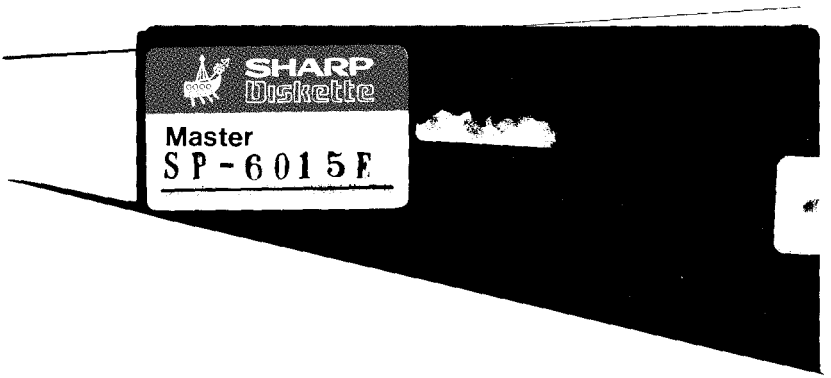
SHARP

SERIES MZ-80

DISK BASIC



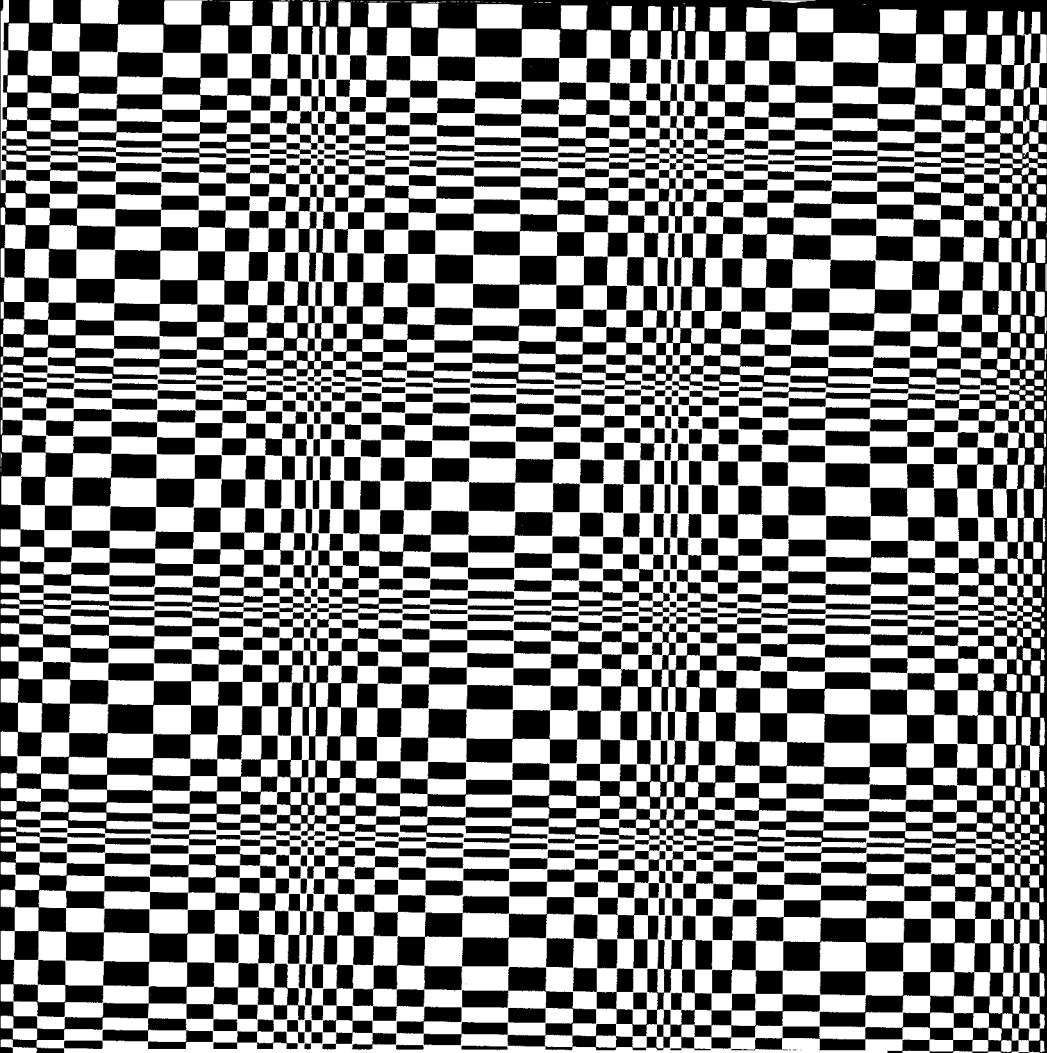
Master Diskette

Introduction

We thank you so much for your purchase of our Disk Basic, a systems software for series MZ-80 Sharp personal computer. We hope you read this book thoroughly before using your Disk Basic so that you can take full advantage of the facilities available by use of floppy disks. **The Disk Basic needs a RAM area of more than 36K bytes. If the RAM area of your machine is less than 36K bytes, it must be expanded.**

Disks and diskettes require to be carefully handled. We wish you to read carefully and practice strictly the precautions given at the end of this book and described in the Instruction Manual of the Floppy Disk.

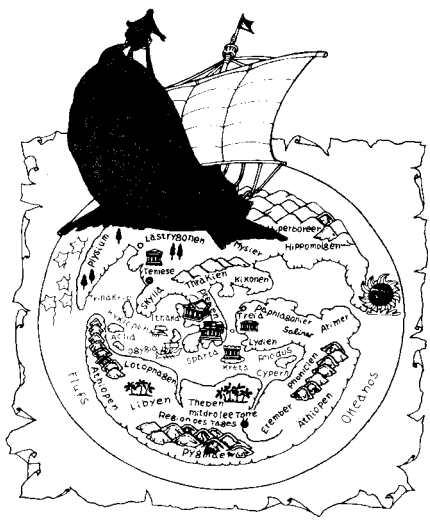
Master diskette has glued "Write Protect" seal — silver-colored seal to inhibit program writing — on the "Write Protect" notch. This seal is intended to protect the contents of the said diskette in the event of wrong file operations or unexpected problems — for example, disconnection of the power plug from the electric outlet or power failure. Never remove the seal. It is good practice to check the seal from time to time and replace it with new one as soon as you have find it damaged at starting to peel.



The Series MZ-80 Personal Computer System has been developed based on the design philosophy of "Clean Computer." Product ratings and command specifications may be changed, added or deleted without prior notice for future improvement. Therefore, you are requested to take special care about version numbers of the monitor (supplied in the form of ROM) and systems programs (supplied stored in a cassette tape or on miniature floppy diskette).

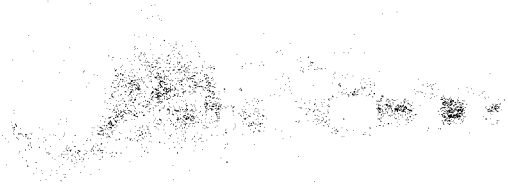
Descriptions, we believe, are free from errors. In principle, we do not take any responsibility for problems that changes in contents due to version-up, erroneous prints and missing descrption could pose, because this book has been written to help you understand the DISK BASIC. This book has been edited based on monitor SP-1002, DISK BASIC SP-6015.

Guide to Series MZ-80 Disk Basic



Man has always have been excited by the wonders of the world from ancient times to the present day.

Who said first that man was the first animal to use fire? Indeed, anything blazes if burnt, but why does it burn and blaze? It is nothing but a wonder that a certain element of substance takes wing and travels with light velocity when a sheet of common paper burns. Don't you think so?
Do you say that semiconductors are made of silicon? Then go to a sandpit, you will find there a large amount of silicon. It is a great pleasure for building a sand castle to dream what computers are concealed there. When you pick pure silicon out of the sand, it computes or stores π. So, you might feel as if you were communicating with the earth through the computer. We should like to dig into this wonder world through our own thoughts even if in a rather round-about way. May this book then serve, as a guide to that unknown world!

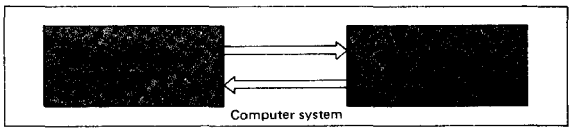


Contents

Preface

Compared with the conventional cassette-based Basic, the Disk Basic is a very file-oriented Basic with a much more powerful file control functions, which will widen the software horizon fo Series MZ-80 Clean Computer System. In other words, this Basic enables files not only to have the ability to accumulate data, but to serve as data area in direct connection with the whole computer system as well, making full use of the advantages of floppy disk files: large capacity and high-speed data accessibility. Also, the Basic allows program texts themselves to be overlayed or linked in unit of job by the use of the commands CHAIN and SWAP, enabling a new technique to be practiced called program segmentation.

The good understanding of this Disk Basic and the exhaustive utilization of its performance, we consider, should enable the full and more sophisticated use of this computer system in small businesses and education. In general, the computer system can be conceptually divided into two major systems: the logical interior system composed of data processing unit and main memory, and the exterior file system composed of data and program banks processed in the above interior system. The floppy disk unit and the Disk Basic play an important role mainly to control the said file system.



This manual will discuss the topic "What is the file?" in Chapter 1 in order that you first understand the role, form, and function of files, as well as some specific concepts pertaining to them. In this chapter, programming is exemplified without the use of commands peculiar to the Disk Basic in consideration of the purpose of that chapter – grasping the concept of file.

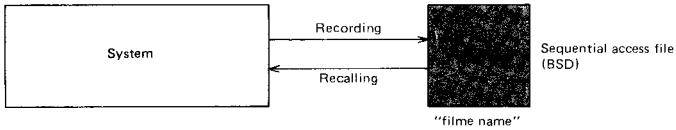
Chapter 2 will discuss new commands employed in the Disk Basic: file control commands, error correction command, logical-open USR function, and others.

In Chapter 3 we will demonstrate three sample programs, "Report Generator Program," "Amateur Radio Program," and "Inventory Control Program," as a guide to applications of the Disk Basic, and further give detailed explanation of them and a variety of techniques.

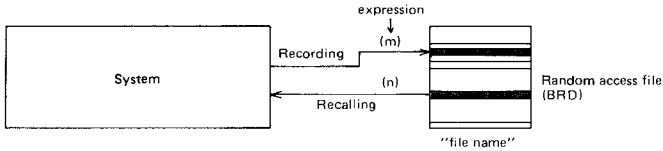
Chapter 4 will summarize Disk Basic commands, and error indications, and demonstrate the way to use utility programs prepared for master diskette.

Data file control

Data files are grouped according to their access form into two types: sequential access file and random access file. The sequential access file handles accessed file data as a string of blocks, in which the file name of a group of data to be accessed is specified and the data is recorded on a specified file or recalled from there in the sequence from the first block to the last block.



By contrast, the random access file provides a random access to file data. In detail, although one random access file comprises a group of data specified by one file name, each individual data item is recorded as an array element on a file, and its writing and reading occur by specifying its array expression.

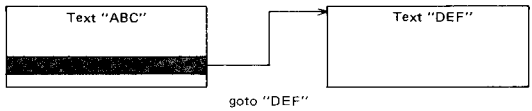


In general, when some collective data is taken to be a chain of information – for example, a series of decimal notation data used in preparing a machine language program with POKE statement, or table elements arranged in sequence from the beginning, it can be said to be effective to record them on the sequential access file. Besides, the random access file is effective when a pack of data is required to be accessed not only as a whole, but by its each individual element as well, for example, when rewriting, retrieval, rearrangement, or deletion of data is required.

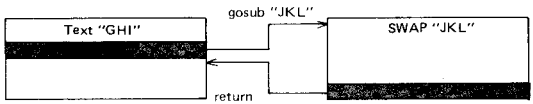
Program file control

Program file control commands – CHAIN, and SWAP –, are intended to overlay programs in a program file under execution.

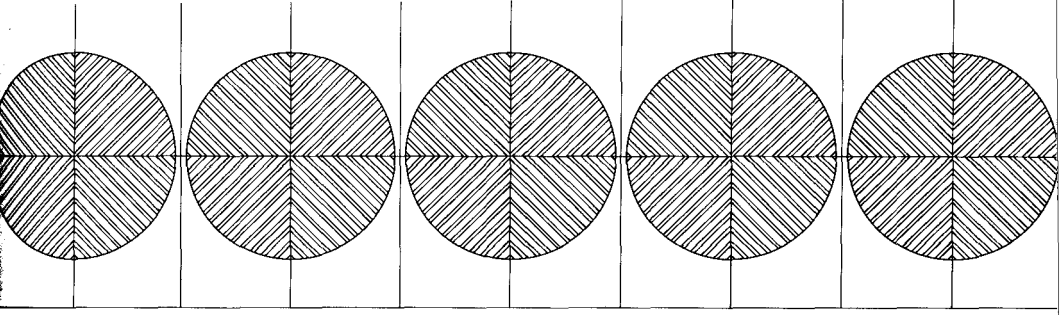
The CHAIN command has a function as "goto 'file name' " as illustrated below.



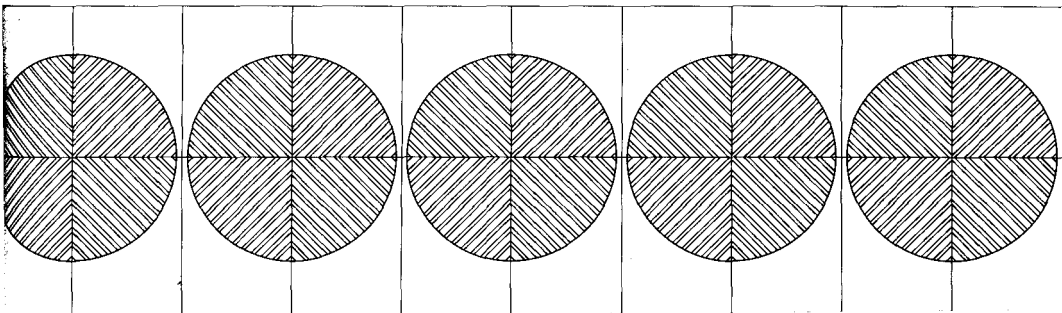
Another command SWAP has a function as "gosub 'filename'," being able to return to the initial program after the completion of the overlay subprogram – an overlay occurs then as well.



In addition to the above-mentioned commands, the Disk Basic has a command to logically open and link machine-language program files. Also, it permits a wide variety of file management using utility programs.



Chapter 1
Concept of File



First_____a file

We frequently use in our daily life the term "file." But we must first of all, understand the meaning of this term in computer terminology. A file of newspapers, for instance, means nothing to the computer.

Then, how about scrapbooks —? No ! Now, how about scrapbooks that have been edited by column — economy, social affairs, sports, household. Not allowed though they are somewhat nearer to file. The term "file" we will use in the following discussions does not mean a file of data collected at random, but a file of data that has been collected and edited for a definite purpose and in a fixed format. A mere file of data, even though so useful, serves no purpose at all. The important point is to arrange data so that it can be quickly retrieved and processed as the occasion demands.

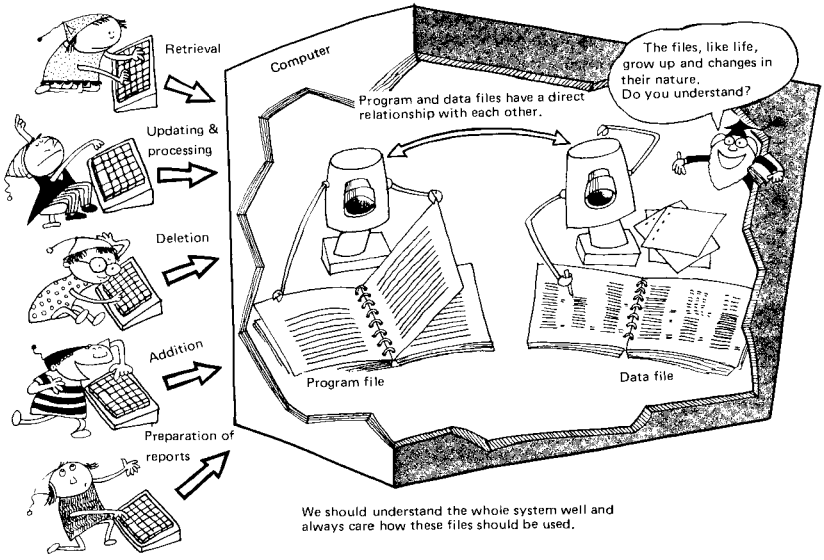
Roughly speaking, the computer file appears in the following two forms, which are related with each other.

Data file

The file is loaded with a chain of data that is collected so that it can be processed by the computer. For example, a pack of data about the stock of a certain product is a data file called inventory file.

Program file

The file contains programs to process or retrieve collected data held in a data file: systems programs, such as BASIC, FORTRAN, COBOL, PASCAL, ASSEMBLER, or application programs made up of these systems programs, for example.

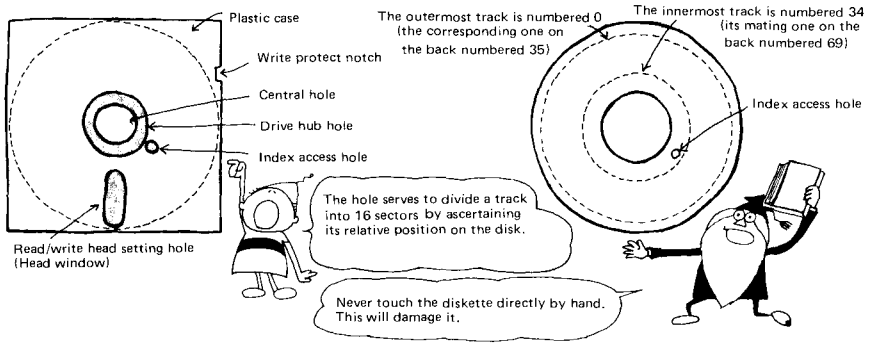


We should understand the whole system well and always care how these files should be used.

Let's find now a clever friend of our Disk Basic called "diskette."

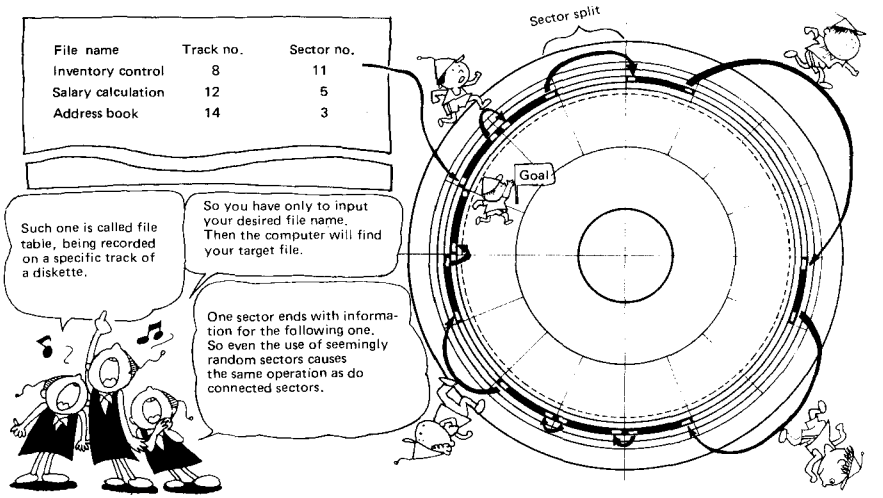
The diskette is square in appearance, but its heart, two magnetic areas on its both faces, is round. Each of these two disk-like areas has 35 tracks ranging from its outer circumference to inner one: that is, a diskette has a total of 70 tracks. The disk-shaped area resembles a record in shape, but substantially differs from it in that while the latter has a single groove, the former has independent closed-circle tracks. All these tracks are individually divided into 16 sectors – one sector is equivalent to 128 bytes – each of which is the minimum unit for reading and writing.

The illustration below will help you to gain a better understanding of the construction of the diskette.



The Disk Basic has a control function to free the operator from the need to know the details of the complicated formation of tracks and sector splits: for example, with what sector of some track the inventory control file begins and how may sectors are occupied with it.

Now, we understand how files are formed on a diskette. Of course, you do not need to know, because the Disk Basic controls it.



What applications does the sequential file fit?

When you want to read the 100th item of data from a some file made up on a cassette tape, you must pass through the 1st to the 99th item without reading them. This is because a cassette tape is designed to permit serial access only.

Meanwhile, the floppy disk is made to be capable of gaining access to any position, but it can be also used in the same accessing mode as a cassette tape. For instance, suppose you recorded individual names and addresses at random without processing at all to make an address book on a floppy disk.

In order to search some address later by specifying the corresponding person's name you would have to read all the addresses ahead of your target in sequence out of your address book. Such a file, though it uses a floppy disk, that permits access to an arbitrary position, appears to be formed into a sequential file.

Hereafter, we will call sequential files by either of the following two terms depending on which phase of them, software or hardware, we place emphasis on when forming those files.

Sequential access file

This term stresses the hardware phase of sequential files with the word "access." A typical example of a sequential access file is a cassette tape.

Sequential file

This term, not containing the word "access", emphasizes the software-oriented meaning of sequential files. However, as the computer is designed so that its hardware phase dominates its software one, we cannot help forming a sequential file for a sequential access file such as a cassette tape. Given below is the summarization of sequential file's merits and demerits.

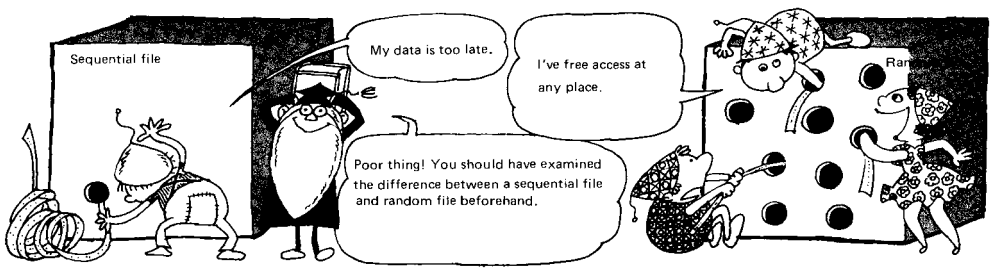
Merits

- Provides serial data storage without interruption, making use of memory efficiently.
- Works efficiently when processing all data from of a file in one stroke: that is, it provides for economical data processing when the retrieval of individual data is not required.

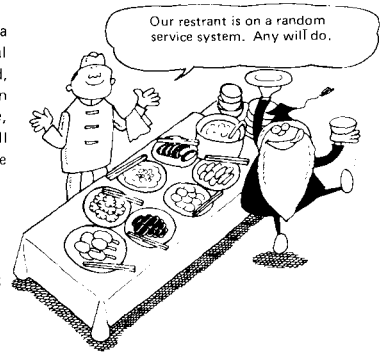
Demerits

- Not capable of providing retrieval of random data.
- Requires the rewrite of all the data behind it when some data is added to or deleted from the file.

We must decide the applications of this file in consideration of these merits and demerits. Monthly salary calculations, for example, can be said to be suitable for this file, because those calculations need nothing but to access all the data stored on a file in order of member's identification number. But, when you need salary data for persons engaged in a specific job or having a specific family composition, the random file should serve your purpose better than this file — sequential file.



What are the best applications for random files?



It is the random file that directly records data in or reads it out of a diskette at any place in order to make up for the demerits of a sequential file. Generally speaking, data the computer is to process is not haphazard, and has some relationship with each other, as far as applied to a common object. If you wish to make a file along those lives, then the random file, we can safely say, is best to use. As for sequential files, we will also call random files by two different terms depending on which of their software and hardware phases are stressed when they are made up.
The different terms are:

Random access file

The term refer to the hardware-oriented meaning of the random file. The floppy disk, through its inherent line allows free access to any point of a file, can be also used as a sequential access file.

Random file

This term favours, by contrast to the above term, the software-oriented meaning. The formation of this file needs an intermediate processing, called "coding."

For details of coding, consult pages 23 thru 25. Here, we will merely give a brief description of a random file's merits and demerits and coding.

Merits

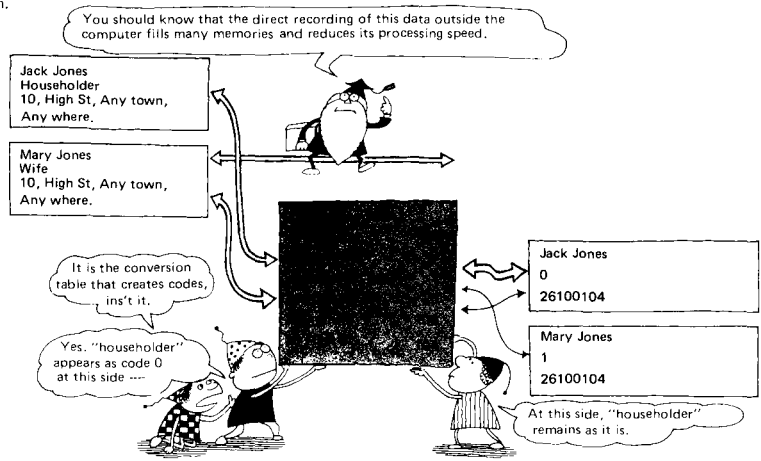
- Provides quick retrieval of desired data.
- Provides for easy addition and deletion of data.

Demerits

- Creates empty space in data records.
- Difficult to optimize through code conversion.

The point of the random file seems somehow to lie in coding. Well, what is the code? What merits does the coding bring? The illustration below tells us about that.

