**ПАСКАЛЬ: ОБРОБКА ТЕКСТІВ**

ПОШУКОВА РОБОТА

**1. Особливості організації текстів**

У мові Турбо Паскаль для подання текстів на зовнішніх носіях означено спеціальний тип файлів з ім’ ям **TEXT**. Файли цього типу будемо називати файлами-текстами.

Елементами файлів-текстів є символи, але тип text відрізняється від типу **file** **of** char наявністю *спеціальних* *підпрограм*, пов’ язаних саме з розбиттям на рядки. Розглянемо деякі з цих підпрограм.

Спосіб, у який задається кінець рядка фізичного файла, залежить від файлової системи, і можна навіть його й не знати. Достатньо розглядати рядок файлової змінної як послідовність, що закінчується спеціальним символом. Він позначається словом **eol**. Рядок, складений лише цим символом, називається *порожнім*.

*Слова* ***eol*** *немає в мові Паскаль, і в програмах воно не вживається*.

Визначити, чи доступний кінець рядка **eol** у файлі під час виконання програми, можна за допомогою функції **EOLN**. З виклику

eoln(f)

повертається значення **true**, якщо доступний **eol**, і **false** у противному разі. У системі Турбо Паскаль ця функція повертає значення **true**, коли доступним є символ chr(13). Цей символ, власне, й задає кінець рядка. У текстах, створених "під DOS", за цим символом, як правило, іде chr(10), що позначає "переведення рядка". Позначимо ці символи як [13] і [10]. Ось приклад тексту, який говорить сам за себе:

**ЦЕЙ ТЕКСТ[13][10]СКЛАДАЄТЬСЯ[13][10][13][10]**

**З П’ЯТИ РЯДКІВ,[13][10]З ЯКИХ ТРЕТІЙ ПОРОЖНІЙ[13][10]**

Звичайно, приємніше читати цю послідовність у вигляді

**ЦЕЙ ТЕКСТ[13][10]**

**СКЛАДАЄТЬСЯ[13][10]**

**[13][10]**

**З П’ЯТИ РЯДКІВ,[13][10]**

**З ЯКИХ ТРЕТІЙ ПОРОЖНІЙ[13][10]**

Для визначення, чи прочитано текст, викликається функція **EOF**. У текстах, на відміну від типізованих файлів, ознака кінця файла може задаватися не лише засобами операційної системи, але й спеціальними символами в самому фізичному файлі. Для програм, написаних мовою Турбо Паскаль, таким символом є chr(26). Наприклад, наведений текст у фізичному файлі насправді закінчувався б не символами [13][10], а [13][10][26]. Якщо в тексті доступний символ chr(26), то з виклику функції eof(f) повертається **true**, за іншого символу – **false**.

Ім’ я input можна не вказувати у викликах та замість eof(input) чи eoln(input) писати eof чи eoln.

**2. Записування в текст**

Описання роботи з текстами, що є файлами на диску комп’ ютера, почнемо з записування, оскільки воно має набагато менше "підводних каменів", ніж читання.

Нехай далі f є ім’ ям файлової змінної типу text. Текст, як і файли інших видів, установлюється в початковий стан для запису викликом rewrite(f). Після цього файл стає порожнім. Для того, щоб дописувати до вже існуючого тексту, треба установити його в ***початковий стан для дописування*** викликом процедури **APPEND**:

append(f).

Після цього всі символи тексту, крім символу кінця файла, залишаються в тексті, і нові символи будуть дописуватися за останнім із них.

Запис у текст задається викликами процедур **WRITE** і **WRITELN**, першим аргументом у яких є ім’ я файлової змінної. При виконанні виклику

write(f, *вираз-базового-типу-або-рядок*)

спочатку обчислюється значення виразу. За цим значенням створюється *послідовність символів, що його зображає*, і копіюється в текст f. Наприклад, виклик

write(f, trunc( sqrt(9999) ) )

задає дописування в файл двох символів '9' і '9', що задають число 99.

У виклику можна записати одразу кілька виразів через кому:

write(f, *вираз1*, *вираз2*, ...).

Наприклад, за x=2 виклик

write(f, 'x=', x, '; x\*\*2=', sqr(x))

задає виведення в файл послідовності символів x=2; x\*\*2=4. До речі, такий виклик виконується як послідовність викликів

write(f, *вираз1*); write(f, *вираз2*); ...

Процедура writeln відрізняється лише тим, що за виклику

writeln(f, *список-виразів-базових-типів-або-рядків*)

за останньою сталою в текст додається **eol** (у Турбо Паскаль – [13][10]). Список виразів тут може бути порожнім – тоді лише додається **eol**.

У викликах процедур write і writeln після виразу через двокрапку можна задати ***ширину поля* *W*** під запис значення виразу, наприклад,

write(f, sqr(x):2).

