатку здійснюється «занурення», працює оператор гілки **else** умовного оператора:

1-й крок: *x = 4, x*— 1=3,
виконується проміжне обчислення 4! =4*3!
2-й крок: x = 3, *x*— 1 =2,
виконується проміжне обчислення 3! = 3*2!
3-й крок: *x = 2, x*— 1 = 1,
виконується проміжне обчислення 2! =2*1!
4-й крок (останній): 1! = 1
за початковим визначенням, працює оператор F: = 1 гілки **then** умовного оператора.

Наступний етап виконання рекурсивного алгоритму — побудова «прямого» визначення, від початкового до отримання результату з вихідними для алгоритму даними (числом 4). При цьому здійснюється підстановка попередніх обчислень (більш пізніх кроків) в більш ранні:

5-й крок: 2! = 2*1 =2
6-й крок: 3! =3*2 = 6
7-й крок: 4! = 4*6 = 24 — отримано результат, який повертається до головної програми і присвоюється змінній Y.

2. Обчислення степеня з натуральним показником можна визначити рекурсивне:

(а) $x_0 = 1$
(b) $k > 0$: $x^k = xx^{k-1}$

Цьому визначенню відповідає рекурсивна функція **power** *(k,x).* Програма має вигляд:

```
program E30;
var y: real;  n: integer;
function power (k: integer; x: real): real; {опис рекурсивної функції}
begin
  if k = O
  then power := 1     { початкове визначення }
  else power := x * power(k-1, x);{рекурсивне визначення}
end; { кінець опису функції }
begin { початок головної програми }
  write(' основа степеня x =');
  readln(y);
```

write(' показник степеня k =');
readln(n);
writeln(y, 'в степені ', n, '=', power (n,y)) { виклик функції і друкування результату }
end. {кінець головної програми}

3. Розглянемо обчислення чисел Фібоначчі. Італійський математик Фібоначчі придумав послідовність натуральних чисел: 1, 1, 2, 3, 5, 8, 11, ... Перші два члени послідовності дорівнюють одиниці, а кожний, починаючи з третього, дорівнює сумі двох попередніх. Для чисел Фібоначчі справедливе співвідношення:

$$F_k = F_{k-1} + F_{k-2}$$

Рекурсивна процедура отримання значення n-го числа Фібоначчі має вигляд:

```
function Fib (n: integer): integer;
begin
  if k< 3
  then Fib := 1
  else Fib := Fib ( n - 1 ) + Fib ( n - 2 )
end;
```

Для чисел Фібоначчі використовується таке рекурсивне визначення:

(a) $n = 1, n = 2$: fib(n) = 1
(b) $n > 2$: fib(n) = fib(n-2) + fib(n-l)

Для того щоб визначити fib (6), застосовуючи дане рекурсивне визначення, зробимо такі обчислення:

fib(6)= fib(4) + fib(5) = fib(2) + fib(3) + fib(5) =
= 1 + fib(3) + fib (5) =
= 1 + fib(l) + fib(2) + fib(5) =
= 1 + 1 + 1 + fib(5) =
= 3 + fib(3) + fib (4) =
= 3 + fib(l) + fib (2) + fib(4) =

= 3 + 1 + 1 + fib(4) =
= 5 + fib (2) + fib(3) =
= 5 + 1 + fib(l) + fib(2) =
= 6 +1 + 1=8

Кількість дій у даних обчисленнях при використанні рекурсивного визначення чисел Фібоначчі різко збільшується, тому що це визначення посилається саме на себе двічі. При обчисленні факторіала кількість дій при виконанні програми з рекурсивною функцією і прикладу Е9 однакова.

4. Рекурсивні алгоритми можуть бути оформлені і у вигляді процедур. Прикладом такої процедури є процедура розв'язування задачі про ханойські вежі (рис. 42).

Ця задача пов'язана з легендою про те, що в одному із східних храмів знаходиться бронзова плита з трьома діамантовими стержнями. На один із них Бог при створенні світу нанизав 64 диски з чистого золота так, як показано на рис. 42. Жерці повинні переносити диски з одного стержня на інший, дотримуючись таких законів:

1) диски можна переносити тільки по одному;

2) по можна класти більший диск на менший.

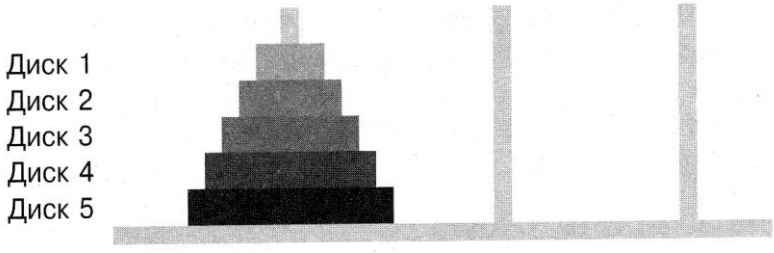


Рис. 42. Ханойські вежі

Згідно з легендою, коли всі диски будуть перенесені з одного стержня на інший, настане кінець світу.

Розв'язування цієї задачі реалізовано у вигляді рекурсивного алгоритму, який являє собою інструкцію щодо переміщення дисків. Сформулюємо задачу, присвоївши імена

стержням *(A, B, C)* і номери дискам (від 1 до n). Треба перенести диски із стержня *A* на стержень *C,* використовуючи *B* як допоміжний і дотримуючись наведених вище правил перенесення дисків.

Опис алгоритму природною мовою має вигляд:

1. Якщо n = 0, слід зупинитись.

2. Перемістити верхні n — 1 дисків із стержня *A* на стержень *B,* використовуючи стержень *C* як допоміжний.

3. Перемістити диск, що залишився, зі стержня *A* на С.

4. Перемістити n — 1 дисків із стержня *B* на стержень С, використовуючи стержень *A* як допоміжний.

У процедурі з'являється новий тип даних — **char,** значення якого — один символ, який береться в апострофи.

Програма матиме вигляд:

```
program E31;
var k: integer;
procedure Hanoy (n: integer;  One, Two, Three: char);
begin
  if n > 0 then
  begin
    Hanoy (n-1, One, Three, Two);
    writeln('перемістити диск', n, ' із стержня',
              One,' на стержень ', Two);
    Hanoy (n-1, Two, One, Three)
 end;
end;
begin
write(' введіть кількість дисків');
readln(k);
Hanoy (k, 'A' , 'B', 'C')
end.
```

Результат роботи програми для n = 3 – це інструкція із семи пунктів (для л = 4 — інструкція з 15 пунктів);

1) перемістити диск 1 із стержня А на стержень С;

2) перемістити диск 2 із стержня A на стержень B;
3) перемістити диск 1 із стержня C на стержень B;
4) перемістити диск 3 із стержня A на стержень C;
5) перемістити диск 1 із стержня B на стержень A;
6) перемістити диск 2 із стержня B на стержень C;
7) перемістити диск 1 із стержня A на стержень C;

ЗАПИТАННЯ І ЗАВДАННЯ

1.Що таке рекурсивний об'єкт і які його властивості?

2.Наведіть приклади рекурсивного визначення в математиці.

3.Що таке рекурсія?

4 Як виконується рекурсивний алгоритм?

5.Поясніть виконання рекурсивної функції обчислення степеня з натуральним показником.

6.Напишіть головну програму для обчислення n-го числа Фібоначчі.

7.Чому використовувати рекурсивний алгоритм обчислення n-го числа Фібоначчі невигідно?

