**Теория!**

* Записи.

**Запись**в ТР является одним из самых общих и структурированных типов. Запись состоит из фиксированного числа компонент (полей), которые могут быть различных типов.

**тип\_запись**::= "**record**"

[ список\_полей ]

"**end**"

**список\_полей** ::= ( фиксированная\_часть

[";" вариантная\_часть] [";"]) |

(вариантная\_часть [";"])

**фиксированная\_часть**::= список\_ИД ":" тип

{ ";" список\_ИД ":" тип }

Примеры определения типа запись с фиксированной частью:

**type**

**Book=record**{определяем тип записи - "Книга"}

**Title:string[40];**{название книги}

**Autor:string[50];**{автор}

**Publ:string;** {издательство}

**Date:integer;** {год издания}

**end;**

**Date=record**{определяем тип записи - "Дата"}

**Year: integer;**{год}

**Month: 1..12;** {месяц}

**Day: 1..31;** {день}

**end;**

Заметьте, что одно из полей записи **Book**имеет то же имя, что и запись **Date**. ТП не считает это ошибкой, так как одинаковые идентификаторы находятся, тем не менее на разных уровнях.

Для обращения к определенному полю переменной типа запись требуется указать имя переменной, точку и имя поля.

Например:

**…**

**var A:Book;**

**begin**

**...**

**a.title:=’Кризис империи’;**

**a.Autor:=’Дэвид Дрейк’;**

**a.Publ:=’Армада’;**

**a.Date:=1997;**

**...**

**end.**

В вариантной части ТП позволяет распределить одну и ту же область памяти компьютера между несколькими полями записи. Т.о. изменение одного поля сказывается на содержимом других. Объем распределенной памяти равен объему наибольшего поля.

**вариантная\_часть**::= "**case**" [ИД\_поля\_признака ":"]

тип\_поля\_признака "of"

вариант {";" вариант}

**тип\_поля\_признака**::= порядковый\_тип.

**Вариант**::=

константа {"," константа} ":" "(" [список\_полей] ")"

Например:

**…**

**type**

**Person=record**{определяем тип записи - "Человек"}

**Name:string[40];**{Имя (фамилия, отчество)}

**BirthDate: Date;** {дата рождения}

**case Citizen:Boolean of**{гражданство в нашей стране}

**true: (BirthPlace: string[40]);** {если гражданин, то определяем место рождения }

**false: (Country:string[20];** {иначе определяем страну, из которой прибыл …}

**EntryDate,ExitDate:Date);** {… даты въезда и выезда}

**end;**

Т.о. в зависимости от гражданства определено либо место рождения, либо страна, дата въезда и дата выезда.

К полям, указанным в вариантной части обращение производится точно так же, как и для фиксированной части. Для каждого варианта должен быть указан индивидуальный набор констант. Доступ к каждому варианту всегда разрешен вне зависимости от значения поля признака. Программа может использовать значение поля признака для определения, какой из вариантов является активным в данный момент. Если поле признака отсутствует, то определение должно происходить по другому критерию.

Для сокращения процесса обращения к полям записи служит оператор работы над записями.

**Оператор\_работы\_над\_записями**::=

"**with**" ссылка {"," ссылка} "do" оператор

**Ссылка**::=ИД переменной типа запись или объект.

Внутри оператора позволяется манипулировать полями записи, ссылки на которые имеются в заголовке, без указания имени переменной типа запись.

**Например:**

**var I: Date;**

**...**

**with I do begin**

**writeln("Год", Year);**

**writeln("Месяц", Month);**

**writeln("День", Day);**

**end;**

**...**

**Определение содержимого записи в разделе типированных констант.**

Объявление константы типа запись содержит идентификатор и значение каждого поля, заключенное в скобки и разделенные точками с запятой.

**Константа\_запись**::=

"(" ИД\_поля ":" типированная\_константа

{ ";" ИД\_поля ":" типированная\_константа } [";"] ")".

