Глава IX. Файловые средства MSX BASIC

Файл (от англ. file — досье, картотека), набор данных,—
1) совокупность упорядоченных и взаимосвязанных записей, имеющая описание для идентификации отдельных записей;
2) последовательность записей, размещаемая на внешних запоминающих устройствах (внешней памяти) и рассматриваемая в процессе пересылки и обработки как единое целое.

—Математический Энциклопедический Словарь

В персональных компьютерах Ямаха КУВТ предусмотрена работа с двумя версиями языка BASIC: MSX BASIC и MSX Disk BASIC. Версия языка MSX BASIC предусматривает работу только с накопителем на магнитной ленте, а в версии MSX Disk BASIC возможно работать как с накопителем на магнитной ленте, так и с магнитными дисками.

Выбор версии языка BASIC, с которой будет работать персональный компьютер Ямаха КУВТ, осуществляется автоматически и зависит от состава подключённых внешних устройств хранения информации.

Версия языка MSX BASIC «прошита» (записана, хранится) непосредственно в ПЗУ компьютера, а версия языка MSX Disk BASIC записана и хранится в ПЗУ интерфейса (устройства сопряжения и согласования) дискового накопителя — *дисковода*. В зависимости от того, подключён ли интерфейс (вместе с дисководом) к компьютеру или нет, и выбирается версия языка BASIC, с которой мы будем работать. Если интерфейс подключён, то при включении компьютера «работает» интерпретатор языка MSX Disk BASIC, а если нет, то «включается» интерпретатор языка MSX BASIC.

Практически для учительского компьютера это делается следующим образом: нажмите кнопку Reset и в момент начальной инициализации компьютера держите нажатой клавишу SHIFT !

Будем предполагать, что Ваш компьютер имеет встроенный дисковод. В таком варианте при включении запускается MSX Disk BASIC, а приоритетным устройством внешней памяти является дисковод A:.

Работая на компьютере без дисковода, Вы запускаете при включении MSX BASIC, а приоритетным устройством является кассетный магнитофон (CAS:), имя которого может опускаться.

Замечание.
В СССР распространены две версии языка MSX BASIC:

- версия 1.0 — на компьютерах серии MSX 1:
  - Yamaha YIS-503IIR
- версия 1.1 — на компьютерах серии MSX 2:
  - Yamaha YIS-503IIIR
  - Yamaha YIS-805-128R2

Для определения номера версии в Вашей программе проверьте содержимое ячейки ПЗУ с адресом &H002D:

```
? PEEK(&H2D)
```

Номер версии	Содержимое ячейки
1.0	0
1.1	1

IX.1. Работа с файлами на дискетах

Под словом *«данные»* мы будем подразумевать содержимое памяти компьютера. Принято различать текст программы, исходные данные для работы программы и результаты работы программы.

Работа с внешней памятью подразумевает запись и чтение данных, размещаемых на внешних носителях информации — кассете магнитной ленты или гибком магнитном диске (*дискете*).

IX.1.1. Форматирование дискеты

Дискеты бывают следующих типов:

1. односторонние одинарной плотности (SS, SD);
2. односторонние двойной плотности (SS, DD);
3. двусторонние двойной плотности (DS, DD).

Односторонность и двусторонность свидетельствует о том, имеет ли дисковод одну или две магнитные головки, которые обеспечивают запись и считывание информации с одной или двух сторон гибкого диска. Лучший способ объяснить понятие «плотность записи» — провести аналогию с грампластинкой: считать, например, что на диске двойной плотности в два раза больше «дорожек», чем на диске одинарной плотности. И хотя память на гибком диске является магнитной и никаких, естественно, дорожек нет, на гибком диске четверной плотности может храниться в четыре раза больше информации, чем на диске одинарной плотности.

Для работы с 3,5–дюймовым (8,89 см) дисководом MSX–компьютера используются односторонние дискеты MF1–DD, двусторонние дискеты MF2–DD или их аналоги (слова: «диск», «дискета» и «флоппи–диск» мы будем рассматривать как синонимы).

Дискета заключена в твёрдый пластмассовый конверт и имеет переключатель защиты записи и скользящую шторку для прикрытия окна, через которое организуется чтение (запись) информации.

Вставлять дискету в карман (щель) дисковода и извлекать её оттуда можно лишь при *включённом* дисководе!

Приведём некоторые характеристики дискеты MF2–DD :

Ёмкость при записи	
всего	720 Кбайт
для файлов пользователя	713 Кбайт
Плотность при записи	8717 бит/дюйм
Среднее время доступа	95 мс
Максимальное количество файлов	112

Разрешается подключать к компьютеру как одиночные, так и спаренные дисководы. В первом случае за устройством закрепляется переменное имя, которое в любой момент времени принимает одно из конкретных значений «A» или «B». Во втором случае основной дисковод получает имя «A», а дополнительный — «B».

Везде ниже, если это не оговорено специально, изложение ориентировано на использование одного дисковода.

Для подготовки диска к работе с командами и функциями MSX Disk BASIC необходимо осуществить специальную разметку его поверхности, называемую *форматированием*. Для этого дискета вставляется в карман дисковода и выполняется команда:

```
CALL FORMAT
```

или

```
_FORMAT
```

,

которая вызывает на экране индикацию запроса:

```
Drive name? (A,B) ▉
```

Ответом на него должен быть ввод одной из латинских букв A или B, в зависимости от имени устройства, в котором находится дискета.

На появившийся второй запрос–«подсказку»:

```
1 - Double sided
2 - Single sided
? 
```

пользователь обязан отреагировать вводом 1, если диск — двусторонний, или 2, если диск — односторонний.

Далее, по указанию:

```
Strike a key when ready 
```

нажмите любую клавишу. После этого начинается процесс форматирования, то есть размещение на диске управляющих меток. При этом ранее имевшаяся там информация разрушается. Правильное завершение этого процесса вызывает индикацию сообщения

```
Format complete
Ok
```

Форматирование *новых* дискет обязательно!

Учтите, что именно проведённая разметка дискеты, а не её потенциальные возможности определяет объем информации, который она может хранить, и с каким дисководом её можно использовать!

В заключении дадим несколько *полезных советов*:

1. В дискетах не предусмотрена идеальная защита записанных на них данных: защитные конверты рассчитаны только на предохранение поверхности носителей от повреждений на коротком пути от «дискотеки» до дисковода.
   Особую осторожность надо соблюдать при нанесении надписей на наклейке, прикреплённой к дискете, так как даже давление карандаша или шариковой ручки может оказаться достаточным, чтобы через защитный конверт повредить магнитный слой дискеты. Во избежании порчи информации, записанной на дискетах, последние следует хранить подальше от телевизоров, видео–мониторов, телефонов и других источников магнитных полей;
2. Для защиты файлов, сохранённых на дискете, позаботьтесь о создании копий файлов на другой дискете или на магнитной ленте;
3. Создайте архив файлов программных текстов и снабжайте дискеты и магнитные ленты архива этикетками;
4. Защищайте Ваши программы от неосторожного обращения путём установки переключателя защиты от записи на дискете.

IX.1.2. Имена файлов

При создании на дискете нового файла ему необходимо присвоить определённое *имя*. Оно должно быть уникальным, то есть не совпадать ни с одним из уже имеющихся имён файлов, сформированных на данной дискете ранее.

Имя служит для идентификации файлов и строится из последовательности символов алфавита MSX BASIC. При этом должны учитываться следующие обстоятельства:

1. Количество символов имени не может превосходить 11. Попытка сформировать имя большей длины ни к чему не приводит. Правые лишние символы отбрасываются;
2. Соответствующие прописные и строчные латинские буквы, используемые в имени, равнозначны; соответствующие прописные и строчные буквы русского алфавита, используемые в имени, *неравнозначны*;
3. В имени не должно быть символов:

   ```
   ; , + " = [ ] * \ / ? пробел
   ```

4. Символ «:» используется только при задании составных имён для разделения их на две части, первая из

которых есть имя устройства, а вторая — имя файла. В этом случае при записи файла на диск фиксируется лишь вторая часть имени;
5. Нельзя задавать имя, состоящее из одних пробелов или начинающееся с пробела;
6. Имя не должно начинаться с точки («.»). В нем не может быть более одного такого символа и более 8 символов перед ним (если их больше восьми, то после восьмого символа устанавливается «.»).

Кроме того, десятичной точкой имя можно разделить на две зоны: *корень* (до точки) и *расширение* (после точки). Расширение состоит не более, чем из трёх символов. Номер начальной позиции корня — 1, а расширения — 10.

Если пользователь фиксирует имя без расширения с количеством символов, большим 8, то компьютер автоматически разбивает его на корень и расширение, вставляя в требуемое место точку. Сообщение об ошибке

«Bad file name» («*Недопустимое имя файла*»)

появляется в том случае, когда имя файла содержит более *восьми* символов перед явно указанным расширением; если же расширение состоит более, чем из *трёх* символов, то интерпретатор игнорирует лишние символы.

Примеры.

Имя, заданное пользователем	**Представление имени в компьютере**
КАТАЛОГ	КАТАЛОГ
LESSON.123	LESSON .123
BASIC	BASIC
ВАЛ.процент	ВАЛ .про
ПРОТИВОГАЗ	ПРОТИВОГ.АЗ

Общепринято, что расширение имени файла должно обозначать его тип. Обычно применяются следующие стандартные расширения имён:

Расширение	**Комментарий**
.ASM	Исходная программа на языке ассемблера
.BAK	Резервный файл или копия некоторого файла, сделанная на случай повреждения оригинала
.BAS	Программа на языке BASIC
.BAT	Командный файл для пакетной обработки
.C	Исходная программа на языке C
.COM	Команда или программа, пригодные для непосредственного исполнения под управлением DOS
.DAT	Файл данных
.DOC	Файл документов (для текстовой обработки)
.FOR	Исходная программа на языке Фортран
.LIB	Библиотека программ
.OBJ	Скомпилированная объектная программа на машинном языке
.PAS	Исходная программа на языке Паскаль
.PIC	Данные выводимого на экран изображения
.TMP	Временный файл
.TXT	Текстовый файл

Маленькая хитрость. Оградить Ваш файл от посторонних взглядов можно, дав ему секретное имя АААААААА.ААА , например. Секрет здесь в том, что в этом имени часть букв А из русского, а часть — из латинского алфавита, и об этом знает только хозяин файла.

В большинстве случаев желательно, чтобы присвоенные файлам имена были индивидуальными и единственными и точно указывали, какой именно файл имеется в виду. Но иногда удобнее обратиться сразу ко всей группе файлов, а не работать с ними по одному.

Для указания *родовых* имён файлов могут использоваться специально предназначенные для этого знаки ? и *. *Вопросительный* знак является единственным неоднозначно интерпретированным символом в имени файла. Например, имена

- MAX1.ASM
- MAX2.ASM
- MAX3.ASM

все соответствуют родовому имени файла MAX?.ASM, но ему не соответствует имя MAX10.ASM.

Звёздочка обозначает любое количество неоднозначно интерпретируемых символов. Например, Zilog*.A* будет обозначать любое имя файла, которое начинается с Zilog, при условии, что его расширение начинается с A.
Однако звёздочка имеет смысл только при использовании её в качестве последнего символа имени файла или расширения. Например, родовое имя файла *CALC.BAS — это то же самое, что и имя *.BAS, которое соответствует каждому файлу, имеющему расширение имени *.BAS.

Отметим, что *.* — родовое имя файлов с любыми именами.

AUTOEXEC.BAS

Имя файла `AUTOEXEC.BAS` происходит от Automatic Execution («автоматическое исполнение»). Файл с таким именем будет автоматически загружен и запущен.

`AUTOEXEC.BAS` это Программа на языке BASIC, это файл можно использовать например, для выполнения инициализации (подготовки компьютера к работе). Подробно об этом процессе написано в разделе: VII.3. Инициализация в языке MSX BASIC.

Пример установки новых значений функциональных клавиш через autoexec.bas
autoexec.bas

Добавлено 2019-10-17 (примечание редактора)

CALL SYSTEM

Fix Me! Оператор `CALL SYSTEM` [92]

CMD

Fix Me! Оператор `CMD` [92]

ATTR$()

Fix Me! Функция `ATTR$()` [92]

IPL

Fix Me! Оператор `IPL` [92]

IX.1.3. Справочная информация о файлах

Старейшим из дошедших до нас каталогов признается

список литературных произведений на шумерской глиняной плитке, относящейся к 2000 г. до н.э..

—А.И.Михайлов

Сейчас мы расскажем Вам о довольно простых средствах для получения списка имён файлов, размещённых на дискете (FILES), размера её свободной части (DSKF) и некоторой другой справочной информации.

Вывод имён файлов осуществляется по команде (оператору!)

```
[L]FILES [I]
```

,

где:

- FILES, LFILES («файлы») — служебные слова;
- I — строковое выражение, значение которого определяет имя файла или родовое имя файлов.

При выполнении этой команды без параметра I список имён всех файлов диска или отображается на экране (если префикс L отсутствует), или распечатывается принтером (префикс L есть). В зависимости от типа экрана и его ширины индикация имён организуется в несколько колонок. Вывод списка файлов на печать проводится в 1 колонку: каждому имени отводится одна строка. При отсутствии файлов на диске выдаётся сообщение:

«File not found» (*«Файл не найден»*).

Если параметр I указан и на диске имеется файл с этим именем, то данный факт подтверждается выводом единственного имени I (в противном случае Вы прочтёте неприятное сообщение: «File not found»).

Для указания диска добавляется имя диска A:, B:, и т.д.

При использовании Nextor к команде можно добавить параметр L и команда может быть использована в таком виде:

```
FILES "B:",L
```

При использовании этого параметра список файлов выводиться в «длинном» формате, кроме имени/расширения выводится дата, размер и атрибуты файла.

Примеры.

```
FILES

100 FILES

FILES "Игры"

LFILES

FILES "B:"
```

Функция DSKF():

```
DSKF(n)
```

,

где:

- DSKF(«DiSK Free» — «свободная часть диска») — служебное слово;
- n — арифметическое выражение, целая часть значения которого должна принадлежать отрезку [0,8], возвращает размер в Килобайт свободной части диска. Целая часть значения n определяет имя диска:

Значение	Имя диска
0	Текущий
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H

Примеры.

```
PRINT DSKF(0):PRINT DSKF(1):PRINT DSKF(2)

X=1:PRINT DSKF(X)

IF DSKF(0)>200 THEN 1000
```

Забегая далеко вперёд, заметим, что для определения размера в байтах области дискеты, отведённой под произвольный файл, можно воспользоваться командами:

```
OPEN "Имя файла" FOR INPUT AS#1:PRINT LOF(1):CLOSE #1
```

Работа с секторами диска

Fix Me! Функция `DSKI$()` [92]

Fix Me! Функция `DSKO$` [92]

IX.1.4. Операторы NAME, COPY и KILL

Изменение имени программного файла или файла данных производится по команде (оператору)

```
NAME I AS J
```
,

где:

- `NAME` («имя») — служебное слово,
- I — строковое выражение, значение которого определяет «старое» имя файла,
- J — строковое выражение, значение которого определяет «новое» имя файла.

При выполнении этой команды прерывание происходит при отсутствии на дискете файла с именем I (нечего переименовывать!), при наличии на ней файла с именем J (имя J уже зарезервировано!) или при закрытой на запись дискете.

В противном случае файл с именем I получает новое имя J.

При задании имени J разрешено использование символов «?». Но при реальном формировании имени J на дискете каждый такой символ будет заменён на элемент из соответствующей позиции имени I.

Примеры.

```
NAME "алфавит" AS "АЛФ"
```

```
name X$ as "X.1"

200 NAME "PQR.XYZ" AS "?.?11"
```

В последнем случае новое имя будет выглядеть так:

```
 P ▲ .X11
      │
7 пробелов
```

Для дублирования на дисках программных файлов и файлов данных применяется команда (оператор)

```
COPY i TO j
```
,

где:

- COPY («копировать»), TO(«в») — служебные слова;
- i — строковое выражение, значение которого определяет имя исходного файла;
- j — строковое выражение, значение которого определяет имя формируемого файла.

При выполнении команды (оператора) COPY по файлу любого формата с именем i создаётся его копия с именем j. Имена i и j не должны совпадать. В том случае, когда файл с именем j уже существует на дискете, содержимое файла с именем j заменяется на содержимое файла с именем i и, естественно, «старое» содержимое файла с именем j пропадает!

Если i или j — файлы данных, то они не должны быть открыты!

Примеры. 1)

```
COPY "AL" TO "LA"
```

2)

```
COPY "X" TO "X.1"
```

3)

```
10 SAVE "U.1"
20 FOR J=1 TO 20
30 COPY"U.1"TO"U."+MID$(STR$(J),2)
40 NEXT
```

4)

```
COPY "A:ПРОБА" TO "B:ПРОБА"
```

Копирование всех файлов с одной дискеты на другую при наличии нескольких (больше двух) дисководов проводится операторами:

```
COPY "A:*.*" TO "B:*.*"
```
(с A на B)

```
COPY "B:*.*" TO "A:*.*"
```
(с B на A)

На одном дисководе эта процедура и ей подобные (см.пример 4) реализуются по соответствующим сообщениям на экране периодической сменой дискет в кармане дисковода.

Если служебное слово TO и второй аргумент будут опущены, то файл будет скопирован на активный дисковод (обычно с B: на A:), например:

```
COPY"b:E87.COM"
```
.

По команде (оператору)

```
KILL I
```
,

где:

- KILL («разрушать», «убивать») — служебное слово;
- I — строковое выражение, значение которого определяет имя файла, происходит разрушение на дискете файла с именем I и увеличение на соответствующую величину размера свободной части дискеты. Фактическому разрушению (стиранию) подвергается лишь справочная информация о файле.

Если I — имя файла данных, то при выполнении команды KILL он не должен быть открыт.

