**ПАСКАЛЬ: область дії, модулі, структури.**

**1. Область дії означень та побічний ефект**

Програма може містити кілька підпрограм, і в них самих можуть бути підпрограми тощо. Виникає питання, чи можна в одній підпрограмі використовувати імена, означені в іншій? І зокрема, чи можна в підпрограмах користуватися іменами з програми, і навпаки? У цьому параграфі ми дамо відповіді на ці та деякі інші питання.

Означення імені задає позначення деякого об'єкта, наприклад, змінної або підпрограми. Так, після означення

**var** aaa : integer;

ім'я aaa позначає змінну цілого типу, а не сталу або щось інше. Таке завдання позначення об'єкта називається його ***іменуванням***. Виникає питання, чи можна різні об'єкти іменувати однаково, тобто давати їм те саме ім'я? Виявляється, можна, але не завжди. Уточнимо це, увівши нове поняття.

***Область*** ***дії*** ***означення*** ***імені***, або ***область*** ***дії*** ***іменування*** – це сукупність місць у програмі, де ім'я позначає об'єкт, указаний саме в цьому означенні. Область дії іменування часто називають "область дії імені", що не зовсім точно.

За правилами мови Паскаль, *означення імені діє від того місця програми або підпрограми, де воно записано, до кінця її блоку. Якщо в цій області є підпрограми, то означення діє й у них. Але якщо вони містять своє власне означення цього імені, то за тими самими правилами до кінця їх блоків діють їх власні означення.*

Можна сказати, що *власне означення в підпрограмі "перекриває" означення, записане вище*.

**Приклад 1.** У програмі

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

**var** t:integer; {1}

**procedure** plus; {1}

**procedure** minus; {1}

**begin** t:=t-1 **end**; {1}

**var** t:integer; {2}

**begin** t:=1; t:=t+1; minus **end**; {2}

**begin** t:=1; plus; writeln(t) **end**. {1}

коментарі {1} і {2} позначають перше й друге означення імені *t* й відповідні рядки програми, у яких вони діють. При виконанні виклику процедури *minus* зменшується значення змінної, означеної в програмі, а при виконанні операторів процедури *plus* збільшується значення зовсім іншої змінної – іменованої в означенні {2}. Означення {1} перекрите в процедурі *plus* її власним означенням {2}. Тому при виконанні програми друкується 0.

Таким чином, різні об'єкти можна іменувати однаково, але в різних підпрограмах (однією з них вважається програма). *В означеннях тієї самої підпрограми усі імена повинні бути різними.*

Тепер підійдемо з протилежного боку. Нехай у деякому місці програми використовується ім'я. Як дізнатися, якому з означень, записаних вище, воно відповідає?

Будемо говорити, що програма або підпрограма ***охоплює*** підпрограми, записані в ній, і що вони ***вкладені*** в неї. Наприклад, програма twovars охоплює процедури plus і minus (і вони обидві вкладені в неї), а процедура plus охоплює minus. З правила, яким задається область дії означення, випливає, що *ім'я в підпрограмі відповідає найближчому з його означень, записаних вище в підпрограмі або підпрограмах, що її охоплюють, зокрема, в програмі.*

Звідси одразу маємо відповіді на питання, поставлені спочатку: у підпрограмі можна використовувати ім'я, означене в підпрограмі, що її охоплює, або програмі, але не можна використовувати ім'я, означене не вище в блоці або не в підпрограмі, що її охоплює.

Основний практичний висновок з цих правил – *намагатися уникати вкладених підпрограм і прагнути до того, щоб усі підпрограми були вкладені тільки в програму*, *тобто щоб у них не було своїх підпрограм*.

У цьому розумінні програма twovarst явно невдала.

Означення підпрограми починаються з її параметрів, а якщо їх немає, то з початку блоку. Тому *підпрограма іменується не в ній самій, а в програмі,* *або, в гіршому випадку, в підпрограмі, що її охоплює*. Звідси випливає, що означення імені підпрограми діє до кінця блоку, в якому її записано. Тому її можна викликати в інших підпрограмах, записаних нижче в цьому ж блоці.

