**ПАСКАЛЬ: РОБОТА З ФАЙЛАМИ**

**1. Фізичні файли та файлові змінні**

Оксфордський тлумачний словник англійської мови подає слово "***файл***" так:

*- це будь-який із різноманітних типів висувних шухляд, полиць, коробок тощо, як правило, з металевим стержнем для тримання паперів разом і в порядку так, щоб їх легко було відшукати;*

*- це ряд людей або речей, розташованих одне за одним.*

Автори операційної системи MS DOS називають файлом *іменовану область на зовнішньому носії інформації*, наприклад, на дискеті. Ця область містить якусь інформацію, наприклад, програму чи вхідні дані до неї. Але як обробку цієї області задавати в Паскаль-програмі? На це питання ми спробуємо відповісти. Принаймні частково й на прикладі системи Турбо Паскаль.

Система програмування Турбо Паскаль, як і інші, має в своєму складі цілий набір засобів роботи з файлами – ***систему введення-виведення***. Мова Паскаль не має операторів, якими описується обробка файлів. Уся обробка задається за допомогою підпрограм і змінних, зібраних у кілька *бібліотек*, або *модулів*. Наприклад, нам уже відомі такі процедури, як readln і writeln.

Почнемо уточнювати поняття файла. Як область на зовнішньому носії інформації, кожний конкретний, фізичний файл є *послідовністю байтів*. Але одну й ту саму послідовність байтів можна розглядати та обробляти і як послідовність байтів, і як послідовність значень якогось типу, і як послідовність символів, розбиту на рядки (інша справа, що за значення чи рядки ми при цьому одержимо).

У Паскаль-програмі файл позначається ім’ ям ***файлової змінної***, тобто вона служить представником файла у програмі. Тип у її означенні задає спосіб, у який розглядається та обробляється файл. Далі будемо називати області на зовнішніх носіях ***фізичними файлами***, а під словом "файл" розуміти саме файлові змінні. У мові Турбо Паскаль є три основні різновиди файлових змінних: типізовані файли, тексти та нетипізовані файли. Оглянемо їх.

***Типізований файл*** розглядається як послідовність змінних деякого типу, заданого в означенні файлової змінної. Її тип задається виразом вигляду

**file of***тип*.

Наприклад,

**var** Fi, Fo : **file of** word;

FF : **file of** real;

або

**type** Student = **record**

Name : String;

Number : Integer;

**end;**

StGroup = **array** [1..25] **of** Student;

**var** Fi, Fo : **file of** StGroup;

Отже, значеннями елементів типізованого файла є значення відповідного типу, наприклад, word, real чи StGroup.

***Текст*** у загальному розумінні – це послідовність символів, розбита на рядки**. *Рядок*** у загальному розумінні – це послідовність символів із явно позначеним кінцем. У мові Турбо Паскаль для файлів-текстів означено спеціальний тип із ім’ ям **TEXT**. Наприклад, можна означити

**var** Fi, Fo : text;

Підкреслимо, що хоча елементами тексту є символи, *цей тип відрізняється від типу* ***file of*** *char*. У текстах є спеціальні символи, якими задаються кінці рядків і навіть кінець тексту. Їх обробка відрізняється від обробки решти символів і задається спеціальними підпрограмами, незастосовними до файлів типу **file of** char, наприклад, readln і writeln.

***Безтипові файли*** розглядаються як послідовності байтів і означаються за допомогою службового слова **file**. Наприклад,

**var** Fi, Fo : **file.**

Для них означено свої підпрограми читання й запису файлів.

Підкреслимо, що *наведені три види файлів відрізняються не стільки можливим змістом їх змінних, скільки наборами підпрограм введення-виведення, застосовних до них*.

Практично всі підпрограми першим параметром мають ім’ я файлової змінної. Далі ім’ я f буде позначати, як правило, саме файлову змінну. Хоча ніхто не заважає давати файловим змінним зовсім інші імена, наприклад, filevariable чи inpfil тощо.

*Робота з файловою змінною починається зі зв’язування її імені 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, зовнішнє ім’я, задається виразом типу string(рядок). Наприклад, якщо рядки s1 і s2 мають значення 'c:\mydir\book' та 'txt', то

assign(f, s1 + '.' + s2);

задає зв’язування з файлом c:\mydir\book.txt.