Например:

**const Man:person=**

**(Name: 'Иванов Иван Иванович';**

**BirthDate: ( Year: 1970;**

**Month: 2;**

**Day: 23);**

**Citizen: true;**

**BirthPlace: 'Шадринск');**

**Jerry:Person=**

**(Name: 'Мышонок Джерри';**

**BirthDate: ( Year: 1970;**

**Month: 2;**

**Day: 23);**

**Citizen: false;**

**Country: 'США';**

**EntryDate: ( Year: 1990;**

**Month: 1;**

**Day: 1);**

**ExitDate: ( Year: 2090;**

**Month: 1;**

**Day: 1));**

**Практика!**

***Задание 1.****Создать запись с информацией об автомобиле (не менее 5 полей). Заполнить с клавиатуры массив автомобилей. Затем ввести числовое значение и вывести на экран информацию об автомобилях с объемом двигателя, равным введенному значению.*

***Задание 2.****Создать запись с информацией о процессоре (не менее 3 полей). Заполнить с клавиатуры массив процессоров. Затем ввести строковое значение и вывести на экран информацию о процессорах, название фирмы-производителя которых соответствует введенному значению.*

***Задание 3.****Создать запись с информацией о студенте, включающую - ФИО, номер зачетки, номер группы. Создать программу, заполняющую с клавиатуры массив студентов и переводящую их на следующий курс. Получившийся результат вывести на экран (Например: если студент был в 181 группе, то после перевода на следующий курс номер его группы изменяется на 281. Если студент был в 504 группе, то в процессе перевода на следующий курс он исключается из массива).*

***Задание 4.****Создать запись с вариантной частью для хранения информации о таких геометрических фигурах, как точка, линия, окружность, треугольник, прямоугольник. Включить информацию о цвете фигуры. Создать программу, заполняющую с клавиатуры и выводящую на экран массив фигур.*

* Множества.

**Множество** в ТП является типом данных, диапазон значений которого является мощностью множества для определенного порядкового (базового) типа. Каждое возможное значение типа множества является подмножеством базового типа. Переменная типа множество может принимать как все значения множества, так и не одного. Множество не может содержит повторяющихся элементов. Размер любой переменной множественного типа составляет 32 байта.

**Тип\_множество**::= "**set**" "**of**" порядковый тип.

Базовый тип не должен иметь более 256 значений, нижняя и верхняя границы типа не должны выходить из промежутка от 0 до 255.

Например:

**type MySet = set of char;**

**var S:set of byte;**

Создание множества получается путем записи выражений, заключенных в []. Пустое множество описывается как [].

**Описатель\_множества**::= "[" [группа {"," группа}] "]"

**группа**::= выражение [".." выражение]

Значение выражения не должно выходить за пределы базового типа.

Например:

**var x:set of byte;**

**c: set of char;**

**k:byte;**

**begin**

**...**

**x:=[1,5,100..145, K div 10];**

**c:=[‘A’ .. ‘Z’, ‘a’ .. ’z’];**

**...**

**end.**

Над множествами можно проводить операции объединения, **разности** и **пересечения** по алгебраическим правилам действий над множествами, а также определять, содержит ли множество некоторый элемент, или является ли одно множество подмножеством (надмножеством) другого (см. параграф 1.4.3).

Например:

***Объединение*:**

Операция **[1,2,5,12]+[0,2,5,7,34]** возвратит множество **[0,1,2,5,7,12,34]** (состоящее из первого и второго множеств)

***Разность*:**

Операция **[1,2,5,12]-[0,2,5,7,34]** возвратит множество **[1,12]**(состоящее из тех элементов первого множества, которые отсутствуют во втором)

***Пересечение*:**

Операция **[1,2,5,12]\*[0,2,5,7,34]** возвратит множество **[2,5]**(состоящее из тех элементов первого множества, которые имеются и во втором)

***Вхождение во множество*:**

Операция **1 in [0,3,5]** возвратит **false**.

Операция **‘x’ in [‘A’, ‘k’..’z’]** возвратит **true**.