**Приклад 7.2.** Раціональне число подається нескоротним дробом *A*/*B*, де *B*>0. Сумою двох дробів *A*/*B* і *C*/*D* є результат скорочення дробу (*AD*+*BC*)/*BD*. Напишемо програму, у якій описується читання двох дробів, їх додавання зі скороченням результату й виведення у вигляді нескоротного дробу. Будемо припускати, що знаменники дробів, що читаються, не рівні 0.

Опис дій із дробами оформимо у вигляді підпрограм: читання *readfr*, додавання *plusfr* і друкування *writefr*. Закінчення *fr* в іменах процедур – це скорочене *fraction*, тобто "дріб".

У програмі означимо цілі змінні *a*, *b*, *c*, *d*, *r1*, *r2* для зберігання прочитаних дробів *a/b* і *c/d* та результату операції *r1/r2*. Не уточнюючи самих підпрограм, запишемо поки лише тіло програми з їх використанням:

**begin**

writeln('задайте дріб (два цілих, друге не 0):'); readfr(a, b);

writeln('задайте дріб (два цілих, друге не 0):'); readfr(c, d);

plusfr(a, b, c, d, r1, r2);

writefr(r1, r2);

**end**.

А тепер почнемо уточнювати підпрограми. Процедура читання очевидна:

**procedure** readfr(**var** x, y : integer);

**begin** readln(x); readln(y) **end**;

Передбачається, що при виконанні програми користувач набере дві цілі сталі.

Процедура додавання повинна повертати результат додавання у вигляді нескоротного дробу. Нехай її параметри *x1*, *y1*, *x2*, *y2* задають два дроби, що додаються, а *x3*, *y3* – результат. Скорочення дробу задамо процедурою *redufr*. Її параметри-змінні задають дріб (скорочений спочатку і в результаті). Тоді додавання задається такою процедурою:

**procedure** plusfr(x1, y1, x2, y2 : integer; **var** x3, y3 : integer);

**begin** x3:=x1\*y2+x2\*y1; y3:=y1\*y2;

redufr(x3, y3)

**end**;

Щоб скоротити дріб, треба його чисельник і знаменник розділити на їх найбільший спільний дільник (задача 4.5). Нехай його обчислення задає функція *gcd*, яку ми тут не уточнюємо. Тоді процедура скорочення redufr очевидна:

**procedure** redufr(**var** x1, y1 : integer);

var t : integer;

**begin**

t:=gcd(x1, y1);

x1:=x1 **div** t; y1:=y1 **div** t

**end**;

Нарешті, наведемо розташування наведених підпрограм у блоці програми. Функція *gcd* повинна бути записана перед процедурою *redufr*, оскільки в ній визивається. Так само ця процедура повинна бути записана перед процедурою *plusfr*. Взаємне розташування *plusfr*, *writefr* і *readfr* не має значення:

…

**function** gcd(x, y : integer) : integer; …

**begin** … **end**;

**procedure** redufr(**var** x1, y1 : integer); …

**begin** … gcd … **end**;

**procedure** plusfr(x1, y1, x2, y2 : integer; **var** x3, y3 : integer);

**begin** … redufr … **end**;

**procedure** readfr(**var** x, y : integer);

**begin** … **end**;

**procedure** writefr(x, y : integer);

**begin** write(x, '/', y) **end**;

Запис програми в остаточному вигляді залишаємо як вправу.

Переозначати ім'я підпрограми в ній самій мова Паскаль забороняє. Але дозволяє використовувати його, тобто позначати ним виклики цієї ж підпрограми. Такі виклики підпрограми усередині її самої називаються ***рекурсивними***; ми познайомимося з ними в розділі 9.

Ім'я, означене в підпрограмі (або програмі), називається ***локальним*** ***у* *ній***. Ім'я, записане, але не означене в підпрограмі, називається ***глобальним*** ***у* *ній***. Воно може бути означеним у одній з підпрограм, що охоплюють, або програмі. У прикладі 7.1 ім'я *t* є глобальним в процедурі *minus* і в процедурі *plus*, але тільки до означення {2}; далі воно є локальним і позначає зовсім іншу змінну, хоча й того ж типу.

