Глава 10

Файли

У цій главі...

- Три різновиди файлів та три набори підпрограм їх обробки
- Створення та читання типізованих файлів
- Особливості читання текстів
- Буферизація та прискорення обробки файлів
- Читання й запис безтипових файлів

Там <...> стояв письмовий стіл, завалений бухгалтерськими книгами,
і довга канцелярська шафа з відкритими полицями. <...>
На полицях лежали пачки ордерів, перев'язані свіжою мотузкою.

І. Ільф, Є.Пєтров

10.1. Основи роботи з файлами

10.1.1. Фізичні файли та файлові змінні

Під *файлом* розуміють іменовану область на зовнішньому носії інформації. Файл, будучи областю на зовнішньому носії інформації, є *послідовністю байтів*. Одну й ту ж послідовність байтів можна розглядати та обробляти як послідовність або *значень певного типу*, або *символів із розбиттям на рядки*, або *байтів*.

Мова Паскаль не має операторів для роботи з файлами. Всю обробку задають за допомогою спеціальних засобів, зібраних у *систему введення-виведення*. Вам уже знайомі процедури `readln` і `writeln`.

У Паскаль-програмі файл задають іменем *файлової змінної*, тобто вона є представником файла в програмі. Тип у її оголошенні задає спосіб, за яким обробляється файл. Далі області на зовнішніх носіях називаються *фізичними файлами*, а під словом "файл" розуміють саме файлову змінну. У мові Турбо Паскаль є три основних різновиди файлів: типізовані, текстові та безтипові.

Типізований файл — це послідовність змінних певного скалярного або структурного (але не файлового) типу. Тип задають виразом вигляду **`file of`** *тип*. Наведемо приклади.

```
type Flchar = file of char;
    Student = record Name : String; Number : Integer;
              end;
     fStudent = file of Student;
var Fi, Fo : fStudent;
    FF     : Flchar;
```

Текст — це послідовність символів, які поділено на рядки. У мові Турбо Паскаль для файлів-текстів означено спеціальний тип з іменем `text`. Елементами цих файлів є символи, проте тип `text` відрізняється від типу **`file of`** `char`. У текстах є спеціальні символи, що задають

кінці рядків і кінець тексту. Тексти обробляються за допомогою спеціальних підпрограм, які не можна застосовувати до типізованих файлів, наприклад, `readln` та `writeln`.

Безтипові файли розглядаються як послідовності байтів і оголошуються за допомогою службового слова **`file`**. Наприклад, **`var`** `Fi, Fo :` **`file`**. Для них також означено специфічні підпрограми.

- Описані вище три різновиди файлів відрізняються не можливим представленням даних, а наборами підпрограм, застосовних до них.

Елементи масивів і записів можуть мати файлові типи. Наприклад, припустимі такі оголошення типів.

```
type Arrfil = array [1..2] of file of Student;
type Rcrfil = record data : text; temp : char end;
```

Типами елементів типізованих файлів *не можуть бути файлові типи* і типи, що містять файлові. Наприклад, такі оголошення типів недопустимі.

```
type FF = file of file of ...; {???}
type FF = file of text; {???}
type FF = file of array [1..2] of file of ...; {???}
```

Практично в усіх підпрограмах першим параметром є ім'я файлової змінної. Нехай нижче ім'я `f` позначає файлову змінну (хоча в реальних програмах краще давати файловим змінним змістовні імена, наприклад `workFile`, `inputFile` тощо).

Робота з файловою змінною починається зі *зв'язування* її з конкретним фізичним файлом. Для цього ідентифікатор файлової змінної та ім'я фізичного файла в операційній системі, або *зовнішнє ім'я*, задаються у виклику процедури `assign`, наприклад `assign(f, 'myfile.dat')`. Другий аргумент, тобто зовнішнє имя — це вираз рядкового типу. Наприклад, якщо рядки `s1` і `s2` мають значення `'c:\mydir\book'` і `'txt'`, то `assign(f, s1+'.'+s2)` зв'язує файлову змінну `f` із файлом `c:\mydir\book.txt`. Після виклику процедури `assign` фізичний файл можна позначати іменем файлової змінної.

Приклад 10.1. Нехай виконується програма `AssignDemo` з такими оголошеннями та операторами.

```
...
var f : text; fn, st : string;
...
begin ...
  fn := 'default.dat'; {default - "за узгодженням"}
  write('Якщо натиснути <ENTER>, f буде ');
  writeln('зв''язано з "default.dat". ');
  write('Для зміни зв''язування f введіть ім''я файла:');
  readln(st);
  if st <> '' then fn := st; assign(f, fn);
...
```

Якщо треба зв'язати `f` із файлом, зовнішнє ім'я якого "`default.dat`", натисніть <Enter>. Для іншого зв'язування наберіть на клавіатурі зовнішнє ім'я, наприклад, `newname.dat`, і натисніть <Enter>.

Задати зовнішнє ім'я файла можна і в *командному рядку* (*рядку виклику*) програми, наприклад, у такому вигляді.

```
AssignDemo newname.dat
```

Для забезпечення цієї можливості використовують функцію `ParamStr`, яка зчитує слова в рядку виклику, тобто послідовності символів, відмінних від пропуску. Назва програми `'AssignDemo'` повертається з виклику `ParamStr(0)`, `'newname.dat'` — з виклику `ParamStr(1)`. Подальші слова в командному рядку можна одержати з викликів `ParamStr(2)`, `ParamStr(3)` тощо.

Кількість слів у рядку виклику повертається з виклику функції `ParamCount` без аргументів. За допомогою цієї функції можна визначити, чи указав користувач програми `AssignDemo` ім'я файла в рядку виклику. Якщо так, файлова змінна зв'язується із заданим файлом, інакше — за узгодженням. Замість наведених операторів тіла програми `AssignDemo` розглянемо такі.

```
if ParamCount > 0
  then fn := ParamStr(1)
else fn := 'default.dat';
assign(f, fn)
```

□

- ✧ Файлові змінні з іменами `input` і `output` зв'язуються *неявно* (без указівки у програмі) із *стандартними файлами введення та виведення* — клавіатурою та екраном комп'ютера. Їх ще називають *стандартними пристроями введення-виведення*.
- ✧ Оголошувати явно імена `input` і `output` та зв'язувати їх з іншими фізичними файлами не заборонено, але не рекомендується. Якщо ім'я `input` зв'язано з іншим фізичним файлом і треба відновити його зв'язок із клавіатурою, то використовується виклик `assign(input, 'con')` (`con` — це скорочення від console, тобто консоль). Якщо треба зв'язати ім'я `output` із принтером, використовується виклик `assign(output, 'prn')` (prn — це принтер).
- ✧ Імена `input` і `output` можна не записувати у викликах підпрограм обробки файлів. Саме такими були виклики `readln` і `writeln` у попередніх главах.

Працювати з елементами зв'язаного файла можна тільки після його відкриття. Його задають процедурами, які представлено нижче. Проте насамперед опишемо *закриття файла* за допомогою процедури `close`: `close(f)`. Ця процедура не розриває зв'язку імені `f` із фізичним файлом, але введення та виведення за допомогою цього імені неможливі до наступного відкриття або нового зв'язування з подальшим відкриттям.

- Відкритий файл обов'язково треба закрити після використання. По-перше, це гарантує внесення змін у фізичний файл (якщо такі були), а, по-друге, в деяких інших системах програмування, наприклад у Visual Basic, дозволяє більш повний доступ до файла іншим процесам ще до завершення виконання всієї програми.
- Спроба закрити вже закриту або ще не відкриту файлову змінну призводить до аварійного завершення програми.

Отже, стандартний порядок дій із файловою змінною можна подати в такому вигляді.

```
зв'язування
відкриття
  обробка
закриття
нове відкриття або зв'язування та відкриття
  обробка
закриття
тощо
```

Перед тим, як представляти підпрограми відкриття та обробки файлів, уточнимо поняття файлової змінної, хоча і на абстрактному рівні.

Після зв'язування фізичний файл стає складовою частиною файлової змінної. Він розташований на зовнішньому носії, а не в оперативній пам'яті, тому його елементи — не звичайні змінні.

Основна особливість файла полягає в тому, що у будь-який момент виконання програми з усіх його елементів можна обробляти (читати або записувати) тільки один елемент; він називається *доступним елементом*. Інші в цей момент недоступні.