***Надмножество и подмножество:***

Операция **[1,2,5,12]>=[0,1,2,5,7,12,34]** возвратит **false**

Операция **[1,2,5,12]<=[0,1,2,5,7,12,34]** возвратит **true**

*Создать фильтр ввода ответа на вопрос (типа "Да"/"Нет")*

Решение:

**uses crt;**

**var c:char;**

**begin**

**writeln('Вы уверены?(Д/Н)');**

**repeat**{начало фильтра}

**c:=readkey;**

**until c in ['Д', 'д', 'Н', 'н'];**{фильтр пропускает только требуемый для ответа символ}

**writeln;**

**if c in ['Д','д'] then writeln(' На вопрос Вы ответили "ДА"')**

**else writeln('На вопрос Вы ответили "НЕТ"');**

**readln;**

**end.**

Для добавления и исключения элемента из множества в ТП предусмотрены две процедуры:

**procedure Include(var S: set of T; I:T);**

**procedure Exclude(var S: set of T; I:T);**

Процедура **Include** включает во множество **S** значение **I**.

Процедура **Exclude** исключает из множества **S** значение **I**.

В предыдущих версиях ТП и стандартном Паскале добавить элемент во множество можно было с помощью операции объединения, а удалить - с помощью операции разности множеств.

Например:

Если **S** - множество натуральных чисел, то операция **[3]+S**и функция **Include(s,3)**имеют один и тот же результат. Аналогичным образом можно оценить операцию **S-[7]**и функцию **Exclude(s,7)**. Однако функции **Include** и **Exclude** выполняются значительно быстрее.

***Практика!***

***Задание 5.****Удалить из строки все символы, входящие во множество цифр и знаков препинания.*

***Задание 6.****Оставить в строке только буквы (Подсказка: множество русских строчных букв разрывно. Поэтому вместо промежутка 'а'..'я' требуется использовать два промежутка - 'а'..'п' и 'р'..'я').*

***Задание 7.****Перевести строку в верхний регистр с учетом русского алфавита.*

* Файловые типы.

**Файловый тип** состоит из линейной последовательности компонент любого типа, кроме файлового или структурного типа, который содержит компоненту файлового типа.

**Файловый\_тип**::= "**text**" | ("**file**" "**of**" тип) | "**file**".

Файлы **TEXT**являются **текстовыми файлами последовательного доступа**. Их обработку можно вести только последовательно, строка, за строкой, слово за словом, символ за символом. Таким образом, если требуется считать строку из середины текстового файла, придется считывать все ей предшествующие. Однако этот недостаток с лихвой перекрывают возможности форматированного вывода и ввода информации произвольного типа из файла стандартными процедурами ввода-вывода. По сути дела, экран и клавиатура сами являются текстовыми файлами соответственно для записи и чтения. В модуле **System** содержатся 2 текстовые файловые переменные с именами **Output** и **Input**, первая из которых сопоставлена с экраном, а вторая - с клавиатурой.

Файлы **file of** **тип**являются **типированными файлами прямого доступа**. Это означает, во-первых, что компоненты файла имеют один и тот же тип, а о вторых, что доступ к содержимому файла ведется произвольно, в любую позицию. Таким образом, например, если нам требуется записать в середину файла вещественных чисел новое число, мы легко можем произвести установку файловой позиции в требуемое местоположение и произвести запись.

Осторожнее с файлами **file of string**. Такое определение подразумевает, что для каждой строки будет резервироваться 256 байт внутри файла. Это очень неэффективный подход с точки зрения экономии дискового пространства. Лучше в такой ситуации, хотя и в ущерб скорости, воспользоваться текстовыми файлами.

Файлы **file** являются **нетипированными файлами прямого доступа**. У нетипированных файлов отсутствует заранее определенная структура. Запись и чтение для таких файлов производится блоками, состоящими из последовательностей байтов. Размер блоков задается при открытии файла.

Типированные и нетипированные файлы отличаются от текстовых тем, что информация в них хранится в том же формате, что и в оперативной памяти компьютера. Это означает, с одной стороны, значительно меньший размер по сравнению с текстовыми, а с другой - невозможность, в большинстве случаев, редактирования содержимого таких файлов с помощью любых текстовых редакторов. К счастью, ТП имеет достаточно средств для создания специальных программ просмотра и редактирования нетекстовых файлов.

