Á³áë³îòåêà «Øê³ëüíîãî ñâ³òó»
Çàñíîâàíà ó 2003 ð.

Андрій Ставровський
Ірина Скляр

ПРОГРАМУЄМО ПРАВИЛЬНО

Посібник

У двох частинах
Частина 1

²íôîðìàòèêà. Á³áë³îòåêà

Êè¿â
«Øê³ëüíèé ñâ³ò»
2007

ББК 32.973-018я721
С76

Ставровський, Андрій.

С76 Програмуємо правильно: Посіб. : У 2 ч. / А. Ставровський, І. Скляр. — К. : Шк. світ, 2007. — Ч. 1. — 128 с. — (Б-ка «Шк. світу»).
ISBN 978-966-451-000-1.
ISBN 978-966-451-075-9.

Посібник містить початковий курс програмування в рамках шкільної дисципліни «Інформатика». У посібнику представлено основи алгоритмізації та програмування мовою Паскаль, міститься необхідний теоретичний матеріал, багато практичних прикладів та докладних розв'язань типових задач.

Призначено для учнів, які вивчають інформатику за програмою фізико-математичних шкіл, а також для вчителів інформатики, студентів педагогічних ВНЗ і всіх, хто бажає навчитися програмувати.

ББК 32.973-018я721

ISBN 978-966-451-000-1(б-ка «Шк. світу»)
ISBN 978-966-451-075-9

ЗМІСТ

Розділ 4. Розгалуження, вибір варіантів та складені оператори

Розділ 5. Циклічні алгоритми

Розділ 6. Допоміжні алгоритми, або підпрограми

ðîçä³ë 1

ÎÑÍÎÂÍ² ÏÎÍßÒÒß ÀËÃÎÐÈÒÌ²ÇÀÖ²¯

1. ПРОГРАМИ, ДАНІ, ІНФОРМАЦІЙНІ МОДЕЛІ

Сучасна людина використовує комп'ютер для розв'язання різноманітних задач — від виконання простих арифметичних дій до моделювання атмосферних явищ та керування польотами в космос. Насправді, все, що «вміє» комп'ютер, — це ***виконання програм***. Щоб комп'ютер міг розв'язувати потрібну задачу, необхідно спочатку створити відповідну програму, а потім запустити її на виконання.

Програма — це опис тих дій, які має виконати комп'ютер, щоб розв'язати деяку задачу. ***Програмування*** — це діяльність, яка полягає у створенні програм. Мета програмування — забезпечити, щоб замість людини ту чи іншу роботу виконував комп'ютер.

Усі дії комп'ютера пов'язано з ***обробкою даних*** — числових, символьних, текстових тощо. Дані ***представляють*** (***позначають***) деякий ***зміст***, або ***інформацію***. Поняття змісту та інформації не мають чіткого означення або тлумачення. Будемо розуміти їх як знання, відомості про щось.

Приклади. Слово НЕБО записано чотирма буквами. Зміст, який позначено ними, не є чітким, але для кожної людини це щось велике над головою, блакитне вдень та чорне вночі. Уточнювати далі не будемо.

Запис 1001 представляє число — уявне поняття, яке виражає кількість. Ми сприймаємо ці дані як «тисяча й один» (предмет, рік тощо) або «тисяча й одна» (ніч, дрібниця тощо).

Коли батьки реєструють народжену дитину, вони отримують свідоцтво про народження. У ньому вказано ім'я та прізвище нової людини, дату й місце народження, імена батьків. Ці ***дані про людину*** представляють факт і деякі з обставин її появи на світ. ▣*

---

* Закінчення прикладу позначатимемо знаком ▣.

Отже, представлення об'єктів реального світу за допомогою даних є основою будь-якої взаємодії, зокрема й комп'ютера із «зовнішнім» світом.

Комп'ютер має розв'язувати задачі для людини. У цих задачах фігурують різноманітні об'єкти реального світу — картинки, фізичні явища, особи, технологічні процеси тощо. Щоб запрограмувати обробку даних, пов'язаних із цими об'єктами, треба спочатку ***представити об'єкти у вигляді даних***.

Представлення об'єкта або явища за допомогою даних про нього називають його ***інформаційною моделлю***. Інформаційна модель, як правило, містить не лише дані, а й деякий опис їх обробки, яка моделює реальні процеси, пов'язані з об'єктом. Узагалі, ***модель*** — це спрощене представлення якогось об'єкта або явища, яке відображає його необхідні суттєві властивості.

Існує безліч різновидів інформаційних моделей. Один і той самий об'єкт або явище може мати кілька різних моделей, що виражають «свої» властивості, потрібні з різних точок зору на об'єкт або явище.

Приклади. Свідоцтво про народження або паспорт є дуже простою інформаційною моделлю людини. Складнішою є, наприклад, медична картка, яка містить дані про фізичні та фізіологічні властивості людини, зміну їх у часі тощо. Зовсім інші дані про людину присутні в табелі навчання або атестаті.

Щоб розв'язати квадратне рівняння вигляду $ax^2 + bx + c = 0$, потрібні лише його коефіцієнти — числа a, b, c, тобто трійка цих чисел (дані) представляє рівняння. Формули коренів, по суті, описують дії з розв'язання рівняння (обчислити дискримінант тощо).

Кожне сучасне підприємство має програмні системи, які автоматизують облік кадрів, заробітної платні, фінансової та виробничої діяльності. Кожна з цих систем має «свої» дані, пов'язані з підприємством, які використовуються та обробляються тільки в ній. Ці дані складають частину відповідної (кадрової, фінансової тощо) моделі підприємства. Інша частина моделі описує, як присутні в моделі дані використовуються та обробляються. ▣

2. АЛГОРИТМИ ТА ЇХНІ ВЛАСТИВОСТІ

Алгоритм — це опис послідовності дій, які треба виконати, щоб розв'язати деяку задачу. Позначення дій в алгоритмі називаються ***командами*** або ***інструкціями***.

Як правило, в алгоритмі вказано деякі ***вхідні***, ***результатні*** (***вихідні***) та ***проміжні дані***, що не є ні вхідними, ні вихідними.

Послідовність дій, яку виконують за алгоритмом, називається ***процесом***. Алгоритм, як правило, визначає не один процес, а деяку їх множину.

Приклад. Розглянемо задачу «обчислити дійсні корені квадратного рівняння $ax^2 + bx + c = 0$, заданого коефіцієнтами a, b, c (за умови $a \le 0$)». Алгоритм розв'язання цієї задачі, тобто опис визначення коренів, може мати такий вигляд:

Прочитати коефіцієнти a, b, c.
Обчислити дискримінант d = b*b – 4*a*c.
Якщо d > 0, то обчислити x1 = $\frac{-b-\sqrt{d}}{2a}$**,**
x2 = $\frac{-b+\sqrt{d}}{2a}$ **та написати ці числа;**
інакше, якщо d = 0, то обчислити
x = $\frac{-b}{2a}$ **й написати це число;**
інакше написати «дійсних коренів немає».

У цьому алгоритмі вхідними даними є коефіцієнти a, b, c, проміжними — дискримінант d, вихідними — два корені (можливо, один) або текст «дійсних коренів немає».

За цим алгоритмом, залежно від конкретних вхідних даних, можливе виконання однієї з трьох таких послідовностей дій.

1. Прочитати коефіцієнти, обчислити d, перевірити, що $d > 0$ (і це так), обчислити $x1, x2$ й написати ці числа.

2. Прочитати коефіцієнти, обчислити d, перевірити, що $d > 0$ (і це не так), перевірити, що $d = 0$ (і це так), обчислити x і написати це число.

3. Прочитати коефіцієнти, обчислити d, перевірити, що $d > 0$ (і це не так), перевірити, що $d = 0$ (і це не так), і написати, що дійсних коренів немає.

Дуже часто алгоритм створюється шляхом поступового уточнення понять, пов'язаних із об'єктом, та потрібних дій.

Приклад. Розглянемо алгоритм одягання дитини. Спочатку нам відомо, що є верхній та нижній одяг, і на першому кроці опис одягання має такий вигляд.

Надягти нижній одяг.
Надягти верхній одяг.

Потім уточнюємо, що таке нижній одяг і що треба зробити, щоб його надягти.

```
Надягти трусики.
Надягти майку.
Надягти шкарпетки.
```

Далі уточнюємо, що таке верхній одяг і в чому різниця в одязі хлопчиків та дівчат. Згадуємо про взуття.

```
Надягти сорочку.
Якщо дитина є хлопчиком, то надягти штанці,
інакше надягти спідничку.
Взути сандалії.
```

Алгоритм одягання дитини має такий остаточний вигляд.

```
Надягти трусики.
Надягти майку.
Надягти шкарпетки.
Надягти сорочку.
Якщо дитина є хлопчиком, то надягти штанці,
інакше надягти спідничку.
Взути сандалії.
```

Очевидно, що цей алгоритм задає два можливих процеси, які відрізняються надяганням штанців або спідниці. ▣

Алгоритми повинні мати кілька загальних властивостей: зрозумілість, результативність, однозначність, дискретність, масовість та виконуваність. Розглянемо їх.

Зрозумілість. Для виконання алгоритму завжди потрібен ***виконавець***. Це може бути людина або деяка технічна система, зокрема й комп'ютер. Наприклад, виконувати арифметичні дії, щоб розв'язати квадратне рівняння (і не тільки), може людина. Але вона може перекласти цю роботу на комп'ютер, якщо створить відповідну програму й примусить комп'ютер її виконати. Дії з одягання дитини здатна виконувати людина, а зі збирання приладів — спеціальна автоматична лінія.

Зрозумілість алгоритму полягає в тому, що виконавець може правильно зрозуміти та виконати команди, записані в алгоритмі. Але команди завжди записуються згідно з деякою системою позначень.

Система позначень деякого змісту називається ***мовою***. Мова містить елементарні позначення, правила утворення складніших позначень із простіших та правила, за якими зміст і позначення відповідають одне одному.

Отже, виконавець має ***розуміти мову запису алгоритму***.

Результативність. Виконання будь-якого алгоритму повинно приносити його виконавцеві або іншій особі ***відчутні результати***. Наприклад, «корені рівняння визначено», «дитину одягнуто», «прилад зібрано» тощо.

Однозначність. Алгоритм не повинен містити команд, зміст яких можна сприйняти неоднозначно. Наприклад, якби в алгоритмі одягання дитини була команда «надягти штанці або спідничку», виконавець не знав би, що ж саме йому робити. Окрім того, після виконання кожної команди виконавець має точно знати, що йому робити далі.

Дискретність. Дискретність алгоритму полягає в тому, що він задає послідовність дій, чітко відокремлених одна від одної. Отже, дії, задані командою, мають починатися лише після закінчення дій за попередньою командою. Окрім того, виконання кожної команди повинно займати обмежений проміжок часу.

Масовість. Конкретні об'єкти, до яких застосовуються дії під час виконання алгоритму, визначають конкретні задачі, які часто називаються ***екземплярами задачі***. Наприклад, конкретна трійка чисел 3, 10, 2, відповідна квадратному рівнянню, яке треба розв'язати, або конкретний хлопчик Вася, якого треба одягнути. Масовість алгоритму полягає в тому, що він описує не один, а деяку ***множину процесів***, які відбуваються при розв'язанні всіх можливих екземплярів задачі. Переважна більшість алгоритмів є масовими, хоча існують і алгоритми, що задають тільки один процес.

Виконуваність. Алгоритм має бути таким, щоб на кожному екземплярі задачі його можна було ***виконати до кінця*** (й отримати результат). Кожен процес, заданий алгоритмом, має бути ***скінченним*** і тривати скінченний час. Окрім того, процес ***не повинен обриватися*** без отримання результату.

Приклад. Ділення чисел у стовпчик, тобто утворення десяткового дробу, може бути, залежно від конкретних чисел, як скінченним, так і нескінченним (десятковий дріб для 5/4 є скінченним, для 5/3 — ні). Ще приклад: якщо серед дій, заданих деяким алгоритмом, є ділення, і дільником є 0, то виконати це ділення неможливо, і незрозуміло, що робити далі. ▣

Алгоритм має враховувати можливість нескінченного виконання дій або неможливість виконати наступну дію й містити команди, які забезпечують закінчення дій з деяким результатом ***за будь-яких умов***.

У деяких ситуаціях розглядають ***реальну виконуваність***, враховуючи вимоги до тривалості процесу, які висуваються в конкретних задачах. Існують такі задачі, для яких секунда — це занадто довго, а є й такі, що розв'язуються протягом кількох діб.

3. МОДЕЛЬ КОМП'ЮТЕРА

Дамо спрощений погляд на виконавця програм — комп'ютер. Його загальну структуру наведено на рис. 1 (насправді вона набагато складніша). З основних елементів комп'ютера назвемо лише материнську ***плату***, центральний ***процесор***, оперативну ***пам'ять*** та зовнішні пристрої.

На материнській платі розташовані: центральний процесор, оперативна пам'ять (ОП), центральна магістраль для зв'язку між усіма пристроями комп'ютера, гнізда для підключення інших плат керування зовнішніми пристроями та деякі інші елементи.

Зовнішні пристрої (пристрої уведення-виведення, або ПУВ) — це дисплей (монітор), клавіатура, маніпулятор «миша», дисководи та інші (сканер, модем тощо). Вони керують обробкою даних на зовнішніх носіях, наприклад, магнітних дисках. ПУВ мають власні процесори, які можуть переносити дані з зовнішніх носіїв до оперативної пам'яті або навпаки.

Програма для комп'ютера являє собою послідовність команд, основний зміст яких — ***обробка даних***. Центральний процесор зчитує дані й команди програми з оперативної пам'яті та виконує їх. Команди задають зчитування даних з пам'яті, створення нових даних і запис їх у пам'ять. Є також команди, за якими дані надходять до зовнішніх пристроїв або зчитуються з них.

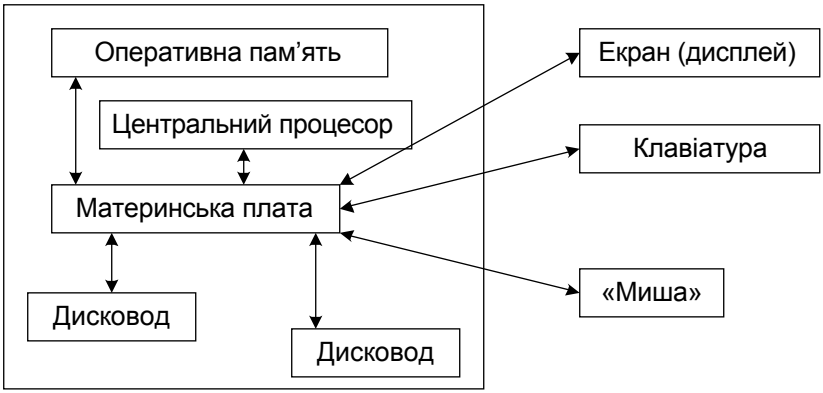


Рис. 1. Загальна схема комп'ютера

Усі дані в комп'ютері представлено у ***двійковій системі***, яка має лише дві цифри — 0 і 1. Значення 0 і 1 відповідають двом стійким станам елемента пам'яті, що називається ***біт*** (***bit***, або ***bi***nary digi***t*** — двійкова цифра).

Вісім послідовних бітів утворюють ***байт***. Байт може мати $2^8 = 256$ станів і представляти кожним з них одне з 256 значень, наприклад, ціле число від 0 до 255. Ці самі стани байта можуть розглядатися як числа від –128 до 127 (їх кількість теж 256) або щось інше.

Оперативна пам'ять являє собою послідовність байтів, в якій кожен байт має свій номер — ***адресу***. Деяке ***значення*** (число, символ тощо) у пам'яті, як правило, займає кілька сусідніх байтів і ***вказується адресою першого з них***.

Представлення чисел в оперативній пам'яті докладніше описано в наступному розділі.

У байтах вимірюють обсяги пам'яті та передавання даних, для чого також використовують такі одиниці, як Кбайт («кейбайт», $2^{10} = 1024$ байт, або не зовсім точно «кілобайт»), Мбайт («мегабайт», $2^{20} = 1048576$ байт) і Гбайт («гігабайт», $2^{30} = 1073741824$ байт). Розмір оперативної пам'яті вимірюється, як правило, десятками й сотнями Мбайт, і середній розмір пам'яті комп'ютерів щороку відчутно зростає.

Машинні команди, як і дані, також записуються послідовностями 0 і 1. Спрощено можна сказати, що команда містить ***код машинної операції*** та ***адреси даних***, до яких застосовується операція.

Система команд, які може виконувати процесор, називається ***машинною мовою***. Для людини машинні мови дуже незручні, вони також вимагають глибоких знань про устрій вузлів комп'ютера та подробиці виконання програми. Цими мовами користуються лише розробники комп'ютерів та деякі інші спеціалісти.

4. МОВИ ПРОГРАМУВАННЯ ВИСОКОГО РІВНЯ

Приблизно у середині 50-х років XX століття було розпочато пошук мов програмування, які мали бути ближчими за формою до мови математичних формул та людських мов, а також вільними від «машинних» подробиць. Водночас, ці мови мали бути такими, щоб записані ними програми можна було ***перекласти*** у відповідні машинні програми ***за допомогою самої машини***. І такі мови програмування невдовзі було розроблено.

Дії комп'ютера в цих мовах було представлено з високим рівнем абстракції, тому ці мови стали називати ***мовами високого рівня***.

Програмувати мовою високого рівня можна, якщо є ***транслятор*** — спеціальна програма, яка здійснює переклад «високорівневої» програми на машинну мову. Цей переклад називається ***трансляцією*** програми.

Протягом кількох десятиліть було створено тисячі мов програмування й трансляторів, і кількість їх постійно зростає.

5. ЕТАПИ СТВОРЕННЯ ПРОГРАМИ

Процес створення програми в найзагальніших рисах має кілька основних етапів:

- аналіз задачі та уточнення її постановки;
- проектування програми;
- розробка програми (кодування);
- остаточна перевірка програми (тестування);
- передача замовникові.

Розкриємо зміст наведених етапів стосовно шкільного навчання.

Аналіз задачі та уточнення її постановки. Спочатку умову задачі дає вчитель або її треба прочитати у задачнику. Дуже часто умову сформульовано недостатньо точно й однозначно, тому її необхідно уточнити, задаючи питання вчителеві. Точна постановка задачі дозволяє зрозуміти, ***які дії треба виконати*** для її розв'язання.

Проектування програми. Тут поступово ***уточнюють дії*** з розв'язання задачі та уточнюють їх опис. З'ясовують і ***уточнюють дані***, потрібні для розв'язанні задачі. Дуже часто в задачі можна виділити кілька ***підзадач*** і описати їх розв'язання окремо. Тоді й алгоритм складається зі зв'язаних та узгоджених між собою частин (***допоміжних алгоритмів***), які описують розв'язання підзадач.

Розробка програми (кодування). Коли дії та дані уточнено до вигляду, в якому їх можна подати в мові програмування, починають розробку програми. Найчастіше програму записують ***мовою високого рівня*** (інколи окремі її частини різними мовами).

Розроблюючи програму, можна припуститися помилок, які виявляються при трансляції або виконанні програми. Помилки необхідно виявляти й виправляти, тобто налагоджувати програму. ***Налагодження програми*** полягає в тому, що її багаторазово запускають зі ***спеціально підібраними*** вхідними даними, які допомагають віднайти помилки.

Остаточна перевірка програми (тестування). В реальних виробничих процесах програму розробляють програмісти, а остаточно перевіряють інші спеціалісти — ***тестувальники***. Тестування починається, коли програмісти упевнені, що програма (або деяка її частина) є правильною. Як правило, спочатку тестувальники все-таки виявляють помилки й цикл «розробка-тестування» повторюється. В умовах школи ситуація аналогічна, тільки в ролі програміста виступає учень, а тестувальника — вчитель.

Передача замовникові. У реальному житті замовникові передається як програма, так і необхідна супроводжувальна документація з її використання. В умовах школи цьому відповідає те, що учень здає програму та звіт із виконаної роботи.

Окрім представлених етапів, реальні програми мають ще етап ***супроводження*** — він полягає в тому, що розробник виправляє помилки, виявлені під час експлуатації програми, модернізує програму й передає її оновлені варіанти користувачам тощо.

6. КРОКИ РОБОТИ З ПРОГРАМОЮ

Типова послідовність роботи з програмою містить такі кроки: набирання тексту; компіляція; компонування; завантаження та виконання або інтерпретація.

Набирання тексту. Текст програми мовою високого рівня (вхідний текст) найчастіше набирають за допомогою спеціальної програми (***текстового редактора***) і, як правило, записують на диск у вигляді вхідного файла (рис. 2). Програма може складатися з кількох файлів — у великих програмах їх можуть бути десятки й сотні.

Компіляція. Компілятор — це програма, при виконанні якої читається вхідний текст і створюється його машинний еквівалент — ***об'єктний код*** (див. рис. 2).

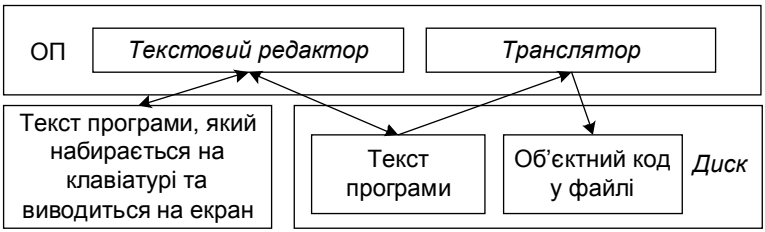


Рис. 2. Створення та компіляція тексту

Як правило, об'єктний код програми містить далеко не всі необхідні команди — програма може складатися з частин; деякі з яких є стандартними.

Компонування. Об'єктний код обробляє ще одна програма — ***компонувальник***. Ця програма «збирає з частин» (компонує) виконуваний код (машинну програму) і записує його або в оперативну пам'ять (***завантажує***), або на диск у вигляді файла, ***готового до виконання*** (рис. 3). Такий файл можна завантажити пізніше.

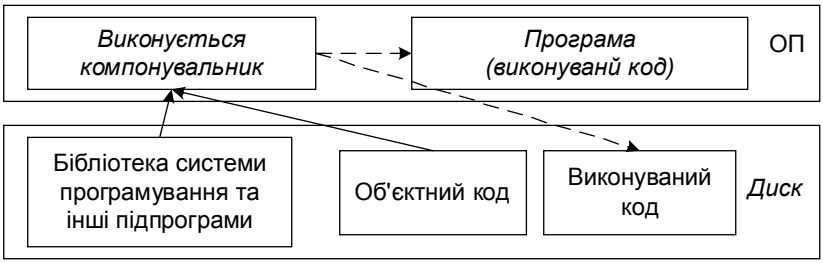


Рис. 3. Створення виконуваної машиною програми

Завантаження та виконання. Запис машинної програми в оперативну пам'ять називається ***завантаженням*** (рис. 4). Його здійснює спеціальна програма — ***завантажувач***, який може входити до складу компонувальника. Якщо завантаження здійснено успішно, починається процес виконання завантаженої програми.

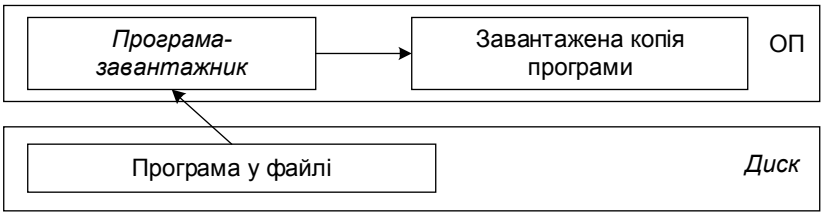


Рис. 4. Завантаження програми

Інтерпретація. Це такий спосіб обробки високорівневої програми, за яким машинна програма не створюється (рис. 5). Вхідна високорівнева програма обробляється спеціальною програмою — ***інтерпретатором***. При цьому дії вхідної програми, які вона задає, одразу виконуються. Як

правило, інтерпретація вхідної програми відбувається повільніше, ніж виконання відповідної машинної програми.

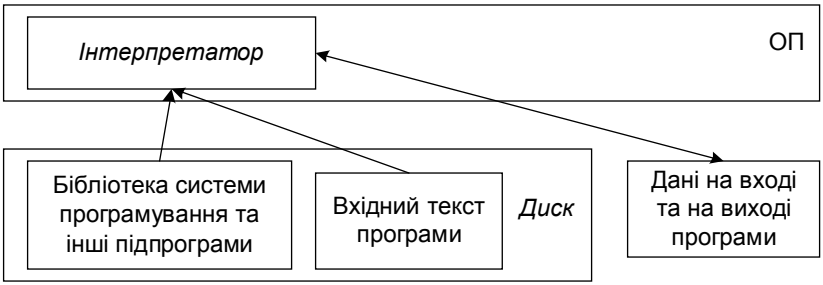


Рис. 5. Інтерпретація високорівневої програми

Інколи компіляцію та інтерпретацію узагальнюють словом ***трансляція***, називаючи трансляторами усі види програм перетворення вхідних текстів до машинного чи проміжного вигляду.

Інтерпретація програми використовується у такому інструменті, як ***налагоджувач***. Він забезпечує інтерпретацію вхідної програми невеликими порціями (кроками) і дає можливість побачити результати виконання після кожного кроку. Це полегшує виявлення помилок у програмі.

Описані засоби (текстовий редактор, компілятор та/або інтерпретатор, компонувальник, завантажувач і налагоджувач), як правило, утворюють ***систему програмування***, або ***інтегроване середовище***. Крім них, до складу цієї системи входить ***бібліотека стандартних підпрограм***, які можна використовувати під час створення програми.

Що краще розробник програми володіє технологіями, інструментом та бібліотеками, то його робота ефективніша.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке інформаційна модель? Назвіть приклади інформаційних моделей.

2. Що таке алгоритм?

3. Чим алгоритм відрізняється від процесу розв'язання задачі?

4. Назвіть властивості алгоритму.

5. Які функціональні блоки має комп'ютер? Яке їх призначення?

6. Що входить до складу зовнішньої пам'яті?

7. Що таке біт і байт? Скільки станів вони можуть мати?

8. Чому числа в комп'ютері подані у двійковій системі?

9. З яких елементів складається машинна програма?

10. Чим мови програмування високого рівня відрізняються від машинних мов?

11. Що таке трансляція?

12. Чим відрізняється компіляція від інтерпретації?

13. Назвіть основні етапи створення програми.

14. У чому полягає наладка програми?

15. Які засоби входять до складу інтегрованого середовища?

ðîçä³ë 2

ÏÐÅÄÑÒÀÂËÅÍÍß ×ÈÑÅË

1. ПОЗИЦІЙНІ СИСТЕМИ ЧИСЛЕННЯ

Людина звикла до десяткової системи запису чисел (***системи числення***). Ця система поступово удосконалювалася протягом тисячоліть, починаючи з давніх Вавилону та Індії. У Середньовіччі вона стала відома арабам і завдяки їм прийшла в Європу.

У десятковій системі є ***десять знаків*** — цифр, якими записують числа від 0 до 9. Великі числа записують тими самими знаками, але не одним, а двома й більше, тобто число записують як ***послідовність знаків***. У цій послідовності знаки мають різні ***позиції*** і тому цифра праворуч позначає кількість одиниць, наступна — кількість десятків, і так далі. Отже, одна й та сама цифра залежно від позиції має «різну вагу». Наприклад, у записі 32 цифра 2 задає дві одиниці, а у записі 23 — два десятки. Тому цю систему запису чисел називають ***позиційною.***

Історія людства залишила у спадок не лише десяткову систему запису чисел. У деяких країнах люди й дотепер підраховують предмети ***дюжинами*** (12 предметів) та ***гросами*** (12 дюжин). Для запису чисел у такій системі потрібні ***12 різних знаків***. Деякі народи використовували ***60 різних знаків***, деякі — ***п'ять.***

Усі вказані системи мають різні ***кількості*** знаків (10, 12, 60‡5), які називаються ***основами.***

Окрім позиційних систем, відомі й ***непозиційні***. Деяке уживання й дотепер має ***римська система*** числення, що виникла в Давньому Римі. У цій системі запис наступного числа утворюється не новою цифрою, а ***додаванням*** цифри: I, II, III, тому її називають ***адитивною***. Звичайно,

правила утворення записів чисел цим не вичерпуються; вони набагато складніші, ніж у позиційних системах, і розглядати їх докладніше не будемо. Особливо незручно в римській системі виконувати арифметичні операції, і недарма вона не стала «робочою» для людства.

Повернемося до позиційної десяткової системи. Цифру праворуч у запису числа називають ***молодшою*** («її записано в молодшому розряді»), ліворуч — ***старшою***. Розряд одиниць називають нульовим, розряд десятків — першим, сотень — другим тощо. За такої нумерації вага розряду відповідає степеню числа 10: одиниця — це 10^0, десяток — це 10^1, сотня — це 10^2 тощо.

Отже, розташування цифри в тому чи іншому розряді є прямою вказівкою, на який ***степінь числа 10 треба помножити цифру***. Звідси число можна записати як суму добутків цифр числа на відповідні степені десятки. Наприклад,

$$5834 = 5 \cdot 10^3 + 8 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0.$$

Якщо запис числа має дробову частину, то додаються ***цифри, ділені на число 10 у відповідному степені***, наприклад,

$$0{,}234 = 2 \cdot \frac{1}{10^1} + 3 \cdot \frac{1}{10^2} + 4 \cdot \frac{1}{10^3}$$

Розглянемо яку-небудь позиційну систему числення з основою, відмінною від 10, наприклад, 7.

Аналогічно до десяткової, ця система повинна мати такі властивості:

- для запису чисел є сім цифр— 0, 1, 2, 3, 4, 5, 6;
- значення цифри залежить від її розташування (позиції) в записі;
- вага кожного розряду числа є відповідним степенем сімки.

Отже, перші числа записуються як 0, 1, Ö, 6, а далі йдуть записи 10, 11, Ö, 16, 20, 21, Ö, 66, 100, … . Сімкове 10 — це звичне десяткове 7 (але ж немає такого знака в сімковій системі!), сімкове 11 — це звичне 8, 20 — це звичне десяткове 14, тобто двічі по 7, тощо. А сімкові 100 та 200 — це десяткові 49 та 98, тобто один та два рази по 7 у квадраті.

А тепер подамо сумами степенів сімки такі числа (про сімковий запис свідчить маленька цифра 7 біля числа):

$$342_7 = 3 \cdot 7^2 + 4 \cdot 7^1 + 2 \cdot 7^0$$

$$6301_7 = 6 \cdot 7^3 + 3 \cdot 7^2 + 0 \cdot 7^1 + 1 \cdot 7^0$$

$$0{,}12_7 = \frac{1}{7^1} + \frac{2}{7^2}$$

$$32{,}56_7 = 3 \cdot 7^1 + 2 \cdot 7^0 + \frac{5}{7^1} + \frac{6}{7^2}$$

Цей спосіб запису використовується у позиційних системах числення з будь-якою основою (більше 1).