8.Напишіть нерекурсивну програму обчислення n-го числа Фібоначчі, використовуючи три змінні (див. п. 2.7).

9.Визначіть рекурсивне множення як додавання і ділення як віднімання і оформіть алгоритми у вигляді рекурсивних функцій з викликом із головних програм.

2.13. Обробка рядків у Паскалі

У нам'яті комп'ютера можуть зберігатися числа й символи Кожний символ займає один байт пам'яті. Для даного, значенням якого є одиночний символ, використовується описувач **char.** Символи можуть об'єднуватися в масиви. Кожному елементу масиву, як і для числових даних, відповідає порядковий номер, а ім'я елемента складається з імені всього масиву і його номера. Значення символьного даного — це символ, взятий в апострофи, наприклад 'A' , '?' , '5'. Знак апострофа представляється при записі двома апострофами.

Приклади описів:

var *a:* **array** [1..50] **of char;** x,y: **char;**

Масив *a* може складатись з 50 символів, йому відводиться при трансляції програми 50 байтів пам'яті. Елементи масиву: a[1], a[2], ..., a[50]. Змінні *x i y* — прості, їх значення — одиночні символи.

Для введення символьного масиву слід використовувати такий цикл:

for i:=1 **to do read**(a[i]);

При введенні такого масиву можна набрати рядок із n символів і натиснути **Enter.**

В описі можна задати таблицю символів і для її введення використати подвійний цикл:

```
const n = 10; m = 15;
var b: array [1..n, 1:m] of char; i, j: integer; begin
 for i:=1 to n do
  begin
    for j:=1 to m do
    read(b[i,j]);
    writeln
   end
end.
```

У прикладі розглядається таблиця b з 10 рядків по 15 символів кожна. При її введенні необхідно набирати рядки по 15 символів і натискувати **Enter.** Незручність такого способу введення полягає в тому, що всі рядки повинні мати 15 символів, тобто слова, що набираються, не повинні складатися більше ніж з 15 літер, а для коротких слів треба додавати пробіли.

При роботі із символьними масивами використовуються такі самі алгоритми, як і при роботі з числовими. Наприклад, слово, задане як масив символів, треба записати у зворотному порядку, тобто справа наліво. При розробці алгоритму можна використати таку постановку задачі: даний числовий мамів переписати так, щоб останній елемент встав на перше місце, передостанній — на друге і т.д., а перший — на останнє, тобто необхідно з масиву $a_{,}$, $a_{,2}$, ... , a_n одержати a_n, a_{n-1} ... , $a_{1,}$, який

буде знаходитись у масиві *b* (рис. 43).

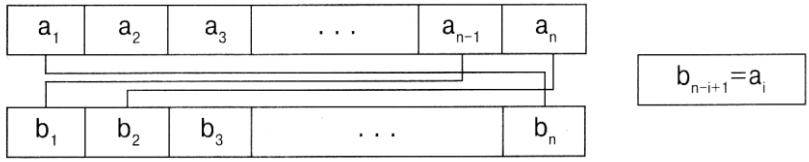


Рис. 43. Переміщення елементів з масиву *a* в масив *b* і формула перерахунку індексів

На рис. 43 рамкою обведена формула перерахунку індексу: коли в масиві а номери перераховуються в прямому порядку, тобто поточний індекс елемента масиву змінюється під 1 до *n,* то в масиві *b* індекси повинні змінюватись від n до I . Таку зміну індексів і забезпечує наведена формула для індексів масиву *b*. Програма, яка виконує переставляння елементів масиву у зворотному порядку, для символьних даних називається програмою обернення слова. Вона має вигляд:

```
program E32;
const n = 15;
var a,b: array[1..n] of char;   i: integer;
begin
  for i:=1 to n do
    begin
      read(a[i]);
    end;
  writeln;
  for i:=1 to n do
  writeln(b[i])
end.
```

Кілька символів, записаних підряд, утворюють рядок. *Рядок* — це взята в апострофи послідовність будь-яких символів. Довжина рядка, що обробляється в Паскалі, не повинна перевищувати 255 символів (обмежувальні апострофи по враховуються). Це пов'язано з тим, що в нульовому байті рядка зберігається значення довжини цього рядка, тобто кількість

символів у ньому, а найбільше ціле число, яке може бути записано в байті, — це 255. Якщо треба опрацювати текст, довжина якого понад 255 символів, то слід використовувати масив рядків.

Опис рядка має вигляд:

var x: **string**[20];

Рядок *x* повинен бути не більший, ніж 20 символів, сі якщо він менший, то займатиме в пам'яті стільки байтів, скільки знаків він містить (плюс 1 байт для зберігання довжини рядка). Тому при введенні рядка немає необхідності доповнювати його до вказаної в описі довжини.

Для опрацювання рядка використовуються спеціальні операції і підпрограми. Такі операції дозволяють працювати з рядками як з окремими об'єктами, а підпрограми застосовуються переважно для опрацювання окремих символів або частин рядка.

Операції над рядками — це об'єднання, порівняння і присвоювання.

Об'єднання рядків. Ця операція дозволяє з'єднати два рядки в один, приєднавши початок другого рядка до кінця першого. Об'єднання позначається знаком « + ». Наприклад:

```
var x, y, z : string[10];
begin
   x = 'тепло';
   y = 'хід';
   z = x + y;
   writeln(z)
end.
```

Змінним *x* та *y* присвоюються значення рядків, а змінній z - результат об'єднання цих рядків в один: «теплохід». При друкуванні рядка буде видано зміст ділянки пам'яті, яка називається z.

Очевидно, що операція об'єднання рядків некомутативна, тобто для неї $a + b \neq b + a$, тому при використанні об'єднання необхідно передбачати, з якого боку до даного рядка приєднується інший: зліва чи справа.

Як і для арифметичних операцій, для даної операції над рядками існує нейтральний елемент, який не впливає на її результат. Це рядок нульової довжини (порожній рядок), який позначається двома апострофами, що стоять поряд (''). Такий рядок можна приєднати до будь-якого рядка зліва або з права, від чого рядок не зміниться.

Порівняння рядків. Для порівняння рядків використовуються операції, які мають такі самі позначення, як і операції підношення для чисел, але вони мають дещо інший зміст. Якщо рядки порівнювати на «дорівнює» (=), то рівність означає посимвольний збіг рядків. Відповідно «не дорівнює» (< >) означає, що рядки не збігаються. При застосуванні решти операторів (<, < = , > = , >) відношення рядків заміняється відношенням перших двох відповідних один одному, але не рівних символів (символи порівнюються за їх порядком у таблиці символів). Якщо така пара символів не знайдена, а рядки збігаються до останнього символу рядка, який має меншу довжину, то більш короткий рядок вважається меншим. Використовуючи ці оператори відношення можна впорядковувати слова за алфавітом (як у словнику).

Присвоювання. Оператор присвоювання при обробці рядків має вигляд:

ім'я _ рядкової _змінної := рядковий вираз;

Ім'я рядкової змінної може бути просте або з індексом (елементом масиву рядків). Якщо в результаті виконання всіх операцій рядкового виразу дістанемо рядок, довжина якого перевищує довжину в описі змінної, що вказана зліва від знака присвоювання, то одержаний рядок скорочується справа до допустимої довжини:

```
var x : string[6];
begin
x := 'коли' + 'барбарисовий';
 writeln(x)
end.
```

У результаті роботи цієї програми буде надруковано слово «колиба», бо допустима довжина *x* - 6 символів, і значення

виразу справа від присвоювання «колибарбарисовий» скоротиться до «колиба», інші символи будуть відкинуті.

