Методические указания

Лабораторная работа 10

Пользовательские типы данных. Работа с файлами.

Пользовательские типы данных

Описание пользовательских типов

Множество типов данных языка Pascal можно разделить на две группы: ***стандартные*** и ***пользовательские.***

Стандартные типы (например, `integer`, `real`, `boolean`, `char`, `string`, `array`) предопределены и действуют в любой точке программы. На самом деле они описаны в стандартном модуле `system`, который по умолчанию подключается в список используемых модулей каждой программы, независимо от того, указан он в разделе `uses`, или нет. Кстати, также в этом модуле описаны стандартные процедуры и функции `write`, `writeln`, `read`, `sin`, `sqrt`, `length` и все другие.

Пользовательские типы – это дополнительные типы, которые программист-пользователь может задавать самостоятельно.

Задать тип данных – значит, **определить множество его допустимых значений** и связать с этим множеством имя типа. Описание пользовательского типа имеет синтаксис:

```
ИмяТипа = МножествоДопустимыхЗначенийТипа
```

Пользовательские типы задаются в **разделе описания типов,** который обозначается ключевым словом `type`, и может содержаться в произвольном месте описательной части программы или подпрограммы:

```
program ИмяПрограммы; {заголовок программы}
uses  ...; {подключение модулей}
const   {задание констант}
  ...
type   {описание пользовательских типов}
  ИмяТипа1=ЗначенияТипа1;
  ИмяТипа2=ЗначенияТипа2;
  ...
procedure   {описание процедур}
  ...
function   {описание функций}
  ...
var   {описание переменных}
  ...
begin  {раздел операторов (тело) программы}
  ...
  ...
end.
```

описательная часть

испол-нительная часть

После описания типов их идентификаторы можно использовать для описания переменных.

В простейшем случае выражение `ЗначенияТипа`, стоящее справа от знака равенства может быть именем одного из стандартных типов, например:

```
type
  ext = extended;
  integer = longint;
```

В первом случае длинное `extended` заменено коротким `ext`. Во втором случае переопределен стандартный тип `integer`, который после этого будет иметь диапазон типа `longint`. Однако и после такого переопределения остается возможность обращения к первоначальному типу `integer` с помощью ***квалифицируемого (уточненного) идентификатора:***

```
var
  a : integer;   {a может изменяться в диапазоне типа longint}
  b : system.integer;  {b может изменяться от -32768 до 32767}
```

В других случаях используются различные виды пользовательских типов, описанные далее.

Виды пользовательских типов

К пользовательским типам относят:

- перечисляемый тип;
- интервальный тип;
- указательные типы;
- структурированные типы:
  - ♦ тип-массив (`array`),
  - ♦ файловый тип (`file`),
  - ♦ тип-множество (`set`),
  - ♦ тип-запись (`record`),
  - ♦ объектный тип (`object`);
- процедурный тип.

будут описаны на следующих занятиях

Из всех видов пользовательских типов только перечисляемый и интервальный типы являются порядковыми.

Перечисляемый тип

Каждое значение перечисляемого типа программист задает сам (т.е. «перечисляет» все возможные значения). Описание перечисляемого типа состоит из списка его элементов (через запятую), заключенного в круглые скобки. Каждый из элементов – уникальный идентификатор. Пример:

```
type
  season = (spring, summer, autumn, winter);
  weekday = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
var
  s1, s2 : season;
  today : weekday;
```

Идентификаторы всех элементов перечисляемого типа интерпретируются как константы. В примере идентификаторы `spring`, `summer`, `autumn`, `winter` – это константы типа `season`. Такие идентификаторы не являются строковыми константами и **в кавычки не заключаются.**

Описание одного и того же идентификатора в разных типах считается **ошибкой.** В следующем примере будет выдано сообщение об ошибке:

```
type
  weekday = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
  workday = (Mon, Tue, Wed, Thu, Fri);
```

Значениями перечисляемого типа не могут быть константы предопределенных типов. Примеры некорректных описаний типов:

```
type
  digits = (0,1,2,3,4,5,6,7,8,9);
  digSymbols = ('0','1','2','3','4','5','6','7','8','9');
  weekday = ('Mon','Tue','Wed','Thu','Fri','Sat','Sun');
```

Перечисление элементов типа определяет упорядоченные наборы констант. Порядковый номер константы определяется ее позицией в списке, причем первая константа имеет порядковый номер 0, вторая – 1, и т.д.

К значениям перечисляемых типов **не применимы** ни стандартные **арифметические операции,** ни стандартные **процедуры ввода-вывода** (`write/writeln`, `read/readln`).