Питання в тому, які знаки є цифрами. Якщо основа не більше десяти, використовують звичний набір десяткових цифр (з нього беруть стільки знаків, скільки треба). Проте для системи з основою більше десяти потрібні додаткові знаки, щоб позначати десяткові числа 10, 11, … . Де їх узяти? У XX столітті для цього стали використовувати послідовні великі літери латинського алфавіту — *A*, *B*, *C* тощо.

Приклади. У двійковій системі числення лише дві цифри — 0 та 1. Двійковий запис 1101 позначає число $1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13$, запис 0,101 — число $1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 0{,}625$.

У шістнадцятковій системі числа від 10 до 15 позначають так: 10†ó†*A*, 11†ó†*B*, 12†ó†*C*, 13†ó†*D*, 14†ó†*E*, 15†ó†*F*. Тоді

$2A_{16} = 2 \cdot 16^1 + 10 \cdot 16^0 = 42$,

$FF_{16} = 15 \cdot 16^1 + 15 \cdot 16^0 = 255$,

$0, C_{16} = 12 \cdot 16^{-1} = 3/4 = 0{,}75$. ▣

- Запис $x_n \ldots x_1 x_0 . x_{-1} \ldots x_{-k}$ з *P*-ковими цифрами задає число

$$x_n \cdot P^n + \ldots + x_1 \cdot P^1 + x_0 \cdot P^0 + x_{-1} \cdot P^{-1} + \ldots + x_{-k} \cdot P^{-k}.$$

Цифри *x* у цій сумі позначають числа від 0 до *P*–1.

2. ПЕРЕТВОРЕННЯ ЧИСЕЛ З НЕДЕСЯТКОВОЇ СИСТЕМИ В ДЕСЯТКОВУ

У комп'ютерах числа подані за допомогою елементів, які мають два стійкі стани, відповідні значенням 0 і 1, тобто двійковим цифрам. Низка таких елементів своїми станами представляє послідовність двійкових цифр, тобто число у ***двійковому*** записі.

Отже, у комп'ютері числа подані та обробляються в двійковій системі. Проте людям зручніше вводити числа в комп'ютер та отримувати їх від

нього в десятковій системі. Так виникає задача ***переведення запису числа*** з однієї системи в іншу, зокрема, з двійкової в десяткову та навпаки.

Запис $x_n \ldots x_1 x_0 . x_{-1} \ldots x_{-k}$ з недесятковими (P-ковими) цифрами задає число, яке є сумою чисел, позначених цифрами й помножених на відповідні степені основи P. Отже, щоб знайти десяткове представлення числа, треба:

```
представити в десятковій системі число P
та цифри запису;
обчислити суму добутків значень цифр
та відповідних степенів числа P.
```

Приклади

$10011_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 19$

$10{,}011_2 = 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 2{,}375$

$1\text{BC}_{16} = 1 \cdot 16^2 + 11 \cdot 16^1 + 12 \cdot 16^0 = 444$

$1\text{B,C}_{16} = 1 \cdot 16^1 + 11 \cdot 16^0 + 12 \cdot 16^{-1} = 27{,}75$;

$12{,}2_3 = 1 \cdot 3^1 + 2 \cdot 3^0 + 2 \cdot 3^{-1} = 5{,}666\ldots$, тобто представляється нескінченним періодичним десятковим дробом. ▣

Нагадаємо: якщо основа P має прості дільники, відмінні від 2 та 5, то дробове число зі скінченним P-ковим записом може мати ***нескінченне*** періодичне десяткове подання. Якщо ж простими дільниками P є лише 2 і 5, то й десятковий дріб скінченний.

3. ПЕРЕТВОРЕННЯ З ДЕСЯТКОВОЇ СИСТЕМИ В НЕДЕСЯТКОВУ

Переведення цілих чисел. Розглянемо, як за натуральним числом T у десятковому записі отримати цифри P-кового представлення. Нам не відомі ні самі цифри, ні їх кількість. Але відомо, що P-ковий запис задає число як суму добутків значень цифр на відповідні степені числа P, тобто при деякому невідомому n і невідомих цифрах x_i справджується рівність

$$T = x_n \cdot P^n + \ldots + x_1 \cdot P^1 + x_0 \cdot P^0.$$

Помітимо: всі доданки, окрім останнього, мають множник P. Тоді значенням молодшої цифри x_0 є остача від ділення T на основу P, а сума $T_1 = x_n \cdot P^{n-1} + \ldots + x_1 \cdot P^0$ дорівнює цілій частці від ділення T на P. Поді-

ливши цю суму на P з остачею, знайдемо остачу x_1 і наступну частку, і так далі, поки на якомусь кроці частка від ділення не стане менше P. Вона й буде старшою цифрою x_n.

Приклади. Перекладемо 202 у вісімкову систему.

202 : 8 = 25 (остача 2 — молодша цифра),

25 : 8 = 3 (остача 1 — наступна цифра).

3 : 8 = 0 (остача 3 — остання старша цифра).

Отже, отримано цифри 2, 1, 3 (від молодшої до старшої), тобто вісімковий запис 312_8. ▣

Перекладемо 202 у 16-кову систему.

202 : 16 = 12 (остача 10 — значення молодшої цифри *A*),

12 : 16 = 0 (остача 12 — значення старшої цифри *C*),

Отже, отримано 16-ковий запис CA_{16}. ▣

Позначимо остачу від ділення T на P націло як $T \bmod P$, частку — як $T \operatorname{div} P$. Опишемо утворення P-кового запису числа T таким алгоритмом.

```
1. Спочатку частка T дорівнює N, а запис порожній.
2. Поки T > 0, обчислювати остачу R як T mod P
   та нову частку T як T div P (шляхом ділення у
   стовпчик), представляти R у P-ковій системі
   числення як цифру C(R) та дописувати до запису
   ліворуч.
```

Переведення дробових чисел. Розглянемо, як за додатним дійсним числом $V < 1$ отримати цифри його P-кового запису (принаймні кілька перших, оскільки в дійсності запис може бути й нескінченним). Припустимо, що $V = x_{-1} \cdot P^{-1} + \ldots + x_{-k} \cdot P^{-k} + \ldots$ з невідомими значеннями цифр. Якщо обидві частини рівності помножити на P, отримаємо рівність

$$V \times P = x_{-1} + x_{-2} \cdot P^{-1} + \ldots + x_{-k} \cdot P^{-k+1} + \ldots, \text{ або}$$

$$V \times P = x_{-1} + P^{-1}(x_{-2} + \ldots + x_{-k} \cdot P^{-k+2} + \ldots).$$

Звідси

$$x_{-1} = [V \times P], \text{ а } P^{-1}(x_{-2} + \ldots + x_{-k} \cdot P^{-k+2} + \ldots) = \{V \times P\},$$

де $[V \times P]$ та $\{V \times P\}$ позначають цілу та дробову частини $V \times P$. Далі так само помножимо $\{V \times P\}$ на P, знову отримаємо цілу та дробову частину (ціла буде значенням x_{-2}), і так далі.

Приклади. Одержимо двійковий запис десяткового дробу 0,75.

0,75×2†=†1,5, [1,5]†=†1 (перша цифра), {1,5} = 0,5;

0,5×2†=†1, [1]†=†1 (наступна цифра), {1} = 0.

Усі подальші цифри будуть 0, тому 0,11 є скінченним двійковим представленням для десяткового 0,75.

Одержимо двійковий запис десяткового дробу 0,1.

0,1×2†=†0,2, [0,2]†=†0 (перша цифра), {0,2} = 0,2;

0,2×2†=†0,4, [0,4]†=†0 (наступна цифра), {0,4} = 0,4;

0,4×2†=†0,8, [0,8]†=†0 (наступна цифра), {0,8} = 0,8;

0,8×2†=†1,6, [1,6]†=†1 (наступна цифра), {1,6} = 0,6;

0,6×2†=†1,2, [1,2]†=†1 (наступна цифра), {1,2} = 0,2.

Далі цифри 0, 0, 1, 1 будуть нескінченно повторюватися, тобто точний двійковий запис десяткового 0,1 є періодичним 0,0(0011), а будь-який початок цього запису позначає наближення до десяткового 0,1, наприклад,

$$0{,}00011 = \frac{1}{16} + \frac{1}{32} = 0{,}09375 \text{(помилка — приблизно шість тисячних).}$$

Одержимо 16-ковий запис десяткового дробу 0,8.

0,8×16 = 12,8, [12,8] = 12 (перша цифра **C**), {12,8}†=†0,8.

Далі цифра *C* буде нескінченно повторюватися, тому *0,(C)* є точним 16-ковим записом десяткового 0,8. ▣

Якщо *V* є скінченним десятковим дробом і основа *P* має обидва прості множники 2 та 5, то число *V* можна представити скінченним *P*-ковим записом. В іншому випадку *P*-ковий запис може бути нескінченним, і тоді за скінченну кількість кроків буде отримано ***наближене представлення*** числа *V*.

Отже, за дійсним числом *V*, *V* < *1*, можна одержати *R* перших цифр його *P*-кового представлення, виконуючи такий алгоритм.

```
1. Спочатку представленням є «0.».
2. Поки одержано менше за R дробових цифр,
   обчислити V×P, обчислити d як [V×P] (ціле
   число від 0 до P–1) та V як {V×P}.
   Представити значення d як P-кову цифру та
   дописати її до представлення праворуч.
```

Цілі та дробові числа переводяться в недесяткову систему за ***різними алгоритмами***. Якщо треба перевести число, що має цілу та дробову частини, треба виділити ці частини й перевести їх ***окремо***. Взагалі, перевести запис числа з недесяткової системи числення в іншу недесяткову не-

просто, оскільки при цьому необхідно виконувати багато арифметичних дій у незвичній недесятковій системі. Проте це зробити просто, використовуючи десяткову систему як проміжну.

4. ЗВ'ЯЗОК ДВІЙКОВИХ, ВІСІМКОВИХ ТА ШІСТНАДЦЯТКОВИХ ЗАПИСІВ

Двійкова система не дуже зручна для людини, оскільки числа записуються в ній досить довгими послідовностями нулів і одиниць. Наприклад, десяткове число **1024** в двійковій системі числення виглядає як **10000000000**. Для скорочення записів у програмуванні традиційно використовують вісімкову та шістнадцяткову системи числення.

Вісімковим цифрам від 0 до 7 взаємно однозначно відповідають усі можливі ***трирозрядні двійкові коди*** (з незначущими нулями).

```
0 000    1 001    2 010    3 011
4 100    5 101    6 110    7 111
```

Аналогічно, ***шістнадцятковим*** цифрам від **0** до **F** відповідають усі можливі ***чотирирозрядні двійкові коди*** (також із незначущими нулями).

```
0 0000   4 0100   8 1000   C 1100
1 0001   5 0101   9 1001   D 1101
2 0010   6 0110   A 1010   E 1110
3 0011   7 0111   B 1011   F 1111
```

Наведений зв'язок між цифрами та групами двійкових цифр забезпечує дуже простий спосіб отримання вісімкового та шістнадцяткового запису за двійковим і навпаки. Для переведення з вісімкової в шістнадцяткову систему й навпаки можна використати двійковий запис як проміжний.

У двійковому записі можна, починаючи з молодшої цифри, замінити трійки цифр відповідними вісімковими цифрами (остання трійка може бути неповною — до неї допишемо незначущі нулі). Навпаки, вісімковий запис перетворюється на двійковий заміною цифр відповідними їм трійками двійкових. Трійка, що відповідає старшій цифрі, може мати незначущі нулі — вони не записуються.

Приклад. Двійковий запис 1111010 розбивається на групи 1, 111, 010. Група 1 доповнюється до трійки 001. Трійці 001 відповідає вісімкова циф-

ра 1, трійці 111†ó†7, і, нарешті, 010†ó†2. Результатом перетворення є вісімкове 172.

Навпаки, у вісімковому 172 цифрі 1 відповідає група двійкових розрядів 001, 7 — група 111, 2 — група 010. Результат перетворення — двійкове 1111010 (незначущі старші 00 не записуємо).▣

Перетворення двійкового запису на шістнадцятковий аналогічне, лише двійковий запис розбивають на групи з чотирьох розрядів, які замінюють відповідними шістнадцятковими цифрами. Навпаки, шістнадцятковий запис перетворюють на двійковий заміною цифр відповідними четвірками двійкових цифр.

Приклад. Двійкове число 1111010 розбивається на групи 111 і 1010. Старша група 111 доповнюється до 0111; їй відповідає шістнадцяткова цифра 7, групі 1010 — цифра *A*. Результат перетворення — шістнадцятковий запис 7A.

У шістнадцятковому записі 7A цифрі 7 відповідає тетрада 0111, цифрі A†ó 1010. Результат перетворення — двійкове число 1111010.▣

Аналогічний зв'язок між цими трьома системами існує й у записі дробових чисел. Групи по три двійкові цифри, починаючи від старших (***перших після коми***), відповідають вісімковим цифрам, а групи по чотири — шістнадцятковим. Остання група може бути неповною — до неї дописують незначущі нулі, але ***праворуч***.

Приклад. Двійковий запис 0,0101111101 розіб'ємо на групи по три цифри: 010, 111, 110 та 1. Останню групу доповнимо до 100 та отримаємо вісімковий запис 0,2764. Цей самий двійковий запис розіб'ємо на групи 0101, 1111, 01, останню групу доповнимо до 0100 та утворимо шістнадцятковий запис 0,5F4. ▣

5. ДОДАВАННЯ ТА МНОЖЕННЯ У ПОЗИЦІЙНИХ СИСТЕМАХ ЧИСЛЕННЯ

ДОДАВАННЯ

Додавання одноцифрових чисел учні вивчають за допомогою таблиць, у яких подано результати додавання чисел від 1 до 9, наприклад, 7 + 8 = 15, 9 + 1 = 10, 9+9 = 18. Аналогічні таблиці додавання неважко скласти для будь-якої системи числення.

Приклад. Для сімкової системи з цифрами 0, 1, 2, 3, 4, 5 та 6 таблиця 1 додавання має такий вигляд.

Таблиця 1

	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	10
2	2	3	4	5	6	10	11
3	3	4	5	6	10	11	12
4	4	5	6	10	11	12	13
5	5	6	10	11	12	13	14
6	6	10	11	12	13	14	15

Таблиця демонструє ***переставний закон додавання*** — заповнивши рядок таблиці, можна відразу отримати відповідний стовпчик.

Якщо сума чисел не менш ніж 7, утворюється двоцифрова сума, тобто з'являється перенос 1 до старшого розряду. Так само в десятковій системі перенос з'являється, якщо сума не менше ніж 10.

У системах з основою більше 10 потрібні цифри A, B, C тощо. Наприклад, два останні рядки таблиці додавання однорозрядних чисел з основою 16 мають такий вигляд:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

При додаванні одноцифрових чисел A та B у будь-якій P-ковій системі значенням цифри є ***остача від ділення*** ($A+B$) на P, переносом до старшого розряду — ***частка від ділення*** ($A+B$) на P. Якщо $A+B < P$, перенос дорівнює 0 («переносу немає»), а якщо $A+B \geq P$, він дорівнює 1.

Наведене правило дає основу для алгоритму додавання чисел у стовпчик у довільній P-ковій системі.

Починаючи від молодшого розряду, цифри та перенос від попереднього розряду додаються, у розряді залишається остача від ділення суми на *P*, а частка є переносом до наступного розряду. Перенос у молодший розряд дорівнює 0.

Приклади. Додамо числа в сімковому записі; у дужках нагорі запишемо переноси.

(110)	(100)	(1110)
+ 45	+ 561	+ 2656
26	64	5614
104	655	11603

Аналогічно додамо числа в двійковому запису, де 0+0 = 0, 1+0 = 0+1 = 1, 1+1 = 10.

(100)	(1100)	(1110)
+ 10	+ 11	+ 111
11	110	1
101	1001	1000

Так само додамо числа в шістнадцятковому запису.

(100)	(010)	(1110)
+ 87	+ 1*F*	+ *FFF*
95	12*E*	1
11*C*	14*D*	1000

МНОЖЕННЯ

Множення в недесяткових системах числення виконується також за допомогою відповідних таблиць. Розглянемо, наприклад, таблицю множення однорозрядних чисел з основами 2, 3 та 4.

	0	1
0	0	0
1	0	1

	0	1	2
0	0	0	0
1	0	1	2
2	0	2	11

	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	10	12
3	0	3	12	21

Таблиці демонструють ***переставний закон множення*** — заповнивши рядок таблиці, можна відразу отримати відповідну колонку.

Наведемо також вісім останніх рядків таблиці 2 множення з основою 16.

Таблиця 2

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	32	3D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

При множенні одноцифрових чисел A та B у будь-якій P-ковій системі значенням цифри є ***остача від ділення*** $A \times B$ на P, переносом до старшого розряду — ***частка від ділення*** $A \times B$ на P. За $A \times B < P$ перенос дорівнює 0.

Наведене правило дає основу для алгоритму множення кількарозрядного числа на одноразрядне число X у стовпчик.

Починаючи з молодшого розряду, обчислюється добуток *Y* значення цифри числа та *X*. До *Y* додається перенос від попереднього розряду й отримується сума *S*. Остача від ділення *S* на *P* у *P*-ковому представленні записується в результат, а частка є переносом до наступного розряду.

Приклад. Помножимо числа відповідно в трійковій, четвірковій та шістнадцятковій системах; у дужках вгорі запишемо переноси.

(110)	(1220)	(8*D*10)
$\times$ 212	$\times$ 123	$+$ 8*E*2
2	3	*F*
1201	1101	853*E*

Виконуючи множення на число, що має більше однієї цифри, множимо на окремі його цифри, записуємо результати з відповідним зсувом вліво та додаємо їх у стовпчик.

Приклад. Помножимо десяткові 12 і 21 у десятковій, двійковій, четвірковій, вісімковій та шістнадцятковій системах відповідно.

×12	× 1100	× 30	×14	× *C*
21	10101	111	25	15
12	1100	30	74	3*C*
24	1100	30	30	*C*
252	1100	30	374	*FC*
	11111100	3330		

- У всіх наведених прикладах ми ніколи ***не змішували записи*** чисел у системах з різними основами. Якщо числа записано в різних системах числення, то виконувати дії з ними чи порівнювати їх ***не можна.***

6. ВНУТРІШНЄ ПРЕДСТАВЛЕННЯ ЧИСЛОВИХ ДАНИХ

Як уже зазначалося, вісім послідовних бітів утворюють байт, який може мати 256 різних станів (комбінацій з 0 і 1). Їм можна поставити у взаємно однозначну відповідність цілі числа від 0 до 255, цілі числа від –128 до 127, пари 16-кових цифр або елементи якоїсь іншої 256-елементної множини.

ПРЕДСТАВЛЕННЯ ЦІЛИХ ЧИСЕЛ

Цілі числа подаються в комп'ютері переважно у двох формах — беззнаковій та знаковій. Ці форми називаються ***кодами***. Далі числа ототожнюються з їх представленням, хоча з погляду математики це є хибним.

Беззнакові коди займають декілька байтів (найчастіше, 1, 2, 4 або 8). Байти нумеруються, починаючи з 0. Біти всередині байтів нумеруються від 0 до 7. *N* байтів містять *8N* бітів, тому використовують також наскрізну нумерацію бітів — від 0 до *8N – 1* (від молодших до старших). Усі можливі послідовності бітів представляють *числа* від 0 до 2^{8N-1}. Відповідність між кодами та числами при різних значеннях *N* представлено у такій таблиці 3.

Знакові коди також займають 1, 2, 4 або 8 байт. Старший біт представляє знак числа: 0†ó ´+ª, 1†ó ´ñª. ***Додатні*** числа представляються так само, як і беззнакові, лише за рахунок знакового біта їх діапазон — від 0 до $2^{8N-1}-1$. Таке представлення називається ***прямим кодом***. Наприклад, число $2^{8N-1}-1$ має прямий код 011…1.

Таблиця 3

Коди	*Числа*	*N = 1*	*N = 2*	*N = 4*
11…11	$2^{8N}-1$	255	65535	4394967295
11…10	$2^{8N}-2$	254	65534	4394967294
11…01	$2^{8N}-3$	253	65533	4394967293
…	…	…	…	…
10…00	2^{8N-1}	128	32768	2147483548
01…11	$2^{8N-1}-1$	127	32767	2147483647
…	…	…	…	…
00…10	2	2	2	2
00…01	1	1	1	1
00…00	0	0	0	0

Від'ємні числа представлено так званим ***додатковим кодом***. Код від'ємного числа A позначається $D(A)$ й утворюється так.

1. За прямим кодом числа |A| заміною всіх 0 на 1 і всіх 1 на 0 будуємо обернений код *R*(A).

2. За *R*(A) як беззнаковим цілим числом обчислюємо код *D*(A) = *R*(A)+1.

Приклад. Побудуємо двобайтовий додатковий код числа –144.

Прямий код |A|	**0000 0000 1001 0000**
Обернений код R(A)	**1111 1111 0110 1111**
Додавання 1	**1**
Додатковий код D(A)	**1111 1111 0111 0000**

«Відновити» за додатковим кодом від'ємне число можна у зворотному порядку.

1. *D*(A) розглядаємо як беззнакове ціле та обчислюємо R(A) = D(A)–1.

2. Код, обернений до R(A), є прямим кодом числа |A|.

Проте можна обійтися без віднімання. Очевидно, що $D(A) = R(|A| - 1)$, тобто, наприклад, додатковий код числа –144 водночас є оберненим кодом числа –143. Тоді код $R(D(A))$ є прямим кодом числа $|A| - 1$. Тоді прямий код $|A|$ можна отримати так.:

1. Будуємо код *R*(*D*(A)), обернений до *D*(A).

2. До *R*(*D*(A)) як беззнакового цілого додаємо 1.

Відповідність між кодами та числами при різних значеннях *N* представлено в такій таблиці 4:

Таблиця 4

Коди	***Числа***	***N = 1***	***N = 2***	***N = 4***
01…11	$2^{8N\tilde{n}1}\tilde{n}1$	127	32767	2147483647
…	Ö	Ö	Ö	Ö
00…10	2	2	2	2
00…01	1	1	1	1
00…00	0	0	0	0
11…11	ñ1	ñ1	ñ1	ñ1
11…10	ñ2	ñ2	ñ2	ñ2
11…01	ñ3	ñ3	ñ3	ñ3
…	Ö	Ö	Ö	Ö
10…00	$\tilde{n}2^{8N\tilde{n}1}$	ñ128	ñ32768	ñ2147483548

Як бачимо, від'ємних чисел на одне більше, ніж додатних.

ПРИНЦИПИ ПРЕДСТАВЛЕННЯ ДІЙСНИХ ЧИСЕЛ

Деякі ***особливості представлення*** дійсних чисел тут ***опущено***, щоб не загромаджувати викладення.

Для дійсних чисел найчастіше використовують 4, 6, 8 або 10 байт, поділених на ***поля*** (послідовності бітів) <*знак*><*порядок*><*мантиса*>. Ці поля означають таке.

Поле <*знак*> має довжину 1 біт, його значення 0 представляє знак «+», 1 — знак «–». Поле <*порядок*> у деякий спеціальний спосіб задає ***показник степеня*** числа 2 (він може бути як додатним, так і від'ємним). Поле <*мантиса*> означає ***дробову частину*** — невід'ємне число строго менше 1.

Поля представляють число в такий спосіб. Мантиса додається до 1, сума множиться на степінь числа 2, заданий порядком, і враховується знак. Отже, число є значенням виразу вигляду:

$$\pm(1+\textit{дробова частина})\times 2^{\textit{показник}}.$$

Сума в цьому виразі не менше 1 і строго менше 2, тому кажуть, що це представлення числа є ***нормалізованим***.

Дійсні числа в описаному представленні утворюють ***обмежену підмножину*** множини раціональних чисел; зокрема, серед них є число, ***найменше за модулем і відмінне від 0***, а також число, ***найбільше за модулем***.

Між двома послідовними числами в описаному представленні є ***нену-***

льова різниця. Вона дуже мала, якщо числа близькі до 0, і досягає десятків тисяч, якщо числа близькі до найбільшого за модулем.

Що більше байтів займає представлення, то більше чисел воно дозволяє закодувати й то менше між ними різниці.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. З чим пов'язують виникнення десяткової системи числення?

2. Які ви знаєте системи числення?

3. Що таке позиційна система числення?

4. Наведіть приклад адитивної системи числення.

5. Чи можна використовувати позиційні системи числення з цифрами, що мають різні основи?

6. Як кодуються цифри в системах числення з основою більше десяти?

7. Подати алгоритм переведення десяткового запису числа в систему числення, відмінну від десяткової.

8. Розповісти алгоритм переведення запису числа з недесяткової системи числення в десяткову.

9. Складіть таблицю додавання для системи числення з основою 16.

10. Складіть таблицю множення для системи числення з основою 11.

11. Які системи числення використовують у роботі з комп'ютером?

12. Розповісти алгоритм переведення двійкового запису числа у вісімковий (шістнадцятковий).

13. Розповісти алгоритм переведення запису числа з основою 8 (16) у двійковий.

14. Чому квадрат числа P – 1 у P-ковому запису обов'язково має старшу цифру P – 2, а молодшу 1.

ЗАДАЧІ

1. Запишіть число як суму степенів основи системи числення з коефіцієнтами:

a) 5768; b)†842,413; c)†12,223; d) A09,7F16.

2. Переведіть десяткове число в римську систему запису:

a) 55; b)†995; c)†1547.

3. Переведіть римський запис чисел в арабський (позиційний десятковий):

a) LX; b)†CXI; c)†MDCCCXII; d)†MCMLXI.

4. Запишіть число як суму степенів основи системи числення з коефіцієнтами та визначте десятковий запис числа:

a) 2,336; b)†291,4*D*615; c)†245,5678; d)†*A*09,C3513;

e)†*B*,1*F*416; f)†2743,1611.

5. Переведіть десятковий запис у двійкову, вісімкову та шістнадцяткову системи:

a) 94,341; b)†13,55; c)†2948,77; d)†382,33.

6. Перетворити десяткові записи чисел на двійкові, вісімкові та шістнадцяткові:

a) 100; b)†255; c)†1024; d)†32767; e)†256;
f)†640; g)†16382; h)†65537.

7. Перетворити шістнадцяткові записи чисел на вісімкові та десяткові:

a) *F*1; b)†*FF*; c)†4*AB*; d)†*FFFE*;
e)†1*FF*; f)†8A1; g)†7*FFF*.

8. Перетворити вісімкові записи чисел на шістнадцяткові та десяткові:

a) 377; b)†1200; c)†1232; d)†400;
e)†1777; f)†3777; g)†7777.

9. Додайте числа в заданій системі числення:

a) $36_6 + 42_6$; b)† $112_3 + 21_3$; c)† $1001_2 + 110_2$; d)† $35_8 + 75_8$;

e)† $79_{12} + A6_{12}$; f)† $45_{11} + A9_{11}$.

10. Перемножте числа в заданій системі числення:

a) $36_6 \cdot 42_6$; b)† $112_3 \cdot 21_3$; c)† $1001_2 \cdot 110_2$; d)† $35_8 \cdot 75_8$;

e)† $79_{12} \cdot A6_{12}$; f)† $45_{11} \cdot A9_{11}$.

11. Подати 36-кові записи чисел ZY, 100 у десятковому записі (36-кові цифри *A, B, …, Y, Z* позначають десяткові числа *10*, 11, …, 34, 35, відповідно).

12. За основою *P* та *P*-ковими записами дробів *Q* вказати їх десятковий запис:

a) *P* = 2; *Q* = 0,0001; 0,1111; b)†*P* = 3; *Q* = 0,22; 0,(11);
c)†*P* = 16; *Q* = 0,1; 0,F; 0,8; 0,(7); d)†*P* =2; *Q* = 0,001; 0,1101; 0,000001;
e)†*P* = 3; *Q* = 0,(20); 0,(02); 0,11; f)†*P* = 16; *Q* = 0,1F; 0,FE; 0,81; 0,3(F).

13. Записати *P*-ковий заприс десяткового дробу *Q*, де:

a) *Q* = 0,5; *P* = 2, 3, 5, 8, 16, 20; b)†*Q* = 0,1; *P* = 3, 5, 8, 16, 20;
c)†*Q* = 0,75; *P* = 2, 3, 5, 8, 16, 20; d)†*Q* = 0,2; *P* = 3, 5, 8, 16, 20.

14. Вказати двобайтовий додатковий код чисел:

a) –1, –8, –9, –32767, –32768; b)†ñ255, ñ256, ñ640, ñ641.

15. Нехай при додаванні та відніманні чисел у знаковому записі перенесення із старшого цифрового розряду перетворюється на зміст знакового розряду, а перенос із знакового розряду втрачається. Указати значення виразу (*maxi* та *mini* позначають максимальне та мінімальне цілі числа): a) *maxi* + 1; b)†*mini* ñ 1.

ðîçä³ë 3

ÏÐÎÃÐÀÌÓÂÀÍÍß Ë²Í²ÉÍÈÕ ÀËÃÎÐÈÒÌ²Â ÌÎÂÎÞ TURBO PASCAL 7.0

1. АЛФАВІТ І СЛОВНИК МОВИ

Вивчення будь-якої мови починається з вивчення ***алфавіту*** — скінченної множини символів, дозволених для використання. Алфавіт мови Turbo Pascal містить:

- *літери* (*букви*) — великі й малі латинські *A*, *B*, … , *Z*, *a*, *b*, …, *z*, а також символ підкреслення _;
- *десяткові цифри* 0, 1, 2, …, 9;
- *спеціальні символи* `+ - * / = > < . , ; : ' ( ) [ ] { } ^ @ $ #`.

З символів алфавіту утворюються «слова» мови — ***лексеми*** (або ***лексичні одиниці***). Це послідовності символів, які розглядаються як неподільні й самі по собі мають деякий зміст.

У мові Turbo Pascal є ***чотири види лексем***:

- константи;
- імена;
- знаки операцій;
- роздільники.

Окрім них, у програмах записують ще коментарі та директиви компілятора.

Константа — це позначення значення (числового або іншого типу). Числові константи мають звичний для нас вигляд, наприклад, 273, 3.1415926, 2.71828, лише відокремлення цілої частини від дробової задається крапкою, а не комою. Булеві константи `false` і `true` позначають те, про що

ми звикли говорити, відповідно, «хибність» та «істина». Існують також ***текстові константи*** — послідовності будь-яких ASCII-символів, обмежених апострофами, наприклад, **'Ліцей'**, **'3.1415926'**, **'-273'**.