**Принципы работы с файлами**

Общий принцип обработки содержимого файлов состоит в следующей последовательности действий:

1. ***Сопоставление файловой переменной и конкретного файла на диске.***

2. ***Открытие файла.***

3. ***Чтение из файла или (и) запись в файл.***

4. ***Закрытие файла.***

Рассмотрим стандартные процедуры, позволяющие реализовать принципы работы с файлами.

|  |
| --- |
| *Сопоставление файлов* |
| procedure **Assign**(var f; name: string); |
| Процедура **Assign** сопоставляет файловую переменную **F** конкретному файлу на диске с именем **Name**. |
| *Открытие файлов* |
| procedure **Reset**(var f); |
| Процедура **Reset** открывает существующие текстовые файлы только для чтения, а типированные - для чтения и записи. **F** - файловая переменная, сопоставленная ранее с помощью процедуры **Assign** конкретному дисковому файлу. |
| procedure **Reset**(var f:file[; recsize: Word]); |
| Процедура **Reset** открывает существующий нетипированный файл **F** для чтения и записи. **F** - файловая переменная, сопоставленная ранее с помощью процедуры **Assign** конкретному дисковому файлу. **Recsize** - размер одной записи в файле. Если **Recsize** отсутствует, считается, что размер одной записи равен 128 байтам |
| procedure **Rewrite**(var f); |
| Процедура **Rewrite** создает текстовые файлы только для записи, а типированные - для чтения и записи. **F** - файловая переменная, сопоставленная ранее с помощью процедуры **Assign** конкретному дисковому файлу. Если файл с таким именем уже существовал на диске, он предварительно удаляется. |
| procedure **Rewrite**(var f:file[; recsize: Word]); |
| Процедура **Rewrite** создает нетипированный файл **F** для чтения и записи. **F** - файловая переменная, сопоставленная ранее с помощью процедуры **Assign** конкретному дисковому файлу. **Recsize** - размер одной записи в файле. Если **Recsize** отсутствует, считается, что размер одной записи равен 128 байтам. Если файл с таким именем уже существовал на диске, он предварительно удаляется. |
| procedure **Append** (var f:text); |
| Процедура **Append** открывает текстовый файл **F** для добавления в него информацией. **F** - файловая переменная, сопоставленная ранее с помощью процедуры **Assign** конкретному дисковому файлу. |
| *Чтение и запись файлов* |
| Для доступа к содержимому текстовых файлов используются процедуры **Read**, **Readln**, **Write**, **Writeln**, причем первым параметром передается имя файловой переменной. Методы обработки текстовых файлов с помощью этих процедур ничем не отличаются от методов обработки клавиатуры и экрана.  Для доступа к содержимому типированных файлов используются процедуры **Read** и **Write**, причем первым параметром также передается имя файловой переменной. Тип записываемых или считываемых переменных не должен отличаться от типа сохраненной в файле информации. В качестве параметров процедуры **Write** запрещено использовать выражения. |
| procedure **BlockRead**(var f: file; var Buf; Count: Word  [; var Result: Word]) |
| Процедура **BlockRead** читает в переменную **Buf** **Count** записей из файла **F**(размер каждой записи определен при открытии файла). В переменной **Result** возвращается число действительно прочитанных записей. |
| procedure **BlockWrite**(var f: file; var Buf; Count: Word  [; var Result: Word]) |
| Процедура **BlockWrite** Записывает в файл **F Count** записей из переменной **Buf**. В переменной **Result** возвращается число действительно сохраненных записей. Процедуры **BlockWrite**и **BlockRead** не могут обрабатывать более 64 килобайт информации за один раз. |
| *Закрытие файла* |
| procedure **Close**(var f); |
| Процедура **Close** закрывает файл**F**. Закрытие файлов требуется для освобождения всех файловых буферов в оперативной памяти, а также ресурсов, затребованных у операционной системы при открытии файла. По завершении работы программы все незакрытые файлы автоматически закрываются. |

*Написать две программы. С помощью первой из них создать и заполнить типированный файл десятью целыми числами. Вторую использовать для просмотра содержимого файла, сформированного первой программой.*

Решение:

{-----------первая программа------------}

**var f:file of integer;**

**i,v:integer;**

**s:string;**

**begin**

**write('Введите имя файла:'); readln(s);**

**Assign(f,s); Rewrite(f);**{сопоставляем файловую переменную и создаем текстовый файл}

**writeln('Эти значения будут сохранены в файле:');**

**for i:=1 to 10 do begin**{заполняем файл}

**v:=random(101)-50;**{получаем случайное значение из промежутка  
[-50,50]}

**write(f,v);**{сохраняем полученное число в файле ...}

**write(v:4);**{... а также выводим его на экран}

**end;**{файл заполнен}

**writeln;**

**close(f);**{закрываем файл}

**readln;**

**end.**

{-----------вторая программа------------}

**var f:file of integer;**

**i,v:integer;**

**s:string;**

**begin**

**write('Введите имя файла:'); readln(s);**

**Assign(f,s); Reset(f);** {сопоставляем файловую переменную и открываем текстовый файл для чтения}

**write('Содержимое файла ',s,': ');**

**for i:=1 to 10 do begin**{считываем содержимое файла}

**Read(f,v);**{считываем очередное число из файла}

**write(v:4);**{выводим полученное значение на экран}

**end;**{содержимое файла прочитано}

**writeln;**

**close(f);**{закрываем файл}

**readln;**

**end.**

*Написать две программы. С помощью первой из них создать и заполнить типированный файл (****file of byte****) размером в 100000 байт. С помощью второй создать аналогичного размера нетипированный файл. Сравнить скорости работы обеих программ.*

Решение:

{-----------первая программа------------}

**var f:file of byte;**

**i:longint;**

**b:byte;**

**s:string;**

**begin**

**write('Введите имя файла:'); readln(s);**

**Assign(f,s); Rewrite(f);**{сопоставляем и открываем типированный файл для записи}

**for i:=1 to 100000 do write(f,b);**{сохраняем 100000 байт в файле}

**close(f);**{закрываем файл}

**writeln('Файл сохранен');**

**readln;**

**end.**

{-----------вторая программа------------}

**var f:file;**

**b:array [1..50000] of byte;**

**s:string;**

**begin**

**write('Введите имя файла:'); readln(s);**

**Assign(f,s); Rewrite(f,1);** {сопоставляем и открываем нетипированный файл для записи, устанавливая размер одной записи в 1 байт}

**BlockWrite(f,b,50000);**{сохраняем первую половину требуемого содержимого файла}

**BlockWrite(f,b,50000);**{сохраняем вторую половину требуемого содержимого файла}

**close(f);**

**writeln('Файл сохранен');**

**readln;**

**end.**

Первая программа на компьютере с процессором Pentium-120 выполняется около 20 секунд, вторая - практически мгновенно. Выводы сделайте сами.

После выполнения программ вы наверняка заметите, что размер первого (типированного) файла постоянен и равен 200 байт (100 значений типа **Integer** умножить на 2 байта памяти для каждого числа), в то время как размер текстового файла изменяется от запуска к запуску второй программы и превышает размер первого файла по крайней мере в полтора раза.

* Файловая позиция.

##### Файловая позиция

Чтение или запись любого файла происходит последовательно, блок за блоком. Местоположение в файле, откуда при последующем обращении к файлу произойдет чтение или куда будет произведена запись, называется **файловой позицией**.

Для всех файлов, кроме текстовых, существует возможность принудительного перемещения файловой позиции в любую точку файла, а также чтение текущей файловой позиции.

Файловая позиция отсчитывается от 0.

|  |
| --- |
| *Получение файловой позиции* |
| function **FilePos**(var f): Longint; |
| Функция **FilePos** возвращает файловую позицию в файле **F**. |
| function **FileSize**(var f): Longint; |
| Функция **FileSize** возвращает последнюю возможную файловую позицию (ее можно рассчитать по формуле: размер файла в байтах/размер одной записи). Для файлов, размер одной записи которых составляет 1 байт, **FileSize** возвращает размер файла. В общем случае, **FileSize** возвращает количество компонентов файла. |
| *Установка файловой позиции* |
| procedure **Seek**(var F; N: Longint); |
| Процедура **Seek**устанавливает файловую позицию файла **F** в **N**. |
| *Операции над файлом, зависящие от текущей файловой позиции* |
| function **Eof**(var F): Boolean; |
| Возвращает **True**, если файловая позиция находится за концом типированного или нетипированного файла. |
| function **Eof** [ (var F: Text) ]: Boolean; |
| Возвращает **True**, если файловая позиция находится за концом текстового файла. Если файловая переменная отсутствует, то используется переменная **Input** (клавиатура). |
| function **Eoln** [(var F: Text) ]: Boolean; |
| Возвращает **True**, если файловая позиция находится за концом строки текстового файла. Если файловая переменная отсутствует, то используется переменная **Input** (клавиатура). |
| procedure **Truncate**(var F); |
| Отсекает конец файла, начиная с текущей файловой позиции. |