Довжина рядка. Функція довжини рядка видає кількість символів у рядку:

length (рядковий_вираз)

Як приклад використання цього оператора розглянемо програму, яка визначає довжину двох рядків та їх об'єднання:

```
program E33;
var x,y: string[20];   k, l, n: integer;
begin
writeln('введіть два рядки');
readln(x);  readln(y);
k := length(x);   I := length(y);  n := length(x+y);
writeln  ('довжина першого рядка' :25,
          'довжина другого рядка' :25);
writeln(k:25, 1:25);
writeln(x+y, 'довжина рядка ', n)
end.
```

У програмі Е33 використовується виведення результату з форматуванням. Перший раз формат (:25) вказано після рядка, що виводиться на екран («довжина першого рядка»). Це означає, що для виведення даного рядка відводиться по 25 позицій. Оскільки текст, що виводиться, коротший (20 символів), то він доповниться на початку пропусками, тобто буде розміщений справа у відведеному йому полі. Аналогічно розміщуються у відведеному для них місці цілі числа, — довжини рядків. Результат роботи наведеної програми матиме такий вигляд:

```
довжина першого рядка          довжина другого рядка
                    7                             10
```

За допомогою форматування можна розміщувати дані, що водяться, в стовпцях, будувати на екрані дисплея таблиці.

Копіювання рядка або його частини. Функція копіювання називається також «вирізанням». Вона дозволяє

скопіювати одну ділянку пам'яті в іншу. Для копіювання необхідно вказати рядковий вираз, із значення якого виділяється частина, а також початковий номер символу й кількість символів частини, що копіюється:

copy (рядковий вираз, початковий номер символу, кількість слів)

Наприклад, результатом роботи функції

copy ('Інформатика',3,5)

буде слово «форма».

Застосуємо дану функцію для розробки іншої версії програми обернення слова. Оброблятимемо слово, вилучаючи з нього літери і приєднуючи до результату зліва. Змінній y, яка містить результат, спочатку присвоюється значення порожнього рядка. Значення змінної циклу змінюється від 1 (першого символу слова) до довжини рядка, що вводиться (номера останнього символу слова). Програма має вигляд

```
program E34;
var x, y: string[10];   i: integer;
begin
  write ('введіть слово');
  readln (x);
  y := '';{присвоєння результату початкового значення - порожнього слова}
  for i := 1 to length( x ) do
  y := copy( x, i, 1 ) + y; {приєднання літери, що копіюється, зліва}
  writeln;
  writeln(y)
end.
```

Пошук підрядка в рядку. Функція пошуку визначає, з якої позиції (номера символу) один рядок (підрядок) міститься в іншому (даному рядку). Опис функції мовою Паскаль має вигляд

pos (підрядок, вихідний рядок)

Якщо входження підрядка в рядок має місце, то результатом роботи функції буде .номер символу у вихідному рядку, із якого починається підрядок. Якщо входження немає, то результат — нуль. Аргументи функції можуть бути рядковими виразами.

Вставляння в рядок. В один рядок можна вставити інший рядок, вказавши номер символу, починаючи з якого здійснюється вставлення. Опис мовою Паскаль процедури вставлення має вигляд

insert (рядок _ що _ вставляється, вихідний _ рядок,
цілочисловий _ вираз);

Вхідні дані процедури — рядок, що вставляється, рядок, в який здійснюється вставляння, і цілочисловий вираз, який задає позицію, із якої слід починати вставлення. Рядки також можуть бути задані рядковими виразами. Результат роботи процедури поміщається у вихідний рядок, і рядок при цьому «розширюється». Якщо довжина вставки разом із довжиною вихідного рядка перевищує допустиму довжину вихідного рядка, то вставка скорочується справа до допустимої довжини.

Вилучення частини рядка. Частину рядка можна вилучити, і рядок при цьому «стискується». Для вилучення необхідно вказати рядок (у вигляді рядкового виразу), номер початкового символу частини рядка, що вилучається, і кількість символів, які потрібно вилучити. Опис мовою Паскаль процедури вилучення має вигляд

delete (рядок, початковий номер, кількість символів);

Розглянемо приклад заміни літери в слові. Перетворимо слово «форма» на слово «фірма». Програма матиме вигляд:

```
program E35;
var x : string [10];
begin
  x := 'форма';
  insert ('i', x,2); { вставка літери 'і', маємо слово 'фіорма' }
  delete ( x, 3, 1); { вилучення третьої літери -'о'}
```

```
  writeln(x)
end.
```

Приклад програми послівного перекладу з англійської мови. Припустімо, треба побудувати програму-перекладач, яка б, не враховуючи правил граматики, просто перекладала кожне слово речення, що вводиться. Оскільки програма демонстраційна, то використовуються невеликі словники — всього по 10 слів. Однак у разі розширення словників програму можна використовувати і для реального послівного перекладу. Ідея виділення слів із речення, що вводиться, ґрунтується на тому, що слова розділяються, як правило, пропусками, і вирізається слово між двома пропусками. Далі при звертанні до словника відшукується збіг виділеного слова зі словом словника. При знаходженні такого збігу зі словника перекладів друкується слово з таким самим індексом елемента, як і у словнику англійських слів. Така ідея пошуку може бути використана і в інших таблицях, коли, наприклад, за назвою хімічного елемента відшукується його маса.

Розглянемо етапи виконання програми на прикладі перекладу речення «I like a cat». Для виділення слів із речення використовуватимемо два вказівники. Перший з них — змінна *m,* її значення завжди 1, оскільки вона вказує на перший символ слова, що вирізається (копіюється) із речення. Другий вказівник — значення змінної *k,* яке завжди вказує позицію пропуску за виділеним словом і є його номером. Для першого слова речення перше значення змінної *k* — це 2. Функція пошуку підрядна в рядку працює таким чином, що пошук входження завжди здійснюється з першої позиції вихідного рядка. Якщо ми не змінюватимемо вихідний рядок, то будемо завжди знаходити перший пробіл. Тому після кожного виділення слова з рядка рядок необхідно «зрізувати», залишаючи тільки неопрацьовану частину рядка. Таке «зрізування» проводиться за допомогою оператора

a := **copy** *(a,* k, n – k + 1);

де n – *k* + 1 — довжина частини рядка, що залишилась після виділення чергового слова. Так, після виділення першого слова з речення (слова «I»), яке буде присвоєно змінній *x,* рядок a набуде вигляду «like a cat». При пошуку в словнику англійських слів (масив *e*) значення змінної *x* порівнюється з кожним словом словника. Якщо відбувся збіг, то друкується слово з словника українських слів (масив r*)* з таким самим порядковим номером, як і в англійському словнику. Якщо весь словник переглянуто, але слово не знайдено, то друкується повідомлення «слова в словнику немає». Програма при обробці визначених і невизначених артиклів пропускає їх і переходить до обробки наступного слова речення.