Ім'я — це послідовність букв алфавіту й цифр, що починаються з букви, наприклад, **A**, **x1**, **_1_2**, **jklmn**. Великі й малі букви в іменах не різняться: **Nam**, **nAM**, **nam** — це одне й те саме ім'я. Імена можуть мати довільну довжину, але до уваги беруться лише перші 63 символи.

Ім'я завжди ***іменує*** якийсь об'єкт; воно виділяє його з-поміж інших, тобто ***ідентифікує***, тому ім'я ще називають ***ідентифікатором***.

Деякі імена (це англійські слова або їх скорочення) позначають певні складові частини програми, наприклад, **program**, **const**, **var**, **begin**, **end**. Вони називаються ***зарезервованими***, ***ключовими*** або ***службовими словами***. Існують також ***стандартні*** слова, наприклад, **integer**, **write**, **readln**, **sin**, **Pi** тощо, які мають певний зміст, але його за бажанням програміста можна змінити.

Автор програми також сам створює та використовує ***користувацькі імена***, позначаючи за їх допомогою певні об'єкти програми.

Знак операції — це позначення операції, виконання якої над числами та іншими значеннями породжує нове значення. Значення, до яких застосовується операція, називаються її ***операндами***, а породжуване значення — ***результатом***. Знаки операцій записують як окремими символами, наприклад **+** або **−**, так і іменами або послідовностями інших символів, наприклад **div** або **<=**. Множина знаків уточнюється у параграфах 5–9.

Роздільниками є дужки **( ) [ ]**, розділові знаки **, ; . :** та пропуск. Ними відокремлюються лексеми та інші, складніші, елементи програми.

Коментар — це допоміжне пояснення, яке має на меті допомагати людині зрозуміти програму, не задає ніяких дій і при трансляції пропускається. Коментар — це «майже» довільна послідовність символів, що починається символом **{** і закінчується найближчим **}**, тобто не містить **}** всередині. «Майже» означає, що відразу за дужкою **{** не може бути символу **$** (тоді це не коментар, а так звана директива компілятору). Замість дужок **{** та **}** можна записувати пари символів **(*** та ***)**. Наприклад,

```
{ це коментар }  (* це теж *)  { і це }
а це вже не коментар, а незрозуміло що}
```

Коментарі можна записувати між будь-якими двома словами, проте краще розташовувати їх праворуч від тексту програми або в окремих рядках.

Директиви компілятора — це записи, які задають ті чи інші режими

компіляції програми і можуть суттєво впливати на зміст машинної програми. Як і коментарі, директиви записуються в дужках **{}**, однак відразу за дужкою **{** має йти символ **$**. Конкретний вигляд та зміст деяких директив буде розглянуто нижче.

Суміжні імена й константи відокремлюються так званими ***порожніми символами*** — символами, які не мають зображення й з'являються в тексті програми, якщо при його створенні натискати клавіші **Space** (пропуск), **Enter** («кінець рядка») і **Tab** (символ табуляції). Порожні символи називаються ***пропусками***. Пропуск між сусідніми лексемами не обов'язковий, якщо хоча б одна з них є роздільником, коментарем або знаком операції (але не іменем). Наприклад, знаки **+** або **<=** не є іменами, а **div** — є. Тому **1+2**, **1 div-2** чи **1<=2** написати можна, а **1div2** — ні (треба **1 div 2**).

2. ТИП — ЗНАЧЕННЯ ТА ОПЕРАЦІЇ

Програми обробляють дані у вигляді чисел, символів та інших ***значень***. Значення мають певне представлення в комп'ютері й до значень застосовують деякі операції. За способом представлення та допустимими операціями значення (дані) поділяються на типи. Поняття типу даних є одним із фундаментальних у програмуванні.

Тип даних — це множина значень (даних) разом із множиною операцій, застосованих до цих даних. Множина значень називається ***носієм*** типу, а множина операцій — ***сигнатурою***.

У мовах програмування використовуються числові, символьні та булеві значення. Вони розглядаються як цілісні елементи, що не мають окремих складових частин, тому називаються ***скалярними*** на відміну від ***структурних***, складених з окремих компонентів. Типи скалярних даних називаються ***скалярними***.

Практично кожна мова програмування забезпечує ціле сімейство скалярних типів. Вони називаються ***базовими*** типами мови. Імена базових типів можна вживати, не оголошуючи у програмі (вони є ***стандартними***). Крім того, мови мають спеціальні конструкції для оголошення власних скалярних та структурних типів.

У базових скалярних типах мови Turbo Pascal є цілі та дійсні числа, булеві значення «істина» та «хибність», а також символи.

Будь-який тип, базовий чи створений програмістом, має своє ім'я й характеризується множиною значень та набором можливих операцій.

3. ТИПИ ЦІЛИХ ЧИСЕЛ

Для представлення цілих чисел мова Turbo Pascal надає п'ять типів:

Ім'я типу	Довжина, байт	Діапазон значень
byte	1	0..255
word	2	0..65535
shortint	1	ñ128..127
integer	2	ñ32768..32767
longint	4	ñ2147483648..2147483647

Чому в цілих типів такі діапазони значень, висвітлено в розділі 1. Ціла константа — це, як і в математиці, послідовність десяткових цифр, можливо, зі знаком «**+**» або «**–**» попереду: 0, +1024, –273 тощо. Діапазони можливих констант різних цілих типів наведено в таблиці. Для найбільшого цілого числа типу **integer** виділено ім'я **maxint** (типу **longint** — іменем **maxlongint**). Найбільше за модулем від'ємне число типу **integer** можна задати виразом **–maxint–1**.

Усі цілі типи мають однакові набори операцій. Розглянемо сигнатуру операцій на прикладі типу **integer**.

Основні ***арифметичні операції*** з цілими числами (додавання, віднімання, множення, обчислення частки та остачі від ділення націло, а також ділення) позначають знаками. Більшість операцій є двомісними. Знак «**–**» задає застосування як двомісної операції віднімання, так і одномісної операції «мінус»: –32768, –(2+3). Знак «+» також може задавати як двомісну, так і одномісну операцію.

Арифметичні операції з цілими числами

Знак	Приклади застосування та результати
+	**1+2 = 3**
–	**1–2 = –1**
– (одномісний)	**–(1) = –1**
*****	**2*2 = 4**
div	**7 div 3 = 2, –7 div 3 = –2,** **7 div –3 = –2, –7 div –3 = 2**
mod	**7 mod 3 = 1, –7 mod 3 = –1,** **7 mod –3 = 1, –7 mod –3 = –1**
/	$7/3 \approx 2.3333333$ **(дійсне),** **6/3 = 2.0 (дійсне)**

Арифметичні операції в цілих типах вважають означеними не для всіх можливих значень цих типів, тобто ***частково означеними***.

Неналежне застосування операцій може призводити залежно від налаштування компілятора до неправильних результатів або до аварійного закінчення програми.

Результат додавання, наприклад 32767+1 = 32768 не представляється в типі **integer**, тому може залежати від типу комп'ютера та налаштування компілятора. Можливо, це буде значення 32768 типу **longint**, але не обов'язково.

В операціях ділення **/**, **div**, **mod** ***дільник не може бути нулем***, інакше програма закінчиться аварійно.

Операції **div** та **mod** є такими, що за будь-яких значень **a** та **b** виконується рівність **a div b + a mod b = a**.

Операція **/**, застосована до будь-яких цілих чисел, має результатом ***не*** ціле (дійсне) число.

Операції порівняння цілих чисел задають знаками **=**, **<>**, **>**, **<**, **>=**, **<=** («дорівнює», «не дорівнює», «більше», «менше», «більше або дорівнює», «менше або дорівнює»). Результатом є значення «істина» або «хибність» ***булевого*** типу, який представлено в наступному параграфі. Наприклад, 1 = 2 — **false** («хибність»), 1 <> 2 — **true** («істина»), 1 >= 1 — **true** тощо.

Деякі операції з цілими числами задають у вигляді **f(**…**)**, де **f** — ім'я. Вирази такого вигляду називаються ***викликами функцій***. Розглянемо три такі функції та відзначимо особливості, пов'язані з представленням та обробкою цілих чисел:

Вигляд виклику	Що обчислюється	Приклади
odd(x)	ознака непарності **x** ó **false** або **true**	**odd(7)=true, odd(12)=false**
sqr(x)	x^2	**sqr(2)=4, sqr(181)=32761**
abs(x)	$\|x\|$	**abs(–1)=1**

Наведені функції називаються ***вбудованими***, оскільки їх застосування реалізується підпрограмами зі стандартної бібліотеки системи програмування. Інші вбудовані функції буде розглянуто нижче.

Кількість байтів, відведених під значення числових типів, є фіксованою в системі Turbo Pascal, але вона може відрізнятися в інших системах. Для визначення кількості байтів, відведених під значення певного типу (окрім файлових, див. статтю 10 на с. 45), радимо користуватися функцією **Sizeof**. Наприклад **Sizeof(Longint)=4**, **Sizeof(Word)=2**.

4. БУЛІВ ТИП

Булів тип має два логічних (булевих) значення «хибність» та «істина», які задано константами **false** (хибність) і **true** (істина). Цей тип має ім'я **boolean** на честь видатного англійця Джорджа Буля, засновника математичної логіки.

До булевих значень застосовують логічні операції «і», «або», «не», які називають, відповідно, булевим ***множенням*** (***кон'юнкцією***), булевим ***додаванням*** (***диз'юнкцією***) та ***запереченням*** і позначають як **and**, **or** та **not**. Ще одна операція називається «виключне або» чи «додавання за модулем 2» і позначається знаком **xor**. Результати застосування цих операцій до булевих значень наведено в таблиці:

A	B	A and B	A or B	A xor B	not A
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false

Окрім булевих, є ще операція «***порядковий номер***» **ord**: **ord(false)** = **0**, **ord(true)** = **1**. Порядковим номерам булевих значень відповідає результат їх ***порівняння***: **false < true**. Очевидним шляхом означено всі інші порівняння **=**, **<>**, **>**, **<=**, **>=**.

Булеве значення представляється його номером в одному байті, тобто **Sizeof(boolean)**=**1**.

5. ТИП СИМВОЛІВ

Тип символів (***літерний тип***) має ім'я **char**. У ньому 256 значень, які представляються в одному байті. Зауважимо, що у відносно нових мовах програмування, наприклад Java, символи займають два байти, тому їх кількість — 65536.

Символ позначається ***символьною константою***, тобто символом у апострофах: **'A'**, **'1'**, **'.'** тощо. Сам символ «апостроф» задається символьною константою **''''**. Підкреслимо: символьна константа — це не символ, а його позначення в мові програмування.

Символам взаємно однозначно відповідають ***номери*** від 0 до 255. Будь-яке символьне значення можна задати виразом **chr(*i*)**, де ***i*** має значення від 0 до 255, тобто номер. Наприклад, **chr(48)** позначає те ж, що й

константа **'0'**, **chr(48+1)** — те ж, що **'1'**, **chr(65)** є синонімом константи **'A'**, **chr(97)** — **'a'**. Запис вигляду **#*i*** позначає те ж, що й **chr(i)**; наприклад, **#48** = **chr(48)**.

Відповідність символів і номерів від 0 до 127 зафіксовано в Американському стандартному коді для обміну інформацією (ASCII), а номерам від 128 до 256 можуть відповідати різні набори символів.

Розглянемо операції з символами. Цілий ***номер символа*** породжується викликом функції «порядковий номер», тобто виразом **ord(*c*)**, де значенням ***c*** є символ. Наприклад, **ord('0')** = **48**, **ord('A')** = **65**, **ord('a')** = **97**.

За означенням, функції **chr** і **ord** ***взаємно обернені***, тобто **chr(ord(c))** = **c** за будь-якого символу **c**, а **ord(chr(n))** = **n** за будь-якого **n** від 0 до 255.

Для символів означено всі порівняння, причому символи упорядковано так само, як і відповідні їм номери. Зокрема, справджуються такі нерівності.

' ' < '_' < '0' < '1' < … < '9' < 'A' < 'B' < … < 'Z' < 'a' < 'b' < … < 'z'

Результатом операції ***конкатенації*** (дописування) зі знаком «**+**» є послідовність з двох символів, або ***рядок***. Наприклад, **'1'+'2'** — це послідовність символів, яку можна задати також виразом **'12'**.

6. ПОНЯТТЯ ПЕРЕЛІЧУВАНОГО ТИПУ

Кожен елемент типу **boolean**, **char** або ціле число, окрім найбільшого в своєму типі, має елемент, ***наступний*** за ним, і кожен, окрім найменшого, — ***попередній*** перед ним. Наприклад, наступним за **false** є **true**, за **chr(0)** — **chr(1)**, за **chr(254)** — **chr(255)**, а попереднім перед **true** є **false**. Окрім того, елементи цих типів можна ***пронумерувати*** послідовними цілими числами.

У мові Паскаль ***перелічуваним*** називається тип, у якому означено операції **succ** (наступний), **pred** (попередній) і **ord** (порядковий номер). Перелічувані типи також називаються ***дискретними***.

Операції **succ**, **pred**, **ord** задають у вигляді викликів вбудованих функцій: вирази **pred(1)**, **succ('a')** та **ord(true)** мають значення, відповідно, **0**, **'b'** та **1**. Для знакових числових типів операцію **ord** означено так, що **ord(i)** = **i**, тобто **ord(0)** = **0**, **ord(-1)** = **-1**. В усіх перелічуваних типах означено також порівняння **=**, **<>**, **<**, **>**, **<=**, **>=**.

Результат застосування **succ** до найбільшого елемента типу та **pred** до найменшого залежить від налаштування компілятора.

7. ТИПИ ДІЙСНИХ ЧИСЕЛ

Для представлення дійсних чисел використовують чотири типи: **single**, **real**, **double**, **extended**. Дійсні типи подано в таблиці:

Назва	Довжина, байт	Кількість біт мантиси	Кількість біт порядку	Найменше за модулем	Найбільше за модулем
single	4	23	8	$\approx 1.5\cdot10^{-45}$	$\approx 3.4\cdot10^{38}$
real	6	39	8	$\approx 2.9\cdot10^{-39}$	$\approx 1.7\cdot10^{38}$
double	8	52	11	$\approx 5.0\cdot10^{-324}$	$\approx 1.1\cdot10^{308}$
extended	10	64	15	$\approx 3.4\cdot10^{-4932}$	$\approx 1.1\cdot10^{4932}$

Як видно з таблиці, тип **extended** має найбільший діапазон та найвищу точність представлення. У процесорі числа обробляються в представленні саме типу **extended**, і під час запису в регістри процесора числа з інших типів перетворюються в цей.

Тип **real** є ***стандартним*** (доступним для використання незалежно від встановленого режиму компіляції). Щоб використовувати інші дійсні типи, треба встановити спеціальний режим компіляції за допомогою меню системи програмування або директиви компілятора.

Дійсні числа позначаються дійсними константами. Розглянемо приклад. Число **1,2345** можна позначити кількома способами, наприклад, $123,45\cdot10^{-2}$. У цьому записі воно має цілу частину «**123**», дробову частину «**,45**» і десятковий порядок «**–2**». Запису відповідає константа мови Паскаль **123.45E-2**, в якій **123** — ***ціла*** частина, **.45** — ***дробова***, а **E-2** — ***порядок***. Це саме число можна представити також константами **0.12345E1**, **0.012345E+2**, **1.2345**, **12345e-04** та іншими.

Дійсні константи мають ***обов'язкову цілу частину***, за якою записано ***дробову частину*** і ***порядок*** (можливо, одне з них). Ціла частина — це непорожня послідовність цифр, дробова — послідовність цифр, можливо, порожня, із крапкою на початку, а порядок — латинська буква **E** або **e** з однією або кількома цифрами, можливо, зі знаком **+** або **–**. Перед константою може стояти знак **–**, і тоді вона задає від'ємне число: **–12.345E–1**.

Представлення дійсного числа константою, у якій ціла частина містить одну цифру від **1** до **9**, називається ***нормалізованим***, наприклад, **9.81** або **1.0E2** (для числа 0 таким є **0.0**).

Множина чисел, які можна представити у дійсних типах, уточнюється за поданою вище таблицею. За будь-якого дійсного типу вона є ***скінчен-***

ною обмеженою підмножиною множини раціональних чисел і містить усі цілі числа, які представлено в типі **longint** (тип **single** містить не всі, але містить носій типу **integer**). Різниця між двома «сусідніми» дійсними числами змінюється в їх діапазоні, для великих чисел досягаючи порядку 10^4.

Операції. До дійсних значень застосовують ті ж ***арифметичні*** операції, що й до цілих, за винятком **odd**, **div**, **mod**, а також **succ**, **pred**, **ord**. Результатом цих операцій завжди є дійсне число. ***Порівняння*** дійсних чисел, як і цілих, має відповідний булів результат: результатом **1.0<=2.5** є значення **true**, **1.0=0.99** — значення **false**.

Дійсні типи є «машинними представниками» множини дійсних чисел, яка є ***всюди щільною*** (між довільними двома дійсними числами існує безліч інших дійсних). Пронумерувати ці числа неможливо, тому ***дійсні типи не є перелічуваними***.

Для дійсних типів означено так звані вбудовані математичні функції. Дійсне значення $|x|$, arctg x, cos x, e^x, ln x, sin x, x^2, $\sqrt{x}$ повертається з виклику функції, ім'я якої, відповідно,

abs, arctan, cos, exp, ln, sin, sqr, sqrt.

Усі ці функції застосовуються й до чисел цілих типів (але лише **abs** та **sqr** у цьому випадку породжують значення цілого типу). Значення аргументу у викликах функцій **cos** і **sin** виражає кількість радіан, а не градусів. Означено також нульмісну функцію **Pi** (її значення є наближенням до числа $\pi \approx 3{,}14159265358979$). Наприклад, **cos(Pi/2)** = **0.0**, **sin(Pi/2)** = **1.0**.

Значення математичних функцій обчислюються наближено, і для деяких аргументів можуть відрізнятися від справжніх математичних.

Для дійсних типів означено декілька специфічних операцій, які задаються викликами функцій. Їх подано в таблиці (означення **trunc**, **int** і **frac** для від'ємних чисел відрізняється від математичного):

Функції, специфічні для дійсних типів

Вигляд виклику	Що обчислюється	Приклади та примітки
round(x)	найближче до **x** ціле число (значення типу **longint**)	**round**(4.12)=4, **round**(1.5)=2, **round**(2.14748364749E9) = 2147483647. при $x \geq 2.1474836475E9$ не визначено (тоді **round(x)** не можна представити в **longint**)

Закінчення

Вигляд виклику	Що обчислюється	Приклади та примітки
`trunc(x)`	ціла частина **x** (значення типу **longint**)	**trunc**(ñ4.12) = ñ4, **trunc**(1.9) = 1, **trunc**(*x*) = 2147483647 при $x \geq 2147483647$
`int(x)`	ціла частина **x** (значення того ж типу, що й **x**)	**int**(1.9) = 1.0, **int**(ñ1.9) = ñ1
`frac(x)`	дробова частина **x** (значення того ж типу, що й **x**)	**frac**(1.6) = 0.6, **frac**(ñ1.6) = ñ0.6

8. ЗМІННІ ВЕЛИЧИНИ

Поняття змінної числової величини вперше з'явилося в роботах Декарта в XVII ст. Пізніше виявилося, що змінна величина може бути не лише числовою. У найширшому розумінні ***змінна величина*** — це узагальнення, абстракція якогось реального чи уявного об'єкта (або його окремої характеристики), який може перебувати в різних станах. Змінні величини задають іменами, наприклад, у записі другого закону Ньютона $a = F/m$ імена *m, a, F* позначають змінні величини — масу тіла, прискорення його руху та силу, що діє на нього.

У програмуванні змінна також є представником об'єкта або його ***моделлю*** у програмі. Вона представляє його в пам'яті комп'ютера, тому ***змінна в програмуванні*** — це ділянка пам'яті, яка своїми станами представляє стани об'єкта.

У мовах високого рівня змінні позначаються іменами, або ідентифікаторами.[1] Імена можна вибирати будь-які, окрім службових слів. Наприклад, **ABRACADABRA**, **temperature**, **number**, **f123qq** тощо. Нагадаємо: великі й малі букви в іменах розглядаються як однакові.

Імена змінних та інших об'єктів бажано обирати так, щоб вони були пов'язані з поняттями, які вони представляють, наприклад, **tCels**, **tKelv** або **KelvC** (температура за Цельсієм, за Кельвіном, константа Кельвіна).[2]

Змінна величина у математиці вважається заданою, якщо визначено множину значень, яких вона може набувати. У Паскаль-програмі змінна

[1] Змінні можуть позначатися також складнішими виразами (див. статті 7, 9 та 11).

[2] Інколи перші букви імені змінної обираються так, щоб вони вказували на її тип.

задається оголошенням, яке вказує ім'я та тип змінної й має такий вигляд.

```
var ім'я змінної : ім'я типу;
```

Наприклад, `var even : boolean;`. Ключове слово **var** є скороченням англійського *variable* — змінна. Кілька однотипних змінних можна оголосити разом, указавши їх імена через кому.

```
var tCels, tKelv : integer;
```

Тип змінної задає множину її можливих значень (що залежить від довжини ділянки пам'яті, яку займає змінна) та операцій, які можна застосувати до неї.

Кожна ділянка пам'яті комп'ютера має ***адресу***, якою ділянка позначається в машинній програмі. Ім'я змінної у Паскаль-програмі в результаті трансляції перетворюється на адресу деякої ділянки пам'яті. При виконанні програми ця ділянка і є змінною, ім'я якої записано в програмі. Ім'я змінної ***вказує*** або ***посилається*** на ділянку пам'яті під час виконання програми (рис. 1). Як бачимо, змінна (ділянка пам'яті) та її позначення (ім'я) — це ***різні поняття***.

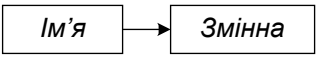


Рис. 1. Змінна та її ім'я

Способи, якими змінним надаються значення, буде описано нижче, а в наступному параграфі будемо використовувати імена змінних у виразах, вважаючи, що вони мають значення (неважливо, які саме). Отримання значення змінної за її іменем називається її ***розіменуванням***.

9. ВИРАЗИ

Застосування операцій до числових та інших значень, у результаті якого утворюється нове значення, у мовах програмування високого рівня задається за допомогою ***виразу***.

Найпростіші вирази — це константи й імена змінних базових типів.

Складніший вираз утворюється з простіших як:

- вираз у дужках;
- два вирази і знак двомісної операції між ними;
- вираз із знаком одномісної операції перед ним;
- виклик функції з виразом у дужках.

Приклади: `((1)), true and false, 1-(2+3), (1-2)+3, (1+2)<>3, -(5+3), odd(2), ord(odd(15))`.

Значення, до якого застосовується операція, називається її ***операндом***. Операнди позначаються виразами. Послідовність застосування операцій до операндів утворює ***процес обчислення виразу***, результатом якого є ***значення виразу***. Тип значення виразу називається ***типом виразу***.

Як бачимо, вираз має ***подвійний зміст*** (***семантику***): задає процес обчислення й має значення певного типу.

Приклади. Вираз **2*2 = 5** задає процес, у якому:

1) обчислюється добуток **2*2** і виникає ціле значення **4**;

2) порівнюються цілі значення **4** і **5** і виникає значення **false**, яке є значенням цього виразу; його тип — **boolean**.

Нехай **a**, **b**, **c** — імена числових змінних, які представляють коефіцієнти квадратного рівняння $ax^2 + bx + c = 0$. При обчисленні виразу **b*b–4*a*c>0** спочатку обчислюється квадрат значення змінної **b**, потім добуток числа **4** та значень змінних **a** і **c**, потім різниця двох отриманих добутків. Ця різниця порівнюється з нулем, і булів результат цього порівняння (значення **true** або **false**) є значенням усього виразу. Отже, цей вираз задає обчислення ознаки того, що вказане рівняння має саме два корені.

Припустимо, що **c** — ім'я символьної змінної, значенням якої може бути один із символів **'0'**, **'1'**, …, **'9'**, тобто десяткова цифра. Вираз **ord(c)–ord('0')** задає обчислення порядкових номерів значення змінної **c** та символа **'0'**. Потім обчислюється різниця цих номерів — ціле число від **0** до **9**, тобто числове значення цифри. ▣

Мова Turbo Pascal, в основному, відповідає угодам математики про порядок застосування операцій у виразах. Це дозволяє не записувати зайві дужки, наприклад, **1–2*3** означає те саме, що й **1–(2*3)**. На порядок застосування операцій за відсутності дужок впливає їх ***старшинство***, або ***пріоритет***.

Якщо поруч із позначенням операнда записано знаки двох операцій, то спочатку виконується старша з них (із вищим пріоритетом).

У таблиці, поданій нижче, усі операції розбито на чотири групи, розташовані у порядку спадання пріоритету (деякі операції означено у подальших главах). Операції всередині кожної групи мають однакові пріоритети. Наприклад, вираз **1+(3+2)*2** задає, що після обчислення **3+2**, тобто значення **5**, воно множиться на **2**, а не додається до **1**.

Окрім пріоритетів, операції мають властивості право- або лівобічного зв'язування. У мові Turbo Pascal усі двомісні операції мають властивість ***лівобічного зв'язування***: якщо ліворуч і праворуч від позначення операнда записано знаки операцій з однаковим старшинством, то операнд

зв'язується з операцією, вказаною ліворуч (ця операція застосовується спочатку). Наприклад, **1–2*4+3** задає те саме, що й **(1–2*4)+3**, але не те, що **1–(2*4+3)**.

Старшинство операцій

Загальна назва операцій	Символи операцій
Унарні	**–, not, ^, @**
Мультиплікативні	***, /, and, mod, div, shl, shr**
Адитивні	**+, –, or, xor**
Відношення	**=, <>, <, >, <=, >=, in**

У системі програмування Turbo Pascal застосовується так зване ***ледаче***, або ***скорочене***, ***обчислення*** булевих операцій **and** і **or**. Спочатку обчислюється їх перший операнд. Якщо для операції **and** це **false**, то другий операнд не обчислюється, оскільки результатом все одно буде **false**. Аналогічно, якщо перший операнд операції **or** є **true**, то другий операнд не потрібен. Наприклад, вираз **(2*2=5) and 323345 div 17 = 0)** задає обчислення лише **2*2=5**, а **(2*2=4) or (323345 div 17 = 0)** — лише **2*2=4**. Повністю булеві вирази обчислюються в спеціальному режимі компіляції (див. додатки А і В).

10. ПРИСВОЮВАННЯ ЗНАЧЕННЯ ЗМІННІЙ

Змінна своїми значеннями представляє стани об'єктів, які моделюються в програмі. Основний спосіб надати значення змінній — це ***оператор присвоювання***. У літературі оператори ще називають ***командами***, ***вказівками*** та ***інструкціями***.

Оператор присвоювання має такий вигляд:

```
ім'я змінної := вираз;
```

Знак присвоювання **:=** — це окрема лексема, яку не слід плутати зі знаком операції порівняння **=**.

Оператор присвоювання позначає такі дії.

1. Обчислити значення виразу, записаного праворуч від знака **:=**.

2. Записати обчислене значення в змінну, позначену іменем ліворуч.[3]

Наприклад, якщо ім'я **z** оголошено як **var z:integer**, то оператор присвоювання **z:=7*(3+1)** задає обчислення значення **28** і запис його в змінну **z**. Після виконання присвоювання змінна **z** має значення **28**.

Оператор присвоювання задає зміну значення (стану) змінної. У вира-

[3] Нагадаємо, що змінні можуть позначатися також складнішими виразами (див. статті 7, 9 та 11).

зах в операторах присвоювання можна записувати імена змінних, яким присвоєно значення за ***попередніми*** операторами програми. Значенням виразу, що є іменем змінної, є її значення, присвоєне їй раніше.

Наприклад, якщо у програмі змінній **z** присвоєно значення **2**, то вираз **z+1** задає додавання **2** та **1** і має значення **3**. При іншому значенні **z** вираз **z+1** мав би інше значення. Якщо цей вираз написати, наприклад, в операторі **z:=z+1**, то оператор задає збільшення значення **z** на **1**, яким би воно не було (звичайно, якесь значення змінній **z** треба присвоїти перед виконанням цього оператора).

Сукупність змінних, імена яких оголошено у програмі, називається ***пам'яттю програми***,[4] а відповідність між іменами змінних та їх станами — ***станом пам'яті програми***. Присвоювання значення змінній означає ***зміну стану пам'яті програми***.

Ім'я змінної у виразі задає її значення на момент обчислення виразу. Отже, вираз обчислюється при поточних значеннях вказаних у ньому змінних, тобто ***на поточному стані пам'яті***.

Далі ми розглянемо інші види операторів, але ***всі оператори задають зміну станів пам'яті програми***. Ця зміна й є їхньою семантикою.

Для перелічуваних типів існує специфічний різновид оператора присвоювання. Він записується як **inc(z)** або **dec(z)**, де **z** — ім'я змінної перелічуваного типу. У дійсності ці оператори є викликами ***вбудованих процедур*** (про процедури йдеться у розділі 6). Виклик **inc(z)** тотожний оператору **z:=succ(z)**, **dec(z)** — оператору **z:=pred(z)**.

Після імені змінної та коми у виклику **inc** та **dec** можна записати цілий вираз. Наприклад, виклик **inc(z,2)** задає збільшення **z** на дві «одиниці» того типу, до якого належить **z**. Якщо змінна **z** типу **char** має значення **'A'**, то після виконання **inc(z,2)** її значенням буде **'C'**. Значення виразу у виклику може бути й від'ємним — тоді **z** зменшиться. Аналогічно, виконання **dec(z,2)** зменшує значення **z** у його типі, наприклад, з **3** до **1** або з **'C'** до **'A'**.

11. СТРУКТУРА ТА ПРИКЛАДИ ПРОГРАМ

Програма мовою Turbo Pascal має такий загальний вигляд:

```
program ім'я;
  розділ підключення модулів
```

[4] Далі ми побачимо, що у процесі виконання програми до її пам'яті змінні можуть додаватися або, навпаки, вилучатися з пам'яті.

```
  оголошення імен та підпрограми
begin
  опис виконуваних дій
end.
```

У першому рядку записано ***заголовок програми***; він містить ім'я програми й не є обов'язковим. Проте краще взяти собі за правило завжди його записувати.

Розділ підключення модулів (він теж не є обов'язковим) починається службовим словом **uses** і містить перелік імен модулів. Докладніше про модулі йдеться в статті 10, а зараз дамо дуже стисле пояснення. Програми часто створюються у вигляді кількох ***програмних одиниць*** — власне програми та цілого набору ***модулів***, які в ній використовуються. Модулі зберігаються й транслюються окремо, а їхні «машинні» варіанти підключаються до програми при компонуванні. Щоб забезпечити підключення потрібних модулів, на початку програми вказують їхні імена.

