Реферат на тему:

**Мітки та переходи**

У мові Паскаль будь-який оператор і кінець складеного оператора (слово **end)** можна ***відмітити***, тобто ідентифікувати, додати йому індивідуальне ім'я. Це ім'я називається ***міткою***. У авторській версії мови мітками могли бути цілі сталі від 1 до 9999, у мові Турбо Паскаль до них додано ідентифікатори. Мітка записується перед оператором або словом **end** через двокрапку, наприклад,

1 : money := 21;

mmm : money:=0;

finita : **end**.

Мітки, використовувані в тілі програми або підпрограми, *повинні бути означені в її ж блоці*. Означення міток має вигляд:

**label** *список-міток-через-кому* ;

наприклад, **label** 1, mmm, finita;.

Мітка, означена в блоці, повинна відмічати рівно один оператор у тілі цього блоку.

Мітки використовуються в ***операторах*** ***переходу***, що мають вигляд

**goto** *мітка*

наприклад,

**goto** 1; **if** x>1000 **then goto** mmm.

*Результатом виконання оператора переходу є те, що слідом за ним виконується оператор, відзначений цією міткою.*

У випадку, коли відзначений кінець складеного оператора, можливі варіанти.

Якщо це не кінець тіла циклу і не кінець програми, то буде виконуватися наступний оператор.

Якщо відзначений кінець програми, то її виконання завершується.

Якщо відзначений кінець тіла циклу, то виконуються дії, які слідують за виконанням тіла (перевірка умови продовження **while-**циклу або неявна зміна параметра **for-**циклу).

Оператор переходу й відповідний відмічений оператор повинні бути записаними в тілі блоку (програми або підпрограми), де цю мітку означено. Іншими словами,

*переходи з одного блоку в інший і використання неозначених у блоці міток заборонено.*

Ми не схильні популяризувати використання операторів переходу. Більше того, у свій час було доведено, що без них узагалі можна обійтися (достатньо умовних операторів і **while**-циклів). Проте є три випадки, коли указання переходу доречно й зручно:

* на кінець підпрограми;
* зсередини циклу на його кінець;
* зсередини циклу на наступний за циклом оператор.

У Турбо Паскаль для указання таких переходів є спеціальні оператори, відповідно, **exit**, **continue** і **break**, тобто "вийти" (з підпрограми), "продовжувати" і "перервати" (виконання циклу). Розумно використовуючи ці оператори, можна обійтися без міток і **goto**. Насправді ці три оператори – виклики процедур. Їх імена не є службовими словами, але ми будемо виділяти їхнім **жирним шрифтом**. Приклади їх використання – у наступному підрозділі.

**Читання послідовностей**

Існує чимало задач, у процесі розв'язання яких читаються та обробляються послідовності значень невідомої заздалегідь довжини. До них віноситься практично все, що пов'язано з обробкою файлів – від завантаження машинної програми до друкування списків. У цьому параграфі ми розглянемо задачі, у яких для обробки вхідної послідовності незалежно від її довжини достатньо кількох змінних. На прикладі цих задач ми опишемо три способи завдання кінця послідовності і, відповідно, три види циклів читання й обробки даних.

**1. Спочатку читається кількість значень *n*, *n* maxint, потім самі значення в кількості, яка визначається за *n***. Для опису читання зручно скористатися for-оператором.

**Приклад 1.** Многочлен, він же поліном *pnxn*+*pn-*1*xn-*1+ … +*p*1*x*+*p*0 задається послідовністю з *n*+1 коефіцієнтів *pn*,*pn-*1, … , *p*1, *p*0. Треба прочитати значення *x*, степінь і коефіцієнти полінома та обчислити його значення в точці *x*. Оскільки

*pnxn*+*pn-*1*xn-*1+ … +*p*1*x*+*p*0=(... (*pnx*+*pn-*1)*x*+ … +*p*1)*x*+*p*0,

то значення *v* полінома можна подати як значення останнього члена послідовності: *v*0=0, *v*1=*v*0 *x*+*pn*, *v*2=*v*1 *x*+*pn-*1, … , *vn*+1=*vn x*+*p*0. Неважко переконатися, що вона задається рекурентним співвідношенням

*vi*=*vi-*1 *x*+*pn*+1-i, *i*=1, 2, … *n*+1,

причому перед його першим застосуванням треба знати лише перший коефіцієнт, перед другим – другий тощо. Тому цикл має складатися з читання чергового коефіцієнта й застосування співвідношення. У будь-який момент виконання програми треба знати лише один, останній прочитаний коефіцієнт, тому для збереження коефіцієнтів достатньо однієї змінної.