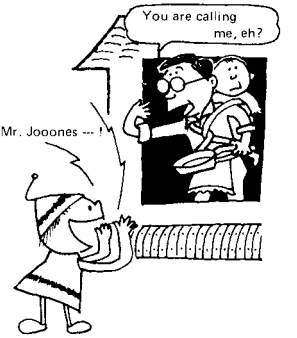
Meanwhile, the extension of the concept of code allows you to record data regardless of the order of input data and of addresses to be recorded on a diskette. In this case the coding applied is not a conversion table but a conversion equation. Then specified data and addresses appear to correspond to each other through the conversion equation.



To make up a file.....

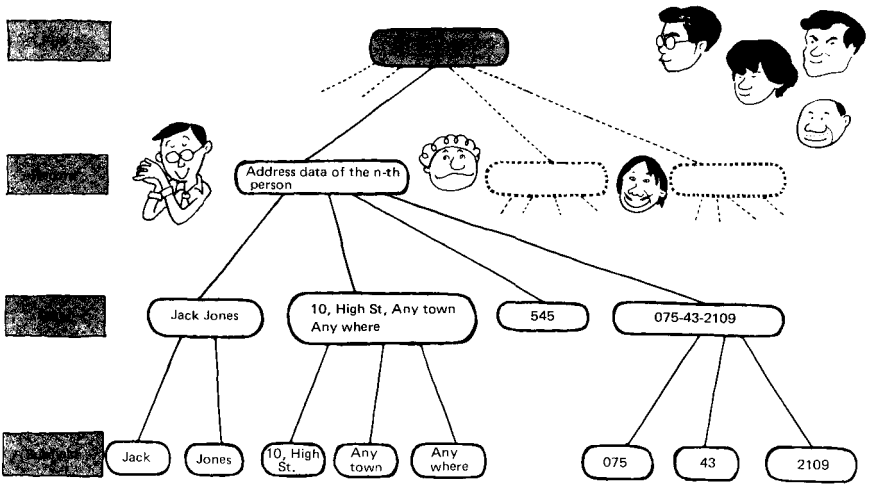
Now, let's try to make an address book. For the present, we shall try to arrange data that seems necessary and take a sample.

[redacted]	Jack Jones
[redacted]	10, High St, Any town, Any where
[redacted]	604
[redacted]	075-43-2109
[redacted]	Nagaike-cho, Abeno-ku, Osaka
[redacted]	545
[redacted]	06-54-3210
[redacted]	Century Software Company
[redacted]	Manager

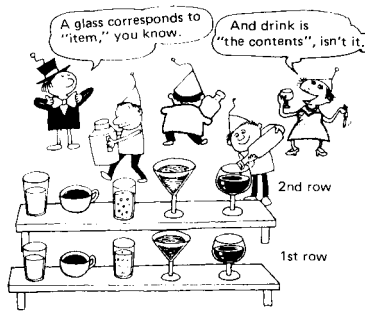


Here, we call a collection of data the computer is to process, such as the above address data of Mr. Jones, a record, and the smallest component element of each record such as name – Jack Jones – or address of his home – 10, High St, Any town – "field." Speaking of the data of Mr. Jones, the record is composed of a total of 8 fields. The field can be further divided into a subordinate unit termed "subfield."

The architecture of such a file, can be better understood in the form of a tree.



The maxim in the Disk Basic world says, "A glass does not always accept fruit juice."

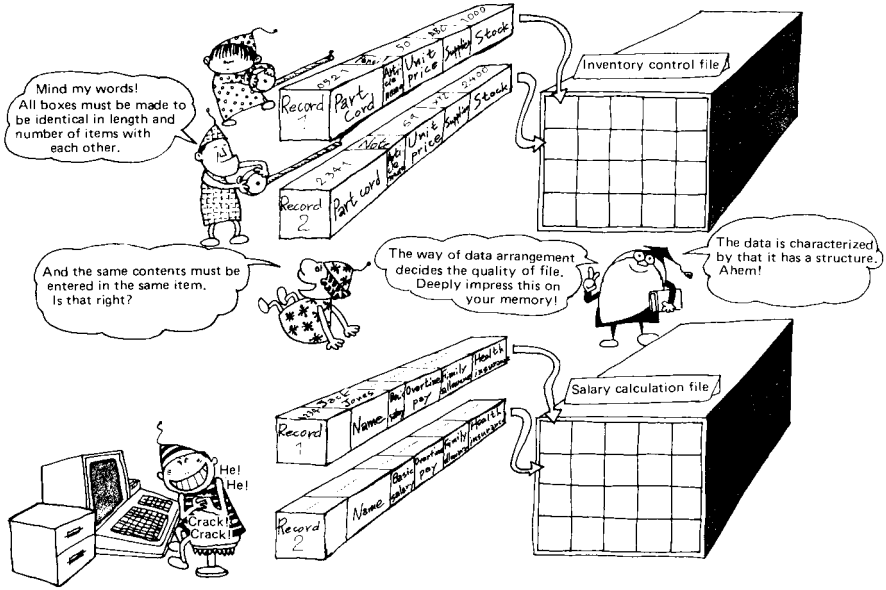


In an actual computer file, specific labels, such as "Name", "Address of home", or "Postcode", are not individually given as data. Hence, data representing "Jack Jones", for example, does not tell us directly for oneself whether it is a person's name or brand name, but merely says that it stays as the first field of the n-th record — address data of the n-th person. Let's take a glass as an example. Suppose a file is composed of large and small glasses arranged in a fixed format. If the first glass in the first row is filled with milk, the second with tea, the third with fruit juice, the fourth with ---, the glasses in the second and subsequent rows need to be filled in the same manner as the ones in the first row. If such a procedure is not followed — for example, if the glass arranging sequence in the first row is different from that in the second one or the sorts of drinks in them is different, though the manner to arrange those glasses is quite the same — the file does not serve its inherent purpose.

The discussion of a file refers to the glass as "item," and each drink as "the contents."

The illustration below depicts the relations of various items and their contents by comparing these to card files.

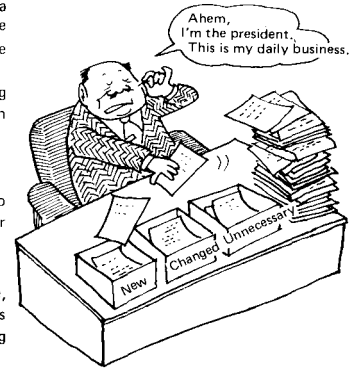
The indispensable point in computer-based filing is to devise a most efficient format for the proposed processing. In this case, the term "efficiency" implies the following: small in storage volume, high in processing speed, versatile, easily programmable, and easily understandable. How to construct a file should be decided by a programmer according to his purpose, and its workmanship directly affects the efficiency of the program to process that file.



Daily data(transactions) are material for the master file.

Such data as inventory control one changes without ceasing. Some data is of no use, though it was necessary when designing a system, and must be removed, some new data requires to be recorded on a file, and some existing data needs updating.

In order to deal with such requests it is required to sort files depending on their purpose. We will illustrate this later by giving an example of an inventory control file.



Master file

This file contains basic data essential to inventory control, being, so to speak, a inventory ledger. It is the most important information source for analysis and preparation of documents.

Transaction file

This file holds data that is created day by day and serves to retrieve, update and abolish the information recorded in the master file. The file is used, for example, to record the article name, date, or number of incoming or outgoing goods each time a despatch or delivery occurs.

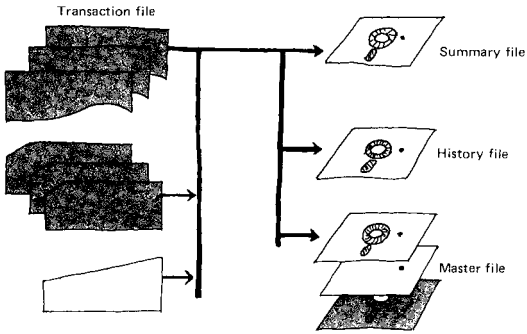
Summary file

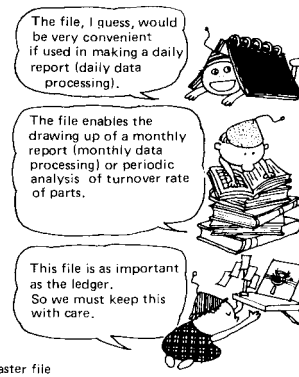
This file serves to summarize data from the transaction file in accordance with a certain categorization. In this file, data is summarized in a simple format by category — number of goods in for each supplier, or sales by product, for example.

History file

This file retains master file data that has become of no use in terms of data processing, or transaction files that were created for a fixed period of time.

Parts distribution files of certain products that are not currently being produced at all are displaced from the master file to this history file, used for maintenance and servicing.



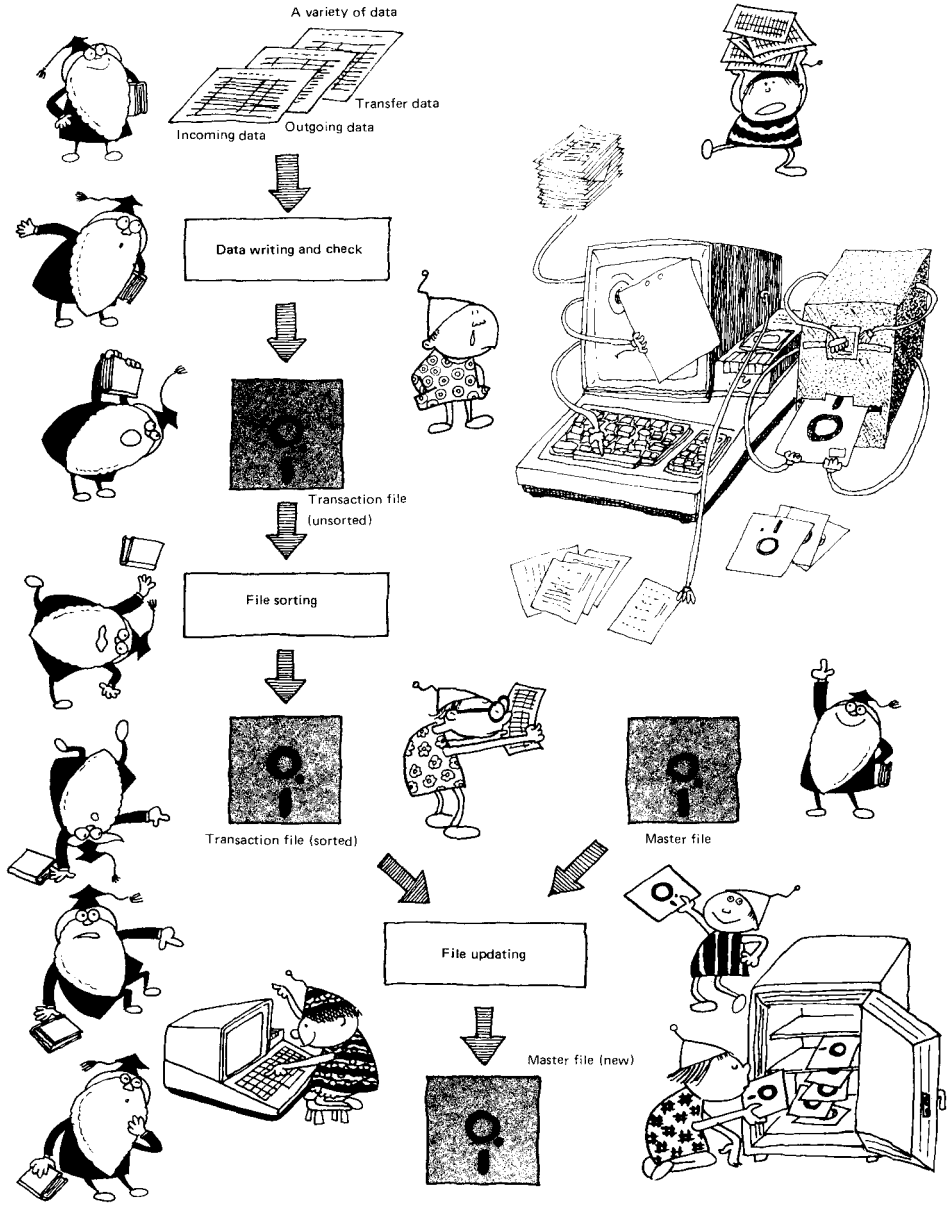


An example of transaction file

Code of department of issue	Slip no.	Part code	Quantity	Date of issue	Code of allottee

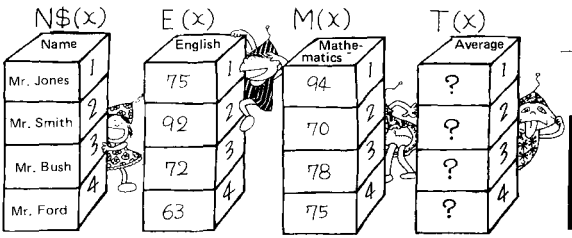
An example of master file

Part code	Part name	Unit price	Stock	Remaining order	Number of the goods in (daily)	Number of goods out (daily)	Number of orders filled	Supplier's code	Applicable product #1	Applicable product #2	. . .


A variety of data
Transfer data
Incoming data
Outgoing data
Data writing and check
Transaction file
(unsorted)
File sorting
Transaction file (sorted)
Master file
File updating
Master file (new)

What does the term "sorting" mean at all?

The term "sorting" means rearranging records contained in a file according to the contents of a certain item. Suppose students names and the results of their examinations in three subjects, French, Mathematics, and English, are listed in a file. It is the object of sorting to rearrange the file into alphabetical order of name or in order of the average score of the three subjects. Sorting is a method of managing files, utilizing the records that have some index to enable their arrangement – for example, magnitude, alphabetical order, or chronological order. Lets's discuss the procedure to rank the following examinees in order of average score.



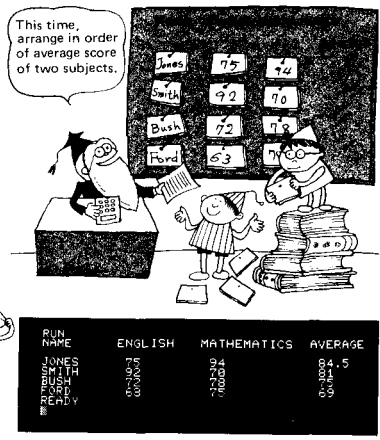


Photo 1

```
10 READ X                                   : Reads the number of examinees.
20 DIM N$(X), E(X), M(X), T(X)
30 FOR I=1 TO X
40 READ N$(I), E(I), M(I)                   : Reads every item.
50 T(I) = (E(I) + M(I))/2                   : Computes average scores and stores them.
60 NEXT I
70 DATA 4
80 DATA JONES, 75, 94
90 DATA SMITH, 92, 70
100 DATA BUSH, 72, 78
110 DATA FORD, 63, 75
120 FOR I=1 TO X-1:T=0
130 FOR J=I TO X
140 IF T(J)<=T THEN 160
150 T=T(J):S=J                              : Sets the largest of T(I) thru T(X) at T,
                                              and fix the pointer as S.
160 NEXT J
170 T(S)=T(I):T(I)=T                        : Replaces I-th average with S-th one.
                                              The value of T(S) is then placed in T.
180 N$=N$(S):E=E(S):M=M(S)
190 N$(S)=N$(I):E(S)=E(I):M(S)=M(I)         : Replaces I-th item with S-th one by way of
200 N$(I)=N$:E(I)=E:M(I)=M                    N$, E, and M.
210 NEXT I
220 PRINT "NAME", "ENGLISH", "MATHEMATICS"; TAB(33); "AVERAGE"
230 FOR I=1 TO X
240 PRINT N$(I), E(I), M(I), TAB(33); T(I)
250 NEXT I:END
```

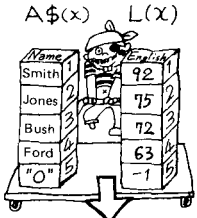
In this chapter, all programming practices are made on the assumption that data read or written in the floppy disk be variables because commands to control the floppy disk are not described yet. It may offer, in a sense, good training for you to understand the idea of programming alone and extend it later.

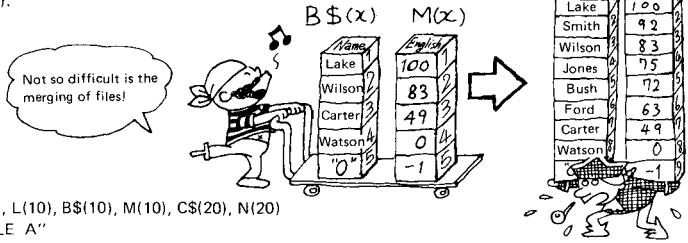
Why don't you try to increase data by merging files?

You imagine! Here are two files, and their records are the same in nature and sorted in the same order. The technique to make up those two files into one — this is of course the same in nature and manner of sorting — is termed "Merging."

Merging two files sorted, for example, in order of score of an examination in English would create such a file as shown below. However, it is then presupposed that both the files should end with zero(0) — an end mark — in their columns of "Name."

The reference program shown below is so devised as to compulsively set his score at −1, not 0(zero), when zero(0) is given instead of person's name in the entry of names and scores. This is because the score could be possibly zero (0).






```
10 DIM A$(10), L(10), B$(10), M(10), C$(20), N(20)
20 PRINT "FILE A"
25 PRINT "NAME ?", "AVERAGE ?"
30 FOR I = 1 TO 10
40 INPUT A$(I), L(I)
50 IF A$(I) = "0" THEN L(I) = −1:GOTO 80
60 NEXT I
70 PRINT "FILE A FILLED":STOP
80 PRINT "FILE B"
85 PRINT "NAME ?", "AVERAGE ?"
90 FOR I = 1 TO 10
100 INPUT B$(I), M(I)
110 IF B$(I) = "0" THEN M(I) = −1:GOTO 140
120 NEXT I
130 PRINT "FILE B FILLED":STOP
140 I = 1:J = 1:K = 1
150 IF L(I) > M(J) THEN 180
160 IF L(I) < M(J) THEN 210
170 IF L(I) = −1 THEN 240
180 C$(K) = A$(I)
190 N(K) = L(I)
200 I = I + 1:K = K + 1:GOTO 150
210 C$(K) = B$(J)
220 N(K) = M(J)
230 J = J + 1:K = K + 1:GOTO 150
240 C$(K) = "0": N(K) = 1
250 PRINT "FILE C (MERGED)"
260 PRINT "NAME", "AVERAGE"
270 FOR I = 1 TO 20
280 IF C$(I) = "0" THEN 310
290 PRINT C$(I), N(I)
300 NEXT I
310 END
```

Reads an examinee's names and scores into file A.

Reads an examinee's names and scores into file B.

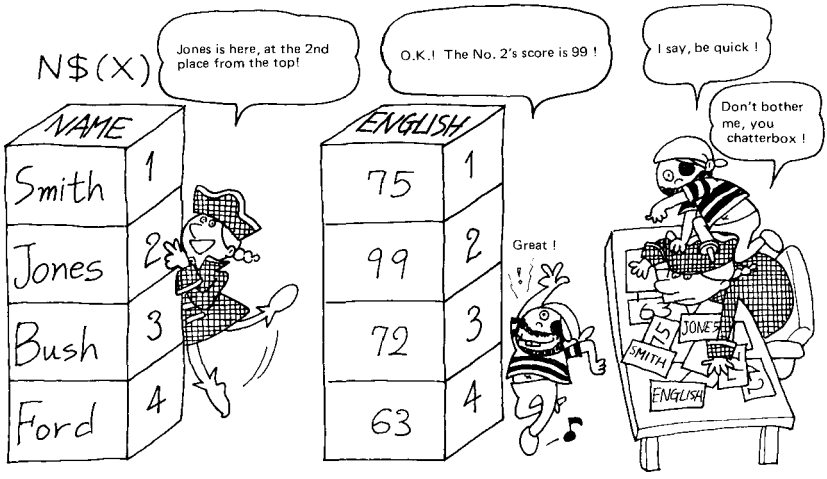
Displaces the larger one of two scores L(I) and M(J) together with the corresponding name to C$(K) and N (K), and then advances the processed two pointers of I, J, and K, for instance, I and K, or J and K, by one step, ready for the following comparison.

The machine comes to this step only when L(I) = M(J) If the score is then −1, it means that the file has come to an end. Hence, the machine then completes the merging by setting an end mark to file C. If the score is not −1, it means that both scores are equal, forcing the machine to perform the same processing as when L(I) > M(J).

To be short, consulting a dictionary is the first step toward retrieval.

Needless to say, the usefulness of a dictionary exists in that it teaches us several meanings when we consult it about certain words.

Discussing this about a file, its usefulness can be said to exist in that it provides several associated items when you have located your desired items. This is true of not only a dictionary but a telephone book, postcode list, and timetable as well. Such a process of searching for an item and thereby finding out other several items related to it is termed "retrieval."




```
10 DIM N$(10), E(10)
20 READ X
30 FOR I = 1 TO X
40 READ N$(I), E(I)
50 NEXT I
60 DATA 4
70 DATA SMITH, 75, JONES, 92
80 DATA FUSH, 72, FORD, 63
90 INPUT "YOUR NAME ?";N$
100 FOR I = 1 TO X
110 IF N$(I) = N$ THEN 150
120 NEXT I
130 PRINT "RECORD IS ABSENT"
140 GOTO 90
150 PRINT N$(I), E(I)
160 GOTO 90
```

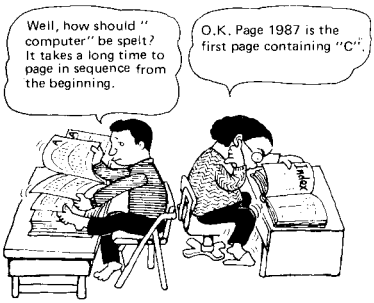
It is decisive in achieving economical retrieval to make efficient programming of the computer.

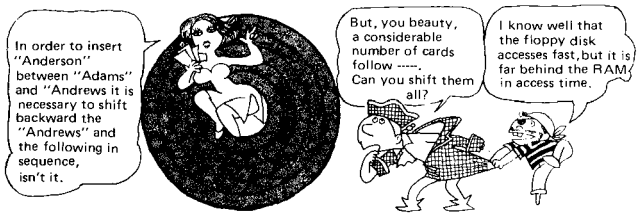
Retrieval is something like consulting a dictionary. But, there might not be such a leisurely person as to page in sequence starting with its first page when he consults a English dictionary about the word "Computer," for example.

It is then rational to roughly locate the word by the letter of the alphabet "C" because the principle of words listing in the English dictionary is to arrange them in alphabetical order.

The above principle of the dictionary is true in general of the programming method of the computer. If given data is sequentially stored at random in the computer, all data ahead of it, when a certain data item is located, must be fetched until that data is obtained. The method to program data at random without following this principle, when the amount of input data grows to as much as 1,000 through 10,000 will overload the computer, no matter how good it is.

Then, let's devise a program to store data in alphabetical order, using the principle of the English dictionary. This practice, however, is intended to demonstrate that the above-mentioned principle is not enough to store a great deal of data in the floppy disk.






```
10 PRINT "HOW MANY EXAMINEES ARE TO BE STORED ?":INPUT Z:Z=INT(Z)
20 DIM N$(Z), E(Z)
30 PRINT "INPUT NAMES AND SCORES"
40 FOR I = 1 TO Z
50 INPUT N$(I), E(I)
60 NEXT I
70 I = I – 1
80 FOR J = 1 TO I – 1
90 FOR K = 1 TO I – J
100 A = LEN(N$(K)):B = LEN (N$(K + 1)):C = 1
110 IF A < B THEN B = A
120 X = ASC (MID$ (N$(K), C, 1)):Y = ASC (MID$ (N$ (K + 1), C, 1))
130 IF X > Y THEN 170
140 IF (X < Y) + (B = C) * (A = B) THEN 200
150 IF (B = C) * (A > B) THEN 170
160 C = C + 1:GOTO 120
170 N$ = N$(K):E = E(K)
180 N$(K) = N$(K + 1):E(K) = E(K + 1)
190 N$(K + 1) = N$:E(K + 1) = E
200 NEXT K, J
210 PRINT
220 FOR I = 1 TO Z
230 PRINT N$(I), E(I)
240 NEXT I
250 END
```

Inputs names and scores.

Compares adjacent names in alphabetical order.

Replaces adjacent names and scores with each other. In other words, displacement occurs here. The procedure seems easy because data is present in the RAM, but it is far complicated when data is held on the floppy disk.

Displays the result.

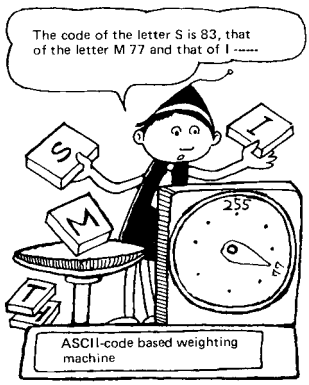
Let's try a retrieval method other than the one of a dictionary.

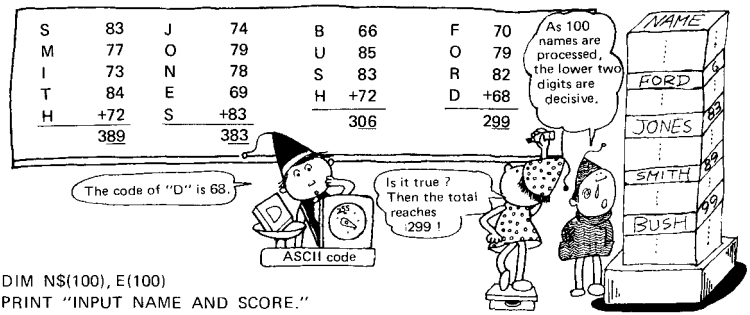
It is effective for a small amount of data to compare names in the block and decide where those should be stored. This, however, is not true of processing a large volume of data, which cannot be easily processed unless their storing places are fixed first.

For easier understanding of the above, suppose that we place the names and scores of 100 examinees in the memory. According to the method so far employed, 100 arrays are then ailoted to those names and scores. This means that the data can be stored and retrieved in a moment when any one of array numbers "0" thru "99" corresponds at once to a certain name.