Номер доступного елемента в послідовності елементів файла є значенням спеціальної неявної змінної — *файлового вказівника* (рис. 10.1). Номер доступного елемента (значення вказівника) змінюється при виконанні підпрограм обробки файлів.

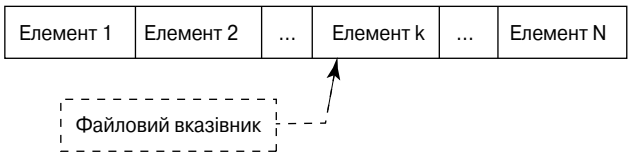

***Рис. 10.1.** Фізичний файл і файловий вказівник*

Основними діями з файлом є введення (читання) — копіювання значення доступного елемента в "звичайну" змінну програми, і виведення (запис) — копіювання значення в доступний елемент. Можливість введення та виведення задається *режимом доступу* до файла, або його *станом* — читання, запису, читання-запису або чогось іншого (наприклад, файл може не бути відкритим).

Отже, до складу зв'язаної файлової змінної входять послідовність елементів фізичного файла, номер доступного елемента й стан. Позначимо це як (F, i, s), де F — послідовність значень $<f_0, f_1, ..., f_n>$, i — номер доступного елемента, s — стан. Нумерація елементів у системі Турбо Паскаль починається з 0. Стани читання, запису та читання-запису позначимо буквами *R*, *W* та *RW* відповідно.

Розглянемо приклади. Вираз (<11, 12, 13, 14>, 1, *R*) позначає стан файла, у якому друге з послідовності чисел 11, 12, 13, 14 (з номером 1) є доступним для читання. Вираз (<>, 0, *W*) позначає порожній файл, до якого можна додати новий елемент (із номером 0) і записати в нього яке-небудь значення. Описані стани подано на рис. 10.2 (стрілки позначають файлові вказівники і вказують на доступні елементи).

***Рис. 10.2.** Стани файла*

✧ Кінець типізованого файла ніяк не задано в ньому самому. Спосіб визначення кінця файла залежить від операційної системи й тут не описується.

10.1.2. Послідовний запис у типізовані файли

Файл, призначений для запису, варто відкрити за допомогою процедури `ReWrite`. Після виклику `ReWrite(f)` значення `f` можна подати як (<>, 0, *RW*), тобто незалежно від попереднього стану `f` послідовність у файлі стає порожньою (див. рис. 10.2, *б*). Цю процедуру викликають для створення нового фізичного файла або відновлення старого зі знищенням попередніх даних. Після її виконання файл буде називатися встановленим у *початковий стан для читання-запису*.

Запис у файл задають процедурою `write`. Після виклику

`write(f, `*`вираз_що_іменує_змінну_типу_компонентів_файла`*`)`

значення змінної присвоюється доступному елементу файла, після чого вказівник доступного елемента зсувається на один елемент. Наприклад, розглянемо таку програму.

```
program ...
  var f : file of integer; x, y : integer;
begin
```

```
  ...
  rewrite(f); x := 2; y := 1;
  write(f, y); write(f, x);
  y := x*x; write(f, y);
  ...
end.
```

Після третього виклику `write` значенням `f` буде (<1, 2, 4>, 3, *RW*).

Взагалі, значення (<f_0, ..., f_{n-1}>, *n*, *RW*) файлової змінної `f` після виконання `write(f, іменувальний_вираз)` змінюється на (<f_0, ..., f_{n-1}, *V*>, *n*+1, *RW*), де *V* — значення змінної.

У виклику `write` допускається довільне число аргументів відповідного типу: `write(f, вираз`$_1$`, ..., вираз`$_n$`)`. Такий виклик виконується насправді як послідовність `write(f, вираз`$_1$`); ...; write(f, вираз`$_n$`)`.

Наприклад, замість `write(f, y); write(f, x);` можна написати `write(f, y, x)`.

- До типізованих файлів не можна застосовувати процедуру `writeln`. Її означено лише для текстів.

Приклад 10.2. Розглянемо створення файла з даними про успішність студентів: прізвище, ім'я та середній бал.

Дані про студента подамо записом такого типу.

```
Student = record
            surName, name : string[30];
            mark          : real
          end;
```

Зв'язування та установлення файла у початковий стан для запису оформимо як процедуру `rewritefStudent`, а заповнення — `makefStudent`. При створенні файла дані про студентів циклічно набираються на клавіатурі та присвоюються допоміжній структурі типу `Student` шляхом виконання процедури `getStud`, а потім виводяться у файл. Всі процедури уточнюються в такій програмі (лістинг 10.1).

Лістинг 10.1. Створення файла з даними про успішність

```
program InPutGroup;
  type Student = record {ім'я, прізвище, середній бал}
         surname, name : string[30];
         mark          : real;
       end;
       fStudent = file of Student; {тип файла}
  var fi : FStudent; {файлова змінна}

  {зв'язування та установка в початок}
  procedure rewritefStudent(var f : fStudent);
    var fileName : string; {зовнішнє ім'я файла}
  begin
    writeln('Задайте зовнішнє ім''я створюваного файла:');
    readln(fileName);
    assign(f, fileName);
    rewrite(f);
  end;

  {зібрати дані про одного студента в структуру St}
  procedure getStud(var st : Student);
  begin
```

```
    writeln('Задайте дані про студента:');
    write('Прізвище>'); readln(st.surname);
    write('Ім''я>'); readln(st.name);
    write('Середній бал>'); readln(st.mark);
  end;

  procedure makefStudent(var f : fStudent);
    var st : Student;
    {запис для формування даних про студента перед виводом у файл}
        ch : char;
  begin
    repeat
      getStud(st);      {одержати дані про студента}
      write(f, St);     {записати структуру у файл}
      write('Чи будуть ще дані? "Y"/"N">');
      readln(ch);
    until (ch = 'n') or (ch = 'N');
    close(f);
  end;

begin
  writeln('Створення файла даних про успішність');
  reWritefStudent(Fi);
  makefStudent(Fi);
end.
```

□

- Параметри підпрограм, оголошені як файли, можуть бути тільки *параметрами-змінними*.

10.1.3. Послідовне читання типізованих файлів

Для читання створеного раніше типізованого файла треба *відкрити* його за допомогою процедури `reset`. Після виклику `reset(f)` файл встановлюється у *початковий стан для читання-запису*, а номером доступного елемента стає 0 незалежно від того, чи є елементи у файлі. Значення `f` можна подати як (*F*, 0, *RW*). Якщо фізичного файла на диску не існувало, можливе аварійне завершення програми.

- Перед читанням файла на диску його установка в початковий стан для читання *обов'язкова*. Встановлювати файлову змінну, зв'язану з клавіатурою, можна, але не обов'язково.

✧ У програмі допускається будь-яке число викликів `reset`, і після них можна не тільки читати з файла, але й записувати в нього, тобто змінювати його перший елемент. Взагалі, система Турбо Паскаль дозволяє при будь-якому стані файла змінювати доступний елемент за допомогою `write`.

Доступний елемент файла читається за допомогою процедури `read`. Її виклик має вигляд `read(f, v)`, де `v` — ім'я змінної типу, *сумісного за присвоюванням* із типом елементів файла. Значення доступного елемента присвоюється цій змінній, а доступним стає наступний елемент. Наприклад, при читанні `read(f, x)` файла `f` із значенням (<11, 12, 13>, 0, *RW*) змінна `x` одержує значення 11, а значення `f` можна подати як (<11, 12, 13>, 1, *RW*), тобто доступним стає елемент зі значенням 12.

Взагалі, якщо файл містить n елементів із номерами від 0 до n–1 і номер доступного компонента i менше n, то виклик `read(f, x)` задає присвоювання змінній `x` значення i-го елемента і перехід від значення файла (F, i, RW) до (F, i+1, RW). Якщо ж $i = n$, то кажуть, що файл `f` *прочитано*, або *вичерпано* (зокрема, якщо він порожній, тобто $F = <>$), і виклик `read` призводить до аварійного завершення програми.

У викликах процедури `read` можна вказувати довільну кількість аргументів. Виклик `Read(f, v1, v2, ..., v)` виконується насправді як послідовність викликів

`read(f, v1); read(f, v2); ...; read(f, v).`

Зрозуміло, при цьому необхідно гарантувати, що у файлі ще достатньо непрочитаних елементів.

- Для типізованих файлів не означено процедуру `readln`, яка застосовується тільки до текстів.