Интервальный тип

Интервальный тип данных – это диапазон (интервал) значений какого-либо порядкового типа, называемого базовым. При описании интервального типа указывается наибольшее и наименьшее значения диапазона, разделенные лексемой «. .» (две точки). Например:

```
1 .. 10          -1..1          'A' .. 'Z'
0 .. 500       -128..127         'A'..'z'
```

Пример описания типов и переменных:

```
const
  min=1;
  max=31;
type
  month = 1..12;
  date = min..max;
var
  m1 : month;
  m2 : 1..12;
  today : date;
```

В качестве границ диапазона можно использовать константы из описания перечисляемого типа:

```
type
  weekday = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
  workday = Mon .. Fri;
var
  d1 : workday;
  d2 : Mon..Fri;
```

Тип-массив

Предварительное описание типа-массива считается более строгим описанием массива. Описание типа одномерного массива:

```
type
  ИмяТипа = array[НижнГран..ВерхГран] of ТипЭлементов;
var
  ИмяМассива : ИмяТипа;
```

Пример, тип-массив из ста вещественных чисел:

```
const
  n=100;
type
  TArr:array[1..n] of real;
var
  m:TArr;
```

Предварительное описание типа-массива **необходимо** при использовании имени массива в качестве **параметра процедуры или функции.** Пример:

```
const n=5;
type
  TArr = array[1..n] of real;

function sum(a:TArr):double;
  var s:double;
      i:integer;
  begin
    s:=0;
    for i:=1 to n do s:=s+a[i];
    sum:=s;
  end;

var
  m:TArr;
  i:integer;
begin
  for i:=1 to n do
    begin
      write('Введите ',i,'-й элемент массива:');
      readln(m[i]);
    end;
  writeln('Сумма элементов массива = ',sum(m));
end.
```

Файловый тип данных

Понятия физического и логического файла

Обычно, понятие ***файл*** используется в одном из двух значений.

Во-первых, ***физический файл*** (т.е. существующий физически на конкретном материальном носителе информации) – это именованная область внешней памяти, содержащая какую-либо информацию.

Во-вторых, ***логический файл*** (т.е. существующий в нашем логическом представлении) – это одна из многих структур данных, используемых в программировании.

Структура физического файла представляет собой последовательность байт памяти носителя информации:

байт	байт	байт	…	байт	байт	байт

Структура логического файла – это способ восприятия файла в программе – «**шаблон**», через который мы смотрим на физическую структуру файла. В языках программирования таким «шаблонам» соответствуют ***файловые типы*** данных. Образное представление логического файла:

элемент	элемент	элемент	…	элемент	элемент	конец файла (`eof`)

где все элементы имеют одинаковый тип.

Структура логического файла похожа на структуру одномерного массива. **Различия** заключаются в следующем, у файла:

- количество элементов в каждый момент времени не известно, оно может изменяться в процессе работы программы;
- нумерация элементов начинается с нуля;
- в конце располагается ***символ конца файла*** (**`eof`** – end of file) – управляющий символ SUB с ASCII-кодом `#26`.

Файловый тип

В языке Pascal имеются три типа файлов:

- типизированные (`file of Тип`),
- текстовые (`text`),
- нетипизированные (`file`).

Для работы с файлами необходимо в разделе описания определить ***файловые переменные*** (которые также можно назвать логическими файлами). Например:

```
var  f1,f2 : file of integer;   {файлы из элементов типа integer}
     list : text;   {текстовый файл}
     ftmp : file;   {нетипизированный файл}
```

Доступ к элементам файла

В любой момент в программе доступен **только один** элемент файла, на который ссылается ***указатель текущей позиции файла (указатель обработки).*** Он определяет место программы, откуда (куда) происходит чтение (запись) данных.

При открытии или создании файла указатель помещается в его начало. Чтение или запись данных из файла вызывает автоматическое перемещение указателя. Указатель ведет себя **подобно курсору** в текстовом редакторе.

По способу доступа к элементам различают файлы ***последовательного*** и ***прямого*** доступа.

Последовательный доступ к элементам файла выполняется в той же последовательности, в какой они записывались. При поиске нужного элемента необходимо перемещать указатель обработки до тех пор, пока не будет найден искомый элемент. При таком способе доступа запрещено совмещение чтения и записи данных.

Прямой доступ осуществляется по адресу (номеру) элементов. При поиске элемента достаточно указать номер его позиции. Совмещение чтения и записи данных допустимо.

К текстовым файлам (типа `text`) всегда выполняется только последовательный доступ.

Общая схема работы с файлами

Последовательность действий

При работе с любым файлом необходимо выполнять следующие действия:

1. Описать переменную файлового типа.
2. Файловую переменную поставить в соответствие конкретному физическому файлу.
3. Открыть файл, т.е. сделать его доступным для ввода и/или вывода. (Если файл отсутствует, то он должен быть создан.)
4. Обработать файл.
5. Закрыть файловую переменную, т.е. сохранить данные на внешнем устройстве после окончания работы с файлом.

Общие процедуры и функции

В языке Pascal содержатся стандартные процедуры и функции (описаны в модуле `system`), применимые для файлов любых типов.

