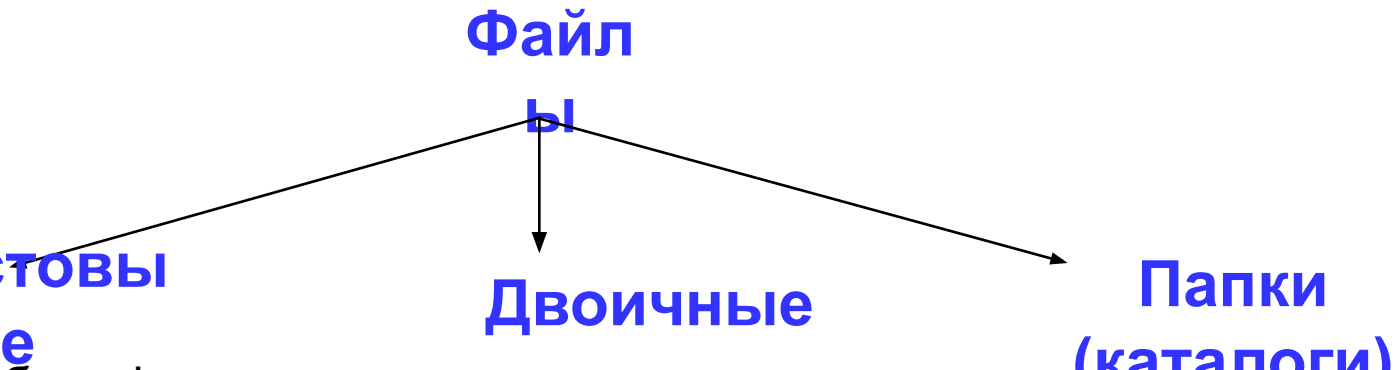
Программирование на языке Паскаль

Типизированные файлы

Файлы

Файл – это область на диске, имеющая имя.



только текст без оформления, не содержат управляющих символов (с кодами < 32)

ACSII (1 байт на символ)

UNICODE (2 байта на символ)

`*.txt, *.log,`
`*.htm, *.html`

могут содержать любые символы кодовой таблицы

`*.doc, *.exe,`
`*.bmp, *.jpg,`
`*.wav, *.mp3,`
`*.avi, *.mpg`

Разновидности файлов

Бинарные файлы бывают двух видов

- **типизированные,**
- **нетипизированные.**

К *типизированным* относятся файлы, содержащие данные строго определенного типа. Обычно такие файлы представляются собой наборы записей.

К *нетипизированным* относятся двоичные файлы, которые могут содержать любые совокупности байтов данных без привязки к какому-нибудь одному типу.

Организация файла

Доступ к компоненту файла осуществляется через *указатель файла.*

При выполнении операции чтения или записи указатель автоматически перемещается на следующий компонент:

Описание типизированных файлов

Файловая переменная типизированного файла описывается как:

```
Type <идентификатор файловой переменной> =
     file of <тип компонента>;
```

где `<тип компонента>` **- любой тип данных, кроме файлового.**

Типизированные файлы используют, когда обрабатывают хранящуюся в файле последовательность компонентов одинаковой длины (чисел, записей и т.п.).

Описание файлов

Как и любая переменная языка Паскаль, файловая переменная может быть описана в инструкции объявления переменных.

Например:

```
Var  F1: file of real;
   F2: file of char;
   F3: file of integer;
```

или с предварительным объявлением типа:

```
Type FF = file of integer;
Var  F1: FF;
```

Инициализация файловой переменной

Связь между физическим файлом и файловой переменной устанавливается специальной процедурой.

Процедура

```
Assign (f, st:string)
```

инициализирует файловую переменную *f*, связывая ее с файлом или логическим устройством, определенным строкой *st*.

Если файл находится в текущем каталоге, то достаточно указать имя файла и его расширение.

В противном случае необходимо указать полное имя файла

Инициализация файловой переменной

Например:

```
Type  F = file of real;
Var f1, f2, f3: F;
...
Assign (f1, 'T1.dat');          {связывание файловой
   переменной с файлом в текущем каталоге}
Assign (f2, 'd:\iva\a.dat'): {связывание
файловой переменной с файлом в указанном      каталоге}
```

Открытие файла

Процедура

```
Reset(f);
```

открывает файл, определенный файловой переменной *f* для чтения.