Отже, спочатку прочитаємо значення *x*, потім степінь полінома *n*, потім *n*+1 коефіцієнт, застосовуючи "по дорозі" рекурентне співвідношення:

V:=0;

writeln('задайте точку дійсної прямої :'); readln(x);

writeln('задайте цілий невід'ємний степінь полінома :'); readln(n);

**for** i:=1 **to** n+1 **do**

**begin**

writeln( 'задайте ', n+1-i, '-й коеф-т :'); readln(p);

V:=V\*x+p

**end**;

{прочитано n+1 коефіцієнт; значення V – шукане}

Оформлення алгоритму у вигляді підпрограми залишаємо вправою. 

**2. Відомо особливе значення, поява якого в послідовності вхідних значень позначає її кінець і як вхідне не розглядається**. Для опису читання зручно скористатися repeat-оператором, тому що треба прчитати не менше, ніж одне значення перед тим, як з'явиться ознака кінця.

**Приклад 2.** На контрольно-пропускному пункті митниці працює одна бригада інспекторів. Автомобілі прибувають, стають у чергу (якщо вона є) і проходять контроль у порядку прибуття. Для кожного автомобіля відома тривалість його контролю *t*: автомобіль покидає митницю через *t* одиниць часу після початку його контролю. Прибуття першого автомобіля задано відносно початкового моменту часу, а прибуття кожного наступного – відносно моменту прибуття попереднього. Отже, вхідними даними є пари цілих чисел *x*1, *t*1, *x*2, *t*2, … , причому *xi* 0, *ti*>0, крім останнього: *t*=0 позначає кінець послідовності вхідних даних. За вхідною послідовністю треба визначити послідовність моментів виїздів автомобілів із контрольно-пропускного пункту.

Перше наближення до розв'язання очевидно:

t0:=0; {особливе значення}

**repeat**

readln(x, t);

**if** t>t0 **then** *обробити x, t і обчислити момент від'їзду y*

**until** t=t0.

Припустимо, що в контролі автомобілів немає пауз: контроль наступного автомобіля, якщо він уже прибув, починається відразу після від'їзду попереднього. За *x*1 і *t*1 можна обчислити момент від'їзду *y*1: *y*1=*x*1+*t*1. Введемо поняття "момент початку контролю автомобіля" і позначимо його *bi*: *bi*=max{*yi-*1,*xi*}. Тоді *yi*=*bi*+*ti*. Звідси очевидним є уточнення фрази "*обробити x, t і обчислити момент від'їзду y*":

**if** y<x **then** b:=x {x – момент приїзду чергового автомобіля}

**else** b:=y; {y – момент від'їзду попереднього автомобіля}

y:=b+t; {тепер y – момент від'їзду чергового автомобіля}

Для першого автомобіля момент від'їзду попереднього відсутній, тому для нього повинно бути b:=x. Щоб не розглядати окремо випадки, перший або не перший автомобіль, будемо вважати початковий момент часом від'їзду "нульового" автомобіля. Тоді алгоритм набуває вигляду:

t0:=0; y:=0;

**repeat**

writeln('момент прибуття й тривалість – два невід'ємних цілих :');

readln(x,t);

**if** t>t0 **then** {обробити x, t і обчислити момент від'їзду y}

**begin**

**if** y<x **then** b:=x **else** b:=y;

y:=b+t;

writeln('час від'їзду : ', y)

**end**

**until** t=t0.

Цикл читання можна записати за допомогою одного старого програмістського трюку. Він полягає у використанні "нескінченного циклу" у сполученні з переходом за кінець циклу. Скористаємося оператором **break** мови Турбо Паскаль:

t0:=0; y:=0;

**while** **true** **do**

**begin**

readln(x, t);

**{!!! }** **if** t=t0 **then** **break**; {ознака кінця: вихід із циклу }

{замість **break** можливо **exit** – вихід із (під)програми}

**if** y<x **then** b:=x **else** b:=y;

y:=b+t;

writeln('час від'їзду : ', y)

**end**

Більш загальним випадком завдання кінця послідовності є повторення найпершого значення послідовності. Схема розв'язання залишається тією самою, тільки спочатку "особливе значення" запам'ятовується в результаті читання, а не присвоювання.