У наступному підрозділі ми розглянемо модулі – спеціальні "збірники означень", які дозволяють використовувати імена взагалі без означення в програмі. Втім, такі імена вже знайомі – наприклад, імена математичних функцій або підпрограм readln і writeln.

Зміна змінної з глобальним ім'ям у підпрограмі називається ***побічним*** ***ефектом*** ***підпрограми***. Наприклад, у програмі *twovars* побічний ефект процедури minus полягає в зміні значень змінної з ім'ям *t*.

Побічний ефект називається ***явним***, якщо він заданий операторами присвоювання. Такий побічний ефект процедури *minus*. ***Неявний*** побічний ефект задається, якщо глобальне ім'я вказати у виклику підпрограми як аргумент, що відповідає параметру-змінній. Наприклад, у виклику процедури читання.

Програміст може задати побічний ефект, іноді навіть не бажаючи цього, через неуважність. Результати такої помилки можуть виявитися цілком несподіваними й навіть дуже сумними для автора програми. Тому *використання глобальних імен вимагає особливої уваги*. Втім, це аж ніяк не означає, що користуватися ними не варто. Як і побічним ефектом. Усе добре в міру й за своїм призначенням.

При імітації виконання програми можна ввести додаткові позначення для локальних імен у підпрограмах, щоб явно відрізняти їх від глобальних. Наприклад, якщо в підпрограмі *S* означено ім'я *N*, то позначимо його *S.N*, а якщо підпрограма *S2* вкладена в підпрограму *S1* і містить означення імені *N*, то позначимо його *S1.S2.N* тощо. Зокрема, у прикладі 7.1 ім'я *t* у підпрограмі minus є глобальним і додаткового позначення не одержує, а в підпрограмі *plus* позначається *plus.t*.

Як відомо з розд.2, при виконанні виклику підпрограми її параметрам-змінним зіставляється пам'ять аргументів. При імітації виконання програми можна "сумістити" параметр-змінну з аргументом. Іншим іменам змінних, означеним у підпрограмі, зіставляються свої власні ділянки пам'яті. Наприклад, виконання програми

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

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

**procedure** ps(a : integer; var b : integer);

**var** t : integer;

**begin** t := a+b; b := t-b; a := t-b; c := t **end**;

**begin**

a := 1; b := 5; c := 2;

ps(b, a);

writeln(a, b, c)

**end**.

можна відбити такою таблицею:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Виконувані дії** | a | b | c |   |   |
| a:=1; b:=5; c:=3 | 1 | 5 | 2 |   |   |
| Виклик ps | ps.b |   |   | ps.a | ps.t |
| Неявне ps.a:=b | 1 | 5 | 2 | 5 | ? |
| ps.t:=ps.a+ps.b | 1 | 5 | 2 | 5 | 6 |
| ps.b:=ps.t-ps.b | 5 | 5 | 2 | 5 | 6 |
| ps.a:=ps.t-ps.b | 5 | 5 | 2 | 1 | 6 |
| c:=ps.t | 5 | 5 | 6 | 1 | 6 |
| writeln(a, b, c) | 5 | 5 | 6 |   |   |

Суміщення імен *a* і *ps.b* в одній колонці вказує, що цим іменам зіставлена та сама ділянка пам'яті. У результаті виконання буде надруковано 5 5 6.

**Задачі**

**1.**\* Укажіть помилкове використання імен у програмі:

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

**function** A : integer;

**function** B : integer;

**function** A : integer;

**begin** A:=1 **end**

**begin** A := 2; B := A **end**;

**begin** A := 3 **end**;

**begin** writeln(A); writeln(B) **end**.

**2.** Імітувати виконання програми:

**program** (input, output);

**var** a, b : integer;

**procedure** badswap(**var** a : integer; t : integer);

**var** d : integer;

**begin**

d := t; t := a; a := d

**end**;

**begin**

a := 1; b := 3;

badswap(a, b);

writeln(a, b)

**end**.

**3.**\* Написати програму, за допомогою якої можна встановити, чи завжди обчислюються праві операнди бульових операцій **and** і **or**.