Then, what is the procedure for that?

We do not handle every name as a whole, but break it down into the letters of the alphabet, each of which is a mere code. Therefore, adding the codes of the individual letters each name makes different integers. If the names of 100 examinees are now put in the memory, the remainder stays within 0 through 99 when the sum of their codes is divided by 100. Such would be the procedure for positioning the arrays to be stored according to the above-mentioned method.



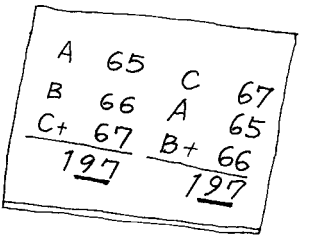


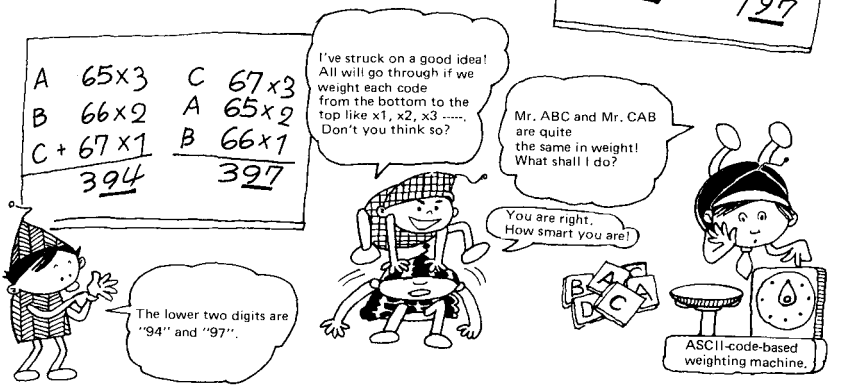

```
10 DIM N$(100), E(100)
20 PRINT "INPUT NAME AND SCORE."
30 PRINT "STOP, HOWEVER, IF NAME IS ZERO !"
40 FOR I = 1 TO 100
50 INPUT N$, E
60 IF N$ = "0" THEN 100
70 GOSUB 1000
80 N$(Y) = N$:E(Y) = E                    : Attention !
90 NEXT I                                   Stores at the array position determined from
100 PRINT "STORAGE COMPLETED"               the code of name directly.
110 PRINT "INPUT NAME TO BE RETRIEVED"
120 INPUT N$
130 GOSUB 1000
140 IF LEN(N$(Y)) = 0 THEN 160            : When nothing is stored, an empty string is present there.
150 PRINT N$(Y), E(Y):GOTO 110            : One-time access prints the name and score.
160 PRINT "NOT FOUND": GOTO 110
1000 L = LEN(N$):X = 0                    : ·Subroutine that decides the location to store
1010 FOR J = 1 TO L                         a name in, from its constituent letters.
1020 X = X + ASC (MID$(N$, J, 1))         ·Sets the sum of codes of those letters at X,
1030 NEXT J                                 or sets the remainder of X ÷ 100 at Y.
1040 Y = X-100*INT(X/100)
1050 RETURN
```

Then "Clash" comes into question.....

It is too early to say "I've got it" because the above-mentioned allocation method using codes translates different names – for example, "ABC" and "CAB" – into the identical code. This is the problem of "clash" that the code-based system inevitably poses.

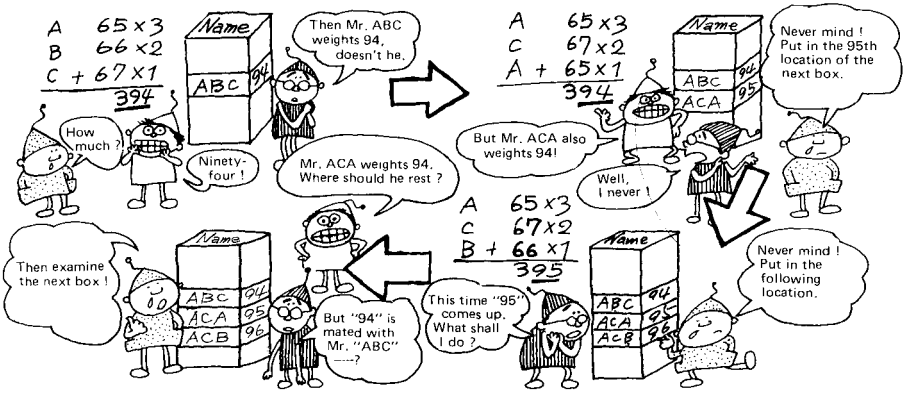
To this problem there are several solutions. One of them is to weight every code depending on where it is located.





Then, is everything O.K.? No, again. Considering possible variations of name, we cannot help thinking the "clash" to be probably inevitable. No matter what method of weighting is employed, a "clash" could occur. This is something like the fate of the code-based allocation. If the "clash" dogs us as something like our fate, we must take some proper measure just when it occurs.

A method called occasional storage is an example of such measures. Its principle is as cartooned below.

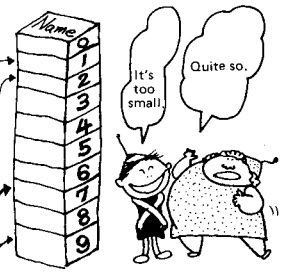


Providing ample storage locations reduces the occurrence of a "clash."

There is another important measure we should take in the code-based allocation. It is to declare storage locations adequate to store the expected amount of data when securing those locations first.

Let's take an extreme example in order to demonstrate the necessity of the above measure. In this example, only ten names' worth of storage locations are provided and the four names so far used are allocated with their codes weighted in a simple method.

F	70 x 4	B	66 x 4	S	83 x 5	J	74 x 5
O	79 x 3	U	85 x 3	M	77 x 4	O	79 x 4
R	82 x 2	S	83 x 2	I	73 x 3	N	78 x 3
D	+68 x 1	H	+72 x 1	T	84 x 2	E	69 x 2
	749		757	H	+72 x 1	S	+83 x 1
					1182		1141



As seen from the above cartoon, a clash occurs at once. That is, if the amount of data is small, the code-based allocation leads to unfavourable results. We cannot absolutely say how much data can be coded with good results or efficiently processed by the simple sequential storage method. Finally, we enumerate the essential hints on the code-based-allocation, and show a simple program that exemplifies the practice of those hints.

- Weight every code in the code-based allocation. (It is better to change the weighting coefficient depending on the item to be coded: for example, person's name, article name, phone number.)
- If a "clash" occurs, try to store data in succession.
- Declare storage locations with a margin allowed when securing them first — in other words, when estimating the amount of data).

```
10 PRINT "HOW MANY NAMES' WORTH OF DATA ARE PROCESSED ?"
20 INPUT H:H = INT(H):Z = 2*H
30 DIM N$(Z), E(Z)
40 PRINT "INPUT NAMES AND SCORES."
50 PRINT "STOPS, HOWEVER, IF NAME IS ZERO !"
60 FOR I = 1 TO H: INPUT N$, E
80 IF N$ = "0" THEN 120
90 GOSUB 1000
100 FOR K = Y TO Z
110 IF LEN (N$(K)) = 0 THEN 150
120 NEXT K
130 PRINT "FILLED": STOP
150 N$(K) = N$:E(K) = E
160 NEXT I
170 PRINT "STORAGE COMPLETED"
180 PRINT "INPUT NAME TO BE RETRIEVED"
190 INPUT N$: GOSUB 1000
210 IF LEN(N$(Y)) = 0 THEN 250
220 FOR I = Y TO Z
230 IF N$ = N$(I) THEN 260
240 NEXT I
250 PRINT "NOT FOUND":GOTO 180
260 PRINT N$(I), E(I):GOTO 180
1000 L = LEN(N$):X = 0
1010 FOR J = 1 TO L
1020 X = X + (L + 1 -J)*ASC (MID$ (N$, J, 1))
1030 NEXT J
1040 Y = X -Z*INT(X/Z)
1050 RETURN
```

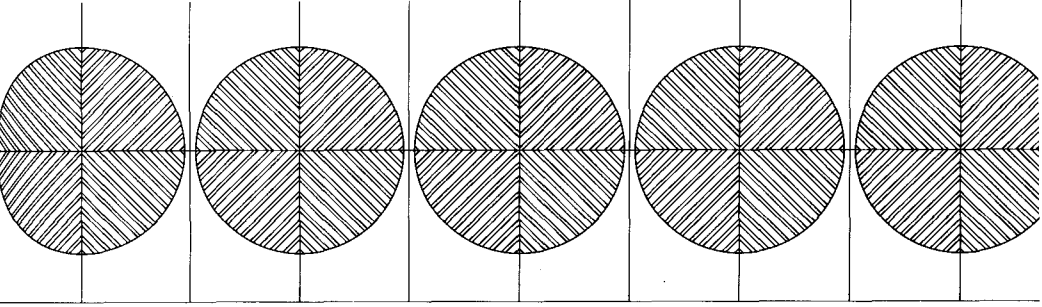
Secures storage location twice as many as the amount of data to be processed.

Checks that a location which has undergone allocation is already filled with another name. If it is found filled, sequential storage occurs. If found empty, it contains an empty string.

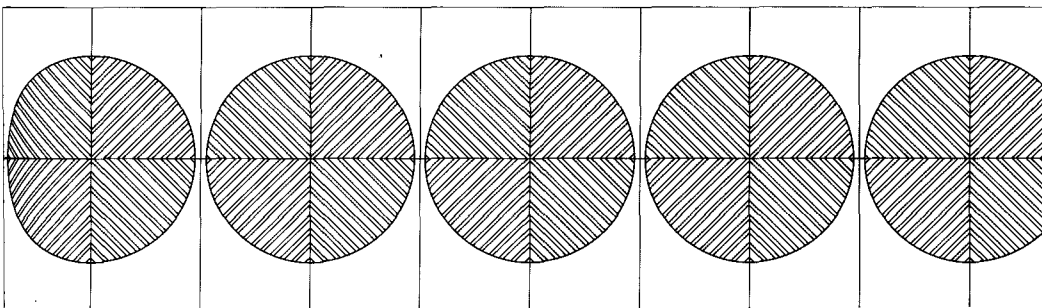
Checks that the location decided to be recalled is surely loaded with the identical name. If name in that location is different from the objective, then the program moves from there searching for the objective name, then jumps to line number 260 if that name is located.

: Weights every code and sets the sum of resulting weight at X.

: Sets the names of arrays to be allocated at Y.



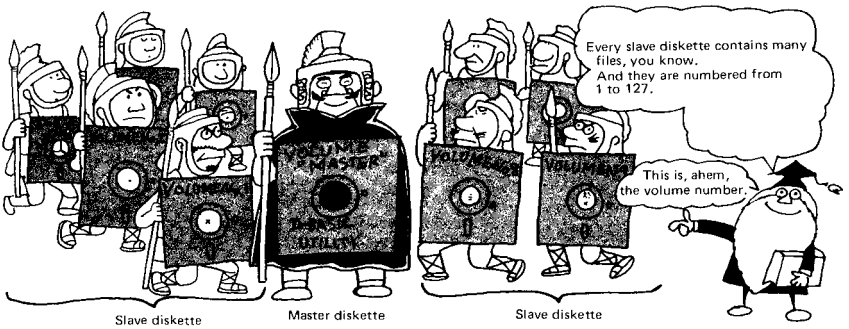
Chapter 2
Guide to Disk Basic Commands



Master diskettes and slave diskettes

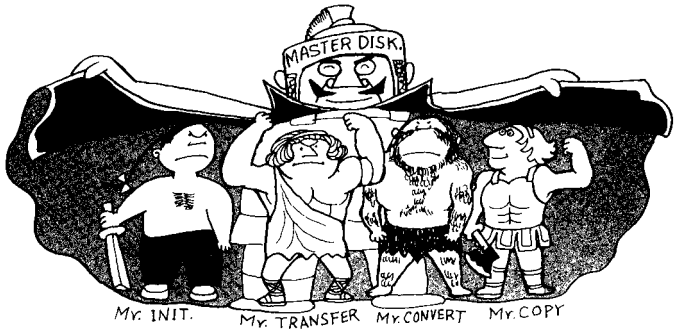
This chapter will introduce you to the practical use of the Disk Basic.

First of all, we explains the nomenclature of diskettes. Diskettes are available in two sorts: master diskette and slave diskette. These two sorts of diskettes are literally in the relation between master and slave. The diskette to containing the Disk Basic and its utility program file is called the master diskette, while the diskette to store a wide variety of files made up based on the Disk Basic is called a slave diskette.



Each individual diskette has its own volume number. The master diskette, unlike slave one, has a volume names "MASTER". Every slave diskette is given any one of the numbers from 1 to 127. It can be arbitrarily decided when initializing each diskette which number should be assigned to it, but for precise file control it is essential to specify a proper slave diskette volume. (For the practical method of initialization, consult page 176.)

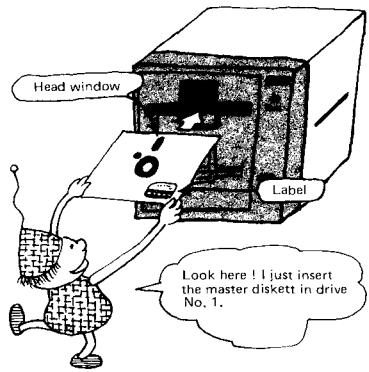
The master diskette contains the following softwares as utility program file: SLAVE DISKETTE INITIALIZE, FILING PROGRAM, DISK COPY, and BASIC SP-5025. Comparing the diskette to a firm, it can be said to be administrated by one president – the DISC BASIC – and four managers – the four utility program files.



Driving the Disk Basic

Now we start to use the Disk Basic. Are the computer and the disk drive unit properly connected? O.K.? Good! Then switch on the computer, interface unit, and disk drive unit and insert the master diskette into the drive. You may insert the diskette into either drive, No.1 or No.2. (If there are two drive units, any one of No.1 thru No.4 will do.) When inserting the diskette, however, use care not to put it in the wrong way.

You must insert the diskette with its labeled face up and its head window at the far side. (See the illustration at the right.)



When the diskette has been placed in the drive, execute the monitor command

FD [CR]

The computer then changes the control mode over to the disk control, asking you which drive is loaded with the master diskette as shown below.

Cursor ↓

BOOT DRIVE ?

Then input the correct drive number (any one of 1 through 4) and press [CR]. (When the drive number is 1, you need only to press [CR].)

If your master diskette is placed correctly in the specified disk drive, then the Disk Basic is read and control is left to it, thus starting the operation. (See photo 1.)

If the diskette is not in the right position, the computer provides a display of "CAN'T BOOT." (See photo 2.)

In this case check that:

- the diskette has been inserted correctly.
- the drive unit door is closed, and
- the drive unit is on.

Photo 1

Photo 2

Your first task is to consult the file list

It is a basic practice indispensable to successful file control to know what files a diskette contain. The command **DIR n** of the Disk Basic has the ability to list on the CRT screen the files recorded in a diskette — master or slave diskette — on the drive which is specified by **n**.

This system enables you to recall files, in the form of a file list, from a diskette to the CRT screen by the aid of the DIR command and find out the files in it just as you would consult a book or catalogue and locate your objective in the library. "DIR" is an abbreviation of "**directory**."

Let's command the computer at first to display the directory of the master diskette. Since this requires a direct execution instruction, give the following input directly through the corresponding keys.

[D] [I] [R] [Drive No.]

The drive number refers to the identification number of a drive that has been loaded with the master diskette and does not need to be entered when it is 1. (n = 1). Tapping on [CR] actuates the drive, providing a display of the directory as in photo 3. This shows the utility program files recorded on the master diskette.

Photo 3

The displayed directory gives the following information.

(1) Diskette volume number
(2) Total of sectors not used
(3) Mode, state, and name of each recorded file.

The file mode is identified by the following four codes.

BTX ..... BASIC text file
BSD ..... BASIC sequential access file
BRD ..... BASIC random access file
OBJ ..... object file (machine-language program, data)

The recording state tells you whether your desired file is locked or not. If the file is locked,"*" appears following the mode, and if not locked, it permits writing, deletion, and changing of file name.

The file name refers literally to the name of a file; it is designated when each file is recorded.

When many files are present on a diskette, the system provides the directory display that reveals one frame of the file list, awaiting the next command with the flickering cursor. Tapping then on [CR] compels the machine to provide a directory display of further file names, but the computer is also capable of moving to another command.

If an optional printer is connected to the system, a hard copy of the directory can be obtained with **DIR n/P** command. The hard copy bears, in addition to directory information, the number of sectors every file uses.

Loading Basic texts on the computer and saving them on a diskette

We will from now practice the operation to load BASIC texts on the computer or save them on the diskette as the initial step to the disk control of the Disk Basic. That is, we will perform the same operation for a diskette that has conventionally been made for a cassette tape.

In principle, the LOAD instruction to read BASIC texts from a diskette and the SAVE instruction to write the said texts on a diskette have the following forms.

LOAD FDd@v, "file name"

This form of LOAD instruction commands the system to load the computer with a BASIC text — its file name is "file name" — which is recorded on a diskette — which is given a volume number of v (v = 1 to 127) — in a drive numbered d (d = 1 to 4).

SAVE FDd@v, "file name"

This form of SAVE instruction commands the system to save the BASIC text present in the computer on a diskette — which is identified by volume number of v (v = 1 to 127) — a drive numbered d (d = 1 to 4). However, the file name of this BASIC text must be specified by "file name," and the file name is composed of up to 16 letters.

The reason for addition of the expression "In principle" in the above description is that drive number d and volume number v may not need to be specified as in the following cases.

(1) When you perform saving or loading for a diskette in the drive that has executed the DIR command latest, you have only to specify "file name" alone. Then the identical drive and diskette will be subjected to saving or loading. (Provided the diskette remains in the same drive.)

(2) When you practice saving or loading for a diskette in a specified drive, you do not need to specify its volume number "@v" no matter what it is.

Let's try to load a sample program.

At first, we will try to load and execute a familiar sample program recorded on the master diskette so that you easily understand the practice of program loading.

Insert the master diskette into drive 1 and execute the DIR command. Then the directory of recorded files appears in the CRT display, you will notice by their file names headed by "BTX" that four files are the identical application programs with those used in the descriptions of the tape-mode Basic.

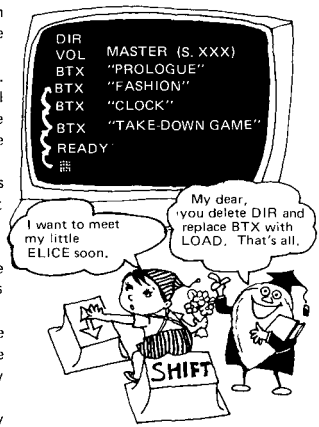
The digital clock program, for example, is recorded with its file name as * "CLOCK." To load the computer with this text you have only to input this:

LOAD "CLOCK" CR

because the (1) above holds true of this case. If CR is pressed, then the drive starts to move. On the completion of loading the computer displays "READY" and awaits the RUN command.

The above LOAD command can be more easily executed if the directory display function is used. In detail, you have only to manipulate the computer like this: shift the cursor to the line of a directory display that bears **BTX "CLOCK",** replace BTX with LOAD and press CR.

A knack of achieving successful file control is to manage the directory display function.



* (Note)
Application programs in your "Master" diskette may have file names "1", "2", "3" and "4".
In this case, the file names "1", "2", "3" and "4" respectively correspond to the programs "PROLOGUE", "FASHION", "CLOCK" and "TAKE-DOWN GAME".

Let's save programs next!

This time, we will teach you the method of saving a BASIC text on a diskette. Prepare a slave diskette. (New diskette cannot be used as slave ones before they are initialized. Consult the method of initialization on page 176.)

Input some short program through the keyboard. Only one line of PRINT instruction will do, for example. It is a complete program, too.

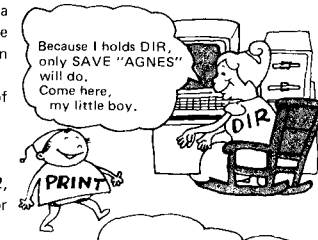
```
10 PRINT "I AM AGNES": END
```

We try to save this on a slave diskette.

In order to save the program on a diskette numbered 15 in drive No. 2, for instance, it is necessary to program the computer as follows, for example, with its file name as "AGNES".

```
SAVE FD2@15, "AGNES"
```

The light of drive No. 2 flashes and program writing begins. When writing is completed, execute DIR2. This will reveals that a BTX file named "AGNES" has been recorded. Then clear the BASIC text area and load this text.

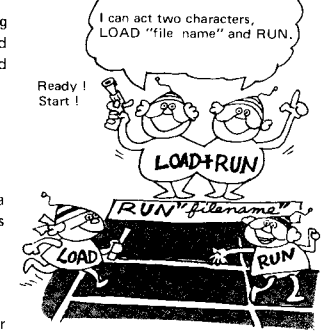


To load and then execute at once is the duty of RUN "file name."

It is the RUN "file name" command that orders the system to load a BASIC text and execute it immediately. This command, like LOAD, has the following general form.

RUN FDd@v, "file name"

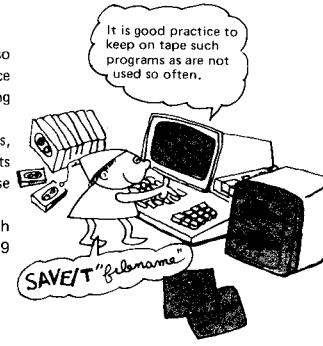
The command permits the omission of drive number and volume number under the same conditions as the LOAD does.



Who is it that is sighing, "What shall I do with the programs I have so far recorded on my cassette tapes?"? Take it easy! We will soon introduce you to a LOAD or SAVE instruction that be capable of loading or saving those programs.

The instructions LOAD and SAVE you want are available in the forms, LOAD/T "file name" and SAVE/T "file name". If you read BASIC texts from your cassette and save them on slave diskettes with these instructions, then you will enjoy quick reading in the future.

The LOAD "file name" and RUN "file name" in Disk BASIC are both applicable to object files. For its design and application, consult pages 39 and 56.



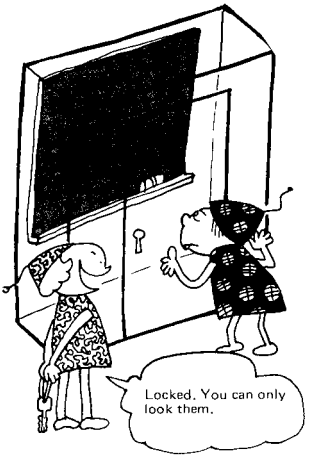
To lock, unlock, delete or rename files

This lesson summarizes commands that are individually responsible for locking, unlocking deleting, and renaming files recorded on diskettes. The command to lock a file is **LOCK**, and the one to release a locked file is **UNLOCK**. So easy to understand aren't they! And the command to rename is **RENAME**, then the last one is **DELETE**. This deletes files. Any one of these commands can be used during programming not only as a direct instruction, but also as a statement.

LOCK

This command locks recorded files just as its label implies. If a file is locked, the computer refuses to accept DELETE or RENAME command. Also, the computer refuses to load new data if a random access file (BRD) is locked. Therefore, any file, if locked, is fixed on the diskette where it stays, refusing any request of the computer.

The names of locked files show up with their file mode symbols followed by asterisks when they are recalled in the form of directory display. Do you remember that the utility programs' names of the master diskette appeared together with asterisks.



The general form of the LOCK command is:

LOCK FDd, "file name"

The form, without FDd – drive number, is executed on the diskette that was last used to execute DIR.

Here, let's try to lock a BASIC text file "AGNES" that has been made up with the SAVE command.

LOCK FD2, "AGNES"

This file, when recalled in the directory display, should appear accompanied by an asterisk as follows.

BTX*"AGNES"

UNLOCK

This is the command to cancel the file lock, and its general form is as given below.

UNLOCK FDd, "file name"

Now, try to free the BTX file "AGNES" locked above. If the file remains in the same drive after the execution of DIR, the form of UNLOCK command is simplified as shown below.

UNLOCK "AGNES"

RENAME

The general form of the RENAME command is

RENAME FDd, "file name 1", "file name 2"

and it renames a file present on the diskette that is in drive d, that is, changing its file name from "file name 1" to "file name 2." Now, we rename, for example, the file "AGNES" to name it "ELICE" by inputting the RENAME command — —

RENAME "AGNES", "ELICE"

— following the UNLOCK command. What is the result? Ascertain, by inputting LOAD "ELICE", that the contents of the renamed file — "AGNES" in the past, and now "ELICE" — remains unchanged.

DELETE

This command deletes files from diskettes and is generally used in the form:

DELETE FDd, "file name"

Let's execute the command right now. We try now to delete the file "ELICE", the renamed one. If the simple form can be used, inputting

DELETE "ELICE"

will delete that file. Command the display to show the directory and make certain the file "ELICE" has been deleted.

If locked files are given DELETE or RENAME command, the directory display only shows

Photo 4

*ER 46

shown in Photo 4,and the system does not execute those commands. Open your book at page 173, and you will find there "DISK BASIC SP-6015 ERROR LIST." Or look at the appendix "DISK BASIC ERROR LIST AND FILE COMMANDS." The display in Photo 4 tells us that an error numbered 46 occurs, namely, that the DELETE or RENAME command is inoperative for "write protect file = locked file."

The Disk Basic, when an error occurs, forces in this way the computer display to show the corresponding error number and stops the operation; also, as mentioned on page 52 and the following, it is then capable of jumping within the program under processing to an appropriate error correction routine without stopping its execution.

[Exercise 1]

Place in drive 2 a diskette that contains BTX file "ELICE," and let the computer display show the relating directory. Insert another slave diskette in place of the said one and input LOCK "ELICE" or DELETE "ELICE." This causes, needless to say, an error. What error number appears on the display?