При выполнении этой процедуры указатель файла устанавливается на первый компонент файла.

Открытие файла

При открытии для чтения несуществующего файла регистрируется ошибка выполнения, а функция *IOResult* типа *Word* возвращает значение, отличное от 0 (см. далее описание функции).

Отключив контроль операций ввода-вывода и используя функцию *IOResult*, можно организовать проверку наличия файла с указанным именем на диске:

```
Var f: file of char;
Begin
  Assign(f, 'a.dat'); {инициализация файловой переменной}
  {$ I- }  {отмена контроля ошибок ввода-вывода}
  ReSet (f); {открытие файла для чтения}
  {$ I+ }  {включение контроля ошибок}
  if IOResult <>0 then
     WriteLn ('Файл не существует');
  else     WriteLn('Файл существует');
```

Открытие файла

Процедура

```
Rewrite(f)
```

открывает файл, определенный файловой переменной *f*, для записи.

- **При открытии для записи существующего файла старый файл *уничтожается без предварительной проверки и выдачи предупреждения пользователю.***
- **Если файла с таким именем не существовало, то он создается и подготавливается к записи (физически – очищается буфер).**

Закрытие файла

Закрытие файла, открытого для записи или чтения, осуществляется процедурой

```
Close(f)
```

При этом вновь созданный файл регистрируется в каталоге.

Поскольку любое обращение к диску осуществляется через буферную память, часть данных, выводимых в файл, может остаться в буфере. Процедура закрытия файла обеспечивает вывод оставшихся компонентов из буфера в файл.

Связь файловой переменной с файлом при закрытии сохраняется, и при повторном использовании этого же файла процедуру *Assign* применять еще раз не требуется.

Стандартные процедуры и функции обслуживания типизированных файлов

Процедура

```
Read(f, c1, c2, ...,cn)
```

осуществляет чтение очередных компонентов типизированного файла.

Список переменных ввода содержит одну или несколько переменных того же типа, что и компоненты файла, разделенных запятыми.

Если файл исчерпан, обращение к процедуре вызывает ошибку ввода-вывода.

Стандартные процедуры и функции обслуживания типизированных файлов

Процедура

```
Write(f, c1, c2, ...,cn)
```

осуществляет запись данных в типизированный файл.

Список вывода содержит одно или более выражений того же типа, что и компоненты файла, разделенных запятыми.

При работе с типизированными файлами процедура *Writeln* не используется.

Стандартные процедуры и функции обслуживания типизированных файлов

Процедура

```
Seek(f, numcomp: word)
```

осуществляет установку указателя файла (переход) на компонент файла с номером *numcomp*.

Например,

```
Seek(f, 9);
```

осуществляет переход к десятой записи в файле *f*.

Стандартные процедуры и функции обслуживания типизированных файлов

Функция

```
FileSize(f): longint
```

возвращает количество компонент файла, указанного файловой переменной.

Может использоваться для установки на конец файла совместно с *Seek()* или на последнюю запись файла соответственно:

```
Seek(f, FileSize(f));
Seek(f, FileSize(f)-1);
```