Тут *W*=2. Нехай для запису значення виразу потрібно *L* символів. В останньому виклику *L*=1 за x<4, *L*=2 за 3<x<10, *L*=3 за 9<x<34 тощо. Якщо *L W*, то перед символами значення додаються *W*-*L* пропусків; а якщо *L*>*W*, то виводяться всі символи. Таким чином, за x=3 друкується пропуск і 9, а за x=100 – всі п’ ять символів 10000.

Після виразу дійсного типу можна також указати кількість *N* цифр дробової частини, що виводяться після десяткової крапки, наприклад,

write(f, sqrt(x): 7 : 3).

Якщо *N* указано, то виводиться стала з фіксованою крапкою та *N* цифрами після крапки, інакше – нормалізована з порядком. В даному разі за x=2 виводиться два пропуски та 1.414. Остання цифра є результатом округлення. Деякі інші деталі можуть визначатися системою програмування.

**Приклад 1.** За виклику

writeln (f, 'СЛОН', 'собака':1, 'кіт':5)

у текст виводиться рядок

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **С** | **Л** | **О** | **Н** | **с** | **о** | **б** | **а** | **к** | **а** |  |  | **к** | **і** | **т** | **[13]** | **[10]** |

за виклику

write (f,1.1239:1:3, 0.11234567 :21)

– символи

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **.** | **1** | **2** | **4** |  | **1** | **.** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **0** | **0** | **0** | **0** | **0** | **E** | **-** | **0** | **0** | **0** | **1** |

за виклику

writeln (f, 123.499999:1:0,-12345.1234567:12)

– рядок

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** |  |  | **-** | **1** | **.** | **2** | **E** | **+** | **0** | **0** | **0** | **4** | **[13]** | **[10]** |

**Приклад 2.** Створити текст можна буквально "власними руками", набираючи його рядки на клавіатурі у відповідь на запрошення при виконанні такої програми:

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

**var** f:text; s: string;

**begin**

assign(f, 'myfile.txt'); rewrite(f);

writeln ( 'Наберіть символи рядка й натисніть Enter:' );

**while** **not** eof ( input ) **do**

**begin**

readln( s ); writeln (f, s );

writeln ( ' Наберіть символи рядка й натисніть Enter:' );

**end**;

close(f)

**end**.

Для задання кінця тексту на клавіатурі треба замість набирання чергового рядка натиснути на Ctrl-Z. І не забути про Enter.

Зазначимо, що при закриванні тексту за процедурою close до нього додається символ кінця chr(26).

**Задачі**

**1.** Переробити програму з прикладу 14.2 так, щоб спочатку з клавіатури задавалося ім’ я створюваного файла.

**2.** Створити текст із табличкою степенів числа 2 від 1 до

а) 30 (скористатися змінними типу Longint);

б) 63 (скористатися змінними типу Compound).

**3. Читання числових сталих із тексту**

Тут ми розглянемо читання з тексту лише числових сталих, оскільки *символи та рядки читаються зовсім інакше*.

Для читання числових сталих зручно скористатися процедурою **READ** – *її* *виконання в разі читання текстів цілком відрізняється від читання типізованих файлів*. Її виклик, як і для читання типізованих файлів, має вигляд

read(f, *список-імен-змінних*),

де f – ім’ я тексту, а змінні мають числові типи. Для безпомилкового виконання виклику файл повинен містити послідовність сталих, типи яких відповідають типам змінних списку.

***Ціла*** ***стала*** – це послідовність цифр, можливо, зі знаком '+' чи '-' на початку, яка задає ціле число з носія відповідного цілого типу, причому між знаком та першою цифрою в тексті не може бути пропусків. ***Дійсна*** ***стала*** – це послідовність цифр та інших символів із структурою сталих мови Паскаль, наприклад, 1.1, 2., .99, 1e-3, -2.73E+02. Кожна ціла стала також може розглядатися як дійсна.

Числові сталі в текстах відокремлюються пропусками в довільній кількості. *Символи табуляції та кінця рядка також будемо називати пропусками*.

Виклик read(f, X) за цілого чи дійсного x виконується так. З тексту читаються пропуски, а за ними символи сталої до найближчого пропуска. Доступним після читання сталої буде перший пропуск за нею. Якщо символи справді утворюють сталу відповідного типу, то за ними обчислюється значення й присвоюється змінній. За дійсного X у тексті може бути й ціла стала – за нею обчислюється відповідне дійсне значення. Наприклад, за цілою сталою 99 дійсне значення 99.0.

Символи можуть не утворювати сталої відповідного типу – тоді виникає помилкова ситуація й виконання програми аварійно завершується. Наприклад, помилковими є послідовності символів - 2 (тут пропуск між знаком і цифрою), 12345m або 123- (присутні нецифрові символи там, де їх не може бути), або 13., коли читається значення цілої змінної.

Зазначимо, що системи програмування забезпечують засоби обробки помилкових ситуацій та запобігання аварійного закінчення. Але ми цього тут не висвітлюємо.