Примеры.

```
KILL"Бочка":KILL"R.T"

X$="ЮюЮ":KILL X$

100 FOR K=1 TO 25:KILL"ЦЕХ."+MID$(STR$(K),2):NEXT
```

IX.1.5. Операторы LOAD, SAVE, RUN и MERGE

По команде (оператору):

```
SAVE I [,A]
```
,

где:

- SAVE («сохранить») — служебное слово;
- I — строковое выражение, значение которого определяет имя файла, программа из оперативной памяти под именем I записывается на дискету. В зависимости от наличия или отсутствия параметра A запись проводится соответственно или в формате ASCII, или в форме внутреннего представления (см. Приложение 2 — 2.2. Внутренние коды служебных слов).

Попробуем прочесть программу

```
10 X=1
```
,

записанную:

- в кодах ASCII:

  ```
  10 X=1
  ```

- во внутреннем представлении (по кодам, см. Приложение 2 — 2.2. Внутренние коды служебных слов):

  ```
            FF   09   80   0A   00   58   EF   12   00   00   00
            ──   ──▲──   ──▲──   ──   ──   ──   ─▲   ──▲──
   Начало ▲    |     |    ▲  |  ▲   |    └ Конец
  программы┘ Адрес следу-  Номер      └ X =  1 ┘     Конец  программы
          ющей строки строки (10)        строки
  ```

Если данная программа впоследствии будет погружаться в память для слияния с другими программными модулями, то её необходимо записывать в коде ASCII.

Заметим что формат ASCII приводит к увеличению в несколько раз времени записи программ из оперативной памяти на дискету, а также их чтения с дискеты в память. К тому же для их размещения на дискете требуется приблизительно на 30% больше места, чем для соответствующих программ, написанных во внутреннем представлении (см. Приложение 2 — 2.2. Внутренние коды служебных слов).

При наличии на диске программного файла или файла данных с именем I и записи на него новой программы под тем же именем, старый файл пропадает и происходит изменение размера свободной части диска в соответствии с новыми размерами файла с именем I.

Примеры.

```
SAVE"PROTOOL.SYS"

X$="Z80.ABC":SAVE X$:SAVE"PRIMER.BAS",A

10 FOR M=1 TO 30:SAVE"палата"+MID$(STR$(M),2):NEXT
```

Для считывания программ с дискеты в оперативную память используются команды (операторы) LOAD и RUN.

1.

   По команде

   ```
   LOAD I[,R]
   ```
   ,

   где:

   - LOAD («загрузка») — служебное слово;
   - I — строковое выражение, значение которого определяет имя файла;
   - R — необязательный параметр;

   прежде всего закрываются все файлы и оперативная память очищается от программ и данных. Далее, программа, записанная на дискете в машинных кодах или формате ASCII под именем I, загружается в оперативную память.

   При наличии параметра R после загрузки производится запуск программы на счёт.

   Примеры.

   ```
   LOAD"X.Y"

   Z$=MID$(A$,5,3):LOAD Z$,R
   ```

   Ещё раз напомним Вам, что загрузка программы «стирает» программу, находящуюся в момент загрузки в памяти компьютера.

1. По команде

   ```
   RUN I[,R]
   ```
   ,

   где:

   - RUN («запуск», «прогон») - служебное слово,
   - I — строковое выражение, значение которого определяет имя файла, в оперативную память загружается и запускается на счёт программа, записанная на диске под именем I в машинных кодах или в формате ASCII.

   При отсутствии параметра R перед загрузкой файла с именем I закрываются все открытые файлы и память очищается от программ и данных.

   При наличии параметра R из памяти удаляются программы и значения переменных. Однако файлы данных *не закрываются* !

Пример[5].

0915-01.bas
0915-01.bas

```
1 'Меню многомодульного файла "MMF"
2 COLOR 1,7,13:SCREEN 1:PRINT:PRINT TAB(7)"Основное меню"
4 PRINT TAB(6)"──────────────────":PRINT
6 PRINT"1. программа A - 1":PRINT"2. программа B - 2"
8 PRINT"3. программа C - 3":PRINT"4. программа Д - 4"
11 INPUT R:IF R<1 OR R>4 THEN 11
13 ON R GOTO 14,15,16,17
14 RUN"прог.A",R
15 RUN"прог.B",R
16 RUN"прог.C",R
17 RUN"прог.Д",R
```

Приведём вид отдельного модуля, например, модуля с именем «прог.A»:

```
10 PRINT"Выполняется программа A"
20 RUN"MMF",R
```

Интересные возможности по «соединению» программ предоставляются командой (оператором!)

```
MERGE I
```

,

где:

- MERGE («слияние») — служебное слово;
- I — строковое выражение, определяющее имя файла, записанного на дискете в формате ASCII.

Выполнение команды MERGE начинается с закрытия всех файлов и чистки оперативной памяти от данных.

Далее производится слияние программы A, находящейся в памяти, с программным файлом, записанным на диске в формате ASCII под именем I.

Примеры

```
MERGE "TTT"

L$="техника":MERGE L$

100 MERGE "LIST.3"
```

Обратите внимание на тот факт, что программы, вызываемые командой MERGE, имеют более высокий приоритет, по сравнению с программами, находящимися в памяти компьютера. Поэтому если в объединяемых программах имеются строки с одинаковыми номерами, то строки программы в памяти компьютера будут заменены соответствующими строками «добавляемой» программы с именем I.

Наиболее целесообразно использовать этот оператор при работе с библиотеками подпрограмм на магнитной ленте или на дискете. Покажем, как это делается.

Пример.

Предположим, что программа, находящаяся в памяти, занимает строки с 10 по 1290. Для добавления подпрограммы, названной при записи на дискету «ROUT1» и занимающей строки с 10 по 190, сначала Вы должны сохранить программу, находящуюся в памяти.

После этого, Вы вызываете подпрограмму «ROUT1», используя команды LOAD или CLOAD.

Теперь используйте команду RENUM для перенумерации строк подпрограммы, начиная, например, со строки 10000.

Далее сохраните подпрограмму в формате ASCII, назвав её новым именем, например «ROUT0», после чего повторно

загрузите основную программу.

Теперь выполнение команды

```
MERGE "ROUT0"
```

приведёт к тому, что в конце текста основной программы будет добавлен текст подпрограммы с номерами строк, начинающимися с 10000.

IX.2. Файлы данных прямого доступа

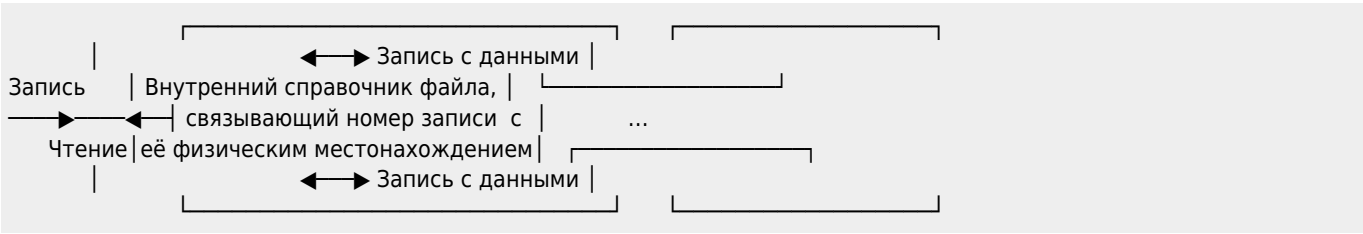
> Я знаю, что положил это в надёжное место, но теперь
> не могу вспомнить, в какое именно!
>
> *—Из диалога на приёме у врача*

Как мы уже отмечали (см. главу V), файл данных *прямого* доступа представляет собой последовательность нумерованных групп значений, называемых *записями*.

Запись представляет собой единицу хранимой в файле информации. Размер записи зависит от решаемой Вами задачи. Запись есть совокупность сведений о некотором объекте. Она образуется из *полей*, или элементов данных, указывающих на различные атрибуты, присущие конкретному объёму.

Так, если «объектами» для зубного врача являются пациенты, то каждая запись должна хранить информацию о каком–либо пациенте, а её полями могут быть имя пациента, его возраст, номер телефона и число поставленных пломб.

На рисунке показано зрительное представление организации файла прямого доступа. Записи с данными хранятся на дискете. При каждом запросе программы на чтение или на запись должен быть указан *номер записи*, по которому во внутреннем справочнике находится её фактическое положение на дискете, что позволяет немедленно произвести чтение или запись. Таким образом осуществляется прямой доступ к записям, требующий примерно одного и того же небольшого (зависящего от скорости вращения дискеты) времени для всех записей.



Нумерация записей выполняется последовательно, начиная с нуля. В запись объединяются логически связанные между собой данные, и номер записи обычно связывается с этими данными естественным образом.

Например, если записи содержат информацию о студентах некоторой группы, то в качестве номеров записей можно использовать номера студентов в списке группы.

Во время выполнения операции *вывода* для размещения записи будет выбрано место, определяемое её номером. При выполнении операции ввода по номеру отыскивается запись и передаётся в память.

Прямой доступ к файлу целесообразен в том случае, когда требуется часто менять содержимое записей и просматривать их в произвольном порядке.

Так, бронированные места в зале театра, меняющиеся от спектакля к спектаклю уместно хранить в файле прямого доступа.

Вы, как программист, должны предусмотреть размещение каждого символа в файле прямого доступа. В этом есть смысл: можно определить структуру файла в соответствии с Вашими потребностями.Вы сами регулируете поток данных в файле, и поэтому важно тщательно продумать его структуру. Разработка структуры файла не составит для

Вас труда, если ясно представлять себе, что происходит в программе.

Перейдём теперь к детальному описанию средств языка MSX BASIC, предназначенных для работы с файлами данных *прямого* доступа.

IX.2.1. Контрольные буферы файлов

Оператор

```
MAXFILES=A
```

,

где:

- `MAX`, `FILES` — служебные слова;
- A — арифметическое выражение, целая часть значения которого принадлежит отрезку [0,15], резервирует A+1 участок оперативной памяти для последующего *временного* хранения записей конкретных файлов.

Например:

```
10 MAXFILES=5

50 MAXFILES=X+Y

MAXFILES=3
```

Эти участки называются *буферами* или, более полно, *контрольными буферами* файлов («Field Control Buffers» — FCB) и обозначаются: FCB 0, FCB 1,... .

Каждый FCB занимает 265 байтов, из которых 9 байтов предназначены для управляющей информации, а 256 байтов — для данных.

Следует иметь в виду, что FCB0 и FCB1 объявлены по умолчанию и что FCB0 используется многими операторами (`SAVE`, `LOAD`, `MERGE`, `RUN` и т.п.).

При выполнении оператора `MAXFILES=` кроме резервирования буферов производится «чистка» значений переменных и закрытие всех ранее открытых файлов данных.

При выполнении операции *чтения* запись из файла помещается в контрольный буфер файла, а при выполнении операции вывода, наоборот, содержимое буфера переносится в файл, в область, отведённую для записи с указанным номером.

После выполнения операции чтения данные необходимо выбирать из буфера файла, а перед выполнением операции *вывода* данные должны быть помещены в буфер файла.

Функция

```
VARPTR(#n)
```

,

где:

- `VARPTR` («VARiable PoinTeR» — «указатель переменной») — служебное слово;
- n — арифметическое выражение, целая часть значения которого определяет номер FCB (от 0 до значения выражения, стоящего в правой части оператора `MAXFILES=`), возвращает *адрес* X байта, начиная с которого расположен в памяти FCBn.

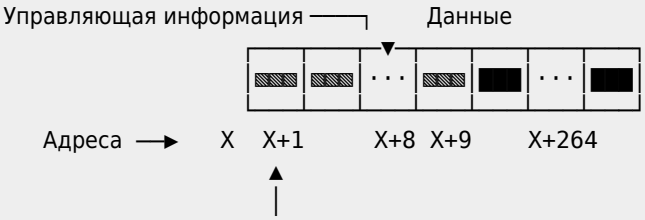
Если файл не существует, то выдаётся сообщение об ошибке:

«Illegal function call».

Приведём простейший пример:
0921-01.bas
0921-01.bas

```
10 MAXFILES=2:X=VARPTR(#0):Y=VARPTR(#1):Z=VARPTR(#2)
20 PRINT Y-X;Z-Y
run
 265  265
Ok
```

Взгляните на схему расположения информации в FCBn :



Контрольный буфер файла (FCB)

Номер байта	Имя байта	Содержимое байта
X	FL.MOD	Тип файла: 1 — OPEN FOR INPUT 2 — OPEN FOR OUTPUT 2 — OPEN FOR APPEND (для диска) 4 — файл прямого доступа (для диска) 4 — файл последовательного доступа, открытый и для чтения, и для записи, например, `OPEN "COM:" AS#1` 8 — OPEN FOR APPEND (для устройства MEM:)
X+1	FL.FCA	Ссылка на область памяти, в которой хранится спецификация файла
X+2	FL.LCA	
X+3	FL.LSA	?
X+4	FL.DSK	Номер текущего устройства: 0 — активное устройство по умолчанию; 1 — устройство A: 2 — устройство B: 3 — устройство C: 4 — устройство D: 5 — устройство E: 6 — устройство F: 7 — устройство G: 8 — устройство H: &hFC — устройство GRP: &hFD — устройство CRT: &hFE — устройство LPT: &hFF — устройство CAS: Нестандартные устройства: &hD4 — устройство COM: (локальная сеть MSX 1) &hC0 — устройство MEM:
X+5	FL.SLB	?
X+6	FL.BPS	?
X+7	FL.FLG	?
X+8	FL.OPS	?

Номер байта	Имя байта	Содержимое байта
X+9	FL.BUF	Содержимое записи файла (256 байт)

Спецификация файла

Номер байта	Содержимое байтов	Количество байт
+00	Номер дисковода (0 — по умолчанию, 1 — A:)	1
+01	Имя файла	8
...		
+08		
+09	Расширение имени файла	3
+12	Номер текущего блока	2
+14	Текущий размер записи (по умолчанию равен 128)	2
+16	Размер файла в байтах	4
+20	Дата изменения файла	2
+22	Время изменения файла	2
+24	Идентификатор устройства	1
+25	Расположение каталога	1
+26	Первый кластер файла	2
+28	Последний кластер файла	2
+30	Последний доступный кластер (относительно начала файла)	2
+32	Номер текущей записи файла последовательного доступа	1
+33	Номер случайной записи для файла прямого доступа (байты 33,34,35 — если размер записи больше 63; байты 33,34,35,36 — если размер записи меньше 64)	4
+34		
+35		
+36		

IX.2.2. Операторы OPEN и CLOSE

Для работы с файлами данных,уже существующими на дискетах или вновь организуемыми, необходимо произвести их *открытие*. Делается это оператором

```
OPEN B AS[#]n [LEN=m]
```
,

где:

- OPEN («открыть»), LEN(«LENgth» — «длина») — служебные слова;
- B — строковое выражение, значение которого определяет имя файла;
- n, m — арифметические выражения, целые части значений которых определяют соответственно *номер* контрольного буфера файла и *длину* его записи в байтах;
    - целая часть значения n должна принадлежать отрезку [0,6], так как дисковод может одновременно работать только с *семью* различными файлами;
    - целая часть значения m должна принадлежать отрезку [1,256] (по умолчанию значение m равно 256).

Все записи файла с *прямым* доступом имеют одинаковую длину, которая (в байтах) задаётся параметром m . Это значение определяет способ разбиения на записи дискового пространства, отведённого для файла.

Например, если длина записи равна 50,то 6–я запись будет располагаться со смещением 300 (50×6) от начала файла;

- # — необязательный символ, никак не влияющий на выполнение оператора OPEN .

При выполнении оператора OPEN файлу с именем B назначается для работы контрольный буфер с номером n. Кроме того:

- с дискеты в буфер заносится управляющая информация о файле (если файл с именем B уже существует) или
- эта информация создаётся на дискете (если файл с именем B формируется заново).

Вся эта процедура и называется *открытием* файла.

Отметим, что один файл B не может быть открыт сразу по нескольким FCB!
Например:

```
10 OPEN "A" AS#1                                90 OPEN "klukva" AS#1 LEN=50
MAXFILES=2:OPEN"Старт"AS#2                      MAXFILES=5:OPEN"Финиш"AS#3 LEN=100
```

Оператор OPEN открывает файл с *прямым* доступом, если спецификации режимов FOR INPUT, OUTPUT или APPEND опущены.

Если программа закончит запись в файл, а в буфере останутся некоторые, не записанные на дискету данные, программа должна тем или иным способом все же завершить перепись содержимого буфера. Сделать это можно с помощью оператора *закрытия* файла, к изучению которого мы и приступаем.

Оператор

```
CLOSE [[#]N1,[#]N2,...,[#]Nk]
```

,

где:

- CLOSE («to close» — «закрывать») — служебное слово;
- N1,N2,...,Nk — арифметические выражения, целые части значений которых должны принадлежать отрезку [0,15],

организует *закрытие* открытых с помощью оператора OPEN файлов, имеющих номера контрольных буферов файлов, равных значениям выражений: N1, N2,..., Nk .

Оператор CLOSE без параметров закрывает *все* открытые на данный момент файлы данных.

Однако большинство программистов предпочитают иметь гарантированный точный результат работы оператора CLOSE и поэтому указывают в нем номера файлов в явном виде.

Закрытый файл становится недоступным для формирования новых или обновления оператором PUT уже имеющихся записей.

Приведём примеры синтаксиса оператора CLOSE :

```
CLOSE

CLOSE #0

CLOSE 1.5,7
```

Заметим, что закрытие всех ранее открытых файлов происходит также при выполнении операторов END, CLEAR, LOAD, MAXFILES=, NEW и любом *изменении* текста программы !

Пока файл не закрыт, существует опасность, что прекращение работы программы — из-за ошибки пользователя, отключения электроэнергии или ошибки в программе — приведёт к утрате данных или даже к повреждению дискеты. Опыт подсказывает, что желательно закрывать файл, как только исчезнет необходимость в том, чтобы он был открыт.

IX.2.3. Оператор FIELD

Как мы уже отмечали,под каждый контрольный буфер файла (FCB) отводится 265 байтов, 256 из которых непосредственно предназначены для временного хранения *записи* файла.