(Ans.) 40, no file

[Exercise 2]

Cover the write protect noth of the diskette carrying the file "ELICE" with a write protect seal, and carry out deletion. What is the result?

(Ans.) Error No. 50 occurs. When a write protect seal is glued on a diskette, the whole diskette refuses writing, deletion, or other commands irrespective of whether the objective file is locked or not. (Gluing a write protect seal has the same effect on a diskette as breaking the tabs does on a cassette.)

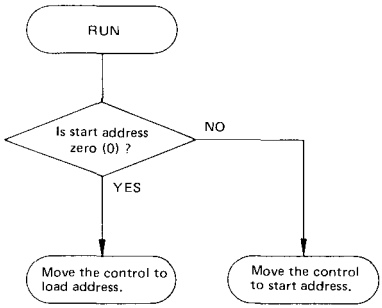
Running machine-language files(OBJ)

If the file mode is BTX, then the program execution with an input of RUN "file name" is under the management of the Disk Basic. Accordingly, the starting and excecution of programs are completely controlled by the Disk Basic. Meanwhile, **if the file mode is OBJ, then RUN "file name" basically depends on the specified file.** Hereunder are given two instructions you must especially note when using the RUN "file name" in this file mode.

Starting programs

Any program has a load address and start address without exception. The load address indicates where the program should be loaded, while the start address where the program execution should be started. The illustration at the right reveals how RUN "file name" uses these two addresses.

OBJ file, made up of "MACHINE LANGUAGE" and "SYMBOLIC DEBUGGER" has a start address of zero (0). Accordingly, its program execution is automatically started at the location its load address indicates. Hence, if its start address is expected to be zero (0), then the program must be made so that its execution should start at the location its load address indicates. Otherwise, illogical program running could occur.



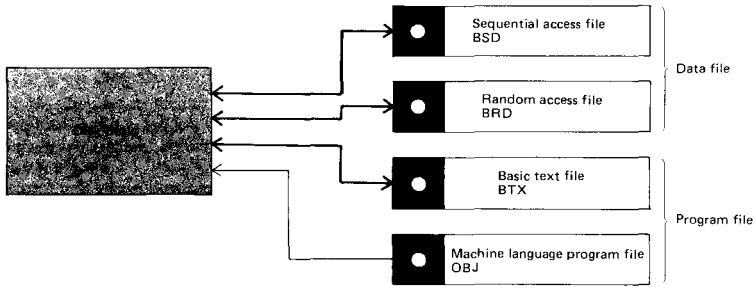
Breaking operation and return to monitor

If the file mode is OBJ, then the program execution depends in principle upon specified files, freed from the control of the Disk Basic. In this mode, therefore, specified programs, unless they are designed to accept BREAK operation and monitor return instructions, cannot be run in the same manner as the Disk Basic.

Starting the data file control

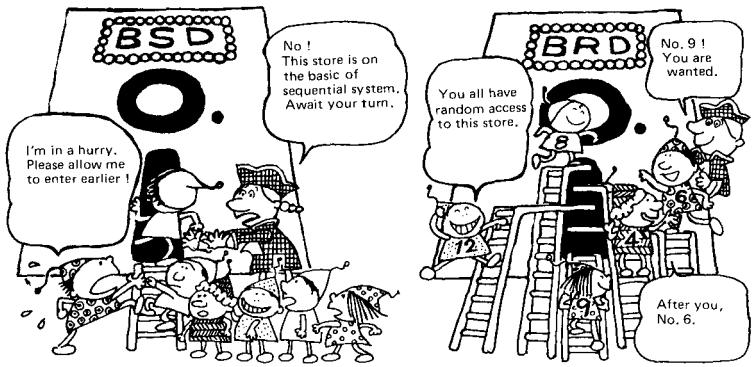
Now, we have arrived at the heart of our DISK BASIC. Yes. We will from now discuss the handling and practical control of various data and program files that make use of the large capacity and high-speed access function of our floppy disk unit.

We have already revealed in the discussion of files that the Disk Basic is capable of handling three kinds of files: two data files – the sequential access file (BSD), random access file (BRD) – plus one program file – the BASIC text (BTX). One more file, the machine language program file (OBJ), has been constructed using systems program or machine language and recorded on a diskette. This file is intended to be singly run or used linked with the BASIC text in the Basic machine language area; hence, the Disk Basic can utilize it, but cannot write it or change its contents.



In discussing individual file control commands, we will first explain the features of the way to construct and use two kinds of data files and second explain the way of using the file commands, CHAIN and SWAP. The method of linking machine language program file and Basic text will be demonstrated in the paragraph – on page 56 – that handles the USR function.

Departure for two sorts of data file controls.



Control of sequential access files

The sequential access file refers to a data file of which the data is recorded or fetched by the sequential access method, which accesses data sequentially from the beginning.

You already know the way to form data files on a cassette file by using Series SP-5000 Basic, don't you. Sequential access with the Disk Basic is the same with that way, except that its medium is not a cassette but a diskette. And the method is of course far more practical and provides high speed access, enabling more versatile file control through several new file control commands.

First, let's compare the differences between Disk Basic and cassette-based Basic sequential access commands.

Recording files (writing data)

	Disk Basic	Cassette-based Basic
File open command	WOPEN #n, "file name"	WOPEN "file name"
Data write command	PRINT #n, data	PRINT/T data
File close command	CLOSE #n	CLOSE
Cancel command	KILL #n	—

Recalling files (reading data)

	Disk Basic	Cassette-based Basic
File open command	ROPEN #n, "file name"	ROPEN "file name"
Data read command	INPUT #n, variable	INPUT/T variable
File close command	CLOSE #n	CLOSE
File end detection	IF EOF(#n) THEN	—

Note:

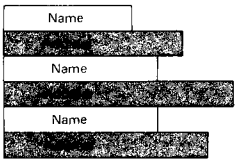
The general form of a file open command includes drive number and diskette volume number to be designated. The above tables, however, do not show them.

As you can see from the above tables, every command of these two Basics nearly corresponds to each other in the composition. Did you notice that each Disk Basic command always contains an unfamiliar symbol "#n"? This is called a logical number, which must be specified whenever a Disk-Basic-based file access is executed.

The cassette-based Basic permits only either writing to or reading from one file, whereas the Disk Basic, utilizing the merits of disk system that multiple files are arranged in parallel and can be arbitrarily accessed, can control a multiple number of files **(maximum of 10 files)** simultaneously. Besides, it is so devised as to define opened files by logical numbers chosen arbitrarily, making it unnecessary to write their file names using corresponding logical numbers every time you want to specify your desired files thereafter.

(Such difference can be said to be referred to the difference between hardware, cassette and disk, namely, that while the cassette is of sequential access type, the disk is of random access type.)

As a simple example of sequential access file control, let's discuss the recording of the names and addresses of person's homes on a sequential access file. Our file, a class list in this example, must be made in the following form.



file name = "3-H, CLASS LIST"

The reason why the above rectangles have not the same length is that data recorded by the sequential access method are not fixed in length. In the random access file, we will later mention, all the data is given the same fixed length of 16 bytes. When all the data is handled in the block as in this example – class list of the 3rd grade, class H – or when most of the data – addresses in this example – are too long to be recorded in 16 bytes and are not fixed in length, the sequential access file is more suitable than the random access file.

Shown below is the method of constructing a program to cause the system to behave as follows: substitute string variables alternately with names and addresses with INPUT statement, record a combination of name and address one by one to make "3-H, CLASS LIST" bearing 50 combinations in all, then read stored data out of the file (list) and display them on the CRT screen by 10 combinations.

(Writing)

```
100 WOPEN #3, FD1@22, "3–H, CLASS LIST"
110 FOR P = 1 TO 50
120 INPUT "NAME="; NA$
130 INPUT "ADDRESS = " ; AD$
140 PRINT #3, NA$, AD$
150 NEXT P
160 CLOSE #3
```

(Reading)

```
200 ROPEN #4, FD1@22, "3–H, CLASS LIST"
210 FOR P = 1 TO 5: FOR Q = 1 TO 10
220 INPUT #4, NA$, AD$
230 PRINT NA$ : PRINT AD$
240 NEXT Q
250 PRINT "STRIKE ANYKEY"
260 GET X$ : IF X$ = " " THEN 260
270 NEXT P
280 PRINT "END"
290 CLOSE #4
```

The data are stored by this program in the slave diskette, volume number 22, that is in drive 1. The meaning of every command used in this program will be described on the next and subsequent pages.

Now, we will describe the sequential access commands along with the simple sequential access file control program we have shown on the preceding page. The description begins with the command to record data on files.

WOPEN #n, FDd @v, "file name".....Write open

The WOPEN command specifies the name of a sequential access file to be loaded with data, defines that file as an arbitrary logical number (between 1 and 127), and opens it to be ready for writing. "To define a file as a logical number" means to declare a file to be shown by a logical number.

In our program, the name of the sequential access file to be made with statement No. 100 is specified as "3-H, CLASS LIST" and that file is defined as logical number #3. And the file is declared to be formed on the diskette with volume number 22 that is in drive 1.

Notice:

The designation of drive number and volume number of the objective slave diskette – FDd@v – can be omitted in the same way as for the LOAD, and SAVE commands.

Major error indication

*ER 42	Indicates that the file already exists.
*ER 54	Indicates that the objective diskette is not initialized.
*ER 64	Indicates that the use of a logical number out of range (1-127) has been tried.

PRINT #n, data

This command sequentially makes data arrays in order to record data in the sequential access file defined as a logical number #n and being in a write-open condition. Accordingly, **every execution of "PRINT #n, data" statement adds one item of data to the data array.** Then, however, an actual sequential file is not made; the data array is not recorded in the sequential access file before the CLOSE statement is executed. When recording data in the block, they can be written punctuated with commas as follows.

PRINT #n, data, data, data, -----

This is the PRINT # statement No. 140 in the sample program shown above.

CLOSE #n

This command formally records the data array prepared with the execution of the PRINT # statement as a string of data directly in the file. A sequential access file (BSD) is formed before the execution of this command.

Notice:

CLOSE statement not accompanied by logical number #n closes all of the file currently open.

KILL #n

This command, not used in the preceding sample program, cancels the WOPEN command. The execution of KILL # statement, instead of CLOSE # statement, cancels the WOPEN, prevents the data array, even if it is under construction, from being recorded in the sequential access file. This command is practical if the need for cancellation occurs during the construction of a sequential data array.

The descriptions of sequential access file call commands follow.

ROPEN #n, FDd @v, "file name".....Read open

This command, by contrast to the WOPEN command, designates a file name to read data from a sequential access file, defines that name as an arbitrary logical number (between 1 and 127) and opens that file to read its data.

In our sample program, this command, used in statement No. 200, specifies the sequential access file "3-H, CLASS LIST" and defines it as logical number #4. Also, the command also declares that the above file should be read from the slave diskette no. 22 in drive no. 1.

Noitce:

The designation of drive number and volume number may be omitted when the command is used in the same way as describing the WOPEN command.

Major error indications

*ER 40	Indicates that application of the command has been attempted to a nonexistent file.
*ER 43	Indicates that the file is already open.

INPUT #n, variable

This command reads data in sequence from the beginning one by one from a read-open sequential access file defined as logical number #n and substitutes variables by them.

In our sample program, this command, used in the statement no. 220, sequentially substitutes NA$ and AD$ with the data from the sequential access file "3-H, CLASS LIST." Such a form as

INPUT #n, variable, variable, variable, ------

multiple variables being arranged with commas between them, enables one statement to read multiple items of data.

Major error indications

*ER 4	Indicates that the forms of variables do not coincide, for example, a string is being read in the form of numerical variable.

CLOSE #n

With the ROPEN command given, the CLOSE #n statement completes the ROPEN action and returns logical number #n to its original undefined state.

Note:

The KILL statement can, under the ROPEN state, be used in the same way as the CLOSE statement, and has the following additional advantage.

The KILL statement, unlike CLOSE, does not activate an access to the floppy disk unit. Therefore, the statement can prevent the occurrence of further errors if used to close a file by the correction routine using the error correction command, for example, when the open command has caused "NOT READY" – error number 50.

How to detect the file end

What is the result when data is read beyond the number of recorded items? In this case, no error occurs and the variables are set at 0 or "empty", then a special function, EOF (#n), detects the file end. **The EOF (#n), when data is read from a file with the INPUT# command, is put in the condition "true" if it comes to the end of that file.** Hence, if the statement

IF EOF (#n) THEN

is placed behind INPUT# statement, instructions behind THEN are executed when the EOF (#n) is true, namely, the file end is detected.

[Problem]

By modifying the sample program on page 39, make a program to read data from the file "3-H, CLASS LIST" by ten combinations of name and address and display them. The number of recorded persons is, however, unknown.

[Example of solution]

The solution would be for example as follows.

```
300 ROPEN #5, FD1@22, "3-H, CLASS LIST"
310 FOR I=1 TO 10
320 INPUT #5, NA$, AD$
330 IF EOF (#5) THEN 400
340 PRINT NA$: PRINT AD$
350 NEXT I
360 PRINT "STRIKE ANYKEY"
370 GET X$: IF X$="" THEN 370
380 GOTO 310
400 CLOSE #5
410 PRINT "FILE END" : END
```

Exercise

[Problem]

Record names and addresses present in the BSD file "3-H, CLASS LIST" separately on the BSD file for name alone and the one for address alone.

[Example of solution]

```
500 ROPEN #6, FD1@22, "3-H, CLASS LIST"
510 WOPEN #7, FD1@22, "NAME"
520 WOPEN #8, FD1@22, "ADDRESS"
530 INPUT #6, NA$, AD$
540 IF EOF (#6) THEN 600
550 PRINT #7, NA$
560 PRINT #8, AD$
570 GOTO 530
600 CLOSE
610 END
```

[Problem]

Record strings entered through keys using INPUT statement on a BSD file. However, "CLOSE" must be input to close the file, and "KILL" to cancel it.

[Example of solution]

```
100 WOPEN #30, "SEQ DATA"
110 INPUT "DATA=" ; A$
120 IF A$="CLOSE" THEN CLOSE #30 : END
130 IF A$="KILL" THEN KILL #30 : END
140 PRINT #30, A$ : GOTO 110
```

Control of random access file

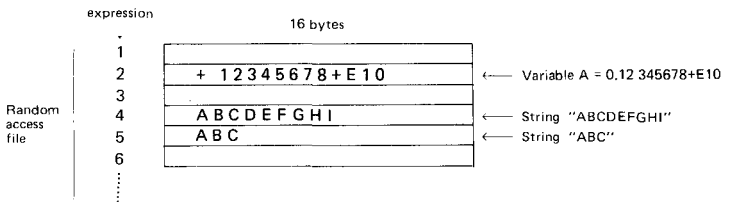
The random access file refers to a data file that permits data to be recorded or called by the "random access" method. The term **"random access"** means the process of recording or calling each item data by specifying it as an array element. Unlike the sequential access file, this file permits random access of any data elements included in a collection of data.

The PRINT# and INPUT# statements used as random access commands contain "expression" – which specifies array elements – that follows the logical number as shown below because the random access file requires to designate proper array elements for a collection of data of which it is made up.

PRINT #n **(expression)**, data
INPUT #n **(expression)**, variable
↑
Designation of array element

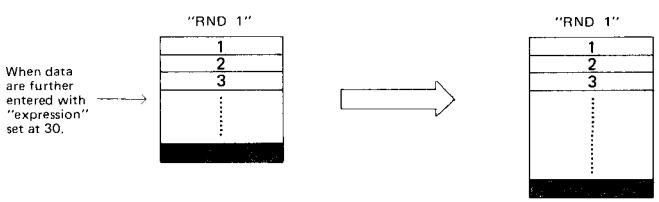
The **"expression"** must be given as a numerical value or variable. The statement
INPUT #7 (21), A$
for example, commands the system to read as string variable A$ the data that is being recorded as the 21st element of a group of data present on the random access file open to logical number #7.

You should notice that such a random data access is conditioned on that every data item is recorded in a fixed length. In other words, the random access file requires to record any numeric and string variables in less than 16 bytes inclusive.



Numeric variables, even the ones expressed by exponential notation, do not exceed 16 bytes without exception, whereas string variables, which may be extended up to 255 bytes, often exceed 16 bytes, when they cannot be recorded in one data element of the random access file.

Another difference of this file from the sequential access file is in that it can be expanded by updating the file after it has been initially created. Provided there is a random access file "RND 1" that has been recorded using "expression" up to 20, for instance, that file expands large enough to accomodate 30 "boxes" when data is newly entered with "expression" set at 30.



Now, let's try to devise a program for making a simple inventory by using the random access file. It is supposed that individual articles are given fixed serial numbers from No. 1 to No. 50 and the inventory has five items: article name, unit price, number of stock, amount (unit price X number of stocks), and comments.

When recording inventory data for each article, its serial number must be entered first.

Recording inventory data

```
100 XOPEN #5, "STORE LIST"
110 INPUT "ARTICLE NO. ="
120 IF K = 0 THEN 300
130 INPUT "ARTICLE NAME =" ; N$
140 INPUT "UNIT PRICE =" ; P
150 INPUT "NO. OF STOCK = " ; S
160 INPUT "COMMENTS =" ; C$
170 T = P*S
180 PRINT #15 (K*5–4), N$, P, S, T, C$
190 GOTO 110
300 CLOSE #5
310 END
```

The random access file made according to the above program is as follows. If article number is set at K = 12, five kinds of data are entered into array elements whose expression individually corresponds to 56 through 60.

expression			
	55		
K*5–4	56	N$:	article name
K = 12	57	P:	unit price
	58	S:	number of stock
	59	T:	amount
	60	C$:	comments
	61		

BRD file "STORE LIST"

In this way, data can be arbitrarily arrayed on the file. Hence, the file, unlike the sequential access file that is filled with data in succession, may exhibit empty locations, providing for simple data rewriting. Next, let's devise a program to call the random access file "STORE LIST" made as shown above and read the inventory data of a certain article.

Calling inventory data

```
500 XOPEN #17, "STORE LIST"
510 INPUT "ARTICLE NO. = " ; J: IF J = 0 THEN 700
520 INPUT #17 (J*5–4), N$, P, S, T, C$
530 PRINT "NO." ; J : PRINT "ARTICLE NAME:" ; N$
540 PRINT "UNIT PRICE:" ; P
550 PRINT "NO. OF STOCK:" ; S
560 PRINT "AMOUNT:" ; T
570 PRINT "COMMENTS:" ; C$
580 GOTO 510
700 CLOSE #17
710 END
```

In this way, that random access file enables the inventory data of desired articles to be called at once by inputting their article numbers, no matter how many articles are inventoried.

Described below are the details of each command, refering to the sample random access file control programs shown on the preceding page.

XOPEN #n, FDd @v, "file name".....Cross open

The XOPEN command specifies the names of random access files to write data to or read data from, defining them as arbitrary logical numbers (between 1 and 127) to open those files. In the access to random access files, "logical open" for writing is not differentiated from that for reading. Letter "X" (= cross) of this command implies this fact.

In the preceding two sample programs, statements nos. 100 and 500 cross-open the random access file "STORE LIST" by defining it as a logical number.

The part "FDd@v" of this command can be omitted when put in the specified conditions. Since this part is omitted in the sample programs, this command works on the diskette in the drive that has processed the DIR command latest.

XOPEN	**"Logical open" for file writing**
(Cross open)	**"Logical open" for file reading**

Major error indications

✳ER 43	Indicates that an attempt has been made to open a file, that is already open.
✳ER 63	Indicates file mode error, for example, a sequential access file has been crossopened.
✳ER 64	Indicates that other logical numbers than specified has been used.

PRINT #n(expression), data

This command writes data as data elements that correspond to (expression) stored in a random access file defined as a logical number #n to be cross-open.

In the sample programs, the command, used in statement no. 180, writes 5 pieces of data – N$, P, S, T, C$ – in sequence on the file, starting the array element of expression = K ✳ 5–4. As seen from this example, the command can be written, in order to record data in succession, in the following form.

PRINT #n(expression), data, data, data, data, -----

This command carries out data recording on random access files. (The sequential access file is, you already know, so designed that data blocks (data arrays) composed of PRINT# statements are not formally recorded as file data before CLOSE# statements are entered.)

Note:

While the array variable of the Basic requires a DIM declaration in order to decide the maximum number of arrays, the random access file does not need such a declaration; the increase in the value specified by (expression) of PRINT# statement leads to the extension of the data area or file.

Besides, as we have so far explained many times, data of more than 16 bytes cannot be recorded in one element in the case of random access file because the file requires each data item to have a fixed length of 16 bytes.

Major error indications

✳ER 46	Indicates that data has been written to a locked random access file.

INPUT #n(expression), variable

This command reads data element corresponding to (expression) as a variable, from a random access file defined as logical number #n. In the sample programs, the command, used in statement no. 520, successively reads five items of data as variables N$, P, S, T, and C$, starting with the data of "expression = J*5 – 4".

This command statement, like PRINT# statement, can be written by placing commas between variables as shown below when multiple successive data is desired to be accessed by one statement.

INPUT #n (expression), variable, variable, variable, -----

Notice:

If INPUT# statement is executed for an empty element, a numeric variable is substituted by zero (0) and string variable by 16 spaces.

Major error indication

*ER 4	Indicates the disaccordance of variables form, for example, data that contains any string other than numeric value are compelled to be read as numeric variable.

CLOSE #n

This command terminates the "logical open" state defined by the XOPEN, accomplishing the formal recording of a random access file, and returns its logical number to the original undefined state.

Note:

In the random access file control, the KILL# statement has the duty to cease the "cross open" state.

Method of using the EOF(#n)

The EOF (#n) can be used to detect the file end just as in the case of the sequential access file.

The program shown below is intended to display the recorded data of the random access file "STORE LIST" on the CRT screen.

```
700 XOPEN #20, "STORE LIST"
710 K = 1
720 INPUT #20 (K*5–4), N$, P, S, T, C$
730 IF EOF (#20) THEN 900
740 PRINT "ARTICLE NO.:" ; K
750 PRINT "ARTICLE NAME:" ; N$
760 PRINT "UNIT PRICE:" ; P
770 PRINT "NUMBER OF STOCK:" ; S
780 PRINT "AMOUNT:" ; T
790 PRINT "COMMENTS:" ; C$
800 K = K + 1 : GOTO 720
900 CLOSE #20
910 PRINT "FILE END HERE" : END
```

Exercise

Solve the following two problems of the sequential access file (BSD) and random access file (BRD).

[Problem]

Count the total number of bytes for all data recorded on some sequential access file, and determine the number of sectors occupied on a diskette.

[Example of solution]

Read data as string variables and use the LEN command as follows. Then the required number of bytes and number of sectors can be calculated. (1 sector = 128 bytes)

```
100 ROPEN #50, "SEQ DATA"
110 INPUT #50, A$
120 IF EOF (#50) THEN 200
130 A = A + LEN(A$)
140 GOTO 110
200 CLOSE #50
210 PRINT A; "BYTES"
220 S = A/128: IF S < > INT (S) THEN S = S + 1
230 PRINT S; "SECTORS"
240 END
```

The directory display — which occurs with the DIR/P command — to transferred to the printer shows the number of sectors each file occupies. Compare it with the number of used sectors that has been computed according to this program, provided that the BSD file is used.

[Problem]

Rerecord numeric data of some sequential access file in the reverse order. Variables accompanied by labels must not be used, however.

[Example of solution]

Being within 16 bytes, each numeric data can be recorded directly on a random access file. The program below uses a random access file.

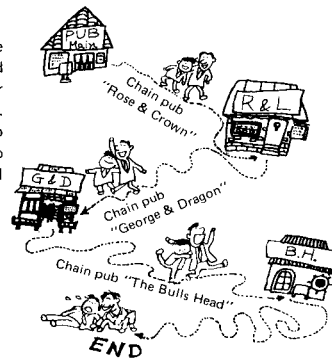
```
100 ROPEN #70, "SEQ DATA"
110 XOPEN #71, "SEQ DATA/RND" : J = 1
120 INPUT #70, A
130 IF EOF (#70) THEN 160
140 PRINT #71(J), A
150 J = J + 1 : GOTO 120
160 CLOSE #70
170 DELETE "SEQ DATA"
180 WOPEN #72, "SEQ DATA"
190 J = J — 1 : IF J = 0 THEN 300
200 INPUT #71(J), B
210 PRINT #72, B
220 GOTO 190
300 CLOSE : END
```

Making a chain of programs

The topic of this section is two program file control commands. These are the CHAIN and SWAP commands. When some programs are recorded on a diskette, the use of these commands enable you to call another program while running the recorded programs and move the control to it. In detail, the CHAIN command enables you to connect any program to the ones recorded on a diskette, and the SWAP command enables you to call any program in the form of subroutine. First is described the CHAIN command to connect or join programs.

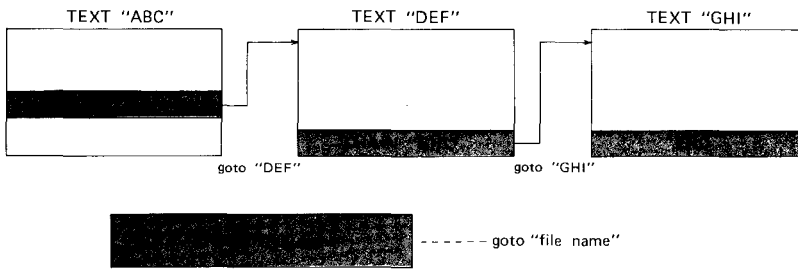
The form of the CHAIN command is as follows.

CHAIN FD1@50, "TEXT 2"



This statement commands the system to clear a program then present in the text area (it, however, keeps the values of variables), overlay that area with the text named "TEXT 2" that is recorded on the diskette of volume number 50 present in drive 1 and move control to the head of that text. The execution of this text frees the system from the control of the then running Basic text and compels it to read the text "TEXT 2" anew, moving control to its head. **When two programs are connected, the values of variables and the function defined by the DEF FN** in the original program are kept.