Програма має вигляд:

```
program E36;
label 1,2,3;
const p = 10; { кількість слів у словнику }
var e,r: array [1..p] of string[10];  x: string[10];
i,k,m,n : integer;  a,b: string;
begin
{ формування словників }
  writeln('введіть ', p, ' англійських слів');
  for i := 1 to p do
  readln (e[i]);
  writeln('введіть українські слова'); for i := 1 to p do
  readln(r[i]);
  { введення слів і виділення слів із речення }
  writeln('введіть речення');
  readln(b);
  a := b; { збереження вихідного речення }
  m := 1; n := length(a); {n - довжина речення}
  1:k := pos (' ', a);  { пошук пропуску }
  if k = 0 then k := length(a) + 1; { виставлення пропуску
                                      після останнього слова }
  x := copy(a, m, k - m); {вирізання слова довжиною k-m
```

```
символів}
  if ( x = 'a') or ( x = 'the') then goto 3; {обробка артиклів}
{пошук слова в словнику}
  for i := 1 to p do
  if e [i] = x then goto 2;
  writeln('слова в словнику немає');
  goto 3;
  2:writeln(r[i]); {виведення слова з українського словника}
  3:k:= k + 1; {перехід до наступного слова в реченні}
  a := copy(a, k, n - k + 1);
  if a <> '' then goto 1
end.
```

ЗАПИТАННЯ І ЗАВДАННЯ

1. Чим відрізняється символьний тип даних від рядкового?
2. Використовуючи символьний масив, визначте, скільки слів у даному тексті.
3. Використовуючи символьний масив, порахуйте, скільки літер «а» в даному слові.
4. Використовуючи засоби обробки рядків, виправте слово «вилисипидисти».
5. Використовуючи ідею програми Е34 обернення слова, подвойте кожну букву в даному слові.
6. Використовуючи програму обернення слова Е34, визначте, чи є дане слово паліндромом, тобто таким словом, яке однаково читається в обох напрямках (наприклад, «потоп», «кок» і т.д.)
7. Дано рядок із кількома комами. Знайдіть текст між першою і другою комами. Розв'яжіть задачу з використанням масиву символів і рядка символів.

2.14. Комп'ютерна графіка

Засоби мови Паскаль дозволяють будувати зображення на екрані дисплея. Для цього використовується спеціальна

бібліотека підпрограм, яка називається Graph. До неї входять графічні процедури і функції для побудови різних за формою фігур і ліній, а також засоби організації графічного режиму. Ці засоби призначені для аналізу можливостей використовуваного дисплея, розподілу поля екрана на різну кількість дрібних квадратиків, кожний з яких вважається окремою точкою зображення і називається пікселем. Для дисплея типу VGA кількість точок може бути 640x480, а кількість використовуваних кольорів — 16. Складніші засоби Паскаля дозволяють збільшити кількість кольорів до 256. Колір, як і координати екранної точки, задається цілим числом. Початок координат екрана знаходиться в лівому верхньому куті (точка (0,0)), вісь *Ox* направлена вправо, *Oy* — вниз (рис. 44). Цю особливість направленості осі O*y* необхідно враховувати при побудові зображень, особливо графіків функцій і діаграм.

При побудові зображень також використовуються так звані графічні примітиви — точка, відрізок, дуга.

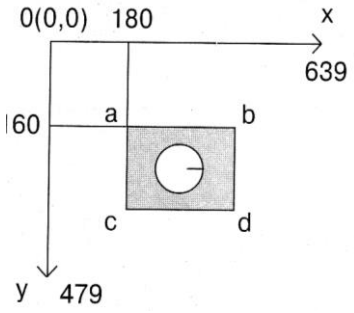


Рис. 44

Точка. Кожне зображення можна будувати по точках, вказуючи для кожної точки її координати й колір. Для видачі точки на екран використовується процедура

PutPixel(x, *y*, номер_кольору);

де *x, y* — координати точки.

За допомогою цього оператора можна побудувати графік функції.

Побудова графіка функції. Щоб побудувати графік функції *y=f(x),* необхідно спочатку вказати відрізок, на якому задана функція, і відобразити цей відрізок на вісь *Ox* екрана дисплея. Точці *x* відрізка [a, b] *з* області задання функції повинна відповідати точка x, екрана дисплея, а точці *y=f(x)* (значенню функції) — точка y_1 екрана (рис. 45). Таким чином, відбувається масштабування графіка, що

дозволяє відображати на екран будь-який відрізок області задання функції. Тут необхідно враховувати роздільну здатність екрана і добирати масштаби так, щоб побачити поведінку функції. Точки на екрані, на відміну від точок геометричної прямої, мають дискретні значення. Точки осі *Ox* нумеруються від 0 до 639, тобто x =0, 1, 2, 3, ..., 639.

З пропорції

$$(x-a)/(b-a)=x/640$$

дістанемо розрахункову формулу для обчислення аргументу функції, що відповідає заданій точці екрана:

$$x = a+x_1(b-a)/.x_{max}$$

де x_{max}= 640. Для одержаного значення аргументу обчислюється значення функції:

$$y = f(x).$$

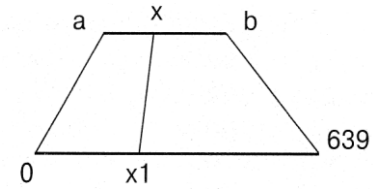


Рис.45. Встановлення відповідності між точками відрізка [a, b]і віссю O_x екрана дисплея.

Тепер необхідно помножити одержане значення функції на коефіцієнт масштабування, який показує, скільки пікселів (фізичних точок екрана) міститься у вибраній одиниці масштабу:

$$k = x_{max}/(b-a).$$

Використовуючи цей масштабний коефіцієнт, знаходимо значення координати у на екрані:

$$y_1 = yx_{max}/(b —a).$$

Тепер слід врахувати особливість розташування осі *Oy* на екрані і «перевернути» графік з урахуванням традиційного положення осей координат, а також змістити графік так, щоб його зручно було аналізувати, наприклад,

$$y_1 = y_{max}/2 - y_1.$$

(у цьому випадку нуль осі *Oy* буде знаходитися на середині екрана).

З одержаними координатами x, і y, точка виводиться на екран. Оскільки реальні координати графіка функції, із якими вона обчислюється, — дійсні числа, а координати точок екрана — цілі, то при обчисленні y_i необхідно округлювати отриманий результат до найближчого цілого числа. Округлення виконується за допомогою функції **trunc,** аргументом якої є вираз дійсного типу:

y_1 **:= trunc(y *** xmax / (b - a));

Програма побудови графічних зображень повинна починатися з оператора підключення бібліотеки графічних процедур і спеціальної бібліотеки, яка дозволяє використовувати функції операційної системи (бібліотеки **Crt).** Цей оператор розміщується відразу за заголовком програми і має вигляд:

uses Graph, Crt;

Сама програма складається з описів процедур побудови зображень, опису функції, для якої будується графік, і опису спеціальної процедури установки графічного режиму. Головна програма при цьому містить оператори виклику цих процедур і оператор який затримує зображення на екрані до натискання будь-якої клавіші (інакше після побудови останньої точки зображення воно зникає).

```
repeat until keypressed; {виконати пустий цикл
                          до натиснення будь-якої клавіші }
```

Програма побудови графіка функції *y= sin(x)* на відрізку [—π;π] має такий вигляд:

```
program E37;
uses Graph, Crt;
procedure Init;
var gr, gm: integer;
begin
  gr := 0;  { автоматичне розпізнавання типу дисплея }
  lnitGraph( gr, gm, ' '); {файл egavga.bgi знаходиться
                            в одному каталозі з turbo.exe}
  if GraphResult <> grOk
```

```
  then Halt (1);   {перевірка правильності
                    встановлення графічного режиму}
end;
function f (x: real): real; { обчислення функції f(x)}
begin
  f := sin(x);
end;
procedure grafic;
var xmax, ymax, x1, y1: integer; x,y,a,b:real;
begin
  xmax := 640;  ymax := 480;  a := -pi;  b := pi;
  for x1 := 1 to xmax do
  begin
  x := a + x1 * (b - a) / xmax;
  y := f(x); { виклик підпрограми-функції}
  y1 := trunc(y * xmax / ( b - a));
  y1 := ymax div 2 - y1;
PutPixel(x1, y1, 2) { виведення точки зеленим кольором }
end
end;
begin {головна програма}
 Init; { виклик процедури установки графічногорежиму}
 Grafic; { виклик процедури побудови графіка функції}
repeat until keypressed
end.
```

При побудові складних рисунків, які виводяться поточково, можна використовувати прийом мозаїки: нарисувати рисунок по клітинках (рис.46) і заповнити таблицю цілих чисел: кожне число відповідає номеру кольору відповідної клітинки, а координати клітинки — індексам елементів таблиці 14.