Файлові змінні з іменами **INPUT** і **OUTPUT** зв’ язуються неявно, тобто без указання в програмі, зі ***стандартними файлами введення й виведення*** – клавіатурою та екраном комп’ ютера. Їх ще називають ***стандартними пристроями введення-виведення***. Означати явно імена input і output та зв’ язувати їх із іншими фізичними файлами не заборонено, але ми цього не рекомендуємо. Вони також можуть не вказуватися у викликах інших підпрограм обробки файлів. У розділі 2 ми вже розглядали найпростіші дії з клавіатурою та екраном.

**Приклад**. Нехай виконується програма з такими означеннями та операторами:

…

**var** f : text;

fn, st : string;

**begin** …

fn := 'default.dat'; {default - "за узгодженням"}

write ('Якщо Ви натиснете ENTER, f буде ');

writeln ('зв''язано з "default.dat". ');

writeln ('Для зміни зв''язування f введіть ім''я файла:');

readln ( st );

**if** st <> '' **then** fn := st;

assign ( f, fn );

…

**end**.

Якщо користувач бажає зв’ язати ім’ я f із файлом, зовнішнє ім’ я якого default.dat, то йому достатньо лише натиснути на клавішу Enter. Якщо йому треба задати інше зв’ язування, то він має набрати на клавіатурі ім’ я фізичного файла, наприклад, newname.dat, і натиснути на Enter.

Отже, після виклику процедури assign фізичний файл можна позначати ім’ ям файлової змінної. Але працювати з його елементами можна лише після ***відкривання*** відповідної файлової змінної. Процедура assign *не задає* *відкривання файла*, воно задається процедурами, які розглядаються далі.

Тут ми опишемо лише ***закривання***файла, що задається процедурою **СLOSE**:

close ( f ) .

Виконання такого виклику не розриває зв’язку імені f із фізичним файлом, але читання та запис за допомогою цього імені неможливі до наступного відкривання чи нового зв’язування з подальшим відкриванням.

*Спроба закрити вже закриту або ще не відкриту файлову змінну призводить до аварійного завершення програми*.

Таким чином, стандартний порядок дій із файловою змінною можна подати у вигляді

*зв’ язування*

*відкривання*

*обробка*

*закривання*

*нове відкривання або зв’ язування та відкривання*

*обробка*

*закривання*

*тощо*

Перш, ніж розглядати підпрограми відкривання та обробки файлів, розглянемо файлові змінні докладніше.

Після зв’ язування фізичний файл стає складовою частиною файлової змінної. Він розташований на зовнішньому носії, а не в оперативній пам’ яті комп’ ютера, тому його елементи не є звичайними змінними програми. Основною особливістю файла є те, що *в будь-який момент виконання програми з усіх його елементів можна обробляти, тобто читати або записувати, лише один*. Він називається ***доступним елементом файла***. Всі інші елементи в цей момент недоступні для обробки.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Елемент 1*** | ***Елемент 2*** | ***...*** | ***Елемент k*** | ***...*** | ***Елемент N*** |

Номер доступного елемента в послідовності елементів файла є значенням спеціальної змінної. Вона не означається в Паскаль-програмі і явно не задається, але вона є й називається ***файловим вказівником*** (рис.13.1). Значення файлового вказівника та відповідно розташування доступного елемента в послідовності змінюється при виконанні підпрограм обробки файлів.

Основними діями при роботі з файлом є читання, тобто копіювання значення доступного елемента в "нормальні" змінні програми, та запис – копіювання значення в доступний елемент. Можливість читання та запису задається ***режимом******доступу*** до файла, або його ***станом***. Таким чином, в будь-який момент виконання програми файл перебуває в певному стані – читання, запису або чогось іншого.

Отже, значення файлової змінної складається з послідовності значень у фізичному файлі, номера доступного елемента та стану. Позначимо це у вигляді (*F*, *i*, *s*), де *F* – послідовність значень елементів <*f*0, *f*1, … , *fn*>, *i* – номер доступного елемента, *s* – стан. Нумерацію елементів ми починаємо з 0, тому що так вона починається в системі Турбо Паскаль. Стани читання та запису позначатимемо літерами *R* та *W* відповідно.