Система програмування Турбо Паскаль має таку особливість. *Якщо доступний кінець файла, то спроба читати число не завершується аварійно, а відповідна змінна набуває значення 0!*

Читання сталих базових типів за процедурою **READLN** аналогічне процедурі read. Відмінність її в тім, що після читання останньої сталої всі символи тексту, що залишилися до найближчого **eol**, пропускаються разом із ним. Доступним стає перший символ наступного рядка тексту. Список імен змінних може бути порожнім; у такому разі виклик readln(f) задає пропуск, тобто читання без зберігання, поточного рядка тексту.

**Приклад**. У тексті записано такі символи:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** |  | **2** |  | **3** | **[13]** | **[10]** |
|  | **5** | **5** | **[26]** |  |  |  |

Нехай x, y, z, t – цілі змінні. Читання read(f, x, y); read(f, z, t) надасть їм значень 1, 2, 3, 55 відповідно, читання readln(f, x, y); read(f, z, t) – 1, 2, 55, 0, читання readln(f, x); readln(f, y, z, t) – 1, 55, 0, 0.

Тепер розглянемо читання числових послідовностей, записаних у текстах. *Найбільш природним є таке подання послідовності вхідних даних, у якому кінець послідовності визначається кінцем файла*. У цьому випадку читання описується за допомогою функції eof у циклі вигляду

**while** **not** eof(f) **do**

**begin**

read(f, v);

*обробка значення змінної v*

**end.**

З виклику eof(f) повертається значення **true**, якщо доступним є кінець файла. За такої організації читання слід забезпечити, щоб між останньою числовою сталою та кінцем файла не було порожніх символів. Якщо цього не зробити, то в системі Турбо Паскаль буде прочитано зайве нульове значення.

Якщо файл порожній, то перше ж виконання виклику eof(f) породжує **true**, і тіло циклу не виконується жодного разу. Змінна v залишається зі старим значенням. А якщо значення їй не було присвоєно, то можливі непередбачувані помилки.

Звернімо увагу, що читання в циклі відбувається після того, як із виклику функції eof повернулося значення **false**, тобто кожному читанню передує перевірка, чи не є файл прочитаним. Ми радимо завжди записувати програми так, що при їх виконанні *спочатку перевіряється, що з файла можна читати, і лише тоді читається*.

**Приклад 3.** У тексті записано дійсні числа, розділені порожніми символами. Обчислити їх середнє арифметичне.

Кількість чисел нічим не обмежена, тому скористаємося циклом із перевіркою ознаки кінця тексту. За порожнього тексту друкується 0:

n:=0; A:=0;

**while** **not** eof(f) **do**

**begin**

read(f, x); n:=n+1;

A:=A+x; {}

**end**;

**if** n>0 **then** A:=A/n;

writeln(A).

Зауважимо, що сума чисел у тексті може бути непредставною навіть у типі real, хоча їх середнє не більше максимального з них. Середнє арифметичне варто обчислювати інакше.

Отже, замість оператора A:=A+x у циклі напишемо оператор

A:=(n-1)/n\*A+x/n,

що виконується за n>0, і вилучимо оператор **if** n>0 **then** A:=A/n.

**Задачі**

**3.**\* Яких значень набудуть дійсні змінні a, b, c, d і який символ стане доступним після виконання викликів

а) read(f, a, b, c, d),

б) readln(f, a, b); read(f, c, d),

в) readln(f, a, b); readln(f, c); readln(f, d),

якщо з доступного символу в тексті починається послідовність

1[13][10] -2.9 +13[13][10]2000 777[13][10][26]

**4.**\* Написати програму підрахунку кількості рядків у тексті.

**5.**\* Результат лижника в перегонах задається трійкою цілих у одному рядку тексту: його стартовим номером та кількістю хвилин і секунд. Кількість рядків із результатами необмежена. Після кожного результату треба вивести на екран поточну десятку найкращих (якщо результатів менше, то подати наявні).

**6.** Є два тексти, у кожному рядку яких записано натуральне число, і послідовності цих чисел неспадні. Записати в третій текст неспадну послідовність чисел, що є результатом злиття двох заданих. Числа у вихідному тексті вивести по 10 на рядок (останній рядок може бути неповним). Наприклад, за заданих послідовностей (2, 2, 4, 6), (1, 3, 6, 7) у текстах створюється послідовність (1, 2, 2, 3, 4, 6, 6, 7).

**4. Особливості читання символів і рядків із тексту**

При виконанні виклику read(f, X) за змінної X типу char їй присвоюється доступний символ тексту, яким би він не був, а доступним стає наступний за ним. Виключенням є випадок, коли доступний кінець файла: значенням X стає символ chr(26), і він же залишається доступним. Отже, *при читанні символів пропуски, табуляції та кінці рядків обробляються так само, як і всі інші символи*.

**Приклад.** Розглянемо виконання такої програми:

**program** eofad;

**const** eo=chr(26);

**var** f : text; ch : char;

**begin**

assign(f, 'eofad.dat'); rewrite(f);

writeln(f, 'abc', eo, 'qq'); {запис 6 символів у файл}

close(f);

reset(f);

**while** **not** eof(f) **do** {у тексті є доступний символ, і він не є chr(26)}

**begin**

read(f, ch); {доступний символ файла читається в змінну ch}

write(ch)

**end**;

close(f);

readln;

**end**.

За її виконання на екрані буде надруковано лише **abc**. Справа в тім, що після того, як прочитано символ **c**, доступним стає chr(26). Далі виконується виклик eof(f), з нього повертається значення **true**, і виконання циклу закінчується.