Перед началом непосредственной работы с контрольным буфером файла необходимо произвести его разбиение на отдельные *поля* для формирования в них конкретных значений. Делается это при помощи оператора

```
FIELD [#]n, M1 AS γ1, M2 AS γ2,… , Mk AS γk
```
,

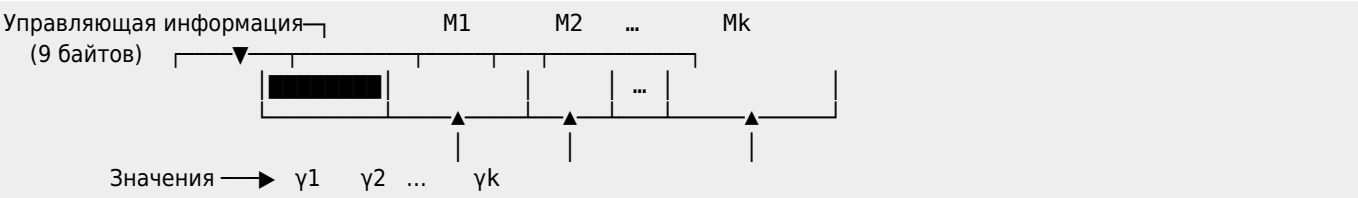
где:

- FIELD («field» — «поле») — служебное слово;
- n — арифметическое выражение, целая часть значения которого определяет номер контрольного буфера файла;
- M1,M2, ...,Mk — целые числа.
  Учтите, что суммарная длина всех объявляемых в операторе FIELD полей не должна превосходить длины записи, установленной оператором OPEN. Нарушение этого правила приводит к ошибке;
- γ1,γ2, ...,γk — строковые переменные (простые или с индексами), называемые *буферными* переменными;
- # — необязательный символ, никак не влияющий на выполнение оператора.

При выполнении команды FIELD производится задание формата записи. Это выражается в выделении каждому значению буферной переменной γs *поля* из Ms байтов (s=1, 2, ...k).

Ни одну из буферных переменных оператора FIELD нельзя использовать ни в каких модификациях оператора INPUT и ни одной из них нельзя присваивать значение оператором LET. Нарушение любого из этих правил приводит к утрате соответствия, установленного оператором FIELD, вследствие чего программа не может больше использовать такую переменную для занесения нужных значений в записи файла или для их извлечения.

Разбиение FCB на поля оператором FIELD покажем на рисунке:



Например, оператор FIELD #1,200 AS X$, 12 AS Y$ отводит 200 байт для переменной X$ и 12 байтов — для переменной Y$.

Заметим, что этот оператор будет выполнен только в том случае, если файл был предварительно открыт.

Пример 1.

В качестве примера рассмотрим файл с прямым доступом, содержащий информацию о личной библиотеке. Каждая запись содержит следующую информацию:

1. номер книги (2 байта);
2. фамилию автора книги (15 байтов);
3. название книги (30 байтов);
4. признак наличия книги (1 байт);
5. информацию о читателе, взявшем книгу (22 байта);
6. дату выдачи книги (10 байтов).

Для открытия этого файла и описания структуры записи можно использовать следующие операторы:
0923-01.bas
0923-01.bas

```
10 OPEN "Библтека" AS#1 LEN=80 'В имени - не более 8 символов!
```

```
20 FIELD #1,2 AS NUMBER$,15 AS AUTHOR$,30 AS BOOK$,1 AS CODE$, 22 AS RADER$,10 AS DATE$
```

Отметим, что разрешается использовать несколько команд FIELD с разными форматами для одного и того же буфера n. Всегда действует то разбиение FCBn на поля, которое предложено последней выполненной командой FIELD.

Пример 2.
0923-02.bas
0923-02.bas

```
10 MAXFILES=3:OPEN"Student" AS#2 LEN=40
20 FIELD #2,28 AS M1$,12 AS M2$
30 FIELD #2,10 AS M1$,5  AS M2$,24 AS K$
```

Оператор FIELD должен выполняться перед обращениями к файлу с помощью операторов PUT и GET (см. раздел IX.2.6.).

Прежде чем обратиться к файлу, Вы должны продумать, какие данные будут составлять записи, каковы их тип и длина (размер). В результате Вы получите документ, часто называемый *логической структурой файла*. С ним Вам предстоит работать в течение всего времени «жизни» файла. Поскольку в этом документе указано число байтов, отведённое под каждую запись, можно заранее установить количество байтов, которое потребуется на дискете для конкретного файла, а так как известен и тип применяемых данных, то становится очевидным, какие придётся вводить переменные. Важно отметить, что при известной логической структуре файла его в дальнейшем могут использовать другие программы.

Ниже приведена структура записи в файле и дан расчёт размера файла для 24 записей о пловцах. Каждая запись содержит поля для имени пловца, клуба, к которому он принадлежит, дисциплины, в которой он выступает, а также поля для его возраста и лучшего показанного результата.

Имя файла: SWIMMER .
Тип файла: прямого доступа.
Содержимое: 24 записи о пловцах.

Имя переменной	Описание поля	Длина поля (байт)	Тип данных
N$	Имя пловца	30	
T$	Принадлежность к клубу	20	20 символов
E$	Дисциплина	8	8 символов
A$	Возраст	2	Целое число
T	Лучший результат	4	Вещественное число
	Итого:	**64**	

Длина и тип каждого поля указаны вместе с переменными, которые в Вашей программе будут хранить данные, извлечённые из записей. Совершенно необязательно использовать одни и те же переменные во всех программах, обращающихся к этому файлу, однако такое единообразие позволит легче проследить ход обработки данных в любой программе.

Ещё раз подчеркнём, что длина записи в файле с прямым доступом *постоянна*. Она определяется при его создании, и каждому полю выделяется внутри записи строго определённое место. Возможные значения полей должны умещаться в этом заранее отведённом объёме памяти. Если это не продумано заблаговременно, то при выполнении программы чрезмерно «длинные» значения подвергаются усечению, а «короткие» дополняются пустующими ячейками памяти. В целях минимизации нежелательного усечения размер каждого поля обычно выбирается достаточно большим, чтобы вместить самую большую возможную величину.

IX.2.4. Операторы LSET и RSET

Напомним Вам, что команда FIELD определяет формат записи файла и набор буферных переменных. Далее буферные переменные не могут использоваться в операторах LET, INPUT, LINE INPUT.

При формировании в памяти конкретной записи значения буферным переменным из FIELD присваиваются операторами:

```
LSET γ=β
```
,

```
RSET γ=β
```

где:

- LSET («Left SET» — «поместить слева»),
- RSET («Right SET» — «поместить справа») — служебные слова;
- γ — имя буферной переменной;
- β — строковое выражение (β≠γ).

Пусть буферной переменной γ отведено в буфере поле размером m байтов. Выполнение оператора LSET начинается с вычисления значения выражения β. Далее, если это необходимо, его длина приводится к величине m за счёт отбрасывания лишних *правых* символов или добавления справа недостающего количества пробелов. Полученное значение присваивается буферной переменной γ и размещается в соответствующем поле буфера файла.

Оператор RSET выполняется почти так же, как и оператор LSET. Разница состоит в том, что при выравнивании значения β до длины m возможные недостающие пробелы добавляются к нему не справа, а *слева*. Неиспользованные байты заполняются пробелами. Таким образом, операторы LSET и RSET гарантируют, что длина значения переменной будет приведена в соответствие с длиной, указанной в операторе FIELD.

Пример 1.
0924-01.bas
0924-01.bas

```
10 OPEN "EF" AS#1 LEN=9
20 FIELD #1,5 AS M$,14 AS L$
30 INPUT R$:LSET M$=R$
40 INPUT R$:RSET L$=R$
50 PRINT M$:PRINT L$
run
? КРАТЕР                    M$
? НЕБОСВОД    ┌───┬───┬───┬───┬───┐
КРАТЕ             │ К │ Р │ А │ Т │ Е │
   НЕБОСВОД   └───┴───┴───┴───┴───┘
Ok                        5 байтов
```

Пример 2.
0924-02.bas
0924-02.bas Приведём фрагмент программы, в котором формируется запись файла, содержащего информацию о личной библиотеке. В буфер заносится информация о романе Л.Н.Толстого «Война и мир». Структура записи была рассмотрена ранее.

```
10 OPEN "Библтека" AS#1 LEN=80
20 FIELD #1,2 AS NUMBER$,15 AS AUTHOR$,30 AS BOOK$,1 AS CODE$, 22 AS RADER$,10 AS DATE$
30 LSET NUMBER$=10                  'Шифр книги
40 LSET AUTHOR$="Л.Н.Толстой"
50 LSET BOOK$="Война и мир"
60 LSET CODE$="X"                   'X - книга хранится
70 LSET READER$=" "                 'Читатель, разумеется, отсутствует
80 LSET DATE$="10.5.1988"
```

Команды LSET и RSET можно использовать не только при работе с файлами.

Пример 3.
0924-03.bas

0924-03.bas

```
10 L$=SPACE$(20):Z$="Волк"
20 RSET L$=Z$:PRINT L$
```

IX.2.5. Функции MKI$(), MKS$(),MKD$(), CVI(), CVS(), CVD()

Вам, конечно, известно, что буферные переменные являются строковыми. Пусть γ — буферная переменная и ей командой `FIELD` в FCB отведено поле длиной m байтов. Если требуется «упаковать» в γ числовое значение, то пользуются функциями:

1. `MKI$(L)`,

   где:

   - `MKI` («MaKe Integer» — «преобразовать целое значение») — служебное слово;
   - L — арифметическое выражение целого типа;
   - m=2;
2. `MKS$(L)`,

   где:

   - `MKS` («MaKe Single» — «преобразовать значение одинарной точности») — служебное слово;
   - L — арифметическое выражение одинарной точности;
   - m=4;
3. `MKD$(L)`,

   где:

   - `MKD` («MaKe Double» — «преобразовать значение двойной точности») — служебное слово;
   - L — арифметическое выражение двойной точности;
   - m=8.

Каждая из этих функций преобразует значение арифметического выражения L к строковому типу. После этого операторами LSET или RSET эти значения можно разместить в γ и соответствующем поле буфера.

Обращаем Ваше внимание на тот факт, что функции `MKI$()`, `MKS$()` и `MKD$()` не переводят числовые величины в эквивалентные им знаки кода ASCII, как это делается при использовании других функций. Они просто «упаковывают» соответствующие значения в два,четыре или восемь знаков по той же схеме, которая реализуется в компьютере для представления числовых величин в оперативной памяти. Именно поэтому с помощью оператора `PRINT` невозможно выдать на печать или экран дисплея результаты работы функций `MKI$()`, `MKS$()` и `MKD$()` !

Пример 1.

1. функция `MKI$(17217)` возвращает строку «AC». Рассмотрим, как это получается:

```
17217=&B0100001101000001=&H4341=&H43·256+&H41=67·256+65
        └──────┘ └──────┘                      ▲       ▲
            C        A                        │       └─ Код символа "A"
                     └──────── Код символа "C"
```

2. `MKS$(4.14141)` возвращает строку "AAAA"

Пример 2.
0925-02.bas
0925-02.bas

```
1 A$=MKI$(&HE276):PRINT HEX$(ASC(MID$(A$,1,1)));" ";HEX$(ASC(MID$(A$,2,1)))
```

```
76 E2
Ok
```

Пример 3.
0925-03.bas
0925-03.bas

```
10 MAXFILES=5:OPEN "LU" AS#4
20 FIELD #4, 2 AS A$, 4 AS B$, 8 AS E$
30 INPUT X%:LSET A$=MKI$(X%)
40 INPUT Y!:LSET B$=MKS$(Y!)
50 INPUT Z#:LSET E$=MKD$(Z#)
60 PRINT A$;" ";B$;" ";E$
70 RSET A$=MKI$(X%):RSET B$=MKS$(Y!):RSET E$=MKD$(Z#)
80 PRINT A$;" ";B$;" ";E$
run
? 1096
? 1.535
? -3.4
H A5 a4
H A5 a4
Ok
```

Функции `MKI$()`, `MKS()` и `MKD()` выполняют обратное преобразование по отношению к функциям `CVI()`, `CVS()` и `CVD()`, к рассмотрению которых мы сейчас приступаем.

Пусть в буфер с дискеты считана запись и в некотором её поле,определённом по команде `FIELD` элементом

```
m AS γ
```

«упаковано» числовое значение. «Распаковка» его или, выборка из γ этого значения, реализуется одной из следующих функций:

1. `CVI(γ)`,

   где:

   - `CVI` («ConVert to Integer» — «преобразовать к целому») — служебное слово;
   - γ — имя буферной переменной;
   - m=2;
2. `CVS(γ)`,

   где:

   - `CVS` («ConVert to Single» — «преобразовать к значению одинарной точности») — служебное слово;
   - γ — имя буферной переменной;
   - m=4;
3. `CVD(γ)`,

   где:

   - `CVD` («ConVert to Double» — «преобразовать к значению двойной точности») — служебное слово;
   - γ — имя буферной переменной;
   - m=8.

Эти функции преобразуют строки символов в числа. Аргумент состоит из 2, 4 или 8 байтов и преобразуется в целое число, либо число с одинарной или двойной точностью, в зависимости от вида используемой функции (`CVI`, `CVS` или `CVD`).

Пример 4.

1. ```
   print CVI("AC")
    17217
   Ok
   ```
   ,

   так как

   ```
   print MKI$(17217)
   AC
   Ok
   ```

2. ```
   print CVS("GWYF")
    5759460
   Ok
   ```
   ,

   так как

   ```
   print MKS$(5759460)
   GWYF
   Ok
   ```

Посмотрите на схему...

```
      "G"         "W"         "Y"         "F"
   0100 0111   0101 0111   0101 1001   0100 0110
   ▲──▲────     5    7      5    9      4    6
   │   │
  │Порядок числа = &B1000111-&B1000000 = &B111 = 7
Знак числа
```

Сравните эти результаты с представлением чисел в памяти компьютера (см. далее главу 10).

Пример 5.
0925-05.bas
0925-05.bas

```
10 OPEN "SL" AS#1 LEN=8:FIELD #1,8 AS R$
20 INPUT X:RSET R$=MKD$(X)
40 PRINT R$,CVD(R$)
run
? 123
C0          123
Ok
```

IX.2.6. Операторы PUT и GET

Fix Me!

Для пересылки («сброса») сформированной в оперативной памяти записи на дискету используется оператор

```
PUT [#]n[,P]
```
,

где:

- PUT («записать», «разместить») — служебное слово;
- n, P — арифметические выражения, целые части значений которых задают соответственно номер контрольного буфера файла и номер записи.

До выполнения оператора PUT в оперативной памяти:

- оператором MAXFILES= должен быть объявлен контрольный буфер файла FCBn ;
- открыт некоторый файл B, связанный с FCBn (оператором OPEN);
- задан формат его записей (оператором FIELD) и, наконец,
- в FCBn создана сама запись (операторами LSET, RSET).

После всей этой процедуры оператором PUT содержимое FCBn будет выведено под номером P на дискету, или, как говорят, *сформирована* новая или *обновлена* старая запись P файла B.

При открытии файла значение P равно 0. *Текущим* значением P считается номер той записи, с которой работал последний выполненный оператор PUT или GET. Далее, если в операторе PUT параметр P не указан, то берётся его текущее значение, увеличенное на 1.

Например, в операторе PUT #1,3 второе число (3) определяет номер записи в файле. Если этот номер будет пропущен, то будет использовано число, на единицу большее,чем в последнем выполненном операторе PUT или GET.

Номер записи может находиться в диапазоне от 1 до 4294967295, в чем Вы можете убедиться самостоятельно.

Пример 1.
0926-01.bas
0926-01.bas

```
10 REM формирование файла
20 MAXFILES=2:OPEN"массив" AS#2 LEN=8:FIELD #2,8 AS M
50 FOR I=1 TO 30
60    INPUT"Введите число";A:LSET M=MKD$(A):PUT #2
90 NEXT
100 CLOSE #2
```

Пример 2.
0926-02.bas
0926-02.bas

```
10 'Объявление FCB0, FCB1, FCB2 и FCB3
20 'Открытие файла "рыба" с буфером #2
30 MAXFILES=3:OPEN "Рыба" AS#2 LEN=13
40 'Определение формата записи
50 FIELD #2,8 AS L$, 5 AS M$
60 'Формирование первых 10 записей
70 FOR K=1 TO 10
80    INPUT"X$-8";X$:INPUT"Y$-5";Y$
90    LSET L$=X$:RSET M$=Y$:PUT #2,K
99 NEXT:END
```

Пусть открыт некоторый файл данных, связанный с буфером FCBn. Полезную информацию о файле можно извлечь с помощью функций LOF и LOC.

Для считывания записей файла с дискеты в оперативную память используется оператор

```
GET [#]n[,P]
```
,

где:

- GET («to get» — «получать») — служебное слово;
- n, P — арифметические выражения, целые части значений которых задают соответственно номер контрольного буфера файла и номер записи.

До выполнения оператора GET необходимо:

- объявить контрольный буфер файла FCBn (оператором MAXFILES=);
- открыть некоторый файл данных B,связанный с FCBn (оператором OPEN);
- задать формат его записей (оператором FIELD).

После всей этой процедуры по оператору GET с диска в буфер # n будет считана запись с номером P. Фактически будут сформированы значения всех строковых переменных буфера (числовые значения буферных переменных надо «распаковать» с помощью функций CVI, CVS и CVD).

Например, оператор GET #1,3 выполняет передачу записи с номером 3 из файла #1 в буфер полей.

Если в записи оператора параметр P отсутствует, то берётся текущее значение этого параметра, увеличенное на единицу (как и в операторе PUT).