**Приклад**. Вираз (<11, 12, 13, 14>, 1, *R*) позначає значення файлової змінної, складене з послідовності чисел 11, 12, 13, 14, друге з яких (його номер 1!) можна прочитати. Зобразимо це значення так:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *F* | 11 | 12 | | 13 | | 14 | | |
|  |  | | **­** | |  | | Стан | *R* |

Стрілка вказує на доступний елемент, тобто є позначенням файлового вказівника.

Вираз (<11, 12>, 2, *W*)позначає файл із двох елементів:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *F* | 11 | 12 |  |  | |  |
|  |  |  | **­** | Стан | *W* | |

Обидва вони недоступні, файл перебуває в стані запису, до нього можна додати новий елемент (із номером 2) і присвоїти йому якесь значення.

Вираз (<>, 0, *R*) позначає порожній файл, у якому немає елементів.

Підкреслимо, що *кінець типізованого файла ніяк не задається в ньому самому*. Спосіб задання кінця визначається операційною системою і тут не описується.

І останнє зауваження. Елементи типізованих файлів можуть мати будь-який скалярний чи структурний тип, окрім файлового чи структурного з елементами-файлами. Файли також не можуть бути елементами інших структур. Наприклад, недопустимі означення типів

**type** FF = **file** **of** **file** **of** *якийсь тип*;{**???**}

FA = **array** [ 1..10 ] **of** **file** **of** *якийсь тип*; {**???**}

FAF = **file** **of** FA {**???**}.

**2. Послідовний запис у типізовані файли**

*Перш, ніж записувати в файл, слід відкрити його для запису*. Це задається процедурою **REWRITE**. Після виклику ReWrite(f) значення f подається як (<>, 0, *W*), тобто незалежно від попереднього значення f *послідовність у файловій змінній стає порожньою*:

|  |  |  |  |
| --- | --- | --- | --- |
| *F* |  |  |  |
|  | **­** | стан | *W* |

Значення 0 вказівника доступного елемента свідчить про те, що можна записати нове значення в перший, тобто з номером 0, елемент файла. Ця процедура викликається для створення нового файла або поновлення старого зі знищенням даних, що були в ньому. Після її виконання файл називається ***встановленим у початковий стан для запису***.

Запис у файл задається процедурою **WRITE**. При виконанні її виклику

write ( f, *вираз-типу-компонентів-файла*)

обчислюється значення виразу та присвоюється доступному елементу файла, після чого вказівник доступного елемента зсувається на 1 елемент. Наприклад, після виконання останього з наведених операторів у програмі

**program** ...

**var** f : **file of** integer; x : integer

**begin** ...

rewrite(f); x:=2;

write(f, 1); write(f, x); write(f, x\*x);

...

**end**.

файлова змінна f має значення, що подається як (<1, 2, 4>, 3, *W*).

Узагалі, значення (<*f*0 , … , *fn-*1>, *n*, *W*) файлової змінної f при виконанні write(f, *вираз*) міняється на (<*f*0, … , *fn*-1, *V*>, *n*+1, *W*), де *V* позначає значення виразу.

У виклику write можна вказувати довільне число аргументів-виразів того самого типу: write ( f, *вираз1*, ... , *виразN* ). Такий виклик виконується насправді як послідовність

write(f,*вираз1*); … ; write(f,*виразN*).

Наприклад, замість write(f, 1); write(f, x); write(f, x\*x) можна написати write(f, 1, x, x\*x).

Підкреслимо, що *процедуру writeln не можна застосовувати до типізованих файлів*. Вона призначена лише для роботи з текстами.

**Приклад 13.1.** Треба створити файл із такими даними про студентів групи: прізвище та ім’ я студента, а також середній бал його успішності.

