**Паскаль: типи дійсних, оператори розгалуження, функції та їх виклики, процедури, під задачі.**

**Тип дійсних**

Дійсні числа позначаються ***дійсними* *сталими*.** Розглянемо приклад. Число **1.2345** можна позначити багатьма різними способами, наприклад, **123.45 10-2**. Тут воно має цілу частину **123**, дробову частину **.45** і десятковий порядок **-2**. Цьому запису відповідає стала мови Паскаль **123.45E-2**, у якій **123** **–** ціла частина, **.45** **–** дробова, а **E-2** **–** порядок. Це ж число можна задати сталою **0.12345E1** або **0.012345E+2**, або **1.2345**, або **12345e-04**. Подання числа сталою, у якій перед десятковою крапкою записано єдину цифру від 1 до 9, називається ***нормалізованим***, наприклад, 9.81 або 1.0E2 (число 0 має нормалізоване подання 0.0).

Дійсні сталі мають обов'язкову ***цілу*** ***частину***, за якою записано ***дробову*** ***частину*** і ***порядок*** (можливо, одне з них). Ціла частина – це непорожня послідовність цифр, дробова – непорожня послідовність цифр із крапкою на початку, а порядок **–** буква "E" або "e", можливо, із знаком "+" або "-", і однією або двома цифрами. Перед сталою може бути знак "-", і тоді вона задає від'ємне число: -12.345E-1.

Не уточнюючи множину представних дійсних чисел, скажемо лише, що вона:

* *є скінченною обмеженою підмножиною множини раціональних чисел;*
* *містить усі цілі числа, представні в типі integer (і багато інших, але все одно їх скінченна множина!)*.

Як бачимо, цілі числа задаються як цілими сталими, так і дійсними, наприклад, **2** і **2.0**. Проте їм відповідають *два цілком різних подання того самого числа, тобто значення двох різних типів*. І в машині вони обробляються по-різному.

До дійсних значень застосовні ті ж самі арифметичні операції, що й до цілих, за винятком odd, **div**, **mod** і деяких інших, про що ми скажемо в розділі 10. Їх можна порівнювати (=, <>, > тощо), і до них, і лише до них, застосовні дві операції round і trunc. Вони задаються у вигляді викликів функцій: round(3.62), trunc(2.71) тощо. Перша породжує ціле значення, найближче до операнда, наприклад, round(4.12)=4, round(3.62)=4, а друга – значення математичної функції "ціла частина", що позначається [*x*]: trunc(3.62)=3. Останнє твердження, утім, є не зовсім точним, тому що для від'ємного числа *x* значенням trunc(x) є не [*x*], а -[-x]: trunc(-3.14)=-3, хоча в математиці [-3.14]=-4.

За числовим значенням *x*, цілим або дійсним, можна обчислити дійсне значення "математичної функції"

|*x*|, , sin*x*, cos*x*, arctg*x*, e*x*, ln*x*.

Вираз із числовим значенням записується як аргумент у виклику функції з ім'ям відповідно

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

наприклад,

abs(-2), sqrt(1-sin(x)), arctan(sin(1)/cos(1)), exp(ln(x)).

Значення аргументу у викликах тригонометричних функцій виражає кількість радіан, а не градусів. Крім того, виклик функції sqr(x) за дійсним значенням *x* породжує дійсне значення *x2*, а за цілим – ціле.

У системі Турбо Паскаль означено також нульмісну функцію Pi (її значенням є число, близьке до числа  ) й одномісні функції Frac і Int, застосовні лише до дійсних. Вони задають обчислення дробової частини й дійсного подання цілої частини свого аргументу. Наприклад, sin(pi/2)=1.0, frac(3.1415)=0.1415, int(3.1415)=3.0.

Дійсні значення й операції, застосовні до них, утворюють ***тип*** ***дійсних*** з ім'ям real.

**Задачі**

**1.**\* Указати нормалізоване подання дійсних чисел:а) 99999; б) 0.00001

**2.** Написати вираз мови Паскаль, що відповідає математичному:

а)\* *ab*; в)\* arcsin*x*;

б)\* ; г)\* arcctg*x*;

д) [*x*] для будь-якого дійсного *x* (додатного чи від'ємного);

е)\* 2 /3 (без використання Pi або сталої, схожої на 3.1415926).

**3.** а) Написати вираз, що задає обчислення відстані між двома точками площини за їх координатами;

б)\* написати оператори, що задають обчислення відстані від точки до кола в площині (точка задана координатами, коло – координатами центру й радіусом; якщо точка в колі, то відстань 0).