Визначення, чи прочитано файл, виконується за допомогою функції `eof`. З виклику `eof(f)` повертається булеве значення `false`, якщо є доступний елемент файла, тобто $i < n$ у значенні файлової змінної ($<f_0, ..., f_{n-1}>$, i, RW). Значення `true` повертається, якщо $i = n$, тобто файл прочитано.

- Виклик функції `eof` дозволяє визначити, чи є у файлі доступний елемент, і запобігти читанню з вичерпаного файла. Будь-якій спробі читання з файла має передувати успішна перевірка того, що файл ще не вичерпано (“***не перевіривши, не читай***”).

Приклад 10.3. Обчислити середнє арифметичне A цілих чисел файла й записати в інший файл усі числа, які менші A.

Алгоритм розв’язання задачі простий.

Прочитати всі числа з файла та обчислити їх середнє A.
Знову прочитати числа та записати в інший файл ті, які менші A.

Нехай числа записано у файлі цілих з іменем `nums.dat`, а числа менше середнього копіюються у файл `littles.dat`. Описані дії задамо такою програмою.

```
program numbers;
  type fInteger = file of integer;
  var fileIn, fileOut : fInteger;
      v, n : integer; a : real;
begin
  assign(fileIn, 'nums.dat');
{Прочитати всі числа з файла та обчислити їх середнє A}
  reset(fileIn);
  a := 0; n := 0;
  while not eof(fileIn) do begin
  {із виклику eof(fileIn) повернулося false,}
  {тому можна читати доступний елемент}
    read(filein, v);
    a := a+v;
    n := n+1
  end;
  {із виклику eof(fileIn) повернувся true; файл прочитано}
  if n > 0 then a := a/n;
{Знову прочитати числа та записати в інший файл числа менше A}
  reset(fileIn);
  assign(fileOut, 'littles.dat'); rewrite(fileOut);
  while not eof(fileIn) do begin
```

```
    read(fileIn, v);
    if v < a then write(fileOut, v)
  end;
  close (fileIn); close(fileOut);
end.
```

□

Приклад 10.4. Вивести на екран комп'ютера дані про студентів із файла, створеного за програмою з прикладу 10.2.

Природно розглядати файл як послідовність записів типу `Student`. Його елементи по одному читаються в допоміжну змінну `St`, і значення її полів (прізвище, ім'я та середній бал), які мають базовий або рядковий тип, виводяться на екран. Після кожного виведення для одержання нових даних треба натиснути <Enter>.

Дані з файла виводяться на екран за допомогою процедури `printfStudent` (лістинг 10.2). Дані з файла циклічно читаються в структуру типу `Student` за допомогою процедури `read`, а окремі поля структури виводяться на екран при виклику процедури `putStud`.

Лістинг 10.2. Виведення даних із файла на екран

```
program PutStudents;
  type Student = record {прізвище, ім'я, середній бал}
         surname, name : string[30];
         mark          : real;
       end;
       fStudent = file of Student; {тип файла}
  var fi : fStudent; {файлова змінна}
      fileName : string;

  {видати дані про одного студента на екран}
  procedure putStud(var st : Student);
  begin
    writeln('Прізвище: ', st.surname);
    writeln('Ім''я:     ', st.name);
    writeln('Середній бал: ', st.mark);
  end;

  {видати дані з файла на екран}
  procedure printfStudent(var f : fStudent);
    var st : Student; ch : char;
  begin
    writeln('Дивитися дані про студентів ? "Y"/"N" ');
    readln(ch);
    if (ch = 'Y') or (ch = 'y')
      then begin
        reset(f);
        while not eof(f) do begin
          read(f, st);
          putStud(st);
          writeln('Для продовження натисніть <Enter>');
          readln;
        end;
    end;
  end;

begin
  writeln('Друк даних про студентів');
  write('Задайте ім''я файла:'); readln(fileName);
```

```
  assign(fi, fileName); reset(fi);
  printfStudent(fi);
  close(fi)
end.
```

□

10.1.4. Знайомство з контролем правильності введення-виведення

Повернімося до програми з прикладу 10.4. Під час її виконання користувач задає ім'я файла, з якого мають читатися дані про студентів. Якщо, наприклад, задати ім'я неіснуючого файла, то при спробі його відкрити за допомогою процедури `reset` виникає помилка, і програма *аварійно завершується*. Проте система Турбо Паскаль дозволяє програмісту контролювати правильність виконання операцій введення-виведення та уникати аварійних завершень в разі помилки.

Щоб при помилковому виконанні підпрограм введення-виведення програма не завершувалася аварійно, перед їх викликом директивою компілятора `{$I-}` вказують *вимкнення режиму* контролю правильності введення-виведення. У режимі контролю компілятор додає до коду програми команди, які фіксують помилку введення-виведення в спеціальній змінній (встановлюється *прапорець помилки*) і ведуть до аварійного завершення програми. Якщо ж режим контролю вимкнено, додаються лише команди, які фіксують помилку.

Наявність помилки можна перевірити за допомогою функції `IOResult` із модуля `Dos`. Якщо прапорець помилки встановлено, ця функція повертає ненульове значення (код помилки) і скидає прапорець, інакше повертає 0. Після "критичних" викликів підпрограм введення-виведення режим контролю введення-виведення можна увімкнути директивою компілятора `{$I+}`.

Зв'язування та відкриття файла з даними про студентів (див. приклад 10.4) оформимо як функцію `resetfStudent`, яка повертає ознаку того, що зазначені процеси пройшли успішно. Якщо користувач помилився, він одержить повідомлення про це та запрошення до введення нового імені. Якщо користувач відмовиться його задавати, з функції `resetfStudent` повернеться значення `false`.

```
{функція зв'язування та установлення в початок}
function resetfStudent(var f : fStudent) : boolean;
  var fileName: string;  {зовнішнє ім'я файла}
      rez     : Integer; {ознака помилки при відкритті файла}
      ch      : char;    {символ для діалогу}
begin
  rez := 1;
  while rez <> 0 do begin
  {rez <> 0 - правильне зовнішнє ім'я файла не задано}
    write('Задайте ім''я файла:'); readln(fileName);
    assign(f, fileName);
    {$I-}  {вимкнення контролю правильності введення/виведення}
    reset(f);
    {$I+}    {увімкнення контролю}
    rez := IOResult;
    if rez <> 0 then begin
      writeln('Перевірте правильність імені файла !');
      writeln('Чи буде повторно задано ім'я? "Y"/"N":');
      readln(ch);
      if (ch = 'n') or (ch = 'N')
        then begin resetfStudent := false; exit end
    end;
```

```
  end;
  resetfStudent := true
end;
```

Додамо наведену функцію до програми `PutStudents` (див. лістинг 10.2), укажемо в програмі підключення модуля `Dos` та змінимо її тіло на таке.

```
begin
  writeln('Друк даних про студентів');
  if resetfStudent(Fi) then begin
    printfStudent(Fi);
    close(Fi)
  end;
end.
```

10.1.5. Прямий доступ у системі Турбо Паскаль

У попередніх підрозділах було подано *послідовний доступ* до елементів файла. Елементи файла не задаються явно, а їх доступність під час виконання програми цілком визначається послідовністю у файлі: спочатку доступний перший елемент, після його обробки — другий тощо.

Послідовний доступ зручний далеко не завжди. Наприклад, у файлі студентів треба знайти елемент за заданим прізвищем і змінити в цьому елементі середній бал. Щоб порівняти задане прізвище зі значенням поля елемента у файлі, треба елемент прочитати. Але після читання він уже недоступний, і для повернення до нього треба знову прочитати файл до цього елемента.

Розглянемо *прямий доступ* до елементів файла. Його засновано на використанні номерів елементів у послідовності, яка утворює файл, і здійснюється за допомогою спеціальних підпрограм.

Головною є процедура `seek`. У її виклику задають ім'я файлової змінної та номер елемента файла, який стає доступним після виконання виклику. Номер задають виразом типу `LongInt`. Наприклад, після виклику `seek(f, 2)` доступним буде третій елемент (нумерація починається з 0). Значення саме цього елемента буде читатися за допомогою виклику `read` або цьому елементу буде щось присвоюватися при виконанні `write`. Після введення або виведення елемента доступним стане наступний за ним.

- ✧ Виклик процедури `seek` записується після відкриття файла за допомогою `reset`, і після нього можна як читати, так і записувати елементи файла, тобто режим доступу не має значення.

У системі Турбо Паскаль є декілька процедур, які застосовуються разом із процедурою `seek`.