Оголошення імені — це опис того, що позначає ім'я у програмі. Ім'я може позначати деяке значення або ділянку пам'яті, в якій зберігаються значення, а також інші, складніші об'єкти. Конкретні види оголошень імен розглядатимуться разом із уточненням відповідних понять (змінних, констант, типів та підпрограм). Кожне з оголошень закінчується роздільником «**;**».

Підпрограма — це спеціальним чином оформлена частина програми. Якщо програма описує дії з розв'язання деякої задачі, то підпрограма описує дії з розв'язання деякої частини цієї задачі, тобто ***підзадачі***. Цей термін буде уточнено в статті 6.

Кожне ім'я, що використовується в програмі, має бути оголошеним. Деякі імена оголошуються в системі програмування або в інших програмних одиницях.

Опис виконуваних дій разом із «дужками» **begin–end** називається ***тілом*** програми. Після тіла обов'язковою є ***крапка*** — символ кінця програмної одиниці. Текст програми між її заголовком та крапкою, тобто оголошення та тіло, називається ***блоком***.

Дії в програмі задаються послідовно записаними ***командами***, або ***операторами***. У літературі оператори ще називають ***вказівками*** та ***інструкціями***.

Приклади. Найкоротша програма, яка не задає жодних дій, має такий вигляд:

```
begin end.
```

У першому рядку наступної програми записано заголовок з її іменем **First** (тобто «програма Перша»). Запис **writeln('Привіт!')** оз-

начає дію: надрукувати на екрані комп'ютера символи, вказані в апострофах. Ім'я **writeln** є скороченням англійського ***write line*** — записати рядок.

```
program First;
begin
  writeln('Привіт!')
end.
```

Наступна програма містить пояснення у фігурних дужках — коментарі, призначені для читача.

```
program Second; { «програма Друга» }
  const KelvC = 273;
  {ім'я KelvC далі позначає число 273}
  var tCels, tKelv : integer;
  {оголошення імен змінних}
begin
 writeln('Обчислення температури за Кельвіном.');
 writeln('Температура за Цельсієм (ціле число)>');
 {запросити до введення цілого числа}
 readln(tCels);{прочитати з клавіатури число}
               {й запам'ятати у змінній tCels}
 tKelv:=tCels+KelvC;{присвоїти суму змінній tKelv}
 writeln('Температура за Кельвіном: ', tKelv);
 { надрукувати текст і суму }
end.
```

За цією програмою комп'ютер спочатку має вивести на екран текст:

```
Обчислення температури за Кельвіном.
Температура за Цельсієм (ціле число)>
```

Далі указано дію з коментарем «прочитати з клавіатури ціле число й запам'ятати у змінній **tCels**». При виконанні цієї дії комп'ютер має чекати, поки ми не наберемо якесь ціле число (наприклад, **127**) і не натиснемо на клавішу **Enter**. Потім він має обчислити суму введеного числа з числом **273**, для якого на початку програми введено позначення **KelvC**. Цю суму **400** (на **273** більше, ніж ми набрали) він має присвоїти змінній **tKelv**. Нарешті, він має вивести текст і значення змінної **tKelv**:

```
Температура за Кельвіном: 400
```

Отже, за цією програмою комп'ютер отримує від клавіатури ціле число, що задає температуру за Цельсієм, обчислює й запам'ятовує температуру за Кельвіном і виводить її значення на екран. ▣

Як бачимо, оператори присвоювання та інші оператори записуються один за одним і відокремлюються роздільником «**;**». Оператори, записані один за одним, утворюють ***послідовність операторів***.

Виконання операторів програми можна ***проімітувати***, указавши послідовність операторів і станів пам'яті програми, після виконання операторів. Якщо в процесі виконання програми змінна ще не отримала значення, будемо вважати його невизначеним і позначати як «**?**».

Розглянемо таку програму:

```
program a2;
  var x, y, z : integer;
begin
  z:=1;
  x:=3+z;
  y:=15-x;
  x:=x+1
end.
```

Її імітацію подамо таблицею:

Виконано оператор	Утворено стан пам'яті		
	x	y	z
	?	?	?
z:=1	?	?	1
x:=3+z	4	?	1
y:=15-x	4	9	1
x:=x+1	5	9	1

▣

• Імітацію програми можна провести за допомогою її покрокового виконання. Замість невизначеності початковими значеннями змінних будуть нулі. Проте на цю властивість системи програмування покладатися ***не варто***.

12. ВИВЕДЕННЯ ЗНАЧЕНЬ ВИРАЗІВ НА ЕКРАН

Мови програмування, як правило, не мають спеціальних операторів для виведення значень з оперативної пам'яті на зовнішні носії та уведення їх з носіїв до оперативної пам'яті. Ці операції здійснюються за спеціальними ***підпрограмами*** (***процедурами***) ***уведення-виведення***. Розглянемо підпрограми виведення.

Виведення, або запис, значення виразу на екран задається у вигляді виклику процедури **writeln(*вираз*)**.[5] Виклик процедури, на відміну від виклику функції, записується не у виразі, а ***як окремий оператор***.

При виконанні підпрограми **writeln** обчислюється значення виразу і за ним створюється відповідна константа, тобто послідовність символів. Константа передається пристрою, частиною якого є екран, і виводиться на нього. Типом виразу може бути лише базовий тип.

Наприклад, при виконанні операторів програми з цілою змінною **z**

```
z:=1; writeln(1+z); writeln(z<=1);
```

на екрані з'являються такі символи:

```
2
TRUE
```

На екрані майже завжди наявна світлова позначка, що миготить, — курсор. При виконанні підпрограми виведення константа друкується, починаючи з того місця на екрані, де перебуває курсор. Після виведення за допомогою процедури **writeln** курсор переходить до наступного рядка екрана.

Всередині дужок **writeln** можна написати кілька виразів через кому. Вони обчислюються один за одним, і відповідні їм константи друкуються підряд в одному рядку; курсор пересувається в наступний рядок після виведення останньої константи.

Наприклад, у результаті виконання операторів

```
z:=1; writeln(1+z, z<=1);
```

на екрані з'явиться **2TRUE**.

Разом із виразами можна записувати послідовності символів між апострофами, наприклад: **'x='**, **'123'** тощо. Вони називаються рядковими константами, або ***літералами***, і виводяться без апострофів.

Наприклад, за **x** цілого типу при виконанні

```
x:=2; writeln(1, ') x = ', x, '; x**2 = ',
x*x, '; x^2 > 2 : ', x*x > 2);
```

на екран виводиться такий текст:

```
1) x = 2; x**2 = 4; x^2 > 2 : TRUE
```

За виразом можна написати двокрапку й цілу константу, наприклад, **x+y:10**. Константа задає так звану ***ширину поля виведення***, до якого виводяться символи, що представляють значення виразу. Якщо символів більше, ніж ширина, то друкуються всі символи, а якщо менше, то спочат-

---

[5] Нагадаємо: ім'я **writeln** є скороченням англійського *write line* — записати рядок.

ку будуть виведені пропуски (доповнивши кількість символів до заданої ширини поля виведення), а потім символи.

Наприклад, при виконанні операторів

```
x:=10; writeln('x = ', x:5, ', x**2 = ', x*x:1)
```

на екран перед **10** виводиться три пропуски та всі цифри числа **100**.

```
x =    10, x**2 = 100
```

Процедура **write** відрізняється від **writeln** лише тим, що після виведення останньої константи курсор не пересувається на наступний рядок екрана. Наприклад, при виконанні операторів

```
x:=2;
write('x = ',x);
write(', x**2 > 2 : ',x*x>2)
```

на екран виводиться такий текст:

```
x = 2, x**2 > 2 : TRUE
```

Курсор залишається праворуч від останньої букви.

На початку тіла програми варто записувати ***виклик процедури виведення з повідомленням*** щодо самої програми та її призначення.

За допомогою процедур **writeln** і **write** можна виводити значення не лише на екран, а й на інші зовнішні носії, наприклад, диск. Про це йдеться в статті 7.

13. УВЕДЕННЯ ЗНАЧЕНЬ У ЗМІННІ

Уведення даних із зовнішніх носіїв до змінних програми задається ***процедурами уведення***, або ***читання***. Виклик однієї з них у найпростішому випадку має вигляд **readln(*ім'я-змінної*)**. Ім'я **readln** є скороченням англійського *read line* — прочитати рядок. При виконанні такого виклику комп'ютер зупиняється й чекає, що на клавіатурі буде набрано константу того ж типу, що й тип указаної змінної. У відповідь слід набрати якусь константу на клавіатурі (її буде показано на екрані) та натиснути на клавішу **Enter**.

Набрана константа після натискання на **Enter** передається процедурі, яка за константою створює відповідне значення та присвоює вказаній змінній. Типом змінної може бути лише базовий тип.

Якщо натиснути **Enter**, не набравши нічого, окрім пропусків, то комп'ютер і надалі чекатиме. Перед числовою константою можна набрати довільне число пропусків (система їх ігнорує).

Якщо замість потрібної константи набрати інші символи (наприклад, недопустиму константу або замість числової константи символьну), то ви-

конання програми на цьому буде завершено (аварійно) і на екрані з'явиться повідомлення про те, що вхідні символи були неправильними. Нехай, наприклад, ім'я **z** позначає змінну типу **integer**. Якщо при виконанні **readln(z)** набрати **1024**, то після натискання на **Enter** значенням **z** стане **1024**. Якщо ж набрати **50.9** або **z=1024**, програму буде аварійно завершено, оскільки число **50.9** не подається в типі **integer** та ціла константа не може починатися символами «**z=**». Якщо ж набрати ціле число, яке виходить за межі типу, наприклад, **50000000**, результат залежить від встановлених директив компіляції і може бути цілком неочікуваним.

У виклику процедури читання можна написати кілька імен змінних, відокремивши їх комами. При виконанні цього виклику треба набрати відповідну кількість констант, відокремивши їх пропусками або натисканнями на **Enter** у довільній кількості. Поки всі константи не буде набрано, виконання виклику не закінчується. Нехай, наприклад, **x:integer**, **y:real**. При виконанні **readln(x,y);** слід набрати цілу константу, додати хоча б один пропуск або натискання на **Enter**, а потім набрати будь-яку цілу або дійсну константу (між цілою та дробовою частиною набрати десяткову крапку) і натиснути на **Enter**.

• Практично ***перед кожним викликом процедури читання*** з клавіатури варто записати ***виклик процедури виведення*** із запрошенням до введення значень і вказівкою, скільки й яких типів.

Наприклад, указати виведення запрошення

writeln('Температура за Цельсієм (ціле число)>'),

а потім виклик **readln(tCels);**.

Процедура **readln** задає читання не лише з клавіатури, але й з інших зовнішніх носіїв, наприклад, з диску (див. статтю 7).

14. ІМЕНУВАННЯ ВИРАЗІВ З КОНСТАНТАМИ ТА ІНІЦІАЛІЗАЦІЯ ЗМІННИХ

Вираз із константами, записаний у програмі, обчислюється не при виконанні програми, а у процесі трансляції. Значення такого виразу можна позначити іменем (***іменувати***) і використовувати це ім'я далі в програмі.

Іменування має такий вигляд:

const ім'я константи = вираз із константами;

Ключове слово **const** означає «константа»; остання «**;**» обов'язкова.[6]

[6] В іменуваннях можна викликати лише 14 з усіх системних функцій.

Іменування виразу є ***оголошенням імені***, і це ім'я можна записувати нижче у програмі замість виразу.

Корисно іменувати вирази, що записуються у багатьох місцях програми. Якщо вираз іменований, то зміни слід внести лише в іменування, а якщо ні — доведеться змінювати вираз усюди, де він зустрічається.

За словом **const** можна записати кілька іменувань, відокремивши їх «**;**», причому у виразах можна використовувати імена вже іменованих виразів із константами.

Ось приклад:

```
const a=12; b=2*a; tt=a+b;
```

Ім'я **tt** після цього позначатиме число **36**. Оголошення імен у Паскаль-програмі часто починають саме з іменування виразів.

Імена констант, як і константи та імена змінних, записують у виразах. Вони також розіменовуються, наприклад, нехай ім'я **Kelvc** оголошено як **const Kelvc=273**. Тоді в операторі **tKelv := tCels+Kelvc;** розіменовуються і **tCels**, і **Kelvc** (тільки **Kelvc** ще під час трансляції, а не за виконання програми).

Оголошення змінної з присвоюванням їй початкового значення називається ***ініціалізацією***. У мові Turbo Pascal ініціалізація має дещо дивний вигляд. Оголошення ***змінної*** починається словом **const** і замість знака присвоювання записується знак «**=**», тому ініціалізовані змінні ще називають ***типізованими константами***.

Ось приклад:

```
const bool:boolean=true; kb8:integer=1024*8;
```

Тут оголошено ім'я булевої змінної **bool** з початковим значенням **true** та ім'я цілої змінної **kb8** з початковим значенням **1024*8 = 8192.**

Вираз, який ініціалізує змінну, обчислюється під час трансляції програми, тому може містити константні вирази[7] та імена константних виразів, оголошені вище у програмі. ***Імена змінних*** (зокрема, ініціалізованих раніше) у ньому ***заборонені***. Наприклад, перший з двох наступних рядків містить правильну послідовність оголошень, а другий — помилкову.

```
const kb1=1024;
kb8:integer=8*kb1;
const kb1:integer=1024;
kb8:integer=8*kb1;{???}
```

[7] З урахуванням обмежень на виклики функцій.

15. СУМІСНІСТЬ ТИПІВ

Сумісність числових типів у виразах. У всіх типах цілих та дійсних чисел означено одні й ті самі операції: **+**, **-**, **/**, *****, порівняння та інші. Проте, наприклад, додавання цілих та дійсних чисел відбувається по-різному, тобто однакові знаки позначають у дійсності різні операції. Наприклад, виразу **1+2** відповідає додавання цілих чисел, а виразу **1.0+2.0** — дійсних. Яку саме операцію слід додати до машинної програми, компілятор визначає за знаком операції та типами операндів.

Мова Turbo Pascal дозволяє задавати у виразах дійсні та цілі операнди разом, наприклад, **1+2.0**. При трансляції таких виразів додаються команди породження дійсного значення за цілим операндом. Отже, якщо один із операндів має дійсний тип, при обчисленні виразу спочатку цілий операнд перетворюється на дійсний і потім виконується вказана операція над дійсними значеннями. Так, при обчисленні **1+2.0** спочатку **1** перетворюється на **1.0** і потім додаються **1.0** і **2.0**.

Можливість указування операндів різних типів у виразах називається ***сумісністю*** цих типів. Типи цілих і дійсних є сумісними.

Цілі типи між собою також сумісні. Проте результат перетворення цілого значення до іншого типу може залежати від того, змінні чи константи записано у виразі, а також від конкретних значень та дії директив компілятора. Правила перетворення цілих типів тут не наводяться.

Сумісність типів за присвоюванням — це можливість значення одного типу присвоювати змінним іншого типу.

Дійсні типи сумісні за присвоюванням із цілими, але не навпаки. Наприклад, якщо діє оголошення **a:real; b:integer;**, то присвоювання **b:=a;** є хибним, але можна написати **a:=b**. Ціле значення **b** перед присвоюванням дійсній змінній **a** перетворюється на дійсне. Так само при виконанні **readln(a);** можна набрати на клавіатурі не дійсну, а цілу константу — **a** одержить дійсне значення. Зворотні перетворення програміст повинен задавати явно за допомогою функцій **trunc** або **round**, наприклад, **b:=round(a)**.

Дійсні типи сумісні за присвоюванням між собою. Проте значення більшого типу перетворюється на значення меншого за допомогою округлення. Якщо після округлення значення не подається в меншому типі, програма аварійно закінчується.

Цілі типи також сумісні за присвоюванням, проте з рядом особливостей, які тут не подано. Зазначимо лише, що значення меншого розміру (у байтах) утворюються за рахунок відкидання старших байтів у значенні,

більшому за розміром, тобто ***можливі помилкові результати або аварійне завершення програми***.

16. ПОБІТОВІ ОПЕРАЦІЇ З ЦІЛИМИ ЧИСЛАМИ

Для цілих типів означено двомісні операції **and**, **or**, **xor** та одномісну **not**. Вони застосовуються як відповідні булеві операції, але окремо по бітах операндів, і називаються ***побітовими булевими***. Булевому значенню **false** відповідає біт **0**, **true** — біт **1**.

Приклад. Припустимо, змінні **x** і **y** мають тип **shortint** та значення **5** і **6**. Тоді **x and y** = **4**, **x or y** = **7**, **x xor y** = **3**, **not x** = **-6**. Якби ці змінні мали тип **byte**, то при тих самих значеннях **not x** = **250**.

Щоб зрозуміти ці результати, запишемо двійкові записи значень змінних та виразів, а на їх основі й десяткові записи:

Вираз	Двійковий запис значення								Десятковий запис
x	0	0	0	0	0	1	0	1	5
not x	1	1	1	1	1	0	1	0	250 (як **byte**), ñ6 (як **shortint**)
y	0	0	0	0	0	1	1	0	6
x and y	0	0	0	0	0	1	0	0	4
x or y	0	0	0	0	0	1	1	1	7
x xor y	0	0	0	0	0	0	1	1	3

Для цілих чисел є ще операції зсуву **shl** і **shr** (***shift left*** та ***shift right*** — зсув ліворуч та зсув праворуч). При виконанні операції **i shl k** (**i shr k**), де **k** ≥ 0, біти в значенні **i** зсуваються ліворуч (праворуч) на **k** розрядів. Молодші (старші) біти при цьому заповнюються значенням **0**, а старші (молодші) — втрачаються.

Наприклад: **6 shl 2** = **24**, **6 shr 2** = **1**.

Операції зсуву цілих чисел виконуються у процесорі набагато швидше, ніж операції множення та ділення, тому, наприклад, цілочислове ділення невід'ємного значення **i** на **2** варто задавати як **i shr 1**.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які основні групи символів є в алфавіті мови Turbo Pascal?

2. Які види лексичних одиниць є в мові Turbo Pascal?

3. Які різновиди імен є в мові Turbo pascal? Чим вони відрізняються?

4. Чи можна створити змінну або константу з іменем ***Pi***, ***sin***, ***program***, ***integer***, ***const***?

5. З яких основних структурних частин складається програма, записана мовою Turbo Pascal?

6. Що таке тип даних?

7. Що таке переповнення?

8. Які набори операцій є сигнатурами типів **Boolean**, **integer**, **real** *та* **char**?

9. Які вбудовані функції означено для типів **Boolean**, **integer**, **real** *та* **char**?

10. Чому типи **Boolean**, **integer** *та* **char** *називаються дискретними, а* **real** *— ні?*

11. Які пріоритети мають операції у виразах?

12. Яка арифметична операція (відсутня в мові Turbo Pascal) повинна мати властивість правобічного зв'язування?

13. Чим відрізняється іменування константи від ініціалізації змінної?

14. Що таке команда (оператор) присвоювання?

15. Що є семантикою послідовності операторів присвоювання?

16. Чи можна імена типізованих констант вказувати у викликах процедур уведення?

17. Що таке сумісність типів? Назвіть приклади сумісних типів даних.

18. Що таке сумісність типів за присвоюванням? Назвіть приклади типів, сумісних за присвоюванням.

19. Чому цілі типи не сумісні за присвоюванням з дійсними?

ЗАДАЧІ

1. Подати булеву операцію **xor** через інші булеві операції.

2. Знайти найбільше натуральне число, квадрат якого подається в типі **Longint**.

3. Обчислити значення виразу:

a) **(2*2 = 4) and true**;

b) **(2*2 = 4) or false**;

c) **(not true) or false**;

d) **(ord(true) = 1) xor (ord(false) = 0)**.

4. Обчислити значення виразу:

a) **2*ord(true) + 3*ord(false)**;

b) **(false = true) = false**;

c) **58 mod 13 div 10**;

d) **9 mod 5 * 12 div 16**.

5. Якщо вираз не є помилковим, обчислити його значення, а якщо є, пояснити, чому:

a) **1 = 2 = 3**;

b) **false = true and false**.

6. Пояснити різницю між записами **0** і **'0'**, **A** і **'A'**, **-** і **'-'**.

7. Обчислити значення виразу:

a) **chr(ord('0') + 9)**; b) **chr(ord('A') + 25)**;

c) **chr(ord('0') – 16)**; d) **'Z' > 'a'**;

e) **ord('9') – ord('0')**; f) **ord('F') – ord('A')**.

8. Написати вираз, який задає обчислення:

a) цілого числа від **0** до **9** за значенням символьної змінної **ch** від **'0'** до **'9'**;

b) символа від **'0'** до **'9'** за цілим значенням змінної **dg** від **0** до **9**.

9. У шістнадцятковій системі числення літерами **'A'**, **'B'**, …, **'F'** позначаються числа, десятковий запис яких **10**, **11**, …, **15**. Написати вираз, що задає обчислення цілого числа від **0** до **15** за значенням символьної змінної **ch**, яким може бути цифра від **'0'** до **'9'** або буква від **'A'** до **'F'**.

10. Намалюйте чотири кола, позначених іменами типів **boolean**, **integer**, **real** та **char**. Проведіть стрілки між колами — стрілка веде від кола *А* до кола *Б*, якщо існують операції з операндами типу *А* й значеннями типу *Б*, наприклад, від кола **integer** до кола **boolean** (порівняння). Позначте стрілки знаками відповідних операцій.

11. Нехай **a** та **b** — імена цілих змінних. Написати арифметичний вираз, значенням якого є:

a) більше з двох значень **a** і **b**;

b) значення останньої цифри в десятковому записі **a**, наприклад, при **a** = 789 це 9;

c) сума значень десяткових цифр двозначного **a** при **a** = 83 це 11;

d) сума значень десяткових цифр тризначного **a** при **a** = 123 це 6.

12. Нехай **a**, **b** — імена цілих змінних із додатними значеннями. Написати булів вираз, значенням якого є **true** тоді й тільки тоді, коли:

a) ціле **a** ділиться на ціле **b** без остачі;

b) цілі значення **a** і **b** мають однакову парність.

13. Нехай **k** — ім'я цілої змінної з додатним значенням, яке виражає кількість секунд від початку доби. Написати арифметичний вираз, значенням якого є:

a) кількість повних годин, які пройшли до цього моменту;

b) кількість повних хвилин, які пройшли до цього моменту.

14. Нехай n — ім'я цілої змінної. Написати вираз, який задає обчислення значення $(-1)^n$.

15. Написати вираз, який задає обчислення відстані між двома точками площини.

16. За допомогою стандартних математичних функцій написати вираз, який за дійсними змінними з іменами **a** та **b** задає обчислення значення: a^b, $\log_b a$, e, $\pi/4$.

17. Написати програму, яка задає читання значень трьох однотипних змінних **a**, **b**, **c**, "циклічний" обмін їх значень ($a \to b, b \to c, c \to a$) та друкування одержаних значень.

18. Нехай **z** — числова змінна. Використовуючи лише операції множення та оператори присвоювання змінним із будь-якими іменами, написати програму читання значення **z** та обчислення:

а) z^{10}; б) z^{12};

в) z^{15}; г) z^{31};

г) z^{48}; д) z^{62}.

Множень повинно бути якомога менше. Наприклад, обчислення z^6 можна задати так: **z2:=z*z; z4:=z2*z2; z6:=z4*z2**. Тут лише три множення замість п'яти у рівності **z6:=z*z*z*z*z*z**.

ðîçä³ë 4

ÐÎÇÃÀËÓÆÅÍÍß, ÂÈÁ²Ð ÂÀÐ²ÀÍÒ²Â ÒÀ ÑÊËÀÄÅÍ² ÎÏÅÐÀÒÎÐÈ

1. УМОВНІ ВИРАЗИ

Вирази з булевими значеннями називаються умовними, або ***умовами***. Вони відіграють особливу роль у програмуванні, оскільки є основою для прийняття рішень з вибору подальшого шляху обчислень.

Наприклад, нам знайома умова того, що квадратне рівняння $ax^2 + bx + c = 0$, задане коефіцієнтами **a**, **b**, **c** (значеннями числових змінних), при $a \neq 0$ буде мати два корені: **b*b-4*a*c>0**. Залежно від її значення **true** або **false** можна, наприклад, надрукувати ці два корені або з'ясувати, чи має рівняння один корінь чи не має жодного. Для цього з'ясування може знадобитися ознака **b*b-4*a*c=0**.

Часто умову використовують як ***ознаку*** деякої властивості, істинну, якщо властивість має місце, й хибну, якщо це не так. Наприклад, при $a \leq 0$ умови **b*b-4*a*c>0** та **b*b-4*a*c=0** є ознаками того, квадратне рівняння має відповідно два та один корінь.

Найпростіші умови, головним чином, записують як рівності або нерівності.

Приклади. Умову того, що значення цілої змінної **x** є парним, можна записати по-різному.

1. Парне число при діленні на 2 дає остачу 0: **x mod 2 = 0**.

2. Нагадаємо: вбудована функція **odd** повертає **true** за непарного аргументу. Тоді умова парності може виглядати так: **not odd(x)**.

Розглянемо умову того, що значення цілої змінної **x** є квадратом цілого числа, тобто $\sqrt{x}$ — ціле число. У цій ситуації ціла частина кореня при

піднесення до квадрату дорівнює x, в іншій — не дорівнює. Скористаємося тим, що функція **trunc** за дійсним значенням породжує ціле, а **sqr** за цілим аргументом обчислює ціле значення. Звідси маємо такий вираз: **sqr(trunc(sqrt(x))) = x**. ▣

Складніші умови формулюють як системи або сукупності, складені з простіших умов. ***Системи умов*** записують мовою Паскаль за допомогою операції **and**.

Розглянемо ознаку того, що значення числової змінної **x** належить проміжку $[a; b]$. Математичні вирази $a \le x \le b$ або $\begin{cases} a \le x \\ x \le b \end{cases}$ є ***системами***, і їм відповідає такий вираз (дужки в ньому обов'язкові!).

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

Математичний вираз $a = b = c$, що виражає рівність трьох значень, теж є системою, й ознака рівності значень змінних **a**, **b**, **c** має аналогічний вигляд.

```
(a = b) and (b = c)
```

Зауважимо, що записувати як ознаку вираз **(a=b) and (b=c) and (a=c)** є зайвим, оскільки за будь-яких значень змінних указані два вирази матимуть однакові булеві значення.

Припустимо, що значення числових змінних **a**, **b**, **c** представляють довжини відрізків. Запишемо умову того, що з цих відрізків можна утворити трикутник. Математика вимагає такого:

$a > 0, b > 0, c > 0, a+b > c, a+c > b, b+c > a$.

Умова, записана мовою Паскаль, може виглядати так.

```
(a>0) and (b>0) and (c>0) and (a+b>c) and
(a+c>b) and (b+c>a).▣
```

Сукупності умов записують мовою Паскаль за допомогою операції **or**.

Приклади. Припустимо, що значення числових змінних **a**, **b**, **c** представляють довжини відрізків, з яких можна утворити трикутник. Умова того, що цей трикутник рівнобедрений, мовою математики записується як сукупність:

$$\begin{cases} a = b \\ a = c \\ b = c \end{cases}, \text{ тобто } a = b \textbf{ або } b = c \textbf{ або } a = c.$$

Їй відповідає такий вираз. **(a = b) or (a = c) or (b = c)**

За тих самих припущень запишемо ознаку прямокутності трикутника. Гіпотенузою може бути будь-яка сторона, тому ознака виглядає так.

```
(a*a+b*b=c*c) or (a*a+c*c=b*b) or (b*b+c*c=a*a)
```

2. ОПЕРАТОРИ РОЗГАЛУЖЕННЯ

Пригадаймо алгоритм обчислення дійсних коренів рівняння $ax^2 + bx + c = 0$ за його коефіцієнтами **a**, **b**, **c** ($a \neq 0$). Залежно від конкретних значень коефіцієнтів, виконання алгоритму може відбуватися одним із трьох шляхів обчислення. Шлях обирається в результаті визначення, чи справджується умова $d > 0$; якщо це не так, то чи дійсна умова $d = 0$.

Розглянемо засоби, за допомогою яких у програмі вказують вибір шляху обчислень залежно від тих або інших умов, а потім повернемося до розв'язання квадратного рівняння.

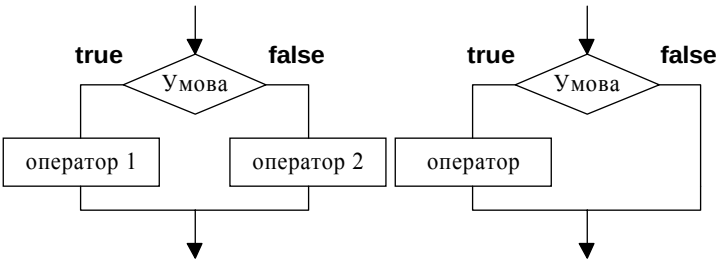
Вибір одного з двох можливих шляхів обчислення задають за допомогою ***операторів розгалуження*** (***умовних операторів***), що мають такий вигляд:

```
if умова then оператор1 else оператор2;
```

Нагадаємо: умова — це вираз типу **boolean**. Дослівний переклад цього запису виглядає так:

```
якщо умова то оператор1 інакше оператор2
```

При його виконанні спочатку обчислюється значення умови. Якщо отримано значення **true**, то виконується оператор, записаний після слова **then**, і на цьому виконання закінчується. Якщо отримано значення **false**, виконується оператор, записаний після **else**. Обидва оператори довільні (присвоювання, розгалуження або інші, представлені нижче). Наведений оператор називається оператором розгалуження ***у повній формі***. Йому відповідає блок-схема на рис. 1, *а*.



а) *б*)

Рис. 1. Блок-схеми двох форм оператора розгалуження

Приклад 1. Розглянемо такі оператори:

```
readln(x);
if x>=0 then z:=1 else z:=-1;
```

Якщо прочитано невід'ємне значення **x**, то обчислення виразу **x>=0** (перевірка умови) дає значення **true**, і змінна **z** отримує значення **1**. Якщо ж прочитане значення є від'ємним, то вираз **x>=0** має значення **false**, і **z** отримує значення **-1**.

Щоб програму було легше сприймати, оператор розгалуження часто записують так:

```
if умова then
  оператор1
else оператор2;
```

При розв'язанні багатьох задач оператори розгалуження ***вкладаються***, тому рекомендуємо записувати їх «східцями», наприклад, ось так:

```
if умова then
  if умова then
    оператор
  else
    оператор
else ...
```

Інколи виникають довгі ланцюги розгалужень, у яких після слова **else** записується наступний оператор розгалуження зі словом **if** на початку. У цих ситуаціях краще записувати оператори у такому вигляді.

```
if умова
  then оператор
else if умова
  then оператор
else if умова ...▣
```

Приклад 2. Напишемо програму обчислення коренів. Нам потрібні дійсні змінні для коефіцієнтів **a**, **b**, **c** та дискримінанта **d**. Вирази для коренів тільки обчислюються й друкуються, тому оголошувати змінні для них не обов'язково.