Функция

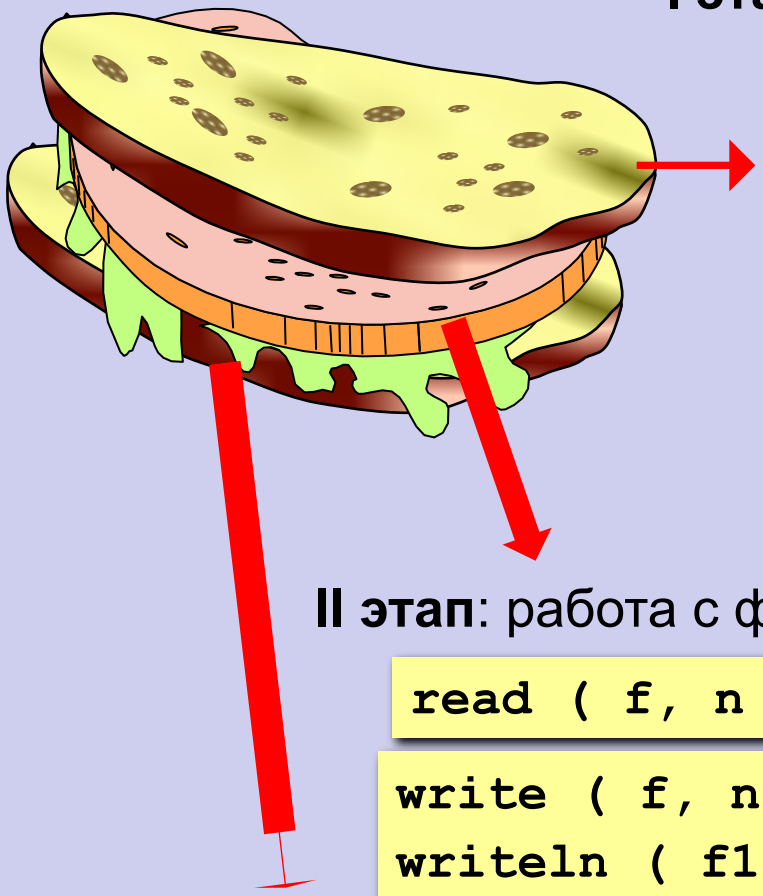
```
FilePos(f): longint
```

возвращает порядковый номер компонента, который будет обрабатываться следующей операцией ввода-вывода.

Принцип сэндвича !!!

Переменная типа «текстовый файл» или типизированный файл:

```
var f: file of <тип>;
    f1:text;
```



I этап. открыть файл :

- связать переменную **f** с файлом

```
assign(f, 'qq.txt');
```

- открыть файл (сделать его активным, приготовить к работе)

```
reset(f); {для чтения}
```

```
rewrite(f); {для записи}
```

```
append(f1); {для дозаписи}
```

II этап: работа с файлом

```
read ( f, n );  {прочитать значение в n }
```

```
write ( f, n );  { записать значение n в файл}
writeln ( f1, n );{с переходом на нов.строку }
```

III этап: закрыть файл

```
close(f);
```

Обработка типизированного файла

После *открытия файла* для чтения или записи указатель файла стоит в его начале и указывает на первый компонент, имеющий номер 0.

После каждого *чтения или записи* указатель сдвигается к следующему компоненту файла.

Поскольку длина каждой компоненты файла строго постоянна, помимо последовательного возможно осуществление *прямого доступа* к компонентам файла.

Удаление компонент обычно требует перезаписи файла.

Обработка типизированного файла

Добавление **компонентов** ***в конец файла*** **выполняется в режиме записи. Для этого указатель файла устанавливается на его конец (как показано выше), после чего все выводимые компоненты дописываются в конец файла.**

Добавление **компонентов** ***в середину или начало файла*** **может выполняться следующим образом:**

- **определяем место, в которое должны быть добавлены элементы,**
- **все последующие компоненты переписываем во временный файл,**
- **вставляем новые компоненты**
- **дописываем в файл компоненты, переписанные во временный файл.**

Работа с файлами

Особенности:

- имя файла упоминается только в команде **`assign`**, обращение к файлу идет через файловую переменную
- файл, который открывается на чтение, должен **существовать**
- если файл, который открывается на запись, существует, старое содержимое **уничтожается,** иначе создается **новый**
- при завершении программы все файлы закрываются автоматически
- после закрытия файла переменную **`f`** можно использовать еще раз для работы с другим файлом

Стандартные процедуры и функции обслуживания файлов

Функция

```
EOF(f):boolean
```

определяет конец файла.

Как было отмечено выше, размер файла при его создании не фиксируется. Поэтому в процессе работы требуется проверка достижения конца файла.

Функция принимает значение *TRUE*, если указатель стоит в конце файла (после последней записи). При этом, если производится чтение, то это означает, что файл исчерпан, а если идет запись, то новая запись дописывается в конец файла.