**4.**\* Дописати необхідні означення до тіла програми, щоб при її виконанні було надруковано не "0", а "1":

**begin**

writeln(b\*c-c\*b)

**end**.

**2. Модуль – збірник означень**

Повернемося до задач 3.19–3.22. У програмах для їх розв'язання використовуються ті самі підпрограми обчислення коефіцієнтів рівняння прямої та перевірки, чи лежать точки по один бік прямої. Ці спільні підпрограми, а також інші означення, можна вилучити з програм і зібрати в спеціальному "збірнику означень". Цьому збірнику можна дати ім'я і вказувати його в програмах замість вилучених означень, помітно скорочуючи текст програм. Стандарт мови Паскаль, правда, такої можливості не дає, але всі системи програмування її забезпечують. Збірник означень називається ***модулем***; конкретний його синтаксис залежить від системи програмування. Розглянемо модулі на прикладі діалекту Турбо Паскаль.

**Приклад 7.3.** Напишемо модуль з означеннями імен *normcoef* і *oneside* – імен підпрограм обчислення коефіцієнтів нормалізованого рівняння та перевірки, чи лежать дві точки по один бік прямої (див. відповідь до задачі 3.19).

Модуль має загальний вигляд:

**Unit** *ім'я-модуля*;

*Інтерфейсний розділ*;

*Розділ реалізації*;

*Розділ ініціалізації*

**end**.

***Інтерфейсний*** ***розділ*** починається службовим словом **Interface** і містить означення імен, призначених для використання за межами модуля. Проте замість підпрограм тут записуються тільки їх заголовки. У нашому прикладі цей розділ виглядає так:

**Interface**

**procedure** normcoef(x1, y1, x2, y2 : real; **var** a, b, c : real);

**function** oneside(x1, y1, x2, y2, a, b, c : real) : boolean;

Слово "інтерфейс" перекочувало з англійської. Одне з його значень – точка з'єднання, взаємодії різноманітних систем. Інше – спосіб сприйняття і виразу даних, заданий програмою. ***Інтерфейс*** ***програми*** можна розуміти як "зовнішній вигляд її виконання". Відповідно інтерфейс модуля – це "*те, що в ньому видно ззовні*". У даному прикладі це заголовки двох підпрограм.

***Розділ*** ***реалізації*** починається службовим словом **Implementation** і містить підпрограми, вказані в інтерфейсному розділі, але зі скороченими заголовками – за словом **function** або **procedure** записується тільки ім'я. У нашому випадку вигляд цього розділу такий:

**Implementation** {"втілення", "реалізація"}

**procedure** normcoef; {скорочений заголовок}

**begin** ... {повний блок; див. відповідь до задачі 3.21} **end**;

**function** oneside; {скорочений заголовок}

**begin** ... {повний блок; див. відповідь до задачі 3.19} **end;**

Якби в цих підпрограмах використовувалися підпрограми, не вказані в інтерфейсному розділі, то їх слід було б записати на початку розділу реалізації. У нас таких поки що немає.

Узагалі, імена, означені в розділі реалізації і не вказані в інтерфейсному розділі, називаються ***іменами, захованими в модулі***. Вони призначені для використання всередині модуля, але не за його межами.

Відзначимо, що *для модуля діють ті ж самі правила дії іменувань, що і для програм.*

***Розділ*** ***ініціалізації*** задає дії, що виконуються один разів на початку виконання програми, яка використовує означення модуля. Він має вигляд:

**Begin** *послідовність операторів*

У цих операторах, як правило, задається присвоювання початкових значень змінним, означеним у модулі (їх ініціалізація). У нашому прикладі ніякі початкові дії не задаються, і слово **Begin** з операторами відсутні.

Запис усього модуля з ім'ям *PlaneGeo* залишаємо як вправу.