Читання коефіцієнтів задається оператором **readln(a,b,c)**, а обчислення дискримінанта — оператором **d:=b*b-4*a*c**. Потім алгоритм розгалужується залежно від значення **d**. Якщо **d>0**, то друкуються два корені, інакше перевіряється умова **d=0**. Якщо вона істинна, треба друкувати значення **-b/2a**, інакше —повідомлення про відсутність коренів.

Отже, програма буде такою:

```
program roots;
  var a, b, c, d : real;
begin
  writeln('Корені квадратного рівняння');
```

```
  writeln('Уведіть дійсні числа a, b, c (a<>0): ')
  readln(a,b,c);
  d:=b*b-4*a*c;
  if d > 0 then
    writeln((-b-sqrt(d))/(2*a ),' ',
            (-b+sqrt(d))/(2*a))
  else
    if d = 0 then
      writeln(-b/(2*a))
    else writeln('Дійсних коренів немає')
end.
```

Якщо при виконанні цієї програми увести значення змінних **a**, **b**, **c** (наприклад, відповідно, **1**, **3** і **2**), то умова **d>0** буде істинною, тому обчислюються й друкуються корені **-2** та **-1**. Якщо увести значення **1**, **2** і **3**, то умова **d>0** хибна і обчислюється умова **d=0**. Її значенням є **false**, і друкується повідомлення **Дійсних коренів немає**. Якщо ж задати значення **1**, **2** і **1**, то умова **d>0** буде хибною, умова **d=0** істинною; друкується значення **-b/(2a)**, тобто **-1**. ▣

Приклад 3. Припустимо, за віком особи чоловічої статі визначається її вікова категорія. Якщо вік менше двох років, це немовля. Від двох до п'яти — дитина, від 6 до 14 — хлопець, від 15 до 22 — юнак, від 23 до 60 — чоловік, більше — дід. Нехай вік є значенням змінної **age**.

Тоді вікова категорія друкується при виконанні такого оператора розгалуження:

```
if age < 2
  then writeln('немовля')
else if age <= 5
  then writeln('дитина')
else if age <= 14
  then writeln('хлопець')
else if age <= 22
  then writeln('юнак')
else if age <= 60
  then writeln('чоловік')
else writeln('дід');▣
```

Оператор розгалуження має ще одну, ***скорочену***, форму:

```
if умова then оператор;
```

Якщо обчислення умови дає значення **false**, то на цьому його виконання закінчується. Йому відповідає блок-схема на рис. 1, *б*.

Наприклад, замість наведеного вище оператора

```
if x >= 0 then z:=1 else z:=-1;
```

можна записати такі:

```
z:=-1; if x >= 0 then z:=1;
```

Якщо значення **x** від'ємне, **z** залишиться зі значенням **-1**.

Приклад. Виведення коренів квадратного рівняння можна задати за допомогою скороченої форми розгалуження.

```
if d>0 then
   writeln((-b-sqrt(d))/(2*a),' ',
           (-b+sqrt(d))/(2*a));
if d=0 then
   writeln(-b/(2*a));
if d<0 then
   writeln('Дійсних коренів немає');
```

Зазначимо недолік цього методу: якщо умова **d > 0** істинна, то в цьому прикладі після друкування все одно обчислюються умови **d = 0** і **d < 0**, а це збільшує час виконання програми.

Друкування коренів ***не*** можна описати в такий спосіб:

```
if d>0 then
   writeln((-b-sqrt(d))/(2*a),' ',
           (-b+sqrt(d))/(2*a));
if d=0 then
   writeln(-b/(2*a));
else writeln('Дійсних коренів немає');
```

Якщо **d=0** або **d<0**, то все буде гаразд, але якщо **d>0**, то друкуються два корені, потім обчислюється умова **d=0**, яка є хибною, і друкується текст **Дійсних коренів немає**, а це, вочевидь, неправильно.

3. СКЛАДЕНИЙ ОПЕРАТОР

Щоб написати кілька операторів там, де за правилами мови має бути один, наприклад, як гілку в умовному операторі, використовують складений оператор.

Складений оператор — це послідовність операторів у «операторних дужках» **begin-end**. Він має такий загальний вигляд:

```
begin
   оператор;
   ...
   оператор
end;
```

Виконання складеного оператора полягає у послідовному виконанні операторів, записаних у ньому.

Тіло будь-якої програми також є складеним оператором.

Приклад. Дано трикутник зі сторонами **a**, **b**, **c**. Визначити, який це трикутник: гострокутний, тупокутний чи прямокутний.

Для розв'язання цієї задачі слід урахувати таке:

1) довжини відрізків мають бути додатними; з відрізків заданої довжини можна утворити трикутник, тільки якщо сума довжин будь-яких двох з них більше довжини третього;

2) якщо трикутник можна побудувати, він буде прямокутним тоді й тільки тоді, коли (за теоремою Піфагора) сума квадратів двох сторін дорівнює квадрату третьої, причому ця рівність може виконуватися ***лише для однієї*** пари сторін;

3) у гострокутному трикутнику сума квадратів ***кожних*** двох сторін більше квадрата третьої, а в тупокутного є ***одна*** сторона, квадрат якої більше суми квадратів двох інших.

Задамо перевірку ознак типів трикутника за допомогою скорочених умовних операторів. Доведеться вкласти ***послідовність*** цих операторів у **else**-гілку, відповідну можливості побудови трикутника, а для цього необхідний складений оператор.

Отже, програма матиме такий вигляд:

```
Var a,b,c:real;
Begin
  write('Введіть довжини сторін: ');
  readln(a,b,c);
  if (a<=0) or (b<=0) or (c<=0) then
    writeln('Помилка вхідних даних.')
  else if (a+b<c) or (a+c<b) or (c+b<a) then
    writeln('З відрізків такої довжини ',
            'утворити трикутник неможливо.')
  else begin { утворити трикутник можна }
    if (sqr(a)+sqr(b)=sqr(c)) or
       (sqr(a)+sqr(c)=sqr(b)) or
       (sqr(c)+sqr(b)=sqr(a)) then
        writeln('Трикутник прямокутний');
    if (sqr(a)+sqr(b)>sqr(c)) and
       (sqr(a)+sqr(c)>sqr(b)) and
       (sqr(c)+sqr(b)>sqr(a)) then
        writeln('Трикутник гострокутний');
```

```
    if (sqr(a)+sqr(b)<sqr(c)) or
       (sqr(a)+sqr(c)<sqr(b)) or
       (sqr(c)+sqr(b)<sqr(a)) then
        writeln('Трикутник тупокутний');
    end
End.
```

4. ОПЕРАТОР ВИБОРУ ВАРІАНТІВ

Приклад 1. Продовжимо приклад про вікову категорію. Наведене розв'язання є правильним, але достатньо громіздким. Розв'язання буде набагато наочнішим і коротшим, якщо використати ***оператор вибору варіантів***, або **case**-оператор (від англ. case — випадок):

```
case age of
  0, 1  : writeln('немовля');
  2..5  : writeln('дитина');
  6..14 : writeln('хлопець');
  15..22: writeln('юнак');
  23..60: writeln('чоловік');
  else writeln('дід')
end;▣
```

Після слова **case** записують вираз довільного перелічуваного типу, який називається ***селектором*** варіантів (тут це ім'я **age**). Після слова **of** записують оператори-варіанти, ***відмічені*** деякими з можливих значень селектора. Ці мітки відокремлюють двокрапками. Варіант може відмічатися списком з кількох констант або діапазонів відповідного типу, записаних через кому.

У наведеній тут ***повній формі*** оператора після слова **else** записано альтернативний варіант. У ***скороченій формі*** його разом зі словом **else** немає.

При виконанні оператора вибору спочатку обчислюється значення селектора. Потім воно послідовно порівнюється з мітками варіантів. Тільки-но значення селектора збігається з міткою, виконується відповідний оператор, і все закінчується. Якщо значення селектора немає серед міток варіантів, то виконується альтернативний варіант, якщо він є. А якщо немає, то виконання оператора вибору закінчується.

Множини значень у списках можуть перетинатися — буде виконано варіант, у списку якого значення селектора знайдено вперше.

Приклад 2. Розглянемо таку задачу: увести з клавіатури число, знак операції (**+**, **-**, ***** або **/**), ще одне число і надрукувати результат застосу-

вання операції до цих чисел. Наприклад, після читання **2**, **+**, **3** друкується **5**, а після **2**, **/**, **3** — приблизно **0.667**.

Для спрощення припустимо, що користувач програми задає числа й знак в окремих рядках і що він правильно вводить числа, але може набрати недопустимий знак операції.

Для чисел оголосимо дійсні змінні **op1** та **op2**, а для знака операції — символьну **sign**. Ознаку відсутності помилки користувача збережемо в булевій змінній **ok**.

Спочатку прочитаємо операнди та знак. Потім за значенням **sign** оберемо одну з операцій: додавання, віднімання, множення або ділення. Результат запам'ятаємо на місці першого операнда. Якщо знак операції недопустимий або користувач хоче ділити на 0, повідомимо про це та присвоїмо **false** змінній **ok**. Якщо помилок не було, то після застосування операції надрукуємо значення першого операнда; якщо були, то нічого не надрукуємо.

Отже, гілка обчислень обирається за значенням змінної **sign** — вона й буде вказана як селектор варіантів.

Усі описані дії можна задати за допомогою **case**-оператора в такий спосіб:

```
writeln('Задайте дійсне число'); readln(op1);
writeln('Задайте символ операції (+, -, *, /)');
readln(sign);
writeln('Задайте дійсне число');
readln(op2);
ok := true;            {поки все гаразд}
case sign of
  '+': op1 := op1+op2;
  '-': op1 := op1-op2;
  '*': op1 := op1*op2;
  '/': if op2 <> 0
       then op1 := op1/op2
       else begin { спроба ділення на 0}
             writeln('Ділення на 0'); ok:=false
            end
  else begin { це "else" відповідає "case" }
        writeln('Недопустимий знак операції');
        ok:=false
       end
end; {це "end" відповідає "case"}
if ok then writeln('Результат: ', op1)
```

Вправа. Напишіть усю програму. Щоб перевірити її, треба задати не менше чотирьох наборів вхідних даних (по одному для кожного знака операції) і ще два набори, пов'язані з помилками (ділення на 0 та недопустимий знак операції).▣

У **case**-операторі перед словом **else** ***можна*** (але не обов'язково) писати «**;**» — на відміну від повної форми умовного оператора, де «**;**» перед **else** є синтаксичною помилкою.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. *Що таке умова? Які існують види умов і як вони формуються?*
2. *Який тип мають умови в умовних операторах?*
3. *Синтаксис і семантика команди розгалуження.*
4. *Що таке складений оператор?*
5. *Як працює оператор вибору варіантів?*
6. *Чи може селектор варіантів мати дійсний тип?*
7. *Чи може селектор варіантів бути іменем змінної або виразом перелічуваного типу?*
8. *Чи можна в якості міток варіантів у* **case***-операторі використовувати імена змінних або вирази перелічуваних типів?*

ЗАДАЧІ

1. Написати вираз, який задає перевірку, чи є значення символьної змінної **ch**:

a) цифрою від **'0'** до **'9'**;

b) малою латинською буквою;

c) латинською буквою (великою чи малою);

d) цифрою або латинською буквою.

2. Нехай a, b, c, … — імена цілих змінних із додатними значеннями. Написати умову того, що:

a) a, b, c задають сторони прямокутного трикутника;

b) a, b, c задають сторони гострокутного трикутника;

c) a, b, c задають сторони рівнобедреного трикутника;

d) a, b, c задають сторони різнобічного трикутника;

e) a, b, c, d задають сторони паралелограма;

f) $a1$, $b1$, $c1$ і $a2$, $b2$, $c2$ задають сторони двох рівних трикутників;

g) цеглину $a \times b \times c$ можна просунути у прямокутне вікно $x \times y$ так, щоб її грані були паралельні сторонам вікна.

3. Нехай a, b — імена цілих змінних із додатними значеннями. Написати умову того, що:

a) значення a ділиться на значення b без остачі;

b) значення a і b обидва парні або обидва непарні.

4. Підлога в кімнаті складається з квадратних клітин і має розміри $n \times m$. На двох клітинах поставлено стовпи. Написати умову того, що підлогу можна покрити дощечками розмірами 2×1.

5. Поле шахівниці визначається парою натуральних чисел від 1 до 8: перше число — номер вертикалі, друге — горизонталі. Нехай k, l, m, n — імена цілих змінних зі значеннями від 1 до 8. Написати умову того, що:

a) з поля (k, l) одним ходом тури можна потрапити на поле (m, n);

b) поля (k, l) і (m, n) є полями одного кольору;

c) з поля (k, l) одним ходом слона можна потрапити на поле (m, n);

d) ферзь, розташований на полі (k, l), загрожує полю (m, n);

e) кінь, розташований на полі (k, l), загрожує полю (m, n).

6. Використовуючи побітові булеві операції та порівняння, запишіть умову того, що беззнакове ціле **x** ділиться на 4 без остачі.

7. Змінні a, b, c, d мають дійсні значення. Написати умову того, що відрізки $[a; b]$ і $[c; d]$ прямої Ox мають хоча б одну спільну точку (гарантовано, що $a \le b$ та $c \le d$).

8. Написати умову того, що значення дійсних змінних з іменами a, b, c задають рівняння $ax + by + c = 0$:

a) прямої на площині;

b) прямої, яка проходить через I, II та III чверті площини;

c) прямої, яка проходить через I, II та IV чверті площини;

9. Написати умову того, що дві прямі на площині, які задано цілими коефіцієнтами двох рівнянь вигляду $ax + by + c = 0$:

a) паралельні й не збігаються; b) паралельні (можливо, збігаються);

c) збігаються; d) перетинаються; e) перпендикулярні.

10. Імена **x**, **y** — це імена змінних цілого типу. Імітувати виконання таких операторів, якщо при уведенні **x** отримує значення:

a) 1; b) 2; c) 3; d) 4.

```
readln(x);
    if x = 1
    then y:=16
    else if x = 2
        then y:=256
        else if x = 3
             then y:=4096
             else y:=10000;
writeln(y)
```

Переписати розгалуження за допомогою оператора **case**.

11. За чотирицифровим цілим числом визначити:

a) чи є сума його цифр парною;

b) чи є воно «щасливим» (сума перших двох цифр дорівнює сумі двох останніх, наприклад, 3407);

c) чи є воно паліндромом (читається зліва направо та справа наліво однаково, наприклад, 2992);

d) чи збігаються його перша й друга половини, наприклад, 4545;

e) чи утворюють його цифри неспадну послідовність, наприклад, 4588;

f) чи має воно хоча б дві однакові цифри, наприклад, 3953;

g) чи є якісь дві його цифри парними, а дві — ні, наприклад, 2376.

12. Написати програму дослідження, тобто обчислення кількості дійсних коренів рівняння $ax^2 + bx + c = 0$ за його коефіцієнтами (можливо, $a = 0$).

13. Дано коефіцієнти a, b, c рівняння $ax^2 + bx + c = 0$. Визначити, чи є хоча б один його корінь парним числом.

14. Написати програму, за якою вводяться три цілих числа, перевіряється, чи задають вони довжини сторін трикутника, і, якщо це так, визначається його вигляд: рівнобічний, рівнобедрений і не рівнобічний, різнобічний; гострокутний, прямокутний, тупокутний.

15. Дано координати двох відрізків на координатній площині. Визначити:

a) чи належать вони одній прямій і якщо так, то чи мають вони хоча б одну спільну точку;

b) чи збігаються відрізки хоча б одним із своїх кінців;

c) чи перетинаються ці відрізки.

16. Дано координати кінців відрізка на координатній площині та координати ще двох точок. Визначити, чи лежать ці точки:

a) на прямій, яка містить відрізок;

b) по один бік прямої або по різні боки прямої, яка містить відрізок.

17. Дано три цілих числа, які задають час, що показує стрілковий годинник (наприклад, **12 30 45** означає 12 год, 30 хв, 45 с). Визначити:

a) чи збігаються годинна та хвилинна стрілки у цей час;

b) чи знаходяться годинна та хвилинна стрілки під кутом 90 градусів;

c) через який найменший час збігатимуться годинна та хвилинна стрілки.

18. У банці знаходяться білі та чорні зернини. З банки виймають навмання дві зернини. Якщо вони мають той самий колір, то їх викидають, а до банки кладуть чорну зернину (чорних зернин достатньо). Якщо ж зернини мають різні кольори, то чорну викидають, а білу повертають до бан-

ки. Ці дії повторюють доти, поки не залишиться одна зернина. Напишіть програму, яка за відомою кількістю чорних та білих зернин визначає колір останньої зернини.

19. Найближча крамниця працює з 7:00 до 19:00 з перервою на обід з 13:00 до 15:00. Гастроном, що знаходиться в 20 хв ходьби, працює з 8:00 до 20:00 та має перерву з 14:00 до 16:00. На відстані, подолати яку можна тільки за 45 хв, знаходиться супермаркет, який працює з 8:00 до 23:00 без перерви. За заданим часом визначити, що краще:

a)відвідати найближчу крамницю;

b)дійти до гастронома (з урахуванням часу на дорогу);

c)рушати в супермаркет (теж з урахуванням часу на дорогу);

d)залишитися вдома, оскільки всі магазини зачинено. Час уводиться так: години — ціла частина числа, а хвилини — дробова, тобто 15:30 означає 15 год 30 хв.

20. Дано ціле число, що визначає вік людини. Дописати до нього слово «рік», «роки», «років» відповідно до правил української граматики. Наприклад, 21 рік, 34 роки, 14 років.

21. Розробити програму з використанням команди вибору, яка за введеною датою народження людини визначає, до якого знаку Зодіака вона належить.

20.01 – 18.02	Водолій
19.02 – 20.03	Риби
21.03 – 19.04	Овен
20.04 – 20.05	Телець
21.05 – 21.06	Близнюки
22.06 – 22.07	Рак
23.07 – 22.08	Лев
23.08 – 22.09	Діва
23.09 – 22.10	Терези
23.10 – 22.11	Скорпіон
23.11 – 21.12	Стрілець
22.12 – 19.01	Козеріг

ðîçä³ë 5

ÖÈÊË²× Í² ÀËÃÎÐÈÒÌ È

1. ОПЕРАТОР ЦИКЛУ З ПАРАМЕТРОМ

Почнемо з прикладу. Функція факторіала $n!$ невід'ємного цілого числа n визначається формулою $n! = (n-1)! \cdot n$ при $n > 0$, а $0! = 1$. Звідси $n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$. Щоб обчислити цей добуток при $n \geq 1$, треба спочатку присвоїти добутку значення **1**, а далі перебрати всі цілі множники від **1** до **n** та домножити добуток на них, щоразу зберігаючи результат у добутку. Уточнимо ці міркування.

Подамо добуток у змінній **fact**. Отже, якщо $n > 0$, треба перебрати множники **k = 1, 2, …, n** і для кожного виконати **fact:=fact*k**. Іншими словами, треба виконати такі дії:

```
fact:=1;
для k = 1, 2, …, n виконати fact:=fact*k
```

Тепер запишемо це мовою Паскаль:

```
fact:=1;
for k:=1 to n do fact:=fact*k;
```

Отже, множення **fact** на **k** відбувається за вказаних значень від **1** до **n** (*for* — це англійське «для», *to* — «до», *do* — «виконати»).

Загальний вигляд оператора циклу з параметром такий:

```
for ім'я := вираз1 to вираз2 do оператор;
```

Ім'я позначає ***параметр циклу*** — змінну перелічуваного типу, сумісного за присвоюванням з типами виразів. Частина оператора від слова **for** до слова **do** називається ***заголовком*** циклу, а **оператор** — ***тілом***.

У прикладі параметром була змінна **k**, виразами — **1** та **n**, тілом — **fact:=fact*k**.

Оператор циклу з параметром ще називають ***for-оператором***. Опишемо дещо ***спрощено*** його виконання (в дійсності воно складніше й має деякі «підводні камені»). Спочатку обчислюються й порівнюються вирази в заголовку. Якщо значення першого виразу **A** більше ніж другого **B**, на цьому виконання оператора циклу закінчується. Інакше параметр циклу послідовно отримує значення **A**, **A+1**, …, **B**, і при кожному з них виконується тіло циклу.

Оператор циклу з параметром має ще одну форму.

for ім'я := вираз1 downto вираз2 do оператор;

Усі позначення мають той самий зміст, що й у першій формі. Слово **downto** («униз-до») замість слова **to** означає, що при **A** ≥ **B** тіло циклу виконується зі значеннями параметра циклу, які послідовно ***зменшуються***: **A**, **A-1**, …, **B**. Відповідно, при **A** < **B** тіло циклу не виконується жодного разу.

Оператори циклу з переліком застосовують, коли ***кількість виконань*** тіла циклу ***відома до початку*** його виконання.

Мова Turbo Pascal не забороняє змінювати параметр циклу в тілі циклу, але наслідки цього можуть бути непередбачуваними, тому варто вважати, що ***змінювати параметр заборонено***.

Граничні значення, задані виразами в заголовку оператора, ***обчислюються один раз*** і запам'ятовуються. Якщо ці вирази містять змінні, що отримують нові значення при виконанні циклу, ці зміни ***не*** впливають на граничні значення.

Варто вважати, що ***за межами*** оператора циклу значення параметра невизначено і там його ***не використовувати***.

2. ПРИКЛАДИ ОРГАНІЗАЦІЇ ЦИКЛІВ З ПАРАМЕТРОМ

Приклад 1. За натуральним значенням змінної **n** обчислити значення виразу $\sqrt{2+\sqrt{2+\ldots+\sqrt{2}}}$ (*n* знаків кореня).

Результат можна обчислити як *n*-й елемент послідовності $\sqrt{2}$, $\sqrt{2+\sqrt{2}}$, $\sqrt{2+\sqrt{2+\sqrt{2}}}$, … . На кожному кроці до попереднього результату додається 2 й береться квадратний корінь із цієї суми. Зазначимо, що $\sqrt{2}$ утво-

рюється як корінь з суми 2 та попереднього результату 0, тому перше значення $\sqrt{2}$ теж можна обчислити в циклі.

```
rez:=0;
for i:=1 to n do
   rez:=sqrt(rez+2);
```

Приклад 2. Скласти програму піднесення до квадрата натурального числа n, використовуючи таку закономірність:

$1^2 = 1,\ 2^2 = 1+3,\ 3^2 = 1+3+5,\ 4^2 = 1+3+5+7, \ldots,$

$n^2 = 1+3+5+7+\ldots+(2n-1).$

У цій закономірності бачимо: квадрат числа n є сумою n непарних натуральних чисел від 1 до $2n-1$. Отже, накопичимо числа вигляду **2i–1** при **i = 1,2,…,n** в сумі **s**, яка спочатку дорівнює **0**.

```
s:=0;
for i:=1 to n do
   s:=s+2*i-1;
```

Приклад 3. Знайти суму дільників заданого натурального числа n.

«Лобове» розв'язання задачі є очевидним. Перебрати всі числа від 1 до n і, якщо n ділиться на поточне число, додати це число до накопичуваної суми.

```
S:=0;
for i:=1 to n do
   if n mod i = 0 then S:=S+i;
```

Кількість повторень циклу можна зменшити приблизно наполовину, адже кожне натуральне число ділиться на 1 і на себе, а наступний за величиною після n множник не більше половини n. Звідси візьмемо $1+n$ за початкове значення суми, а далі переберемо числа від 2 лише до половини n.

```
S:=1+n;
for i:=2 to n div 2 do
 if n mod i = 0 then S:=S+i;
```

Проте кількість повторень можна зменшити ще, врахувавши таку особливість: якщо добуток двох дільників числа n дорівнює n, то хоча б один з них не більше $\sqrt{n}$. Це дозволяє, додавши до суми спочатку 1 і n, потім перебрати числа i від 2 до $[\sqrt{n}]$ і додати до суми кожен дільник **i** разом з **n div i**.

```
S:=1+n;
```

```
for i:=2 to round(sqrt(n)) do
   if n mod i = 0
   then S:=S+i+S div i;
```

Значення квадратного кореня округлюється функцією **round**, оскільки в заголовку циклу **for** можна використовувати вирази тільки перелічуваного типу.

Приклад 4. Формула ***бінома Ньютона*** $(a+b)^n = \sum_{k=0}^{n} C_n^k a^k b^{n-k}$ — одна з найбільш відомих у математиці. Коефіцієнти $C_n^k = \frac{n!}{k!(n-k)!}$ називаються ***біномними***. Обчислимо біномний коефіцієнт за заданими n і k.

Можна написати три цикли для обчислення факторіалів (див. параграф 4.1), але це дуже нераціонально. По-перше, одні й ті самі значення, що виникають у процесі обчислення факторіалу, обчислюються тричі. По-друге, функція $m!$ швидко зростає при збільшенні m, тому існує чимало значень n і k, при яких значення C_n^k подається в цілих типах, а $n!$ і $k!$ — ні.

Подивимося на формулу біномного коефіцієнта:

$$\frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdot\ldots\cdot(n-k+1)}{1\cdot 2\cdot\ldots\cdot k}.$$

У ній закладено обчислення послідовності значень $1, n/1, n\cdot(n–1)/(1\cdot 2), \ldots, n(n–1)\cdot\ldots\cdot(n–k+1)/(1\cdot 2\cdot\ldots\cdot k)$, в якій кожне наступне значення отримується з попереднього шляхом множення на дріб вигляду $(n–i+1)/i$. Усі члени цієї послідовності цілі, тому ці обчислення можна задати так:

```
d:=1;
for i:=1 to k do
   d:=d*(n-i+1) div i;
```

З урахуванням тотожності $C_n^k = C_n^{n-k}$ при k ? $n/2$ кількість повторень циклу можна зробити рівним k, інакше — $n – k$. Не забудемо також, що $C_n^k = 0$ при $k > n, k < 0$ або $n < 0$.

Оголосимо змінні n і k типу **integer**, C типу **longint**. Опишемо обчислення такими операторами (значення n і k вважаються вже заданими, обчислений коефіцієнт зберігається в змінній C).

```
if (k>n) or (k<0) or (n<0)
then C:=0
else begin
  if k > n div 2
  then k:=n-k; { C(n,k) = C(n,n-k) }
  C:=1;
  for i:=1 to k do
   C:=C*(n-i+1) div i;
end;
```

Для перевірки, чи правильно запрограмовано обчислення, треба задати три пари значень **n** і **k**, при яких результатом має бути 0, а також пари, при яких цикл з параметром виконується один і кілька разів. Особливої уваги потребують випадки, коли **k** = **n div 2** і **k** = **n div 2 + 1**, оскільки за таких значень **k** при фіксованому **n** біномні коефіцієнти максимальні. Варто також дослідити граничні пари значень **n** і **k**, за яких правильний результат подається в типі **longint**.

Приклад 5. Відомо, що коробка цукерок коштує *B* гривень, шоколадка — *C*, а пачка печива — *P* *(B > C > P*, усі числа натуральні). У покупця є *N* гривень. Чи можна купити на ці гроші *T* предметів, витративши гроші повністю?

Спочатку розглянемо простий і не надто раціональний спосіб. Організуємо повний перебір всіх варіантів, враховуючи, що коробок з цукерками може бути не більше *N* div *B*, шоколадок — не більше *N* div *C*, пачок печива — не більше *N* div *P*. Отже, утворимо три вкладених цикли й у внутрішньому вкажемо перевірку умови задачі.

```
for box:=0 to N div B do
 for chock:=0 to N div C do
  for pack:=0 to N div P do
    if (box*B+chock*C+pack*P = N) and
       (box+chock+pack = T) then
        writeln('Коробок: ',box,'; шоколадок: ',
                  chock,'; пачок: ',pack);
```

Проте кількість повторень тіла внутрішнього циклу повторень можна суттєво зменшити. По-перше, якщо придбано **box** коробок, то залишається сума грошей **N-box*B**, на яку можна придбати не більше **(N-box*B) div C** шоколадок. Між іншим, це гарантує, що **box*B+chock*C ≤ N**. По-друге, якщо придбано **box** коробок і **chock** шоколадок, то за умовою задачі кількість пачок ***має дорівнювати*** **T-**

(box+chock). Тоді замість перебору значень **pack** достатньо виконати присвоювання **pack:=T-(box+chock)**, зменшивши цим кількість вкладених циклів.

```
for box:=0 to N div B do
  for chock:=0 to (N-box*B) div C do
  begin
    pack:=T-box-chock;
    if (box*B+chock*C+pack*P = N) then
       writeln('Коробок: ',box, '; шоколадок: ',
              chock,'; пачок: ',pack);
  end;
```

Можна було б утворити зовнішній цикл по кількості **pack** пачок печива або по кількості **chock** шоколадок, а не по кількості **box** коробок з цукерками. Але тоді кількість виконань тіла була б більшою, оскільки $B > C > P$.

Наведений фрагмент можна удосконалити, якщо врахувати, що розв'язок задачі існує лише за умови $T*P \le N$.

Приклад 6. Послідовність чисел 1, 1, 2, 3, 5, 8, 13, … , у якій перші два числа дорівнюють 1, а кожне наступне, починаючи з третього, дорівнює сумі двох попередніх, називається ***послідовністю чисел Фібоначчі***. Обчислити за номером n ($n > 0$) n-е число Фібоначчі.

Якщо $n = 1$ або 2, результатом є 1. Якщо $n > 2$, будемо визначати числа одне за одним, поки не отримаємо числа із заданим номером. За першими двома обчислимо третє, за другим і третім — четверте, за третім і четвертим — п'яте, і так далі. Як бачимо, для визначення нового числа з усіх попередніх чисел потрібні тільки ***два останніх***.

Отже, у нас з'являються поняття: ***передостаннє число***, ***останнє*** та ***наступне***, а також ***номер*** числа, визначеного останнім. Представимо їх змінними, відповідно, **fa**, **fb**, **fc** та **i**. Розглянемо, як мають змінюватися їх значення:

	i	**fa**	**fb**	**fc**
На початку першого кроку	**2**	**1**	**1**	**?**
Обчислення суми	**2**	**1**	**1**	**2**
На початку другого кроку	**3**	**1**	**2**	**2**
Обчислення суми	**3**	**1**	**2**	**3**
На початку третього кроку	**4**	**2**	**3**	**3**
Обчислення суми	**4**	**2**	**3**	**5**
На початку четвертого кроку	**5**	**3**	**5**	**5**

Щоб реалізувати ці зміни, треба на кожному кроці виконувати **fc:=fa+fb**, після чого збільшувати номер **i** та готуватися до наступного кроку: **fa:=fb; fb:=fc**. Зауважимо, що перестановка цих двох присвоювань була б грубою помилкою.