The function of the CHAIN command can be grasped as one of **"GOTO"** statement.

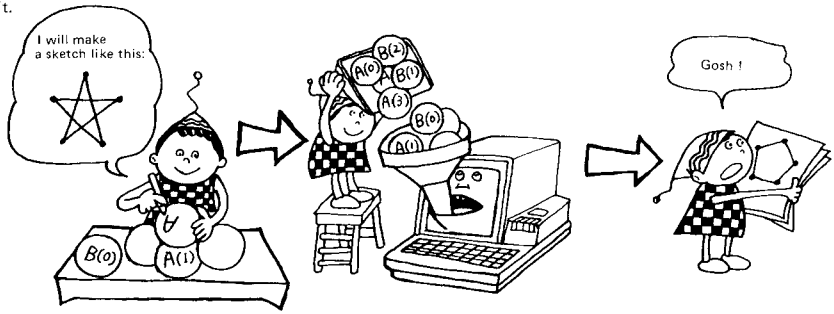


The use of the CHAIN command enables you to process such a huge program as to overflow the Basic text area by dividing it into pieces and then uniting them again as illustrated above. That is, the CHAIN command joins component programs every time they are processed. Therefore, the command and the SWAP command we will later refer to can be said to be an indispensable aid in coping with complicated, versatile data processing in small businesses.

Apart from such a sophisticated application, it is quite exciting and interesting to join various texts on a diskette. The Disk Basic, as seen from this, has an original world – which cannot be created by the conventional Basic – in that enables programs to extend themselves.

[Exercise]

Let's try one-stroke drawing by the aid of the computer. First we input the values of X- and Y-coordinates — the number of those coordinates must be less than 50. Because nothing appears on the computer display during inputting data, we must keep the input data in mind exactly. Otherwise, an unexpected letter or sketch could be input. When all data has been completely entered, we output it to the display in the same order as we have input it.



The program is composed of a routine to gather array variables in A (50) and B (50), and a routine to make a sketch, these two routines being connected through the CHAIN command.

Data A (K) and B (K) or X- and Y-coordinates are input by statments nos. 140 and 150. If 99 is then input as A(K), data is taken to have been completely entered, and the two routines are joined. The SET X, Y instruction cannot specify any other values than these: 0 through 79 for the X-coordinate, 0 through 49 for the Y-coordinate. Therefore, the program inhibits inputting any values out of range by means of statement no. 160. When the program moves through the CHAIN command to the drawing routine, it continues one-stroke drawing until some key is input.

[Program present in the text area]

```
100 DIM A (50), B (50)
110 PRINT " ■    ＊＊ ONE-STROKE DRAWING ＊＊"
120 PRINT " ■■■■■ DATA IN"
125 FOR K = 1 TO 50
130 PRINT "NO." ; K
140 INPUT "X =" ; A (K) : IF A (K) = 99 THEN 180
150 INPUT "Y =" ; B (K)
160 IF (A (K) < 0) + (A (K) > 79) + (B (K) < 0) + (B (K) > 49) THEN 130
170 NEXT K
180 CHAIN FD1, "GRAPH"
```

[Program file "GRAPH"]

```
100 PRINT "■" : K1 = K − 1
110 FOR J = 2 TO K1
120 X1 = A (J) −A (J−1)
130 Y1 = B(J)−B (J−1)
135 IF X1 = 0 THEN 230
140 Z1 = Y1/X1
145 E = 1: IF X1 <0 THEN E = −1
150 FOR X = A (J−1) TO A (J) STEP E
160 SET X, Z1＊(X−A (J−1)) + B (J−1)
170 NEXT X
180 NEXT J
210 END
230 FOR Y = B (J−1) TO B(J) STEP E
240 SET A (J−1), Y
250 NEXT Y
260 GOTO 180
```

Swapping programs

The SWAP command reads a program from a diskette file, overlays another program with it or link them, and leaves control to that program text, resuming control by the original program the instant the execution of the text has been completed. Such behaviour is just the same as referring to a subroutine in a text; a fetched program returns to the location next to the one that has been subjected to the SWAP command. Hence, the SWAP command can be grasped as a subroutine call. To achieve the above-mentioned action correctly a program text that has the SWAP command must be temporarily stored in a diskette before the execution of swapping. The program control process cannot then return to the stored original program text before the text area is renewed and the subprogram is called and completely executed. The SWAP command is generally available in the following form.

SWAP FDd@v, "file name"

This form orders the system to swap a subprogram specified by "file name" that is stored on the diskette with volume number v present in drive d (d = 1 to 4). Storing of a program text prior to execution of a subprogram occurs onto the diskette present in the drive that has last executed the DIR command. This means that the drive must be loaded with a diskette that allows temporary writing of a program text. The swapping level must be less than 1.

Let's follow the program file behaviour by taking a simple example in order to understand the SWAP command. How does the file when the DIR 1 command is executed?

[Program present in the text area]

```
10 REM COMPOSER
20 M1$ = "A7B6□C3A7A3"
30 M2$ = "B□C□D□E6A3"
40 M3$ = "□F6A3□E7"
50 PRINT "PLAY THE CELLO"
60 SWAP FD2@7, "PLAYER"
70 PRINT "VERY GOOD"
80 END
```

[Program file "PLAYER"]

```
10 REM CELLO PLAYER
20 MUSIC M1$, M2$, M3$
30 PRINT "OK?"
40 END
```

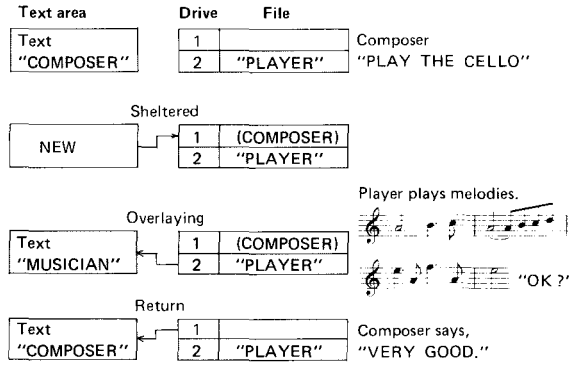
This file is present on slave diskette no. 7 inserted in drive no. 2.

Initially, the text "COMPOSER", present in the text area, is executed.

First the SWAP command, statement no. 60, shelters the text on the diskette present in the drive FD1 that has executed the DIR command, and renews the text area.

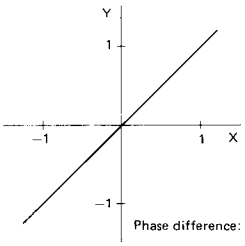
Second the text area is overlayed with BTX "PLAYER" and the program is executed to play melodies.

On the completion of playing, the sheltered COMPOSER returns, saying "VERY GOOD."

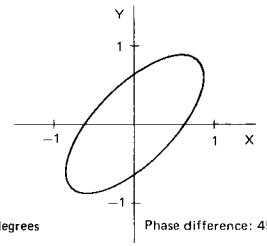


[Exercise]

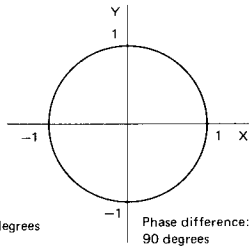
The Lissajou's figure refers to a figure composed of two sine waves drawn on the rectangular coordinates. The figure, when the two sine waves are identical in frequency and phase, is a straight line that crosses the X- and Y-axis at an angle of 45 degrees. And the figure becomes roundish gradually as the phase difference between the two sine waves is increased, shaping a circle when that difference reaches 90 degrees.



Phase difference: 0 degrees



Phase difference: 45 degrees



Phase difference: 90 degrees

The two sine waves can only make a simple figure as above when their frequencies are identical, but they shape a variety of interesting figures as the ratio of their frequencies is changed.

The Basic has instructions SET and RESET to be capable of making dots at arbitrary points on the display. Now, let's devise a program to provide various Lissajou's figures by using the "SET" instruction.

The main program gives the frequency ratio of two sine waves to variable P and make the system execute the routine of the file "LISSAJOUS" six times. Here, variable Q indicates the phase difference between two sine waves. The statement no. 140 of the main program causes swapping, moving the program execution to the file "LISSAJOUS". The FOR ~ NEXT statement varies the value of R from 0 to 1. When J = 40 or R = 0.5, X and Y take the following values if Q = $\pi/3$ and P = 1 then.

X = 25 Y = 12.5 (R = 0.5, Q = $\pi/3$, P = 1)

In this way, the values of X and Y can be determined by varying the value of R. The SET X, Y of statement no. 150 makes one of dots that make up the Lissajou's figure. X and Y, when J = 80, take the same values as when J = 0. It is obvious that the periodicity of the trigonometric function causes such a relationship for other values of J. Returning to the main program, it is obvious that because the phase difference varies by $\pi/12$, a Lissaujou's figure made by the next swapping slightly changes in shape compared with the previous one. In this way, we can have many different Lissajou's figures one after the other on the computer display.

A sample program to obtain such Lissajou's figures is shown below. The program consists of a main program and a subroutine program "LISSAJOUS" to be fetched with the SWAP command. These are required to be recorded in advance on diskette number 10, which must be set in drive no. 2.

MAIN PROGRAM

```
100 PRINT "**      LISSAJOUS     **"
110 INPUT "↓↓↓↓ FREQ.  X : Y = 1 : " ; P
120 FOR I = 0 TO 6
130 Q = I*π/12
140 SWAP FD2@10, "LISSAJOUS"
150 NEXT I
160 END
```

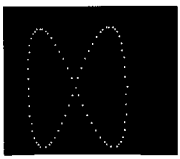


Photo 5
Phase difference: 0
Frequency ratio: 2

SWAP ROUNTINE "LISSAJOUS"

```
100 PRINT "●"
110 FOR J = 0 TO 80
120 R = J/80
130 X = 25 + 25*SIN (2*π*R)
140 Y = 25 + 25*SIN ((2*π*R)*P + Q)
150 SET X, Y
160 NEXT J
170 END
```

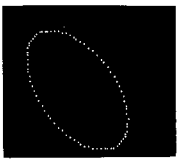


Photo 6
Phase difference: 2/3
Frequency ratio: 1

Error correction control

If we had the skill to ascertain the cause of, and correct various errors we encounter while running BASIC programs, we should be able to resolve the errors without stopping the program execution.

The Disk Basic, when an error occurs during the program execution, causes the computer to display the corresponding error number and stop the program execution. Using the error correction instruction, however, we can restore our program and continue to execute it by ascertaining the cause of the error and taking action against it. The instruction is composed of three subordinate instructions: ON ERROR instruction to leave program control to the error correction routine on the occurrence of an error, variables ERN and ERL to ascertain the kind of error and locate it, and the RESUME instruction to return the system to the original program.

ON ERROR GOTO line number

This instruction transfers control to a routine specified by line number at the occurrence of an error, serving as a declaration statement to connect the occurrence of an error and the error correction routine. The statement causes the system to transfer control to a specified statement number without fail whatever error occurs.

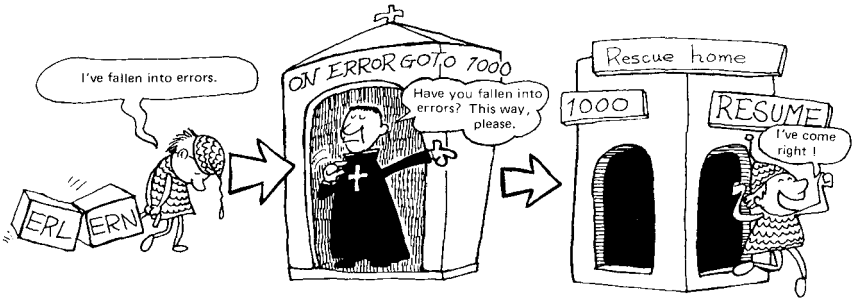
ERN, ERL

ERN and ERL are special variables that are individually substituted by error number and error line number at the occurrence of an error. The kind and location of an error can be determined by testing the values of these variables with an IF statement.

RESUME

This instruction returns the control to the original program after the error has been cleared, and can be used in the following modes according to the way of returning the control.

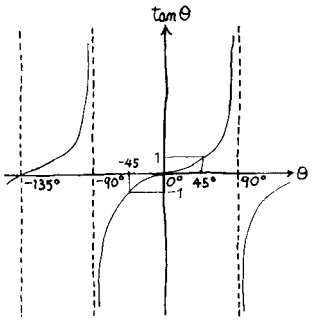
RESUME	Starts the program execution again at the statement where an error occurred.
RESUME NEXT	Moves control to the statement that follows the statement where an error occurred.
RESUME line number	Moves the control to the location specified by the line number.
RESUME 0	Moves control to the start of the program.



We show below some examples of the use of the error correction instruction.

Calculation of tangent

```
100 ON ERROR GOTO 1000
110 FOR TH = 0 TO 180 STEP 30
120 PRINT TH, TAN (π*TH/180)
130 NEXT : END
1000 PRINT "OVERFLOW"
1010 RESUME NEXT
```



[Execution]

```
RUN
 0          0
 30         0.57735027
 60         1.7320508
 90         OVERFLOW
 120        −1.7320508
 150        −0.57735027
 180        0
```

The sample program shown above is made to show "OVERFLOW" on the computer display if a data overflow occurs in calculation of tangent specified by the statement no. 120. And since a data overflow of TAN $(\pi/2) \rightarrow \infty$ occurs in fact when TH comes to 90, the design, when the program is executed, makes the computer display "OVERFLOW", returns the computer to the original program with the RESUME NEXT statement of statement no. 1010 and continues further computation, as the program execution list reveals, by leaving control to the error correction routine (statement no. 1000) that has been declared by the ON ERROR statement when TH = 90.

Now, we modify this program as follows in order to identify the errors.

```
100 ON ERROR GOTO 1000
110 FOR TH = 0 TO 180 STEP 30
120 PRINT TH, TAN (π*TH/180)
130 NEX: END
1000 IF (ERN = 2)*(ERL = 120) THEN 1020
1010 PRINT "ERROR" ; ERN ; "  IN" ; ERL : END
1020 PRINT "OVERFLOW"
1030 RESUME NEXT
```

[Execution]

```
RUN
 0          0
ERROR 1 IN 130
```

This modified program has NEX instead of NEXT in statement no. 130. The program execution list above reveals that the statement no. 120 is executed when TH = 0 and then a syntax error of the "NEX" occurs, causing the program to jump to the statement no. 1000. From this it is known that the condition is false with the error identification statement using variables ERN and ERL, thus forcing the computer to execute statement no. 1010 to stop the program execution. If the "NEX" is replaced with "NEXT", the execution result should be the same as that of the initial program.

Error correction by file control

[Problem]

A certain program reads and writes data using a sequential access file. What measure should be taken in this case in order to display all the expected file control errors in the block using the error correction routine.

[Solution]

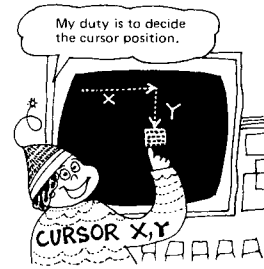
First declare the statement number of the error correction routine by placing the ON ERROR statement, then arrange the program so that expected errors are collected in the error correction routine to be properly corrected.

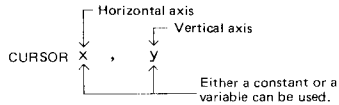
```
  10 ON ERROR GOTO 1000
           :
      Main program
           :
 999 PRINT "ERROR IN" ; ERL : END
1000 IF ERN = 40 THEN PRINT "NOT FOUND" : GOTO 1080
1010 IF ERN = 41 THEN PRINT "HARD ERROR" : RESUME 999
1020 IF ERN = 46 THEN RPINT "LOCKED FILE" : GOTO 1040
1030 GOTO 1100
1040 INPUT "UNLOCK ? Y/N " ; A$
1050 IF A$ = "Y" THEN 1070
1060 RESUME 999
1070 UNLOCK F$: RESUME
1080 INPUT "FILE NAME ? " ; FN$
1090 RESUME
```

According to this solution, the computer gives a display of "NOT FOUND" and awaits the designation of another file name with the statement no. 1080 when the error "Not found", error number 40, of file control errors occurs. And if some file name is entered, the system restarts the program execution with the RESUME command. Also, the computer, when the error "Data", error number 41, occurs, displays "HARD ERROR" with the statement 1010 and the corresponding error line number with the RESUME 999, stopping the program execution. The computer then asks us if locked files should be unlocked or not when the "write protect file" of error number 46 is accessed (loaded with some data). If our answer is "Unlock!", the computer executes the UNLOCK command and then resumes the original program again, whereas if our answer is not "Unlock!", it displays the corresponding error number with the RESUME 999, stopping the program execution.

CURSOR command

The action of the CURSOR command is to bring the cursor to any position on the CRT screen. The horizontal movement of the cursor — along the X-axis — is made when a constant from 0 to 39 or a variable is input; "0" moves the cursor to the leftmost position and "39" to the rightmost position. The vertical movement — along the Y-axis — is made when a constant from 0 to 24 or a variable is entered; "0" moves the cursor to the top and "24" to the bottom. The CURSOR function may be taken to be the variation of the TAB( ) function that permits the up-and-down movement in addition to the horizontal movement, facilitating so much easily the construction of a display program.





We will demonstrate how the CURSOR command simplifies programming.

First move the cursor to the 7th line from the top of screen and move it from there to the 8th position from the left end along that line. Then let the display show there "PRINT". Shown below are two programs to display "DISK BASIC" at the place five lines below the said place — where "PRINT" is shown. The left one uses the CURSOR command and the right one does not use it.

```
100 CURSOR 7, 6
110 PRINT "PRINT"
120 CURSOR 7, 11
130 PRINT "DISK BASIC"
```

```
100 PRINT "■↓↓↓↓↓↓↓"
110 PRINT TAB (7)
120 PRINT "PRINT"
130 PRINT "↓↓↓↓"
140 PRINT TAB (7) ; "DISK BASIC"
```

You will notice when comparing the above two programs that the use of the CURSOR command not only shortens the programs's length, but enables you to ditermine very easily the current cursor position or display position. Considering that making an easy-to-read program is the essential step to making a program work, the command can be siad to benefit us greatly.

Demonstrated below is a sample program to display 40 items of data. The program of course uses the CURSOR command. The array variable A (I) is presupposed to be filled with the necessary data.

```
200 CURSOR 2,2 : PRINT "NO.          DATA"
210 FOR I = 1 TO 20
220 CURSOR 3,4 + I : PRINT I, A (I)
230 NEXT I
250 CURSOR 22,2 : PRINT " NO.          DATA"
260 FOR I = 21 TO 40
270 CURSOR 23, −16 + I : PRINT I, A (I)
280 NEXT I
290 END
```

Logical-open USR function

An ordinary USR function serves as an instruction to fetch a machine-language program formed in the machine-language user's area as a subroutine from a Basic program. The Disk Basic is given an additional function to enable the logical opening of the USR function.

The data access with the logical-open USR function can be accomplished in the same statement composition as applied to sequential access file. The logical open (WOPEN, ROPEN) is used with the USR function first, then the practical access is executed with the PRINT# and INPUT#statements, and finally, the execution ends with the CLOSE# statement.

With the USR function in the logical-open state, the execution of the PRINT# and INPUT# statements is made under the following conditions.

Writing

```
100 WOPEN #10, USR (49152)
110 PRINT #10, A$
120 CLOSE #10
```

100 ——→ The system defines USR (49152) as logical number #10 and opens it for writing.

110 ——→ The system writes a row of ASCII codes, the contents of string variable A$, in the data write buffer, places the start address of that buffer into the DE register and the data length in the BC register, then executes the above USR.

When A$ = "ABCD", for example, the data write buffer is loaded with ASCII codes as illustrated below and the BC register is loaded with the number of ASCII codes exclusive of [CR] code, when the USR (49152) is executed.

"A"	(41H)	←——→ DE
"B"	(42H)	
"C"	(43H)	BC ← 0004H
"D"	(44H)	
[CR]	(0DH)	

In statement no. 110, the print data A$ is not followed by a semicolon. It decides the mode of the control to be practiced next when the machine returns from the machine-language program whether or not a semicolon follows the A$.

(1) When a semicolon is present ---- The system moves to the following statement
(2) When no semicolon is present --The system sets the [CR] code (0DH) into an address indicated by the DE register and executes the USR (n) again with BC = 0001.

For example, when the write routine has been made so as to control the line printer, the system can decide based on this control action whether or not line feed should be made after printing the data.

120 ——→ The system closes the logical number #10.

The machine-language program itself fetched with the USR function depends on the user's applications. In short, the logical-open USR function can deliver in the above-mentioned conditions the contents of variables used in the Basic text to the region under the control of the USR function.

Reading

```
200 ROPEN #11, USR($C100)
210 INPUT #11, B$
220 CLOSE #11
```

200 → The system defines the USR ($C100) as the logical number #11, and opens it for reading.

210 → The system executes the USR. In the machine-language routine for the execution, however, input data must be placed in the identical input/output data buffer with that used in the execution of the aforementioned PRINT# statement, sequentially from the start and the total number of ASCII codes of that data must be placed in the BC register. When these requirements are fulfilled, the system substitutes the B$ by that data.

220 → The system closes the logical number #11.

On reading, the system also can substitute variables used in the Basic text by data introduced from external units in the form of a user's machine language program so that the above conditions should be satisfied.

In the above example, the address designation of USR ( ) is written like this: $C100. This is the hexadecimal representation of an address to be called. That is, the use of sign "$" enables you to specify any address in four hexadecimal digits. (Such hexadecimal address representation can be made for the LIMIT instruction as well.)

Simple Exercise

Lets's try to call a monitor subroutine in order to get used to using the logical-open USR function.

(Problem)

Using the INPUT# statement instead of the INPUT statement and the PRINT# statement in place of the PRINT statement, place strings to input variables, and call them to the computer display.

(Solution)

Devise the following program, for example.

```
10 ROPEN #1, USR($0003)
20 WOPEN #2, USR($0015)
100 INPUT #1, A$, B$
110 IF (A$ = "  ")*(B$ = "  ") THEN 1000
120 PRINT #2, "A$ =", A$
130 USR($0006)
140 PRINT #2, "B$ = ", B$
150 USR($0006)
160 GOTO 100
1000 CLOSE : END
```

In this program, two string variables A$ and B$ are loaded with data in statement 100. The USR functions employed in statements nos. 130 and 150 are used in the ordinary manner, having nothing to do with the "logical open" function. The contents of the monitor subroutine are as follows. (For details, consult the Manual "Systems Program.")

$0003 - - - - One line of data is input through the keyboard. 80 characters including [CR] (0DH) can be set starting from the address indicated by the DE register when called, until [CR] key is input.

$0015 - - - - The system displays a message that starts at the cursor position. It displays the data stored in the address that is specified by the DE register when the call is made, to [CR] code, but does not perform a carriage return.

$0006 - - - - The system advances to the next line, and sets the cursor to the head of that line.

Errors that occur during the execution of a machine-language program can be linked with each other in the following conditions on purpose to move control to the error correction routine that has been defined by the ON ERROR command of the Disk Basic.

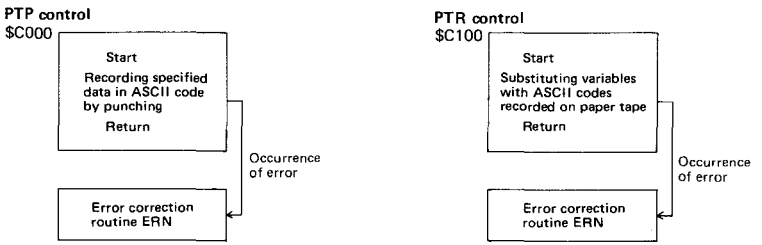
The index registers IY and IX contain special value when the USR(n) is executed. When error correction is required in the user area, a proper error code is entered into the address indicated by the IY register, namely (IY + 0), then the system is jumped to the address indicated by the IX register. That is, the control, when an error correction rotuine is declared with the ON ERROR, can be moved there with JP (IX).

Here, "a proper error code" refers to a number that replaces the error number ERN. The ERN can be replaced with any number of 0 to 255, however, you must not use any number that is used in the Disk Basic. Use other numbers, otherwise confusion could occur.

Example of PTP and PTR control

We will write a program to control the PTP (paper tape puncher) and PTR (paper tape reader) as a practical application of the logical-open USR function.

The composition of an OBJ program to be made is as follows.



If such machine-language programs are present in the machine-language area, a program, for example, to record the contents of string variable A$ on paper tape by punching or to read ASCII data up to CR code to string variable B$ can be composed of very simple statements as given below.

[Outputting the contents of string variable A$ to the PTP]

```
300 WOPEN #3, USR ($C000)
310 PRINT #3, A$
320 CLOSE #3
```

[Inputting string from the PRT to B$]

```
400 ROPEN #4, USR ($C100)
410 INPUT #4, B$
420 CLOSE #3
```

PTP and PTR control programs exemplified on the next and subsequent pages are intended for the model RP-600 PTP & PRT of Nada Electronics Laboratories. (The assembly list has been made using the dot printer MZ-80P3 dump with the Assembler SP-2102.)