*Ввести набор из N чисел, сохранить его в файле. Организовать цикл, в котором по введенному индексу определяется и выводится на экран числовое значение из файла. Предусмотреть выход их цикла.*

**var f:file of real;**

**v:real;**

**i,n:integer;**

**begin**

**write('Введите количество чисел: '); readlnno;**

**assign(f,'Real.fil');**

**Rewrite(f);**

**for i:=1 to n do begin**{заполняем файл вещественными числами}

**v:=random\*1000-500;**

**write(f,v);**

**end;**

**close(f);**{файл заполнен, закрываем его}

**reset(f);**{снова открываем файл (закрытие и повторное открытие файла использовано здесь только для того, чтобы показать возможность повторного открытия файла без повторного использования процедуры **Assign**)}

**write('Введите номер: '); readln(N);**{ввели индекс}

**while (n>=0) and (n<=FileSize(f)-1) do begin**{пока введенный индекс не вышел за пределы файла}

**seek(f,n);**{устанавливаем файловую позицию}

**Read(f,v);**{читаем число из файла}

**writeln(v);**{выводим его на экран}

**write('Введите номер: '); readln(N);**{снова вводим индекс}

**end;**

{вышли из цикла, так как введен недопустимый индекс}

**Close(f);**{закрываем файл}

**readln;**

**end.**

*Создать программу, которая выводит на экран саму себя.*

Решение данной задачи становится тривиальным, если воспользоваться файлом текста программы.

**var f:text; s:string; i:byte;**

**begin**

**s:=paramstr(0);**{получаем имя исполняемого файла}

**s:=copy(s,1,length(s)-3)+'pas';**{заменяем расширение "**exe**" исполняемого файла на расширение "**pas**"}

**assign(f,s);**

**Reset(f);**

**while not eof(f) do begin**{пока не конец файла ...}

**readln(f,s);**{считываем строку}

**writeln(s);**{выводим ее на экран}

**end;**

**close(f);** {обработка завершена, закрываем файл}

**readln;**

**end.**

*Каждая строка (до 1000) текстового файла содержит в себе набор целых чисел (до 100), разделенных пробелами. Вывести на экран содержимое файла с использованием форматирования, а так же подсчитать среднее арифметическое каждой строки файла.*

Решение:

**var f:text;**{текстовый файл}

**Ac: array[1..100] of integer;**{массив чисел текущей строки}

**Ar: array[1..1000] of real;**{массив средних арифметических строк}

**Row, Col : integer;**{количество строк в файле и количество чисел в текущей строке}

**i:integer;**

**begin**

**assign(f,'numbers.txt');**{сопоставление}

**reset(f);**{открытие для чтения}

**writeln('Содержимое файла NUMBERS.TXT:');**

**Row:=0;**

**while not eof(f) do begin**{пока не конец файла ...}

**Col:=0;**

**while not eoln(f) do begin**{пока не конец строки файла ...}

**inc(Col);**{переходим к следующему числу строки}

**read(f, ac[Col]);**{считываем очередное числовое значение из файла в массив **AC**}

**write(ac[Col]:6);**{производим форматированный вывод}

**end;**

{информация из строки считана}

**readln(f);**{считываем маркер конца строки из файла}

**writeln;**{разделяем выводимые на экран строки}

**inc(Row);**{переходим к следующей строке массива **AR** для подсчета среднего арифметического}

**ar[row]:=0;**

**for i:=1 to Col do ar[row]:=ar[row]+ac[i];**{подсчитываем сумму элементов строк}

**ar[row]:=ar[row] / col;**{рассчитываем среднее арифметическое}

**end;**

{обработка файла закончена}

**Close(f);**

**writeln('Средние арифметические строк:');**

**for i:=1 to Row do write(ar[i]:7:2);**{выводим на экран средние арифметические строк файла}

**writeln;**

**readln;**

**end.**

Если, например, файл **number.txt** имеет следующее содержимое:

**1 34 -2 35 43 1 2 -2 -2 7 -8**

**23 3 45 -3 2**

**2 12 -21 2 43 -43 5 6 7**

**2 3**

**7 89 7**

то в результате работы программы на экране появится:

**Содержимое файла NUMBERS.TXT:**

**1 34 -2 35 43 1 2 -2 -2 7 -8**

**23 3 45 -3 2**

**2 12 -21 2 43 -43 5 6 7**

**2 3**

**7 89 7**

**Средние арифметические строк:**

**9.91 14.00 1.44 2.50 34.33**

**Дополнительные процедуры и функции для работы с файлами**

|  |
| --- |
| *Обработка ошибок* |
| function **IOResult**: Integer; |
| Во время работы с файлами могут возникать различные ошибки. Если установлена директива компилятора **{$I+}**, то работа программы при этом аварийно завершается. Если установлена директива компилятора **{$I-}**, то с помощью функции **IOResult** можно определить номер последней ошибки ввода-вывода. **IOResult** возвращает 0, если ошибки не произошло. Если произошла ошибка, то до вызова **IOResult** все операции с файлами будут заблокированы |
| *Обработка файлов, не связанная с их содержимым* |
| procedure **Rename**(var F; Newname:string); |
| Переименовывает файл, сопоставленный с файловой переменной **F** в **Newname**. Файл не должен быть открыт. |
| procedure **Erase**(var F); |
| Удаляет файл, сопоставленный с файловой переменной **F**. Файл не должен быть открыт. |
| *Обработка каталогов* |
| procedure **ChDir**(S: String); |
| Переходит в каталог с именем **S**. |
| procedure **MkDir**(S: String); |
| Создает каталог с именем **S**. |
| procedure **RmDir**(S: String); |
| Удаляет каталог с именем **S**. |

*Дано имя файла. Проверить, существует ли файл с таким именем он на диске.*

**var f:file;**

**s:string;**

**begin**

**write('Введите имя файла:'); readln(s);**

**{$I-}**{отключаем проверку ошибок ввода-вывода}

**assign(f,s);**

**Reset(f);**{пытаемся открыть файл}

**{$I+}**{включаем проверку ошибок ввода-вывода}

**if ioresult<>0 then**{попытка открытия файла окончилась неудачно}

**writeln ('Такого файла на диске нет')**

**else begin**{файл открыт успешно ...}

**writeln('Файл существует');**

**close(f);**{... не забудем его закрыть}

**end;**

**readln**

**end.**

Практика!

***Задание 8.****Создать программу, в которой вводятся с клавиатуры произвольные числа, пока не будет введен 0, и сохраняются в текстовом файле только те из них, которые кратны 7.*

***Задание 9.****Дан текстовый файл, заполненный целыми числами (создать в любом текстовом редакторе). Вывести на экран только те из них, порядковый номер которых равен значению числа.*

***Задание 10.****Дан текстовый файл, заполненный целыми числами и одномерный массив из 5 целых чисел. Вывести на экран в только те числа из файла, которые равны одному из чисел массива.*

***Задание 11.****Дан текстовый файл, заполненный целыми числами (создать в любом текстовом редакторе). Вывести их на экран в обратном порядке.*

***Задание 12.****Создать и заполнить одними и теми же строками 2 файла. Первый - текстовый файл, второй - типированный файл строк. В операционной системе сравнить размеры и содержимое обоих файлов. Сделать выводы.*

***Задание 13.****Дан файл, компоненты которого являются целыми числами. Найти сумму всех компонент файла.*

***Задание 14.****Создать две программы. Первая из них заполняет файл геометрическими фигурами. Вторая программа по введенному номеру считывает из файла запись и выводит ее содержимое на экран.*

***Задание 15.****Создать программу копирования одного файла в другой. Оптимизировать по скорости выполнения.*