Якщо X рядкового типу, то при виконанні виклику read(f, X) символи до найближчого **eol** читаються та присвоюються елементам X; доступним стає **eol**, тобто chr(13) в системі Турбо Паскаль. Якщо символів тексту до **eol** більше, ніж уміщається в X, то X заповнюється до кінця, і доступним стає перший символ за тими, що прочитано в X.

*Якщо перед читанням рядка X доступний* ***eol****, то він залишається доступним*, *а значенням X стає порожній рядок. Ось чому ми рекомендуємо не застосовувати процедуру read при читанні рядків*.

*Особливо небезпечним є застосування read при введенні рядків у циклі*. Якщо після введення рядка доступним символом стане **eol**, то за подальших уведень рядків вони одержуватимуть значення '' (порожній рядок), **eol** залишиться доступним і виконання програми може "зациклитися". Наприклад, при виконанні циклу вигляду

**while** **not** eof(f) **do**

**begin**

read (f, x); {***???***}

*обробка x*

**end**,

де *x* – змінна-рядок, читаються лише символи першого рядка тексту, після чого кінець рядка залишається доступним "назавжди". Цикл собі виконується, комп’ ютер "висить", а користувач програми починає нервувати й лаятися.

Коли при читанні рядка доступний кінець тексту, він аналогічно залишається доступним, а змінній-рядку присвоюється значення '' (порожній рядок).

*Читання символів і рядків за процедурою readln аналогічне процедурі read, але після читання символів решта рядка тексту пропускається*, тобто доступним стає символ, наступний за найближчим **eol**. *Радимо задавати читання рядків за допомогою процедури readln*.

Список імен змінних у виклику може бути порожнім; у такому разі при виклику readln(f) пропускається поточний рядок тексту.

**Приклад 4.** Нехай діють означення

f : text; c1, c2 : char; s1, s2 : string[4],

а **eol** явно позначає кінець рядка в тексті, з яким зв’ язано f:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **t** | **r** | **u** | **e** | **eol** |
| **f** | **a** | **l** | **s** | **e** | **eol** |
| **eol** |  |  |  |  |  |

Після виклику

read (f, c1, c2, s1, s2 )

змінні набудуть значень: c1=' ', c2='t', s1='rue' та s2=''; доступним буде перший із указаних **eol**. Якщо далі виконати read(f, c1, c2), то значеннями с1 і c2 будуть chr(13) і chr(10), а доступним стане символ f.

За такого самого тексту виклик

readln (f, s1, c1, c2, s2 )

надає змінним значень s1=' tru', c1='e', c2=chr(13), s2=chr(10)+'fal'. Символи 's' і 'e' пропускаються, і доступним стає третій **eol**, тобто chr(13).

**Задачі**

**7.**\* Указати значення, одержані пpи читанні тексту f змінними

**var** c, c1 : char; i : integer; r : real; s, s1, s2 : string,

пpи виконанні послідовності опеpатоpів

read( f, c, i, r, s );

readln( f, s1 ); readln( f, s2 );

readln( f, c1 ),

якщо текст, починаючи від доступного символу, містить послідовність символів:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **eol** | 1 |  | 2 | . | 1 | **eol** | l | e | s | s | o | n | **eol** | x | **eol** | y | **eol** | **eol** |

Указати також доступний символ тексту після читання.

**5. Читання тексту з рядками обмеженої довжини**

У багатьох текстах довжина рядків неоднакова, але *обмежена*, як правило, числом, що не перевищує 255 – максимальну довжину рядків типу string. Використання змінних цього типу дозволяє дуже просто описувати читання текстів.

Алгоритми розв’ язання задач із обробки таких текстів часто мають загальний вигляд

**while** **not** eof(f) **do**

**begin**

readln(f, s); {s має тип string}

*обробка рядка s;*

**end**.

**Приклад 5.** Написати процедуру копіювання тексту за умови, що рядки тексту мають довжину не більше 80, а рядки порожні або такі, що містять лише пpопуски, не копіюються.