Пример 3.
0926-03.bas
0926-03.bas

```
10 REM Считывание из файла "Массив"
20 MAXFILES=2:OPEN"Массив" AS#2 LEN=8:FIELD 2,8 AS M
50 FOR I=30 TO 1 STEP-1
60     GET 2,I:PRINT CVD(M);
80 NEXT
```

Пример 4.
0926-04.bas
0926-04.bas

```
1 OPEN"числа" AS#1 LEN=10
2 FIELD #1,2 AS A$(1),2 AS A$(2),2 AS A$(3),2 AS A$(4),2 AS A$(5)
3 FOR I=1 TO 5:LSET A$(I)=MKI$(I):NEXT I
8 PUT #1,1:CLOSE #1:END
10 OPEN"числа" AS#1
11 FIELD #1,2 AS B$(1),2 AS B$(2),2 AS B$(3),2 AS B$(4),2 AS B$(5)
12 GET #1,1
13 FOR I=1 TO 5:A(I)=CVI(B$(I)):NEXT I
18 P=1:FOR T=1 TO 5:P=P+A(T):NEXT T
19 PRINT P:CLOSE #1:END
```

Пример 5.
0926-05.bas
0926-05.bas

```
10 CLOSE:OPEN"числа" AS#1 LEN=25
30 FIELD #1,5 AS A$(1),5 AS A$(2),5 AS A$(3),5 AS A$(4), 5 AS A$(5)
40 FOR T=1 TO 5
50     LINEINPUT"Еще слово:";J$(T):LSET A$(T)=J$(T)
60 NEXT T
70 PUT#1,1:CLEAR 'Чистка оперативной памяти
90 OPEN"числа" AS#1
100 FIELD #1,5 AS B$(1),5 AS B$(2),5 AS B$(3),5 AS B$(4), 5 AS B$(5)
110 GET#1,1:U$=""
120 FOR T=1 TO 5:U$=U$+B$(T):NEXT T
150 PRINT"Сумма:";U$
```

Пример 6.
0926-06.bas
0926-06.bas

```
10 CLOSE:OPEN"числа" AS#1 LEN=25
20 FIELD #1,5 AS A$(1),4 AS A$(2),5 AS A$(3),4 AS A$(4), 5 AS A$(5)
30 FOR T=1 TO 5 STEP 2
```

```
40     LINEINPUT"Еще слово:";J$(T)
50     LSET A$(T)=J$(T):NEXT
60 FOR T=2 TO 4 STEP 2
70     INPUT"Еще число";J(T)
80     LSET A$(T)=MKS$(J(T)):NEXT
90 PUT #1,1:CLEAR  'Чистка оперативной памяти
110 OPEN"числа" AS#1
120 FIELD #1,5 AS B$(1),4 AS B$(2),5 AS B$(3),4 AS B$(4), 5 AS B$(5)
130 GET #1,1:U$=B$(1)
140 FOR T=2 TO 4 STEP 2
150     U$=U$+STR$(CVS(B$(T)))+B$(T+1):NEXT
160 PRINT"Сумма:";U$
```

Отметим ключевой момент при работе с файлами *прямого* доступа: отслеживание «невидимых» указателей. Их два: для дискеты и для буфера.

Первый указатель содержит ссылку на текущую запись и задаётся номером в операторах PUT или GET.

Буферный указатель ссылается на элемент данных — переменную, которая в этот момент пишется или читается. Он задаётся длиной полей в операторе FIELD.

Ниже приведены *две схемы*, дающие наглядное представление о последовательности операций, которые должны быть выполнены при перемещениях *записей* файлов прямого доступа из оперативной памяти на дискету и в обратном направлении.

1. Общая схема формирования *новых* записей уже существующего или вновь создаваемого файла F.
    - A. В оперативной памяти резервируется буфер (оператором MAXFILES=)
    - B. *Открывается* файл F (оператором OPEN)
    - C. Производится распределение полей буфера файла под значения переменных (оператором FIELD)
    - D. В буфере формируется запись (при помощи операторов SET, RSET и функций MKI$(), MKS$(), MKD$())
    - E. Производится «сброс» записи из буфера на дискету с присваиванием ей некоторого номера (оператором PUT)
    - F. Файл F *закрывается* (оператором CLOSE)

Пример 7. Запись слова в файл прямого доступа.
0926-07.bas
0926-07.bas

```
10 MAXFILES=2:OPEN "EASY" AS#2:FIELD #2,18 AS N$,110 AS D$
20 LSET N$="Психокомпьютерапия":LSET D$=" ":PUT #2,1:CLOSE #2
```

2. Общая схема *считывания* записей из уже существующего файла.
    - A. В оперативной памяти резервируется буфер (оператором MAXFILES=)
    - B. *Открывается* файл F (оператором OPEN)
    - C. Производится распределение полей буфера файла под значения переменных (оператором FIELD)
    - D. Производится считывание с диска в буфер оперативной памяти записи с конкретным номером (оператором GET)
    - E. По записи из буфера в соответствии с распределением его поля формируются значения требуемых переменных (при помощи функций CVI(), CVS(), CVD()). После перенесения записи в буфер Вы можете манипулировать данными, полученными из файла, так же, как любыми другими переменными.

      Пункты D и E повторяются столько раз, сколько это требуется

    - F. Файл F *закрывается* (оператором CLOSE)

Пример 8. Чтение слова из файла прямого доступа.
0926-08.bas
0926-08.bas

```
10 MAXFILES=2:OPEN "EASY" AS#2:FIELD #2,18 AS N$,110 AS D$
20 GET #2,1:PRINT N$:CLOSE #2
run
Психокомпьютерапия
Ok
```

В заключении этого пункта приведём примеры, иллюстрирующие действие изученных Вами операторов и функций при работе с дисковыми файлами данных прямого доступа.

Пример 9.
0926-01.bas
0926-09.bas

```
5 CLS
10 OPEN "ЭКЗАМЕН" AS#1 LEN=45:FIELD#1,5 AS G$,25 AS F$,1 AS O$
25 INPUT "Введите количество студентов";N
30 FOR I=1 TO N
40    INPUT "Группа";GG$:INPUT "Ф.И.О";FF$:INPUT"Оценка";OO$
50    LSET G$=GG$:LSET F$=FF$:LSET O$=OO$:PUT #1,I
65 NEXT:CLS
66 FOR I=1 TO N
70    GET #1,I:IF O$="5" OR O$="4" THEN PRINT G$,F$;O$
80 NEXT
90 CLOSE #1
```

Пример 10. Дана таблица:

Ф.И. ученика	**Тема**	**Балл**
Фомина Н.	Графика	5
	Массивы	5
Бобкова Н.	Графика	4
	Массивы	3
...		
Герасимова Е.	Графика	5
	Массивы	3

Заполнить два файла:

1. список учеников, у которых средний балл по двум темам ≥ 4 ;
2. список учеников, у которых хотя бы по одной теме балл 4 или 5.

0926-10.bas
0926-10.bas

```
10 MAXFILES=3
20 OPEN "экзамен"AS#1 LEN=39
30 FIELD #1, 15 AS I$,  10 AS T1$,  2 AS O1$,  10 AS T2$,  2 AS O2$
31'               ▲          ▲          ▲          ▲          ▲
32'               |          |          |          |          |
33'    Фамилия,имя  Первая   Балл по   Вторая   Балл по
34'            тема   первой теме   тема   второй теме
40 OPEN "Фамилия"AS#2 LEN=15
50 FIELD #2,15 AS F$      'Фамилия ученика, у которого средний балл  по 'двум темам ≥ 4
60 OPEN "особенное"AS#3 LEN=15
70 FIELD #3,15 AS F1$     'Фамилия ученика, у которого хотя бы по одной 'теме балл 4 или 5
75 INPUT "Введите количество учащихся";N
80 FOR I=1 TO N
90        INPUT  "Ф.И.";D$:LSET  I$=D$:INPUT  "Тема1";D$:LSET  T1$=D$:INPUT"Балл";D%:LSET
O1$=MKI$(D%):INPUT  "Тема2";D$:LSET  T2$=D$:INPUT  "Балл";D%:LSET  O2$=MKI$(D%):PUT#1,I
110 NEXT:A=0        'A  -  количество учеников, у которых средний балл по 'двум темам ≥ 4
120 FOR K=1 TO N:GET#1,K                              'Заполнение файла #2
125     L=(CVI(O1$)+CVI(O2$))/2
130     IF L>=4 THEN LSET F$=I$:A=A+1:PUT #2,A
140 NEXT K
146 PRINT"Ф.И. хорошистов и отличников:"
```

```
150 FOR I=1 TO A:GET #2,I:PRINT F$:NEXT I          'Вывод на экран
170 B=0            'B  -  количество учеников,у которых хотя бы по одной 'теме балл 4 или 5
175 FOR K=1 TO N                                   'Заполнение файла #3
177     GET #1,K
180     IF CVI(O1$)=4 OR CVI(O1$)=5 OR CVI(O2$)=4 OR CVI(O2$)=5 THEN LSET F1$=I$:B=B+1:PUT #3,B
190 NEXT K
195 PRINT"Ф.И. успевающих"
200 FOR I=1 TO B:GET#3,I:PRINT F1$:NEXT I:END      'Вывод на экран
```

Пример 11.
0926-11.bas
0926-11.bas

```
10 MAXFILES=3:OPEN"A3" AS#2 LEN=2:FIELD #2,2 AS K$
20 POKE VARPTR(#2)+9,66:POKE VARPTR(#2)+10,65  'Вместо LSET K$="ba"
30 PUT #2,2:CLOSE#2
40 OPEN"a3"AS#2 LEN=2:FIELD #2,2 AS K$:GET #2,2:CLOSE #2
run
ba
Ok
```

IX.3. Файлы данных последовательного доступа

При использовании дискеты достаточно указать имя файла, чтобы компьютер начал поиск с использованием каталога. Все участки дискеты в равной степени доступны для записи и чтения, т.е. для обращения к любому участку требуется примерно одинаковое время. Напомним Вам, что

файл, в котором информация может быть найдена без полного просмотра носителя, мы назвали файлом с *прямым* доступом.

Файл будем называть файлом с *последовательным доступом*, если компьютер должен «просматривать» его с самого начала до момента нахождения нужной части.

Поиск файла на кассетной ленте называется *последовательным*, потому что компьютер ищет требуемый файл, последовательно перематывая магнитную ленту.

Любой файл можно организовать так, чтобы можно было использовать либо последовательный, либо прямой доступ, но не оба способа одновременно. Каждый способ имеет свои «за» и «против». Последовательный способ намного проще для программирования и при его реализации требуется меньше дисковой памяти, но может понадобиться больше времени для поиска информации, находящейся «в конце» длинного файла. Кроме того, обновление существующих записей в файлах с последовательным доступом трудноосуществимо, а иногда и просто невозможно.

Файлы с прямым доступом требуют более сложного программирования и занимают обычно больше места на дискете, но очень легко поддаются обновлению, и поиск нужной информации осуществляется с одинаковой скоростью.

Если при каждом обращении к файлу Вы собираетесь использовать почти все данные, а менять их содержимое часто не предполагается, то выбирайте метод *последовательного* доступа. Его применение будет и более оптимальным и облегчит Вам программирование.

Файлы данных последовательного доступа состоят из отдельных групп значений, называемых *логическими* строками (или просто *строками*).

Логические строки не имеют номеров и их можно считывать с дискеты или магнитной ленты в оперативную память или записывать из памяти на дискету или магнитную ленту лишь последовательно друг за другом, причём запись производится от начала или конца файла, а чтение — только от его начала.

Каждая логическая строка имеет специальную *концевую* метку, а весь файл имеет ещё и метку EOF («End Of File» —

«конец файла»).

Кроме того, любой конкретный файл открывается или только для чтения, или только для записи, но не для одновременного выполнения этих операций.

Для работы с файлами данных *последовательного* доступа в MSX Disk BASIC предусмотрен ряд операторов и функций.

IX.3.1. Оператор MAXFILES=, OPEN и CLOSE

Перемещение логических строк последовательных файлов из оперативной памяти на дискету и в обратном направлении производится через *контрольные буферы файлов*. Последние объявляются оператором MAXFILES=. Максимальное количество файлов, которые могут быть открыты одновременно, устанавливается оператором

```
MAXFILES=A
```
,

где:

- MAX, FILES — служебные слова;
- A — арифметическое выражение, целая часть значения которого принадлежит отрезку [0,15].

При включении компьютера значение параметра A равно 1. Таким образом, если Вы хотите открыть *только один* файл, то использование оператора MAXFILES необязательно.

Файл #0 всегда открыт. MSX Disk BASIC использует контрольный буфер этого файла для «своих нужд» (он используется при выполнении операторов LOAD, SAVE, KILL, RUN, MERGE).

При работе с последовательными файлами буфера используются самым простым способом: пересылаемые в файл данные накапливаются в буфере файла, а по заполнении буфера его содержимое сразу целиком переписывается в дисковый файл;при этом буфер файла очищается для следующей порции выходных данных. При чтении файла соответствующие данные поступают из того же самого буфера, а не считываются непосредственно с поверхности дискеты. Как только все необходимые данные считаны из буфера, средства MSX BASIC обеспечивают повторное его заполнение информацией из дискового файла.

Учтите,что при использовании оператора MAXFILES= автоматически выполняется оператор CLEAR, но если Вы используете в программе оба оператора, то располагайте оператор MAXFILES= вслед за оператором CLEAR.

Вам, конечно же, ясно, что перед использованием файла Вы должны объявить его *открытым*. Открытие файла осуществляется оператором OPEN, в котором задаётся:

- имя устройства (A:, B:, MEM:, CAS:, CRT:, GRP:, LPT:, COM:);
- имя файла;
- направление потока данных;
- номер, присвоенный файлу для передачи данных.

Синтаксис оператора OPEN:

```
          ⎧ INPUT                          ⎫
OPEN B FOR ⎨ OUTPUT  AS[#]n                 ⎬
          ⎩ APPEND                         ⎭
```
,

где:

- OPEN(«открыть»), FOR(«для»), INPUT(«ввод»), OUTPUT(«вывод»), APPEND(«присоединение») — служебные слова;
- B — строковое выражение, значение которого задаёт имя устройства и имя файла.

Сейчас мы познакомим Вас с именами *устройств*:

- устройство `CRT:` отвечает за вывод информации на *текстовый* экран («CaRet» — «символ»);
- устройство `GRP:` отвечает за вывод информации на экран в *графических* режимах («GRaPhics» — «графика»);
- устройство `LPT:` отвечает за вывод информации на *принтер* («Line PrinT» — «печать строки»);
- устройство `COM:` (только для компьютеров MSX 1) отвечает за движение информации по *локальной* сети («COMmunication» — «передача данных»);
- устройство `MEM:` (только для MSX 2) — это не что иное, как RAM диск («MEMory» — «память»);
- устройство `CAS:` является накопителем на магнитной ленте (*магнитофоном*) («CASsette type» — «кассетная лента»);
- n — арифметическое выражение, целая часть значения которого принадлежит:
  - отрезку [0,6] для файлов на дискете,
  - отрезку [0,15] для файлов на других устройствах, и определяет номер контрольного буфера файла;
- # — необязательный символ.

При выполнении оператора `OPEN` файлу B «назначается» для работы контрольный буфер файла с номером n.

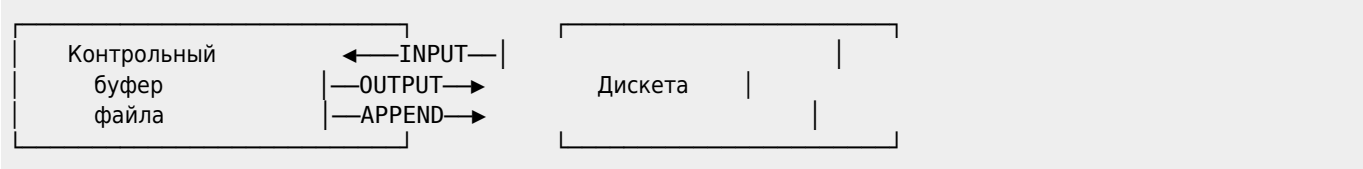
Параметры `INPUT`, OUTPUT и APPEND указывают направление последующего движения информации относительно компьютера.

Оператор `OPEN` с параметром `INPUT` открывает файл *для ввода* строк с дискеты в буфер. Чтение будет организовано от начала файла.

Оператор `OPEN` с параметром `OUTPUT` открывает файл *для вывода* строк с буфера на дискету. Запись будет осуществляться от начала файла. Оператор `OPEN` с параметром `APPEND` открывает файл для вывода строк с буфера на дискету,но запись будет производиться с *конца* файла.

Будьте осторожны: параметр `APPEND` не может использоваться при открытии файлов на устройствах `CAS:`, `CRT:`, `GRP:`, `LPT:` и `COM:` !

Параметр `INPUT` не может использоваться при открытии файлов на устройствах `CRT:`, `GRP:` и `LPT:`



Оператор `OPEN` с параметром `FOR OUTPUT` создаёт новый дисковый файл с заданным именем. Если файл с таким именем уже есть,то он автоматически уничтожается и создаётся новый файл с тем же именем.

Оператор `OPEN…FOR APPEND` обеспечивает поиск названного файла среди уже существующих, с тем чтобы добавить новые записи в его конец, но если требуемого файла найти не удалось, то автоматически создаётся новый файл с заданным именем.

Модификация оператора с параметром `FOR INPUT` реализует поиск данных с конкретным именем, отсутствие которого означает ошибку.

Примеры:

1. `OPEN"CAS:ADDRESS" FOR INPUT AS #1`

Устройство	CAS:
Файл	ADDRESS
Направление	INPUT (файл, сохранённый на ленте, нужно ввести)
Номер файла	1

2. `OPEN"ADDRESS" FOR OUTPUT AS #1`

Устройство: A:	(по умолчанию в MSX Disk BASIC)
Файл	ADDRESS
Направление	OUTPUT (файл должен быть создан на дискете)
Номер файла	1

3. `OPEN"B:ADDRESS" FOR APPEND AS #2`

Устройство	B:
Файл	ADDRESS
Направление	APPEND (добавление к уже существующему файлу)
Номер файла	2

Можно открыть файл с последовательным доступом в режиме записи, используя один номер файла, и в режиме чтения, используя другой. Однако, каждому номеру файла будет при этом соответствовать *свой буфер*, никак не связанный с другими.

Если же Вы используете только один буфер, то для изменения режима доступа к файлу последний необходимо закрыть, а затем вновь открыть. Этот приём особенно удобен, когда Вы хотите использовать в программе в любой момент только один открытый файл (`MAXFILES=1`).