Функція `filepos` повертає номер доступного елемента типу `longInt`. Аргумент у її виклику — ідентифікатор файлової змінної. Значення, яке повертається, можна використовувати, наприклад, для переходу до попереднього елемента.

```
seek(f, filePos(f)-1)
```

Число елементів у файлі (типу `longInt`) повертається функцією `filesize`. Аргумент у її виклику — ідентифікатор файлової змінної. Наприклад, за її допомоги можна дописати значення нового елемента в кінець файла.

```
seek(f, filesize(f));
write(f, v)
```

- Процедура `seek` у парі з `read` або `write` дозволяє прочитати будь-який елемент файла або змінити його значення.

Процедура `truncate` знищує залишок файла, починаючи від доступного елемента. Наприклад, після викликів `seek(f, 3); truncate(f)` у файлі залишаються тільки елементи з номерами 0, 1 і 2. За допомогою процедури `truncate` легко також вилучити елемент із

файла. Потрібний елемент знаходиться, обмінюється значеннями з останнім елементом файла, який потім знищується за допомогою `truncate`. Зауважимо, що при цьому змінюється порядок значень елементів у файлі.

Значення другого аргументу у виклику `seek` може не бути номером елемента у файлі, тобто може бути від'ємним або більше, ніж число елементів файла. При виклику нічого не відбудеться, але подальша спроба прочитати елемент із установленим від'ємним номером може призвести до аварійного завершення програми, оскільки елемента з таким номером немає!

Ще гірше, якщо у виклику `seek` задати номер, який більше числа елементів у файлі. Подальший запис у цей нібито доступний елемент файла призведе до того, що у файлі "чомусь" з'являться зовсім незрозумілі дані — "сміття". Перевірте це й поміркуйте, чому так станеться. І будьте уважні, записуючи другий аргумент у виклику `seek`.

10.2. Найпростіша робота з текстами

10.2.1. Особливості організації текстів

Для представлення текстів на зовнішніх носіях означено спеціальний тип з іменем `text`. Файли цього типу називаються *файлами-текстами* або просто *текстами*. Для текстів означено специфічні підпрограми, частину яких пов'язано з поділом текстів на рядки.

Поділ тексту на рядки здійснюється за рахунок особливої інтерпретації спеціальної послідовності символів, яку найчастіше вважають єдиним символом. Ми будемо позначати цю послідовність <***eol***> ("end of line"). У системі Турбо Паскаль такою послідовністю є `#13#10` або `#13`. Рядок файла, що містить лише цей символ, називається *порожнім*. Наведемо очевидний приклад тексту.

```
ТЕКСТ<eol>СКЛАДАЄТЬСЯ<eol><eol>З П'ЯТИ РЯДКІВ;<eol>ТРЕТІЙ
ПОРОЖНІЙ<eol>
```

Звичайно, приємніше читати цю послідовність у вигляді

```
ТЕКСТ<eol>
СКЛАДАЄТЬСЯ<eol>
<eol>
З П'ЯТИ РЯДКІВ;<eol>
ТРЕТІЙ ПОРОЖНІЙ<eol>
```

Визначити, чи доступний <***eol***> у файлі в певний момент, можна за допомогою функції `eoln`. З виклику `eoln(f)` повертається `true`, якщо доступний <***eol***> або кінець файла, і `false` — в іншому випадку.

Щоб визначити, чи прочитано текст, викликають функцію `eof`. Певні спеціальні символи всередині тексту можуть тлумачитися як ознака його кінця. Для програм, написаних мовою Турбо Паскаль, таким символом є `#26`. Якщо в тексті `f` доступним є символ `#26`, то з виклику `eof(f)` повертається `true`, а якщо інший символ — `false`.

- ✧ Ім'я `input` можна не вказувати у викликах і, наприклад, замість `eof(input)` або `eoln(input)` писати `eof` або `eoln`.

10.2.2. Запис у текст

Нехай `f` — ім'я файлової змінної типу `text`. Текст установлюється в початковий стан для запису (та спустошується) при виклику `rewrite(f)`. Щоб дописати дані до вже існуючого тексту, треба встановити його в *початковий стан для дописування*, викликавши процедуру `append`: `append(f)`. Усі символи, крім кінця файла, залишаться в тексті, і нові символи будуть дописуватися до нього.

Символи записуються в текст при викликах процедур `write` і `writeln`. Їх перший аргумент — ім'я файлової змінної. При виконанні виклику `write(f, вираз_ска-`

лярного_типу_або_рядок) обчислюється значення виразу, потім за значенням створюється константа, тобто послідовність символів, які представляють значення, і копіюється в текст `f`. Наприклад, при виконанні виклику

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

у файл допишуться два символи `'9'` і `'9'`, що представляють число `99`.

У виклику можна записати кілька виразів через кому.

```
write(f, вираз1, вираз2, ...)
```

Наприклад, при `x` = 2 і виклику

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

у файл буде виведено послідовність символів `x=2; x**2=4`. Такий виклик насправді виконується як послідовність викликів

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

Процедура `writeln` відрізняється лише тим, що при виклику

```
writeln(f, список_виразів_скалярних_типів_або_рядків)
```

за останньою константою в текст додається ***<eol>*** (символи `#13#10`). Список виразів може бути порожнім — тоді тільки додається ***<eol>***.

У викликах процедур `write` і `writeln` після виразу через двокрапку можна вказати ширину поля *W* для запису значення виразу, наприклад `write(f, sqr(x):2)`. Тут *W* = 2. Припустімо, що для запису значення виразу насправді потрібно *L* символів. В останньому виклику *L* = 1 при `x` < 4, *L* = 2 при 3 < `x` < 10, *L* = 3 при 9 < `x` < 34 тощо. Якщо $L \le W$, то перед символами значення додається *W*–*L* пропусків; якщо ж $L > W$, виводяться всі символи. Отже, при `x` = 3 друкуються пропуск та `9`, а при `x` = 100 — усі п'ять символів `10000`.

Після виразу дійсного типу можна також указати кількість *N* цифр дробової частини, які виводяться після десяткової крапки, наприклад `write(f, sqrt(x):7:3)`. Якщо *N* указано, то виводиться константа з фіксованою крапкою та *N* цифрами після крапки, інакше — нормалізована з порядком. У цьому випадку при `x` = 2 виводяться два пропуски та `1.414`. Остання цифра є результатом округлення. Деякі інші деталі можуть визначатися системою програмування.

Приклад 10.5. Створити текст можна буквально "власноруч", набравши його рядки на клавіатурі у відповідь на запрошення при виконанні такої програми (лістинг 10.3).

Лістинг 10.3. Створення тексту вручну

```
program CreateText;
  var f : text;   {файл-текст}
      s : string; {рядок для введення з клавіатури}
begin
  writeln('Створення тексту. Кінець роботи - <Ctrl+Z> замість рядка');
  assign(f, 'myfile.txt'); rewrite(f);
  writeln('Наберіть рядок і натисніть <Enter>:');
  while not eof(input) do begin
    readln(s);
    writeln(f, s );
    writeln('Наберіть рядок і натисніть <Enter>:');
  end;
  close(f)
end.
```

Для закінчення роботи треба замість введення нового рядка натиснути <Ctrl+Z> і <Enter>.

Ви можете подумати, що довжина рядків тексту обмежена числом 255. Проте спроби набрати рядок довше 127 символів будуть невдалими. Причини буде пояснено в розділі 10.3. □

10.2.3. Читання числових констант

Читання тексту на диску комп'ютера має набагато більше "підводних каменів", ніж запис. Розглянемо читання числових констант; символи та рядки обробляються інакше.

- Читання текстів суттєво відрізняється від читання типізованих файлів.

Для введення числових констант зручно користуватися процедурою `read` — її виклик має такий вигляд (`f` — ім'я тексту).

`read(f, `*`список_іменувальних_виразів_числових_типів`*`)`

Для безпомилкового виконання виклику файл повинен містити послідовність констант, типи яких відповідають типам змінних, указаних у списку.

Ціла константа — це послідовність цифр, можливо, зі знаком `'+'` або `'-'` на початку, яка задає ціле число відповідного цілого типу, причому між знаком та першою цифрою в тексті не може бути пропусків. Дійсна константа — це послідовність цифр та інших символів зі структурою констант мови Паскаль, наприклад `1.1`, `2.`, `0.99`, `1e-3`, `-2.73E+02`.[21]

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

Виклик `read(f, `*`X`*`)`, де *X* — ім'я цілої або дійсної змінної, виконується так. З тексту читаються пропуски, а потім — символи константи до найближчого пропуску (можливо, до символу `#26` або кінця файла). Доступним після читання константи буде перший пропуск після неї (відповідно, `#26` або кінець файла). Якщо символи утворюють константу потрібного типу, то за ними обчислюється значення й присвоюється змінній. При дійсній змінній *X* у тексті може знаходитися й ціла константа — за нею обчислюється дійсне значення.