**3.4.**\* Які з перерахованих вище операцій над дійсними усюди визначені, а які – ні?

**1.2. Поліморфізм**

З означення типів цілих і дійсних чисел очевидно, що і до тих, і до інших застосовні ті самі операції: +, -, /, порівняння та інші. Але насправді ті самі знаки позначають різні операції! Наприклад, цілі додаються або порівнюються зовсім інакше, ніж дійсні.

У програмуванні властивість операції бути означеною для різних типів називається ***поліморфізмом***, а сама операція – ***поліморфною***. За знаком операції та типами виразів, що позначають операнди в Паскаль-програмі, можна визначити, яку саме операцію слід указати в машинній програмі. І це визначається під час трансляції Паскаль-програми (або при обчисленні виразу в процесі її інтепретації).

Слово "поліморфізм" буквально означає "багатоформність", тобто наявність багатьох форм у того самого змісту. У даному випадку та сама за змістом операція, наприклад, додавання, має різні машинні форми для різних типів.

**1.3. Сумісність цілих і дійсних**

Мова Паскаль допускає різнотипні числові, тобто цілі й дійсні, операнди у виразах, наприклад, 2+1.0. При трансляції таких виразів додаються команди породження дійсного значення за цілим операндом. Отже, при обчисленні виразу насправді *спочатку виконується перетворення цілого операнда в дійсний і потім указана операція над дійсними значеннями.* Так, при обчисленні 2+1.0 спочатку 2 перетворюється в 2.0 і потім додаються 2.0 і 1.0.

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

Є ще один вид сумісності – ***сумісність*** ***за*** ***присвоюванням***, коли значення одного типу можна присвоювати змінним іншого. *Дійсний тип сумісний за присвоюванням з цілим, але не навпаки*. Наприклад, якщо a:real; b:integer, то можна написати a:=b, але не можна b:=a. Аналогічно до обчислення виразів, ціле значення перед присвоюванням перетвориться в дійсне. З цієї ж причини, до речі, при виконанні readln(z) із змінною z:real можна набрати на клавіатурі не дійсну, а цілу сталу – z одержить дійсне значення. Зворотні перетворення програміст повинен задавати явно за допомогою функцій trunc або round, наприклад, b:=round(a).

**Задача**

**3.6.** Намалюйте три кола, відзначених іменами типів цілих, дійсних і бульових. Проведіть стрілки між ними – стрілка веде від кола А до кола Б, якщо означено операції з операндами типу А и значеннями типу Б, наприклад, від кола integer до кола boolean. Позначте стрілки знаками відповідних операцій. Назвіть поліморфні й неполіморфні операції.

**2. Комп'ютер сам вирішить,**

**що робити і чого не робити**

**2.1. Оператори розгалуження та складений**

Майже кожний, хто провчився в школі років вісім, пам'ятає, як обчислювати дійсні корені квадратного рівняння *ax2*+*bx*+*c*=0 (природно, за умови a 0):

(1) *прочитати коефіцієнти a, b, c;*

(2) *обчислити d=b2-4ac;*

(3) *якщо d>0, то обчислити x1=(-b- )/(2a), x2=(-b+ )/(2a);*

*у противному випадку*

*якщо d=0, то обчислити x1=-b/(2a),*

*інакше нічого не робити.*

Майже кожний розуміє, що він задає три різні послідовності дій. Яка саме виконується, залежить від конкретних значень *a*, *b*, *c*. Пункт (3) алгоритму задає перевірку, яка з умов *d*>0, *d*=0 або *d>*0 справджується, і залежно від цього ті або інші дії.

***Умову*** будемо розуміти як фразу, що може бути або істинною, або хибною. У мові Паскаль *умову можна відтворити бульовим виразом*, як правило, із змінними. Його значеннями можуть бути **true** або **false** – це залежить від значень змінних. Звичайно, умови можуть бути тотожно істинними або тотожно хибними – вони відтворюються виразами, швидше за все, без змінних. Втім, вирази z **or** **not** z і z **and** **not** z мають значення відповідно **true** і **false** незалежно від значення z.