Використання імен модуля в програмі або в іншому модулі називається ***успадкуванням***, або ***використанням*** ***модуля*** й у діалекті Турбо Паскаль указується відразу після заголовка програми або слова **Interface** у модулі в такому вигляді:

**uses** *ім'я-модуля*;

Якщо програма або модуль використовує кілька інших модулів, то їх імена записуються через кому. Програма розв'язання задачі 3.21 із використанням модуля *PlaneGeo* стає істотно коротше:

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

**uses** PlaneGeo;

**var** x, y, {точка}

xt1, yt1, xt2, yt2, xt3, yt3, {вершини}

a1, b1, c1, a2, b2, c2,

a3, b3, c3 : real; {коефіцієнти прямих}

**begin**

writeln('задайте дві координати точки : ');

readln(x, y);

writeln('задайте 3 пари координат вершин трикутника : ');

readln(xt1, yt1, xt2, yt2, xt3, yt3);

normcoef(xt3, yt3, xt2, yt2, a1, b1, c1);

normcoef(xt3, yt3, xt1, yt1, a2, b2, c2);

normcoef(xt1, yt1, xt2, yt2, a3, b3, c3);

**if** oneside(x, y, xt1, yt1, a1, b1, c1) **and**

oneside(x, y, xt2, yt2, a2, b2, c2) **and**

oneside(x, y, xt3, yt3, a3, b3, c3)

**then** writeln('Так, точка всередині трикутника ')

**else** writeln('Ні, точка за межами трикутника ')

**end**.

Як бачимо, імена *normcoef* і *oneside* у програмі не означені, але використовуються завдяки тому, що на початку програми вказано використання модуля **uses**.

Подібно до Паскаль-програм, "Паскаль-модулі" записуються у файли з розширенням .*pas*. При трансляції модуля його означення транслюються в машинну мову й записуються у файл із розширенням .*tpu*. Спочатку треба транслювати модуль, і тільки після цього можна транслювати програми й інші модулі, що використовують його. Так, якщо модуль із прикладу 7.3 записаний у файлі з ім'ям *PlaneGeo*.*pas*, то в результаті його трансляції з'являється файл *PlaneGeo*.*tpu*. Після цього можна транслювати програму *Intriang*. У процесі її трансляції означення з модуля, а точніше, із файла *PlaneGeo*.*tpu*, додаються в машинну програму завдяки вказівці

**uses** PlaneGeo.

Отже, *використання модулів дозволяє не повторювати ті самі означення в різних програмах і цим істотно їх скорочувати.*

Відзначимо, що всі стандартні підпрограми та інші означення в системах програмування збираються в кілька модулів. Серед них, як правило, є один "найголовніший", що використовується практично кожною програмою. Він містить, наприклад, процедури читання й запису та математичні функції (записані машинною мовою). Означення з нього, що використовуються програмою, при її трансляції додаються неявно, тобто без указання **uses.** Використання всіх інших модулів треба задавати явно.

**Задача**

**5.** Оформити модуль PlaneGeo і використати його в розв'язанні задач 3.20–3.22.

**3. Структури, вони ж записи**

У програмі Intriang і підпрограмах модуля *PlaneGeo* із прикладу 7.3 точки площини подано двома дійсними числами, а прямі – трьома. Ці пари та трійки дійсних значень по суті подають *два нових види об'єктів – точки та прямі*. У модулі було також означено специфічні операції над цими об'єктами, наприклад, побудова коефіцієнтів рівняння прямої за парою точок. Ще раніш було означено функцію обчислення відстані між двома точками.

Усе це дозволяє говорити про дві множини об'єктів і операції над ними, тобто про два *нові типи*. Елементами цих типів є точки площини та прямі на ній, або відповідно пари та трійки дійсних чисел. Для подання пар (трійок тощо) значень, типи яких уже означено (у тому числі базові типи), використовуються вирази вигляду

**record**

*список означень імен полів*

**end**.

Наприклад, тип точок площини, або пар дійсних, можна задати так:

**type** Point =

**record**

x : real; y : real

**end**;

Це означення задає ім'я типу *Point* для елементів, утворених парами дійсних значень. Змінна типу *Point* складається з двох компонентів – дійсних змінних, що називаються її ***полями*** та іменуються *x* і *y*.

Слово **record** означає "запис", і цей термін не зовсім удалий. Слово **structure** (структура), напевно, було б краще. Значення типу "запис", тобто пари, трійки тощо значень, а також змінні таких типів називаються ***записами***, або ***структурами***. Типи структур називаються ***структурними***. На відміну від скалярних типів, їх елементи складені з частин, які *ідентифікуються окремо*.