Після читання чергового рядка початкового тексту треба визначити, чи є в ньому хоча б один символ, відмінний від пропусків. Тільки в цьому випадку він копіюється в текст g:

**procedure** cpnonemp(**var** f, g : text);

**var** s : string[80];

k : integer; emp : boolean;

**begin**

**while** **not** eof(f) **do**

**begin**

readln(f, s); k:=1; emp:=**true**;

**while** (k<= length(s)) **and** emp **do**

**if** s[k]<>' ' **then** emp:=**false**

**else** k:=k+1;

**if** **not** emp **then** writeln(g, s)

**end**

**end**;

**Задача**

**9.** На вхід програми подається послідовність груп рядків тексту. Групи відділяються рядком із першим символом '='. За останньою групою такого рядка може не бути. Група може бути порожньою. Рядки в групах непорожні й містять числа, відокремлені пропусками в довільній кількості. Всі рядки разом із наступними повідомленнями треба виводити в інший текст. Якщо в рядку 3 числа, то слід визначити, чи задають вони довжини сторін трикутника, і надрукувати повідомлення "трикутник" або "не трикутник". За іншої кількості чисел треба повідомити: "помилкові дані" (див. задачу 12.17). Для кожної групи слід надрукувати слова "кількість рядків: " і число її рядків, що задають трикутники, та рядок, яким група відокремлена від наступної. Для останньої групи такий рядок не друкується. Наприклад, за послідовності рядків

**2 3 4**

**1 3 5**

**=123=**

**=**

**1**

**1 2 3 4**

**= = =**

треба надрукувати в іншому тексті

**2 3 4**

**трикутник**

**1 3 5**

**не трикутник**

**кількість трикутників: 1**

**=123=**

**кількість трикутників: 0**

**=**

**1**

**помилкові дані**

**1 2 3 4**

**помилкові дані**

**кількість трикутників: 0**

**6. Посимвольне читання тексту**

**Приклад 6.** У тексті з рядками необмеженої довжини записано слова в латинському алфавіті, довжина яких не перевищує 255. Слова відокремлюються пропусками в довільній кількості та з рядка на рядок не переносяться. Треба визначити кількість повторень першого слова в подальшому тексті.

Для розв’ язання задачі треба

*прочитати перше слово (якщо воно взагалі є в тексті), а далі по одному читати слова й порівнювати їх із першим, за рівності збільшуючи значення лічильника.*

Слово є ***лексичною*** ***одиницею***, або ***лексемою*** тексту, тобто такою послідовністю, що має самостійне значення, тому функцію читання слова з тексту назвемо getlex (взяти лексему). Ось її заголовок:

**function** getlex(**var** f:text; **var** lex:string):boolean.

З її виклику повертається ознака наявності слова в частині тексту, прочитаній за її виклику. Слово зберігається як значення параметра-змінної lex (лексема), а коли його в решті тексту немає, значенням стає порожній рядок. Отже, нехай s1, s2 – рядки, nrep – лічильник повторень у такому алгоритмі:

nrep:=0;

**if** getlex(f, s1) **then**

**begin**

**while** getlex(f, s2) **do**

**if** s1=s2 **then** nrep:=nrep+1;

writeln(nrep)

**end**

**else** writeln('у тексті немає слів');

Щоб прочитати слово, треба

*від поточного доступного символу прочитати пропуски та кінці рядків, що передують слову, та запам’ ятати його символи в рядковій змінній*.

Для визначення, чи є символ латинською літерою, скористаємося функцією isletter:

**function** isletter(c : char) : boolean;

**begin**

isletter := ('a'<c) **and** (c<'z') **or** ('A'<c) **and** (c<'Z')

**end**;

Запишемо функцію getlex. Коли під час її виклику завершується виконання першого циклу, можуть бути істинними обидві умови, eof(f) та isletter(ch). Це можливо, якщо останній символ тексту є водночас першою літерою слова. У цьому разі символ дописується до порожнього слова.

**function** getlex(**var** f : text; **var** lex : string) : **boolean**;

**var** ch : char; isl : boolean;

**begin**

ch:=' '; lex:=''; getlex:=**false**;

**while** **not** eof(f) **and** **not** isletter(ch) **do**

read(f, ch);

{eof(f) or isletter(ch)}

**if** isletter(ch) **then**

**begin** {створення рядка-лексеми}

getlex:=**true**; lex:=lex+ch; isl:=**true**;

**while** **not** eof(f) **and** isl **do**

**begin**

read(f,ch);

**if** isletter(ch) **then** lex:=lex+ch

**else** isl:=**false**

**end**;

{eof(f) or not isl}

**end**;

**end**;

Тіло складеного оператора, що задає створення рядка-лексеми й виконується після того, як умова isletter(ch) стає істинною, можна записати за допомогою оператора **repeat**-**until:**

getlex:=**true**; isl:=**true**;

**repeat**

lex:=lex+ch;

**if** **not** eof(f) **then**

**begin**

read(f,ch); isl:=isletter(ch);

**end**

**else** isl:=**false**

**until** (**not** isl);

У функції getlex локальна змінна ch зберігає останній прочитаний символ тексту. Функцію побудовано так, що за виконання її виклику читаються не тільки символи найближчого слова, а й символ, наступний за словом. Отже, після виконання виклику цей символ втрачається. Але, оскільки слова відокремлюються принаймні одним порожнім символом, можна на початку функції присвоїти пропуск змінній ch. Він замінює символ, втрачений за попереднього виклику.

Описана особливість функції getlex є скоріше її недоліком, ніж перевагою. Краще означити змінну ch як ***локальну*** в функції, але ***статичну***, значення якої зберігалися б і після виконання виклику функції (див. підр. 8.6). В даному разі слід було б у функції getlex означити змінну ch як

**const** ch : char = ' '

та вилучити присвоювання ch:=' ' на початку тіла функції.