*Перевірка умови при виконанні програми – це обчислення відповідного бульового виразу.*

Перевірка умов і виконання залежно від цього різних дій задається в мові Паскаль ***операторами розгалуження***. Вони мають дві форми **–** повну та скорочену. ***Оператор розгалуження в* *повній* *формі*** має вигляд:

**if** *умова* **then** *оператор* **else** *оператор*

Ключові слова **if**, **then**, **else** **–** це англійські "якщо", "то", "інакше". Для полегшення читаності програми оператор розгалуження часто записують "східцями":

**if** *умова*

**then**

*оператор*

**else**

*оператор*

або

**if** *умова* **then**

*оператор*

**else** *оператор*

Виконання його полягає в тім, що спочатку обчислюється значення умови, записаної після слова **if**. Далі, якщо цим значенням є **true**, виконується оператор, записаний після слова **then**, і на цьому виконання закінчується. Але якщо це значення хибне, те виконується не перший, а другий оператор, записаний після **else**. Наприклад, при виконанні послідовності операторів

readln(x);

**if** x>=0 **then** z := 1 **else** z := -1

змінна z одержить значення 1, якщо прочитано невід'ємне значення x. Якщо ж прочитано значення від'ємне, то z одержить значення –1.

Оператор розгалуження в ***скороченій* *формі*** має вигляд:

**if** *умова* **then** *оператор*

Він відрізняється лише тим, що якщо обчислення умови дає значення **false**, то на цьому його виконання закінчується.

Як бачимо, оператори розгалуження містять умови, з обчислення яких і починається їх виконання. Тому ці оператори ще називаються ***умовними***.

Застосуємо оператори розгалуження для перекладу алгоритму обчислення коренів на мову Паскаль. Пункт (3) можна, здавалося б, перекласти так:

**if** d>0 **then** x1:=(-b- sqrt(d))/(2\*a); x2:=(-b+sqrt(d))/(2\*a)

**else**

**if** d=0 **then** x1:=-b/(2\*a);

{інакше нічого не робити}

*Але це неправильно!* Оператор розгалуження закінчується оператором присвоювання змінній x1. Оператор x2 := (-b+sqrt(d))/(2\*a) записано уже за роздільником ";", тобто після оператора розгалуження. Те, що написано далі, взагалі не є оператором.

Як же записати *послідовність із двох або більше операторів там, де має бути один*? Напрошується відповідь, що їх треба взяти в дужки. І такі дужки, що перетворюють послідовність операторів у один оператор, у мові Паскаль є. Це так звані відкриваюча та закриваюча операторні дужки: ключові слова **begin** і **end** (початок і кінець).

Запис вигляду

**begin** *послідовність операторів* **end**

називається ***складеним*** ***оператором***.

Отже, опишемо обчислення одного або двох коренів таким оператором розгалуження в повній формі:

**if** d>0 **then**

**begin** x1:=(-b+sqrt(d))/(2\*a); x2:=(-b-sqrt(d))/(2\*a) **end**

**else**

**if** d=0 **then** x1:=-b/(2\*a)

Як бачимо, після слова **then** записано складений, а після слова **else** – оператор розгалуження в скороченій формі.

Оформимо алгоритм обчислення коренів у вигляді програми:

**program** roots(input, output);

**var** a, b, c: real; x1, x2: real;

**begin**

{1} readln(a,b,c); {припускаємо, що a<>0! }

{2} d:=b\*b-4\*a\*c;

{3} **if** d>0 **then**

**begin**

x1:=(-b+sqrt(d))/(2\*a);

x2:=(-b-sqrt(d))/(2\*a)

**end**

**else**

**if** d=0 **then** x1:=-b/(2\*a)

**end**.

Якщо при виконанні цієї програми задати значення змінних a, b, c, наприклад, відповідно 1, 3, 2, то справджується d>0, і обчислюються x1 і x2. Якщо задати значення 1, 2, 3, то умова d>0 хибна, обчислюється умова d=0, її значенням є **false**, і на цьому все закінчується. При значеннях 1, 2, 1 умова d=0 істинна, і обчислюється лише x1.

До програми слід додати оператори виведення, щоб вона не була занадто "мовчазною". Це залишається як вправа.

І останнє зауваження щодо структури операторів розгалуження. Розглянемо такий оператор:

**if** z>0 **then if** z>5 **then** k:=2 **else** k:=1

Хибності якої умови, z>0 чи z>5, відповідає **else**-гілка? Тобто чи є оператор

**if** z>5 **then** k:=2

оператором розгалуження в скороченій формі, чи він має повну форму

**if** z>5 **then** k:=2 **else** k:=1 ?

Відповідь на це питання дає наступне неформальне правило.