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

- ✧ Якщо доступний кінець файла чи від поточної позиції в файлі до його кінця або найближчого символу `#26` записано лише пропуски, то спроба читати число не завершується аварійно, а відповідна змінна одержує значення 0.

Читання констант базових типів за процедурою `readln` аналогічно процедурі `read`. Відмінність полягає в тому, що після читання останньої константи всі символи тексту, які залишилися до найближчого <***eol***>[22], пропускаються разом із ним. Доступним стає перший символ наступного рядка тексту. Список імен змінних може бути порожнім — тоді `readln(f)` пропускає поточний рядок тексту. Наприклад, нехай у тексті `f` записано такі символи.

1		2		3	<**eol**>
	5	5	#26		

Нехай `x`, `y`, `z`, `t` — імена цілих змінних. Після викликів

`read(f, x, y); read(f, z, t)`

вони отримають значення 1, 2, 3, 55, відповідно, після

`readln(f, x, y); read(f, z, t)` –

1, 2, 55, 0, а після

---

[21] *Насправді можуть бути прочитаними деякі константи, які «не пропускаються» компілятором.*

[22] *Якщо за процедурою `readln` читають несимвольні змінні, а в тексті символ #26 передує найближчому <eol>, то читання завершується перед символом #26, і він же стає поточним. Процедура read при читанні в несимвольні змінні також зупиняється перед символом #26.*

`readln(f, x); readln(f, y, z, t)` —

1, 55, 0, 0.

Читання числової послідовності, записаної в тексті, можна описати за допомогою функції `eof` у такому циклі.

```
while not eof(f) do begin
  read(f, v); обробка значення змінної v
end.
```

З виклику `eof(f)` повертається значення `true`, якщо доступний кінець файла або `#26`. При такій організації читання необхідно забезпечити, щоб між останньою числовою константою та кінцем файла не було порожніх символів. Якщо цього не зробити, буде прочитано зайве нульове значення.[23]

Читання в циклі відбувається після того, як із виклику `eof` повернулося значення `false`, тобто кожному читанню передує перевірка, чи не прочитано файл. Якщо файл порожній, то перше ж виконання виклику `eof(f)` породжує `true`, і тіло циклу не виконується жодного разу. Змінна `v` залишається зі своїм старим значенням, що не завжди є очікуваним і бажаним.

10.2.4. Особливості читання символів і рядків

При виконанні виклику `read(f, X)` змінній `X` типу `char` присвоюється доступний символ тексту, яким би він не був, а доступним стає наступний за ним. Виняток — якщо доступний `#26`, то `X` отримує значення `#0`, і символ `#26` залишається доступним.

Якщо `X` рядкового типу, то при виконанні `read(f, X)` символи до найближчого <***eol***> (або до кінця файла чи його ознаки `#26`) читаються та присвоюються елементам рядка `X`; доступним буде <***eol***> (або `#26`). Сам <***eol***>, тобто відповідні символи, у рядок-змінну не записується. Якщо символів тексту до <***eol***> більше, ніж уміщається в `X`, то `X` заповнюється до кінця, і доступним стає перший символ після прочитаних у `X`.

- Якщо перед читанням рядка `X` за допомогою процедури `read` доступний <***eol***>, то він залишається доступним, а значенням `X` стає порожній рядок. Тому застосовувати процедуру `read` для читання рядків *не рекомендується*.

Особливо небезпечно застосовувати `read` при введенні рядків у циклі. Якщо після введення рядка доступним символом стане <***eol***>, то подальші спроби читання рядків даватимуть їм значення `''` (порожній рядок), <***eol***> залишатиметься доступним, і виконання програми "зациклиться". Розглянемо, наприклад, такий цикл (`x` — змінна-рядок).

```
while not eof(f) do begin
  read(f, x);              {???}
  обробка x
end,
```

При його виконанні читаються символи тільки першого рядка тексту, а потім кінець рядка залишається доступним "назавжди".

Якщо при читанні рядка доступний кінець тексту, він також залишається доступним, а змінній-рядку присвоюється порожній рядок.

Читання символів і рядків за процедурою `readln` аналогічно процедурі `read`, але після читання символів решта рядка тексту пропускається, тобто доступним буде символ, що йде за найближчим <***eol***>. Якщо список імен змінних у виклику `readln` порожній, пропускається поточний рядок тексту (але не далі `#26`).

---

[23] *Взагалі, якщо всередині тексту з числовими константами присутній символ #26, то наведений фрагмент обробятиме лише частину тексту до символу #26.*

Якщо рядки тексту "уміщаються" в змінних типу `string`, то читання тексту дуже просто описати за допомогою змінних-рядків та процедури `readln`.[24] Алгоритми розв'язання багатьох задач із обробки текстів мають такий загальний вигляд.

```
while not eof(f) do begin
  readln(f, s); {s має тип string}
  обробка рядка s;
end.
```

Приклад 10.6. Написати процедуру копіювання тексту за умови, що його рядки мають довжину не більше 255, а порожні рядки або ті, що містять лише пропуски, не копіюються.

Припустимо, що вхідний і цільовий файли-тексти задають при виклику процедури. За умовою рядки тексту можна читати в змінну типу `string`. Обробляючи рядок, треба визначити, чи є в ньому хоча б один символ, відмінний від пропуску. Якщо є, рядок копіюється в цільовий текст.

```
{процедура копіювання тексту f у текст g без порожніх рядків}
procedure copyNonEmpty(var f, g : text);
  var s : string;   {поточний рядок тексту}
      k : integer;  {лічильник позицій}
begin
  reset(f); rewrite(g);
  while not eof(f) do begin
    readln(f, s);
    {обробка рядка s}
    k := 1;
    while k <= length(s) do
      if s[k] <> ' '
        then begin writeln(g, s); break end
      else k := k+1;
    {рядок оброблено}
  end;
  close(f);close(g);
end;
```

Для перевірки цієї процедури достатньо тексту, в якому лише три рядки — порожній, такий, що складається з кількох пропусків, а також рядок із буквами або цифрами. У цільовий текст має скопіюватися тільки останній рядок.

□

10.3. Буферизація введення та виведення

10.3.1. Ідея буферизації

Зовнішні носії даних улаштовано так, що обмін даними між ними та оперативною пам'яттю відбувається великими порціями (десятки й сотні Кбайт). Проте в програмах дані, як правило, вводяться або виводяться окремими скалярними значеннями, що займають одиниці байтів. Виникає суперечність між можливостями фізичних пристроїв і потребами програм. Для її розв'язання використовуються буфери.

[24] *Обробка довших рядків здійснюється за допомогою символьних масивів або рядків невизначеної довжини (див. розділ 11.5), а також алгоритмів лексичного аналізу (див. главу 15).*

Буфер — це спеціальна ділянка пам'яті програми, яка надається кожній файловій змінній при її зв'язуванні. Використання буферів називається *буферизацією*, а введення-виведення з буферизацією — *буферизованим*.

Спочатку розглянемо фізичні файли на диску. Дискова пам'ять розбивається на блоки — ділянки фіксованого розміру 4096 або більше байт).[25] Дані копіюються на диск або з диска блоками, тобто блок є одиницею фізичного обміну між диском і оперативною пам'яттю. Якщо виконується виклик підпрограми читання, у буфер копіюється один або декілька блоків даних із файла, і значення в змінні програми потрапляють не з файла, а з буфера. При наступних спробах читання дані між диском та оперативною пам'яттю не пересуваються, а беруться з буфера.

Розмір буфера визначається операційною системою. Як правило, це 2, 8, 16 і більше блоків.

З кожним буфером пов'язано додаткову змінну, яка вказує на його поточний елемент. Значення цього елемента копіюється при читанні, а поточним стає наступний за ним, тобто поточний елемент у буфері виступає *представником доступного елемента файла*. Якщо буфер прочитано, то за нової спроби введення наступні блоки файла будуть копіюватися в буфер.

- Буфер і вказівник поточного елемента в Паскаль-програмі явно не оголошуються й не використовуються.