Однажды открытый, но не закрытый файл не может быть открыт повторно.

В противном случае компьютер сообщает: «File already open».

Закрытие файла позволяет Вам использовать буфер с тем же номером для открытия другого файла. Оператор, закрывающий файлы, выглядит так:

```
CLOSE [[#]N1,[#]N2,… ,[#]Nk]
```
,

где:

- `CLOSE` — служебное слово;
- N1, N2,..., Nk — арифметические выражения, целые части значений которых определяют номера контрольных буферов файлов (FCB).Эти значения должны принадлежать отрезку [0,15].

Оператор `CLOSE` без параметров закрывает *все* открытые ранее файлы.

При записи данных оператор `CLOSE` обеспечивает вывод в файл,который был открыт ранее в режиме `FOR OUTPUT`, кода конца файла EOF («End Of File») с кодом ASCII, равным &h1A.

IX.3.2. Операторы PRINT#, PRINT#n, USING. Функции LOF() и LOC()

> Да,— повторял он себе,— если бы на борту «Пилигрима» я знал все, что должен знать настоящий моряк, сколько несчастий можно было бы избежать!
>
> *—Жюль Верн. Дети капитана Гранта*

Для формирования строк файла с последовательным доступом используются операторы `PRINT#` и `PRINT#USING`.

Оператор `PRINT#` записывается в таком виде:

```
          ⎡ ␣┘
PRINT #n,S ⎨  ;
          ⎣  ,
```
,

где:

- `PRINT` («печатать», «вывести») — служебное слово;
- n — арифметическое выражение, целая часть значения которого определяет номер контрольного буфера

файла;
- S — список арифметических и (или) строковых выражений: γ1,γ2,... ,γk;
- ␣ — обозначение символа пробел (" ").

При выполнении оператора PRINT# значение выражения γi (i=1,2,... ,k) последовательно байт за байтом выводится в буфер почти так же, как это делается командой PRINT. Разница состоит лишь в трактовке зонного формата.

Если между двумя выражениями в списке S оператора PRINT# разделителем является *запятая*,то при выводе в буфер соответствующих значений между ними помещается четырнадцать дополнительных пробелов.

Если последним символом в операторе PRINT# является *пробел*, то выполнение оператора PRINT# приводит к формированию полной строки, но вместе с *меткой* её окончания. Иначе создаётся лишь часть строки, а завершается её формирование или последующими операторами PRINT#, или закрытием файла.

Длина строки вместе с концевой меткой не может превысить 256 байтов, а данные выводятся на диск в формате ASCII и только тогда, когда заполняется буфер или закрывается файл.

Для экономного размещения данных на диске и избежания ошибок при их считывании целесообразно список S в операторе PRINT# записывать в виде:

```
                          ⎡ ␣
γ1;";";γ2;";"; ... ;";";γk ⎨
                          ⎣  ;
```

В этом случае значения в создаваемой строке располагаются в *плотном* формате и отделяются друг от друга запятыми.

Будем считать, что информация в одном операторе PRINT# составляет отдельную логическую строку. Строка разбивается на *поля*. Каждому полю логической строки соответствует переменная из списка S.

Так логическая строка, состоящая из трёх полей — фасон одежды S$, цвет одежды C$, цена одежды P — будет выведена в файл оператором

```
         PRINT #1,S$;",";C$;",";P
                  ▲      ▲      ▲
                  │      │      │
Переменные, определяющие поля данной логической строки
```

Если значение какого–либо строкового выражения γi содержит символы «,» и (или) «;», то в качестве разделителей можно использовать символ «;», задаваемый строковой функцией CHR$(34):

```
CHR$(34) γs CHR$(34)
```

Пример 1 [7]. Запись данных в файл последовательного доступа.
0932-01.bas
0932-01.bas

```
10 OPEN "GAMES" FOR OUTPUT AS #1
20 FOR X=1 TO 5
30    INPUT"Введите название игры";F$:PRINT #1,F$
50 NEXT X
60 CLOSE #1
run
Введите название игры? Футбол
Введите название игры? Гольф
Введите название игры? Баскетбол
Введите название игры? Хоккей
Введите название игры? Регби
Ok
```

Если файл с именем «GAMES» уже существовал, то данная программа сотрёт его старое содержимое и будет писать туда заново.

Оператор PRINT#, не оканчивающийся символом «,» или «;» автоматически порождает печать символа CHR$(13) (код клавиши Ввод ⏎) .

Этот символ попадает в буфер и в конце концов в файл, где он служит разделителем между текущей записью и записью, следующей за ней.

Пример 2 [5]. Открывается новый или ранее созданный файл «Щука» для вывода в него
0932-02.bas
0932-02.bas строк от начала. Программная строка 50 позволяет сформировать в буфере одну строку.

```
20 OPEN "Щука" FOR OUTPUT AS#1
30 A=5:B$="трава":C=83.157
50 PRINT #1,A;",";B$;",";C
60 CLOSE #1
```

Пример 3 [5].
Открывается ранее созданный файл «Шука» для вывода в него строк от конца. Затем программная строка 50 формирует очередные строки файла.По мере заполнения буфера данные выводятся на дискету,а за последней строкой ставится метка окончания файла (EOF).
0932-03.bas
0932-03.bas

```
20 OPEN "Щука" FOR APPEND AS#1
30 INPUT"A=";A:INPUT"B$=";B$:INPUT"C=";C
50 PRINT #1,A;",";B$;",";C
55 IF C<>0 THEN 30 ELSE CLOSE #1
```

Пример 4 [5]. Открывается новый или ранее созданный файл «Карась» для вывода в него строк от начала и далее. Двумя командами 50 и 60 формируется строка.
0932-04.bas
0932-04.bas

```
10 CLEAR 1000:MAXFILES=2:OPEN "Карась" FOR OUTPUT AS#2
30 A$="FLOPPY DISK":E=45:F=54.8
50 PRINT #2,A$;",";          '◄── Последний символ ";" !
60 PRINT #2,E;",";F          '◄── Последний символ "пробел" !
70 CLOSE #2
```

Пример 5 [5]. В файл «Карп» от его начала с клавиатуры вводится S элементов числового массива.
0932-05.bas
0932-05.bas

```
10 MAXFILES=5:OPEN "Карп" FOR OUTPUT AS#4
30 INPUT "Количество элементов";S
40 FOR K=1 TO S:INPUT M:PRINT #4,M;",";:NEXT K
50 CLOSE #4
```

Пример 6 [5]. При формировании строки в качестве разделителей используется символ ", задаваемый кодом CHR$(34).
0932-06.bas
0932-06.bas

```
20 OPEN "Сазан" FOR OUTPUT AS#1
30 R=123.789:K$="KRA,,N;W2"
40 I$=CHR$(34)
50 PRINT #1,R;",";I$;K$;I$
60 CLOSE #1
```

Оператор PRINT#,USING записывается в таком виде:

```
PRINT #n,USING T,S
```
,

где: PRINT, USING — служебные слова;

- T — строковое выражение, значение которого определяет формат вывода;
- n — арифметическое выражение, целая часть значения которого определяет номер контрольного буфера файла;
- S — список арифметических и (или) строковых выражений: γ1, γ2,..., γk.

При выполнении данного оператора значения выражений

```
γi    (i=1,2,… ,k)
```

выводятся в соответствии с форматом T в буфер точно так же, как это «делает» «обычный» оператор PRINT USING.

Пример 7.
0932-07.bas
0932-07.bas

```
10 CLEAR 1000:MAXFILES=5:DIM M(200)
20 OPEN "Лещ" FOR OUTPUT AS#4
30 INPUT"Количество элементов";S
40 FOR K=1 TO S:X=INT(RND(1)*3000)/99:PRINT #4,USING "###.##";X:NEXT K
50 CLOSE #4
```

Однако, следует весьма осторожно употреблять префиксы, обозначающие денежные единицы ("$$" , "**" и "**$"), так как они вызывают форматирование числовых величин как нечисловых. По этой причине числовые величины, снабжённые в дисковом файле такими префиксами, не могут быть считаны оттуда в их исходном виде.

Пример 8.
0932-08.bas
0932-08.bas

```
10 OPEN "SAMPLE.DAT" FOR OUTPUT AS #1
20 PRINT #1,USING"$$###.##";99.50:CLOSE#1
40 OPEN "SAMPLE.DAT" FOR INPUT AS#1
50 INPUT #1,A:PRINT"Первое значение в файле:";A:CLOSE#1
run
Первое значение в файле: 0
Ok
```

Пусть открыт файл B, связанный с буфером FCBn.

Функция

```
LOF(n)
```
,

где:

- LOF («Length Of File» — «размер файла») — служебное слово;
- n — арифметическое выражение, целая часть значения которого принадлежит отрезку [0,15] и определяет контрольного буфера файла, соответствующий файлу, возвращает размер файла в байтах.

Функция

```
LOC(n)
```
,

где:

- LOC («LOCation» — «определение позиции») — служебное слово;
- n — арифметическое выражение, целая часть значения которого принадлежит отрезку [0,15] и определяет номер контрольного буфера файла, соответствующий файлу, для файлов последовательного доступа возвращает текущее количество байтов информации, считанных в буфер или записанных на дискету.
    - Или возвращает номер *текущей* номер записи, использованный в последнем выполненном операторе PUT или GET.
    - Для RAM-диска возвращает позицию в файле, начиная с номера файла. Например, LOC возвращает 0 сразу после того, как файл открыт для ввода и вывода (OPEN FOR INPUT / OPEN FOR OUTPUT) и возвращает длину файла сразу после того, как файл открыт для добавления (OPEN FOR APPEND).

IX.3.3. Операторы INPUT, LINE INPUT#n. Функции INPUT$ и EOF

Для чтения строк файла последовательного доступа с дискеты в оперативную память используются операторы INPUT#, LINE INPUT# и функции INPUT$ и EOF.

Оператор

```
LINE INPUT #n,γ
```

,

где:

- LINE, INPUT — служебные слова;
- n — арифметическое выражение, целая часть значения которого определяет номер контрольного буфера файла;
- γ — строковая переменная, позволяет считать очередную строку файла β, связанного с буфером n, и присвоить результат переменной γ.

Оператор LINEINPUT#, считывающий одиночное строковое значение, позволяет решить «проблему запятой».

Пример 1.
0933-01.bas
0933-01.bas

```
10 OPEN "TEST.DAT" FOR OUTPUT AS #1
20 PRINT #1,"Агибалова, Алевтина":CLOSE#1
40 OPEN "TEST.DAT" FOR INPUT AS#1
50 LINE INPUT#1,A$:PRINT"Первое значение в файле:";A$:CLOSE#1
run
Первое значение в файле: Агибалова, Алевтина
Ok
```

Пример 1.
0933-02.bas
0933-02.bas

```
10 CLEAR 1000:MAXFILES=5:DIM M(200)
20 OPEN "LINP" FOR OUTPUT AS#4
30 INPUT"Количество элементов";S
40 FOR K=1 TO S:PRINT #4,K;",";:PRINT LOC(4);:NEXT K:CLOSE #4
70 OPEN "LINP" FOR INPUT AS#4
80 PRINT:PRINT LOF(4)
90 LINE INPUT #4,M$
100 PRINT LOC(4),:PRINT M$
110 IF NOT EOF(4) THEN 90    'Обратите внимание на данную строку!
120 PRINT LOF(4):CLOSE #4
```

Оператор INPUT# записывается в виде:

```
INPUT #n,S
```

где:

- INPUT — служебное слово;
- n — арифметическое выражение, целая часть значения которого определяет номер контрольного буфера файла;
- S — список переменных.

Опишем, как происходит выполнение этого оператора.

Если очередная считываемая строка файла состоит из значений: α1, α2,..., αk, то в соответствующей команде INPUT# список S должен иметь такой вид: β1, β2,..., βk, а каждая переменная βi обязана быть того же типа,что и значение αi. В этом случае выполнение оператора приводит к присваиваниям переменным βi значений αi.

Пример 3. Чтение из файла последовательного доступа.
0933-03.bas
0933-03.bas

```
10 OPEN "GAMES" FOR INPUT AS #2
20 FOR X=1 TO 5:INPUT #2,G$:PRINT G$:NEXT X:CLOSE #2
run
Футбол
Гольф
Баскетбол
Хоккей
Регби
Ok
```

Пример 4.
0933-04.bas
0933-04.bas

```
10 OPEN "SAMPLE.DAT" FOR OUTPUT AS #1
20 PRINT #1,12,567:CLOSE#1
40 OPEN "SAMPLE.DAT" FOR INPUT AS#1
50 INPUT #1,A,B:PRINT"Второе значение в файле:";B:CLOSE#1
run
Второе значение в файле: 567
Ok
```

Пример 5.
0933-05.bas
0933-05.bas

```
10 OPEN "TEST.DAT" FOR OUTPUT AS #1
20 PRINT #1,"Кремний";14;28.0855:CLOSE#1
40 OPEN "TEST.DAT" FOR INPUT AS#1
50 FOR K%=1 TO 3:INPUT#1,A$:PRINT"Значение номер";K%;":";A$:NEXT K%
70 CLOSE#1
run
Значение номер 1: Кремний 14 28.0855
Input past end in 50
Ok
```

При выполнении оператора PRINT# (строка 20) в файл записываются три различных значения, но поскольку в операторе они разделяются символами «;», в файл они будут заноситься без разделителей. Когда оператор INPUT# реализует считывание этих значений (строка 50), они все «сливаются» в одно. Таким образом, в файле не будет существовать ни второго, ни третьего значений, поэтому при попытке чтения программой какой-либо информации за последним значением в файле фиксируется ошибка «Считывание за пределами последнего значения».

Разделителем для строковых данных может быть запятая или символ «Возврат каретки». Те же самые знаки или пробелы служат для разграничения числовых величин. Существует несколько способов гарантированной расстановки разделителей между отдельными значениями. Один из самых простых состоит в том, чтобы присвоить значение «,» строковой переменной и записывать её после каждого значения в операторе PRINT#.

Пример 6.
0933-06.bas
0933-06.bas

```
10 OPEN "TEST.DAT" FOR OUTPUT AS #1
15 D$=",":PRINT#1,"Кремний";D$;14;D$;28.0855:CLOSE#1
40 OPEN "TEST.DAT" FOR INPUT AS#1
50 FOR K%=1 TO 3:INPUT #1,A$:PRINT"Значение номер";K%;":";A$:NEXT K%
70 CLOSE#1
run
Значение номер 1: Кремний
Значение номер 2: 14
Значение номер 3: 28.0855
Ok
```

Отметим, что строковая величина в дисковом файле, могущая содержать запятую, должна быть записана в кавычках; оператор INPUT# обеспечивает устранение этих кавычек при считывании.

Пример 7.
0933-07.bas
0933-07.bas

```
10 OPEN "TEST.DAT" FOR OUTPUT AS #1
20 PRINT#1,CHR$(34);"Манагуа, Никарагуа";CHR$(34):CLOSE#1
40 OPEN "TEST.DAT" FOR INPUT AS#1
50 INPUT#1,A$:PRINT"Первое значение в файле:";A$:CLOSE#1
run
Первое значение в файле: Манагуа, Никарагуа
Ok
```

Пример 8 [7]. *Обновление* файла последовательного доступа.
0933-08.bas
0933-08.bas

```
10 OPEN "GAMES" FOR INPUT AS #1    'Открывается "старый" файл
20 OPEN "GAMES2" FOR OUTPUT AS #2 'Открывается "новый" файл (пустой)
30 FOR X=1 TO 5
40    INPUT #1,F$
50    IF F$="Баскетбол" THEN F$="Водное поло"
60    PRINT #2,F$
70 NEXT X
80 CLOSE 1,2                       'Оба файла закрываются
90 KILL"GAMES"                     'Удаление исходного файла
100 NAME "GAMES2" AS "GAMES"       'Переименование "нового" файла
```

Пример 9.
0933-09.bas
0933-09.bas

```
10 CLEAR 1000:MAXFILES=5:DIM M(200)
20 OPEN "Шар" FOR OUTPUT AS#4:INPUT"Количество элементов";S
40 FOR K=1 TO S
50    X=RND(1):Y=RND(1)
60    X$=CHR$(INT(RND(1)*31)+224):Y$=CHR$(INT(RND(1)*31)+224)
70    PRINT #4,X$+Y$;",";X;",";Y;
```

```
80 NEXT K:CLOSE #4
90 OPEN "Шар" FOR INPUT AS#4
100 INPUT #4,M$,A,B:PRINT M$;A,B
120 IF NOT EOF(4) THEN 100
130 CLOSE #4
run
Количество элементов? 5
ВЯ .59521943994623   .10658628050158
КЩ .73474759503023   .18426812909758
ЫО .63799556899423   .47041117641358
ОВ .18586284343823   .12951037284958
ТМ .02818381196223   .48775999680558
Ok
```

Заметим, что строка 100 в этом примере могла быть заменена на один из фрагментов: a)

```
100 INPUT #4,M$
101 INPUT #4,A,B
```

b)

```
100 INPUT #4,M$
101 INPUT #4,A
102 INPUT #4,B
```

Для чтения очередного фиксированного количества символов из файла без учёта его разбиения на строки используется функция

```
INPUT$(m[,[#]n])
```
,

где:

- INPUT$ — служебное слово;
- m — арифметическое выражение, целая часть значения которого должна принадлежать отрезку [1,255];
- n — номер контрольного буфера файла.

В любой момент работы с файлом функция INPUT$ «читает» INT(m) его очередных символов. Иными словами, последовательность этих символов становится значением функции INPUT$(). При этом метки окончания строк воспринимаются так же, как любые другие символы.

Функция INPUT$ возвращает строку, содержащую заданное количество символов, прочитанных из файла.

Все символы, включая разделители, передаются в программу, за исключением специального символа с кодом ASCII &H1A, который воспринимается как признак конца файла и «прекращает выполнение» функции.

Функция INPUT$ предоставляет возможность последовательно выбирать символы из файла независимо от разделителей строк!