*Будемо рухатися по тексту програми від слова* ***else*** *назад до найближчого слова* ***if****,**пропускаючи при цьому складені оператори. Цьому слову* ***if*** *та хибності умови, записаної за ним, і відповідає* ***else****-гілка. Але якщо на шляху ми зустріли слово* ***else****, то за цим самим правилом спочатку відшукаємо відповідне йому* ***if****, і лише після цього продовжимо наши пошуки.*

За цим правилом у останньому прикладі **else**-гілка k:=1 відповідає хибності умови z>5, а не z>0. В операторі

**if** z>0 **then**

**begin** readln(x); **if** x=0 **then** k:=1 **end**

**else** k:=5

**else**-гілка k:=5 відповідає хибності умови z>0, а не умови x=0, пропущеної у складеному операторі. За цим самим правилом у операторі

**if** x>0 **then** {квадранти перший або четвертий}

**if** y>0 **then** k:=1 **else** k:=4

**else** {квадранти другий або третій}

**if** y>0 **then** k:=2 **else** k:=3

гілка з початком "**else if** y>0" відповідає хибності умови x>0, а хибності першої умови y>0 відповідає гілка " **else** k:=4".

**2.2. Масовість задач і програм**

При виконанні оператора розгалуження, булів вираз у якому не тотожно істинний або хибний, можливі принаймні два різних процеси обчислень. Який із них здійснюється, залежить від значень змінних, що входять в умову, тобто від стану пам'яті програми. Таким чином, оператор розгалуження задає різні дії, що їх має виконати комп'ютер при різних станах пам'яті.

Різні стани пам'яті після виконання тих самих операторів програми можуть утворюватися, якщо її змінним "присвоюються з зовнішнього світу" різні набори значень. Отже, з використанням оператора розгалуження можна описати розв'язання задачі для різних наборів значень, що надходять "із зовнішнього світу" (***вхідних* *значень***, або ***вхідних*** ***даних***).

Програми, як правило, пишуться для того, щоб перекласти на комп'ютер розв'язання задач, які людина не хоче або не може розв'язати сама. Звичайно *задача ставиться в загальному вигляді з указанням параметрів*, від значень яких залежить хід і результат розв'язання, наприклад, "розв'язати квадратне рівняння *ax*2+*bx*+*c*=0, задане коефіцієнтами *a*, *b*, *c*". Параметри тут – коефіцієнти рівняння.

Задачі, що ставляться в загальному вигляді з параметрами, називаються ***масовими***. Задача, поставлена не в загальному вигляді, а з конкретним набором значень параметрів, називається ***екземпляром*** ***задачі***. Наприклад, "розв'язати рівняння *x*2+3x+2=0, задане коефіцієнтами 1, 3, 2".

*Всі можливі конкретні набори значень параметрів утворюють екземпляри задачі й задають конкретні процеси її розв'язання.*

Алгоритм розв'язання масової задачі теж повинний бути ***масовим***, тобто таким, що за ним можна здійснити процеси розв'язання всіх екземплярів задачі. Наприклад, розв'язати всі можливі конкретні квадратні рівняння. Таким чином, *програми, як правило, мають властивість масовості*. І оператор розгалуження **–** це один із засобів, яким масовість забезпечується.

**2.3. Блок-схеми**

Процеси, задані оператором розгалуження **if** *умова* **then** *оператор* **else** *оператор*, можна зобразити як гілки одного процесу, на які він розділяється. Позначимо обчислення умови ромбом, із якого виходять два стрілки, позначені можливими значеннями умови **true** і **false**. Стрілки задають послідовність дій. Позначимо виконання оператора прямокутником; рис.3.1 виражає "розгалуження" процесу виконання оператора розгалуження на *два можливих процеси*, хоча при будь-якому його виконанні здійснюється в точності один із них.

Зображення, складені з прямокутників, ромбів указаного вигляду й стрілок, називаються ***блок-схемами***. Одна зі стрілок звичайно починається з "нізвідки" і позначає початок блок-схеми. Якщо рухатися по стрілках і виписувати дії, позначені в блоках (ромбах і прямокутниках), утворяться позначення процесів, що задаються блок-схемою. Отже, блок-схема – це теж алгоритм, тільки виражений в іншій формі. Такого, нехай не зовсім точного, тлумачення блок-схем нам буде достатньо, оскільки ми скористаємося ними лише для ілюстрації семантики окремих операторів.

Пункт (3) алгоритму обчислення коренів квадратного рівняння за його коефіцієнтами можна задати блок-схемою з рис. 3.2.