Імена однотипних полів запису можна означити разом, подібно до імен однотипних змінних. Наприклад, в означенні типу *Point* можна було б написати x,y:real. Аналогічно, тип трійок дійсних коефіцієнтів рівнянь, якими задаються прямі на площині, можна означити так:

**type** Line = **record**

a, b, c: real

**end**

У загальному випадку вираз вигляду

**record**

*ім'я1* : *тип-1*;

*ім'я2* : *тип-2*;

...

*ім'яN* : *тип-N*

**end**

задає ***структурний*** ***тип***. Його носій – множина *N*-ок, у яких на першому місті стоїть значення *типу-1*, на другому – *типу-2* тощо. Ця множина називається ***декартовим******добутком***носіїв типів, указаних у списку. Такий вираз можна використовувати в означеннях як типів, так і змінних. Наприклад,

**var** lin : **record**

a, b, c : real;

**end**;

Компоненти структур можуть мати довільні типи, у тому числі структурні. Наприклад, можна означити тип "відрізок на площині", що задається парою точок – його кінців:

**type** Segm = **record**

pe1, pe2 : Point

**end**;

Компонент змінної-структури, тобто поле, позначається його ім'ям, яке записується після імені змінної та відокремлюється крапкою. Наприклад, якщо змінна *pn* має тип *Point*, то її компоненти позначаються *pn.x* і *pn.y*.

Вирази такого вигляду прийнято називати ***селекторами*** ***запису***, тому що ім'я поля задає його *вибір*, або *селекцію*,серед інших полів. Такі вирази використовуються так само, як і імена змінних відповідних типів. Наприклад, якщо є означення

**var** p1, p2 : Point;

то ми вправі написати p1.x:=1; p1.y:=sqr(p1. x) і т.п.

Якщо поле саме є структурою, то його компоненти позначаються точно так само, через точку після імені поля. Наприклад, дійсні поля змінної *sg* типу *Segm* позначаються *sg.pe1.x*, *sg.pe1.y* тощо.

*Значення змінних-структур можна присвоювати змінним, однотипним із ними.* Наприклад, якщо діють означення **var** a, b : Segm, то можна написати:

a.pe1.x := 1; a.pe1.y := a.pe1.x+1; a.pe2 := a.pe1; b:=a.

Очевидно, що ці оператори задають присвоювання обом відрізкам того самого значення [(1; 2); (1; 2)].

*У мові Турбо Паскаль змінні структурних типів можна ініціалізувати*. Стала структурного типу записується в дужках і схожа на вираз, що задає тип, тільки замість імен типів указуються значення. Наприклад,

**const** p1 : Point = ( x : 1; y : 2 );

Як бачимо, поля відокремлюються знаком ";" незалежно від того, як вони відокремлювалися в означенні типу. Аналогічно ініціалізується структура з полями-структурами, наприклад, типу відрізка площини:

**const** z : Segm = (pe1 : (x : 1; y : 1); pe2 : (x : 2; y : 2));

Тут змінна *z* ініціалізується як відрізок [(1;1);(2;2)].

*Операцій над структурами як цілісними значеннями в мові Паскаль немає.* Тому обробка структур описується через обробку їх компонентів із застосуванням операцій базових типів.

Тип значень, що повертаються з функції, не може бути структурним. Проте параметри підпрограм можуть бути довільними структурами. Єдине обмеження в мові Турбо Паскаль – *тип параметра в заголовку підпрограми можна задавати лише ім'ям, а не довільним виразом*. Таким чином, необхідні типи треба іменувати вище в програмі, до їх використання в заголовках підпрограм.

*Всі поля того самого списку означень імен полів повинні мати різні імена*. Проте ім'я поля може повторюватися в інших списках або збігатися з ім'ям змінної. Наприклад, допустимі такі означення:

**type** rectyp = **record** x : real; z : point **end;**

**var** x : rectyp;

Отже, ми маємо засоби для опису множин пар, трійок тощо, складених із компонентів базових типів. Проте

*означення типу (у математичному змісті) вимагає задання не тільки елементів множини, але й операцій над ними.*

Операції можна реалізувати у вигляді підпрограм, описавши обробку структур через обробку їх компонентів. Означення множин елементів і підпрограми можна зібрати в модулі – утвориться *реалізація нестандартного типу елементів*. Приклади такої реалізації типів ми розглянемо в наступному підрозділі.