Таблиця 14

0	15	0	15
15	0	15	0
15	15	15	0

Задання кольору. Деякі графічні оператори, такі, як

побудова точки, вміщують параметр кольору, але для багатьох інших використовується поточний колір, який визначений останнім оператором задання кольору, що має вигляд

SetColor (колір);

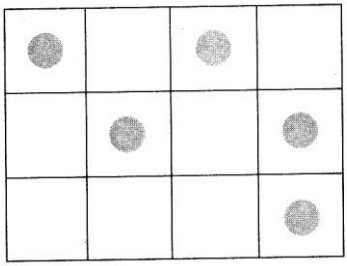


Рис. 46. Рисунок по клітинках

Наводимо таблицю кольорів (таблиця 15), де кожному кольору відповідає певний номер.

Таблиця 15

Чорний	Синій	Зелений	Бірюзовий
0	1	2	3
Червоний	Малиновий	Коричневий	Сірий-с
4	5	6	7
Сірий-т	Голубий	Зелений-с	Бірюзовий-с
8	9	10	11
Червоний-с	Малиновий-с	Жовтий	Білий
12	13	14	15

Відрізок. Для побудови відрізка необхідно вказати координати його кінців. Відрізок зображується поточним кольором. Оператор побудови відрізка має вигляд

Line (x1, y1, x2**,** y2);

При побудові зображення комп'ютер ніби малює променем. Промінь зупиняється в останній точці зображення, координати якої запам'ятовуються. Можна побудувати відрізок, вказавши або зміщення від цієї точки вздовж осей *Ox* і *Oy,* або координати кінця відрізка (початок там, де зупинився промінь). Зміщення вказується у вигляді цілого числа: додатне означає

збільшення останньої координати на величину зміщення, від'ємне — відповідне зменшення координати. Зміщення зручно використовувати для побудови зображення у відносних координатах, коли достатньо вказати координати першої точки, а координати решти точок обчислюються. Отже, можна переміщувати картинку по екрану.

Оператор побудови відрізка з вказуванням зміщення має вигляд

LineRel (dx, dy);

Для побудови відрізка з вказуванням останньої його точки використовується такий оператор:

LineTo (x, y);

Цей оператор зручно використовувати для побудови графіка функції, яка або різко зростає, або різко спадає. У такому разі зображення, побудоване по точках, має розрив, оскільки для двох сусідніх аргументів два значення функції знаходяться далеко одне від одного. Перша точка графіка виводиться за допомогою оператора **PutPixel,** а всі інші точки, які обчислюються й будуються в циклі, — оператора **LineTo.**

Промінь, який будує зображення, може переміщуватись, не залишаючи сліду. Це можна зробити, використовуючи оператори, аналогічні двом попереднім:

MoveRel (dx, dy);

MoveTo (x, y);

Аналогічно до відрізка будується прямокутник: для його побудови необхідно вказати координати кінців будь-якої діагоналі. Прямокутник може бути побудований у вигляді контуру або зафарбованої фігури.

Для контурного зображення прямокутника використовується оператор

Rectangle (x1, y1, x2, y2);

а для зафарбованого —

Bar(x1, y1**,** x2**,** y2);

Побудова стовпчикової діаграми. На рис. 47 подана стовпчикова діаграма падіння успішності учня по чвертях навчального року. У першій чверті успішність становила 75%, у другій — 50% , у третій — 25%. Побудуємо таку діаграму за допомогою програми, розмістивши найвищий стовпець у центральній частині екрана (його довжина — 480:3 = 160 пікселів) і розрахувавши решту пропорційно їх значенням відносно цього розміру. За шириною вся діаграма займає також 1/3 частину екрана, тобто початкове значення координати *x* дорівнює 210 пікселів, ширина стовпців по 35 і відстань між ними також 35 пікселів. Для обчислення висоти другого стовпця складемо пропорцію 75:50 = 160:x, Звідси *x*= 106 (з округленням до цілого). Висота другого стовпця 53 пікселі.

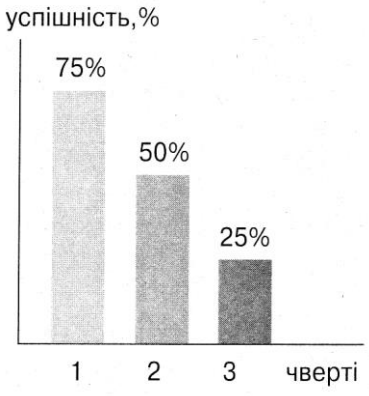


Рис. 47. Стовпчаста діаграма успішності учня по чвертях

Оскільки нижні основи прямокутників знаходяться на одному рівні, на якому *y*— 320, то інші координати (вздовж осі Oy) обчислюються відніманням від 320 висоти відповідного стовпця. Розміщення тексту на зображенні здійснюється за допомогою оператора

OutTextXY (x, y, 'текст');

де *x, y* — координати, починаючи з яких буде виведено текст.

Програма побудови стовпчастої діаграми має таку саму структуру, як і програма Е36:

```
program E38;
uses Graph, Crt;
procedure Init;
var gr, gm: integer;
```

```
begin
  lnitGraph( gr, gm, ' '); {файл egavga.bgi знаходиться
                            в одному каталозі з turbo.exe}
  if GraphResult <> grOk
then Halt (1)   {перевірка правильності встановлення графічного режиму}
end;
procedure Diagram 1;
begin
SetColor(14); Bar(210, 160, 245, 320);
SetColor(IO); Bar(210 + 2*35, 320 - 106, 210 + 3*35,320);
SetColor(2);  Bar(210 + 4*35, 320 - 53, 210 + 5*35, 320);
OutTextXY(210, 320 + 30, 'Успішність по чвертях')
end;
begin {головна програма}
  Init; {виклик процедури встановлення графічного режиму}
  Diagram 1;
  repeat until keypressed
end.
```

Побудова дуги. Для побудови дуги потрібно вказати центр відповідного кола *(x, y),* початковий кут, кінцевий кут і радіус. Кути відраховуються проти годинникової стрілки, їх величина вказується в градусах (від 0 до 360). Відлік кута проводиться від направленого вправо по горизонталі радіуса. Оператор побудови дуги має вигляд

Arc (x, y, початковий кут, кінцевий кут, радіус);

Для зафарбованого сектора використовується оператор із такими самими параметрами:

PieSlice (x, y, початковий_кут, кінцевий_кут, радіус);

Якщо вказати значення кутів відповідно від 0 до 360, то одержимо коло або зафарбований поточним кольором круг. Коло можна також зобразити оператором

Circle (x, y, r**);**

Розглянемо приклад побудови з кіл «рогу достатку».

Центри кіл переміщуються по колу, що є траєкторією точки, одна координата якої задається косинусом, а друга — синусом того самого кута *x/k.* При *k* = 20 маємо замкнене коло, при *k*= 10 — два кола, при A = 30 — півколо.