Процедуры (далее `ФайлПерем` – файловая переменная):

`assign(``ФайлПерем,ИмяФайла`**`);`**	Логический файл `ФайлПерем` ставится в соответствие (***связывается***) физическому файлу с именем `ИмяФайла` (выражение строкового типа). Процедуру `assign` нельзя использовать для уже открытого файла.
	`assign(f1,'myfile.dat');` `assign(f2,'C:\WORK\report.txt');`
`reset(``ФайлПерем`**`);`**	Открывает существующий файл для **чтения.**
	`reset(f1);`
`rewrite(``ФайлПерем`**`);`**	Создает и открывает новый файл для **записи.** Если файл уже существует, то он *удаляется,* и создается новый *пустой* файл с тем же именем.
	`rewrite(f2);`

close(ФайлПерем**);**	**Закрывает** открытый файл (иначе может произойти *потеря данных*). При закрытии физический файл обновляется, и в конец добавляется символ конца файла (eof).
	close(f1); close(f2);
rename(ФайлПерем,ИмяФайла**);**	Переименовать внешний файл (где ИмяФайла – новое имя).
	rename(f,'newname.dat');
erase(ФайлПерем**);**	Удалить внешний файл.
	erase(f);

Функции:

ioresult	Возвращает **код** последней выполненной **операции ввода/вывода.** Код=0, если операция прошла успешно.
	if not (ioresult=0) then writeln('Ошибка ввода/вывода.');
eof(ФайлПерем**)**	Выполняет проверку, достигнут ли **конец файла** при чтении данных. Если да, то возвращает true.

	while not eof(f2) do
	begin
	read(f2,a);
	writeln(a);
	end;

Типизированные файлы

Описание типизированных файлов

Типизированный файл состоит из последовательности элементов **одного типа.** Нумерация начинается с нуля. В каждый момент времени доступен только один текущий элемент. После выполнения чтения или записи элемента в позиции N, указатель автоматически перемещается к следующему по порядку элементу N+1.

Объявление типизированных файлов:

```
type
  ИмяТипа = file of Тип;
var
  ФайлПерем : ИмяТипа;
```

или

```
var
  ФайлПерем : file of Тип;
```

где Тип – любой тип, кроме файлового и опирающегося на файловый.

Процедуры и функции для типизированных файлов

Процедуры:

read(ФайлПерем,x1,x2,...,xN**);**	**Чтение** из файла.
write(ФайлПерем,x1,x2,...,xN**);**	**Запись** в файл.
seek(ФайлПерем,НомерПозиции**);**	**Перемещает указатель** обработки с текущей позиции к позиции с указанным номером, не выполняя чтение или запись.
truncate(ФайлПерем**);**	**Усекает размер файла** до его текущей позиции. Все элементы за текущей позицией удаляются, и она становится концом файла.

здесь x1,x2,...,xN – список переменных одного типа (через запятую).

Функции:

filepos(ФайлПерем**)**	Возвращает **текущую позицию** в файле (целое число типа longint). (Нумерация начинается с нуля.)
filesize(ФайлПерем**)**	Возвращает **число элементов** файла (целое число типа longint). (Если файл пуст, то – нуль.)

Текстовые файлы

Описание текстовых файлов

Текстовый файл состоит из последовательности символьных строк произвольной длины. Каждая строка в текстовом файле оканчивается составным ***символом конца строки*** (**eoln** – end of line), состоящим из двух символов – «перевод строки» **#10**(LF) и «возврат каретки» **#13**(CR). В конце последней строки ставится символ конца файла (eof). Схема текстового файла:

символ	символ	...	символ	символ	конец строки (eoln)	
символ	символ	...	символ	символ	символ	конец строки (eoln)
символ	символ	...	символ	конец строки (eoln)		
символ	символ	конец строки (eoln)				
символ	символ	...	символ	символ	символ	конец файла (eof)

Объявление текстовых файлов:

```
type
  ИмяТипа = text;
var
  ФайлПерем : ИмяТипа;
```

или

```
var
  ФайлПерем : text;
```

К текстовым файлам выполняется только последовательный доступ.

Процедуры и функции для текстовых файлов

Процедуры:

append(ФайлПерем**);**	Открывает существующий файл для **дозаписи в конец.** (Указатель устанавливается в конец файла.)
read(ФайлПерем,x1,x2,...,xN**);** **readln(**ФайлПерем**);** **readln(**ФайлПерем,x1,x2,...,xN**);**	**Чтение** из файла. (…**ln** – с переводом на новую строку.)
write(ФайлПерем,x1,x2,...,xN**);** **writeln(**ФайлПерем**);** **writeln(**ФайлПерем,x1,x2,...,xN**);**	**Запись** в файл. (…**ln** – с переводом на новую строку.)

здесь x1,x2,...,xN – список переменных разных типов (через запятую).

Функции:

eoln(ФайлПерем**)**	Выполняет проверку, достигнут ли **конец строки или конец файла.** Если да, то возвращает true.
seekeoln(ФайлПерем**)**	Аналогична eoln, но пропускает пробелы и табуляции перед проверкой.
seekeof(ФайлПерем**)**	Аналогична eof, но пропускает пробелы и табуляции перед проверкой.

Содержание