Для полного уяснения семантики функции INPUT$ рекомендуется «прогнать» следующий пример при различных значениях параметров S и X.

*Пример*10 [5].
0933-10.bas
0933-10.bas

```
10 CLEAR 1000:MAXFILES=5:DIM M(200)
20 OPEN "TOP" FOR OUTPUT AS#4
30 INPUT "Количество элементов";S
40 FOR K=1 TO S:PRINT #4,"L";",";"W";",":NEXT K:CLOSE #4
70 OPEN "TOP" FOR INPUT AS#4
```

```
80 INPUT"Длина значения в  INPUT$";X
90 Y=(LOF(4)-1)/X: Z=(LOF(4)-1)MOD X
105 IF Y=0 THEN 130
106 FOR K=1 TO Y
110 X$=INPUT$(X,#4):PRINT X$
115 NEXT
120 IF Z=0 THEN 140
130 PRINT INPUT$(Z,#4)
140 CLOSE #4
```

Заметим, что «длину» файла не всегда можно определить заранее: представьте себе файл адресов, который все время приходится открывать в режиме добавления данных (`FOR APPEND`). Функция `EOF` позволяет Вам решить эту проблему: специальный символ с кодом &h1A добавляется к концу файла, когда этот файл закрывается. Этот символ помечает *конец* файла.

Функция `EOF`, синтаксис которой

```
EOF(n)
```
,

где:

- `EOF` («End Of File» — «конец файла») — служебное слово;
- n — арифметическое выражение, целая часть значения которого определяет номер файла, сравнивает код очередного символа с кодом &h1A и возвращает:
    - -1 («true»), если найден код конца файла, и
    - 0 («false») в противном случае.

Пример 11. Продемонстрируем способ использования функции `EOF()`.
0933-11.bas
0933-11.bas

```
10 OPEN "TESTF" FOR INPUT AS #1
20 IF EOF(1) THEN 60
30 INPUT #1,A:PRINT A:GOTO 20
60 CLOSE #1
```

Ниже приведены 3 схемы, определяющие последовательность операций, которые должны быть выполнены при перемещениях строк файлов *последовательного доступа* из оперативной памяти на дискету и в обратном направлении.

1. Общая схема формирования строк на диске от *начала* файла B:
    - A. В оперативной памяти резервируется буфер (оператором `MAXFILES=`).
    - B. Открывается новый или уже существующий файл B (оператором `OPEN` с параметром `OUTPUT`).
    - C. Формируется требуемое количество строк файла(операторами `PRINT#` или `PRINT#,USING`). На дискету из буфера они сбрасываются автоматически.
    - D. Файл B закрывается.

    Пример 12 [7].
    0933-12.bas
    0933-12.bas

    ```
    10 OPEN "DRESS" FOR OUTPUT AS #1
    20 FOR X=1 TO 5
    30    INPUT"Данные об одежде";S$,C$,P:PRINT #1,S$;",";C$;",";P
    50 NEXT X:CLOSE #1
    run
    Данные об одежде? А35,Красный,68.95
    Данные об одежде?  А36,Голубой,75
    Данные об одежде? Б44,Розовый,105.50
    Данные об одежде? В78,Серый,85
    Данные об одежде? ЖД5,Черный,78.89
    ```

```
Ok
```

Отдельные сегменты программных файлов — это специфические файлы данных последовательного доступа. Поэтому программы или их фрагменты можно загружать в память как данные. Ограничимся здесь лишь двумя примерами.

Пример 13. Выполнение этой программы приводит к считыванию и выводу на экран всех строк программы «Симпсон».
0933-13.bas
0933-13.bas

```
10 'Чтение программного файла
20 OPEN "Симпсон" FOR INPUT AS#1
30 LINE INPUT #1,R$:PRINT R$
50 IF NOT EOF(1) THEN 30
60 CLOSE #1
```

Пример 14. Выполнение программы даёт возможность «просмотреть» в той же самой программе «Симпсон» строки с номерами от 30 и до 70.
0933-14.bas
0933-14.bas

```
10 'Чтение диапазона строк
20 OPEN "Симпсон" FOR INPUT AS#1
30 LINE INPUT #1,R$
40 X$=MID$(R$,1,2)
50 IF X$<"30" OR X$>"70" THEN 80
60 PRINT R$
80 IF NOT EOF(1) THEN 30
90 CLOSE #1
```

2. Общая схема пополнения файла B строками с его *конца*:
    - A. В оперативной памяти резервируется буфер (оператором MAXFILES=).
    - B. Открывается уже открытый файл B (оператором OPEN с параметром APPEND).
    - C. Формируется требуемое количество строк файла(операторами PRINT# или `PRINT#,USING`). На дискету из буфера они сбрасываются автоматически.
    - D. Файл B закрывается.

Пример 15 [7]. Добавление в файл спортивной информации названия двух новых видов спорта.
0933-15.bas
0933-15.bas

```
10 OPEN "GAMES" FOR APPEND AS#1
20 PRINT #1,"Пинг-понг":PRINT #1,"Теннис":CLOSE #1
```

3. Общая схема *считывания* строк с диска от начала файла B
    - A. В оперативной памяти резервируется буфер (оператором `MAXFILES`)
    - B. Открывается уже существующий файл B (оператором `OPEN` с параметром `INPUT`)
    - C. Считывается требуемое количество строк файла(операторами `INPUT$` или `LINEINPUT$`, или `INPUT$`)
    - D. Файл B закрывается

Пример 16 [7].
0933-16.bas
0933-16.bas

```
10 OPEN "DRESS" FOR INPUT AS#2
20 FOR X=1 TO 5
30    INPUT #2,S$,C$,P:PRINT S$,C$,P
50 NEXT X:CLOSE #2
run
A35  Красный  68.95
A36    Голубой  75
Б44  Розовый  105.50
B78  Серый   85
ЖД5  Черный   78.89
Ok
```

IX.3.4. Примеры

1.

Гидрометцентр ведёт статистику выпадения снега по регионам, для каждого из которых заведён файл последовательного доступа.Во всех файлах имеется по три элемента данных: имя метеоролога, название региона, количество выпавшего за зиму снега в мм.

- Напишите программу ввода данных; заполните файлы для трёх регионов.
- Просмотрите все три файла и подсчитайте средний уровень снежных осадков по трём регионам. Результат выведите на экран.

0934-01.bas
0934-01.bas

```
10 MAXFILES=6:OPEN "region1" FOR OUTPUT AS #1
21 OPEN "region2" FOR OUTPUT AS #2:OPEN "region3" FOR OUTPUT AS #3
30 FOR I=1 TO 3
31    PRINT "region";I
35    INPUT "Введите количество работающих метеорологов на станции";N
36 FOR E=1 TO N
40 INPUT "Введите:имя метеоролога";I$:INPUT "название региона";R$:INPUT "количество выпавшего за зиму снега в мм";K
50  PRINT #I,I$;",";R$;",";K
55  NEXT E
60  CLOSE #I
70 NEXT I:CLS
71 OPEN "region1" FOR INPUT AS #1:OPEN "region2" FOR INPUT AS #2
73 OPEN "region3" FOR INPUT AS #3
80 FOR O=1 TO 3
90    S=0
100   FOR E=1 TO N:INPUT #O,I$,R$,K:S=S+K:NEXT E
140   S=S/N:PRINT"Средний уровень снежных осадков по ";R$;" региону";S
150   CLOSE #O
160 NEXT O
170 END
```

2.

Хоккейные команды «Чёрные ястребы» и «Красные крылья» хранят в файлах последовательного доступа имена всех своих 12 нападающих, число заброшенных ими шайб, сделанных голевых передач и заработанное штрафное время.

- Создайте файлы «BLACK» и «RED», содержащие информацию о каждой из двух команд.
- Ваша программа по данным, извлечённым из этих файлов, должна порождать новый файл «ALLSTARS», в котором содержались бы имя, команда и сумма очков (голы + передачи) для шести лучших игроков обеих команд. Выведите имена и показатели результативности хоккеистов на экран дисплея.

] 0934-02.bas
0934-02.bas

```
10 MAXFILES=3
20 OPEN "BLACK" FOR OUTPUT AS #1:OPEN "RED" FOR OUTPUT AS #2
25 FOR K=1 TO 2
30     FOR X=1 TO 6 'В каждой команде - 6 игроков
40        PRINT "Данные о команде";K;":"
50          INPUT "Имя нападающего";I$:INPUT "Число заброшенных шайб";F:INPUT "Сделанных голевых передач";G:INPUT "Заработанное штрафное время";W
60        PRINT #K,I$;",";F;",";G;",";W
70     NEXT X
80     CLOSE #K
81 NEXT K
90 OPEN "BLACK" FOR INPUT AS #1:OPEN "RED" FOR INPUT AS #2
100 OPEN "ALLSTARS" FOR OUTPUT AS #3
110 DIM MI$(12):DIM MK$(12):DIM MO(12)
115 I=1
120 FOR K=1 TO 2
130     FOR X=1 TO 6
140        INPUT #K,I$,F,G,W
145        IF K=1 THEN A$="BLACK" ELSE A$="RED"
150        O=F+G:MI$(I)=I$:MK$(I)=A$:MO(I)=O:I=I+1
```

```
155     NEXT X
156     CLOSE #K
157 NEXT K
165 FOR I=1 TO 11
166     L=I
170     FOR J=I+1 TO 12
175        IF MO(L)<MO(J) THEN L=J
176     NEXT J
180     SWAP MO(I),MO(L):SWAP MI$(I),MI$(L):SWAP MK$(I),MK$(L)
185 NEXT I
190 FOR I=1 TO 6
194     H$=MI$(I):D$=MK$(I):V=MO(I):PRINT #3,H$;",";D$;",";V
200 NEXT I
210 CLOSE #3
220 OPEN "ALLSTARS" FOR INPUT AS#3
230 FOR N=1 TO 6
240    INPUT #3,H$,D$,V
250    PRINT "Фамилия игрока",H$:PRINT "Команда",D$:PRINT "Очки",V
280 NEXT N:CLOSE #3
```

3.

Помогите декану факультета организовать файл академических занятий студентов, для чего:

- создайте файл последовательного доступа и заполните его именами студентов, названиями академических курсов и оценками;при этом воспользуйтесь следующими данными:

Имя студента	Академический курс	Отметка
Сергей	Французский язык	5
Дмитрий	Английский язык	4
Олег	Математика	5
Борис	Французский язык	3
Михаил	Английский язык	3
Александр	Математика	5
Тимофей	Математика	4

- выберите «умных» студентов, т.е. тех, кто имеет оценку выше 4,и запишите сведения о них в файл «SMART»;
- пусть программа помогает декану формировать на основе этого файла группы углублённого обучения.По названию курса она должна выдавать список «умных» студентов, зачисленных в такую группу.

0934-03.bas
0934-03.bas

```
10 MAXFILES=2:INPUT "Количество студентов";N
20 OPEN "СПИСОК" FOR OUTPUT AS #1:OPEN "SMART" FOR OUTPUT AS #2
30 FOR X=1 TO N
50 INPUT"Ф.И. студента:";I$:INPUT"Название курса:";K$:INPUT"Оценка:";O
60 PRINT #1,I$;",";K$;",";O
70 NEXT X:CLOSE #1
90 OPEN "СПИСОК" FOR INPUT AS #1
130 FOR X=1 TO N
140    INPUT #1,I$,K$,O:IF O>=4 THEN D=D+1:PRINT #2,I$;",";K$
155 NEXT X:CLOSE #1:CLOSE #2
220 OPEN "SMART" FOR INPUT AS#2
225 INPUT "Введите название курса";E$
230 FOR I=1 TO D
240    INPUT #2,I$,K$:IF K$=E$ THEN PRINT I$
280 NEXT I:CLOSE #2
290 PRINT "Хотите закончить Д/Н"
300 IF INPUT$(1)="Д" THEN END ELSE GOTO 220
```

IX.3.5. Вывод файлов данных на экран и принтер

Текстовые экраны 0 и 1, графические экраны, а также принтер могут быть объявлены устройствами вывода файлов последовательного доступа.

Открытие таких файлов осуществляется оператором

```
     ⎧ "CRT:"                                  ,
OPEN ⎨ "GRP:" FOR OUTPUT AS[#]n
     ⎩ "LPT:"
```

где:

- OPEN, FOR OUTPUT, AS — служебные слова;
- n — арифметическое выражение, целая часть значения которого определяет номер контрольного буфера файла;
- CRT:, GRP:, LPT: — имена устройств.

При выполнении оператора OPEN в качестве устройства вывода файлов назначаются: *текстовые* экраны (CRT:), *графические* экраны (GRP:), *принтер* (LPT:).

Вывод строк в любом случае осуществляется операторами PRINT# и PRINT#,USING.

Поговорим о выводе *программных* файлов на экран и принтер. Используя устройства CRT:, GRP: и LPT:, можно выводить текст программы на экран и принтер.

1. Команда SAVE "LPT:" предназначена для вывода текста программы на *принтер*.
2. Команда SAVE "CRT:" позволяет выводить текст программы на *текстовый* экран.
3. Команда SAVE "GRP:" позволяет выводить текст программы на *графический* экран.

Использование «графического» оператора SAVE "GRP:" в текстовом режиме приводит к сообщению об ошибке:

«Illegal function call».

Вполне естественно, что команды LOAD, RUN и MERGE не могут «работать» с устройствами CRT:, GRP: и LPT: !

Примеры:

1. 0935-01.bas
   0935-01.bas

   ```
   10 COLOR 1,11,7:SCREEN 2:OPEN "GRP:" FOR OUTPUT AS#1
   30 PSET (69,80):PRINT #1,"Текст на графическом экране  SCREEN 2"
   50 GOTO 50
   ```

2. 0935-02.bas
   0935-02.bas

   ```
   10 COLOR 1,11,7:SCREEN 3:OPEN "GRP:" FOR OUTPUT AS#1
   30 PSET (50,80):PRINT #1,"Текст на графическом экране  SCREEN 3"
   50 GOTO 50
   ```

3. Fix Me!

   Программа «777», записанная на диске в формате ASCII, выводится на графический экран:
   0935-03.bas
   0935-03.bas

   ```
   10 COLOR 1,15,8:SCREEN 3:PSET(0,10),15:MAXFILES=2
   30 OPEN "GRP:" FOR OUTPUT AS#1:OPEN "777" FOR OUTPUT AS#2
   50 LINE INPUT #2,R$:PRINT #1,R$
   80 IF NOT EOF(2) THEN 50
   90 CLOSE:GOTO 90
   ```

4. Fix Me!

Файл данных последовательного доступа, записанный на дискете под именем «КАРП», выводится на принтер.
0935-04.bas
0935-04.bas

```
15 MAXFILES=2
18 OPEN "Карп" FOR OUTPUT AS#1:OPEN "LPT:" FOR OUTPUT AS#2
30 INPUT #1,M:PRINT #2,M;
35 IF NOT EOF(1) THEN 30
```

Пример 5 [76]. Простейший *графический* редактор.

При формировании изображения Вам могут понадобиться такие команды:

- L При нажатии этой клавиши определяется начальная точка прямой. При повторном нажатии вычерчивается прямая.
- P Закрашивается участок экрана.
- U Отменяется последняя команда.
- DEL Уничтожается программа, запущенная из памяти, а экран очищается.
- ESC Генерируется программа.

После окончания формирования изображения текст программы на языке MSX BASIC можно вызвать на экран, либо записать на ленту или дискету (по команде SAVE. При необходимости программу с ленты или с дискеты можно вновь загрузить (командой LOAD) или «слить» с другой программой, хранящейся в памяти (командой MERGE).