Програма матиме вигляд

```
program E39;
uses Graph, Crt;
procedure Init;
var gr, gm: integer;
begin
gr := 0;   { автоматичне розпізнавання типу дисплея }
InitGraph ( gr, gm, ' '); {файл egavga.bgi знаходиться
                           в одному каталозі з turbo.exe}
if GraphResult <> grOk
then Halt(1)  {перевірка правильності
                встановлення графічного режиму }
end;
procedure Rog;
var xmax, ymax, x1, y1, x, y, r: integer;
begin
  xmax := 640; ymax := 480;
  for x1 := 1 to 125 do
begin
  x := trunc(xmax /2 + ymax/2 * cos( x1/20 ));
  y := trunc(ymax /2 + xmax/4 * sin( x1/20 ));
  r := trunc(ymax /4 - x1 Y,
  Circle (x, y, r)
end
end;
begin {головна програма}
Init; {виклик процедури встановлення графічного режиму}
          Rog;
          repeat until keypressed
end.
```

Побудова кругової діаграми.

Діаграми такого типу являють собою круги, поділені на сектори різної площі. Площа кожного сектора відповідає деякій частині круга, як правило, у відсотковому відношенні, а за одиницю (ціле) береться площа всього круга. Замість площі сектора можна обчислити його кутовий розмір. Побудуємо кругову діаграму успішності учня, що відповідає діаграмі рис. 48.

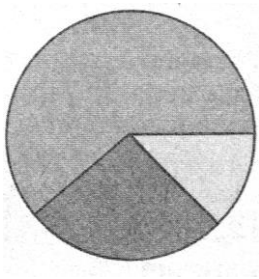


Рис. 48. Кругова діаграма успішності

Програма має вигляд:

```
program E40;
uses Graph, Crt;
procedure Init;
var gr, gm: integer;
begin
  gr := 0;   { автоматичне розпізнавання типу дисплея }
  InitGraph (gr, gm, ' '); {файл egavga.bgi знаходиться
                            в одному каталозі з turbo.exe}
  if GraphResult < > grOk
  then Halt (1)  {перевірка правильності
                  встановлення графічного режиму }
  end;
  procedure Diagram2;
  var xmax, ymax, x1, y1, x, y, r: integer;
begin
  xmax := 640; ymax := 480;
  x := 320;  y := 240;  r := 200; {координати центра та радіус
діаграми}
  x1 := trunc( 75 * 36/15 );{обчислення довжини першої
дуги}
  SetColor (14);    PieSlice ( x, y, 0, x1, r );
```

```
    y1 :=trunc(50 * 36/15) + x1;
    SetColor (10);
    PieSlice( x, y, x1 + 1, y1, r); {друга дуга + перша дуга}
    SetColor (2);
    PieSlice( x, y, y1 +1, 360, r ) {замикання кола}
  end;
  begin {головна програма}
    Init; {виклик процедури встановлення графічного режиму}
    Diagram2;
    repeat until keypressed
  end.
```

Запитання і завдання

1. Як розташовані осі координат на екрані дисплея?
2. Що таке графічні примітиви?
3. Як зафарбувати один піксел екрана?
4. Як відобразити відрізок [*a, b]* на всю ширину екрана?
5. Скільки пікселів буде містити одиниця виміру на відображеному на екрані відрізку *[a*, b]?
6. Як розмістити графік на екрані з врахуванням направленості осі *Оу в* протилежному від традиційного напрямку?
7. Якого типу дані відповідають екранним координатам?
8. В якій бібліотеці Паскаля містяться графічні підпрограми і як підключити цю бібліотеку при виконанні програми?
9. Як затримати зображення на екрані після повної його побудови?
10. Як задати колір для побудови ліній, прямокутників і дуг?
11. За допомогою якого оператора можна побудувати відрізок, використовуючи тільки одну пару координат?
12. Як перемістити промінь, що рисує, по екрану так, щоб не було сліду?
13. Як побудувати графік функції, яка різко зростає або

різко спадає?

14. За допомогою яких операторів можна побудувати контурний і зафарбований прямокутники?

15. Як побудувати лінійну діаграму?

16. Як підписати рисунок?

17. Як побудувати дугу кола? Як за допомогою оператора побудови дуги побудувати коло?

18. Як зобразити зафарбований сектор?

19. Для чого використовується функція **trunc?**

20. Що таке кругова діаграма?

21. У програму побудови графіка функції додати виведення на екран координатних осей і позначок одиниць виміру на них.

22. За допомогою програми побудови графіка функції побудувати графіки $\log_2 x$, tg x, cos x на різних частинах областей визначення цих функцій.

23. Програму побудови стовпчастої діаграми доповнити так, щоб на зображенні були всі підписи, як на рис. 47, і в кожному секторі помістити прямокутник білого кольору з необхідним текстом.

24. Побудувати концентричні кола, зафарбувавши кожне новим кольором.

25. Побудувати ялинку з *n* трикутників, визначивши її попередньо рекурсивно і використавши в програмі рекурсивну графічну процедуру.

26. Використовуючи ідею дитячої гри «мозаїка», побудувати зображення різнокольорового метелика, зберігши його попередньо у вигляді таблиці цілих чисел (значень кольорів) і виводячи потім цю таблицю по точках на екран, застосовуючи подвійний цикл.

27. Використовуючи аркуш у клітинку з рамкою 32x24 клітинки (макет екрана), побудувати довільне зображення і потім повторити його за допомогою програми.

2.15. Записи

Раніше запис був визначений як послідовність байтів на носії, обмежена з двох боків спеціальними позначками. Таке визначення дає уявлення про запис як про одиницю обміну між зовнішньою й оперативною пам'яттю комп'ютера. Однак сам запис може бути складною структурою, яка містить різні дані. Запис може, наприклад, відповідати рядку відомості заробітної плати, в якій вказані прізвище і кілька чисел, або рядку класного журналу, де також стоять прізвища й оцінки. Отже, запис є складною конструкцією. Тому в Паскалі й інших програмних системах слово «запис» має подвійний зміст: це і складна структура, і одиниця даних на носії (наприклад, диска).

Запис — це сукупність різнорідних даних, що описуються та обробляються як одне ціле.

Дані, із яких складається запис, називаються його полями. Поля можуть бути як простими даними, так і складеними, наприклад масивами або записами. За допомогою записів зручно описувати властивості об'єктів, зберігаючи їх разом. Із записів складаються бази даних, що включають описи кількох об'єктів.

Опис запису складається з ключового слова **record,** після якого вказуються імена полів і тип кожного поля. Тип поля відділяється від імені двокрапкою. Опис запису закінчується словом end і крапкою з комою. Записи описуються в розділі типів даних **type.** У цьому розділі вказується ім'я класу об'єктів (ім'я типу) і опис цього класу. Для кожного об'єкта класу є своє ім'я в розділі змінних **var** з описом даного типу. Це ім'я використовується далі в програмі.