Реалізуємо наведені міркування в такій програмі.

```
program fibonacci;{обчислити n-е число Фібоначчі}
var fa, fb, fc : longint;
   {два попередні й нове числа Фібоначчі}
    n, i : integer;
Begin
  writeln('Задайте номер числа Фібоначчі: ');
  readln(n);
  if n<=2 then writeln(1)
  else begin
    fa:=1; fb:=1; {пара останніх чисел}
    {цикл переходу до нової пари останніх чисел}
    for i:=3 to n do
    begin
        fc:=fa+fb ; {наступне останнє число}
        fa:=fb; {підготовка до наступного кроку }
         fb:=fc; {або «переприсвоювання»}
    end;
    writeln(fb)
  end { гілки else }
End;
```

Щоб перевірити цю програму, задайте номери 1, 2 й деякі інші так, щоб цикл виконувався один і декілька разів, наприклад, номери 3 й 8. Результатами мають бути числа, відповідно, 1, 1, 2 та 21. Цікаво також побачити, що 144 — це 12-е число Фібоначчі (1 і 144 — єдині числа Фібоначчі, рівні квадратам своїх номерів).

Вправа. Змініть програму так, щоб можна було визначити найбільше число Фібоначчі типу **integer** (**word**, **longint**) та його номер. ▣

Числа Фібоначчі мають величезне значення в усій математиці й носять ім'я видатного італійського математика Леонардо Пізано Фібоначчі (точніше, Боначчі-молодший). Він сформулював кілька цікавих задач; найвідомішою з них стала така «задача про розмноження кролів».

Першого січня ми маємо пару кролів. Ця пара може народити нову пару через два місяці, тобто у перший день березня, а потім на перший день кожного наступного місяця. Кожна народжена пара кролів виростає

за місяць і ще через місяць народжує нову пару кролів. Скільки пар кролів буде через 12 місяців, тобто через рік?

Неважко зрозуміти, що першого січня, лютого, березня тощо буде саме 1, 1, 2, 3, 5, 8, … пар кролів. Продовживши цей ряд, отримаємо, що першого грудня буде 144 пари кролів, а першого січня наступного року — 233.

Цікаво, що багато інших задач, зовні не схожих на задачу про кролів, мають такий самий розв'язок.

Наведемо приклади.

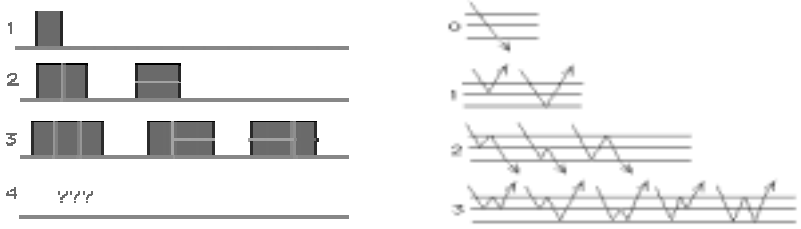
1. Треба пройти по сходах, які мають n східців. З чергового східця можна ступати або на наступний, або через один. Скільки існує варіантів пройти по сходах?

2. Визначити кількість послідовностей довжини n, складених з символів **0** і **1**, в яких жодні дві одиниці не стоять поруч.

3. ***Шлях бджоли***. Бджола починає свій рух у клітину 1 або 2 і прямує до клітини з номером n. Пересуватися бджола може лише по сусідніх клітинах від меншого номера до більшого. Скількома різними шляхами бджола може потрапити до клітини з номером n?

4. Цеглина має висоту 2 й ширину 1. Треба збудувати стіну висотою2 та шириною n. Визначити кількість способів укладки цеглини залежно від n (рис. 1, *а*).

5. Дві скляні пластини накладено одна на одну. Скільки існує способів проходження променя світла крізь пластини або відбиття від них, якщо промінь змінює свій напрямок n разів (рис. 1, *б*)?



а) цеглини в стіні б) проходження променя

Рис. 1. Кількості способів є числами Фібоначчі

3. ОПЕРАТОР ЦИКЛУ З ПЕРЕДУМОВОЮ

Продовжимо приклад з обчисленням факторіала. Після виконання тіла циклу зі значеннями параметра **1**, **2**, …, **n** змінна **fact** отримує послідовні значення 1!, 2!, …, *n*!. Щоб перейти від попереднього значення **fact** до наступного, треба збільшити значення **k** та домножити **fact** на нове значення **k**. Але це можна повторювати, поки **k** не досягло **n**. Отже, в циклі ще повинна бути перевірка умови **k** < **n**. Уточнимо опис цих дій, врахувавши, що починати треба зі значенням **k** = **0**.

```
k:=0; fact:=1;
Поки k < n виконувати:
  k:=k+1; fact:=fact*k
```

Мовою Паскаль ці дії описуються так.

```
{ ініціалізація множника та факторіала }
k:=0; fact:=1;
while k < n do
begin { «поки k < n виконувати» }
  k:=k+1;        { наступний множник }
  fact:=fact*k; { нове значення факторіала }
end;
{ k = n, fact = n! }
```

Оператор циклу з передумовою (**while**-оператор) має такий загальний вигляд:

```
while умова do оператор;
```

Тут **while** *умова* **do** — заголовок циклу, а ***оператор*** — тіло (слова **while** та **do** зарезервовані).

Оператор циклу виконується так. Спочатку обчислюється умова в заголовку. Якщо вона істинна, виконується тіло циклу й знов обчислюється умова. Якщо вона істинна, все повторюється. Виконання циклу закінчується, коли при обчисленні умови утворюється **false**. Отже, останній раз в циклі тільки обчислюється умова, а тіло циклу не виконується. Якщо при першому обчисленні умови вона виявляється хибною, тіло циклу не виконується жодного разу.

Оператору циклу з передумовою відповідає блок-схема, подана на рис. 2.

Умову в операторі циклу називають ***умовою продовження***, оскільки, якщо вона істинна, виконання оператора циклу продовжується. Виконання циклу починається саме з обчислення умови, тому її ще називають ***передумовою***.

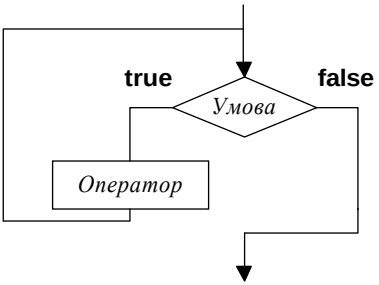


Рис. 2. Блок-схема оператора циклу з передумовою

Оператори циклу з передумовою застосовують, як правило, коли кількість повторень циклу заздалегідь ***невідома***.

Після **while**-оператора варто записати ***коментар*** із запереченням умови продовження. Це допомагає відстежити значення змінних після закінчення оператора циклу.

4. ПРИКЛАДИ ОРГАНІЗАЦІЇ ЦИКЛІВ З ПЕРЕДУМОВОЮ

Приклад 1. Обчислити суму цифр заданого натурального числа.

Нехай задане число є значенням змінної **n**. Щоб знайти суму його цифр, можна діяти так. Спочатку сума **sum** дорівнює **0**. Далі беремо молодшу цифру числа, додаємо її до суми та викреслюємо з числа. Повторюємо ці дії, доки в числі є цифри. Молодша цифра числа **n** є значенням виразу **n mod 10**, її «викреслення» з числа можна подати як **n := n div 10**. Наприклад, **123 mod 10** = **3**, **123 div 10** = **12**. Отже, маємо такі оператори:

```
sum := 0; { початкова сума цифр }
while n <> 0 do begin
  sum := sum + n mod 10;{додамо молодшу цифру}
  n := n div 10 {та «викреслимо» її}
end;
{ n=0, значення sum є сумою всіх цифр }
```

Зауважимо, що застосувати цикл **for** у цій ситуації неможливо, оскільки кількість цифр наперед не відома.

Приклад 2. Число називається ***автоморфним***, якщо його цифри збігаються з останніми цифрами його квадрата. Наприклад, $5^2 = 25$, $6^2 = 36$, $25^2 = 625$. Визначити, чи є натуральне число N автоморфним.

Спочатку обчислимо N^2, а потім будемо циклічно порівнювати останні цифри обох чисел і, якщо вони рівні, вилучати їх з обох чисел. Процес обірветься, коли від початкового числа залишиться **0** або останні цифри не будуть збігатися.

```
var N, sqr_N : longint;
Begin
  write('Введіть число ');
  readln(N);
  sqr_N:=sqr(N);
  while (N<>0) and (N mod 10=N_sqr mod 10) do
  begin
    N := N div 10;
    N_sqr := N_sqr div 10;
  end;
  if N=0
  then writeln('Число автоморфне.')
  else writeln('Число не автоморфне.');
End.
```

Приклад 3. Довжини сторін прямокутника є натуральними числами. Від прямокутника відрізають квадрати максимальної площі, поки від нього не залишиться квадрат. Знайти кількість утворених квадратів.

Нехай *a* та *b* — довжини сторін прямокутника. Очевидно, що сторона квадрата максимальної площі рівна меншій стороні прямокутника. Нехай це буде *a*. Відрізаючи квадрат, ми зменшуємо більшу сторону прямокутника (**b:=b-a**). Якщо $a > b$, то зменшується *a*: **a:=a-b**. Відрізання припинимо, коли прямокутник «виродиться» у квадрат. Кількість квадратів — це кількість відрізань плюс один, оскільки останнє відрізання утворює два квадрати. Отже, при кожному відрізанні будемо збільшувати ***лічильник відрізань*** **count**, почавши з **1**.

```
var a, b, count : longInt;
Begin
  write('Введіть сторони прямокутника: ');
  readLn(a,b);
  count:=1;
  while a <> b do begin
    if a > b then a := a-b
```

```
    else b := b-a;
    inc(count)
  end;
  writeLn('Кількість квадратів: ', count)
End.
```

Проімітуємо підрахунок квадратів при $a = 5$, $b = 3$.

Операція	a	b	count
count := 1	5	3	1
a := a-b; inc(count)	2	3	2
b := b-a; inc(count)	2	1	3
a := a-b; inc(count)	1	1	4

Якщо з цього алгоритму вилучити підрахунок кількості квадратів, отримаємо добре відомий ***алгоритм Евкліда***, за яким можна обчислити найбільший спільний дільник (НСД) двох натуральних чисел. НСД є остаточним значенням змінних **a** та **b**.

Наведений алгоритм працює ***дуже довго***, якщо одне з чисел велике, а друге мале. Для наочності запустіть програму й уведіть, наприклад, 2147483647 (найбільше число типу **longint**) та 1. Проте роботу можна суттєво прискорити. Якщо $a > b$, то в результаті послідовних віднімань b від a в a залишається ***остача*** від ділення a на b. Отже, замість віднімання обчислимо **a:=a mod b**, збільшивши **count** відразу на **a div b**. При цьому, якщо **a** кратне **b**, то **a mod b** стане рівним 0, а кількість **a div b** враховує як квадрати, що відрізаються, так і квадрат-залишок. Отже, змінимо умову продовження на **(a<>0) and (b<>0)** та почнемо з **count = 0**.

Оператори між уведенням та виведенням набудуть такого вигляду:

```
count:=0;
while (a<>0) and (b<>0) do begin
  if a > b
  then begin
        a:=a mod b; inc(count,a div b)
       end
  else begin
        b:=b mod a; inc(count,b div a)
       end
end;
```

Розглянувши уважно остачі від ділення на двох послідовних кроках, неважко переконатися, що більше з двох чисел за два кроки алгоритму ***зменшується більш ніж удвічі***. Завдяки цьому алгоритм працює швидко за будь-яких вхідних чисел. Цікаво, що якщо вхідними є два послідовних числа Фібоначчі, то всі остачі від ділення **a mod b** дорівнюють різницям $a - b$, тобто другий алгоритм працює не краще ніж перший. При цьому послідовними значеннями змінних **a** та **b** є числа Фібоначчі, а результатом — номер меншого з вхідних чисел у послідовності чисел Фібоначчі.

Якщо в другому алгоритмі вилучити обробку лічильника **count**, отримаємо ***алгоритм Евкліда в сучасній версії***. НСД вхідних чисел — це значення тієї зі змінних **a** та **b**, яке після закінчення циклу не дорівнює 0.

Приклад 4. Напишіть програму визначення останньої ненульової цифри в запису числа $N!$ (N — типу **integer**). Наприклад, у $5! = 120$ цією цифрою є 2, у $7! = 5040$ — 4.

Безпосередньо обчислити $N!$ не можна, оскільки факторіали швидко зростають. Наприклад, найпотужніший цілий тип **longint** «вміщує» лише $12! = 518918400$. Існує цікавіше розв'язання. Увага: старші цифри факторіалу не впливають на молодші цифри наступного факторіалу, а нулі в кінці факторіалу залишаються і у подальших факторіалів. Ці міркування дозволяють замість факторіалів зберігати лише декілька їх останніх ненульових цифр та відкидати молодші нулі.

```
var f, i, N : longInt;
Begin
  write('Введіть натуральне число: ');
  readln(N);
  f:=1;
  for i:=1 to N do
  begin
    f:=f*i;
    while f mod 10 = 0 do
       f:=f div 10;
    f:=f mod 10000;
  end;
  writeln('Остання ненульова цифра: ', f mod 10);
End.
```

Приклад 5. Розкласти натуральне число більше 1 на прості множники, тобто подати у вигляді добутку простих чисел.

Наприклад, $10 = 2 * 5$; $16 = 2 * 2 * 2 * 2$.

Будемо перебирати всі натуральні числа *k*, починаючи з 2, та, поки задане число *N* ділиться на *k*, виводити *k* й ділити *N* на нього. Процес продовжимо, поки *N* більше 1. Помітимо: після всіх можливих ділень на простий дільник він відсутній в *N*. Звідси, коли дійдемо до складеного *k*, *N* не поділиться на *k*, оскільки всі прості дільники числа *k* вже відсутні в *N*. Наприклад, 20 двічі ділиться на 2 й потім не ділиться на 4, а тільки на 5: 20 = 2*2*5. Отже, запропоноване розв'язання не потребує перевірки числа на простоту.

```
var N, k : longint;
Begin
    write('Введіть число: ');
    readln(N); write(N, ' = ');
    k:=1;
    while N > 1 do
    begin
     inc(k); { збільшуємо дільник на 1 }
     while N mod k = 0 do
     begin
     write(k); {поділилося - виводимо множник}
       N:=N div k;
       {якщо залишилося більше 1,то буде продовження}
        if N > 1 then write('*');
   end;
  end;
End.
```

Перевірте цю програму для чисел 2, 5, 20, 105, 1073741824 (це 2 в степені 20). Додайте обробку ситуації $N = 1$.

Спробуйте написати програму, яка організовує виведення результату у іншому форматі з використанням позначки степеня «^». Тоді розклади виглядатимуть так: 48=2^4*3, 200=2^3*5^2.

Приклад 6. Відома формула для обчислення числа π:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots.$$

Узявши суму *n* доданків (*n* — натуральне число), отримаємо ***наближення до*** π. Знайти *n*, при якому наближення відрізняється від π не більше ніж на 0.001.

Варіант 1. Скористаємося стандартною функцією **Pi**: значення, яке вона повертає, вважатимемо еталонним. Будемо обчислювати суму, додаючи або віднімаючи дроби по одному, поки різниця між сумою та еталоном не стане менше 0.001. Нумерацію доданків почнемо з 1. Перший доданок відразу додамо до суми. Далі в циклі дроби з парними номерами віднімаються, з непарними — додаються. Знаменники доданків щоразу збільшуються на 2.

```
program p4_1;
  const difference = 0.001; { різниця }
  var p4 : real; { наближена сума }
    k, n : integer; { знаменник та номер дробу }
Begin
  k:=1; n:=1; { перші знаменник та номер }
  p4:=1; { перший доданок 1 враховано в сумі }
  while abs(Pi-p4*4) > difference do begin
    inc(k,2);inc(n);{наступні знаменник і номер}
    if odd(n)
    then p4 := p4+1/k
    else p4 := p4-1/k;
  end;
  { abs(Pi-p4*4) <= difference }
  writeln(n);
End.
```

Варіант 2. Спробуємо обійтися без еталонного значення. Послідовні доданки в сумі за абсолютною величиною монотонно зменшуються та по черзі додаються й віднімаються. Звідси знак нескінченного «хвоста» суми у формулі збігається зі знаком першого доданка «хвоста». Тому число π обов'язково знаходиться між двома сусідніми значеннями суми, причому ближче до останнього з них. Заведемо дві змінні: **p41** має значеннями суми парного числа дробів, **p42** — непарного. На парному кроці за значенням **p41** обчислюється **p42**, на непарному — навпаки, за **p42** обчислюється **p41**. Якщо різниця між ними буде не більше 0.002, це гарантує, що різниця між останньою з них та π не більше 0.001.

```
program pi4_2;
  const difference = 0.001;
  var p41, p42 : real; k, n : longint;
begin
  p41:=1; p42:=2/3; n:=2; k:=3;
  while 4*abs(p41-p42) > 2*difference do
```

```
  begin
    inc(k,2); inc(n);
    if odd(n) then p41:=p42+1/k
    else p42:=p41-1/k;
  end;
  { abs(Pi-p4*4) <= 2*difference }
  if odd(n) then p42 := p41;
  writeln(n);
end.
```

5. ОПЕРАТОР ЦИКЛУ З ПІСЛЯУМОВОЮ

Оператор циклу з ***післяумовою***, або **repeat**-оператор, має такий загальний вигляд:

```
repeat
 послідовність операторів
until умова;
```

Слова **repeat** і **until** («повторювати» та «доти, поки не») є ключовими; вони відіграють роль операторних дужок, у які вкладено тіло циклу — послідовність операторів.

Оператор циклу з післяумовою виконується так. Спочатку виконуються оператори тіла циклу, потім обчислюється умова. Якщо вона істинна, цикл завершується, інакше повторюється тіло й знову обчислюється умова. На відміну від оператора з передумовою, цикл ***починається діями в тілі*** й закінчується обчисленням умови.

Циклу з постумовою відповідає блок-схема на рис. 3.

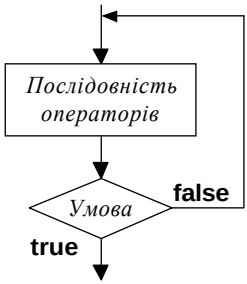


Рис. 3. Блок-схема оператора циклу з постумовою

Істинність умови веде до завершення виконання оператора циклу, тому її називають ***умовою завершення***, або ***виходу з циклу***. Умова перевіряється після виконання тіла циклу, тому її називають ***післяумовою***.

Тіло циклу, заданого оператором з післяумовою, виконується обов'язково ***хоча б один раз*** (на відміну від оператора циклу з передумовою).

Оператор циклу з післяумовою використовують, коли спочатку треба один раз виконати тіло циклу, а потім перевіряти умову його закінчення.

6. ПРИКЛАДИ ОРГАНІЗАЦІЇ ЦИКЛІВ З ПІСЛЯУМОВОЮ

Приклад 1. Знову повернемося до обчислення факторіалу та опишемо його за допомогою оператора **repeat-until**. Цикл домноження на новий множник треба починати за **n>0**. Цикл закінчується, коли множник **k** досягне верхньої межі **n**, тому умовою завершення буде вираз **k=n**.

```
k:=0; fact:=1;
if n > 0 then
  repeat
    inc(k);
    fact:=fact*k;
  until k = n;
{ k = n, fact = n! }
```

Приклад 2. Прочитати три числа, які задають довжини сторін трикутника, та обчислити його периметр і площу. Якщо трикутник з такими сторонами не існує, треба повторити введення.

Нехай **a**, **b**, **c** — дійсні змінні, яким присвоюються введені значення, **p** — змінна для півпериметра. Значення змінних треба спочатку прочитати, і лише потім перевіряти умову того, що вони додатні й задовольняють нерівності трикутника. Якщо ця умова задовольняється, повторювати введення не треба, тому вона має стати умовою виходу з циклу.

Отже, скористаємося **repeat**-оператором:

```
Repeat
  writeln('Задайте довжини сторін трикутника ');
  readln(a, b, c);
until (a>0) and (b>0) and (c>0) and
      (a+b>c) and (a+c>b) and (b+c>a);
```

```
p := (a+b+c)/2; { півпериметр }
writeln('Периметр: ', 2*p);
writeln('Площа:    ', sqrt(p*(p-a)*(p-b)*(p-c)));
```

Приклад 3. Відомо, що сума $1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}$ зі збільшенням числа доданків *n* необмежено зростає. Визначити мінімальну кількість доданків **n**, за яких $\sum_{k=1}^{n}\frac{1}{k}$ більше від заданого додатного дійсного числа *T*.

Будемо обчислювати суми одного, двох, трьох тощо доданків, поки сума не стане більше *T* — кількість доданків у сумі й буде потрібним числом. На кроці обчислення переходимо до наступного номера *k* та додаємо $1/k$ до суми.

```
k:=0; sum:=0;
repeat
  inc(k);
  sum:=sum + 1/k;
until sum > T;
{ k має потрібне значення }
```

Приклад 4. Перевірити, чи утворюють цифри заданого натурального числа спадну послідовність. Наприклад, для числа 96521 відповідь буде «так», а для 9768 — «ні».

Розв'язок цієї задачі базується на таких міркуваннях. Якщо взяти останню цифру числа за еталон, то наступна ліворуч цифра повинна бути більшою за нього. При справдженні цієї умови, можна відкинути останню цифру числа шляхом ділення на 10 і повторити процес. Процес закінчиться, якщо знайдено цифру, що не перевищує еталон, або всі цифри числа розглянуті (число перетворилося на нуль).

```
var N : longint; etalon : byte;
Begin
  write('Введіть число: ');
  readln(N);
  repeat
    etalon := N mod 10;
    N := N div 10;
```

```
  until (N=0) or (N mod 10 <> etalon);
  if N = 0 then writeln('Так.')
  else writeln('Ні.');
End.
```

Приклад 5. Визначити всі прямокутники, що мають натуральні довжини сторін і задану площу S (натуральне число).

Очевидно, що принаймні один прямокутник з площею S існує (одна з довжин сторін дорівнює 1, друга — S). Отже, будемо перебирати довжини k, починаючи від 1, і для кожної перевіримо, чи ділиться S на k. Якщо ділиться, виведемо k та S div k. Значення k більше від кореня квадратного з S, задають прямокутники, в яких сторони просто помінялися місцями, тому розглядати їх не потрібно.

```
var S, k : longint;
Begin
  write('Введіть площу: ');
  readln(S);
  k := 1;
  repeat
    if S mod k = 0
    then writeln(k, '*', S div k, ' = ', S);
    inc(k);
  until k > sqrt(S);
End.
```

Приклад 6. За двома натуральними числами N і M визначити, чи можна N подати як суму двох ненульових доданків, сума квадратів яких дорівнює M. Наприклад, $10 = 7+3$, $58 = 7^2 + 3^2$.

Незважаючи на те, що необхідно підбирати два доданки, достатньо одного циклу, адже другий доданок визначається на основі умови. Якщо позначити доданки через x і y, то $y = \sqrt{M - x^2}$.

Цикл перебору працює, поки не знайдено розв'язок та x не стало більше y. Якщо вихід з циклу відбудеться за першою умовою, то розв'язок знайдено, інакше розв'язків немає.

```
var x, y, N : longint;
begin
  write('Введіть два числа: ');
```

```
  readln(N,M);
  x := 0;
  repeat
    inc(x);
    y := trunc(sqrt(M-sqr(x)));
  until (x>y) or (sqr(x)+sqr(y)=M);
  if x>y
  then writeln('Розв'язок відсутній.')
  else writeln('Розв'язок: ', x, '  ', y);
End.
```

Приклад 7. За натуральним числом N визначити всі трійки натуральних чисел x, y, z, за яких $x^2 + y^2 + z^2 = N$.

Використаємо вкладені цикли. У зовнішньому циклі (зі змінною **x**) переберемо цілі числа від 0 до кореня квадратного з N (це найбільше значення, якого можуть набувати змінні). Діапазон значень у середньому циклі (зі змінною **y**) можна дещо скоротити, оскільки на початку його виконання відоме значення «зовнішньої» змінної **x**. Аналогічно скорочується й діапазон значень у внутрішньому циклі (зі змінною **z**), оскільки на початку його виконання відомі значення і **x**, і **y**.

Задача може не мати розв'язків, тому використаємо змінну булевого типу, яка фіксує, чи знайдено хоча б один розв'язок. Спочатку цій змінній присвоїмо **false** (розв'язок ще не знайдено).

```
var x, y, z, N : longint;
 flag : Boolean;
Begin
  write('Введіть число: ');
  readln(N);
  flag:=false;
 writeln('Розв''язки рівняння x^2+y^2+z^2 = ', N);
  x:=-1;
  repeat
    inc(x); y:=-1;
    repeat
      inc(y); z:=trunc(sqrt(N-x*x-y*y))-1;
      repeat
        inc(z);
          if sq(x)+sqr(y)+sqr(z) = N
```

```
        then begin
               writeln(x,y,z);
               flag:=true;
             end
      until sqr(z) > N - sqr(x) - sqr(y);
    until sqr(y) > N - sqr(x);
  until x > sqrt(N);
  if not flag
  then writeln('Розв''язків немає.');
End.
```

7. ПОНЯТТЯ СТРУКТУРНОГО ПРОГРАМУВАННЯ

Структурне програмування — це написання коду, який має певну структуру й завдяки цьому є зручним для розуміння, наладки, внесення змін та використання.

Структурне програмування полягає у використанні невеликого набору простих керуючих структур (структурних операторів), правильність яких легко проаналізувати й установити. При цьому оператори складаються з інших операторів, ***вкладених*** у них.

Властивість структурності операторів полягає в тому, що ***кожен оператор має один вхід і один вихід***. Програма, записана структурними операторами, називається ***структурованою***.

Терміни «структурний оператор» і «структурована програма» з'явилися в 60-ті роки. Спочатку структурних операторів було три — складений **begin-end**, розгалуження **if-then-else** і циклу **while**. Їх структурність очевидна (див. рис. 4.1, 5.2). Було доведено, що будь-яку неструктуровану програму можна перетворити на структуровану, яка задає ті самі обчислення, хоча й за рахунок додавання нових змінних і дублювання фрагментів програми.

У мові Паскаль до структурних операторів додано оператори вибору варіантів **case** та циклу **repeat-until** і **for**. Вони теж мають один вхід і один вихід і є структурними. За допомогою структурних операторів мови Паскаль утворюються структуровані програми.

У мові Паскаль є засіб, який порушує структурованість програми — це ***оператор безумовного переходу на мітку***. Але в цій книжці мітки та оператори переходу не використовуються.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яким може бути тип параметра **for***-циклу?*

2. У чому різниця між операторами **for-to** *і* **for-downto***?*

3. В якому випадку значення параметра циклу після завершення виконання оператора **for** *залишається невизначеним?*

4. Яке значення має параметр **for***-циклу після його закінчення, якщо тіло циклу було виконано не менше одного разу й не переривалося операторами переходу?*

5. Чи може цикл з параметром **for** *виконуватися нескінченно?*

6. Якими можуть бути причини зациклювання програми?

7. У чому різниця між операторами циклу з передумовою та з післямовою?

8. В якому випадку тіло циклу з передумовою не виконується?

9. Опишіть загальний вигляд операторів циклів у мові Паскаль.

10. Чи будь-який циклічний процес можна запрограмувати, з різних форм циклів використовуючи тільки одну?

ЗАДАЧІ

1. Імітувати виконання послідовності операторів (усі змінні типу **longint**). Пояснити зв'язок між значеннями **i** та **x** і **y**:

```
a) i:=1; x:=1; y:=2;
   while x < y do begin
     inc(i); x := x*i; y := y*2;
     writeln(i, ' ', x, ' ', y)
   end;

b) i:=1; x:=1; y:=2;
  while i <= 10 do begin
    inc(i); x := x*i; y := y*2;
    writeln(i, ' ', x, ' ', y)
  end;
```

2. Знайти найбільше додатне ціле число n, для якого виконується умова $3n^2 - 730n < 5$.

3. За заданими a, b і h (або m) надрукувати значення функції **arctg(sin x)** у точках $a, a+h, \ldots, a+mh$, де:

a) m обчислюється за h, виходячи з того, що $a+mh \;?\; b$ і $a+(m+1)h > b$;

b) h обчислюється за m з рівності $a+mh = b$.

4. За заданим n обчислити значення виразу

$$\sqrt{3+\sqrt{6+\ldots+\sqrt{3(n-1)+\sqrt{3n}}}} \quad (n \text{ коренів}).$$

5. Трикутник Паскаля має такий початок.:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Числа в трикутнику — це біномні коефіцієнти, тобто в i-му рядку ($i = 0$, 1, 2, …) на j-му місці ($j = 0, 1, \ldots, i$) записано C_i^j. Написати програму друкування перших 13 рядків трикутника Паскаля (їх номери від 0 до 12). Числа повинні відокремлюватися не менш ніж одним пропуском.

6. За цілим $A > 0$ обчислити:

a) найменше n, при якому $n! > A$;

b) найбільше n, при якому $n!$? A.

7. Надрукувати перші k дробових знаків двійкового представлення дійсного числа x.

8. Дано натуральне число N. Визначити:

a) чи зустрічається в ньому задана цифра;

b) яка цифра в ньому зустрічається частіше *a* чи *b;*

c) яка в ньому цифра мінімальна (максимальна).

9. Написати програму, яка вводить два натуральних числа M і N і виводить кожне ціле число від M до N, що має таку властивість:

a) всі десяткові цифри числа парні;

b) послідовність десяткових цифр числа є монотонною (як неспадною, так і незростаючою);

c) p-кові цифри числа утворюють арифметичну прогресію;

d) p-кові цифри числа утворюють геометричну прогресію;

10. Дано натуральне число N. Визначити, чи є це число:

a) таким, що різниця його максимальної та мінімальної цифр число парне;

b) «щасливим» (сума цифр першої половини числа дорівнює сумі цифр другої половини, наприклад, 1203, 205007, 23004001);

c) таким, що сума його цифр на непарних місцях дорівнює сумі цифр на парних місцях, наприклад, 1210, 432410);

d) симетричним (перша половина числа співпадає з другою, наприклад, 22, 1414, 453453);

e) паліндромом (читається в обидва боки однаково, наприклад, 232, 456654);

f) Армстронга (число дорівнює сумі своїх цифр, піднесених до степеню, що дорівнює кількості цифр в числі, наприклад, $153 = 1^3 + 5^3 + 3^3$);

g) досконалим (число дорівнює сумі всіх своїх натуральних дільників, крім самого числа, наприклад, 6=1+2+3, 28=1+2+4+7+14).