Дані про студента можна подати записом типу

Student = **record**

SName, Name : string[20];

Ball : real

**end**.

Зв’ язування та встановлення файла в початковий стан для запису оформимо процедурою OpenFile, а створення – процедурою CreatFile у такій програмі:

**program** InPutGroup;

**type**

Student = **record**

Sname, Name : string[20];

Ball : real;

**end**;

**var**

Fi : **file of** Student; { Файлова змінна Fi – інформаційний файл }

FileName: string; { Змінна для введення імені файла }

**procedure** OpenFile;

**begin**

writeln('Задайте ім''я файла, що буде створюватися: ');

readln(FileName);

assign(Fi, FileName);

rewrite(Fi);

**end;**

**procedure** InPutFile;

**var** St : Student;

Ch : char;

**begin**

**repeat**

writeln('Задайте дані про студента:');

write('Прізвище>'); readln(St.SName);

write('Ім''я>'); readln(St.Name);

write('Середній бал успішності>'); readln(St.Ball);

write(Fi, St); {\*\*\*}

write('Чи треба вводити дані про нового студента? "Y"/"N"> ');

readln**(**Ch);

**until** (Ch = 'n') **or** (Ch = 'N');

close(Fi);

**end**;

**begin**

writeln('Програма створення файла даних про успішність');

openfile; { Виклик процедури відкривання файла }

creatfile; { Виклик процедури створення файла }

**end.**

**3. Послідовне читання типізованих файлів**

*Для того, щоб читати вже створений типізований файл, треба відкрити його для читання* викликом процедури **RESET**(f). Після її виклику файлова змінна установлюється в ***початковий стан для читання***, а номером доступного елемента стає 0 незалежно від того, чи є взагалі елементи в файлі. Значення f можна подати як (*F*, 0, *R*), тобто

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *F* | *f*0 | *f*1 | ... | |
|  | **­** |  | Стан | *R* |

Якщо *F* є порожньою послідовністю, *F*=<>, то спроба читання з файла завершується аварійно, оскільки доступного елемента просто немає!

Файл із диску можна читати *тільки після установлення файлової змінної в початковий стан для читання*, тобто *установлення обов’ язкове*. Можливість чи заборона установлення файлової змінної, зв’ язаної з клавіатурою, залежить від системи програмування. У системі Турбо Паскаль таке установлення можливе, хоча й не обов’ язкове.

Зазначимо, що в програмі може бути довільна кількість викликів процедури reset, і що система Турбо Паскаль дозволяє після виконання її викликів не тільки читати з файла, а й записувати в нього, тобто змінювати його перший елемент. І взагалі, система дозволяє змінювати за допомогою write доступний елемент файла, яким би не було його місце в послідовності і за будь-якого стану файла.

Читання доступного елемента файла задається процедурою **READ**. Її виклик має вигляд

read(f, v),

де v – ім’ я змінної того ж типу, що і в елементів файла. В результаті значення доступного елемента присвоюється цій змінній, а вказівник доступного елемента переміщується на наступний елемент.

Наприклад, при читанні read(f, x) файлової змінної f із значенням

(<11, 12, 13>, 0, *R*)

змінна x набуває значення 11, а значення f подається як

(<11, 12, 13>, 1, *R*),

тобто доступним стає елемент 12.

Узагалі, якщо файл містить *n* елементів, пронумерованих 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 та перехід від значення файлової змінної (*F*, *i*, *R*) до (*F*, *i*+1, *R*). Якщо ж *i*=*n*, то кажуть, що файл f ***прочитано***, або ***вичерпано*** (зокрема, коли він порожній, тобто *F* = <>). *За такого значення i виклик read призводить до аварійного завершення програми*.

У виклику процедури read можна вказувати довільну кількість аргументів, що є іменами однотипних змінних:

read ( f, v1, v2, … , v*N* ).

Такий виклик виконується насправді як послідовність викликів

read(f, v1 ); read(f, v2 ); ... read(f, v*N* ).

Зрозуміло, треба гарантувати при цьому, що в файлі залишилося достатньо непрочитаних елементів.

Підкреслимо, що до типізованих файлів незастосовна процедура читання readln – її можна вживати лише для читання текстів.

Визначення того, чи прочитано вже файл, задається функцією **EOF**. За виклику eof(f) повертається бульове значення **false**, якщо доступний який-небудь елемент файла, тобто значення виразу *i*<*n* у значенні файлової змінної (<*f*0, … , *fn*-1>, *i*, *R*). Значення **true** повертається за *i*=*n*, що можливо після читання всіх елементів файла без установлення його в початковий стан.

Виклик функції eof дозволяє визначити, чи є ще у файлі непрочитані елементи, та запобігти читання з вичерпаного файла.