Приклад 1. Об'єкт — фізичне тіло з параметрами a (довжина), *b* (ширина), *c* (висота). Його опис може мати вигляд а) коли для кожного поля вказано тип даного, або б) коли однотипні поля, що йдуть підряд, описані разом:

а) type z **= record**
a: **integer;**
b: **integer;**
c: **integer**

б) type z **= record**
a, b, c: **integer**
end;
var x: z;

```
end;
var x: z;
```

Приклад 2. Об'єкт — товар, який характеризується назвою і ціною:

```
type tovar = record
  sign: string[ 20];    price: real
end;
```

Приклад 3. Об'єкт — дата народження: день, місяць, рік. День можна вказати як діапазон значень — такий тип даних називається інтервальним. Він уже траплявся при описі масивів, де вказані інтервали номерів елементів. Цей тип можна використовувати для цілочислових і символьних даних в описах, а також як позначення оператора варіанта. Даними інтервального типу можна задавати значення констант у розділі **const:**

```
type
  date_of_birth = record
  day: 1 ..31;
  month: string[10];
  year: integer
end;
var date: date _ of_ birth;
```

Записи можна об'єднувати в масиви. Масив записів може бути описаний у розділі **type** або **var.** Для прикладу 2 розділ змінних може мати вигляд

```
var x: array [1..100] of tovar;   y: tovar;
```

x — масив записів, до кожного елемента якого використовується звичне звернення, наприклад, *x[i]; y* — проста змінна.

Для звернення до поля запису використовується складене ім'я, яке складається з двох імен, розділених крапкою. Перше *з* цих імен — ім'я змінної типу запис з розділу **var,** друге — ім'я поля цього запису з розділу **type.** Так, для товарів імена полів у

програмі мають вигляд:

y.sign , y.price, x[1].sign, x[i].price

Приклад 4. Припустімо, потрібно описати відомості про робітника підприємства: прізвище, посаду, дату народження й зарплату (таблиця 16).

Таблиця 16

Зміст відомостей	Прізвище	Посада	Дата народження	Зарплата
Ім'я поля запису	name	position	date	salary
Тип даних поля	string [20]	string [10]	Date _ of _ birth	real

Кожному полю запису потрібно спочатку присвоїти ім'я, потім визначити, який тип найзручніший для обробки цих даних. Описувані відомості включають структуру типу запис у вигляді поля дати народження, її також потрібно уточнити і описати в розділі типів раніше, ніж запис, що стосується робітника.

В описі даного запису використовується тип «день народження» *(date _of _ birth)* з прикладу 3. Загальний опис запису має вигляд:

```
type
date_of_birth = record
  day    : 1 .. 31;
  month : string[10];
  year   : integer
end;
worker = record
  name      : string[ 20 ];
  position   : stringf 10 ];
  date       : date_of_birth;
  salary    : real
end;
var x: array [ 1..7] of worker; w : worker;
```

Поле запису *date* містить запис із трьох полів. При формуванні імені поля цього внутрішнього запису необхідно використовувати потрійне ім'я: ім'я змінної розділу **var,** ім'я поля запису *worker* та ім'я поля запису *date _of _ birth.*

Наприклад, для змінної *w* звернення до місяця народження робітника в програмі матиме вигляд

w.date. month

У програмі введення й виведення запису виконується окремими полями, але можна присвоїти одному запису значення іншого, при цьому проходить копіювання ділянки пам'яті:

x[1] := w;

Оператор приєднання. Для обробки запису використовується оператор **with,** який дозволяє вказати один раз ім'я запису з розділу змінних, а потім, у всій області дії оператора, вказувати тільки імена полів цього запису з розділу **type.**

Після слова **with** можна написати кілька імен полів із розділу змінних.

Оператор приєднання має вигляд:

with список імен записів **do** оператор;

Оператор може бути простим або складеним, обмеженим операторними дужками. Список імен записів може складатись з одного імені.

Приклад 5. Дано масив записів, який містить відомості про робітників підприємства (див. приклад 4). Надрукуйте:

1) список бухгалтерів;

2) список робітників від 30 до 50 років;

3) середню заробітну плату по підприємству.

Як і в інших задачах, де сказано, що «дано n чисел» або що-небудь інше, вихідні дані необхідно описати і ввести в

комп'ютер. Для записів цієї задачі визначаються, як і в прикладі 4, їх структура, імена й типи полів. Потім визначається ім'я масиву записів, який використовується в програмі, і допоміжні змінні, а також імена результатів.

Програма розв'язування задачі має вигляд

```
program E41;
const n = 3;
type date_of_birth = record
  day: 1 ..31;
  month: 1..12;
  year: integer
end;
worker = record
name: string[20];
position: string[10];
date: date_of_birth;
salary: real
end;
var x: array [ 1..n ] of worker; i,j,g:integer;
          S:real; p: string[10];
begin
{формування масиву записів}
  for i:= 1 to n do
    with x [i] do
begin
  writeln(' відомості про ', i, 'робітника');
  writeln('прізвище, ініціали');
  writeln('nocafla');
  readln(position);
  writeln( 'число, місяць і рік народження');
  readln(date.day, date. month, date, year);
  writeln(date.day, date. month, date. year);
  writeln( 'зарплата' ) ;
  readln(salary)
end;
{розв'язування задачі 1 - друкування списку бухгалтерів}
```

```
p := 'бухгалтер';
j := 0; {лічильник рядків списку - кількість бухгалтерів}
for i := 1 to n do
  with x[i] do
    if p = position
   then
     begin
        writeln(j, '.', name); {після номера в списку
                                друкується крапка та ім'я}
     end;
{розв'язування задачі 2 - список робітників від 30 до 50 років}
 writeln( 'список робітників від 30 до 50 років ');
 j:=0;
 write(' введіть поточний рік у вигляді чотиризначного
числа ');
          readln(g);
          for i := 1 to n do
            with x[i] do
              begin
       if ( g - date. year < 50 ) and ( g - date. year > 30 )
       then
         begin
            j: = j+1
            writein(j, '.', name)
         end
end;
{розв'язування задачі - обчислення середньої зарплати}
  S:=0;
  for i := 1 to n do
          S := S + x[i].salary;
  writeln('середня зарплата =', S/n:10:2)
end.
```

ЗАПИТАННЯ І ЗАВДАННЯ

1. Як розуміється слово «запис»?
2. Як звернутися в програмі до поля запису?

3. Що необхідно зробити, щоб описати в програмі об'єкт, характеристиками якого є різнотипні дані?

4. Яке ім'я даного, що містить відомості про те, якого числа народився робітник підприємства?

5. Поясніть, що означають імена: x[2].position, x[5].date.month, w.name.

6. Для чого використовується оператор with?

7. Опишіть об'єкт «учень 11 класу», використовуючи запис.

8. Створіть базу даних для свого класу і з її допомогою отримайте відомості:

а) про найстаршого учня в класі;

б) про наймолодшого учня;

в) про оцінки кожного учня з усіх предметів;

г) про відмінників;

д) про найбільш відстаючого учня;

е) про середній бал для кожного учня за оцінками останньої чверті;

ж) про середню успішність класу з усіх предметів.

Файли в Паскалі

Файл у Паскалі складається з однотипних даних. З даними файла можна проводити дві операції: запис або читання. У Паскалі проводиться обробка послідовних файлів, в яких дані записуються або зчитуються одне за одним. Запис можна читати, пропускаючи попередні, якщо відомий його порядковий номер у файлі. Для того щоб із даними файла проводити дії, файл потрібно відкрити для відповідної операції.

Розглянемо етапи, які необхідно виконати для кожної операції при роботі з файлом.

Операція запису. Запис у файл означає введення до нього нових даних. Файл розміщується на носії, як правило, на магнітному диску. Дане для занесення у файл формується в оперативній пам'яті як значення деякої змінної. Операцією запису у файл це дане копіюється з оперативної пам'яті до зовнішньої. Отже, форма подання даного, його тип і структура

повинні бути однакові і для записів файла, і для змінної, Із якої це дане копіюється.