У деяких випадках блок-схеми дуже наочно подають можливі процеси виконання програми. На зорі програмування вони використовувалися дуже широко, і перед написанням програм навіть необхідно було креслити блок-схеми. Тепер можна обходитися і без них.

**Задачі**

**3.7.**\* Імітувати виконання операторів, де x, y **–** імена змінних цілого типу:

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),

якщо при читанні x одержує значення:

а) 1; б) 2;

в) 3; г) 4.

**3.8.** Написати програму обчислення та друкування дійсних коренів квадратного рівняння, заданого коефіцієнтами *a*, *b*, *c*,

а) де *a* 0; б) де, можливо, *a*=0.

**3.9.**\* Написати програму дослідження, тобто обчислення кількості коренів рівняння *ax*2+*bx*+*c*=0 за його коефіцієнтами *a*, *b*, *c* (можливо, *a*=0).

**3.10.** Написати програму дослідження вигляду множини розв'язань нерівності *ax*2+*bx*+*c*>0 (два інтервали, інтервал і т.п.).

**3.11.** Зобразити аналогічно рис.3.2 алгоритми розв'язання задач 3.8–3.10.

**3.12.** Написати програму визначення виду трикутника за трьома довжинами його сторін (можна припускати, що вони додатні й задовольняють нерівності трикутника):

а)\* рівносторонній, рівнобедрений і не рівносторонній, різнобічний;

б) гострокутний, прямокутний, тупокутний.

**3. Функція та її виклики**

*Status in statu.*

(лат.: *Держава в державі*)

Розглянемо задачу: обчислити мінімальну з відстаней між точками площини *A*(*x1*; *y1*), *B*(*x2*; *y2*) і *C*(1;2). Алгоритм розв'язання цієї задачі очевидний:

1) обчислити відстані *d1*=*AB*, *d2*=*AC*, *d3*=*BC*;

2) обчислити *m*= min{*d1*, *d2*, *d3*}.

Відстань між точками з довільними координатами (*x*; *y*), (*x'*; *y'*) виражається формулою *d*=, і для обчислення відстаней нам необхідно тричі написати "Паскалівський" варіант цієї формули з різними наборами координат: *x1*, *y1*, *x2*, *y2*, потім *x1*, *y1*, 1, 2, потім *x2*, *y2*, 1, 2. Ці вирази досить громіздкі й задають по суті ті самі обчислення, тільки з різними наборами значень. Все це можна записати інакше.

*Мова Паскаль дозволяє описати повторювані обчислення один раз, дати цьому опису ім'я і далі не описувати самі обчислення, а тільки позначати їх цим ім'ям.*

Отже, у мові Паскаль є *описи* обчислень і є їх *позначення*. Опис обчислень, як правило, є ***параметризованим***, подібно до алгоритму обчислення коренів квадратного рівняння, де параметрами були коефіцієнти рівняння. Конкретні значення, з якими треба зробити обчислення, вказуються в позначенні обчислень разом із ім'ям цього опису й називаються ***аргументами***. Опис обчислень деякого значення називається ***функцією***, а їх позначення – ***викликом* *функції*.**

У даному випадку параметрами будуть чотири координати двох точок. Назвемо їх a1, b1, a2, b2. Опис обчислень задається у вигляді функції, якій ми дамо ім'я dd:

**function** dd(a1, b1, a2, b2: real):real;

**begin**

dd:=sqrt( sqr(a1-a2)+sqr(b1-b2) )

**end;**

*Цей опис є означенням імені dd*, тому поміщається серед інших означень програми. Позначення цієї функції, тобто виклики її з конкретними аргументами записуються в тілі програми:

**program** minimdis(input, output);

**var** x1, y1, x2, y2, d1, d2, d3, m : real;

**function** dd(a1, b1, a2, b2: real):real;

**begin**

dd:=sqrt( sqr(a1-a2)+sqr(b1-b2) )

**end;**

**begin**

writeln('введіть координати двох точок:');

readln(x1, y1, x2, y2);

d1:=dd(x1, y1, x2, y2);

d2:=dd(x1, y1, 1, 2);

d3:=dd(x2, y2, 1, 2);

**if** d1<d2 **then** m:=d1 **else** m:=d2;

**if** d3<m **then** m:=d3;

writeln('найменша відстань: ', m)

**end.**

При виконанні цієї програми після читання значень змінних виконується виклик функції dd: значення змінних x1, y1, x2, y2 присвоюються відповідним параметрам a1, b1, a2, b2 *як звичайним змінним* і потім обчислюється значення dd. Воно і є значенням виразу dd(x1, y1, x2, y2), що присвоюється змінній d1.