Example of PTP control program

```
    **  Z80 ASSEMBLER SP-2102 PAGE 01  **

0000                    ;
0000                    ;     PTP CONTROL FOR DISK BASIC
0000                    ;        CALL PTPC
0000                    ;           DE:DATA ADDRESS
0000                    ;           BC:BYTE SIZE
0000                    ;              MAX=255
0000                    ;           IX:ERROR RETURN ADDRESS
0000                    ;           IV:ERROR CODE
0000                    ;                 100=MORE DATA THAN 255
0000                    ;                 101=BREAK IN
0000                    ;
0000  P                 POTA:   EQU   E0H              ; DATA PORT
0000  P                 POTB:   EQU   E1H              ; CONTROL PORT
0000                    ;
0000  P                 BREAK:  EQU   001EH
0000                    ;
0000                    PTPC:   ENT
0000 78                         LD    A,B
0001 B1                         OR    C                ; IF BC=0 THEN RETURN
0002 C8                         RET   Z
0003 3EF0                       LD    A,F0H
0005 D3E1                       OUT   (POTB),A         ; START
0007 1B                         DEC   DE
0008 DBE1                       IN    A,(POTB)
000A E602                       AND   02H
000C FE02                       CP    02H
000E 20F8                       JR    NZ,-6
0010 26FF                       LD    H,FFH
0012 13                 PTP4:   INC   DE
0013 CD1E00                     CALL  BREAK
0016 3E65                       LD    A,101            ; BREAK CODE
0018 2825                       JR    Z,PTP3
001A 1A                         LD    A,(DE)
001B 2F                 PTP5:   CPL
001C D3E0                       OUT   (POTA),A         ; DATA OUT
001E DBE1                       IN    A,(POTB)
0020 E602                       AND   02H
0022 20FA                       JR    NZ,-4
0024 DBE1                       IN    A,(POTB)
0026 E602                       AND   02H
0028 28FA                       JR    Z,-4
002A 0B                         DEC   BC
002B 78                         LD    A,B
002C B1                         OR    C                ; IF DATA END THEN RET.
002D 2807                       JR    Z,PTP2
002F 25                         DEC   H
0030 3E64                       LD    A,100            ; MORE DATA ERROR CODE
0032 280B                       JR    Z,PTP3
0034 18DC                       JR    PTP4
0036 3EF2               PTP2:   LD    A,F2H


0038 D3E1                       OUT   (POTB),A
003A AF                         XOR   A
003B 2F                         CPL
003C D3E0                       OUT   (POTA),A
003E C9                         RET
003F                    ;
003F FD7700             PTP3:   LD    (IV+0),A         ; ERROR CODE
0042 3EF2                       LD    A,F2H
0044 D3E1                       OUT   (POTB),A
0046 AF                         XOR   A
0047 2F                         CPL
0048 D3E0                       OUT   (POTA),A
004A DDE9                       JP    (IX)
004C                            END
```

Example of PTR control program

```
    **  Z80 ASSEMBLER SP-2102 PAGE 01  **

01 0000                 ;
02 0000                 ;   PTR (NADA) CONTROL
03 0000                 ;     CALL PTRC
04 0000                 ;       DE    :DATA ADDRESS
05 0000                 ;       BC    :DATA COUNTER
06 0000                 ;       BRK   :ERN<--110
07 0000                 ;       TAPE E:ERN<--111
08 0000                 ;       IX    :JUMP ERROR
09 0000                 ;       IY    :ERROR CODE
10 0000                 ;
11 0000  P              POTA:   EQU   E0H
12 0000  P              POTB:   EQU   E1H
13 0000  P              BRK:    EQU   001EH
14 0000                 ;
15 0000                 PTRC:   ENT
16 0000 AF                      XOR   A
17 0001 4F                      LD    C,A
18 0002 3EEF                    LD    A,EFH             ; START SIGN
19 0004 D3E1                    OUT   (POTB),A
20 0006 DBE1            PTR2:   IN    A,(POTB)
21 0008 E630                    AND   30H
22 000A CB6F                    BIT   5,A               ; CHECK TAPE END
23 000C 266F                    LD    H,111
24 000E 2033                    JR    NZ,PTR7
25 0010 CB67                    BIT   4,A
26 0012 28F2                    JR    Z,PTR2
27 0014 DBE1            PTR3:   IN    A,(POTB)
28 0016 E630                    AND   30H
29 0018 CB6F                    BIT   5,A
30 001A 266F                    LD    H,111
31 001C 2025                    JR    NZ,PTR7
32 001E CB67                    BIT   4,A
33 0020 20F2                    JR    NZ,PTR3
34 0022 DBE0                    IN    A,(POTA)          ; DATA IN
35 0024 2F                      CPL
36 0025 B7                      OR    A
37 0026 280C                    JR    Z,PTR4
38 0028 12                      LD    (DE),A
39 0029 13                      INC   DE
40 002A FE0D                    CP    0DH
41 002C 2823                    JR    Z,PTR8
42 002E 0C                      INC   C
43 002F 79                      LD    A,C
44 0030 FEFE                    CP    FEH
45 0032 2819                    JR    Z,PTR5
46 0034 DBE1            PTR4:   IN    A,(POTB)
47 0036 E630                    AND   30H
48 0038 CB67                    BIT   4,A
49 003A 28F8                    JR    Z,PTR4
50 003C CD1E00                  CALL  BRK


01 003F 266E                    LD    H,110
02 0041 20D1                    JR    NZ,PTR3
03 0043 7C              PTR7:   LD    A,H
04 0044 FD7700                  LD    (IY+0),A
05 0047 3EFF                    LD    A,FFH
06 0049 D3E1                    OUT   (POTB),A
07 004B DDE9                    JP    (IX)
08 004D                 ;
09 004D 3E0D            PTR5:   LD    A,0DH
10 004F 12                      LD    (DE),A
11 0050 13                      INC   DE
12 0051 3EFF            PTR8:   LD    A,FFH
13 0053 D3E1                    OUT   (POTB),A
14 0055 C9                      RET
15 0056                         END
```

PTP/PTR control

The PTP/PTR (model RP-600) of Nada Electronics Laboratories relies on the following control and data signals.

PTP

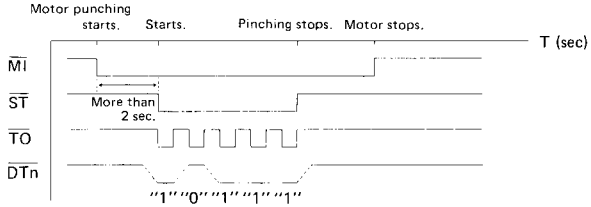
$\overline{MI}$:	Motor on/off input signal
$\overline{TO}$:	Timing output signal
$\overline{ST}$:	Start/stop input signal
$\overline{DT_1}$ ~$\overline{DT_8}$:	Data input signal

PTR

$\overline{STA}$:	Start/stop input signal
$\overline{RB}$:	Operational indication output signal (tape end and unusual stop)
$\overline{SPR}$:	Sprocket output signal
$\overline{RP_1}$ ~ $\overline{RP_8}$:	Data output signal

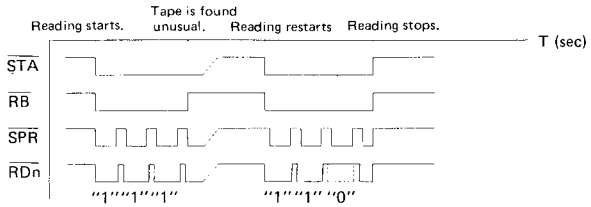
These signals act according to the following timing, controlling the PTP/PTR.

PTP



Note) Data to be recorded is prepared when $\overline{TO}$ is at High and held while it is at Low.

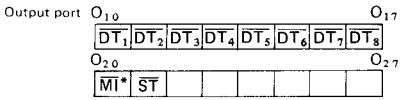
PTR

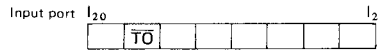


The control and data signals are given the following port distribution. The PTP/PTR control program makes the judgement and setting of the PTP/PTR state by inputting and outputting those control signals.

As an interface to be used, the universal I/O card (MZ-80I/O-1) is practical. The port numbers shown below refer to those shown below in the symbol figure of the universal I/O card. For precautions on connection, consult the instruction manual of that card.)

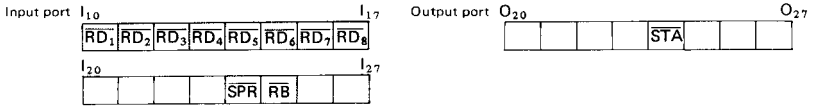
PTP





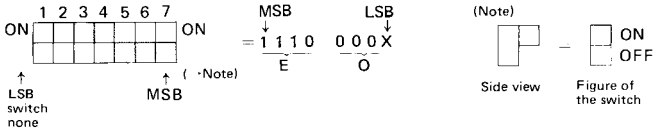
* Since this program does not provide remote control, nothing must be connected to $\overline{MI}$.

PTR

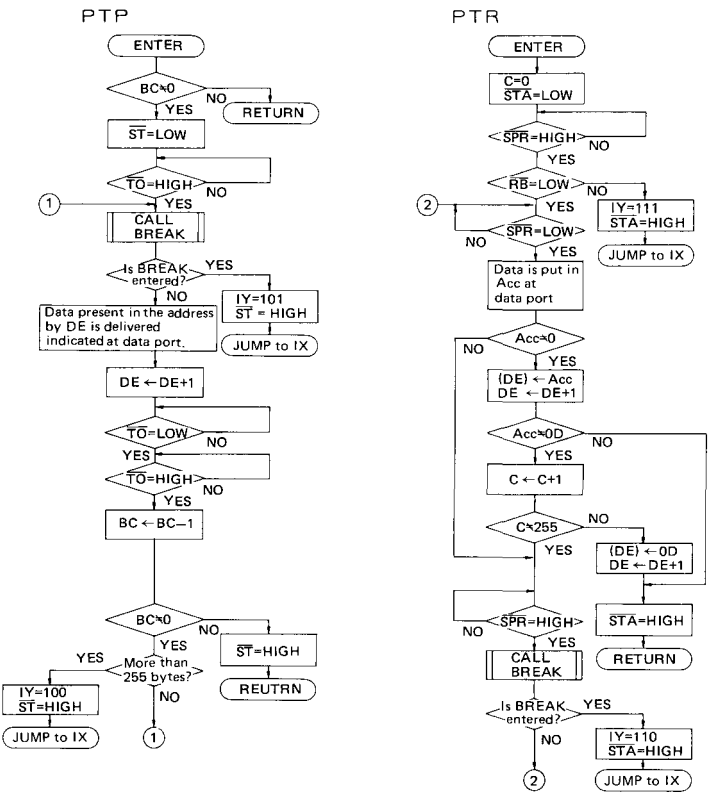


The control program uses "E0" as data signal port and "E1" as control signal port. Because the universal I/O card enables two successive ports to be input and output, the following setting of the port switch can actuate both the PTP and PTR.

Illustration of the port switch



Shown below is the flow chart of the PTP/PTR control program.

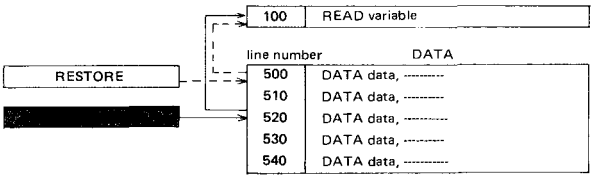


Updated commands

Described below are updated commands in Disk Basic SP-6015 version, as well as their forms.

RESTORE line number

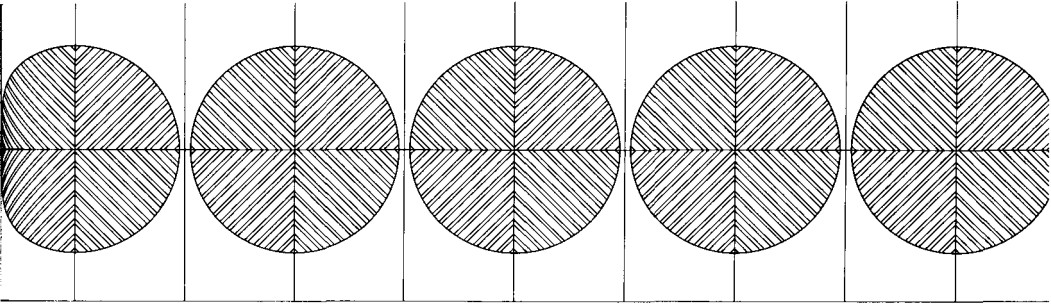
The RESTORE statement used in the Basic Series SP-5000 is served as a command to return the data read pointer used in the READ ····· DATA statement to the beginning of the DATA statement, whereas this "RESTORE line number" statement moves the data read pointer to the DATA position specified by statement number.



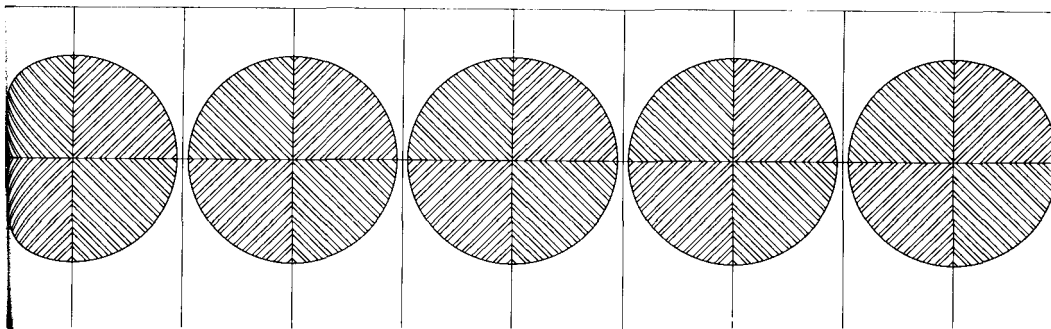
Provided a program has DATA statement of statement nos. 500 thru 540 as shown above, for example, the data reference caused by the READ statement, if the RESTORE command is given, begins with the start data of the DATA statement (dotted line), and the data reference caused by the next READ statement starts with the beginning of the DATA statement in statement no. 520 (solid line) if the statement of RESTORE 520 is executed.

INP @port, variable OUT @port, variable

The command reads data from and delivers it to external devices through the universal interface card respectively. The updated variation is different from the same type of command employed in the SP-5025 only in that it uses symbol "@" instead of the symbol "#" in order to indicate the input/output port number.



Chapter 3
The Disk-Basic-based sample programs



In this chapter are introduced three typical examples of programs using DISK BASIC in order that the user may utilize the fundamental programming technique described in these examples. The programs have the following applications.

I. Report generator

The Report Generator is employed to perform summarization of data, calculation of their mean values, and to obtain standard deviations, and approximate regression curve and regression line on the basis of data distribution. Then, this data is printed out on the printer in a form of a report. As a data file a sequential access file is used and many CHAIN and SWAP commands are used in the program.

II. Amateur radio log recording

Arrangement and preparation of an amateur radio log is maintained by a computerized program. As the log is filed in the diskette file, it enables easy addition and correction to communication records as well as allowing fast data retrieval and list generation. A random access file is used in this program to maintain ever-growing and time-to-time varying groups of data.

III. Inventory control program

In this program, the typical functions of "Inventory management" are all contained, with which merchandize handled by a store are classified by the merchandize name and code and a list is generated for controlling the number of stock, unit price and stock value. A random access file is also used by this program. In addition, an arithmetical routine which enables multiplication up to 16 digits (i.e. multiplication of unit price with quantity) is provided in this program.

Report generator program

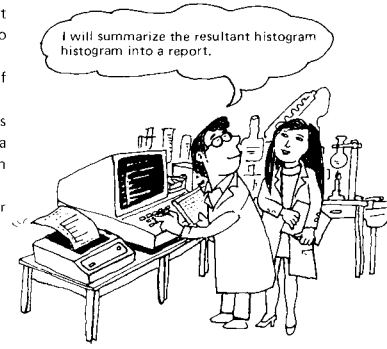
Have you ever been urged to prepare a report in a short time? Well, everyone has met such a case once or twice up to now.

It is much better for a microcomputer not to bury himself under a stock of paper and pencils for preparing the report.

The floppy disk will help you a lot for this purpose. It is easy to mount the disk file into the unit and use it with a program previously prepared. A simple example will be used in the following lines to introduce this program.

Why don't you modify your current program to meet your requirements when using this program?

Let's try it now ! !



Objectivity

This program has been developed for the following functions:

a) Data is classified and summarized on the list and can be shown in the form of a histogram or graph.

b) That the Report Generator program is connectable with a program already in existence.

c) That it enables you to create a single report and that the data are accessible to the disk file for both the summary list and graph and enables you to list up the data on the printer.

d) That the group of input data can be approximated to a simple expression of first or second degree.

The program units are composed of several subroutines for the purposes.

Program usage

a) File name and function of each subroutine text . . . . (Table 1)

	Total, mean value, standard deviation and percentage are listed in the table by every 32 items. This program takes part as a main program.
	Data of positive and negative value are shown by the graph and connected with the text, by which the data is approximated by the formula.
	Data of positive value is shown by the bar graph.
	Three lines of notation can be filled in for each of the next four items; purpose, method, result, and comment.
	Full data on the CRT screen is arranged and printed on the printer or written to the disk.
	Data group distributed on the graph is approximated to "$Y = AX + B$".
	Data group distributed on the graph is approximated to "$Y = AX^2 + BX + C$".

b) Linkage of programs

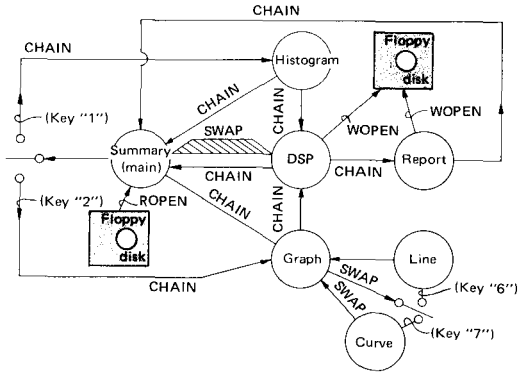


Fig. 1 Linkage of programs

The subroutine texts having functions shown as in Table 1 are linked to other program by means of the CHAIN and SWAP commands (see Fig. 1). And, data can be stored in the disk file by the WOEPN command through "DSPTO ? + F TX" or "Report TX". On the other hand, data on the disk can be read with the ROPEN command through "Tabulation TX".

As all of the texts are linked together in this manner, the program can handle multiple sets of data, if they are filed on the disk in groups of 32 items.

c) Using the program in actual applications

Now, let's us prepare a report using this program. First of all, make sure that the diskette is installed in the disk drive unit. Then, execute DIR. Texts will now appear one after another. Bring the cursor to the "Tabulation TX", then RUN.

It must be noted here, however, that such as the "Histogram TX", "Graph TX", and "DSPTO ? + F TX" can not be used alone, but they will be linked to the "Tabulation TX" in an automatic manner when these three texts are run, starting from the statement number described in the program usage without executing the opening process for "Tabulation TX". But, the rest of the texts can be used independently.

Meanwhile, when one of the keys is depressed after the opening indication upon completion of the tabulation text execution, there appears a description regarding the program usage on the screen. When the key "S" is depressed at this stage, there appears a request whether the previously filed data or report is to be confirmed or not.

Take a look at Fig. 1. When a file name is entered after depressing the key "Y", the data previously filed in the disk appears on the screen and it can then be printed out on the printer. This operation is repeated until the key "N" is depressed at the time of file confirmation. It will go into data input processing when the key "N" is depressed (statement number 400 through 450 and 5000 through 5200). Normally, entry of a non-existant file name at the stage of file name entry will cause it to indicate an error number to stop prgoram execution, but, using the ON ERROR GOTO or RESUME command as in statement number 5015 and 6000~7100, it is able to process the error. When an error is made in the entry of a file name, it will cause the program to return to the previous statement number when the key "X" is depressed after confirming the file name.

These two commands are important and beneficial when used in stroing the data upon completion of key data entry, as the program execution can be continued even if an error is encountered.

With the depression of the key "N" at the previous stage, it goes into data entry. It permit entry of an item up to 5 characters and 9,999,999 for a numeric (up to 7 digits). Data input up to 32 items can be executed at one time.

When too much data is to be entered, either terminate the loop shown in Fig. 1 or use the return to the tabulation TX from the DSP TX, graph TX, or histogram TX. Details for these will be described at each section designated for the relevant text.

When the item is indicated by a number, distribution of data is approximated to the expression in first or second degree in the graph TX. When key entry is over, the table is displayed on the screen and notation will be displayed under the table regarding the data error, addition, and deletion, which should be dealt with at that stage. Then, there appears on the screen the table shown on the right, indicating the total, mean value, maximum value, minimum value and percentage to the total value. Standard deviation will appear when the key "Y" is depressed.

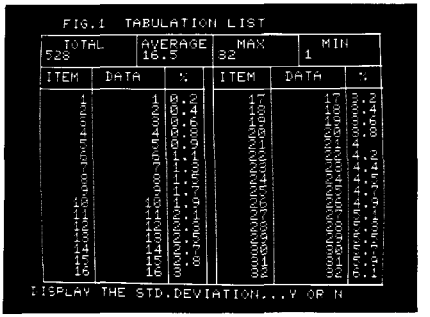


Photo 1 Results of summarization

Depression of the key "Y" according to the notation will cause the SWAP command in the statement number 2700 to be executed. This command will eliminate the tabulation TX out of the program and the file DSPTO ? + F is then called to the computer, then, so that the control is handed to DSP TX now.

With DSP TX, the display contents are then converted into ASCII code from display code by each address starting from the top address of V-RAM to be further converted into character code to make it a string variable of 40 characters per line, and arranges 25 lines to X$ (1). Because the display contents has to be maintained as it is, the program execution takes place each time beeping sound is generated. Therefore, the operator must not depress the BREAK key since the screen contents does not change.

When the arrangement is over, depress either the key "Y" or "N" according to your intention whether the data is to be printed on the printer or not. Depression of the key "Y" will cause a move to the statement number of the file after printout, and depression of the key "N" will move directly to the statement. Avoid double definition at this stage in entering the file name, though commands such as ON ERROR GOTO and RESUME are used in statement number 265 and 3000 ~ 3620 (the same procedure is seen at statement number 142 and 4000 ~ 4100). This time, the previously read data must be written to the disk using the WOPEN and PRINT commands and the file is closed by the CLOSE command (statement number 270 ~ 310).

It has been determined that the program will terminated when "NW = 5" is encountered at statement number 320. Although it is not necessary for the SWAP command, "NW = 5" is defined as a sign to return to the tabulation TX ignoring the CHAIN command in statement number 440, as this DSP TX is employed for the point of each text.

Now, the program is returned under the control of the tabulation TX with NW = 5, and ask for the selection of the graph. Depression of the key "1" will make the file name of the string variable NW$ the histogram TX and depression of the key "2" will make NW$ the graph TX, so as to start executing the CHAIN command in statement number 3310. With this, the control is separated completely from the tabulation TX and reads and executes the file desingated by NW$. Although single use of the SWAP and CHAIN commands differ in their performance, multiple use of the CHAIN command can offer advantageous use (see Fig. 1).

The histogram and the graph seem almost the same, but their distinction can be recognized from Table 1, and that the histogram, as shown in Fig. 1, can not perform primary and secondary regression. For accuracy as a graph the histogram may be better.

Examples of data shown by the histogram and the graph are shown in Photograph 2 and 3, respectively.

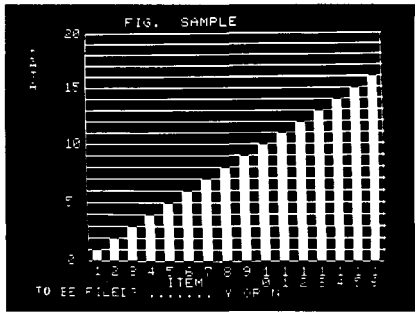


Photo 2 Results on a histogram

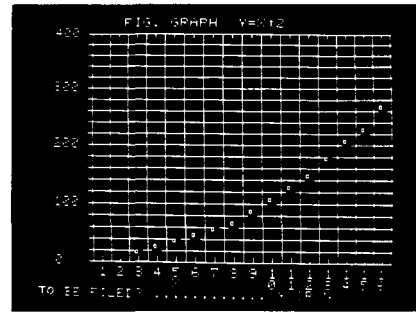


Photo 3 Results on a graph

As both are similar as a program, discussion will be made regarding the graph TX here.
As it is linked with the CHAIN command from the tabulation TX, variables in the tabulation TX remain as they are (item, data, maximum, and minimum values). We must remind you here that the horizontal axis of the histogram or the graph is only arranged with the number in order. You may change them as you wish.

When the title, and unit of vertical and horizontal axes or the classification are entered, the maximum scale value of the graph is determined according to the maximum digits and capacity transfered from the tabulation TX (it will be determined by the higher absolute value, if the maximum or minimum value is of negative value), determines scale the dividing ratio, determines the unit scale, and convert the figure of data to the height in the scale on the graph and places the scale and unit for the horizontal and vertical axes after the preparation of the graph. Here, on the CRT screen is indicated whether the data is to be indicated by an expression or not. Depress the key "6" after the depression of the key "Y" for a straight line graph or the key "7" for a curve graph. Text of regression line or regression curve is executed by the statement number 1205 or 1305 and its result is approximated to "$Y = AX + B$" or "$Y = AX^2 + BX + C$" and the values A, B, and C, are printed out on the printer together with the input data, to be returned to the graph TX.

If the graph is to be filed after returning to the graph TX, it will be printed our or filed by the CHAIN command with DSP TX. Or, the graph can be changed when the key "N" is depressed. Depression of the key "Y" at this stage will cause a return to the tabulation TX after setting "GX = 4" and start to execute the program from statement number 3000 designated by the parameter "GX = 4", where the graph is selected Depression of the key "N" makes "GX = 2", so as to return to the tabulation TX to begin execution from the data entry.

Different point encountered in moving from the graph TX to the DSP TX is that the selection has to be made whether to go to report generation text or not, as the statement number in the DSP TX is neglected this time as the value of NW is set to "NW = 1" or "NW = 2" by statement number 3260 and 3265 of the tabulation TX. Depressing the key "N" at this stage will cause the program to return to the tabulation TX with "GX = 2", so as to execute the program from the data entry in statement number 300.

On the other hand, depressing the key "Y" at this stage will cause the program to go to the report TX by the CHAIN command in statement number 420, so as to execute the report TX.