*Практично завжди програму варто записувати так, щоб у процесі її виконання перевірка невичерпаності файла передувала виклику процедури read.*

**Приклад 2.** Напишемо програму обчислення середнього арифметичного *A* цілих чисел непорожнього файла nums.dat та запису в інший файл всіх його чисел, менших *A*.

Розв'яжемо задачу в такий спосіб:

1) *прочитати всі числа з файла, обчислити їх суму й кількість і визначити A;*

1. *повторно прочитати всі числа, копіюючи з них лише менші від A.*

Нехай числа записано в файлі цілих з ім'ям nums.dat, а числа, менші за середнє, переписуються в файл littls.dat. Наведені дії задаються програмою

**program** numbers;

**var** f , g : **file** **of** integer;

v : integer; a : real;

n : integer;

**begin**

assign ( f, 'nums.dat' );

{1}reset ( f );

read ( f, v ); {читання першого елемента без перевірки !}

a:=v; n := 1;

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

**begin** {з виклику eof(f) повернулося **false**,}

{тому можна читати доступний елемент}

read ( f, v );

a := a + v; n := n + 1

**end**;

{з виклику eof(f) повернулося **true**}

a := a/n;

{2}reset ( f );

assign(g, 'littls.dat'); rewrite(g);

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

**begin**

read ( f, v ); **if** v < a **then** write(g, v )

**end**;

close ( f ); close(g);

**end**.

Як бачимо, читання першого елемента задано без перевірки, чи можна взагалі його прочитати. У разі порожнього файла виконання програми завершується аварійно.

**Приклад 3.** Вивести на екран комп’ ютера дані про студентів з файла, створеного за програмою з прикладу 13.1.

Природньо розглядати файл як послідовність записів типу Student. Його елементи по одному читаються в допоміжну змінну St та значення її полів (прізвище, ім’ я й середній бал), що мають базовий тип або є рядками, виводяться на екран із його попереднім очищенням. Очищення екрану задається викликом процедури **CLRSCR**,що входить до складу модуля Crt.Після кожного виведення для одержання нових даних слід натиснути на клавішу Enter.

У наступній програмі використовується змінна **IOResult**, означена в модулі **Dos**. Вона набуває ненульового значення, якщо при виконанні процедури Reset виникла якась помилка, наприклад, зв’ язана з тим, що користувач задав ім’ я неіснуючого файла. Перед її викликом вимикається режим перевірки правильності введення-виведення, оскільки в цьому режимі за помилкового виконання Reset програма аварійно завершується. Після виклику Reset режим перевірки вмикається знов.

**program** OutPutGroup;

**uses** Crt**,** Dos; { Підключення модулів Crt та Dos }

**type** Student = **record**

Sname, Name : string[20]; Ball : real;

**end**;

**var** Fi : **file** **of** Student;

FileName: string; { Змінна для введення імені файла }

**procedure** OpenFile;

**var** Rez : Integer; Ch : char;

**begin**

Rez:=1;

**while** Rez <> 0 **do** { Rez=0 – ознака того, що введено} {правильне зовнішнє ім’я файла}

**begin**

writeln('Задайте ім''я файла'); readln(FileName);

assign(Fi, FileName);

{$I-} {Вимкнення контролю правильності читання/запису}

reset(Fi);

{$I+} { Увімкнення контролю }

Rez:=IOResult;

**if** Rez <> 0 **then**

**begin**

writeln('Перевірте правільность імені файла !');

writeln('Чи буде повторне задання імені? "Y"/"N":');

readln(Ch);

**if** (Ch = 'n') **or** (Ch = 'N') **then halt**(0);{Вихід із програми}

**end**;

**end**;

**end**;

**procedure** OutPutFile;

**var** St : Student; Ch : char;

**begin**

writeLn('Чи  бажаєте дивитися дані про студентів ? "Y"/"N" ');

readln(Ch);

**if** (Ch = 'Y') **or** (Ch = 'y') **then**

**begin**

**while not eof**(Fi) **do**

**begin**

read(Fi, St); clrscr;

writeln('Прізвище : ', St.SName);

writeln('Ім''я: ', St.Name);

writeln('Середній бал: ', St.Ball);

readln;

**end**;

**end**;

close(Fi);

**end**;

**begin**

clrscr; writeln('Програма друкування даних про студентів');

openfile; outPutfile;