Так само, тільки з іншими аргументами, виконуються другий і третій виклики функції, і інші значення присвоюються змінним d2 і d3.

Отже, ми бачимо, що мова Паскаль дозволяє не тільки користуватися викликами "стандартних" функцій, наприклад, odd або sin, але й створювати свої власні.

Функція має такий загальний вигляд:

**function** *ім'я*(*означення параметрів*) : *ім'я типу*;

*означення*

**begin**

*послідовність* *операторів*

**end;**

У першому рядку функції записано ***заголовок***, де вказано її ім'я й ***означення* *параметрів***. Наприкінці заголовка обов'язково записується ім'я типу значень, що обчислюються в результаті виконання викликів функції. Ці значення називаються такими, що ***повертаються***.

Параметрів у функції може не бути, тоді й дужки відсутні, а виклик такої функції є просто її ім'ям.

Після заголовка структура функції повторює структуру програми за винятком лише точки в кінці. У функції можна визначати свої змінні, сталі та функції. Проте функція істотно відрізняється від програми тим, що:

1) функція записується серед означень програми;

2) ім'я самої програми ніде в програмі не вказується, тоді як серед операторів функції *обов'язково повинні бути оператори присвоювання з ім'ям функції в лівій частині*, причому при виконанні виклику функції хоча б один із них повинен бути виконаним.

*Виклик функції є виразом того типу, який указано в її заголовку*. І він, як усякий вираз, може бути частиною складнішого виразу. Наприклад, за необхідності ми могли б написати d1:=sqr(dd(x1, y1, 1, 2)+1).

Повернемося до прикладу. Нескладно написати функцію обчислення меншого з двох значень:

**function** min(x1, x2 : real):real;

**begin**

**if** x1<x2 **then** min:=x1

**else** min:=x2

**end;**

і помістити її слідом за функцією dd у програмі minimdis. З її використанням обчислення мінімального зі значень змінних d1, d2, d3 можна в тілі програми задати так:

m:=min(d1, d2); m:=min(m, d3)

або навіть так:

m:=min(min(d1, d2), d3)

При обчисленні останнього виразу спочатку виконується "внутрішній" виклик min(d1, d2). Значення, обчислене при його виконанні, стає аргументом у "зовнішньому" виклику.

**Задачі**

**3.13.**\* Написати функцію even, тобто "парне", що задає обчислення ознаки парності цілого.

**3.14.** Написати функцію обчислення за дійсним параметром x:

а) його знака (sign(x)=-1, 0 або 1 відповідно при x<0, x=0, x>0);

б)\* ceil(x) – найменшого цілого, що не менше, ніж значення параметра (для від'ємних значень параметра можливі два варіанти означення).

**3.15.** Написати програму обчислення периметра й площі трикутника за координатами його вершин.

**3.16.** Написати тригонометричні функції з дійсним параметром, значення якого вимірюються в градусах.

**3.17.** Написати функцію означення за довжинами трьох відрізків, чи утворюють вони трикутник З її використанням написати програму обчислення, скільки трикутників можна утворити з чотирьох заданих різних відрізків.

**4. Процедури, підпрограми та параметри**

Розглянемо задачу: довільні значення трьох змінних *a*, *b*, *c* переставити за необхідності так, щоб вони були упорядковані за неспаданням, тобто щоб мали місце нерівності *a b c*. Алгоритм розв'язання цієї задачі простий:

*якщо a>b, то обміняти значення змінних a і b;*

{гарантовано, що a b}

*якщо b>c, то обміняти значення змінних b і c;*

{гарантовано, що b c і a c; але нерівність a b не гарантована, тому:}

*якщо a>b, то обміняти значення змінних a і b.*

Обмін значень двох змінних, наприклад, a і b, задається трьома операторами з допоміжною змінною: t:=a; a:=b; b:=t. Мовою Паскаль алгоритм записується так:

**program** sort3(input, output);

**var** a, b, c, t : integer;

**begin**

writeln('задайте три цілих'); readln(a, b, c);

**if** a>b **then** **begin** t:=a; a:=b; b:=t **end**;

**if** b>c **then** **begin** t:=b; b:=c; c:=t **end**;

**if** a>b **then** **begin** t:=a; a:=b; b:=t **end**;

writeln('упорядкування: ', a, ' ', b, ' ', c)

**end.**

Проте три майже однакові складені оператори, що задають ті самі дії, тільки з різними змінними – це нудно. Аналогічно функціям,

*можна один раз описати обмін значень двох змінних, представлених параметрами, дати ім'я цьому опису, а потім тільки позначати його, тобто вказувати ім'я опису й змінні, чиї значення повинні обмінятися*.