Буфер відіграє роль вікна, через який із програми "видно" файл. Якщо фізичний файл за розміром менше буфера, тобто весь "уміщається у вікні", то для його читання, навіть неодноразового, достатньо одного звернення до файла.

При записі у файл дані записуються в буфер до його заповнення, і фізичне копіювання на диск, або *скидання буфера*, виконується лише при спробі додати дані в заповнений буфер. Після скидання буфер вважається порожнім. Незаповнений буфер очищується при виконанні процедури `close`.

10.3.2. Буферизація текстів

Буферизація текстів має додаткові особливості. З текстом пов'язаний не один, а два буфери. Перший, *зовнішній*, обробляється, як написано в попередньому підрозділі. Робота з другим, *внутрішнім*, відбувається інакше. При введенні дані копіюються з тексту в зовнішній буфер, а звідти їх частина — у внутрішній. Яка саме частина, залежить від розміру внутрішнього буфера. При введенні символи тексту насправді читаються з внутрішнього буфера, а якщо цей буфер вичерпано, то в нього копіюється наступна частина зовнішнього буфера (можливо, зі зверненням до фізичного файла), і читання продовжується.

Внутрішній буфер текстів у системі Турбо Паскаль має розмір 128 байт. Його можна змінити в межах від 1 до 65 536 байт за допомогою процедури `setTextBuf`. Її виклик має вигляд

```
setTextBuf(f, Buf, Bufsize)
```

або

```
setTextBuf(f, Buf)
```

і записується після зв'язування файла, але перед установкою в початковий стан. Аргумент `f` — це ім'я файла типу `text`, `Buf` — ім'я буфера, тобто змінної, тип якої не має значення, а `Bufsize` — вираз цілого типу зі значенням у межах `1..65535`.

✧ Установка внутрішнього буфера, розмір якого кратний довжині блоку (як правило, це 4 Кбайт), може значно прискорити виконання програми.

[25] *На дискетах стандартна довжина блоку дорівнює 512 байт.*

Змінну Buf звичайно оголошують як масив символів або байтів. Якщо розмір Bufsize (у байтах) у виклику не вказано, він визначається довжиною змінної Buf. Якщо значення Bufsize указано і менше довжини змінної Buf, воно задає довжину буфера в межах змінної Buf. Але якщо значення Bufsize більше довжини Buf, то змінні, розташовані в пам'яті відразу після Buf, стають частиною буфера, що може призвести до непередбачених наслідків.

Приклад 10.7. У програмі GreatBufferManager встановлюється "іграшковий" буфер довжиною 4 байт, що починається в масиві розміром 2 байт.

```
program GreatBufferManager;
  var f : text;
    Hugebuf : array [1..2] of char; {"величезний буфер"}
    x, y : char;
    s : string[4];
begin
  assign(f, 'huge.dat');
  settextbuf(f, Hugebuf, 4); {буфер у 4 байт починається з hugebuf}
  x := 'x'; y := 'y';
  reset(f); readln(f, s);
  writeln('x = ', x, '; y = ', y);
  close(f)
end.
```

Якщо першим рядком тексту у файлі huge.dat є '1234', то при виконанні цієї програми на екрані замість очікуваного

```
x = x; y = y
```

з'явиться

```
x = 3; y = 4.
```

Справа в тім, що під час компіляції змінні розташовуються в пам'яті згідно з порядком їх оголошення у програмі. При читанні чотирьох символів рядка файла в буфер заповнюються масив Hugebuf і ще два байти за ним, тобто змінні x і y. Якщо зробити рядок у файлі ще довше, то результати будуть ще більш несподіваними. Перевірте це, але обережно — можна пошкодити програму, і не одну.

□

При виведенні в текст символи накопичуються у внутрішньому буфері. Його буде скинуто в зовнішній буфер при заповненні або виклику процедури writeln або close. Примусове очищення внутрішнього буфера тексту задає процедура flush. Її можна використовувати для періодичного скидання буфера, а також по закінченні виведення. При виклику процедури flush внутрішній буфер буде скинуто в зовнішній, але очищення зовнішнього буфера при цьому відбувається тільки у випадку його заповнення.

- Якщо наприкінці роботи з файлом не викликати ні flush, ні close, то вміст внутрішнього буфера не потрапить до зовнішнього буфера і до файла.

Змінну під внутрішній буфер краще оголошувати на рівні програми або модуля, оскільки вона, як правило, використовується в декількох підпрограмах. Але якщо всю роботу з файлом, від assign до close, описано в межах однієї підпрограми, то й буфер краще оголосити в ній же.

10.3.3. Буферизація екрана та клавіатури

Клавіатура та екран за узгодженням розглядаються як текстові файли з іменами input і output. Для роботи з ними також використовуються буфери.

При виведенні на екран символи потрапляють у буфер екрана (відеопам'ять) і відразу копіюються на екран. Якби вони затримувалися в буфері, дані на екрані з'являлися б із небажаними затримками.

Робота з клавіатурою складніше. Символи, утворені при натисканні клавіш, накопичуються в *буфері клавіатури*. Він уміщує 15 символів. У цьому легко переконатися, запустивши таку програму (без заголовка).

```
uses crt; begin delay(20000) end.
```

При її виконанні протягом 20 секунд можна натиснути якусь клавішу 16 разів і при останньому натисканні почути звуковий сигнал комп'ютера, що свідчить про переповнення буфера клавіатури. Буфер переповнюється, оскільки при виконанні цієї програми символи з нього не переносяться до внутрішнього буфера. Набрані далі символи не з'являються на екрані й узагалі "зникають без вісті".

Перенесення символів у внутрішній буфер відбувається при виконанні процедур читання `readln` і `read`. Як і для інших текстів, його розмір становить 128 байт. У цьому легко переконатися, запустивши програму (без заголовка)

```
begin readln end.
```

Її виконання починається чеканням натискань клавіш. При кожному натисканні (крім <Enter>) відповідний символ з'явиться в буфері клавіатури. Символ відразу буде перенесено до її внутрішнього буфера й через екранний буфер без затримки відображено на екрані. Після набору 128 символів новий символ до внутрішнього буфера не перенесеться й на екрані не з'явиться. Замість цього звуковий сигнал засвідчить про переповнення внутрішнього буфера.

Натискання <Enter> призводить до появи відповідного символу в буфері клавіатури й переведення курсору в новий рядок екрана. Якщо цей символ з'являється у внутрішньому буфері, рядок у ньому розглядається як *завершений символом* <***eol***>.

Завершений рядок у внутрішньому буфері аналізується; за константами у ньому обчислюються значення базових типів і присвоюються змінним, указаним у виклику `read` або `readln`. Якщо констант менше, ніж змінних у виклику, тобто внутрішній буфер вичерпано, то починається чекання нових символів від клавіатури. Якщо в рядку проаналізовано останню константу, поточним стає наступний за нею символ у внутрішньому буфері. Виконання виклику `read` на цьому закінчується. При наступному виклику пошук і аналіз констант почнеться з поточного символу буфера, що залишився від попереднього виклику.

Процедура `readln` відрізняється тим, що після аналізу останньої константи інші символи у внутрішньому буфері пропускаються разом із найближчим <***eol***>, тобто буфер скидається.

При виклику функції `eof` аналізується внутрішній буфер клавіатури. Якщо він порожній, виконання програми зупиниться до найближчого натискання клавіш. Тоді символи з буфера клавіатури буде перенесено у внутрішній до натискання <Enter>. При наявності символів у внутрішньому буфері аналізується поточний символ. Якщо він відповідає комбінації клавіш <Ctrl+Z>, якою на клавіатурі задають кінець файла, то з виклику `eof` повернеться значення `true`. При іншому першому символі, тобто при натисканні клавіші, відмінної від <Ctrl+Z>, повернеться `false`.

Приклад 10.8. Нехай діє оголошення **`var`** `V : integer`, і клавіші не натискаються до того, як почне виконуватися такий фрагмент програми.

```
V := 0;
while not eof do begin
  write('Задайте ціле число>'); read(V)
end;
writeln ('V = ', V)
```

При виконанні `eof` комп'ютер чекає натискань клавіш. Запрошення до введення числа на екрані ще не з'явилося. Якщо натиснути <Ctrl+Z> і <Enter>, то з виклику `eof` повертається `true`, тіло циклу не виконується, і друкується текст "`V = 0`". При натисканнях цифрових клавіш цифри відображаються на екрані й накопичуються у внутрішньому буфері клавіатури. Після натискання <Enter> із виклику `eof` повертається значення `false`. Після цього, тобто після набору на клавіатурі першої константи, виконується тіло циклу й з'являється запрошення `Задайте ціле число>`.