1. Опис файла. Опис файла може бути в розділі типів або в розділі змінних. Нехай файл *f* складається *з* цілих чисел. Його опис має вигляд:

var f: file **of integer;** ***a*: integer;**

де *а* — компонента файла, даного того самого типу, що й записи файла. Тип даних файла вказується після слова **of** в описі - це може бути числовий або символьний тип, масив або запис. Складений тип запису файла необхідно попередньо описати в розділі **type.**

2. Встановлення відповідності між логічним і фізичним Іменами файла. Логічне ім'я - це ім'я змінної з розділу **var**. За цим іменем до файла звертаються в програмі. Фізичне ім'я — це ім'я, під яким файл записаний на диску. Оператор встановлення відповідності між іменами файлів має такий вигляд:

assign (логічне ім'я файла, фізичне ім'я);

Для наведеного опису цей оператор має вигляд:

assign (f, 'F.DAT');

Фізичне ім'я взято в апострофи. Воно з'явиться в змісті диска в тому самому каталозі, в якому знаходиться файл *turbo.exe.*

3. Відкриття файла для операції «запис». Ця дія виконується оператором

rewrite (f);

При відкритті файла для занесення до нього даних на диску з'являються два спеціальні записи: початок файла, який містить фізичне ім'я, і ознака кінця файла. Кожне відкриття файла для

запису означає створення файла. Якщо для операції «запис» відкрити файл з уже існуючими даними, то всі дані файла зникнуть. Тому відкривати для запису можна тільки файли з новими іменами. При занесенні даних до файла вони будуть розміщуватись у кінці файла.

Файл може містити довільну кількість даних. Обмеження розміру файла в програмі ніяк не оговорюється. В оперативній пам'яті досить однієї області, яка збігається за форматом із записами файла, а на диску файл може бути такого розміру, скільки є вільного простору в момент його створення.

Запис даних у файл здійснюється за допомогою оператора

write (f, a);

Приклад 1. Нехай потрібно створити файл з 10 цілих чисел. Програма має вигляд

```
program E42;
var f: file of integer;  a,i: integer;
begin
  assign(f, 'F.DAT'); rewrite(f);
  writeln('введіть 10 цілих чисел, після кожного');
  writeln('натискайте Enter');
  for i := 1 to 10 do
  begin
        readln (a);
        write (f,a) end end.
```

Операція читання. Для читання даних із файла його необхідно описати, встановити відповідність між логічним і фізичним іменами, відкрити для читання і зчитувати дані. Перші два кроки — опис і встановлення відповідності імен — такі самі, як і для операції запису. Якщо з файлом виконуються різні операції, то перед виконанням наступної операції його необхідно закрити оператором

close (f);

Відкриття файла для читання проводиться оператором

reset (f);

Для читання даних із файла використовується оператор

read (f, a);

Після створення файла і кількох перетворень може бути невідома кількість його записів. Тому при читанні даних із файла зручно використовувати спеціальну функцію, яка контролює ознаку кінця файла. Ця функція набуває значення «істинно», якщо трапляється ознака кінця файла, і «хибно», якщо прочитано інший запис. Оскільки при відкритті файла для читання вже зчитується перший його запис, який містить ім'я файла, то можна поставити контроль ознаки кінця файла, навіть не зчитавши жодного запису оператором **read(/,** a);

Аналіз ознаки кінця файла виконується функцією

eof (f)

Оскільки кількість записів у файлі невідома, то використовувати при читанні даних файла цикл «перелік» не можна, використовується цикл «доки». Його заголовок

while not eof (f) do

треба розуміти так: поки не зустрілась ознака кінця файла, виконувати цикл.

Приклад 2. Дано файл цілих чисел. Порахувати кількість додатних, від'ємних і нульових елементів у файлі. Використаємо у даному прикладі програму створення файла цілих чисел Е40:

```
program E43;
const k = 5;
var f: file of integer;  a,i, n, p, z: integer;
begin
  assign(f, 'F.DAT');
  rewrite(f); {створення файла}
  writeln('введіть ', k,' цілих чисел, після кожного натискайте Enter');
```

```
    for i := 1 to k do begin readln(a);
    write(f, a) end;
    close(f); {закриття файла для операції запису}
    {розв'язування задачі - підрахунок різних елементів}
n:=0; p := 0; z := 0;
    { n - від'ємні, p - додатні, z -нулі} reset(f);
    while not eof(f) do begin read(f,a);
    if a = 0 then z := z +1;
    if a < 0 then n := n + 1;
    if a > 0 then p := p + 1;
    end;
  write('n =', n, 'z=',z, '   p=', p);
  end.
```

Приклад 3. Нехай потрібно розширити даний файл, додавши до нього нові дані. Як відомо, файл із даними не можна відкривати для запису, тому для розв'язування подібних задач необхідно використовувати допоміжний файл.

Розв'язування задачі розширення файла складається з таких етапів:

1) відкрити даний файл /для читання, а допоміжний *д* — для запису;

2) читати дане з вихідного файла / і відразу записувати його у файл *д;*

3) після закінчення переписування даних закрити файл /;

4) вводити нові дані з клавіатури і записувати їх у файл *д,* додаючи до вже існуючих там даних файла /;

5) закрити файл *д;*

6) відкрити файл / для запису, а файл *д* — для читання;

7) читати дані з файла *д і* записувати їх у файл /.

Таким чином, у файл / до наявних там даних додаються нові. Якщо потрібно вставити нові дані в середину файла, то треба в другому пункті розв'язування контролювати зчитані дані і, дійшовши до місця вставлення, припинити читання, записати потрібні дані до файла *д, а* потім дописати туди решту з вихідного файла (пункти 3 і 4), далі виконати пункти 5, 6, 7.

Програма доповнення даними файлів (використовуємо готовий файл / із прикладу Е40) має вигляд:

```
program E44;
var f,g :file of integer;  a: integer;
begin
  assign(f, 'F.DAT');
  assign(g, 'G.DAT');
{ перезапис даних з вихідного файла в допоміжний}
  reset(f);
  rewrite(g);
  while not eof(f) do
  begin
        read(f,a);
        write (g,a)
  end;
close(f);
{ додання даних до допоміжного файла}
readln (a);
while a <> 0 do { ознака закінчення введення нових даних - нуль}
 begin
        write (g,a);
        readln (a)
  end;
close(g);
reset(g);
rewrite (f);
{ перезапис даних назад у вихідний файл}
writeln('модифікований файл');
while not eof(g) do
begin read(g.a);
        write(f,a);
        writeln(a)
  end
end.
```

ЗАПИТАННЯ І ЗАВДАННЯ

1. Які операції можна проводити з даними файла?

2. Чому другий параметр а операторів **write(f, a) і read(f,** а) повинен бути такого самого типу, як і дані файла?

3. Що таке фізичне ім'я файла, чим воно відрізняється від логічного імені?

4. Як відкрити файл для запису?

5. Що буде, якщо раніше створений файл із даними відкрити для запису?

6. Чим обмежена кількість даних у файлі?

7. Чим відрізняється файл від масиву?

8. Як прочитати дані з файла, не знаючи кількості цих даних?

9. Як додати дані до існуючого файла?

10. Для цілочислового файла знайти найбільший елемент даних.

11. Для файла цілих чисел переписати додатні числа в один додатковий файл, а від'ємні - в інший.

12. Для задачі 8 попереднього параграфа створіть файл із записами про учнів класу і виконайте всі пункти завдання, використовуючи дані файла.