На відміну від функцій, при обміні відбувається не обчислення якогось одного значення, а змінюється стан пам'яті програми (недарма обмін заданий складеним оператором). Тому такий опис оформляється й використовується інакше. Опис обміну задається ***процедурою***, а її *виклик являє собою окремий оператор*.

Процедура має загальний вигляд

**procedure** *ім'я*(*означення* *параметрів*);

*означення імен*

**begin**

*послідовність* *операторів*

**end;**

*Процедура, як і функція, є означенням імені і записується серед означень програми.*

На відміну від функції, в її заголовку немає імені типу для значень, породжуваних у результаті виклику, тому що ніякі значення не породжуються. За цією ж причиною *в тілі процедури не може бути операторів присвоювання з її ім'ям у лівій частині*. Виклик процедури складається з імені й аргументів у дужках і записується як окремий оператор, наприклад, readln(x, y).

Отже, напишемо інший варіант програми упорядкування трьох значень:

**program** sort31(input, output);

**var** a, b, c : integer;

**procedure** swap(xx, yy : integer);

**var** t : integer;

**begin** t:=xx; xx:=yy; yy:=t **end**;

**begin**

writeln('задайте три цілих:'); readln(a, b, c);

**if** a>b **then** swap(a, b);

**if** b>c **then** swap(b, c);

**if** a>b **then** swap(a, b);

writeln('упорядкування: ', a, ' ', b, ' ', c)

**end.**

*КРАСИВО, АЛЕ НЕПРАВИЛЬНО!*

Справа в тім, що при виконанні виклику, наприклад, swap(a, b), змінні xx і yy одержать значення змінних a і b, потім ці значення поміняються місцями, виконання виклику закінчиться, а в змінних a і b залишаться ті ж самі значення, що були перед викликом. Наприклад, якщо змінним a, b, c присвоїти "з зовнішнього світу" значення відповідно 3, 1, 2, то буде надруковано **упорядкування: 3 1 2**. Слушність цього напису дуже сумнівна.

Отже, при виконанні виклику процедури (чи функції) *спочатку параметри одержують значення аргументів, а потім їх зміни ніяк не відбиваються на аргументах* (рис.3.3). Тому параметри, що дотепер розглядалися, називаються ***параметрами-значеннями***.

Мова Паскаль допускає в заголовках процедур і функцій означати параметри іншого виду. Вони називаються ***параметрами***-***змінними*** і означаються зі словом **var** попереду. Так, процедура swap набуває вигляду:

**procedure** swap(**var** xx, yy : integer);

**var** t : integer;

**begin**

t:=xx;

xx:=yy;

yy:=t

**end**;

Таке означення параметрів забезпечує, що при виконанні виклику процедури або функції *іменам параметрів ставляться у відповідність змінні, тобто ділянки пам'яті, уже зіставлені аргументам. При виконанні виклику зміна значення параметра-змінної насправді є зміною значення аргументу* (рис.3.4).

|  |  |  |  |
| --- | --- | --- | --- |
| **Момент виконання програми** | **a** | **b** | **c** |
| *перед першим викликом* | 3 | 2 | 1 |
| *після першого виклику* | 2 | 3 | 1 |
| *після другого виклику* | 2 | 1 | 3 |
| *після третього виклику* | 1 | 2 | 3 |
|  |   |   |   |

Якщо в програмі sort31 означити параметри процедури як параметри-змінні, то за виконання виклику swap(a, b) імені xx зіставляється та ж сама ділянка пам'яті, що й змінній a, а імені yy – та ж, що b. *У результаті обмін місцями значень xx і yy є обміном a і b*. Що й було потрібно. Таким чином, якщо в змінні a, b, c програми було прочитано значення 3, 2, 1 відповідно, то результати виконання викликів процедури swap можна подати станами пам'яті програми, як на рис.3.5.

Функції та процедури в мові Паскаль мають загальну назву: ***підпрограми***. У заголовках підпрограм можна означати як параметри-значення, так і параметри-змінні. Означення однотипних параметрів того самого виду називається ***секцією***, і означення параметрів насправді є послідовністю секцій.