Написання програми з усіма означеннями та операторами залишаємо вправою.

**Приклад 7.** Текст із рядками необмеженої довжини містить слова в латинському алфавіті. Довжини слів не більше 255, вони відокремлюються пропусками в довільній кількості та з рядка на рядок не переносяться. Треба надрукувати слова тексту, що містять задане з клавіатури слово як підслово разом із номерами рядків, де вони розташовані.

Скористаємося алгоритмом і підпрограмами з попереднього прикладу, дещо їх змінивши. Влаштуємо в програмі "занудне" читання слова з клавіатури в змінну-рядок s1 із перевіркою, чи є воно непорожнім, і всі символи його – латинські літери. Цю перевірку повинна задавати функція isword.

Далі з тексту читаються слова в змінну-рядок s2 та за рівності їх із s1 друкуються разом із номером рядка тексту nlin. Це змінна програми з початковим значенням 1, яке збільшується за виконання вікликів функції getlex1.

**repeat** readln(s1); **until** isword(s1);

nlin:=1;

**while** getlex1(f, s2) **do if** pos(s1, s2)<>0 **then** writeln(s2, ' ', nlin);

Функція getlex1 відрізняється від getlex збільшенням глобальної у ній змінної nlin. Воно відбувається, коли читається символ, що задає кінець рядка. Ми "забудемо" про те, що цим символом у мові Турбо Паскаль є chr(13), і скористаємося функцією eoln. Зверніть увагу, що її виклик передує читанню символу, оскільки за його виконання аналізується доступний, ще не прочитаний, символ тексту:

**function** getlex1(**var** f:text; **var** lex:string):**boolean**;

**const** ch : char=' '; **var** isl : boolean;

**begin**

lex:=''; getlex1:=**false**;

**while** **not** eof(f) **and** **not** isletter(ch) **do**

**begin**

**while** eoln(f) **and** **not** eof(f) **do**

**begin** nlin:=nlin+1; readln(f) **end**;

read(f, ch);

**end**;

{eof(f) or isletter(ch)}

**if** isletter(ch) **then**

**begin**

getlex1:=**true**; lex:=lex+ch; isl:=**true**;

**while** **not** eof(f) **and** isl **do**

**begin**

**if** eoln(f) **then** **exit**;

read(f,ch);

**if** isletter(ch) **then** lex:=lex+ch

**else** isl:=**false**

**end**;

{eof(f) or not isl}

**end**;

**end**;

Написання повної програми також залишаємо вправою.

**Приклад 8.** Коментар – це послідовність символів, що починається символами '(\*', закінчується символами '\*)' і не містить '\*)' усередині. Написати програму читання та копіювання тексту з вилученням коментарів.

У попередніх прикладах ми вже бачили, що обробка прочитаного символу залежить від його місця в тексті. Наприклад, у функції getlex порожні символи просто читалися, а символи слова дописувалися до рядка. Аналогічно й тут: символи зовні коментаря повинні копіюватися в інший текст, а символи самого коментаря – ні. Але коли прочитано дужку '(', то невідомо, чи є вона початком коментаря, чи ні. Якщо наступний за нею символ відмінний від '\*', то дужку треба копіювати, а якщо це '\*' – ні. Крім того, дужка всередині коментаря не копіюється. Так само, якщо читаються символи коментаря, то обробка закриваючої дужки ')' залежить від того, чи був попередній символ '\*', чи ні.

Придивившися уважніше до текстів із коментарями, можна зрозуміти, що можливі чотири випадки, у кожному з яких останній прочитаний символ обробляється по-своєму.

1. Коментар ще не починався або вже закінчився. Тут символ, відмінний від '(', копіюється, а '(' може означати початок коментаря, тому її копіювати поки що зарано. Натомість треба запам’ ятати, що прочитано дужку, після якої символи обробляються згідно наступного пункту.

2. Перед цим було прочитано '(', тобто коментар, можливо, починається. Нова така ж дужка означає, що попередня дужка була зовні коментаря і її треба скопіювати, а нову запам’ ятати. Символ '\*' означає, що ми вже потрапили в коментар, і попередню дужку копіювати не треба. За іншого символу попередня дужка разом із новим символом копіюється, і ми залишаємося зовні коментаря.

3. Якщо ми потрапили в коментар, то символ '\*' може означати, але не обов’ язково, "початок закінчення" коментаря (наступний випадок). Всі інші символи, в тому числі й ')', ніяк не обробляються.