При виконанні `read` аналізуються символи, накопичені у внутрішньому буфері під час роботи `eof`. Якщо вони утворюють константу, відповідне значення присвоюється змінній `V`, а потім повторюється виклик `eof` тощо. Отже, введення символів із клавіатури в такому циклі відбувається при виконанні викликів `eof`, а не `read`!

Щоб запрошення друкувалося до введення першої константи, треба перед циклом додати друк запрошення `write('Задайте ціле число>')`, а в тілі циклу поміняти місцями друк запрошення й виклик процедури читання.

Буферизація ілюструє різницю між процедурами `read` і `readln`. Якщо при виконанні циклу після константи набрати ще пропуск і непорожні символи, то вони залишаться у внутрішньому буфері. Потім із виклику `eof` повернеться `false`. Якщо непорожні символи, які залишилися в буфері, не утворюють константу, то їх аналіз при виконанні `read` призведе до аварійного завершення програми. Якщо ж замість `read` використовувати `readln`, то після обробки константи ці символи будуть пропущені, оскільки їх набрано перед натисканням <Enter>, і програма виконується нормально.

□

10.4. Тип безтипових файлів

Розглянемо програму посимвольного копіювання файлів, які незалежно від походження розглядаються як символьні.

```
program StupidCopy; {“нерозумне копіювання”}
  var f, g : file of char; {вхідний та цільовий файли}
      c    : char; {поточний символ}
      s    : string; {рядок для зовнішнього імені}
begin
  writeln('Задайте ім''я вхідного файла');
  readln(s); assign(f, s);
  writeln('Задайте ім''я цільового файла');
  readln(s); assign(g, s);
  reset(f); rewrite(g);
  while not eof(f) do begin
    read(f, c); write(g, c);
  end;
  close(f); close(g);
end.
```

Здається, що при виконанні цієї невигадливої програмки все гаразд, оскільки за рахунок буферів фізичні файли читаються й записуються порціями по декілька блоків, пристрої працюють якнайкраще, а переміщення інформації відбуваються переважно всередині оперативної пам’яті, тобто швидко. Проте насправді ця програма виконується дуже неефективно.

Розглянемо один із засобів прискорення роботи з файлами — *явне використання власного буфера*. Це виявляється набагато ефективнішим, ніж обробка буферів засобами системи програмування, і реалізується за допомогою безтипових файлів.

Тип *безтипових файлів* задається словом `file`. Для обробки безтипових файлів також використовуються два буфери. При введенні дані копіюються з фізичного файла в зовнішній буфер, а потім у внутрішній; при виведенні порядок обмінів зворотний. Використання внутрішнього буфера явно позначається у Паскаль-програмі.

Процедури відкриття `reset` і `rewrite` для файлів цього типу мають по два параметри. Крім імені файлової змінної, у їх викликах указується розмір *блока* — одиниці обміну між зовнішнім та внутрішнім буферами в байтах. Розмір блока має належати до діапазону 1..65535. Як не дивно, але слід установити його рівним 1: `ReSet(f, 1)` або `ReWrite(f, 1)`. Чому саме так, сказано нижче.

Процедура читання безтипового файла `blockread` має чотири параметри. Усі вони, крім третього, — параметри-змінні. У виклику процедури першим аргументом є ім'я файлової змінної, другий задає місце в пам'яті, з якого починається внутрішній буфер, третій — кількість блоків, які треба прочитати з файла, а в четвертому, типу `Word`, повертається кількість блоків, які насправді прочитано при виконанні виклику. Наприклад, нехай діють такі оголошення.

```
var f   : file; {безтиповий файл}
  inbuf : array [1..50] of char; {буфер для введення}
  blkSize : word;                {розмір блока читання}
  blkQuantity, resultblkQuantity : word; {число блоків - очікуване та
справжнє}
```

Тоді оператори та виклики процедур

```
blkSize := 2; blkQuantity := 25;
reset(f, blkSize);
blockread(f, inbuf, blkQuantity, resultblkQuantity)
```

установлюють розмір блока рівним 2 байтам і задають читання 25 таких блоків із файла. Якщо розмір файла насправді не менше 2×25 = 50 байт і помилок при введенні не було, то значенням змінної `resultblkQuantity` є 25. Після читання масив `inbuf` заповнюється до кінця й далі обробляється залежно від конкретної задачі. При виконанні наступного виклику процедури `blockread` файл `f` буде читатися, починаючи з 51-го байта.

✧ Для безтипових файлів поняття "доступний елемент" замінюється поняттям "доступний байт".

Можливо, у файлі менше 50 байт або при введенні щось трапилося, і насправді прочитано менше ніж указані 25 блоків. Тоді значення змінної `resultblkQuantity` буде меншим 25. Після виклику можна перевірити рівність `resultblkQuantity = blkQuantity`; якщо вона хибна, виконати відповідні дії.

Якщо задати читання менше 25 блоків, масив `inbuf` заповнюється не до кінця, а якщо більше — заповнюється масив `inbuf` та змінні, розташовані після масиву в пам'яті. Оскільки змінні розміщено в порядку їх оголошення, першими "жертвами" в цьому випадку будуть змінні `blkSize`, `blkQuantity`, `resultblkQuantity`. Вони мають тип `word` і займають по 2 байти, тому при виконанні `blockread(f, inbuf, 26, resultblkQantity)` буде змінено першу з них, при `blockread(f, inbuf, 27, resultblkQantity)` — перші дві тощо.

Якщо зовнішній буфер не заповнюється до кінця, то кількість блоків, прочитаних насправді, менше заданої. Щоб уникнути неприємностей, треба забезпечити, щоб розмір файла ділився на розмір блока. Оскільки розмір блока насправді не впливає на швидкість читання, найкраще давати йому значення 1. Тоді проблем не виникне при будь-якому розмірі файла.

Якщо обробляється файл записів фіксованого розміру, то цей розмір можна задавати і для блока. Наприклад, записи типу `Student = record ... end` складаються з *S* байт. Це значення повертається з виклику `sizeOf(Student)`, тому безтиповий файл `f`, зв'язаний із файлом записів типу `Student`, можна відкрити викликом `reset(f, sizeOf(Student))`. Після цього виклик вигляду

```
blockRead(f, Buf, n, nreal)
```

задає читання `n` блоків по *S* байт у пам'ять змінної `Buf`.

Найбільше на швидкість читання безтипових файлів впливає розмір внутрішнього буфера. Що він більше, то меншою буде кількість звернень до зовнішнього носія й швидшою обробка файла. Проте, якщо розмір буфера кратний 512 байтам і більше 8 Кбайт, швидкість читання файла практично не залежатиме від розміру буфера. Щоб переконатися в цьому, напишіть програму читання безтипового файла й запустіть її декілька разів на якомусь великому файлі, щоразу змінюючи розмір буфера.

Процедура блокового виведення `blockWrite` має 4 аналогічних параметри. Відмінність полягає в тому, що дані з внутрішнього буфера через зовнішній записуються у файл з його поточного доступного елемента. Попередньо для файла треба встановити розмір блока за допомогою виклику вигляду `rewrite(f, m)` або `reset(f, m)`.

Повернімося до задачі копіювання й напишемо програму QuickCopy, яка виконується в сотні (!) разів швидше, ніж програма StupidCopy. Внутрішнім буфером є масив символів Buf розміром bufSize = 32 Кбайт. Спочатку за допомогою виклику fileSize визначається розмір вхідного файла в байтах, а потім файл читається в масив порціями по bufSize байт. Обробка буфера в цьому випадку полягає в блоковому копіюванні в цільовий файл. Остання порція може містити менше bufSize байт — масив заповнюється й копіюється у файл не до кінця.

Якщо імена файлів є однаковими, буде викликано процедуру exit, яка завершить виконання програми.