Секція параметрів-значень – це список імен, за яким після двокрапки записано ім'я типу, наприклад, a1, a2 : real. Секція параметрів-змінних починається словом **var**, за яким записано список імен параметрів таім'я типу, наприклад,

**var** xx, yy : integer.

Секції розділяються ";". За необхідності ми могли б написати, наприклад,

**procedure** qq(x, y : integer; **var** z, t : integer).

Як ми вже говорили, у викликах підпрограм вказуються аргументи – вирази, однотипні з параметрами. Але є суттєва відмінність між аргументами, що можуть відповідати параметрам-значенням і параметрам-змінним.

*Аргументом для параметра-значення може бути будь-який вираз, тип якого сумісний за присвоюванням із типом параметра.*

*Аргументом для параметра-змінної може бути тільки ім'я змінної того ж типу, що й параметр.*

У літературі часто параметри підпрограм називаються ***формальними*** ***параметрами***, а аргументи у викликах – ***фактичними***.

**Задачі**

**3.18.**\* Як Ви гадаєте, процедури readln і writeln мають параметри-значення або параметри-змінні?

**3.19.**\* Як відомо, будь-які дві різні точки площини задають єдину пряму, що проходить через них. Рівняння прямої *ax*+*by*+*c*=0 називається ***нормалізованим***, якщо (b=1) або (b=0 і a=1). Пряма може бути задана не єдиним рівнянням, але її нормалізоване рівняння єдине.

Написати процедуру обчислення коефіцієнтів нормалізованого рівняння прямої за координатами двох різних точок.

Написати функцію перевірки, чи лежать дві точки площини по один бік прямої, заданої коефіцієнтами нормалізованого рівняння.

З використанням цих підпрограм написати програму читання координат точки і вершин трикутника і перевірки, чи лежить точка всередині його.

**3.20.** Прочитати координати двох пар точок, якими задано два відрізки, та визначити, чи мають вони хоча б одну спільну точку.

**3.21.** Прочитати координати точок *A*, *B*, *C*, *D*. Обчислити довжину найкоротшого шляху з точки *A* в точку *B* з урахуванням того, що відрізок *CD* перетинати не можна.

**3.22.** Прочитати координати точок *A*, *B*, *C*, *D* і визначити, чи є замкнена ламана *ABCDA*:

а) чотирикутником; б) неопуклим чотирикутником; в) опуклим чотирикутником.

**5. Підзадачі, підпрограми та бібліотеки підпрограм**

Підпрограми, як очевидно з попередніх двох параграфів, використовуються для організації програми. Якщо в кількох місцях програми треба описати по суті ті самі обчислення, але з різними значеннями або змінними, то використання підпрограм може скоротити програму, зробити її більш зрозумілою й заощадити час на її створення. Якщо програма – це опис розв'язання якоїсь задача, то підпрограма, як правило, – це опис розв'язання частини цієї задачі.

У багатьох випадках частину задачі можна сформулювати так само чітко, як і самому задачу, тобто виділити її як ***підзадачу***. Наприклад, у задачі обчислення найкоротшої з довжин відрізків, утворених точками, виділяється підзадача обчислення довжини відрізка, а в задачі про переупорядкування значень трьох змінних – підзадача обміну значень двох змінних. Таким чином,

*прагматика підпрограм, тобто те, для чого вони призначені, є опис розв'язання підзадач.*

Розв'язання переважної більшості задач на програмування починається з аналізу їх умови. При цьому дуже важливо правильно виділити підзадачі – це дозволить використовувати підпрограми і прискорить створення програми в цілому.

Кожна система програмування має у своєму складі цілий набір уже готових підпрограм для розв'язання різноманітних задач. Ці задачі виникають як підзадачі практично в будь–який задачі програмування і по суті є стандартними. До них відносяться, наприклад, задачі обчислення математичних функцій (sin, exp тощо) або читання значень із зовнішніх носіїв даних. Підпрограми розв'язання деяких таких задач нам уже знайомі, про інші ми ще дізнаємося.

***Стандартні*** ***підпрограми*** в системах програмування зібрано в спеціальний набір – ***бібліотеку***. У процесі побудови машинної програми вони додаються до програми, начебто були в ній визначені. Якщо ж програма інтепретується, вони завантажуються з бібліотеки й виконуються. *Знати їх корисно і необхідно в практичному програмуванні, адже користуватися готовими деталями набагато легше, ніж створювати їх самому*.

**Задача**

**3.23.** Виділити у задачах 3.19–3.22 підзадачі та сформулювати їх окремо. Чи є в цих задачах спільні підзадачі?