4. Перед цим у коментарі була прочитана '\*', тобто коментар, можливо, починає закінчуватися. Нова '\*' означає новий "початок кінця" коментаря. Символ ')' означає, що коментар закінчено, а будь-який інший – що коментар продовжується.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ch**  **стан** | **(** | **\*** | **)** | **інший**  **символ** |
| **out** | стан:=bgn; | видати ch; | видати ch; | видати ch; |
| **bgn** | видати (; | стан:=incm; | видати (;  видати ch;  стан:=out | видати (;  видати ch;  стан:=out |
| **incm** |  | стан:=bgend; |  |  |
| **bgend** | стан:=incm; |  | стан:=out; | стан:=incm; |
|  |  |  |  |  |

Введемо поняття "***стан тексту після останнього прочитаного символу***". У нашому випадку такими станами є "зовні коментаря", "початок коментаря", "всередині коментаря", та "початок кінця коментаря". Стан тексту цілком визначається тим станом, який був раніше, та останнім символом. Пункти 1-4 описують обробку символів, відповідну цим станам, а також зміни стану.

Позначимо вказані стани відповідно словами out (зовні), bgn (початок), incm (всередині коментаря), bgend (початок кінця). Значенням останнього прочитаного символу ch може бути '(', '\*', ')' або інший символ. Подамо дії, описані в пунктах 1-4, у вигляді таблиці на рис.14.1. Стовпці відмічено символами, рядки – станами. У клітині на перетині рядка й стовпця вказано зміну стану та інші дії, відповідні цим стану й останньому символу. Зміна станів подається присвоюванням, відсутність якого означає, що стан не міняється.

Зміст таблиці подамо також у вигляді діаграми станів (рис.14.2). Стрілки показують зміну станів залежно від останнього прочитаного символу. Кожну стрілку відмічено дробом: угорі вказано символ, унизу – його обробку. Символ  позначає довільний символ, відмінний від '(' ,  – відмінний від '(' та '\*',  – від '\*' та ')'.

Початковим станом тексту природньо вважати out. Копіювання тексту з вилученням коментарів можна *імітувати* пересуванням по діаграмі та виконанням дій, указаних на стрілках. На кожному кроці імітації читається черговий символ тексту і згідно діаграми за ним та поточним станом визначаються дії та зміна стану.

За наведеними таблицею чи діаграмою неважко побудувати програму копіювання тексту з вилученням коментарів. У програмі переписано зміст таблиці за допомогою **case**-операторів. Нехай змінна ch зберігає останній прочитаний символ, а g є ім’ ям тексту-копії. Означимо тип-перелік станів:

**type** States=(out, bgn, incm, bgend)

та змінну q цього типу. Спочатку q:=out. А далі

**while** **not** eof(f) **do**

**begin**

read(f, ch);

**case** q **of**

out: **case** ch **of**

'(': q:=bgn

**else** write(g, ch)

**end**;

bgn: **case** ch **of**

'(': write(g, '(');

'\*': q:=incm

**else**

**begin**

write(g, '(', ch); q:=out

**end**;

**end**;

incm: **case** ch **of**

'\*': q:=bgend

**end**;

bgend: **case** ch **of**

'\*': ;

')': q:=out

**else** q:=incm

**end**;

**end**; {case q of}

**end**; {while not eof(f) }

Як бачимо, виконання наведеного циклу відповідає описаній вище імітації діаграми. Оформлення програми залишаємо вправою.

**Задачі**

**9.**\*"Бінарський алфавіт" складено латинськими буквами A та B. Слова "бінарської мови" задаються так:

1) порожнє слово є словом "бінарської мови";

2) якщо послідовність символів X – слово "бінарської мови", то послідовність символів AXB також є словом "бінарської мови";

3) якщо послідовності символів S і T – слова "бінарської мови", то послідовність символів ST також є словом "бінарської мови".

Написати програму визначення за заданою послідовністю символів, чи є вона словом "бінарскої мови". Послідовності символів A, B задано по одній на рядок тексту. Довжини рядків обмежені лише найбільшим значенням типу longint. Всі результати – символи 1 або 0 ("так" або "ні") – виводяться в один рядок іншого тексту.

Написати програму генерації тексту з рядками, довжина яких обмежена лише значенням maxint типу longint. Текст повинен містити рядки з символів A та B, на яких слід перевірити програму визначення слів "бінарської мови".

**10.**\* Написати підпрограму читання найближчого ідентифікатора тексту (його в тексті може і не бути). Вважати, що довжина ідентифікатора не більше 255.

**11.**\* Написати підпрограму читання найближчої цілої сталої, що повинна задавати число типу integer. Якщо число не представне в типі integer, то треба видати відповідне повідомлення (наявність сталої в тексті гарантується). Процедуру val не вживати.

**12.**\* Літералом є послідовність символів у апострофах, коментарем – послідовність символів, що починається '(\*', закінчується '\*)', не містить '\*)' усередині. Написати програму копіювання тексту з вилученням коментаpів за умови, що коментар всередині літерала як коментар не розглядається і не вилучається.

**13.** Створити тексти для перевірки правильності програм для задач 14.8–14.12.

**7. Використання рядків для виведення в текст**

**Приклад** **9.** Редактор Word символом chr(13) задає розбиття текстів не на рядки, а на абзаци. З точки зору редактора ДОС або Турбо Паскаль такий абзац є рядком, причому, як правило, дуже довгим, і читати його незручно. Напишемо програму копіювання тексту з довгими рядками в текст із рядками довжини не більше ніж, наприклад, 80.