**end.**

**Задачі**

**1.**\* Пояснити, що задає та як використовується пpогpама

**program** filcrout;

**type** Tel = **record** nam : string; num : integer **end**;

**var** f : **file** **of** Tel; x : Tel; s : string; eel : boolean;

**procedure** readel ( **var** x : Tel );

**begin**

x.num:=0; readln ( x.nam);

**if** x.nam <> '' **then** readln ( x.num )

**end**;

**begin**

writeln ( 'Введіть ім''я файла:' ); readln ( s );

assign ( f, s ); rewrite ( f );

**repeat**

readel ( x ); eel := (x.nam = '');

**if** **not** eel **then** write ( f, x );

**until** eel;

reset(f);

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

**begin**

read ( f, x ); writeln ( x.num, ': ', x.nam )

**end**

**end**.

**2.**\* Переписати програму з прикладу 13.2, щоб у разі порожнього файла її виконання не завершувалося аварійно.

**3.** Переписати програму з прикладу 13.2, щоб числа не копіювалися в інший файл, а друкувалися на екрані.

**4.**\* Написати пpоцедуpу пpисвоювання файлів шляхом копіювання.

**5.**\* Написати функцію пеpевіpки побайтової pівності двох файлів.

**6.**\* Написати пpоцедуpу дописування до елементів пеpшого файла елементів другого, із зберіганням pезультату

а) в новому файлі; б) в першому файлі.

**7.**\* Написати пpоцедуpу виведення змісту файла з даними про студентів на екpан "стоpінками": після друкування на екрані даних про чергових 5 студентів виводиться запит щодо продовження, і виконання програми призупиняється до того, як користувач підтвеpдить або не підтвердить пpодовження.

**4. Прямий доступ у системі Турбо Паскаль**

Як ми побачили в трьох попередніх підрозділах, *доступність* елемента файла, тобто можливість його читання чи створення в ході виконання програми, залежить від його розташування в послідовності. Досі ми розглядали підпрограми *послідовного**доступу до елементів файла*. Він полягає в тім, що елементи файла не задаються явно, а доступність їх у ході виконання програми цілком визначається їх розташуванням у послідовності. Спочатку доступний перший елемент, після його обробки – другий тощо.

Але послідовний доступ елементів не завжди зручний. Чи не замислювався читач над тим, як запрограмувати читання з типізованого файла елемента за його номером або його заміну, додавання чи вилучення ?

Зрозуміло, що задати читання елемента за номером k можна так:

reset(f);

**for** i:=1 **to** k-1 **do** read(f, x); {пропущено k-1 елемент – доступний k-й}

read(f, x).

Для заміни елемента файла за його номером k можна "вийти на нього" шляхом читання попередніх. Далі можна скористатися одним недоліком системи Турбо Паскаль. Справа в тім, що *система дозволяє в стані читання записувати в файл значення змінних (і лише змінних!)*. Отже, заміну елемента можна описати так:

reset(f);

**for** i:=1 **to** k-1 **do** read(f, x);

{пропущено k-1 елемент - доступний якраз k-й}

x:=...; write(f, x).

Описати в такому ж дусі вилучення й додавання елемента до файла ми залишаємо вправою для наддопитливих читачів. *Але все це "штучки", якими користуватися не варто.*

Натомість розглянемо *прямий**доступ до елементів файла*. Його суть у тім, що елементи задаються номерами в послідовності, яка утворює файл. Такий доступ здійснюється за допомогою спеціальних підпрограм.

Основною є процедура **SEEK**. У її виклику задається ім’ я файлової змінної та номер того елемента файла, який стає доступним після виконання виклику. Номер задається виразом типу LongInt. Наприклад, після виклику

Seek ( f, 2)

доступним стає третій елемент, оскільки нумерація починається з 0:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *f*0 | *f*1 | *f*2 | *f*3 | ... | *fN* |

**­**

Значення саме цього елемента буде читатися за виконання виклику процедури введення *read* чи цьому елементу буде щось присвоюватися за виконання *write*. В обох випадках доступним стане наступний елемент:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *f*0 | *f*1 | *f*2 | *f*3 | ... | *fN* |

**­**

Підкреслимо, що виклик процедури Seek записується після відкривання 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 також кілька допоміжних процедур, що застосовуються разом із процедурою Seek.