11. Дано два натуральних числа *N* та *M*. Знайти всі спільні дільники цих чисел.

12. Дано натуральне число *N*. В інтервалі від 1 до *N* знайти натуральне число, яке має максимальну суму дільників.

13. За заданим *n* обчислити значення виразу

$$\sum_{i=1}^{n} \frac{\sum_{j=1}^{i} \cos j}{\sum_{j=1}^{i} \sin j}.$$

Тригонометричні функції для кожного аргументу мають обчислюватися тільки по одному разу.

14. Значення функцій $\sin x$ та $\cos x$ треба вивести на екран у два стовпчики для значень $x = a, a+h, a+2h, a+3h, \ldots$ у такий спосіб. Спочатку друкуються перші *m* значень і виводиться запит, чи продовжувати друкування. Після відповіді користувача «1» друкуються наступні *m* значень і запит, і так далі. Робота завершується після відповіді «0». Написати програму введення значень *a*, *h*, *m* і описаного друку значень «сторінками».

15. Дано два натуральних числа *N* ($N<100000$) та *M* ($M<10$). Розробити власний навчальний алгоритм множення у стовпчик даних чисел, демонструючи по кроках процес виконання дій: запис одиниць результату у поточний розряд та перенесення десятків у наступний розряд. Перехід до наступного кроку здійснюється натисканням будь-якої клавіші.

16. Дано два натуральних шестицифрових числа *N* та *M*. Розробити власний навчальний алгоритм додавання (віднімання) у стовпчик даних чисел, демонструючи по кроках процес виконання дій: запис одиниць результату у поточний розряд та перенесення десятків у наступний роз-

ряд. Перехід до наступного кроку здійснюється натисканням будь-якої клавіші.

17. Дано два натуральних числа *N* та *M*. Знайти НСК (найменше спільне кратне) цих чисел.

18. Трикутники, в яких довжини сторін і площ — натуральні числа, називають трикутниками Герона (наприклад, трикутник зі сторонами 13, 14, 15 та площею 84 є трикутником Герона). Скласти програму визначення трикутників Герона, довжини сторін яких не перевищують даного натурального *N*.

19. Дано натуральне число *N*. Визначити всі нескорочувані дроби із знаменником *N* в інтервалі (0,1).

20. Задано натуральне число *k*. Вивести *k*-ту цифру послідовності 12345678910111213..., в якій виписані підряд усі натуральні числа.

ðîçä³ë 6

ÄÎÏÎÌ²ÆÍ² ÀËÃÎÐÈÒÌÈ, ÀÁÎ Ï²ÄÏÐÎÃÐÀÌ È

1. ПІДПРОГРАМА — ОПИС РОЗВ'ЯЗАННЯ ПІДЗАДАЧІ

Програма — це опис розв'язання деякої інформаційної задачі. Процес побудови програми починається з аналізу задачі. Практично в кожній задачі, аналізуючи її, можна виділити окремі ***підзадачі***, допоміжні до неї. Розв'язання підзадачі можна описати в окремій, спеціально оформленій, частині програми. Такий опис розв'язання підзадачі називають ***підпрограмою***. Щоб вказати виконання підпрограми, у програмі достатньо записати її ***виклик***.

Деякі підзадачі доводиться розв'язувати майже в усіх задачах, наприклад, уведення даних із зовнішніх носіїв у змінні програми та їх виведення на носії. Для таких ***стандартних*** підзадач кожна система програмування надає великий набір ***готових підпрограм***. Нам уже знайомі підпрограми уведення/виведення **`readln`**, **`writeln`**, а також обчислення математичних функцій **`sin`**, **`abs`** тощо. У їх викликах ми лише вказуємо вирази або змінні, що мають оброблятися під час їх виконання.

Уявіть собі програму читання двох чисел, їх додавання та друкування суми, якби в ній замість викликів підпрограм **`readln`** та **`writeln`** був опис усіх дій з уведення та виведення даних. Її текст був би у багато разів довшим, а її сутність, додавання чисел, у цьому тексті просто б загубила-

ся. Отже, згадана програма додавання двох чисел є простою саме завдяки стандартним підпрограмам.

У практичному програмуванні знати стандартні підпрограми корисно й необхідно, адже застосовувати готові деталі набагато легше, ніж створювати їх самому.

Стандартні підпрограми у системах програмування зібрано у спеціальні набори — *бібліотеки*. Ці підпрограми додаються до машинної програми у процесі компонування.

Проте користуватися можна не лише стандартними підпрограмами. Мови програмування дозволяють створювати ***власні підпрограми***, тобто описувати розв'язання потрібних підзадач у окремих програмних одиницях.

Виділення окремих підзадач суттєво полегшує проектування та розробку програми для розв'язання всієї задачі.

Використання підпрограм для розв'язання підзадач дозволяє зробити програму добре структурованою й зрозумілою.

Алгоритм, реалізований підпрограмою, може бути довгим і складним, але він «захований» у підпрограму. Завдяки цьому програма стає простою й короткою, її легше написати й налагодити.

2. ПРОЦЕДУРА — ПІДПРОГРАМА, ЯКА РЕАЛІЗУЄ ОКРЕМИЙ ОПЕРАТОР

У мові Turbo Pascal є два види підпрограм — процедури та функції. Вони відрізняються формами запису. Окрім цього, виклики процедур являють собою окремі оператори (як виклики **writeln** або **readln**), а виклики функцій — вирази (арифметичні, булеві тощо — як виклики **sin**, **ord** або **odd**). Спочатку розглянемо ***процедури***.

Приклад 1. Прочитати два числа типу **integer**, визначити мінімальне з них та вивести його на екран.

У цій задачі легко виділити підзадачі: *прочитати числа*, *визначити мінімальне з двох чисел, вивести відповідь*. Перша з них розв'язується за допомогою стандартних підпрограм **writeln** і **readln** — спочатку треба вивести на екран запрошення до введення чисел, а потім отримати їх. Щоб вивести відповідь, достатньо одного виклику **writeln**.

Для розв'язання підзадачі «визначити мінімальне з двох чисел» напишемо ***власну процедуру***. Звичайно, щоб визначити мінімальне з двох чи-

сел, можна обійтися без підпрограм, адже треба лише перевірити нерівність вигляду **a<b** та виконати одне з двох присвоювань вигляду **c:=a** або **c:=b**. Проте на цьому простому прикладі ми розглянемо важливі поняття, пов'язані з підпрограмами.

Підпрограми, як і програми, складаються із ***заголовка*** й ***блоку***, лише заголовок має дещо інший вигляд, а після блоку замість «**.**» має бути «**;**». Блок підпрограми, як і блок програми, складається з оголошень імен та тіла підпрограми.

Заголовок підпрограми є ***оголошенням*** її імені, оскільки визначає спосіб використання цього імені у подальшому.

У нашій підзадачі присутні три величини — два «вхідних» числа та одне «результатне» (мінімальне з тих двох). Вони є ***параметрами підзадачі***. Ці величини вкажемо як ***параметри процедури в її заголовку***. Дії з ними опишемо в тілі процедури. Програма з процедурою та її викликом має такий вигляд:

```
program MinOfTwoNumbers;
  var a, b, c : integer; { змінні програми }
{ процедура обчислення мінімуму }
procedure min(x, y : integer; var z : integer);
begin
  if x<y then z:=x else z:=y
end;
{ тіло програми }
Begin
  writeln('Задайте два цілих числа:')
  readln(a,b);
  min(a,b,c);     { виклик процедури min }
  writeln('Мінімальне з них: ',c);
End.
```

У першому рядку підпрограми записано її заголовок. Службове слово **procedure** свідчить, що це саме процедура, **min** є ***іменем процедури***. Вхідні числа при виконанні виклику процедури є значеннями параметрів **x** і **y**, результат запам'ятовується як значення параметра **z**. Ніяких інших імен, окрім параметрів, у нашій підзадачі немає, тому оголошення імен у цьому блоці відсутні. Що означає слово **var** перед іменем **z**, уточнимо нижче.

При виконанні програми після читання значень змінних **a** та **b** виконується виклик процедури **min**; у виклику вказано величини, з якими треба виконати процедуру (**a**, **b** та **c**).

На початку виконання виклику утворюються ***нові окремі змінні***, відповідні параметрам **x** і **y** (***але не z!***). Значення **a** та **b** присвоюються цим новим змінним. Далі виконуються оператори, записані в тілі процедури: змінна з іменем **z** отримує потрібне значення. Завдяки слову **var** перед **z** у заголовку підпрограми, ця змінна ***не є новою*** — цю саму змінну позначає ім'я **c**, оголошене у програмі.

Отже, ***присвоювання змінній z при виконанні процедури в дійсності є присвоюванням змінній c***! Завдяки цьому значення, отримане **z**, після закінчення виклику підпрограми залишається в змінній **c**.

Проілюструємо виконання програми таблицею, припустивши, що змінні **a** та **b** під час уведення отримують значення **1** і **2**. У стовпчиках таблиці, що показують стани пам'яті, подано змінні, тобто ділянки пам'яті. Змінні, відповідні параметрам процедури, позначимо, додавши ім'я процедури: **min.x**, **min.y**, **min.z**. Як бачимо, імені **c** та параметру **min.z** відповідає один і той самий стовпчик, тобто одна й та сама змінна:

Виконувані дії	**Пам'ять програми**				
Утворення змінних програми	**a**	**b**	**c**		
...readln(a, b)	**1**	**2**	**?**		
min(a, b, c)	**1**	**2**	**?**		
Утворення нових змінних	**1**	**2**	**min.z**	**min.x**	**min.y**
Присвоювання **x** *та* **y**	**1**	**2**	**?**	**1**	**2**
Обчислення **x < y**	**1**	**2**	**?**	**1**	**2**
z := x	**1**	**2**	**1**	**1**	**2**
Закінчення виклику **min**	**1**	**2**	**1**		
writeln(…, c)	**1**	**2**	**1**		

Зауважимо: якби в тілі процедури було змінено **x** або **y**, це не вплинуло б на значення **a** та **b**, оскільки іменам **x** та **y** відповідають «власні» змінні.

Приклад 2. Прочитати три числа типу **integer**, визначити мінімальне з них та надрукувати його.

Скористаємося тим, що у нас уже є процедура обчислення мінімального з двох значень. Спочатку визначимо мінімальне з перших двох чисел, а потім знайдемо мінімальне з цього «першого мінімуму» та третього числа. Отже, двічі викликаємо процедуру **min**, указавши у викликах відповідні величини.

```
program MinOfThreeNumbers;
  var a, b, c, m1, m2 : integer;
procedure min(x, y : integer; var z : integer);
begin
  if x<y then z:=x else z:=y
end;
Begin
  writeln('Задайте три цілих числа:')
  readln(a,b,c);
  min(a,b,m1);  { m1 -- перший мінімум }
  min(m1,c,m2); { m2 -- другий мінімум }
  writeln('Мінімальне з них: ',m2);
End.
```

Другий виклик можна записати як **min(m1,c,m1)** і потім надрукувати значення **m1**, а не **m2**, тобто взагалі позбутися змінної **m2**. На початку цього виклику значення змінної **m1** присвоюється параметру **x**, а потім значення **m1** змінюється при виконанні **z:=x** або **z:=y**.

Приклад 3. Прочитати три цілих числа та видати їх, упорядкувавши за неспаданням.

Уведемо числа в змінні a, b, c та за потреби обміняємо їх значення так, щоб справджувалися нерівності $a \le b \le c$, а потім виведемо значення a, b, c.

Уточнимо дії з упорядкування значень.

```
if a>b then обміняти значення змінних a і b;
  { нерівність a<=b істинна }
if b>c then обміняти значення змінних b і c;
  { нерівності b<=c і a<=c істинні, тобто
    значення c - найбільше, }
  { але a<=b може стати хибним, тому: }
if a>b then обміняти значення змінних a і b.
```

У цьому алгоритмі явно виділяється підзадача «обміняти значення двох змінних», яку треба розв'язати тричі, лише з різними парами змінних. Для розв'язання цієї задачі напишемо процедуру; її параметри вказують на змінні, значення яких обмінюються.

```
program reorder3;
  var a, b, c : integer;
  { процедура обміну значень її параметрів }
```

```
  procedure exchange(var x, y : integer);
    var t : integer; { допоміжна змінна }
  begin
    t:=x; x:=y; y:=t
  end;
Begin
  writeln('Задайте три цілих числа');
  readln(a,b,c);
  if a>b then exchange(a,b); { a <= b }
  if b>c then exchange(b,c); { c - найбільший }
  if a>b then exchange(a,b); { a <= b }
  writeln('За неспаданням: ',a,' ',b,' ',c)
End.
```

У процедурі **exchange** оголошено допоміжну змінну **t**, яка використовується в тілі процедури. При виконанні першого та третього викликів **exchange** її параметри **x** та **y** посилаються на змінні **a** та **b**, при виконанні другого — на **b** та **c**.

ПІДСУМКИ ТА УТОЧНЕННЯ

У мові Turbo Pascal підпрограми записуються ***серед оголошень*** імен програми (в інших мовах може бути по-іншому).

Процедура (один з двох різновидів підпрограм) має такий загальний вигляд.

```
procedure ім'я(оголошення параметрів);
  оголошення імен та підпрограми
begin
  послідовність операторів
end;
```

Імена, оголошені в заголовку підпрограми, називаються ***параметрами***, або ***формальними параметрами***. Вирази або імена змінних, записані у виклику підпрограми, називаються ***аргументами***, або ***фактичними параметрами***.

Параметри, оголошені без слова **var** перед їх іменем, називаються ***параметрами-значеннями***, а оголошені зі словом **var** перед їхнім іменем — ***параметрами-змінними***.

Оголошення параметрів у заголовку підпрограми записують у дужках як послідовність ***секцій***. У секції оголошено або однотипні параметри-значення, або однотипні параметри-змінні (зі словом **var** на початку

секції). Секції відокремлюють знаком «**;**». Проте у виклику незалежно від секцій усі аргументи відокремлюються комами.

Якщо підпрограма не має параметрів, дужки **()** в її заголовку не записують. Тоді її викликом є її ім'я.

3. ДВА СПОСОБИ ПІДСТАНОВКИ АРГУМЕНТІВ НА МІСЦЕ ПАРАМЕТРІВ

ПІДСТАНОВКА ЗА ЗНАЧЕННЯМ

Параметр-значення позначає у підпрограмі «власну» змінну. На початку виконання виклику підпрограми цій змінній ***присвоюється значення аргументу***. Відповідним аргументом може бути ***довільний вираз*** того ж типу, що й тип параметра (або типу, сумісного за присвоюванням з типом параметра).

Приклади аргументів, що є виразами, нам знайомі з викликів стандартних підпрограм: **sqrt(b*b-4*a*c)**, **chr(ord('0')+1)**, **writeln(x*x)** тощо.

У викликах нашої процедури **min** перші два аргументи теж можуть бути довільними виразами типу **integer**, наприклад **min(10+x,3*x,v)**, де **x**, **v** — імена цілих змінних.

Цей спосіб передачі даних у підпрограму називається підстановкою аргументу на місце параметра ***за значенням***.

ПІДСТАНОВКА ЗА ПОСИЛАННЯМ

Параметр-змінна під час виконання підпрограми позначає ту саму змінну, яку позначає відповідний йому аргумент, вказаний у виклику. Щоб забезпечити це, до пам'яті підпрограми передається ***посилання*** на цю змінну. Відповідним аргументом може бути ***ім'я змінної*** того ж типу, що й у параметра.

Приклади аргументів, що відповідають параметрам параметрам-змінним: **readln(a,b,c)** або **inc(x)**, а також третій аргумент у викликах нашої процедури **min**.

Цей спосіб передач даних у підпрограму називається підстановкою аргументу на місце параметра ***за посиланням***.

ЩО ВИБИРАТИ?

Якщо після виклику підпрограми використовується ***старе значення*** аргументу або аргументом може бути довільний ***вираз***, йому має відповідати ***параметр-значення***.

Якщо після виклику підпрограми має використовуватися ***нове значення*** аргументу, отримане при виконанні виклику, то параметр треба оголосити як ***параметр-змінну***.

Перше правило має винятки, пов'язані з масивами (див. розділ 7). Параметри **x** та **y** процедури **min** (див. вище) ілюструють перше правило, параметр **z** — друге.

ПАРАМЕТРИ-КОНСТАНТИ

У мові Turbo Pascal є ще один різновид параметрів — ***параметри-константи***; вони оголошуються зі словом **const** на початку. Аргументом для них може бути довільний ***вираз*** того ж типу, що й параметр.

Підстановка відбувається так. Значення аргументу обчислюється й запам'ятовується в ділянці пам'яті підпрограми, яка викликає (аналогічно до параметров-значень), а у підпрограму, яку викликано, передається посилання на цю ділянку (як для параметрів-змінних). Отже, цей механізм дозволяє використовувати ***аргументи-вирази*** й ***не вимагає копіювання*** їх значень до пам'яті виклику підпрограми. Присвоювання параметрам-константам заборонено; їх виявляє транслятор.

Параметри-константи зручні, коли потрібні параметри великого розміру, не змінювані у підпрограмі; приклади їх застосування наведено в розділі 7.

4. ФУНКЦІЯ — ПІДПРОГРАМА ОБЧИСЛЕННЯ ЗНАЧЕННЯ, ПОТРІБНОГО У ВИРАЗІ

Функція є другим різновидом підпрограми. Виклик функції є ***виразом*** (числовим, булевим тощо). Значенням цього виразу стає значення, яке обчислюється при виконанні виклику функції та в деякий спеціальний спосіб ***повертається у програму***.

Приклад 1. Згадаємо обчислення мінімального з двох цілих значень. Результатом цього обчислення є одне скалярне значення (число), яке зручно використовувати як вираз, наприклад, вказати його як аргумент у виклику процедури **writeln**.

У наведеній вище процедурі **min** потрібне значення поверталося у пам'ять програми як значення аргументу, відповідного параметру-змінній. Проте це повернення можна організувати інакше. Розглянемо програму обчислення мінімального з двох заданих цілих чисел за допомогою функції.

```
program MinOfTwoNumbers;
  var a, b : integer;
{ функція обчислення мінімуму }
function min(x,y : integer) : integer;
begin
  if x<y then min:=x else min:=y
end;
Begin
  writeln('Задайте два цілих числа:')
  readln(a,b);
  {виклик функції-аргумент у виклику процедури}
  writeln('Мінімальне з них: ', min(a,b));
End.■
```

Підпрограма, що є функцією, починається службовим словом **function**. Після дужок з параметрами записується двокрапка та ***тип значення, яке повертається з виклику функції***.

Повернення значення відбувається за допомогою спеціальної змінної, яку позначає ім'я функції (у наведеному прикладі це **min**). У тілі функції обов'язково мають бути оператори ***присвоювання з іменем функції*** в лівій частині, причому при виконанні виклику має виконуватися хоча б один з них. Остаточне значення цієї змінної ***повертається з виклику*** функції.

Проілюструємо виконання наведеної програми, вважаючи, що змінні **a** та **b** під час уведення отримують значення **1** і **2**. Змінна, однойменна з функцією, має «власну» ділянку пам'яті й у таблиці позначена **min.min**. Коли закінчується виклик **min**, її значення присвоюється додатковій змінній, яка є у «машинній» програмі й позначена «Аргумент **writeln**» (див. табл. на с. 106).

Виклик функції є виразом і може бути частиною складнішого виразу (ми бачили це раніше, використовуючи стандартні функції).

Приклади використання викликів min

1. Якщо з якоюсь метою нам треба обчислити квадрат значення, що повертається з виклику **min**, або додати до нього **1**, можна написати вираз **sqr(min(a,b))** або, відповідно, **min(a,b)+1**.

2. Для обчислення мінімального з трьох цілих значень **a**, **b**, **c** можна записати такі виклики функції **min**.

```
m:=min(a,b); m:=min(m,c)
```

Таблиця

Виконувані дії	Пам'ять програми					
Утворення змінних програми	a	b	Аргумент writeln			
... readln(a, b)	1	2	?			
min(a, b)	1	2	?			
Утворення нових змінних	1	2		min.x	min.y	min.min
Присвоювання x *та* y	1	2	?	1	2	?
Обчислення x < y	1	2	?	1	2	?
min := x	1	2	?	1	2	1
Закінчення виклику тіп	1	2	1			
writeln(...)	1	2	1			

Водночас, мінімальне з трьох цілих значень **a**, **b**, **c** є значенням виразу **min(min(a,b),c)**. При його обчисленні, коли починає виконуватися «зовнішній» виклик **min**, треба обчислити значення першого аргументу, а ним є «внутрішній» виклик **min(a,b)**. Тому виконується цей «внутрішній» виклик. Лише потім значення, що повертається з нього, присвоюється першому параметру у виконанні «зовнішнього» виклику.

Аналогічно мінімальне з чотирьох значень задає вираз **min(min(a,b),min(c,d))**.

Приклад 2. Прочитати довжини чотирьох відрізків (гарантовано, що це попарно різні додатні цілі числа) та обчислити кількість трикутників, які з них можна утворити.

Щоб обчислити кількість трикутників, треба для кожної з чотирьох можливих трійок відрізків перевірити, чи можна утворити з них трикутник. Отже, нам треба одні й ті самі обчислення провести чотири рази, тільки з різними числами.

Напишемо функцію, яка обчислює ознаку, чи можна з трьох відрізків утворити трикутник. Але повертати з неї будемо не булеве значення цієї ознаки, а ціле (**0** відповідає **false**, **1** — **true**). Це дозволить просто надрукувати суму цих ознак:

```
program NumberOfTriangles;
  var a, b, c, d : integer;
function triangle(x, y, z : integer) : integer;
begin
  triangle:=ord((x+y>z) and (x+z>y) and (y+z>x))
end;
Begin
  writeln('Задайте чотири додатних цілих числа:')
  readln(a,b,c,d);
  writeln('Кількість трикутників: ',
    triangle(a,b,c)+triangle(a,b,d)+
    triangle(a,c,d)+triangle(b,c,d))
End.
```

Ім'я функції, записане ***на місці виразу*** (праворуч у операторі присвоювання, в умові розгалуження або циклу, як аргумент у виклику тощо), позначає ***виклик*** цієї функції. Якщо ця функція має параметри, а у виразі записано лише її ім'я, це є ***синтаксичною помилкою***.

Записувати ім'я функції у викликах процедур уведення заборонено (на відміну від запису в лівих частинах операторів присвоювання).

5. ОБЛАСТЬ ДІЇ ОГОЛОШЕНЬ ІМЕН

Під час використання програм з підпрограмами постає природне питання: чи можна у програмі та підпрограмах використовувати ***одні й ті самі імена***, що позначають ***різні об'єкти*** (константи, змінні, типи, підпрограми)? Відповідь на це запитання пов'язана з поняттям «область дії оголошення імені». Розглянемо його.

Область дії оголошення імені — це сукупність місць у програмі, в яких ім'я позначає об'єкт, заданий саме в цьому оголошенні.

За правилами мови Паскаль, оголошення імені діє ***від місця запису*** у програмі або підпрограмі ***до кінця її блоку***. Якщо в цій області є підпрограми, то воно діє і в них. Але якщо вони містять своє власне оголошення цього ж імені, то за тим же правилом до кінця їх блоків діють їх власні оголошення.

Отже, власне оголошення у підпрограмі «перекриває» оголошення, записане вище. Наприклад, оголошення імен **x**, **y** у заголовку функції **min** діють до кінця тіла цієї функції та перекривають можливі оголошення

цих імен у програмі. У процедурі **exchange** оголошення імені **t** діє до кінця тіла **exchange**, тобто у програмі змінну з іменем **t** «не видно».

Різним об'єктам у різних підпрограмах (однією з них вважається програма) ***можна давати однакові імена***. Це дозволяє оголошувати імена у підпрограмах, ***не замислюючись***, чи вже оголошено ці імена у програмі або інших підпрограмах.

Заголовок підпрограми є оголошенням її імені, розташованим у програмі (або іншій підпрограмі). Тому підпрограму можна викликати не тільки у програмі, а й ***у підпрограмах***, ***які записано нижче*** в цій самій програмі.

Заголовок підпрограми передує її блоку, тому ***викликати підпрограму можна в ній самій*** (такі виклики називаються рекурсивними — див. нижче). Також ***не можна*** оголошувати ім'я підпрограми в її ж блоці.

Ім'я, оголошене у підпрограмі, називається ***локальним у цій підпрограмі***, а ім'я, яке використовується у підпрограмі без оголошення — ***глобальним у ній***. Змінні з глобальними іменами називаються ***глобальними***.

Щоб при імітації програми відрізняти локальні імена підпрограми, перед локальним іменем додають ім'я підпрограми й крапку (див. імітацію процедури та функції **min** у прикладах вище). Тож, якщо в підпрограмі **S** оголошено ім'я **N**, його при імітації позначають **S.N**. Під час виклику підпрограми її параметрам-змінним відповідає пам'ять, «належна» аргументам. При імітації програми параметр-змінна підпрограми «накладається» на аргумент, а імена інших змінних, оголошених у підпрограмі, вказують на власні ділянки пам'яті (див. приклади імітації вище).

6. ВИКОНАННЯ ВИКЛИКУ ПІДПРОГРАМИ

ПАМ'ЯТЬ ВИКЛИКУ ПІДПРОГРАМИ

Сукупність змінних, які утворюються при виконанні виклику підпрограми, має назву ***пам'ять виклику підпрограми***. Змінні в цій пам'яті відповідають ***параметрам-значенням*** та ***локальним іменам змінних*** підпрограми. Цю пам'ять часто ще не зовсім точно називають ***локальною пам'яттю*** підпрограми.

Змінні в пам'яті виклику підпрограми називаються ***локальними***. Якщо підпрограма є функцією, то її локальна пам'ять містить також змінну, однойменну з функцією; її значення повертається з виклику. Наприклад, локальну пам'ять процедури **exchange** (див. приклад вище) утворює

змінна з іменем **t**, а функції **min** (див. приклад вище) — змінні, відповідні параметрам **x**, **y** та імені функції **min**.

Локальна пам'ять містить ще один елемент — посилання на місце, з якого треба продовжити виконання програми після виклику. Це місце називається ***точкою повернення з підпрограми***, а посилання на неї зберігається до закінчення виконання виклику підпрограми.

ПРОЦЕС ВИКОНАННЯ ВИКЛИКУ ПІДПРОГРАМИ

1. Виділяється пам'ять для точки повернення та параметрів-значень підпрограми. Посилання на точку повернення з підпрограми запам'ятовується.
2. Обчислюються значення аргументів для параметрів-значень, посилання на пам'ять аргументів для параметрів-змінних (а також і перше, і друге для параметрів-констант). Аргументи підставляються.
3. Виділяється пам'ять, відповідна локальним іменам змінних.
4. Виконуються оператори тіла підпрограми. Зокрема, в локальній пам'яті функції запам'ятовується значення, що повертається з виклику.
5. Якщо підпрограма є функцією, то значення, що повертається з її виклику, копіюється з її пам'яті до пам'яті програми (або підпрограми, яка викликає).
6. Програма (підпрограма, яка викликає) продовжується з точки повернення.

Змінні в локальній пам'яті підпрограми не відповідають іменам програми (підпрограми, яка викликає), тобто ця пам'ять ***недоступна*** у програмі і ***вважається звільненою***; її можна використовувати для наступного виклику цієї ж або іншої підпрограми.

Відбувається так зване ***логічне звільнення***: зміст локальної пам'яті не змінюється й перетворюється на «сміття».

АВТОМАТИЧНА ПАМ'ЯТЬ, АБО ПРОГРАМНИЙ СТЕК

Змінні, оголошені на рівні програми, існують протягом усього часу виконання програми й тому називаються ***статичними***. Область пам'яті з ними також називається ***статичною***. Локальним іменам і параметрам-значенням підпрограм відповідають змінні з іншої області пам'яті — ***автоматичної***.

Назва «автоматична» пояснюється тим, що при виконанні викликів підпрограм пам'ять виділяється і звільняється без явних вказівок у програмі, написаній мовою високого рівня, тобто «автоматично».

Локальні імена, оголошені у підпрограмах, можуть збігатися з іменами у програмі та інших підпрограмах, оскільки вони вказують на цілком різні об'єкти в різних областях пам'яті. Ось чому на локальні імена у підпрограмах немає ніяких обмежень.

При виконанні викликів підпрограм ділянки автоматичної пам'яті виділяються й звільняються за принципом «останньою зайнято — першою звільнено».

Якщо складати аркуші паперу в стос і брати їх тільки зверху, то аркуш, що потрапив до стосу останнім, забирають першим. Англійською мовою такий стос називається *stack* (стек), а кладуть і беруть аркуші за принципом «*Last In — First Out*» (LIFO), тобто «останнім прийшов — першим пішов». Тому автоматичну пам'ять програми також називають ***програмним стеком***.

Аналогічно потрапляють набої в магазин автомата й вистрілюються з нього. Останній вилітає першим, а перший (він на дні магазина) — останнім. Можна вистрілити набій з магазина й додати згори новий. Аналогічно закінчується виконання виклику підпрограми, записаного в деякому тілі (програми або підпрограми), і починається виконання наступного виклику з цього ж тіла.

7. ПРИКЛАДИ ВИКОРИСТАННЯ ПІДПРОГРАМ

Приклад 1. Прочитати два натуральних числа a і b, де $0 \le a \le b$, та надрукувати всі прості числа від a до b.

Простий спосіб розв'язання цієї задачі — перебрати всі натуральні числа від a до b і для кожного визначити, чи є воно простим. Відразу виділимо підзадачу «визначити, чи є задане натуральне число простим». Результат цього визначення, тобто відповідь «так» або «ні», представимо булевим значенням, яке повертається з функції. Ця функція матиме такий заголовок.

```
function isPrime(n : longint) : boolean;
```

Спочатку напишемо програму з викликом цієї функції, не уточнюючи її:

```
program primes;
function isPrime(n : longint) : boolean;
    ... { див. нижче }
end;
```

```
var a, b,         { межі пошуку простих чисел }
    i : longint; { поточне натуральне число }
begin
  write('Задайте два натуральних числа ',
          '(друге не менше першого): ');
  readln(a,b);
  for i:=a to b do
    if isPrime(i) then write(' ', i)
end.
```

Як бачимо, програма дуже проста. Її можна трохи прискорити, якщо врахувати, що всі парні числа, більші за 2, є складеними. Тоді треба окремо перевірити, чи міститься 2 у заданих межах, а потім перебрати лише непарні числа.