**Задачі**

**6.** Означити тип структури з мінімально можливим числом полів для подання:

а) відрізка прямої; б) відрізка площини; в) кола на площині;

г) трикутника на площині; д) прямокутника на площині; е) раціонального дробу;

ж) прямокутника на площині зі сторонами, паралельними координатним осям.

Написати функцію перевірки рівності двох елементів відповідного типу.

**4. Приклади створення та використання модулів**

**Приклад 7.4.** Ще раз повернемося до задачі 3.21 і напишемо варіант її розв'язання, реалізувавши поняття "точка площини" і "пряма площини" у новому модулі *Geoplan*. Його інтерфейсний розділ має починатися з означень імен типів для множин точок і прямих:

**type** Point : **record** x, y : real **end**;

Line : **record** a, b, c : real **end**;

Далі запишемо заголовки підпрограм *normcoef* і *oneside*, указавши замість пар і трійок дійсних параметрів параметри типу *Point* і *Line*:

**procedure** normcoef(p1, p2 : Point; **var** lin : Line);

**function** oneside(p1, p2 : Point; lin : Line) : boolean;

Далі додамо заголовок підпрограми "читання точки", яку природно викликати в програмах, що мають справу з точками:

**procedure** RdPoint(**var** p : Point);

Далі нам будуть потрібні інші типи та підпрограми – додаватимемо їх до модуля за необхідності. А поки інтерфейсний розділ на цьому закінчується.

У розділі реалізації запишемо блоки підпрограм із скороченими заголовками, описавши обробку параметрів через обробку їх компонентів:

**procedure** normcoef;

**begin**

**if** p1.x=p2.x **then**

**begin** lin.b:=0; lin.a:=1; lin.c:=-p1. x **end**

**else**

**begin**

lin.b := 1; lin.a := (p1.y - p2.y)/(p2.x - p1.x);

lin.c := -p1.y - lin.a \* p1.x

**end**

**end;**

**function** oneside;

**begin**

oneside:=(lin.a\*p1.x+lin.b\*p1.y+lin.c)\*(lin.a\*p2.x+lin.b\*p2.y+lin.c)>0

**end;**

Читання точки задається через читання її координат:

**procedure** RdPoint;

**begin** readln(p.x, p.y) **end**;

От, власне, і весь модуль *Geoplan* – його остаточне оформлення залишаємо як вправу (див. підрозділи 7.1, 7.2).

З використанням цього модуля програма *Intriang* набуває вигляду:

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

**uses** Geoplan;

**var** p, {точка}

pt1, pt2, pt3 : Point; {вершини}

lin1, lin2, lin3: Line; {прямі}

**begin**

writeln('задайте координати точки площини:'); RdPoint(p);

writeln('задайте координати трьох точок площини:');

RdPoint(pt1); RdPoint(pt2); RdPoint(pt3);

normcoef(pt3, pt2, lin1);

normcoef(pt3, pt1, lin2);

normcoef(pt1, pt2, lin3);

**if** oneside(p, pt1, lin1) **and**

oneside(p, pt2, lin2) **and**

oneside(p, pt3, lin3)

**then** writeln('Так, точка всередині трикутника ')

**else** writeln('Ні, точка зовні трикутника ')

**end**.

**Задачі**

**7.** Розв'язати задачі 3.20–3.22 з використанням модуля Geoplan, за необхідності додавши в нього нові означення.

**8.** Написати програму перевірки:

а) чи лежать три точки площини на одній прямій;

б) чи проходить пряма через дану точку площини;

в)\*чи належить точка площини заданому многокутнику (про його завдання див. приклад 7.5).

**9.** Раціональне число подається нескоротним дробом *A*/*B*, де *B*>0. Написати модуль, що містить означення типу раціональних чисел, підпрограми їх читання й запису, підпрограми застосування алгебраїчних операцій та порівняння (приклад 7.2). Додати в розділ реалізації модуля допоміжні підпрограми обчислення найбільшого спільного дільника двох цілих і скорочення дробу. З використанням модуля написати "найпростіший калькулятор" раціональних чисел (див. приклад 5.3).