Один из спрайтов задействуется как курсор, другой — в качестве маркера, отмечающего начало прямой, и ещё шесть в качестве «координатных», чтобы отмечать текущие координаты курсора на экране. Когда курсор «касается» одного из «координатных» спрайтов, они пропадают. Это выполняется с использованием прерывания по ON SPRITE GOSUB.
Fix Me! 0935-05.bas
0935-05.bas

```
10 GOTO 210
30 'Изображение дополнительных координат
40 '      (подпрограмма)
60 PUT SPRITE 0,(16,P),1,H
70 PUT SPRITE 1,(22,P),1,T
80 PUT SPRITE 2,(28,P),1,U
90 PUT SPRITE 3,(16,P+8),1,H1
100 PUT SPRITE 4,(22,P+8),1,T1
110 PUT SPRITE 5,(28,P+8),1,U1
120 RETURN
130 ' О с н о в н а я  программа
210 SCREEN 0:KEY OFF:WIDTH 38
220 DEFINT A-Z
230 MAXFILES=2
240 DIM BUF(100,5)
260 ' Открытие файла
280 PRINT"ЭКРАННЫЙ ГЕНЕРАТОР"
290 PRINT
300 PRINT"Включите на всякий случай принтер!"
310 PRINT:PRINT
320 PRINT"Введите имя файла:";
330 A$=INPUT$(1)
340 IF A$=CHR$(13) AND F$<>""THEN 390
350  IF  A$=CHR$(8)  AND  POS(0)-1>16  THEN  LOCATE  POS(0)-1:PRINT  CHR$(32);:LOCATE
POS(0)-1:F$=LEFT$(F$,LEN(F$)-1):GOTO 330
360 IF LEN(F$)=6 THEN GOTO 330
370 IF (A$>="A"AND A$<="Z")OR (A$>="a" AND A$<="z")THEN F$=F$+A$ ELSE GOTO 330
380 PRINT A$;:GOTO 330
390 PRINT:PRINT
400 PRINTUSING"Открывается файл:&";F$
410 PRINT:PRINT
420 OPEN F$ FOR OUTPUT AS#2
```

```
430 PRINT USING"Файл'&'Открыт";F$
440 PRINT:PRINT
450 PRINT"Для продолжения нажмите любую клавишу"
460 A$=INPUT$(1)
470 SCREEN 2,0,0
480 GOSUB 800
490 IX=-1:P=8:Q=168:X=128:Y=96
500 H=1:T=2:U=8:H1=0:T1=9:U1=6
510 MX=1:MY=1:HX=8:HY=8:LX=0:LY=0
520 ON SPRITE GOSUB 1780:SPRITE ON
530 PUT SPRITE 6,(X,Y),1,10
540 GOSUB 60
550 A$=INKEY$:IF A$=""THEN 550
560 IF A$=CHR$(127)THEN IX=-1:CLS
570 IF A$=CHR$(27)THEN GOTO 1320
580 IF A$="P"THEN GOSUB 910
590 IF A$="L"THEN GOSUB 1000
600 IF A$="U"THEN GOSUB 1160
610 D=STICK(0):IF D=0 THEN 550
620 IF D=1 THEN Y=Y-MY
630 IF D=2 THEN Y=Y-MY:X=X+MX
640 IF D=3 THEN X=X+MX
650 IF D=4 THEN X=X+MX:Y=Y+MY
660 IF D=5 THEN Y=Y+MY
670 IF D=6 THEN Y=Y+MY:X=X-MX
680 IF D=7 THEN X=X-MX
690 IF D=8 THEN X=X-MX:Y=Y-MY
700 IF X>255 THEN X=255
710 IF Y>191 THEN Y=191
720 IF X<0 THEN X=0
730 IF Y<0 THEN Y=0
740 H=INT(X/100):T=(XMOD100)/10:U=X MOD10
750 H1=INT(Y/100):T1=(YMOD100)/10:U1=Y MOD10
760 GOTO 530
780 'О п и с а н и е  спрайтов
800 FOR I=0 TO 11
810    S$=""
820    FOR J=1 TO 8
830       READ A$:S$=S$+CHR$(VAL("&h"+A$))
840    NEXT
850    SPRITE$(I)=S$
860 NEXT
870 RETURN
890 'Подпрограмма закраски
910 IF POINT(X,Y)=15 THEN RETURN
920 IF IX+1=500 THEN RETURN
930 IX=IX+1
940 PAINT(X,Y)
950 BUF(IX,0)=1:BUF(IX,1)=X:BUF(IX,2)=Y
960 RETURN
980 'Изображение  п р я м о й
1000 IF L=0 THEN 1070
1010 L=0
1020 LINE(X,Y)-(BUF(IX,1),BUF(IX,2))
1030 IF(X=BUF(IX,1))AND(Y=BUF(IX,2))THEN BUF(IX,0)=2:PUT SPRITE 7,(0,209):SPRITE ON:RETURN
1040 BUF(IX,3)=X:BUF(IX,4)=Y
1050 PUT SPRITE 7,(0,209)
1060 BUF(IX,0)=3:SPRITE ON:RETURN
1070 IF IX+1=100 THEN RETURN
1080 IX=IX+1
1090 L=1:SPRITE OFF
1100 BUF(IX,1)=X:BUF(IX,2)=Y
1110 PUT SPRITE 7,(X-1,Y-2),1,11
1120 RETURN
```

```
1140 'Отмена предыдущего действия
1160 IX=IX-1
1170 CLS
1180 IF IX<=-1 THEN IX=-1:BUF(0,0)=0:RETURN
1190 FOR I=0 TO IX
1200     ON BUF(I,0) GOSUB 1230,1250,1270
1210 NEXT
1220 RETURN
1230     PAINT(BUF(I,1),BUF(I,2))
1240     RETURN
1250          PSET(BUF(I,1),BUF(I,2))
1260          RETURN
1270               LINE(BUF(I,1),BUF(I,2))-(BUF(I,3),BUF(I,4))
1280               RETURN
1300 'Генерация программы
1320 SPRITE OFF:SCREEN 0'В оригинале оператор SPRITE OFF отсутствует!
1330 PRINT"Вывод на принтер(P)"
1340 PRINT"Вывод на экран(S)"
1350 PRINT:PRINT
1360 PRINT"Выбирайте скорее режим:"
1370 LOCATE 23,4
1382 A$=INPUT$(1)
1390 PRINT A$
1400 IF A$<>"P"ANDA$<>"S" THEN BEEP:GOTO 1370
1410 IF A$="P"THEN OPEN"LPT:"AS#1
1420 IF A$="S"THEN OPEN"CRT:"AS#1
1430 CLS
1450 'З а п и с ь  п р о г р а м м ы
1470 FOR D=1 TO 2
1480    PRINT#D,"10 REM*"
1490    PRINT#D,USING"20 REM*Имя программы:&";F$
1500    PRINT#D,"30 REM*"
1510    PRINT#D,"40 SCREEN 2"
1520    LN=50
1530    FOR I=0 TO IX
1540       PRINT#D,MID$(STR$(LN),2);SPC(1)
1550       ON BUF(I,0)GOSUB 1610,1650,1690
1560       LN=LN+10
1570    NEXT
1580    PRINT#D,MID$(STR$(LN),2);SPC(1);"GOTO";MID$(STR$(LN),2)
1590 NEXT
1600 END
1610 T$=MID$(STR$(BUF(I,1)),2)
1620 U$=MID$(STR$(BUF(I,2)),2)
1630 PRINT#D,USING"PAINT(&,&)";T$;U$
1640 RETURN
1650 T$=MID$(STR$(BUF(I,1)),2)
1660 U$=MID$(STR$(BUF(I,2)),2)
1670 PRINT#D,USING"PSET(&,&)";T$;U$
1680 RETURN
1690 T$=MID$(STR$(BUF(I,1)),2)
1700 U$=MID$(STR$(BUF(I,2)),2)
1710 V$=MID$(STR$(BUF(I,3)),2)
1720 W$=MID$(STR$(BUF(I,4)),2)
1730 PRINT#D,USING"LINE(&,&)-(&,&)";T$;U$,V$,W$
1740 RETURN
1760 'Подпрограмма обработка прерываний
1780 SPRITE OFF
1790 SWAP P,Q
1800 GOSUB 60
1810 FOR D=1 TO 100:NEXT
1820 SPRITE ON
1830 RETURN
1840 DATA 70,88,98,A8,C8,88,70,0
```

```
1850 DATA 20,60,20,20,20,20,70,0
1860 DATA 70,88,08,30,40,80,F8,0
1870 DATA F8,08,10,30,08,88,70,0
1880 DATA 10,20,50,90,F8,10,10,0
1890 DATA F8,80,F0,08,08,88,70,0
1900 DATA 38,40,80,F0,88,88,70,0
1910 DATA F8,08,10,20,40,40,40,0
1920 DATA 70,88,88,70,88,88,70,0
1930 DATA 70,88,88,78,08,10,E0,0
1940 DATA 80,40,20,10, 8, 4, 2,1
1950 DATA 40,E0,40, 0, 0, 0, 0,0
```

IX.4. Использование RAM–диска (только для компьютеров MSX 2)

> RAMdisk (псевдодиск) — логическое устройство, обеспечивающее хранение файлов в специально выделенной области оперативной памяти.Используется на микроЭВМ и ПЭВМ.
>
> *—Англо–русский словарь по программированию и информатике*

Свободное пространство памяти, размер которого можно определить при помощи функций `FRE(0)` и `FRE("")`, автоматически распределяется интерпретатором MSX BASIC. Когда программа слишком длинна или использует слишком много переменных, для временного хранения данных следует использовать либо внешние устройства (магнитные ленты, дискеты), либо дополнительную область памяти MSX–компьютера (так называемый RAM *диск*).

Эта дополнительная область не распределяется *автоматически* т.е. если программа, например, использует слишком много переменных, это ещё не значит, что значения этих переменных будут автоматически перенесены в дополнительную область оперативной памяти!

Вы сейчас убедитесь в том, что доступ к этой области осуществляется аналогично доступу к *дисководам* !

Оператор

```
CALL MEMINI [(R)]
```

,

где:

- `CALL` («вызвать»), `MEMINI` («MEMory INIcialisation» — «инициализация памяти») — служебные слова;
- R — арифметическое выражение, целая часть значения которого принимает значения от 6655 до M-32769, где M — максимальный объем RAM, инициализирует RAM диск и сообщает его текущий свободный объем. Этот оператор должен быть выполнен до первой попытки обращения к RAM диску.

Помните, что это обращение *уничтожает* все файлы, записанные до этого на RAM диск.

Отметим, что для MSX–компьютеров с общим объёмом RAM в 128 Кбайт значение параметра R не должно превышать 98303 байт.

Фактический же размер RAM диска (в байтах) равен значению арифметического выражения R-6399 .

Если значение параметра R меньше указанного минимального значения, то RAM диск «отключается» и компьютер выдаёт сообщение:

«No RAM disk»

, если же значение параметра R больше максимального значения, то появляется сообщение о неправильном вызове функции

«Illegal function call».

Когда параметр R опущен, то его значение по умолчанию устанавливается равным максимальному, т.е. 98303-6399 = 91904 ≈ 92 Кбайта.

Параметр R необходим, если Вы хотите использовать дополнительные области RAM как для временного хранения файлов, так и для других целей.

Если же это не так, то используйте команду

```
CALL MEMINI
```
.

Имена файлов, находящихся на RAM диске, должны включать:

1. имя устройства «MEM:»,
2. имя файла, состоящее не более чем из 8 символов (причём символы «*» и «?» не могут использоваться!), и
3. расширение имени файла, состоящее из точки и не более чем из 3–х символов. Впрочем, расширение имени файла не обязательно; если оно пропущено, то точка также может быть опущена.

Например:

```
MEM:PROGRAM1.BAS
MEM:HRU
```

Команды для работы с RAM диском очень похожи на команды `FILES`, `KILL` и `NAME` версии MSX Disk BASIC.

1) Команда

```
CALL MFILES
```
,

где: `CALL`, `MFILES` — служебные слова; выводит на экран список файлов, записанных на RAM диске, и размер оставшегося свободного пространства.

2) Команда

```
CALL MKILL ("Имя файла")
```
,

где: `CALL`, `MKILL` — служебные слова; «стирает» файл с указанным именем с RAM диска.

Обратите внимание на то, что параметр "Имя файла" заключается в круглые скобки!

3) Команда

```
CALL MNAME ("Имя старого файла" AS "Имя нового файла") ,
```

где:

- `CALL`, `MNAME` — служебные слова;
- параметр "Имя старого файла" обязательно должен присутствовать, параметр "Имя нового файла" может быть опущен; позволяет изменить имя файла.

Перечислим теперь операторы и функции, которые можно использовать для работы с программными файлами и с файлами *последовательного* доступа:

Fix Me!

- SAVE
- LOAD
- MERGE
- RUN
- OPEN
- CLOSE
- PRINT#
- PRINT#USING
- INPUT#
- INPUT$
- LINEINPUT#
- EOF
- LOC
- LOF
- FPOS

Помните, однако, что имя устройства MEM: обязано присутствовать всякий раз при упоминании имени файла!

Приведём несколько примеров записи операторов для работы с RAM диском:

```
MERGE"MEM:файл"
OPEN "MEM:файл" FOR INPUT  AS#1
OPEN "MEM:файл" FOR OUTPUT AS#6
OPEN "MEM:файл" FOR APPEND AS#15
SAVE"MEM:файл"
LOAD"MEM:файл"
RUN"MEM:файл"
```

Пример.
094-01.bas
094-01.bas

```
CALL MEMINI
Ok
10 OPEN "MEM:GAMES" FOR OUTPUT AS#1
20 FOR X=1 TO 3
30    INPUT"Введите название игры";F$:PRINT #1,F$
50 NEXT X:CLOSE #1
60 PRINT"*****************"
80 OPEN "MEM:GAMES" FOR INPUT AS#1
90  FOR X=1 TO 3
100    INPUT #1,F$:PRINT F$
110 NEXT X
120 CLOSE #1:SAVE"MEM:GAMES"
run
Введите название игры? Футбол
Введите название игры? Шашки
Введите название игры? Хоккей
*****************
Футбол
Шашки
Хоккей
Ok
```

Команда SAVE всегда записывает программу в формате ASCII, после чего она «готова к употреблению» операторами MERGE и RUN.

Отметим, что на RAM диске может существовать один файл с *пустым* именем. Таким образом, синтаксически

допустимы следующие конструкции:

```
SAVE"MEM:"

LOAD"MEM:"

RUN"MEM:"

MERGE"MEM:"

OPEN"MEM:"
```

Функция

```
FPOS(Номер файла)
```

возвращает оставшийся объем RAM диска.

Все остальные операторы и функции, перечисленные выше,используются аналогично соответствующим конструкциям,используемым в версии MSX Disk BASIC.

IX.5. Файлы на магнитной ленте

Вывод программных файлов и файлов данных из оперативной памяти на магнитную ленту и обратное считывание с ленты в память реализуется через устройство, называемое *магнитофоном*. Для идентификации файлам присваиваются *имена*, конструируемые из символов алфавита языка.

Длина имени не может превосходить 6 символов (операторы SAVE, CSAVE) или 11 (оператор BSAVE).

Уникальность имён не обязательна.

Работа с магнитофоном может быть организована в рамках любой версии языка MSX BASIC. Для этих целей предусмотрены приведённые ниже команды и функции.

Для управления мотором магнитофона есть специальная команда [92]:

```
MOTOR
```

Для включения мотора используется параметр ON:

```
MOTOR ON
```

,

для выключения OFF:

```
MOTOR OFF
```

Команда без периметров переключает значение ON/OFF.

SAVE	команда записи в формате ASCII программ из оперативной памяти на магнитную ленту
LOAD	команда чтения в оперативную память программ, записанных на магнитной ленте в формате ASCII
MERGE	команда подгрузки в память программ, записанных на ленте в формате ASCII
RUN	команда загрузки в память программ, записанных на ленте в формате ASCII, с автоматическим запуском их на счёт
CSAVE	команда записи во внутреннем представлении программ (см. Приложение 2 — 2.2. Внутренние коды служебных слов) из оперативной памяти на ленту

CLOAD	команда чтения в оперативную память (или верификации) программ, записанных на ленте во внутреннем представлении (см. Приложение 2 — 2.2. Внутренние коды служебных слов)
BSAVE	команда записи во внутреннем представлении программных файлов или файлов данных из конкретного участка оперативной памяти на ленту
BLOAD	команда считывания в определённую область оперативной памяти программных файлов и файлов данных, записанных на ленте во внутреннем представлении (см. Приложение 2 — 2.2. Внутренние коды служебных слов)
OPEN	операторы и функции для работы с файлами данных последовательного доступа
CLOSE	
MAXFILES=	
EOF()	
PRINT#	
PRINT#USING	
INPUT#	
LINE INPUT#	
INPUT$()	

Приступим к краткому описанию синтаксиса и семантики упомянутых выше средств организации файлов на магнитной ленте и работы с ними.

IX.5.1. Работа с программными файлами

В этом пункте описаны команды SAVE, LOAD, MERGE, RUN.

- A.

  Команда

  ```
  SAVE α [,A]
  ```
  ,

  где:

  - SAVE — служебное слово;
  - α — строковое выражение;

  записывает в формате ASCII на магнитную ленту программу из оперативной памяти.

  Имя программы формируется из первых 6 символов значения строкового выражения α. Оно должно начинаться префиксом "CAS:", который в этом случае рассматривается как *имя устройства* и не учитывается при подсчёте длины α, например:

  ```
  SAVE"CAS:COCHA"
  ```
  ,

  ```
  SAVE"CAS:анкета",A
  ```
  .

  Заметим, что имя программы может быть опущено.

  Например:

  ```
  SAVE"CAS:"
  ```

- B. Выполнение команды

  ```
  LOAD α [,R]
  ```
  ,

  где:

- LOAD — служебное слово;
- α — строковое выражение, значение которого определяет имя файла;

наличие параметра R приводит к автоматическому запуску загруженной программы на счёт, начинается с закрытия всех файлов и чистки оперативной памяти от программ и данных. Далее программа, записанная на ленте в формате ASCII под именем, равным значению строкового выражения α, считывается в память.

Например:

```
LOAD"CAS:труба"
```
,

```
LOAD"CAS:King",R
```
.

Поиск конкретной программы на ленте сопровождается индикацией сообщений обо всех «попавшихся под руку» при этом файлах с другими именами, записанных в формате ASCII.

Приведём пример сообщения на экране дисплея при выполнении команды LOAD(учтите, что: «to skip» — «перескочить», «to found» — «находить»):

```
Skip:PRIM1
Skip:       ◄— встретилась программа без имени

Found:PRIM3
Found:      ◄— найдена и загружается программа без имени
```

Команда

```
LOAD"CAS:" [,R]
```

считывает в память первую встреченную на ленте программу в формате ASCII.

- C.

```
RUN α
```

Эта команда, если в ней указаны имя устройства и имя программы, загружает и запускает на выполнение программу, сохранённую в формате ASCII. Она похожа на команду LOAD (с параметром R), за тем исключением, что она загружает программы только в формате ASCII и закрывает все открытые файлы.

В обоих случаях перед загрузкой программы автоматически выполняется команда NEW.

Пример. Команда RUN«CAS:PROG1» загружает и запускает программу с именем «PROG1», сохранённую на магнитной ленте командой

```
SAVE"CAS:PROG1"
```

- D. При выполнении команды

```
MERGE α
```
,

где:

- MERGE — служебное слово;
- α — строковое выражение, значение которого определяет имя файла, закрываются файлы и организуется «чистка» оперативной памяти от данных.

Затем, программа, записанная на ленте в формате ASCII под именем, равным значению строкового выражения α, «сливается» с программой β, находящейся в памяти. При этом, если в α и β совпадают некоторые номера строк,то сохраняются лишь соответствующие строки загружаемой программы α. Например:

```
MERGE"CAS:Мама!"
```

Выдача сообщений на экране при поиске программы α здесь такая же,как и в случае с командой LOAD.

Команда

```
MERGE"CAS:"
```

подгружает в память первую встреченную на ленте в формате ASCII программу.

Обратите внимание на тот факт,что программы, вызываемые командой MERGE, имеют более высокий приоритет, по сравнению с программами, находящимися в памяти компьютера. Поэтому если в объединяемых программах имеются строки с одинаковыми номерами, то строки программы в памяти компьютера будут заменены соответствующими строками «добавляемой» программы.

- A.