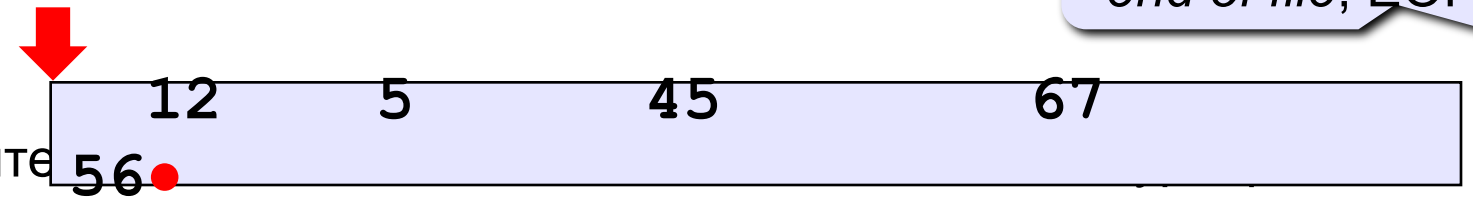
Функция принимает значение *FALSE*, если конец файла еще не достигнут.

Последовательный доступ

- при открытии файла курсор устанавливается в начало

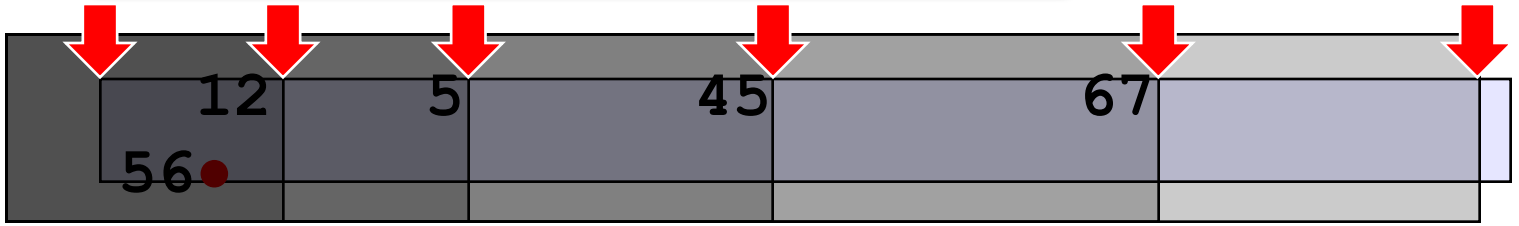
```
assign ( f, 'qq.dat' );
reset ( f );
```

конец файла
end of file, EOF(f)



- чте
- после чтения курсор сдвигается на первый непрочитанный символ

```
read ( f, x );
```



- как вернуться назад?

```
close ( f );
reset ( f ); { начать с начала }
```

Пример

Задача: в файле `input.dat` записаны числа, сколько их – неизвестно. Записать в файл `output.dat` их сумму.

Можно ли обойтись без массива?

Алгоритм:

1. Открыть файл **`input.dat`** для чтения.
2. **`S := 0;`**
3. Если чисел не осталось, перейти к шагу 7.
4. Прочитать очередное число в переменную **`x`**.
5. **`S := S + x;`**
6. Перейти к шагу 3.
7. Закрыть файл **`input.dat`**.
8. Открыть файл **`output.dat`** для записи.
9. Записать в файл значение **`S`**.
10. Закрыть файл **`output.dat`**.

цикл с условием **«пока есть данные»**

Программа

```
program qq;
var s, x: integer;
    f: file of
    integer;
begin
    assign(f, 'input.dat');
    reset(f);
    s := 0;
    while not eof(f
          ) do begin
      read(f, x);
      s := s + x;
    end;
    close(f);
    assign(f, 'output.dat');
    rewrite(f);
    write(f, s);
    close(f);
end.
```

логическая функция, возвращает **True**, если достигнут конец файла

запись результата в файл **output.dat**

Задания

В файле `data.dat` записаны числа, сколько их – неизвестно.

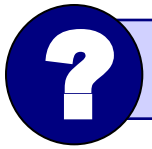
«3»: Найти сумму чётных чисел и записать её в файл **`output.dat`**.

«4»: Найти минимальное и максимальное из четных чисел и записать их в файл **`output.dat`**.

«5»: Найти длину самой длинной цепочки одинаковых чисел, идущих подряд, и записать её в файл **`output.dat`**.

Обработка массивов

Задача: в файле `input.dat` записаны числа, сколько их – неизвестно, но не более 100. Переставить их в порядке возрастания и записать в файл `output.dat`.