Функція **FILEPOS** задає повернення номера доступного елемента. Єдиним аргументом у її виклику є ідентифікатор файлової змінної, а повертається значення типу LongInt. Наприклад, за останнього зображеного значення файлової змінної f присвоювання

A := FilePos ( f );

надає змінній А типу LongInt значення 3.

Для визначення загальної кількості елементів у файлі використовують функцію **FILESIZE**. Її единим параметром є ідентифікатор файлової змінної, і з її виклику повертається значення типу LongInt. Наприклад, значенням змінної N типу LongInt після присвоювання

N := FileSize ( f )

стає кількість елементів у файлі.

Зрозуміло, що використовуючи у програмі виклик процедури seekв парі з викликами read або write, ми зможемо прочитати будь-який елемент файла чи зробити заміну його значення.

Зокрема, за допомогою процедур seek, filesizeі write можна розширити файл, дописуючи значення нового елемента в кінець:

seek ( f, filesize ( f ));

write ( f, v ).

Дійсно, після виклику seek файловий вказівник встановлюється за останнім елементом, тобто

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *f*0 | *f*1 | *f*2 | ... | *fN* |

**­**

а після виклику write значення *v* записується в новий елемент, після чого файловий вказівник переміщається вправо:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *f*0 | *f*1 | *f*2 | ... | *fN* | *fN*+1 |

**­**

Процедура **TRUNCATE** задає знищення решти файла, починаючи від доступного елемента. Наприклад, після виконання викликів

seek(f, 3); truncate(f)

елементи з 3-го по останній знищуються, а залишаються з номерами 0, 1 і 2.

Використання процедур прямого доступу дозволяє *вилучати елементи з файла*.

**Приклад 4.** Наведемо програму, яка задає вилучення непотрібних елементів файла, тобто його *стискання*.

Нехай у файлі Group.dat зберігається інформація про студентів групи: прізвище, ім’я та середній бал. З клавіатури задається прізвище студента, який вибув – запис про нього треба вилучити з файла.

За наступною програмою файл читається до кінця і в допоміжний файл копіюються ті записи, поле-прізвище яких відрізняється від заданого. Далі файли закриваються, і засобами модуля System старий файл просто знищується, а допоміжному присвоюється зовнішнє ім’ я старого.

**program** OutFromGroup;

**type** Student = **record**

Sname, Name : string[20];

Ball : real;

**end**;

**var** Fi, Fo : **file** **of** Student; { інформаційний та допоміжний файл }

FileName: string; { ім’ я файла }

**procedure** OpenFile;

**begin**

**writeln**('Задайте ім''я файла'); **readln**(FileName);

**assign**(Fi, FileName); **reset**(Fi);

**assign**(Fo, 'NewFile.dat'); **rewrite**(Fo);

**end**;

**procedure** ClearFile; { Процедура стискання файла }

**var** St : Student; { Змінна для обміну }

StudtoOut : string[20];

**begin**

**writeln**('Задайте прізвище студента, що вилучається:');

**readln**(StudtoOut);

**while not eof**(Fi) **do**

**begin**

**read**(Fi, St);

**if** St.SName <> StudtoOut **then**

**write**(Fo, St)

**end**;

**close**(Fi);

**close**(Fo);

{Виклики процедур модуля System }

**Erase**(Fi); {для знищення}

**ReName**(Fo, FileName); {та переіменування файла }

**end**;

**begin**

OpenFile;

ClearFile;

**end.**

Крім операцій заміни та вилучення елементів файла, опишемо операцію вставки елемента в довільне місце файла. Нехай місце задається номером нового елемента в файлі. Для вставки використовують один із двох алгоритмів.

У першому алгоритмі використовується допоміжний файл, в який переписуються всі елементи, що передують заданому.

*Відкрити основний та допоміжний файли.*

*У циклі переписати з основного файла в допоміжний всі елементи, номери яких менші заданого. Для цього можна використати допоміжну змінну того ж типу, що і в елементів файла.*

*У допоміжний файл записати значення, яке треба вставити.*

*У циклі переписати з основного файла в допоміжний усі елементи, що залишились.*

*Закрити основний та допоміжний файли.*

*Знищити основний файл.*

*Переіменувати допоміжний файл в основний.*

У другому алгоритмі замість допоміжного файла використовуються дві допоміжні змінні того ж типу, що і в елементів файла, та допоміжна змінна-лічильник типу LongInt для запам’ятовування поточного місця вставки.