Лістинг 10.4. Буферизоване копіювання безтипового файла

```
program QuickCopy;
  const bufSize = 32768;    {розмір буфера}
  var f, g   : file;        {вхідний та цільовий файли}
    Buf      : array [1..bufSize] of char; {буфер}
    rSize    : longint;     {розмір залишку файла}
    portion  : word;        {розмір чергової порції}
    rCount,                 {число реально прочитаних}
    wCount   : word;        {і записаних байтів}
    s1, s2   : string;      {зовнішні імена файлів}
begin
  writeln('Задайте ім''я файла-джерела:'); readln(s1);
  writeln('Задайте ім''я цільового файла:'); readln(s2);

  if s1 = s2 then exit;    {завершення програми без зміни файлів}
  assign(f, s1); assign(g, s2);
  reset(f, 1); rewrite(g, 1);
  rSize := fileSize(f);    {число байт для копіювання}
  while rSize > 0 do begin
    if rSize > bufSize
      then portion := bufSize
    else portion := rSize;
    blockRead(f, Buf, portion, rCount);
    if rCount <> portion then begin
      writeln('Помилка читання файла ', s1);
      close(g); close(f); exit
    end;
    Blockwrite(g, Buf, portion, wCount);
    if wCount <> portion then begin
      writeln('Помилка запису файла ', s2);
      close(g); close(f); exit
    end;
    dec(rSize, portion); {залишок зменшується на розмір скопійованої
порції}
  end;
  {rSize <= 0 або відбулася помилка}
end.
```

□

Резюме

- *Файл* — це іменована область на зовнішньому носії інформації. Він є послідовністю байтів, які можна розглядати та обробляти як послідовність або значень певного типу (*типізований* файл), або символів із поділом на рядки (*текст*), або байтів (*безтиповий* файл).

- Обробку файлів у мовах програмування задають за допомогою спеціальних засобів, зібраних у систему введення-виведення. У Паскаль-програмі фізичний файл подається файловою змінною. Тип у її оголошенні задає спосіб обробки файла.
- Основна особливість файла полягає в тому, що в будь-який момент виконання програми з усіх його елементів можна обробляти (читати або записувати) тільки один, *доступний*. Інші в цей момент недоступні. Номер доступного елемента в послідовності елементів файла є значенням спеціальної неявної змінної — файлового вказівника. Номер доступного елемента змінюється при виконанні підпрограм обробки файлів.
- *Послідовний доступ* до елементів файла полягає в тому, що елементи файла явно не задають, а їх доступність під час виконання програми цілком визначається послідовністю у файлі: спочатку доступний перший елемент, після його обработки — другий тощо.
- При використанні послідовного доступу будь-якій спробі читання з файла повинна передувати успішна перевірка того, що файл ще не прочитано, тобто є доступний елемент (“***не перевіривши, не читай***”).
- *Прямий доступ* до елементів файла полягає в явному використанні номерів елементів файла й здійснюється за допомогою спеціальних підпрограм.
- *Буфер* — це спеціальна ділянка пам’яті програми, яка надається кожній файловій змінній при її зв’язуванні. Використання буферів називається *буферизацією*. Буферизация дозволяє поєднати можливості зовнішніх пристроїв, які виконують обмін даних великими порціями, із потребами програм, при виконанні яких обмін відбувається невеликими порціями. Буферизація, як правило, зменшує кількість фізичних переміщень даних між зовнішніми носіями та оперативною пам’яттю та прискорює введення-виведення.

Контрольні запитання

10.1. Файлами якого типу є клавіатура та екран?

10.2. Чи можна для обробки типізованих файлів використовувати процедури `readln` і `writeln`?

10.3. Що таке <***eol***>?

10.4. У який спосіб буферизація уточнює поняття доступного елемента файла?

Задачі

10.1.* Описати роботу та використання такої програми.

```
Program fCreate_and_Print;
  type PhoneNumber  = record nam : string; num : integer
       end;
       FphoneNumber = file of PhoneNumber;
  var  f : FPhoneNumber; x : PhoneNumber;
       s : string; last : boolean;

  procedure readElem(var x : PhoneNumber);
  begin
    x.num := 0; readln(x.nam);
    if x.nam <> '' then readln(x.num)
  end;

begin write('Задайте ім''я файла:');
  readln(s); assign(f, s); rewrite(f);
  repeat readElem(x); last := (x.nam = '');
    if not last then write(f, x);
  until last;
```

```
  close(f); reset(f);
  while not eof(f) do begin
    read(f, x); writeln(x.num, ': ', x.nam)
  end;
  close(f)
end.
```

10.2.* Написати функцію побайтової перевірки рівності двох файлів.

10.3.* Написати процедуру дописування до елементів першого файла елементів другого зі зберіганням результату в першому файлі.

10.4.* Написати процедуру виведення змісту файла з даними про студентів (див. Приклад 10.4) на екран "сторінками": після друку на екрані даних про чергових п'ять студентів виводиться запит, чи продовжувати виведення, і виконання програми призупиняється, поки користувач не вкаже, продовжувати чи завершити роботу.

10.5.* Написати "дурнестійкий" варіант процедури `Seek`, за яким при записі у файл не дозволяється задавати номер елемента,що перевищує кількість елементів файла або є від'ємним.

10.6. Створити текст із таблицею степенів числа 2 від 1 до:

а) 30 (використати змінні типу `Longint`);

б) 63 (використати змінні типу `Compound`).

10.7.* Припустімо, що з доступного символу в тексті `f` починається така послідовність символів.

`1`***<eol>*** `-2.9 +13`***<eol>***`2000 777`***<eol>***`<chr(26)>`

Які значення одержать дійсні змінні `a`, `b`, `c`, `d` і який символ стане доступним після виконання:

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)`?

10.8.* Написати програму підрахунку кількості рядків у тексті.

10.9.* "Бінарський алфавіт" складається з букв `a` і `b`. "Бінарська мова" описується РБНФ `S ::= ab | aSb | SS`. Написати програму, яка визначає за словами тексту, чи є вони словами "бінарської мови". Слова, що складаються з символів `a` і `b`, задано по одному на рядок тексту. Довжини рядків не більше максимального значення типу `longint`. Усі результати у вигляді символів `1` або `0` ("так" або "ні") треба вивести в одному рядку іншого тексту.

10.10.* Написати програму генерації тексту з рядками, довжина яких не більше максимального значення типу `longint`. Текст повинен містити рядки, що складаються з символів `a` та `b`, на яких треба перевірити програму визначення слів "бінарської мови".

10.11.* Є два непорожніх тексти, у кожному рядку яких записано додатну цілу константу, і послідовності заданих константами чисел неспадні. Записати в третій текст неспадну послідовність констант, яка є результатом злиття двох даних. Константи вивести по 10 на рядок (останній рядок може бути неповним). Наприклад, при заданих послідовностях (2, 2, 4, 6), (1, 3, 6, 7) утворюється (1, 2, 2, 3, 4, 6, 6, 7).

10.12. Множину цілих представлено файлом цілих, дані якого упорядковано за зростанням. За двома такими файлами створити третій файл, який також є упорядкованим і подає їх:

а) об'єднання;

б) перетин;

в) різницю;

г) симетричну різницю.

10.13. Написати підпрограму, яка форматує текст, зберігає результат у новому файлі та виводить його на екран сторінками. Абзаци у початковому тексті відокремлюються порожнім рядком. Довжина рядків (ширина) у цільовому тексті має становити *W* символів, абзацний відступ — *A* символів. У межах ширини кожен рядок треба:

а) притиснути ліворуч;

б) притиснути праворуч;

в) центрувати;

г) розтягнути по всій ширині, додавши потрібну кількість пропусків.

Згенерувати номери сторінок, на яких уміщається *L* рядків (враховуючи рядок із номером). Номери сторінок відокремлюються від основного тексту одним порожнім рядком і розташовуються на сторінці:

а) знизу;

б) згори

та:

а) праворуч;

б) ліворуч;

в) дзеркально відносно центра (непарні номери праворуч, парні — ліворуч);

г) по центру;

д) праворуч разом із іменем файла;

е) ліворуч, але на першій сторінці номер не друкується.

Слова переносяться тільки тоді, коли вони не уміщаються в рядку цільового тексту (враховуючи абзацний відступ).

10.14. Написати процедуру копіювання текстів із власними внутрішніми буферами розміром у 64 блоки по 512 байт, тобто 32 768 байт, або 32 Кбайт.

10.15. Нехай змінна `c` має тип `char`, при виконанні наведного нижче фрагменту програми послідовно натискаються <Ctrl+Z>, <Enter>, <Enter>, а перед цим буфер клавіатури був порожнім. Що залишиться на екрані?

***а)**
```
read(c); write(' ', ord(c));
if eof then writeln('eof');
if eof then writeln('eof');
read(c); write(' ', ord(c));
read(c); write(' ', ord(c))
```

б)
```
read(c); write(ord(c));
if eoln then writeln('eoln');
if eof then writeln('eof');
read(c); write(' ', ord(c));
read(c); write(' ', ord(c))
```