**3. Кінець послідовності значень при читанні з клавіатури задається не їх кількістю і не особливими значеннями – замість набирания чергової сталої натискаються спеціальні клавіші**. В усіх системах програмування мовою Паскаль є функція з ім'ям eof. Для читання послідовності значень із клавіатури її виклик (без аргументу або з аргументом input) записується, як правило, в умові продовження while-циклу такого вигляду:

writeln('задайте значення :');

**while** **not** eof **do**

**begin**

readln(v); *використання та обробка значення v;*

writeln('задайте значення:');

**end**

Після друкування запрошення "задайте значення" виконується виклик функції eof, під час чого комп'ютер очікує натискання на клавіші. Якщо натиснути "особливу клавішу **Ctrl**" і, тримаючи її, натиснути клавішу "Z", то з виконання виклику функції eof повертається **true**. У цьому випадку умова продовження **not**eof хибна, і виконання циклу завершується. Якщо ж натиснути будь-яку іншу клавішу, наприклад, почати набирати сталу, то з виклику eof повертається значення **false**, і починається виконання тіла циклу. При виконанні виклику readln змінній *v* "присвоюється з зовнішнього світу", тобто від клавіатури, відповідне значення. Далі за програмою воно обробляється, потім з'являється запрошення, потім при обчисленні умови продовження викликається eof тощо.

Якщо замість набирання сталої натиснути на Ctrl-Z, то значення змінної *v* не читається, і вона зберігає своє старе значення. Якщо задати кінець послідовності відразу, то змінна *v* залишиться з невизначеним значенням. Тому *радимо ініціалізувати змінні, значення яких читаються в циклі. Втім, варто ініціалізувати всі змінні, значення яких мають бути прочитані.*

Відзначимо, що з повторних викликів функції eof, що виконуються вже після натискання на Ctrl-Z, буде повертатися значення **true**.

**Приклад 3.** Напишемо програму, за якою комп'ютер працює "майже як найпростіший калькулятор".

"Найпростіший калькулятор" працює так. Вхідні дані для нього утворюють послідовність вигляду

*стала знак стала знак … стала,*

де сталі задають дійсні числа, а знаки – операції +, -, \*, /. Результат застосування чергової операції виводиться на екран і стає першим операндом наступної операції (якщо вона буде задана). Наприклад, результатом читання послідовності 1+2\*4 буде 12, а не звичні 9 тому, що обчислюється 1+2=3 і потім 3\*4=12.

Алгоритм роботи "найпростішого калькулятора" дуже простий:

*Спочатку читається перше число і стає поточним результатом. Далі циклічно читаються знак операції і нове число і результат застосування операції до поточного результату і нового числа стає новим поточним результатом.*

Ми поки що не знаємо, як прочитати й подати символи "+", "-", "\*", "/". Замість знаків "+", "-", "\*", "/" будемо вживати цілі сталі 1, 2, 3, 4 відповідно. Нехай кожна стала набирається з нового рядка на клавіатурі й ознакою закінчення є натискання на Ctrl-Z замість уведення чергового знака операції.

Означимо змінні first і second для зберігання першого й другого операндів чергової операції. Позначення операції читається в змінну signop і після читання другого операнда застосовується операція. Результат операції записується на місце її першого операнда.

Найперше число не виводиться, тому що воно з'являється на екрані при набирании першої сталої. Одержуємо програму simplecalculator ("простий калькулятор"):

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

**var** first, second : real;

signop : integer;

**begin**

writeln('Операції +, -, \*, / позначаються числами 1, 2, 3, 4');

writeln('Задайте дійсне число : '); readln(first);

writeln('Задайте знак операції (1, 2, 3, 4) : ');

**while** **not** eof **do**

**begin**

readln(signop);

{Увага 1! }

writeln('Задайте дійсне число : '); readln(second);

**if** signop=1 **then** first:=first+second **else**

**if** signop=2 **then** first:=first-second **else**

**if** signop=3 **then** first:=first\*second **else**

first:=first/second; {Увага 2! }

writeln('результат : ', first);

writeln('Задайте знак операції (1, 2, 3, 4) : ');

**end**

**end.**

Недоліком цієї програми є те, що в ній не вказано обробку можливих помилкових дій того, хто нею користується (користувача). Наприклад, якщо задати сталу 5, 6 тощо як знак операції, то буде виконуватися ділення. Навряд чи таке рішення має сенс. Крім того, якщо після знака ділення задати другий операнд 0, то при виконанні first/second відбувається ділення на 0, результат якого комп'ютеру "невідомий", і програма аварійно завершиться.