*Першій допоміжній змінній присвоїти значення, яке треба вставити в файл.*

*Встановити файловий вказівник на місце вставки за допомогою процедури Seek.*

*Запам’ятати місце вставки в змінній-лічильнику за допомогою функції FilePos.*

*Прочитати значення того елемента, на який вказує файловий вказівник, і присвоїти другій допоміжній змінній.*

*Знов установити файловий вказівник на місце вставки за допомогою процедури Seek, використавши значення лічильника.*

*В циклі, поки файл не прочитано:*

*записати в доступний елемент файла значення з першої змінної;*

*запам’ятати нове місце вставки, збільшивши лічильник;*

*першій допоміжній змінній присвоїти значення другої;*

*прочитати значення того елемента, на який вказує файловий вказівник, і присвоїти другій допоміжній змінній;*

*установити файловий вказівник на місце вставки за допомогою процедури Seek, використавши значення лічильника.*

*Записати в доступний елемент файла значення з першої змінної.*

Наостанок зауважимо, що значення другого аргументу у виклику процедури Seek *може не бути номером елемента* в файлі, тобто може бути від’ ємним або більшим кількості елементів файла. За її виклику нічого не трапиться, але за подальшої спроби читати елемент із установленим номером виконання програми аварійно завершиться. Адже елемента з таким номером немає!

Ще цікавіше буде, якщо у виклику Seek задати номер, більший від кількості елементів у файлі, а потім спробувати записати в цей ніби доступний елемент файла. Все буде гаразд, але після цього в файлі чомусь з’ явиться зовсім незрозуміла інформація, "сміття". Перевірте цю інформацію та поміркуйте, звідки береться "сміття"! І будьте уважні, коли записуєте другий аргумент процедури Seek.

**Задачі**

**8.** Написати "дурнестійкий" варіант процедури Seek, за яким не дозволяється задавати від’ ємний чи більший від кількості елементів файла номер елемента. Зауважимо, що варіант такої процедури для подальшого читання відрізняється від варіанта для запису тим, що при читанні найбільшим номером доступного елемента є Filesize(f)-1, а при записі – Filesize(f).

**9.** Написати програми, що реалізують алгоритми вставки нового значення в файл.

**10.** Написати програму вилучення елемента з файла даних про студентів групи, аналогічну програмі з прикладу 13.4, в якій не використовується допоміжний файл (доцільно застосувати процедуру Truncate).

**11.** Написати процедуру вилучення з файла елемента за його номером.

**12.** Написати програму створення, поповнення, перегляду та обробки файлів записів про студентів груп засобами послідовного та прямого доступу. Використання файлів ведеться за допомогою меню. Воно повинно бути змістом окремого файла та складатися з пунктів, що позначають конкретні операції з файлами чи кінець роботи з меню. Використання меню полягає в тім, що воно виводиться на екран, а виконання програми забезпечує можливість указання з клавіатури, який пункт меню вибирається. У відповідь на указання залежно від обраного пункту забезпечується виконання операції з файлом. Кінець роботи з меню означає завершення програми.

Файли повинні задовольняти *умови правильності*: в одному класі не може бути двох учнів з однаковими прізвищами та іменами одночасно.

Програма має забезпечити такі операції з файлами:

"***установка* *файла***" – задання імені файла;

"***поповнення***" – введення записів із клавіатури й додавання їх до файла (з перевіркою умов правильності);

"***перегляд***" – виведення записів файла на екран сторінками.

"***вибірковий перегляд за полем***" – задання значення поля та перегляд лише записів файла з таким значенням цього поля;

"***вибіркове копіювання за полем***" – задання значення поля та копіювання в інший файл записів тільки з таким значенням цього поля;

"***вибіркове вилучення за полем***" – задання значення поля та вилучення записів із заданим значенням цього поля, тобто копіювання в інший файл записів із іншими значеннями цього поля;

"***перевірка умов правильності***" – файл не повинен вміщувати записи-дублі та записи, що порушують умови правильності;

"***злиття***" – поповнення файла записами іншого файла з відкиданням тих записів, що порушують умови правильності.