  Команда (оператор) CSAVE «сохраняет»программу, записывая её на кассету магнитной ленты. Формат оператора:

  ```
  CSAVE β;[S]
  ```
  ,

  где:

  - CSAVE («Casette SAVE» — «запись на кассету») — служебное слово;
  - β — строковое выражение, значение которого определяет имя файла, в имени программы допускается использование любых символов, причём значащими в имени являются только первые шесть символов;
  - S — арифметическое выражение, целая часть значения которого равна 1 или 2 и определяет скорость записи данных (по умолчанию значение параметра S равно 1):
    - 1 — 1200 бод (1200 бит/с)
    - 2 — 2400 бод (2400 бит/с)

  Скорость 2400 бод требует использования хорошего магнитофона и высококачественной ленты.Скорость работы по умолчанию может быть переопределена в операторе SCREEN (см.раздел V.7.).

  Команда CSAVE «сохраняет» программу во *внутреннем* формате , т.е. в том виде, в каком она хранится в памяти компьютера (см. Приложение 2 — 2.2. Внутренние коды служебных слов).

  Примеры:

  - α) команда

    ```
    CSAVE"PROG1"
    ```

    сохраняет файл, имя которого «PROG1», во внутреннем формате (см. Приложение 2 — 2.2. Внутренние коды служебных слов) со скоростью 1200 бод (или со скоростью, определённой оператором SCREEN);

  - β) команда

    ```
    CSAVE"PROG2",2
    ```

    сохраняет файл с именем «PROG2» на кассетной ленте во внутреннем формате (см. Приложение 2 — 2.2. Внутренние коды служебных слов) со скоростью 2400 бод.

- B. Команда

  ```
  CLOAD β
  ```
  ,

  где:

  - CLOAD («Cassette LOAD» — «загрузка с кассеты») — служебное слово;
  - β — строковое выражение, значение которого определяет имя файла, позволяет загружать программу, сохранённую на кассетной ленте во внутреннем формате (с помощью команды CSAVE). Загрузка выполняется на той же скорости, на которой программа была сохранена. Таким образом, Вам не нужно повторно указывать скорость.

  Если имя программы не указано в операторе, то будет загружена первая же программа, сохранённая во внутреннем формате (см. Приложение 2 — 2.2. Внутренние коды служебных слов).

  Если Вы указываете имя программы,то следите за правильностью ввода имени (т.е., например, если Вы использовали

прописные буквы при написании имени, Вы должны ввести имя программы также прописными буквами).

Заметим, что оператор CLOAD автоматически выполняет команду NEW перед началом загрузки в память найденной программы.

- C. Оператор CLOAD? проверяет правильность сохранения программы на кассетной ленте.После сохранения программы перемотайте ленту назад и введите команду

  ```
  CLOAD? β
  ```
  ,

  где:

  - CLOAD? («Casette LOAD verify» — «загрузка с ленты») — служебное слово;
  - β — строковое выражение, значение которого определяет имя файла.

  Записанная программа, имя которой указано в операторе, сравнивается байт за байтом с текстом программы в памяти компьютера. При обнаружении несовпадения эта процедура прекращается и на экране появляется сообщение об ошибке:

  «Verify error» (*«Ошибка верификации»*).

  Имя программы может быть опущено. В этом случае с содержимым памяти будет сравниваться первая введённая с ленты программа.

  Если имя программы не совпадает с именем, указанным в операторе, поиск программы с указанным именем будет продолжен. На экране дисплея появится следующее сообщение:

  ```
  Skip:имя программы
  ```

  В нем указано имя найденной «по пути» программы или файла данных.

- D. По команде

  ```
  BSAVE α,β1,β2[,β3]
  ```
  ,

  где:

  - BSAVE («Binary SAVE») — служебное слово;
  - α — строковое выражение;

  β1,β2,β3 — арифметические выражения, значения которых задают адреса в RAM, содержимое оперативной памяти от β1 и до β2 записывается во внутреннем представлении (см. Приложение 2 — 2.2. Внутренние коды служебных слов) на ленту под именем, равным значению строкового выражения α. Значение выражения β3 определяет адрес, с которого программа при загрузке её в память будет запускаться на счёт (по умолчанию значение β3 равно значению β1).

  Если вывод осуществляется не на дисковод A, то следует указать имя устройства. Затем укажите:

  - имя файла,
  - начальный адрес сохраняемых данных,
  - конечный адрес сохраняемых данных и, возможно,
  - адрес выполнения; если последний пропущен, то адрес, отмечающий начало данных, будет автоматически рассматриваться как адрес выполнения при загрузке этих данных (эта часть информации полезна только в том случае, если сохраняются подпрограммы на машинном языке).

  Примеры.

  - ```
    BSAVE "CAS:ГОСТ",&HF000, &HF0EE
    ```
  - ```
    BSAVE "CAS:" A,B,C
    ```

  Команда BSAVE позволяет сохранить часть оперативной памяти на ленте или дискете. Это может быть подпрограмма на MSX BASIC, данные, подпрограммы ROM, рабочая область.

  Оператором BSAVE может быть сохранена часть памяти между ячейками с адресами &H0 и &HFFFF

  На компьютерах MSX 2 возможна и другая форма записи оператора BSAVE:

  ```
  BSAVE"CAS: имя файла",β1,β2,S
  ```
  .

Этот оператор сохраняет на кассетной ленте содержимое видеопамяти.

-

E. По команде

```
BLOAD α,[,R][,β]
```

,

где:

- BLOAD («Binary LOAD») — служебное слово;
- α — строковое выражение;
- R — параметр;
- β — арифметическое выражение;значение параметра β должно быть целым, оно определяет величину смещения адресов β1, β2 и β3 загрузки и запуска (по умолчанию β=0), в оперативную память загружается файл, записанный на ленте во внутреннем представлении (см. Приложение 2 — 2.2. Внутренние коды служебных слов) под именем, равным значению выражения α.При наличии параметра R прочитанная программа автоматически запускается на счёт.

Примеры.

1. `BLOAD "CAS:DD"`
2. `BLOAD "CAS:LEN", &H1500`
3. `BLOAD "CAS:TEST", R`
4. `BLOAD "CAS:",R`
5.

0951-05.bas
0951-05.bas Fix Me!

```
5 'Инициализация компьютера.
10 CALL NETEND 'Для компьютеров серии MSX 2
20 BSAVE"CAS:initio",0,3
30 BLOAD"CAS:initio",R,&HA000
```

В последнем примере подпрограмма ROM, расположенная между адресами &h0 и &h3, загружается в RAM по адресу &HA000 и запускается на выполнение.

Команда BLOAD позволяет загрузить данные, сохранённые командой BSAVE. Должно быть указано имя устройства, если это не дисковод A, и имя файла. Может присутствовать параметр R, если данные относятся к программе на машинном языке. В этом случае программа запустится после загрузки. Данные загружаются на то место, которое они занимали перед сохранением, если не указан адрес смещения. В этом случае данное число добавляется к начальному, конечному и адресу смещения, заданным в команде BSAVE.

На компьютерах MSX 2 возможна и другая форма записи оператора BLOAD:

```
BLOAD"CAS: имя файла",S
```

Содержимое файла будет скопировано в видеопамять.

Обратите внимание, что BLOAD может использоваться в программе, не вызывая выполнения команды NEW или возврата в командный режим. Этот оператор может использоваться также для загрузки частей программы на MSX BASIC.

Пример 1. Приведём программу, которая позволяет указать начальный и конечный адреса загрузки двоичного файла.
Fix Me! 0951-11.bas
0951-11.bas

```
1 INPUT"Имя файла";A$
10 OPEN A$ FOR INPUT AS#1
20 FOR I=1 TO 7
30    I$(I)=INPUT$(1,#1)
40    PRINT ASC(I$(I)),HEX$(ASC(I$(I)))
50 NEXT
60 CLOSE
65 IF I$(1)<>"Ч" THEN PRINT"Это не BLO-файл! ":END
66 PRINT"Вы хотите вывести адреса на принтер (д/н)?"
67 Y$=INKEY$:IF Y$="" THEN 67
68 IF Y$="д" OR Y$="Д" THEN LPRINT A$
69 A1=256*ASC(I$(3))+ASC(I$(2))
70 PRINT "Начальный адрес:";HEX$(A1);" или";A1
```

```
71 IF Y$="д"OR Y$="Д" THEN LPRINT"Начальный адрес:";HEX$(A1);" или";A1
79 A2=256*ASC(I$(5))+ASC(I$(4))
80 PRINT "Конечный адрес:";HEX$(A2);" или";A2
81 IF Y$="д" OR Y$="Д" THEN LPRINT"Конечный адрес:";HEX$(A2);" или";A2
89 A3=256*ASC(I$(7))+ASC(I$(6))
90 PRINT "Адрес запуска:";HEX$(A3);" или";A3
91 IF Y$="д" OR Y$="Д" THEN LPRINT"Адрес запуска:";HEX$(A3);" или";A3
```

Замечание. Команды BSAVE и BLOAD можно применять и для работы с произвольными дисковыми файлами (устройства A: и B:).

Пример 2. Запись компьютерных игр с дискеты на магнитную ленту.

1. Загрузите программу с дискеты в оперативную память командой

```
BLOAD "имя файла"
```

2. Включите Ваш магнитофон на запись, затем выполните команду

```
BSAVE "CAS:имя файла",α,β,γ
```

где:

- α — *шестнадцатеричный* начальный адрес;
- β — *шестнадцатеричный* конечный адрес;
- γ — *шестнадцатеричный* адрес загрузки.

Значения α,β,γ Вы можете найти с помощью программы из примера 1.

Пример 3. Загрузка компьютерных игр с магнитной ленты в оперативную память.

1. Включите магнитофон
2. Выполните команду

```
BLOAD "CAS: Имя файла",R
```

3. На экране появится надпись:

```
Found: Имя файла (без расширения)
```

4. ...И «игрушка» запустится!

IX.5.2. Работа с файлами данных

Команда OPEN для устройства CAS: имеет следующий вид:

```
          ⎧ INPUT
OPEN β FOR ⎨         AS #n ,
          ⎩ OUTPUT
```

Здесь β должно начинаться с символов "CAS:" и, кроме того, отсутствует возможность, предоставляемая параметром APPEND для «наращивания» файлов. Все остальное полностью аналогично работе с дисковыми файлами (устройства A:, B: и MEM:).

Пример.
Строки 10÷30 создают файл данных на магнитной ленте. Для его считывания в память требуется запустить этот фрагмент со строки 50.
0952-01.bas
0952-01.bas Fix Me!

```
10 OPEN"CAS:хи-хи" FOR OUTPUT AS #1
20 FOR K=1 TO 500:PRINT #1,K;:NEXT
30 CLOSE #1
40 STOP
50 CLEAR 1000
60 OPEN"CAS:хи-хи" FOR INPUT AS #1
```

```
70 LINE INPUT #1,M$:PRINT M$;
80 IF NOT EOF(1) THEN 70
90 END
```

IX.6. Дополнение

Три очень *полезных* программы.

1. Программа, осуществляющая посекторное копирование диска на кассету.
096-01.bas
096-01.bas Fix Me!

```
6 CLS:PRINT:PRINT"<<Посекторное копирование диска на кассету>>":PRINT:PRINT
11 DEFINT A-Z:GOSUB 51:I=&H9000:DEFUSR=I:J=&HA000:DEFUSR1=J:PRINT"Сколько минут протягивается одна сторона кассеты ??":INPUT CT:PRINT:PRINT"С какого по какой сектор копировать S1, S2 (0-1500)??":INPUT S1,S2
16 READ D$:IF D$<>"#" THEN POKE I,VAL("&h"+D$):I=I+1:GOTO 16
21 DATA 2A,51,F3,11,00,00,01,00,02,CD,5C,00,C9,#
22 READ D$:IF D$<>"#" THEN POKE J,VAL("&h"+D$):J=J+1:GOTO 22
23 DATA 2A,51,F3,11,00,B0,01,00,02,ED,B0,C9,#
31 FOR S=S1 TO S2:D$=DSKI$(1,S):CLS:U=USR(0):LOCATE 0,17:PRINT "Сектор N";S:I=USR1(0):BSAVE"CAS:C"+STR$(S),&HB000,&HB1FF:GOSUB46:IF T>CT-5 THEN GOSUB 56
36 NEXT:END
46 GET TIME T$:KEY 2,T$:C1$=LEFT$(T$,2):C1=VAL(C1$)-C:M1$=MID$(T$,4,2):M1=VAL(M1$)-M:T=C1*60+M1:RETURN
51 GET TIME T$:KEY1,T$:C$=LEFT$(T$,2):C=VAL(C$):M$=MID$(T$,4,2):M=VAL(M$):RETURN
56 CLS:LOCATE 10,10:PRINT "Проверь ленту":LOCATE 10,12:PRINT"Кончилась ??? (D/N)":SET BEEP 4,4
61 BEEP:O$=INKEY$:IF O$="" THEN 61 ELSE ON INSTR("@@@DdдДNnнH",O$)\4 GOTO 66,76:GOTO 61
66 PRINT "Вставил новую ? (D)":SET BEEP 3,4
71 BEEP:O$=INKEY$:IF O$="" THEN 71 ELSE IF INSTR("DdдД",O$)THEN GOSUB 51:RETURN ELSE 71
76 RETURN
```

2. Программа, осуществляющая посекторное восстановление диска с кассеты.
096-02.bas
096-02.bas Fix Me!

```
6 CLS:PRINT:PRINT"<<Посекторное восстановление диска с кассеты>>":PRINT:PRINT
11 DEFINT A-Z:GOSUB 51:I=&H9000:DEFUSR=I:J=&HA000:DEFUSR1=J:PRINT"Сколько минут протягивается одна сторона кассеты ??":INPUT CT:PRINT:PRINT"С какого по какой сектор копировать S1, S2 (0-1500) ??":INPUT S1,S2
16 READ D$:IF D$<>"#" THEN POKE I,VAL("&h"+D$):I=I+1:GOTO 16
21 DATA 21,00,B0,11,00,00,01,00,02,CD,5C,00,C9,#
22 READ D$:IF D$<>"#" THEN POKE J,VAL("&h"+D$):J=J+1:GOTO 22
23 DATA 01,00,02,2A,51,F3,54,5D,21,00,B0,ED,B0,C9,#
31 FOR S=S1 TO S2:BLOAD"CAS:C"+STR$(S):CLS:U=USR(0):LOCATE 0,17:PRINT"Сектор N";S:I=USR1(0):DSKO$ 1,S:GOSUB 46:IF T>CT-5 THEN GOSUB 56
36 NEXT:END
46 GET TIME T$:KEY 2,T$:C1$=LEFT$(T$,2):C1=VAL(C1$)-C:M1$=MID$(T$,4,2):M1=VAL(M1$)-M:T=C1*60+M1:RETURN
51 GET TIME T$:KEY 1,T$:C$=LEFT$(T$,2):C=VAL(C$):M$=MID$(T$,4,2):M=VAL(M$):RETURN
56 CLS:LOCATE 10,10:PRINT "Проверь ленту":LOCATE 10,12:PRINT "Кончилась ??? (D/N)":SET BEEP 4,4
61 BEEP:O$=INKEY$:IF O$="" THEN 61 ELSE ON INSTR("@@@DdдДNnнH",O$)\4 GOTO 66,76:GOTO 61
66 PRINT "Вставил новую ?(D)":SET BEEP 3,4
71 BEEP:O$=INKEY$:IF O$="" THEN 71 ELSE IF INSTR("DdдД",O$)THEN GOSUB 51:RETURN ELSE 71
76 RETURN
```

3. Программа, позволяющая определить тип файла на кассете.
096-03.bas

096-03.bas

Внимание! При наборе программы *строго* соблюдайте пробелы между символами и нумерацию строк.

Fix Me!

```
1 WIDTH40:PRINT"<<<  FORMAT FILES ON CASSETTE  >>>":PRINT:KEYOFF:PRINT"<<<  BY LEO BOYARSKY (c)
1988  >>>":POKE&H80CA,143'□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
2
CLEAR1000:MAXFILES=1:ONSTOPGOSUB15:STOPON:ONERRORGOTO14:RESTORE4:A=VARPTR(#1):NEW'□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□
3 DEFFNA$(I)=RIGHT$("000"+HEX$(PEEK(I)+256*PEEK(I+1)),4)
4 FORI=0TO8:READX:POKEA+I,X:NEXT:DATA1,0,0,0,255,0,0,0,0'□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
5 DATA&HDB,&HC3,&H32,&H58,&HD3,&HC3,&H74,&H42
6 PRINT:X$=INPUT$(255,1)
7 W$="":FORI=A+9 TOA+18:W=PEEK(I):W$=W$+CHR$(W):NEXT
8 N$="":FORI=A+19TOA+24:N=PEEK(I):N$=N$+CHR$(N):NEXT
9  IFW$="ccccccccc"THENW$="BASIC    (cload)":GOSUB11ELSEIFW$="ЙЙЙЙЙЙЙЙЙЙ"THENW$="BASIC    (
load)":GOSUB11ELSEIFW$="ппппппппп"THENW$="OBJEKT  (bload)":GOSUB11:I=A+25:GOSUB12
10 PRINT:PRINT"<<<  Checking next file        >>>":RUN2
11 S$=" The "+N$+" is "+W$+" file.":PRINTS$:RETURN
1 2
SA$=FNA$(I):IFNOT(("8"<=SA$)AND(SA$<="F380"))THEN13ELSEEA$=FNA$(I+2):IFNOT(("8"<=EA$)AND(EA$<="F
380"))THEN13ELSERA$=FNA$(I+4):IFNOT(("8"<=RA$)AND(RA$<="F380"))THEN13ELSES$=S$+CHR$(13)+"Start:&
h"+SA$+"; End: &h"+EA$+"; Run: &h"+RA$:PRINTS$:RETURN
13 I=I+1:IFI>A+255THENPRINT:PRINT"Don't check address !":RETURNELSE12
14 RESUMENEXT
15 KEYON:POKE&H80CA,58'□□□□□□□□□□□□□□□□
16 END
```

Диск с примерами

Загрузить образ диска

Открыть диск в WebMSX

MSX, BASIC, □ Пособие по программированию на MSX BASIC