*Уточнення*. Пропуски, кінці рядків і символи табуляції будемо називати пропусками. Можна вважати, що послідовності символів, відмінних від пропусків, у тексті, тобто слова, мають довжину не більше 80. Слова в тексті відокремлюються пропусками в довільній кількості. У новому тексті між словами рядка повинен бути один пропуск, тобто ***текст має ущільнюватися***.

У загальному вигляді розв’ язання задачі полягає в тім, що *з тексту по одному "витягуються слова" й записуються в новий текст*.

Якщо читання слова задати функцією getlex, яка повертає ознаку наявності слова, а запис слова – процедурою putlex, то головним у програмі буде цикл вигляду

**while** getlex **do** putlex.

Отже, прочитане слово треба записати в новий текст. Але замість цього запишемо його в допоміжний рядок довжини 80, який назвемо "рядок слів".

*Слова накопичуються в рядку слів, і коли чергове слово вже не вміщається в ньому, він записується окремим рядком у текст за допомогою процедури writeln. Після цього нове слово записується в рядок слів із його початку. Наприкінці, коли початковий текст уже прочитано, треба не забути переписати рядок слів у новий текст.*

Наведений алгоритм уточнюється далі у вигляді процедури putlex.

Для того, щоб "витягнути" слово з тексту, треба *прочитати пропуски й накопичити в рядку-слові символи-не пропуски, що йдуть поспіль до наступного пропуска або до кінця тексту.*

Серед пропусків, що читаються, можуть бути кінці рядків. Перший із них означає, що треба переписати накопичений рядок слів у новий текст, а всі інші – що записати порожній рядок. І тільки після цього записувати нове слово з початку рядка слів. Таким чином, читаючи пропуски, треба підраховувати кінці рядків.

Нехай str – це ім’ я типу string[80]. Читання чергового слова уточнимо у вигляді наступної функції getlex. Символами-пропусками у ній вважаються символи табуляції chr(9), переведення рядка chr(10), нового рядка chr(13) і власне пропуск chr(32). Її останній параметр nume зберігає кількість кінців рядків, що передували знайденому слову. Ця кількість використовується на початку виконання процедури putlex.

**function** getlex(**var** f : text; **var** lex : str; **var** nume : integer) : boolean;

**const** empsym : **set** **of** char=[chr(9), chr(10), chr(13), chr(32)];

**var** c : char; inlex : boolean;

**begin**

lex:=''; inlex:=**false**; nume:=0;

**while** **not** eof(f) **and** **not** inlex **do**

**begin**

**if** eoln(f) **then**

**begin** inc(nume); readln(f) **end**

**else**

**begin** read(f, c); inlex:=**not** (c **in** empsym) **end**;

**end**;

**while** inlex **do**

**begin**

lex:=lex+c;

**if** **not** eof(f) **then**

**begin**

**if** eoln(f) **then** inlex:=**false**

**else begin** read(f, c); inlex:=**not** (c **in** empsym) **end**;

**end**

**else** inlex:=**false**

**end**;

getlex:=(lex<>'');

**end**;

Запис накопиченого рядка слів (глобальна змінна з ім’ ям buff) у текст задамо такою процедурою:

**procedure** outbuff(**var** f : text);

**begin**

writeln(f, buff); buff:=''; bp:=1;

**end**;

Як уже зазначалося, запис слова в рядок слів задається процедурою putlex:

**procedure** putlex(**var** f : text; lex : str; nume : integer);

**var** llx, k : integer;

**begin**

**if** nume > 0 **then**

outbuff(f);

**for** k:=1 **to** nume-1 **do**

writeln(f);

llx:=length(lex);

**if** (bp>1) **and** (bp+llx>mx) **then**

outbuff(f)

**else**

**if** buff<>'' **then**

**begin**

buff:=buff+' '; bp:=bp+1;

**end**;

buff:=buff+lex; bp:=bp+llx;

**end**;

Нарешті, програма копіювання тексту з перетворенням рядків має вигляд:

**program** f80;

**const** mx=80;

**type** str=string[mx];

**var** f, g : text;

**const** buff : str=''; bp : integer=1;

**var** lex : str; nume : integer;

**function** getlex(**var** f : text; **var** lex : str; **var** nume : integer):boolean;

...

**end**;

**procedure** outbuff(**var** f : text);

...

**end**;

**procedure** putlex(**var** f : text; lex : str; nume : integer);

...

**end**;

**begin**

assign(f, 'in.txt'); assign(g, 'out.txt');

reset(f); rewrite(g);

**while** getlex(f, lex, nume) **do**

putlex(g, lex, nume);

**if** buff<>'' **then** outbuff(g);

close(f); close(g);

**end**.

**Задачі**

**14.** Переробити програму f80 таким чином, щоб

а) порожні рядки початкового тексту не виводилися в новий;

б) при виведенні зберігалися всі пропуски, що є між словами початкового тексту.