Можно ли обойтись без массива?

Проблемы:

1. для сортировки надо удерживать в памяти все числа сразу (массив);
2. сколько чисел – неизвестно.

Решение:

3. выделяем в памяти массив из 100 элементов;
4. записываем прочитанные числа в массив и считаем их в переменной `N`;
5. сортируем первые `N` элементов массива;
6. записываем их в файл.

Чтение данных в массив

Глобальные переменные:

```
var A: array[1..100] of integer;
    f: file of integer;
```

Функция: ввод массива, возвращает число элементов

```
function ReadArray(var m: array[1..100] of integer):integer;
var i: integer;
begin
  assign(f, 'input.dat');
  reset(f);
  i := 0;
  while (not eof(f)) and (i < 100) do begin
    i := i + 1;
    read(f, M[i]);
  end;
  close(f);
  ReadArray :=
        i;
end;
```

цикл заканчивается, если достигнут конец файла или прочитали 100 чисел

Программа

```
program qq;
var A: array[1..100] of integer;
    f: file of integer;
    N, i: integer;
function ReadArray(var m:array[1..100] of
integer):integer;
...
end;

   N := ReadArray(A);
   { сортировка первых N элементов }

      assign(f, 'output.dat');
      rewrite(f);
      for i:=1 to N do
        write(f, A[i]);
      close(f);
end.
```

запись отсортированного массива в файл

Задания

В файле `input.dat` записаны числа, известно, что их не более 100.

«3»: Отсортировать массив по убыванию и записать его в файл **`output.dat`**.

«4»: Отсортировать массив по убыванию последней цифры и записать его в файл **`output.dat`**.

«5»: Отсортировать массив по возрастанию суммы цифр и записать его в файл **`output.dat`**.

Пример

Разработать программу, создающую файл, компонентами которого являются символы, введенные с клавиатуры. Затем эта программа должна:

- ✔ **организовывать чтение символов из файла**
- ✔ **находить указанный символ в файле и удалять его из файла.**

Пример

```
Program ex;
Var f, f1: file of char;     { две файловые переменные}
    ch, i: char;
    name: string;
Begin
  WriteLn('Введите имя файла:');
  ReadLn(name);

   {создание и открытие файла}
  Assign(f, name + '.dat'); {связываем файл
                  с файловой переменной}
  ReWrite(f);     {открываем файл для записи (создаем)}
  WriteLn('Вводите символы или `*`:');
```

Пример

```
 {занесение записей в файл}
while ch<>'*' do {пока не введено символ '*'}
begin
 ReadLn(ch);   {вводим символ с клавиатуры}
 Write(f, ch);    {записываем символ в файл}
end;
Close(f);    {закрываем файл}
WriteLn;
 {последовательное чтение записей из файла}
Reset(f);       {открываем файл для чтения}
while not EOF(f) do {пока не достигнут конец файла}
begin
 Read(f, ch);  {читаем символ из файла}
 Write(ch, ' ');  {выводим символ на экран}
end;
WriteLn;
```

Пример

```
WriteLn ('Введите символ для удаления:');
ReadLn(ch);
    {подготовка к удалению записей: переименование исходного
   файла и открытие нового файла с тем же именем}
Close(f);   {закрываем файл}
ReName(f, name + '.bak');    {переименовываем файл}
ReSet(f);       {открываем файл для чтения}
Assign(f1, name + '.dat');  {связываем новый файл с
             переменной}
ReWrite(f1);       {открываем новый файл для записи}
    {удаление записей - перепись остающихся записей в др. файл}
while not EOF(f) do begin
  Read(f, i);       {читаем символ из файла}
  if i<>ch then Write(f1, i); {если символ
     не подлежит удалению, то записываем его в новый файл}
end;
Erase(f); {удаляем старый файл, после закрытия в нем ничего
не изменилось, поэтому повторно его можно не закрывать}
```

Пример

```
   {последовательное чтение записей из нового файла}
  ReSet(f1);  {открываем новый файл для чтения}
  while not EOF(f1) do  begin
   Read(f1, Ch);    {читаем из файла}
   Write(ch, ' ');
  end;
  WriteLn;
End.
```