Доповнимо програму, щоб її осмислене виконання продовжувалося й після описаних помилок користувача. Такі доповнення називаються "захистом від дурня", а програма з ними – "дурнестійкою".

По-перше, якщо вказано недопустимий знак операції, повернемо користувача до задання нового знака операції, не примушуючи задавати другий операнд. Для цього скористаємося оператором **continue** (продовжувати виконання циклу з обчислення умови продовження).

По-друге, *перед діленням варто перевірити, чи не рівний дільник нулю.*

Така перевірка не завадить у будь-який програмі та в будь-якому її місці, де вказано ділення. У даному випадку ділення на 0 можна запобігти, також повернувши користувача до повторного задання знака операції і нового операнда (зрозуміло, можливі й інші рішення).

Отже, замість рядка з коментарем {Увага 1! } напишемо:

**if** (signop>4) **or** (signop<1) **then**

**begin**

writeln('-----Недопустимий знак операції-----');

writeln('Задайте знак операції (1, 2, 3, 4) : ');

**continue**

**end**;

Замість рядка з коментарем {Увага 2!}, що задає ділення, помістимо оператор:

**if** second<>0 **then** first:=first/second

**else begin**

writeln('-----Спроба ділення на 0: ігнорована-----');

writeln('Задайте знак операції (1, 2, 3, 4) : ');

**continue**

**end**;

Проте навіть із доповненнями програма "не захищена" від набирания символів, що не утворюють числову сталу. Для такого захисту потрібні засоби, які ми почнемо розглядати в розд. 14.

Нарешті, розглянемо задачу, у якій умова продовження читання й обробки чергового значення може виявитися порушеною ще до того, як буде задано закінчення вхідних даних.

**Приклад 4.** Відрізок [*a*; *b*] прямої задається координатами його кінців, тобто парою чисел *a*, *b*, де *a b*. Перетином двох відрізків є або відрізок, або порожня множина точок, наприклад, [1;3] [2;4]=[2;3], [1;2] [3;4]= , [1;2] [2;3]=[2;2].

Треба прочитати послідовність пар чисел, що задають відрізки, і знайти їх перетин.

Припустимо, кінець послідовності відрізків задається за допомогою "Ctrl-Z". Проте *немає сенсу продовжувати читання відрізків після того, як перетин уже прочитаних став порожнім*. У цьому випадку треба відразу видати відповідь і закінчити виконання програми. Припустимо, що можливо "неправильне" задання відрізків у вигляді пари чисел *a*, *b*, де *a*>*b*. У цьому випадку *b* і *a* міняються місцями.

Для збереження поточного перетину означимо змінні *lb* і *hb* (скорочення від "low bound" і "high bound" – нижня й верхня межа). Cпочатку відрізків немає, і перетин порожній – виразимо це ініціалізацією *lb*=1, *hb*=0, тобто відрізком з неможливими межами. Перший відрізок має стати значенням [*lb*;*hb*]. Потім у циклі вводяться інші відрізки та обчислюється перетин:

lb=1; hb=0;

writeln('задайте дійсні кінці відрізка:'); readln(a, b);

**if** a>b **then**

**begin** lb := b; hb := a **end**

**else begin** lb:=a; hb:=b **end**

{прочитано перший відрізок}

{далі читаються інші та обчислюється їх перетин}

**while** **not** eof **do**

**begin**

writeln('задайте дійсні кінці відрізка:'); readln(a, b);

**if** a>b **then**

**begin** t:=a; a:=b; b:=t **end**;

**if** a>lb **then** lb:=a;

**if** b<hb **then** hb:=b;

**if** lb>hb **then break**

**end**;

{введення закінчено або перетин порожній}

**if** lb>hb **then**

writeln('перетин порожній')

**else** writeln('перетин: [', lb, ';', hb, ']')

Оформлення програми залишаємо вправою.