With the Report Generator program it is possible to make an entry of three lines of notation with one line consisting of 37 characters regarding the purpose, method, results, and comments, after the entry of the date, programmer's name, and title. As this program does not require DSP TX and is able to perform the storing of data and printout independently, it must be selected according to the volume of data. In this program, 37 characters in a line are arranged in the string variable R$ (23).

As every input item is displayed after the entry of the data, error in the input item can be corrected by indicating the line number of the statement through the keyboard. It is also possible to make correction of the programmer's name, date, and title.

Also, the error processing routine to avoid double definition of the file name during the time of file name entry and another error processing routine to check error in entering a non-existant file name, in deleting the unnecessary file are implemented in statement number 513, 520, and 2000 ~ 2070. As this report TX enables you to delete the file that had become unnecessary, proceed to cancel the file after making good confirmation of the file name.

After storing the report on the disk or when it was not filed, it will start program execution from statement number 300 after returning to the tabulation TX by the CHAIN command as it is directly defined to "CX = 10".

It will be essential to fully understand the functions of each program as the programs are endlessly looped to continue subroutine texts one after another.

List of variables

Majority of variables used in this program are listed in Table 2.

a) DISPTO ? + F text (Table 2-a)

Variable	Description
Y (222)	Array of the display code corresponding with the ASCII code.
Z	Shows the CRT screen address in decimal.
ZZ	Indicates the contents in the address Z by display code.
P$	String variable consisting of 40 characters per line.
X$ (24)	Arrays P$ 1 ~ 24 lines.
NW = 5	Variable to inform that it is the SAWP command from the tabulation TX.
GX = 2	Variable used to determine the statement number to be executed after returning to the tabulation TX.

b) Tabulation text (Table 2-b)

Variable	Description
A$ (32)	String variable used to arrange items.
B$ (32)	String variable used to arrange data.
B (32)	Arranges data by numerical value.
PT$ (32)	String variable used to arrange item by percentage.
SS, SS$	Minimum value of data and string variable.
SD, SD$	Maximum value of data and string variable.
HC, HC$	Mean value of data and string variable.
TT, TT$	Total value of data and string variable.
NI$	String variable of file name to read.
X$ (24)	String variable used to read data from file and arrange.
NW$	String variable of the file name of the text to be executed next.
CX = 10	Command variable used to instruct execution from statement number 300 of the report text.
GX = 2	Command variable used to instruct execution from statement number 300 of the DSP text.
NW = 5	Variable to inform the DSP text that the SWAP command is under execution.

c) Histogram text (Table 2-c)

Variable	Description
X (32)	Indicates data by the number of characters (by integral number).
C (32)	Array of characters to indicate odd number of X (32).
TN$	Variable of title name.
KN$	Name variable of item (X-axis).
DN$	Name variable of data (Y-axis).
Z1$ (8)	Array of each character for displaying the name of the Y-axis toward the horizontal direction.
LD	Number of digits of the maximum value (recognized by SD$).
LM$	String variable to indicate MSD of the maximum value.
LM	Numerical variable to indicate MSD of the maximum value
SM	Maximum scale value.
V	Variable engaged to designate space between scale points on the vertical axis.
TS	Variable to indicate the height of data by each character (1 partition).
HS	Variable to indicate index value of each scale.
A (32)	Used only in the graph TX for arraying item in numerical value.
GX = 6 GX = 7	Variable used to swap by this number to the text in which the expression is contained for the approximation of data.

d) Report, regression text (Table 2-d)

Variable	Description
R$ (23)	String variable to array each line of report.
A$	String variable for the date.
B$	String variable for the programmer's name.
D$	String variable for the title name.
NR$	String variable of the file name of the report.
CX = 10	Variable which is used to return to the the tabulation TX to start program execution from statement number 300.
X1 (32)	Variable to express the array variable data in the quadratic expression on the basis of the first item.
Y1 (32)	Variable to express the array variable data in the quadratic expression on the basis of the first data.
Z (32)	Array variable of Y1/X1.

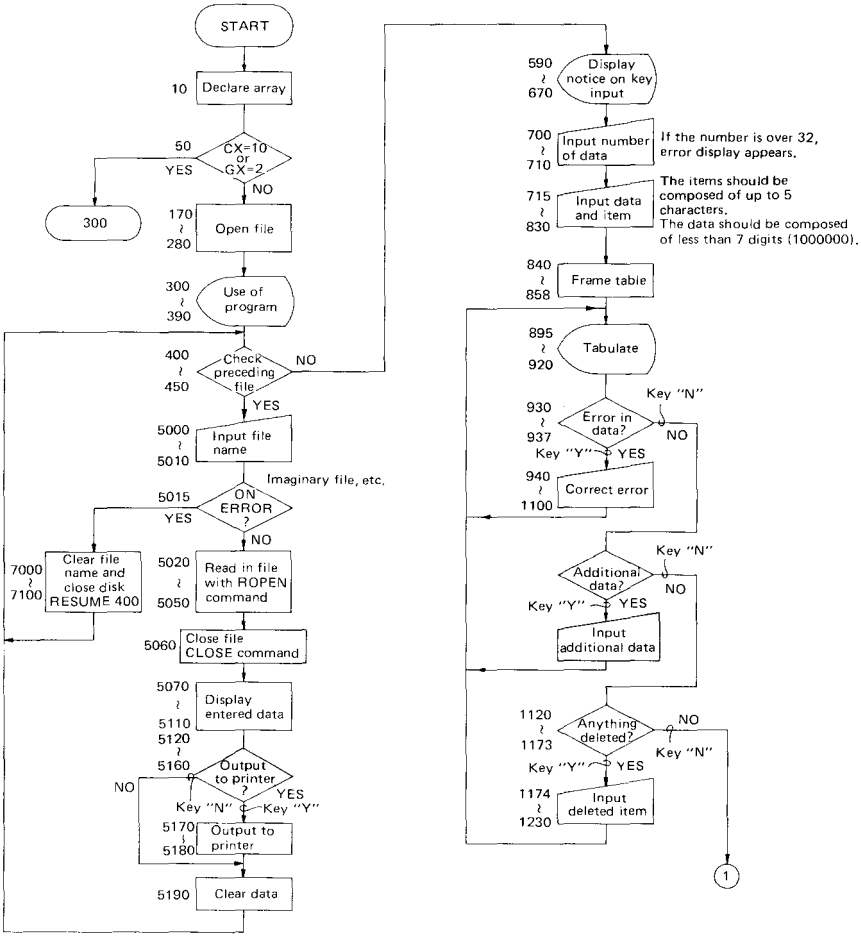
At the end of this section

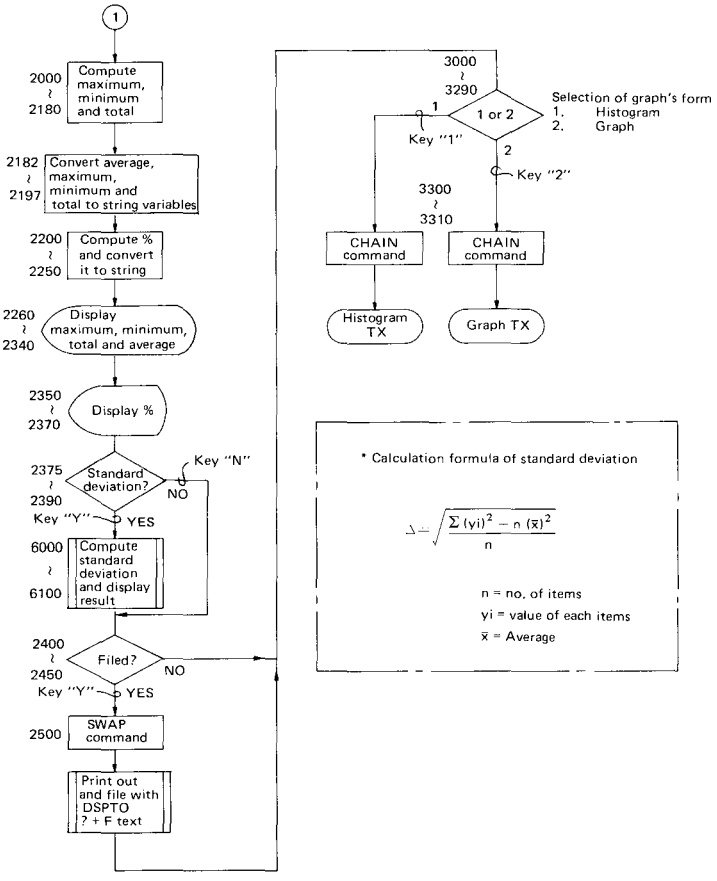
As described in the preceding paragraphs, any small program can be linked by means of the SWAP and CHAIN commands and this allows convenient program preparation.

Although the ON ERROR command is used only in defining the file name in this chapter, there are a number of other uses.

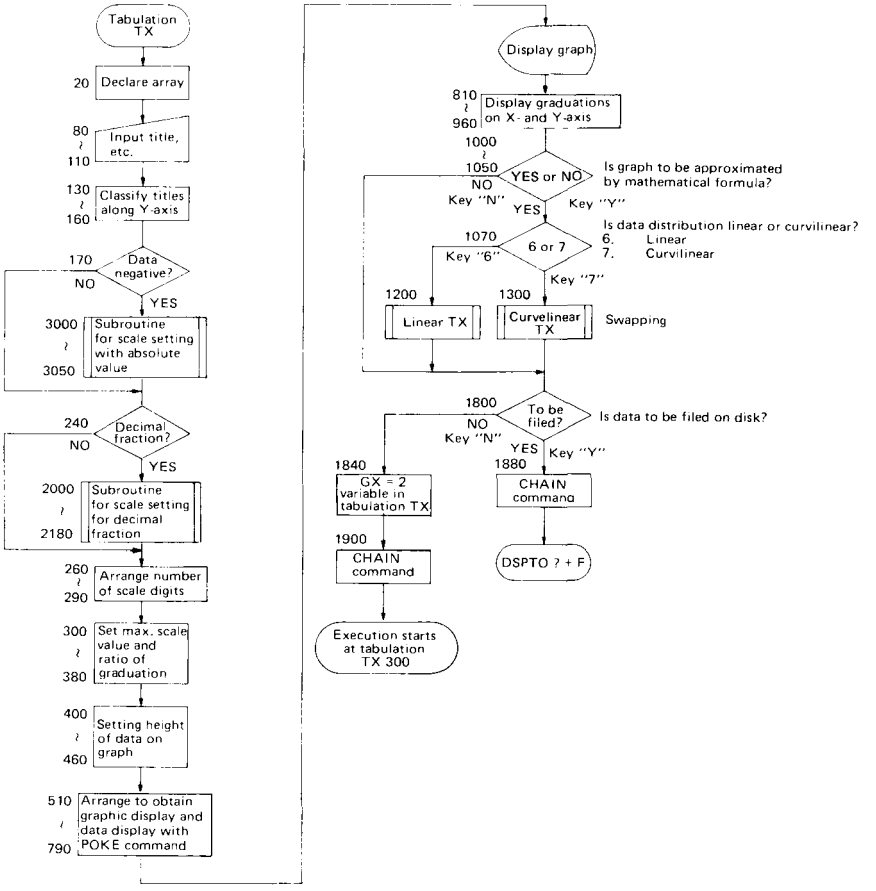
As mentioned at the beginning, the program introduced in this chapter is only for example. Use of CHAIN, SWAP, WOPEN, ROPEN, and ON ERROR commands will allow you to expand program as far as the disk capacity remains available. As flowcharts and lists are shown for each text, please make use of such material.

Tabulation program flowchart

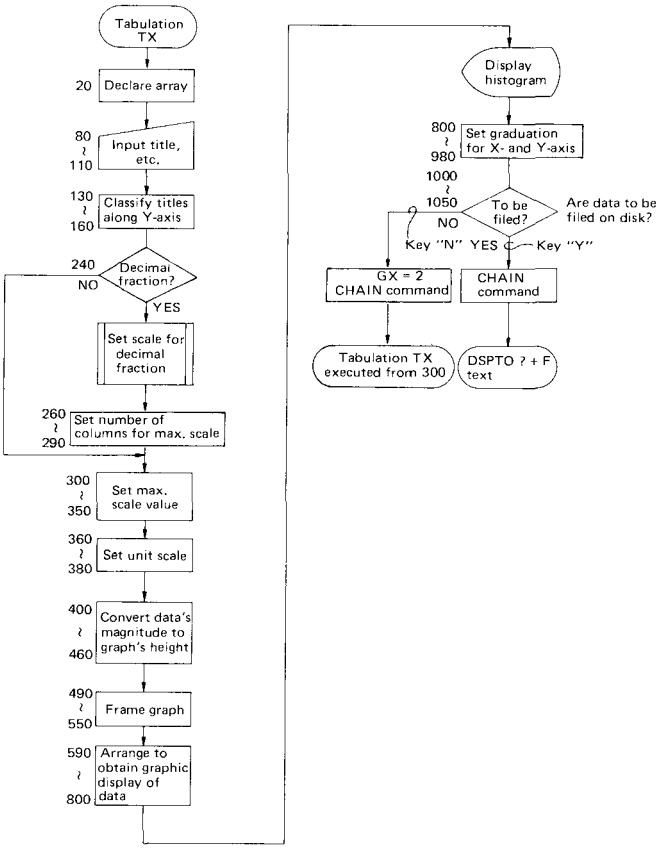



1
2000 ~ 2180
Compute maximum, minimum and total
2182 ~ 2197
Convert average, maximum, minimum and total to string variables
2200 ~ 2250
Compute % and convert it to string
2260 ~ 2340
Display maximum, minimum, total and average
2350 ~ 2370
Display %
2375 ~ 2390
Standard deviation?
Key "N"
NO
Key "Y"
YES
6000 ~ 6100
Compute standard deviation and display result
2400 ~ 2450
Filed?
NO
Key "Y"
YES
2500
SWAP command
Print out and file with DSPTO ? + F text
3000 ~ 3290
1 or 2
Selection of graph's form
1. Histogram
2. Graph
1
Key "1"
2
Key "2"
3300 ~ 3310
CHAIN command
CHAIN command
Histogram TX
Graph TX
* Calculation formula of standard deviation
$\sigma = \sqrt{\frac{\Sigma (yi)^2 - n (\bar{x})^2}{n}}$
n = no. of items
yi = value of each items
$\bar{x}$ = Average

Graphing subroutine text

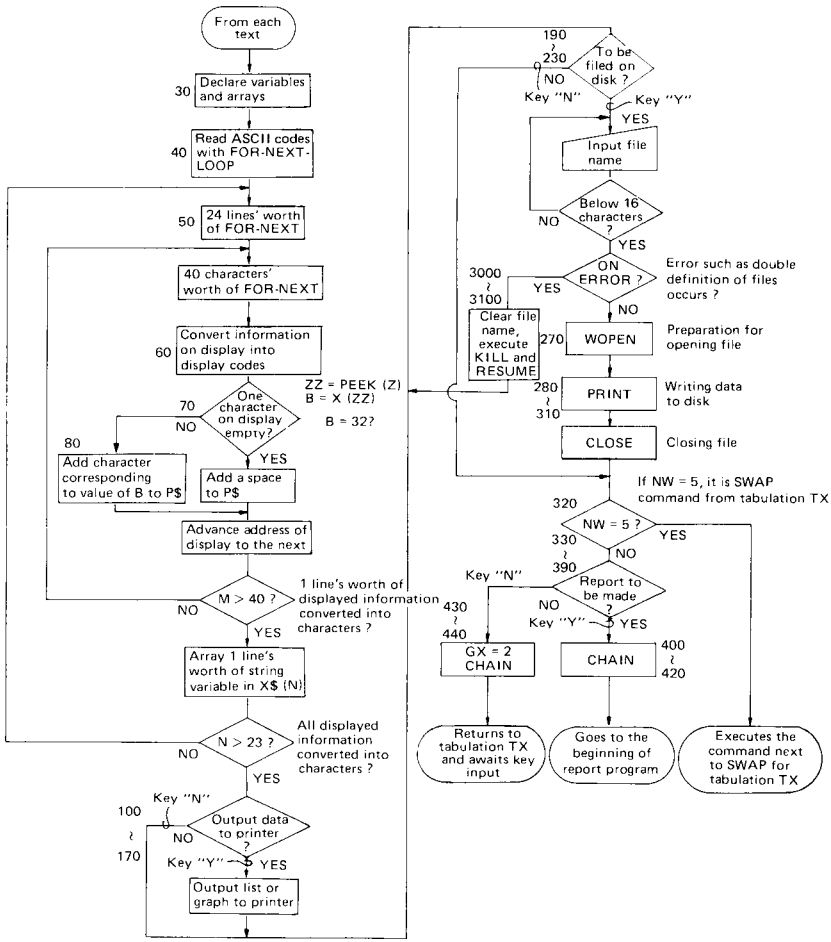


Histogram making subroutine text

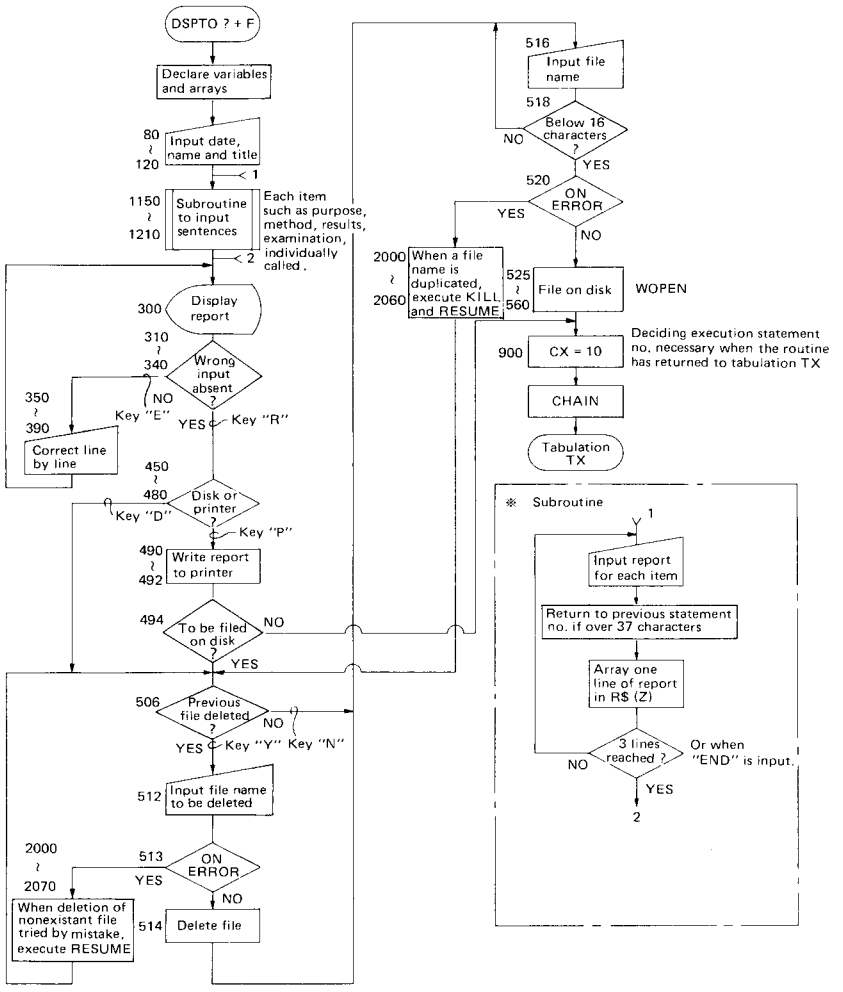


DSPTO ?+F subroutine text

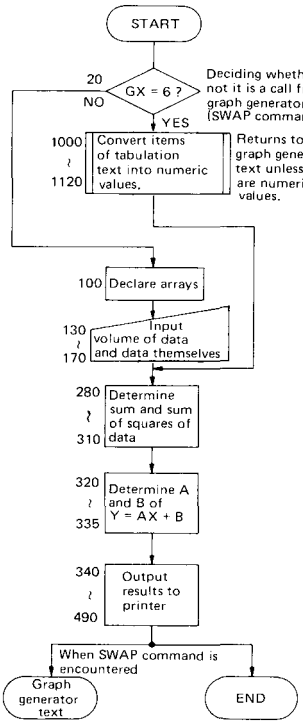
The program converts one frame of information into characters and array them. Then it outputs the data to the printer or disk and, depending on the set value of a variable, either moves to the next text or returns to the original text to execute it.



Report generator subroutine text



Regression line subroutine text



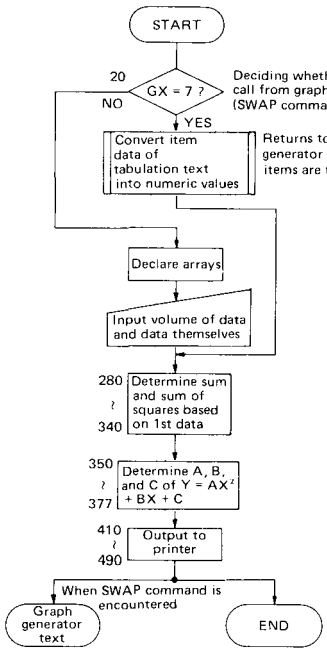
Calculation formula

$$A = \frac{\Sigma xiyi - \frac{\Sigma xi \Sigma yi}{n}}{\Sigma xi^2 - \frac{\Sigma (xi)^2}{n}} \quad ; \; Y = AX + B$$

$$B = \frac{\Sigma yi}{n} - b\frac{\Sigma xi}{n}$$

n: number of data
xi: value for X-axis
yi: value for Y-axis

Quadratic regression curve subroutine text



Calculation formula

$$A = \frac{\Sigma Pi Zi - \frac{\Sigma Pi \Sigma Zi}{n-1}}{\Sigma Pi^2 - \frac{(\Sigma Pi)^2}{n-1}}$$