Розглянемо алгоритм визначення, чи є задане число простим, який буде реалізовано функцією **isPrime**. Число n, відмінне від 1, є простим, якщо має тільки два додатних дільники — 1 і n. Число 2 є простим, а n, $n > 2$, — якщо не ділиться без остачі на жодне з чисел 2, 3, …, $n - 1$; в іншому випадку число є складеним. Отже, треба перебрати всі дільники k від 2 до $n - 1$ і перевірити подільність n на k. Для цього знадобиться цикл з умовою продовження $k < n$, у якому значення k збільшується.

Проте обчислення можна скоротити. Якщо n є складеним, то $n = k * m$, де й k, й m більше одиниці, а менше з них обов'язково не більше $\sqrt{n}$. Отже, щоб дізнатися, чи є простим число n, достатньо перевірити його подільність лише на числа від 2 до $[\sqrt{n}]$ ($[x]$ позначає цілу частину x). Це дозволяє зменшити кількість виконань тіла циклу приблизно в $\sqrt{n}$ разів.

Якщо n ділиться на деяке k, то очевидно, що воно складене, і подальші перевірки не потрібні. Отже, умова продовження матиме вигляд **(k<=t) and (n mod k<>0)**, а тіло циклу міститиме лише збільшення **k**.

Оголосимо цілі змінні **t** і **k** для верхньої межі та поточного дільника й реалізуємо модифікований алгоритм. Особливий випадок значень n — 0 і 1, які не є простими; перебирати для них можливі дільники не потрібно.

```
function isPrime(n:longint):boolean;
{ визначити, чи є простим значення параметра }
  var k,           { поточний дільник }
      t : longint; { верхня межа }
Begin
  if n<2 then isPrime:=false
```

```
    else begin
           k:=2; {перший дільник 2}
           t:=round(sqrt(n));
       { sqrt(n) обчислюється в дійсному типі, і
         навіть для квадратів цілих чисел може
         виникнути похибка, тому для гарантії
         використовуємо не trunc, а round }
           while (k<=t)and(n mod k <> 0) do inc(k);
       { (k > t) або (n mod k = 0) }
       {якщо k>t, то число просте, інакше складене}
      isPrime := k > t
    end
End;
```

Як бачимо з цього прикладу, розробку підпрограми ***відокремлено*** від розробки програми, в якій ця підпрограма використовується. Цей принцип є дуже важливим у програмуванні.

Приклад 2. Увести два дроби, обчислити та вивести їх суму. Кожен дріб задають парою цілих чисел a і b, де $b \neq 0$. Результатом має бути нескоротний дріб, тобто дріб вигляду a/b, де $b > 0$ і НСД($|a|$, $|b|$) = 1.

Алгоритм розв'язання простий — увести два дроби, додати їх і надрукувати результат. У ньому легко виділити підзадачі «увести дріб», «вивести дріб», «додати два дроби». Реалізуємо розв'язання цих підзадач за допомогою підпрограм з такими заголовками (імена **num** і **den** є скороченнями від numerator — чисельник, denumerator — знаменник, **Fract** є скороченням fraction — дріб).

Процедура ***уведення*** дробу в пару параметрів. Параметри мають бути параметрами-змінними.

```
procedure readFract(var num, den : integer);
```

Процедура ***додавання*** дробів, заданих першими двома парами параметрів-значень, та збереження суми в третій парі параметрів-змінних.

```
procedure plusFract(num1,den1,num2,den2:integer;
                    var num3,den3:integer);
```

Процедура ***виведення*** дробу, заданого парою параметрів-значень.

```
procedure writeFract(num, den : integer);
```

Використаємо їх у такій очевидній програмі:

```
program SumFractions;
  ...підпрограми, які представлено нижче...
```

```
  var a1,b1, a2,b2, a3,b3 : integer; {три дроби}
begin
  writeln('Додавання двох дробів');
  write('Дріб (два цілих, друге не 0)>');
  readFract(a1,b1); { уведення першого дробу }
  write('Дріб (два цілих, друге не 0)>');
  readFract(a2,b2); { уведення другого дробу }
  plusFract(a1,b1, a2,b2, a3,b3);
  { a3/b3 = a1/b1 + a2/b2 }
  writeFract(a3,b3); { виведення результату }
end.
```

Після того, як зафіксовано заголовки та виклики підпрограм, можна переходити до їх розробки. Але спочатку звернемо увагу на те, що одне й те саме дробове число має безліч записів у вигляді дробів і лише один у вигляді ***нескоротного дробу з додатним знаменником***, зокрема, дріб з чисельником 0 подамо як 0/1. Тому дроби будемо подавати тільки як нескоротні.

Обробка дробів саме як нескоротних ставить ще одну підзадачу — «скоротити дріб». Її треба розв'язувати при уведенні та додаванні дробів. Для неї напишемо ***додаткову процедуру скорочення дробу***, заданого парою параметрів-змінних.

```
procedure reduceFract(var num, den : integer);
```

Почнемо розробку з процедур, які забезпечують уведення та виведення даних.

Процедура введення дробу. Тут треба лише ввести з клавіатури два цілих числа та скоротити дріб. Але якщо знаменник дорівнює 0, введення треба повторити.

```
procedure readFract(var num, den : integer);
Begin
  repeat
    read(num); readln(den);
    if den=0 then
      write('Знаменник 0. Задайте інший дріб: ')
  until den<>0;
  if den<0
  then begin
        { забезпечимо додатність знаменника }
```

```
        den:=-den; num:=-num;
      end;
  reduceFract(num,den) { скорочення дробу }
End;
```

Процедура виведення дробу. Надрукуємо дріб із символом «**/**» між чисельником і знаменником.

```
procedure writeFract(num, den : integer);
begin writeln(num, '/', den) end;
```

Процедура додавання. Найпростіший спосіб додати два дроби — реалізувати формулу $\frac{n_1}{d_1}+\frac{n_2}{d_2}=\frac{n_1d_2+n_2d_1}{d_1d_2}$ і скоротити цей дріб.

```
procedure plusFract(num1,den1, num2,den2 : integer;
                    var num3,den3 : integer);
  var m:integer;
begin
  den3:=den1*den2;
  num3:=num1*den2+num2*den1;
  reduceFract(num3,den3) { скорочення дробу }
end;
```

Допоміжна процедура скорочення дробу. Скоротити дріб — означає поділити його чисельник і знаменник на їх найбільший спільний дільник (НСД). Отже, маємо ще одну допоміжну підзадачу — «обчислити НСД двох натуральних чисел». Її розв'язання реалізуємо у вигляді функції GCD (це ім'я є скороченням від англійського *greatest common divisor* — найбільший спільний дільник).

```
procedure reduceFract(var num, den : integer);
  var t : integer;
begin
  if num=0 then den:=1 end
  else begin
      t:=GCD(abs(num),abs(den)); {обчислення НСД}
      num:=num div t;
      den:=den div t
  end
end;
```

Допоміжна функція обчислення НСД. Скористаємося сучасною версією алгоритму Евкліда.

```
function GCD(a, b : integer) : integer;
begin
  while (a<>0) and (b<>0) do
    if a>b then a:=a mod b
    else b:=b mod a;
  { a=0 або b=0 }
  if a=0 then GCD:=b
  else GCD:=a
end;
```

Розташування створених підпрограм у програмі. Підпрограму можна використовувати після її оголошення. Тому процедуру скорочення треба записати після функції обчислення НСД, а процедури введення та додавання — після процедури скорочення. Розташування процедури виведення дробу не має значення. Отже, у програмі **SumFractions**, наведеній вище, розташуємо підпрограми в такій послідовності:

```
program SumFractions;
function GCD ... див. вище ... end;
procedure reduceFract
... повний текст див. вище ...
end;
procedure plusFract
... повний текст див. вище ...
end;
procedure readFract
... повний текст див. вище ...
end;
procedure writeFract
... повний текст див. вище ...
end;
  var a1,b1, a2,b2, a3,b3 : integer; { три дроби }
Begin
  writeln('Додавання двох дробів');
  write('Дріб (два цілих, друге не 0)>');
  readFract(a1, b1); { уведення першого дробу }
  write('Дріб (два цілих, друге не 0)>');
  readFract(a2, b2); { уведення другого дробу }
```

```
    plusFract(a1,b1, a2,b2, a3,b3);
    { a3/b3 = a1/b1 + a2/b2 }
    writeFract(a3, b3); { виведення результату }
end.
```

Вдале розбиття задачі на підзадачі дозволяє отримати просту програму й декілька простих підпрограм, налагодити які набагато легше, ніж одну складну й громіздку програму.

Розбиття програми на підпрограми є одним із втілень принципу ***«розділяй і панує»***, який у програмуванні має дуже велике значення.

РЕКУРСИВНІ ПІДПРОГРАМИ

Підпрограми, які містять виклики самих себе, називаються ***рекурсивними***, а використання таких підпрограм — ***рекурсією***.

Приклад 1. Функція «факторіал» від натурального n позначається $n!$ і задається такими рівностями: $0! = 1$, $n! = n \cdot (n-1)!$ при $n > 0$. Згідно з ними, $1! = 1 \cdot 0! = 1$, $2! = 2 \cdot 1! = 2$, $3! = 3 \cdot 2! = 6$ тощо. Як бачимо, щоб обчислити значення функції для n, де $n > 0$, треба знайти її значення для $n-1$. У цьому ***зверненні*** до значення функції з ***меншим*** аргументом і полягає рекурсивність.

Безпосередньо за наведеним означенням дуже просто написати таку рекурсивну функцію (вважаємо, що $n! = 1$ при $n < 0$):

```
function f(n : byte) : longint;
begin
  if n<=0 then f:=1
  else f:=n*f(n-1) {f(n-1) – рекурсивний виклик}
end;
```

Виклик рекурсивної підпрограми виконується так само, як виклик будь-якої іншої підпрограми.

Розглянемо схематично виконання виклику `f(2)`. Спочатку виділяється локальна пам'ять для `n` і `f`, тобто ***програмний стек починає заповнюватися***. Значення аргументу `2` більше `0`, тому починається рекурсивний виклик `f(1)`. Знову виділяється локальна пам'ять для `n` і `f`, тобто ***у програмний стек додаються нові значення***. У цьому виклику значення аргументу `1` знову більше `0`, тому починається рекурсивний виклик `f(0)`, і ***зайнята частина програмного стека збільшується втретє***. У цьому третьому виклику значення аргументу `0` не більше `0`, тому виклик закінчується поверненням `1`. Програмний стек ***починає звільнятися***. Далі за-

кінчується рекурсивний виклик **f(1)**, і з нього повертається **1**. Програмний стек ***звільняється вдруге***. Нарешті, закінчується «зовнішній» виклик **f(2)**, повертається **2**, і програмний стек ***звільняється повністю***.

З кожним рекурсивним викликом зайнята частина програмного стека збільшується, а із закінченням виклику — зменшується. Оскільки розмір стеку обмежений, можлива ситуація (особливо при виконанні рекурсивних підпрограм), коли пам'яті в стеку не вистачить. Тоді програма аварійно завершується з повідомленням *Stack overflow*.

Приклад 2. На клавіатурі набираються цілі числа, не рівні нулю. Поява 0 означає кінець уведення. Прочитати числа та видати їх у зворотному порядку (кінцевий 0 не виводити).

Перше число треба вивести останнім, друге — передостаннім і так далі. Отже, для обробки послідовності чисел треба прочитати перше число і, якщо це не 0, ***так само*** обробити решту чисел і потім вивести перше число. А якщо прочитано 0, то нічого робити не треба. Звідси отримуємо рекурсивну процедуру **revertInput** у такій програмі:

```
program printNumbers;
procedure revertInput;
  var x : integer;
begin
  read(x);          { уведення числа }
  if x<>0 then begin
    revertInput;    { обробка решти чисел }
    write(x, ' '); { виведення числа }
  end
end;
Begin
  revertInput;
End.
```

Уведене число запам'ятовується в ***локальній*** змінній **x** процедури і з кожним викликом у програмному стеку з'являється новий екземпляр **x**. Після уведення 0 виклики закінчуються у порядку, зворотному до того порядку, в якому починалися. Звідси й значення локальних змінних **x** виводяться у зворотному порядку.

Приклад 3. Прочитати натуральне число (у звичайному десятковому записі) та основу системи числення p, $p \le 36$. Вивести p-ковий запис

числа. Значення цифр 10, 11, 12, …, 35 представити цифрами **A**, **B**, **C**, …, **Z**.

Значення молодшої цифри у *p*-ковому запису числа є остачею від ділення числа на *p*. Далі, узявши замість числа частку від його ділення на *p*, ***так само*** отримаємо значення наступної цифри, і так далі, поки не залишиться число менше *p*. Але цифру, ***отриману першою***, треба ***вивести останньою***, тому запам'ятаємо значення молодшої цифри, потім рекурсивно виведемо старші цифри, а потім виведемо молодшу цифру.

Можливим значенням цифр від 0 до 35 мають відповідати цифри — символи від **'0'** до **'9'** та від **'A'** до **'Z'**. Для отримання цифри за її значенням напишемо допоміжну функцію **digit**.

```
program numberOutput;
function digit(v : byte) : char;
begin  { повернення цифри за її значенням v }
  case v of
   0..9 : digit:=chr(v+ord('0'));
   10..35 : digit:=chr(v-10+ord('A'));
  end;
end;
procedure writeNP(n : longint; p : byte);
begin
  if n>=p then writeNP(n div p, p);
  { за n < p заглиблення в рекурсію немає }
  write(digit(n mod p));
end;
  var n : longint; p : byte; { число та основа }
Begin
  readln(n,p);
  writeNP(n,p);
end.
```

Для перевірки програми обов'язково задайте число 2147483647 (найбільше число типу **longint**) та основи 2, 8, 16, 32. Відповідними результатами мають бути **11**...**1** (31 «**1**»), **17777777777**, **7FFFFFFF**, **1VVVVVV**. Перевірте також декілька чисел з максимальною основою 36, наприклад, 35, 71 та 1295 (результатами мають бути **Z**, **1Z** та **ZZ**).

У рекурсивній підпрограмі обов'язково повинна бути ***умова***, за істинності якої відбувається повернення з виклику (див. приклади, наведені вище). Ця умова визначає «***дно рекурсії***», яке при виконанні підпрограми

обов'язково має досягатися, інакше виклики призведуть до переповнення програмного стека або інших непередбачуваних наслідків.

Приклад 4. Розглянемо процедуру, в якій умову повернення з рекурсії ***сформульовано невдало***.

```
procedure badProc(x : integer);
begin
  if x<>2 then badProc(x-2); { заглиблення та }
  write(x);                  { вихід з рекурсії }
end;
```

При виконанні виклику **badProc(6)** відбудуться рекурсивні виклики з аргументами **4** та **2**. При **x=2** заглиблення в рекурсію не буде, тому буде надруковано **2**, а потім виклики закінчаться з друкуванням **4** та **6**. Проте виклик **badProc(5)** призведе до рекурсивних викликів з аргументами **3**, **1**, **−1**, **−3**, … . Отже, умова повернення **x=2** ніколи не стане істинною, тому виконання рекурсивних викликів ***переповнить програмний стек***.

Приклад 5. Найбільший спільний дільник НСД(*a*, *b*) натуральних *a* і *b* можна обчислити рекурсивно на основі таких рівностей:

якщо *b* = 0, то НСД(*a*, *b*) = *a*,
якщо *a* mod *b* = 0, то НСД(*a*, *b*) = *b*,
якщо *a* mod *b* > 0, то НСД(*a*, *b*) = НСД(*b*, *a* mod *b*).

Наведемо відповідну рекурсивну функцію **GCD** (*Greatest Common Divisor* — Найбільший Спільний Дільник). Вважається, що значення аргументів у виклику невід'ємні.

```
function GCD(a, b : integer) : integer;
begin
  if b=0 then GCD:=a
  else if a mod b = 0 then GCD:=b
       else GCD:=GCD(b, a mod b)
end;
```

Як бачимо, «дно рекурсії» утворюють ситуації, в яких $b = 0$ або a кратне b, і тоді НСД обчислюється просто без рекурсії.

Приклад 6. Напишемо функцію, яка обчислює натуральний степінь дійсного числа a на основі такого алгоритму.

$$a^{2n} = (a^2)^n;\ a^{2n+1} = a \cdot (a^2)^n;\ a^0 = 1.$$

Цей алгоритим був відомий ще давнім індійцям, тому часто називається ***індійським***. Його основна мета — скоротити кількість множень. Наприклад, за цим алгоритмом $a^5 = a \times \left(a^2\right)^2$, тобто з трьома множеннями замість чотирьох: $a \cdot a \cdot a \cdot a \cdot a$. Одне множення буде зекономлено за рахунок того, що a^2 зберігається як проміжне значення і множиться само на себе. Так само $a^{10} = \left(a^5\right)^2$, що потребує лише чотирьох множень (три з них для обчислення a^5). Тут зберігається спочатку a^2, а потім a^5.

```
function pow(a:extended; n:longint):extended;
begin
  if n=0 then pow:=1.0
  else if odd(n) then
         pow:=a*pow(a*a,n div 2)
       else pow:=pow(a*a,n div 2)
end;
```

Проміжні множники зберігаються в локальній пам'яті викликів функції, точніше в змінних, які відповідають імені **pow**.

З'ясуємо, як максимальна кількість арифметичних дій залежить від *n*. У кожному наступному вкладеному виклику значення аргументу *n* менше попереднього значення принаймні вдвічі. При *n* = 0 відбувається повернення з виклику, тому кількість цих зменшень значення аргументу *n* не більше, ніж $\log_2 n$, тобто виклик з аргументом *n* породжує не більше ніж $\log_2 n$ рекурсивних викликів. При кожному виконанні виклику відбувається не більше одного ділення, піднесення до квадрата і множення, тому загальна кількість арифметичних операцій не більше $3 \cdot \log_2 n$. Наприклад, при *n* = 1000 це приблизно 30.

У цьому посібнику розглядається тільки так звана ***пряма*** рекурсія, коли підпрограма містить виклики самої себе. У програмуванні також використовується ***непряма*** рекурсія, коли підпрограми містять взаємні виклики.

Приклад 7. На дошці — три стрижні: 1, 2, 3. На першому розміщено вежу з *n* дисків; нижній диск має найбільший діаметр, а діаметр кожного наступного менший за діаметр попереднього. За один хід із будь-якого стрижня можна взяти верхній диск і перемістити на інший стрижень, але дозволено класти диск лише на дошку або на диск більшого діаметра. Треба перемістити усю вежу зі стрижня 1 на стрижень 3.

Ця гра називається «Ханойські вежі». За легендою, цю гру почали понад тисячу років тому ченці в одному монастирі поблизу Ханою у В'єтнамі; вони мали $n = 64$ диски. Коли вони закінчать гру, настане кінець світу. Розв'язком цієї гри-задачі є послідовність перенесень дисків. Напишемо програму друкування позначень цих переносів.

Для перенесення вежі з n дисків зі стрижня 1 на стрижень 3 необхідно перенести вежу з $n-1$ диска на стрижень 2, потім перенести нижній диск на стрижень 3 і вежу з 2 на 3. При перенесенні вежі з 1 на 2 допоміжним буде стрижень 3, а при перенесенні з 2 на 3 — 1, причому ніяка інша послідовність дій неможлива. Отже, розв'язання задачі для вежі висотою n описано за допомогою розв'язання для вежі висотою n–1, тобто рекурсивно.

Нехай **disk(a,b)** позначає перенос одного диска зі стрижня **a** на стрижень **b**, а **tower(h,a,b,c)** — перенесення вежі висотою **h** з **a** на **b** з використанням стрижня **c** як допоміжного. При **h** > 1 виконання **tower(h,a,b,c)** буде таким.

```
tower(h-1,a,c,b);
disk(a,b);
tower(h-1,c,b,a)
```

При **h** = 1 переносять тільки один диск, тобто виконують **disk(a,b)**. Отже, очевидною є така програма.

```
program HanoiTowers;
procedure disk(diskFrom, diskTo : integer);
begin writeln(diskFrom, '->', diskTo) end;
procedure tower(H:byte; F,T,V:byte);
{Height – висота, From - з, To - на, Via – через}
begin
  if H=1 then disk(F,T)
  else begin
         tower(H-1,F,V,T);
         disk(F,T);
         tower(H-1,V,T,F)
       end
end;
var n : byte;
begin
  writeln('Імітація гри «Ханойські вежі»');
  write('Натуральна кількість дисків>');
  readln(n);
```

```
  tower(n,1,3,2)
end.
```

Визначимо кількість перенесень дисків у вигляді функції $s(n)$, де n — висота вежі. Очевидно, що $s(1) = 1$ і $s(n) = 2 \cdot s(n–1)+1$. Як бачимо, $s(2) = 3$, $s(3) = 7$, $s(4) = 15$ тощо. Кожне наступне значення подвоюється, тому неважко переконатися, що $s(n) = 2^n - 1$. Значення $s(64)$ приблизно дорівнює 10^{19}. Якщо припустити, що ченці переносять один диск щосекунди, то для перенесення такої вежі потрібно більше 10^{12} років! У ченців ще багато роботи… .

За рекурсивною підпрограмою без жодного оператора циклу можуть легко ховатися величезні обсяги обчислень. Як і будь-який потужний засіб, рекурсія вимагає обережності у використанні.

8. ГЛИБИНА РЕКУРСІЇ ТА ЗАГАЛЬНА КІЛЬКІСТЬ РЕКУРСИВНИХ ВИКЛИКІВ

З рекурсивними підпрограмами пов'язано два важливих поняття — глибина рекурсії та загальна кількість викликів, породжених викликом рекурсивної підпрограми.

Глибина рекурсії виклику підпрограми — це кількість рекурсивних викликів, розпочатих і не закінчених у момент початку цього виклику.

Наприклад, при обчисленні $n!$ виклик з аргументом n починається першим, тому має глибину 0. Рекурсивний виклик з аргументом $n – 1$ має глибину 1, з аргументом $n – 2$ — глибину 2 тощо. Найбільшу глибину n має виклик з аргументом 0. Аналогічно в задачі «Ханойські вежі» виклик процедури **`tower`** з висотою h має глибину рекурсії $n – h$.

Коли виконується виклик підпрограми з глибиною рекурсії m, одночасно «існує» $m + 1$ екземпляр локальної пам'яті підпрограми. Кожен екземпляр займає ділянку певного розміру, тому збільшення глибини, як було сказано вище, може призвести до ***переповнення програмного стеку***.

Загальна кількість рекурсивних викликів, породжених викликом рекурсивної підпрограми, — це кількість викликів, виконаних між початком і завершенням цього виклику.

Наприклад, у задачі «Ханойські вежі» кожен виклик задає одне перенесення диску та, окрім найбільш заглибленого, породжує два рекурсивних

виклики. Нам уже відомо, що перенесення вежі висотою *n* вимагає $2^n - 1$ перенесень дисків, тому між початком і закінченням виклику з аргументом *n* виконується $2^n - 2$ рекурсивних викликів.

Загальна кількість вкладених викликів визначає тривалість виконання виклику. Як бачимо, кількість викликів у задачі «Ханойські вежі» за значень *n* порядку кількох десятків призводить до неприпустимо довгого виконання програми.

Використання рекурсивних підпрограм потребує уміння оцінити можливу глибину рекурсії, розмір пам'яті виклику підпрограми та загальну кількість рекурсивних викликів.

Приклад. Повернемося до чисел Фібоначчі, які визначаються рівностями $f_1 = f_2 = 1$, $f_n = f_{n-2} + f_{n-1}$ за $n > 2$. Безпосередньо за цими рівностями дуже просто написати таку функцію визначення числа Фібоначчі за його номером (вважається, що додатність *n* гарантовано).

```
function f(n:integer):longint;
begin
  if (n=1)or(n=2) then f:=1
  else f:=f(n-2)+f(n-1) {два рекурсивних виклики}
end;
```

ЦЕ ДУЖЕ ПОГАНА ФУНКЦІЯ!!! Її головний недолік: вона ***породжує абсолютно не потрібні багаторазові обчислення тих самих величин***. Розглянемо, наприклад, обчислення **f(6)**. Очевидно, що виклик **f(4)** виконується двічі, **f(3)** — тричі, **f(2)** — п'ять разів, **f(1)** — тричі.

У загальному ж випадку за $n > 3$ обчислення f_n породжує два виклики **f(n-2)**, три виклики **f(n-3)**, п'ять викликів **f(n-4)**, …, f_{n-1} викликів **f(2)** та f_{n-2} викликів **f(1)**. Для виконання всіх цих дій потрібно часу не менше ніж для перенесення Ханойської вежі висотою *n*. Але, на відміну від задачі про вежі, усі ці дії не є обов'язковими, адже циклічний алгоритм у главі 5 вимагає лише одного додавання та двох присвоювань на кожне з чисел $f_3, f_4, ..., f_n$ (плюс присвоювання, додавання та порівняння номерів).

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке підпрограма?

2. Які типи підпрограм ви знаєте? Чим вони відрізняються?

3. Яким елементом програми є заголовок підпрограми?

4. Чим відрізняються заголовки функцій та процедур?

5. Яким елементом програми є виклик процедури, а яким – виклик функції?

6. Як ви вважаєте, процедури **`readln`** *і* **`writeln`** *мають параметри-значення чи параметри-змінні?*

7. Що таке формальні та фактичні параметри (параметри та аргументи)?

8. Чим відрізняється підстановка аргументів на місце параметрів-значень та на місце параметрів-змінних?

9. Чому пам'ять, утворену локальними змінними викликів підпрограм, називають програмним стеком?

10. За якої умови у підпрограмі можна використовувати імена, не оголошені в ній?

11. Яке ім'я називається у підпрограмі локальним, а яке — глобальним?

12. Чи можна використовувати локальне ім'я підпрограми в інших підпрограмах, записаних після цієї підпрограми?

13. Яка підпрограма називається рекурсивною?

14. Що таке глибина рекурсії? На яку характеристику програми вона впливає?

15. Що таке загальна кількість рекурсивних викликів, породжених даним викликом? На яку характеристику програми вона впливає?

ЗАДАЧІ

1. Написати функцію визначення максимального зі значень двох її цілих параметрів.

2. Написати функцію **`even`**, тобто «парне», яка повертає ознаку парності свого цілого параметра.

3. Написати функцію з дійсним параметром x, яка обчислює:

а) –1, 0, 1, відповідно, при $x < 0, x = 0, x > 0$;

б) найменше ціле, що не менше за значення параметра.

4. Проімітувати виконання такої програми:

а)
```
var a,b:integer;
  function f(x:byte; var y:byte):byte;
  begin b:=b+1; y:=a+b; f:=y end;
   begin
    for a:=2 to 4 do begin
      b:=a; writeln(f(a,b)); writeln(b)
```

```
   end;
   writeln(a)
 end.
```

b)
```
var a,b:byte;
  function f(var x:byte; y:byte):byte;
  begin b:=b+1; y:=a+b; f:=y end;
 begin
   for a:=2 to 4 do begin
      b:=a; writeln(f(a,b)); writeln(b)
   end;
  writeln(a)
end.
```

c)
```
function f(a:byte):byte;
  begin a:=a+1; f:=a end;
  function g(var x:byte):byte;
  begin x:=2; g:=x end;
  var a,b:byte;
  begin
    a:=3; b:=4;
    writeln(f(a)); writeln(g(b));
    writeln(a,b)
  end.
```

d)
```
var a,b:byte;
  function f(var a : byte) : byte;
  begin a:=a+4; f:=a end;
  function g(x:byte):byte;
  begin x:=3; g:=x end;
  begin
    a:=2; b:=1;
    writeln(f(a)); writeln(g(b));
    writeln(a,b)
  end.
```

e)
```
var a,b,c:byte;
  function f(x:byte; var y:byte):byte;
```

```
  begin
    x:=x+1; y:=y+2; c:=a+b; f:=x*y
  end;
  begin
     a:=1; b:=1;
      writeln(f(a,b)); writeln(a,b,c)
  end.
```

f)
```
  var a,b,c:byte;
    function f(var x:byte; y:byte):byte;
    begin
      x:=x+1; y:=y+2; c:=a+b; f:=x+y
    end;
begin
     a:=2; b:=2;
     writeln(f(a,b)); writeln(a,b,c)
end.
```

5. Написати підпрограму, яка за трьома дійсними додатними числами, що задають довжини відрізків, визначає, чи можна з них утворити трикутник, і якщо так, повертає периметр, площу, радіуси вписаного та описаного кіл (якщо не можна, повертає нулі). Написати програму, яка вводить трійки чисел, визначає, чи можна з них утворити трикутники, та обчислює вказані вище параметри цих трикутників. Ознака кінця введення — хоча б одне з трьох уведених чисел дорівнює 0.

6. Написати програму друкування всіх «близнюків», тобто пар простих чисел вигляду $6m - 1$ і $6m + 1$ при $m > 0$, наприклад, 5 і 7, 11 і 13, 17 і 19 (але не 23 і 25, оскільки 25 — складене число). Обмежитися числами типу **integer** (**longint**).

7. Написати процедуру друкування розкладу значення її натурального параметра на прості множники (див. приклад з розділу 5) так, щоб між множниками ставився знак ***** і кожен з них друкувався один раз, але зі знаком **^** і показником степеня, якщо той більше 1. Наприклад, 169 = 13^2, 144 = 2^4*3^2, 50 = 2*5^2.

8. Написати функцію, яка знаходить найменше спільне кратне двох цілих чисел.

9. Вказати, що буде виведено при виконанні програми, якщо на вході задати:

a) 10; b) 7; c) 5.

```
program P;
  var x:integer;
procedure S(N:integer);
begin
write(N,' ');
  if N>7 then S(N-1);
  write(N,' ');
end;
Begin
writeln('ціле число: '); readln(x);
  S(x);
End.
```

10. Написати рекурсивну функцію обчислення біноміального коефіцієнта $C_n^k = \frac{n!}{k!\cdot(n-k)!}$, при виконанні якої виконувалося б не більше $k/2$ рекурсивних викликів.

Науково-виробниче видання

Бібліотека «Шкільного світу»

***Ставровський** Андрій, **Скляр** Ірина*

Програмуємо правильно
Посібник
Частина 1

Художній редактор *О. Голик*
Літературний редактор *Ю. Желєзна*
Коректор *І. Бірюкович*
Комп'ютерна верстка *К. Яскевич*
Набір *Т. Корольова*

Підписано до друку 06.06.07. Формат 60x84/16.
Папір офсетний № 1. Гарнітура Таймс. Друк офсетний.
Умовн. друк. арк. 7,44. Обл.-вид. арк. 7,2. Тираж 1300 пр.
Зам. №

ТОВ Видавництво «Шкільний світ»
01014, м.Київ, вул. Тимірязєвська, 2
Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції
серія ДК № 775 від 21.01.2002 р.

Видрукувано з готових діапозитивів в ОП «Житомирська облдрукарня»
10014, Житомир, вул. Мала Бердичівська, 17
Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції
серія ЖТ № 1 від 06.04.2001 р.