$$B = \frac{\Sigma Zi}{n-1} - \frac{A\Sigma Pi}{n-1} - 2A \cdot Xi$$

$$C = A \cdot (x_1)^2 - x_1 (B + 2Ax_1) + y_1$$

$Pi = xi - x_1$, $Qi = yi - y_1$, $Zi = Qi/Pi$

n: number of data
xi: value for X-axis
yi: value for Y-axis

```
10 PRINT"■";:DIM A$(32),B$(32),B(32),PT(32),PT$(32),X$(24)
30 IF  GX=4 THEN 3000
50 IF (CX=10)+(GX=2) THEN 300
200 REM "** TITLE **"
210 CURSOR 8,4:PRINT"┌──────────────────────────┐"
220 CURSOR 8,5:PRINT"│                          │"
230 CURSOR 8,6:PRINT"│ *  REPORT GENERATOR  *   │"
235 CURSOR 8,7:PRINT"│                          │"
240 CURSOR 8,8:PRINT"│   TABULATION PROGRAM     │"
245 CURSOR 8,9:PRINT"│                          │"
250 CURSOR 8,10:PRINT"└──────────────────────────┘"
260 CURSOR 16,18:PRINT"DEPRESS ANY KEY  !":USR(62)
270 GET A$:IF A$="" THEN 270
280 IF A$="S" THEN USR(62):GOTO 300
300 REM * HOW TO USE *
310 PRINT"■";:CURSOR 9,1:PRINT"*  HOW TO OPERATE  *"
320 CURSOR 2,5:PRINT"IF DATA IS INPUT ACCORDING THIS"
330 CURSOR 2,7:PRINT"PROGRAM. YOU CAN MAKE A REPORT."
340 CURSOR 2,10:PRINT"PLEASE ENTER THE DATA FOLLOWING"
350 CURSOR 2,12:PRINT"THE INSTRUCTION ON THE SCREEN."
360 CURSOR 4,16:PRINT"** PRESS 'S' TO START"
370 GET A$:IF A$="" THEN 370
380 IF A$="S" THEN USR(62):GOTO 400
390 GOTO 370
400 PRINT"■";:TX=5:CY=5:CURSOR TX,CY:TX=TX+4:CY=CY+2
410 PRINT"■■  PREVIOUS FILE? ...Y OR N"
415 A$=""
420 GET A$:IF A$="" THEN 420
430 IF A$="Y" THEN USR(62):GOTO 5000
440 IF A$="N" THEN USR(62):GOTO 590
450 GOTO 415
590 REM * INPUT DATA BY KEY *
600 PRINT"■";:CURSOR 9,2:PRINT"* DATA INPUT *"
610 CURSOR 2,6:PRINT"THIS PROGRAM PERMITS DATA ENTRY"
620 CURSOR 2,8:PRINT"OF 32 ITEMS."
630 CURSOR 2,10:PRINT"INPUT DATA FOR EACH ITEM."
640 CURSOR 2,12:PRINT"ITEM:    5 CHARACTERS"
650 CURSOR 2,14:PRINT "DATA:     9999999 MAX"
670 CURSOR 2,18:PRINT"* PRESS 'S' KEY WHEN READY *"
680 GET A$:IF A$="" THEN 680
685 IF A$="S" THEN 700
690 GOTO 680
700 PRINT"■";
705 INPUT"NUMBER OF ITEM...";A:PRINT
710 IF A>32 THEN PRINT"  ※ ERROR ":USR(62):A=0:M=100:GOTO 705
712 IF M=100 THEN PRINT"■";:K=2:GOTO 720
715 K=2
720 FOR I=1 TO A
730 CURSOR 0,K:PRINTI:CURSOR 4,K:INPUT"ITEM....";A$
740 L1=LEN(A$):IF L1>5 THEN PRINT" ※ ERROR":USR(62):A$="":M=50:GOTO 730
750 CURSOR 20,K:INPUT"DATA....";B$
760 L2=LEN(B$):IF L2>7 THEN PRINT"■■※ ERROR":USR(62):B$="":M=50:GOTO 750
770 B=5-L1:C=7-L2:A$=SPC(B)+A$:B$=SPC(C)+B$
775 A$=LEFT$(A$,5):B$=LEFT$(B$,7)
780 A$(I)=A$:B$(I)=B$
820 IF M=50 THEN CURSOR 0,K+1:PRINT SPC(10):A$="":B$="":K=K+2:GOTO 825
822 A$="":B$="":K=K+2
825 IF (K=22)+(K=44) THEN K=2 :PRINT"■";
830 NEXTI
840 PRINT"■";"■";
841 PRINT" ┌──────┬──────┬──┬──────┬──────┬──┐"
842 PRINT" │  TOTAL   │AVERAGE│  MAX   │  MIN   │"
844 PRINT" │          │       │        │        │"
846 PRINT" ├──┬──────┼──┼──┬──┼──┬──┼──┬──┤"
848 PRINT" │ITEM │ DATA  │ % ││ITEM │ DATA  │ % │"
850 PRINT" ├──┴──┴──┴──┴──┴──┴──┴──┴──┴──┴──┴──┤"
```

```
852 FOR I=1 TO 16
854 PRINT" |     |       |   ||     |       |   |"
856 NEXT I
858 PRINT" └─────┴───────┴───┴┴─────┴───────┴───┘"
895 PRINT"■"::CURSOR 0,7
900 FOR I=1 TO A
905 IF I=<16 THEN 915
910 IF I=>17 THEN PRINT TAB(21);A$(I);TAB(27);B$(I):GOTO 920
915 PRINT TAB(2);A$(I);TAB(8);B$(I)
918 IF I=16 THEN PRINT"■"::CURSOR 0,7
920 NEXT I
930 PRINT"■"::CURSOR 0,24:PRINT"DATA CORRECT?....Y OR N";
931 USR(62):A$=""
934 GET A$:IF A$="" THEN 934
935 IF A$="Y" THEN USR(62):GOTO 940
936 IF A$="N" THEN USR(62):GOTO 1120
937 GOTO 934
940 CURSOR 0,24:PRINT"WHICH ITEM IS WRONG?   ";
942 CURSOR 25,24:INPUT" ";N
944 PRINT"■"::CURSOR 2,2:PRINT"ITEM";A$(N);"     DATA:";B$(N):PRINT
950 PRINT"* WHICH IS WRONG DATA OR ITEM? ":PRINT
960 PRINT"ITEM: 'K'  DATA: 'D'  BOTH: 'B' ...PUSH ":PRINT
970 GET C$:IF C$="" THEN 970
980 IF C$="K" THEN A$(N)="":GOTO 1000
990 IF C$="D" THEN B$(N)="":GOTO 1010
995 IF C$="B" THEN B$(N)="":A$(N)="":GOTO 1020
998 GOTO 970
1000 INPUT"ITEM CORRECTION: ";A$(N):L=LEN(A$(N))
1003 IF L>5 THEN PRINT" X ERROR(UP TO 5 CHARS)":A$(N)="":GOTO 1000
1005 B=5-L:A$(N)=SPC(B)+A$(N):GOTO 1100
1010 INPUT"DATA CORRECTION: ";B$(N):L=LEN(B$(N))
1013 IF L>7 THEN PRINT" X ERROR(UP TO 7 CHARS)":B$(N)="":GOTO 1010
1015 C=7-L:B$(N)=SPC(C)+B$(N):GOTO 1100
1020 INPUT"ITEM CORRECTION: ";A$(N):L=LEN(A$(N))
1023 IF L>5 THEN PRINT" X ERROR(UP TO 5 CHARS)":A$(N)="":GOTO 1020
1025 B=5-L:A$(N)=SPC(B)+A$(N):PRINT
1030 INPUT"DATA CORRECTION: ";B$(N):L=LEN(B$(N))
1033 IF L>7 THEN PRINT" X ERROR(UP TO 7 CHARS)":B$(N)="":GOTO 1030
1035 C=7-L:B$(N)=SPC(C)+B$(N)
1100 GOTO 840
1120 IF (32-A)=0 THEN 1290
1140 A$=""
1150 CURSOR 0,24:PRINT"ADDITIONAL DATA...Y OR N     ";
1155 BW$="X CAN'T ADD":CW$="   (WITHIN)"
1160 AW$="HOW MANY ITEMS DO YOU WANT TO ADD? "
1162 GET A$:IF A$="" THEN 1162
1164 IF A$="Y" THEN USR(62):GOTO 1170
1166 IF A$="N" THEN USR(62):GOTO 1290
1168 GOTO 1162
1170 CURSOR 0,24:PRINT AW$;
1171 CURSOR 36,24:INPUT" ";D
1173 IF A+D>32 THEN CURSOR 0,24:PRINT BW$;CW$;32-A;")"::D=0:GOTO 1171
1174 PRINT"■"::CURSOR 10,2:PRINT"* INPUT AN ADDITIONAL ITEM"
1175 CURSOR 0,5
1177 FOR N=A+1 TO A+D
1180 INPUT"ADDITIONAL ITEM: ";A$(N):L=LEN(A$(N))
1183 IF L>5 THEN PRINT" X ERROR(UP TO 5 CHARS)":A$(N)="":GOTO 1180
1185 B=5-L:A$(N)=SPC(B)+A$(N):PRINT
1190 INPUT"ADDITIONAL DATA: ";B$(N):L1=LEN(B$(N))
1193 IF L1>7 THEN PRINT" X ERROR(UP TO 7 CHARS)":B$(N)="":GOTO 1190
1195 C=7-L1:B$(N)=SPC(C)+B$(N):PRINT
1230 NEXT N:A=A+D:GOTO 840
1290 A$=""
1300 CURSOR 0,24:PRINT"DELETING DATA....Y OR N     ";
1320 GET A$:IF A$="" THEN 1320
1330 IF A$="Y" THEN USR(62):GOTO 1355
```

```
1340 IF A$="N" THEN USR(62):GOTO 2000
1350 GOTO 1320
1355 A$=""
1360 CURSOR 0,24:PRINT"WHICH ITEM DO YOU WANT TO DELETE? ";
1370 CURSOR 36,24:INPUT" ";H:PRINT"■";
1380 CURSOR 7,2:PRINT"* INPUT DELETION DATA"
1390 CURSOR 4,4:PRINT"ITEM:  ";A$(H);"   DATA:  ";B$(H)
1400 CURSOR 4,6:PRINT"CORRECT ?......Y OR N"
1410 GET A$:IF A$="" THEN 1410
1420 IF A$="Y" THEN A$(H)="":B$(H)="":GOTO 1455
1430 IF A$="N" THEN 840
1440 GOTO 1410
1455 IF H=32 THEN  1520
1457 IF H>32 THEN PRINT"■";CURSOR 3,3:PRINT"CAN'T DELETE ! ":GOTO 840
1470 FOR N=H+1 TO A
1480 A$(N-1)=A$(N):B$(N-1)=B$(N)
1510 NEXT N
1520 A=A-1:GOTO 840
2000 REM * CLASSIFICATION OF DATA *
2080 TT=0:HC=0
2090 SS=VAL(B$(1)):SD=VAL(B$(1))
2100 FOR N=1 TO A
2140 B(N)=VAL(B$(N))
2150 IF B(N)>SD THEN SD=B(N)
2160 IF B(N)<SS THEN SS=B(N)
2170 TT=TT+B(N)
2180 NEXT N
2182 HC=TT/A
2184 IF (HC<10000)*(HC>0) THEN HC=INT(HC*1000+.5)/1000:GOTO 2195
2186 IF (HC<100000)*(HC>0) THEN HC=INT(HC*10+.5)/10:GOTO 2195
2190 HC=INT(HC)
2192 IF TT>100000000 THEN TT=TT/1000:TT$=STR$(TT)+"K":GOTO 2197
2195 TT$=STR$(TT)
2197 SD$=STR$(SD):SS$=STR$(SS):HC$=STR$(HC)
2200 FOR N=1 TO A
2210 PT=(B(N)/TT)*100
2215 IF (PT<10)*(PT>0) THEN PT(N)=INT(PT*10+.5)/10:GOTO 2245
2240 PT(N)=INT(PT):PT$(N)=STR$(PT(N))
2245 PT$(N)=STR$(PT(N))
2250 NEXT N
2260 PRINT "■";
2270 CURSOR 4,0:PRINT"FIG.1  TABULATION LIST"
2310 CURSOR 2,3:PRINT TT$
2312 CURSOR 13,3:PRINT HC$
2314 CURSOR 21,3:PRINT SD$
2316 CURSOR 30,3:PRINT SS$
2340 CURSOR 2,7
2350 FOR N=1 TO A
2355 IF N=17 THEN PRINT"■";:CURSOR 0,7:GOTO 2365
2360 IF N=<16 THEN PRINT TAB(16);PT$(N):GOTO 2370
2365 PRINT TAB(35);PT$(N)
2370 NEXT N
2375 CURSOR 0,24:PRINT"DISPLAY THE STD.DEVIATION...Y OR N    ";
2380 GET A$:IF A$="" THEN 2380
2384 IF A$="Y" THEN USR(62):GOSUB 6000
2386 IF A$="N" THEN USR(62):GOTO 2400
2390 GOTO 2380
2400 PRINT"■";:CURSOR 0,24:PRINT"DATA TO BE FILED...Y OR N";SPC(10);
2410 A$=""
2420 GET A$:IF A$="" THEN 2420
2430 IF A$="Y" THEN NW=5:GOTO 2500
2440 IF A$="N" THEN 3000
2450 GOTO 2420
2500 CURSOR 0,24:PRINT"SWAP BEING EXECUTED!  WAIT A MINUTE! ";:USR(62)
2698 ON ERROR GOTO 8000
2700 SWAP,"DSPTO ?+F"
```

```
3000 PRINT"■";
3020 CURSOR 6,3:PRINT"♦♦ GENERATING A GRAPH ♦♦"
3030 CURSOR 6,7:PRINT"X-AXIS OF GRAPH...ITEM";" ┐"
3040 CURSOR 29,8:PRINT"|"
3050 CURSOR 6,9:PRINT"Y-AXIS OF GRAPH...DATA";" ┘"
3220 CURSOR 2,13:PRINT"* WHICH GRAPH ?"
3225 CURSOR 6,16:PRINT"HISTOGRAM.......PUSH 1"
3230 CURSOR 6,18:PRINT"SECTION.........PUSH 2"
3260 GET E:IF E=1 THEN NW$="HISTOGRAM TX":NW=1:GOTO 3305
3265 IF E=2 THEN NW$="GRAPH TX":NW=2:GOTO 3305
3290 GOTO 3260
3300 REM * READING DATA *
3305 PRINT"■";:CURSOR 4,10:PRINT"* FILE READING!    WAIT A MINUTE! *"
3308 ON ERROR GOTO 8000
3310 CHAIN,NW$
4000 END
5000 CURSOR TX,CY:INPUT"FILE NAME?...";NI$:CY=CY+6
5010 CURSOR TX-4,CY:PRINT"■ READING FILE      WAIT A MINUTE!"
5015 ON ERROR GOTO 7000
5020 ROPEN#1,NI$
5030 FOR I=0 TO 23
5040 INPUT#1,A$
5050 X$(I)=A$:A$=""
5060 NEXT I:CLOSE #1
5070 PRINT"■";
5080 FOR I=0 TO 23
5090 IF LEN(X$(I))<39 THEN PRINT X$(I):GOTO 5110
5100 PRINT X$(I);
5110 NEXT I
5120 CURSOR0,24:PRINT"PRINT ON PRINTER ?...Y OR N";
5130 GET A$:IF A$="" THEN 5130
5140 IF A$="Y" THEN 5165
5150 IF A$="N" THEN 5190
5160 GOTO 5130
5165 PRINT/P"■";
5170 FOR I=0 TO 23:PRINT/P"■";X$(I):NEXT I
5180 FOR I=0 TO 14:PRINT/P"■";:NEXT I
5190 NW$="":FOR I=0 TO 23:X$(I)="":NEXT I
5200 GOTO 400
6000 REM * STANDARD DEVIATION *
6010 D=0:HX=0
6020 FOR I=1 TO A
6030 D=B(I)*B(I)+D
6040 NEXT I
6050 HX=((D-A*HC*HC)/A)↑.5
6060 HX=INT(HX*1000+.5)/1000
6070 HX$=LEFT$(STR$(HX),7):L8=LEN(HX$):IX=7-L8:HX$=HX$+SPC(IX)
6080 CURSOR 13,2:PRINT"STD.DEV"
6090 CURSOR 13,3:PRINTHX$
6100 A$="N":RETURN
7000 PRINT"■";:CURSOR 5,5:PRINT"■■  CHECK FILE NAME AGAIN !!"
7010 CURSOR 7,10:PRINT"■  X.....PUSH"
7020 GET A$:IF A$="" THEN 7020
7030 IF A$="X" THEN A$="":GOTO 7100
7040 GOTO 7020
7100 NI$="":CLOSE #1:RESUME 400
8000 PRINT"■";:CURSOR 5,5:PRINT"■■  INSERT DISKETTE"
8010 CURSOR 7,10:PRINT"■  X.....PUSH"
8015 A$=""
8020 GET A$:IF A$="" THEN 8020
8030 IF A$="X" THEN A$="":GOTO 8100
8040 GOTO 8020
8100 IF NW=5 THEN RESUME 840
8110 IF (NW=1)+(NW=2) THEN RESUME 3000
```

```
10 REM * GRAPH TX *
20 DIM C(32),X(32),Z1$(8)
30 PRINT"■";:USR(62)
40 IF A=0 THEN CURSOR 10,10:PRINT"SIFT TO TABULATION TX!!":GX=2:GOTO 1900
80 CURSOR 5,6:PRINT"■■ TITLE?(UP TO 25 CHARS)"
90 CURSOR 5,8:INPUT"FIG. : ";TN$:TN$=LEFT$(TN$,25)
100 CURSOR 5,10:INPUT"X-AXIS(UP TO 7 CHARAS): ";KN$:KN$=LEFT$(KN$,7)
110 CURSOR 5,12:INPUT"Y-AXIS(UP TO 7 CARAS): ";DN$:DN$=LEFT$(DN$,7)
130 L5=LEN(DN$)
140 FOR D=1 TO L5
150 Z1$(D)=MID$(DN$,D,1)
160 NEXT D
170 IF(LEFT$(SS$,1)="-")+(LEFT$(SD$,1)="-") THEN MP=2: GOSUB 3000
230 LD=LEN(SD$)
240 IF(LEFT$(SD$,2)="0.")+(LEFT$(SD$,1)=".") THEN GOSUB 2100
250 IF SX=100 THEN 320
260 FOR I=1 TO LD
270 IF MID$(SD$,I,1)="." THEN LD=I-1:GOTO 310
280 LD=LEN(SD$)
290 NEXT I
300 REM * SCALE SETTING *
310 LM$=LEFT$(SD$,1):LM=VAL(LM$)
320 IF (LM=2)+(LM=6)+(LM=8) THEN LM=LM+1:GOTO 340
330 IF (LM=5) THEN LM=7:GOTO 340
340 SM=(LM+1)*(10↑(LD-1))
350 SM$=STR$(SM):SM$=LEFT$(SM$,1)
355 IF (MP=2)*((SM$="1")+(SM$="5")) THEN U=2:GOTO 372
360 IF (SM$="1")+(SM$="5") THEN U=4:GOTO 372
365 U=5
372 IF (MP=2) THEN U5=10:GOTO 380
374 U5=20
380 TS=SM/U5:HS=TS*U
400 FOR N=1 TO A
410 IF B(N)=0 THEN X(N)=0:C(N)=0:GOTO 460
420 K=(B(N)/TS):X(N)=INT(K)
430 IF (K<0)*(MP=2) THEN X1=(X(N)-K)*2:C(N)=INT(X1+.5):GOTO 450
440 X1=(K-X(N))*2:C(N)=INT(X1+.5)
460 NEXT N
490 PRINT"■";
500 CURSOR 10,0:PRINT"FIG. ";TN$
510 PRINT"      ┌──────────────────────────────────┐"
520 FOR I=1 TO 19
530 PRINT"      ├┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┼┤"
540 NEXT I
550 PRINT"      └──────────────────────────────────┘"
560 PRINT"■";
570 IF (MP=2)*(A<17) THEN C5=53693:GOTO 610
580 IF (MP=2) THEN C5=53694:GOTO 610
600 IF A<17 THEN C5=54093:GOTO 610
605 C5=54094
610 FOR N=1 TO A
620 IF (X(N)=0)*(C(N)=0) THEN 750
640 IF A>16 THEN T=1:GOTO 700
650 T=2
700 IF (C(N)=0) THEN POKE C5+N*T-(X(N)*40),184 :GOTO 790
720 IF (C(N)=1) THEN POKE C5+N*T-(X(N)*40)-40,189 :GOTO 790
730 IF (C(N)=2) THEN POKE C5+N*T-(X(N)*40)-40,184 :GOTO 790
750 IF (C(N)=-1) THEN POKE C5+N*T-(X(N)*40)-40,189 :GOTO 790
760 IF (C(N)=-2) THEN POKE C5+N*T-(X(N)*40)-40,184 :GOTO 790
790 NEXT N
810 CURSOR 0,21
815 U$=""
820 FOR J=0 TO U:U$=U$+"■":NEXT J
823 IF LEN(STR$(HS))>4 THEN 0=2:GOTO830
825 O=5-LEN(STR$(HS))
827 IF MP=2 THEN GOSUB 3830:GOTO 880
```

```
830 FOR I=0 TO 20
834 IF ABS(HS)>1000 THEN HS$=STR$(HS*I/U/1000):SX=3:GOTO 850
836 IF ABS(HS)<0.01 THEN HS$=STR$(INT((HS*I/U)*1000+.5)):SX=-3:GOTO 850
840 HS$=STR$(HS*I/U)
850 IF I=0 THEN PRINT TAB(Q);"0":GOTO 870
860 IF I/U=INT(I/U) THEN PRINT U$;TAB(Q);HS$
862 IF (I=20)*(SX=3) THEN PRINT"■■■■■";"K"
864 IF (I=20)*(SX=-3) THEN PRINT"■■■■";"MILIMETERS"
870 NEXT I
880 CURSOR 0,21
890 FOR I=1 TO A
900 I$=STR$(I)
910 IF A<=16 THEN T=5+I*2:P=15:GOTO 930
915 IF I/2<>INT(I/2) THEN 960
920 T=6+I:P=T
930 IF I=>10 THEN I1$=LEFT$(I$,1):I2$=RIGHT$(I$,1):PRINT"■";:GOTO 950
935 IF I$="2" THEN PRINTTAB(P);"■■";KN$;"■■■"
940 PRINTTAB(T);"■";I$;"■";:GOTO 960
950 PRINTTAB(T);"■";I1$;"■■";I2$;"■■";
960 NEXT I
970 PRINT"■";"■■■";
980 FOR D=1 TO L5:PRINT Z1$(D):NEXT D
1000 CURSOR 0,24:PRINT"GRAPH REPRESENTED BY EXPRESSION Y OR N";:USR(62)
1010 A$=""
1020 GET A$:IF A$="" THEN 1020
1030 IF A$="Y" THEN P=0:Q=24:GOTO 1070
1040 IF A$="N" THEN 1400
1050 GOTO 1020
1070 CURSOR 0,24:PRINT"DATA: LINE(6), CURVE(7) ..... 6 OR 7";
1090 A$=""
1100 GET A$:IF A$="" THEN 1100
1110 IF A$="6" THEN GX=6:GOTO 1200
1120 IF A$="7" THEN GX=7:GOTO 1300
1130 GOTO 1100
1200 ON ERROR GOTO 6005
1205 SWAP,"REGRESSION LINE"
1210 IF GX=6 THEN 1400
1300 ON ERROR GOTO 6005
1305 SWAP,"REGRESSION CURVE"
1400 CURSOR 0,24:PRINT"TO BE FILED? ..... Y OR N      ";:USR(62)
1410 A$=""
1420 GET A$:IF A$="" THEN 1420
1430 IF A$="Y" THEN P=0:Q=24:GOTO 1470
1440 IF A$="N" THEN P=3:Q=8:GOTO 1600
1450 GOTO 1420
1470 CURSOR 0,24:PRINT"FILE READING!      WAIT A MINUTE!     ";
1475 ON ERROR GOTO 6005
1480 CHAIN,"DSPTO ?+F"
1600 CURSOR 0,24:PRINT"GRAPH CHANGED? ....Y OR N           ";:USR(62)
1610 A$=""
1620 GET A$:IF A$="" THEN 1620
1630 IF A$="Y" THEN GX=4:GOTO 1900
1640 IF A$="N" THEN GX=2:GOTO 1900
1650 GOTO 1620
1670 CURSOR 0,24:PRINT"FILE READING!     WAIT A MINUTE!!     ";
1900 ON ERROR GOTO 6000
1910 CHAIN,"TABULATION TX"
1960 END
2000 REM * DECIMAL POINT *
2100 FOR J=1 TO LD
2140 IF MID$(SD$,J,1)="." THEN P=J:GOTO 2160
2150 IF VAL(MID$(SD$,J,1))<>0 THEN S=J:LM1$=MID$(SD$,J,1)
2160 NEXT J
2170 SX=100:LD=P-S+1:LM=VAL(LM1$)
2180 RETURN
3000 REM * +&- SCALE (3000-4000) *
```

```
3005 SD=VAL(SD$):SS=VAL(SS$)
3010 IF ABS(SD)=>ABS(SS) THEN RETURN
3030 LD=LEN(SS$)-1:SS$=RIGHT$(SS$,LD)
3040 SD$=SS$
3050 RETURN
3830 FOR I=0 TO 20
3840 IF ABS(HS)>1000 THEN HS=(HS/1000):SX=3:GOTO 3860
3850 IF ABS(HS)<0.01 THEN HS=INT(HS*1000+.5):SX=-3:GOTO 3860
3860 HS1$=STR$(HS*(I-10)/U):HS2$=STR$(-1*HS*(10-I)/U)
3870 IF I=0 THEN PRINT TAB(0-1):HS2$:GOTO 3920
3880 IF (I<10)*(I/U=INT(I/U)) THEN PRINT U$:TAB(0-1);HS2$
3890 IF I=10 THEN PRINT U$;TAB(0);"0":GOTO 3920
3900 IF (I>10)*(I/U=INT(I/U)) THEN PRINT U$:TAB(Q);HS1$
3920 IF (I=10)*(SX=3) THEN PRINT"■■■■■";"K"
3940 IF (I=10)*(SX=-3) THEN PRINT"■■■■";"MILIMETERS"
3960 NEXT I
4000 RETURN
6000 PRINT"■";
6005 CURSOR P,0:PRINT"■■ INSERT DISKETTE  ........PUSH X";
6010 A$=""
6020 GET A$:IF A$="" THEN 6020
6030 IF A$="X" THEN 6050
6040 GOTO 6010
6050 RESUME
```

```
10 REM * HISTOGRAM TX*
20 DIM C(32),X(32),Z1$(8)
30 PRINT"■";
40 IF A=0 THEN CURSOR 10,10:PRINT"SHIFTING TO TABULATION TX!!":GX=2:GOTO 1500
80 PRINT"■■■■* TITLE(UP TO 25 CHARAS)":PRINT
90 INPUT"FIG. : ";TN$:TN$=LEFT$(TN$,25):PRINT
100 INPUT"X-AXIS(UP TO 7 CHARS): ";KN$:KN$=LEFT$(KN$,7):PRINT
110 INPUT"Y-AXIS(UP TO 7 CHARS): ";DN$:DN$=LEFT$(DN$,7)
130 L5=LEN(DN$)
140 FOR D=1 TO L5
150 Z1$(D)=MID$(DN$,D,1)
160 NEXT D
230 LD=LEN(SD$)
240 IF (LEFT$(SD$,2)="0.")+(LEFT$(SD$,1)=".") THEN GOSUB 2100
250 IF SX=100 THEN 320
260 FOR I=1 TO LD
270 IF MID$(SD$,I,1)="." THEN LD=I-1:GOTO 310
280 LD=LEN(SD$)
290 NEXT I
300 REM * SCALE SETTING *
310 LM$=LEFT$(SD$,1):LM=VAL(LM$)
320 IF (LM=2)+(LM=6)+(LM=8) THEN LM=LM+1:GOTO 340
330 IF (LM=5) THEN LM=7:GOTO 340
340 SM=(LM+1)*(10↑(LD-1))
350 SM$=STR$(SM):SM$=LEFT$(SM$,1)
360 IF (SM$="1")+(SM$="5") THEN V=4:GOTO 380
370 V=5
380 TS=SM/20:HS=TS*V
400 FOR N=1 TO A
410 IF B(N)=0 THEN X(N)=0:C(N)=0:GOTO 460
420 K=(B(N)/TS)-.5:X(N)=INT(K)
440 X1=(K-X(N))*4:C(N)=INT(X1+.5)
460 NEXT N
490 PRINT"■";
500 PRINT"■■■■■■■■■■■■FIG.  ";TN$
510 PRINT"      ┌──────────────────────────────────┐"
520 FOR I=1 TO 19
530 PRINT"      ├──────────────────────────────────┤"
540 NEXT I
550 PRINT"      └──────────────────────────────────┘"
590 PRINT"■";
600 G$="■":FOR I=1 TO 21:G$=G$+"■":NEXT
610 FOR N=1 TO A
640 T=5+N*2
650 IF A>16 THEN T=6+N
655 PRINT G$:TAB(T);
660 IF X(N)>=0 THEN PRINT"■";"■■";
665 IF (X(N)<0)*(C(N)<=2) GOTO 800
667 IF X(N)<0 THEN PRINT"■";:GOTO 800
670 IF X(N)=0 THEN 700
680 FOR I=1 TO X(N):PRINT"■■■";:NEXT
700 IF C(N)=1 THEN PRINT"▁";:GOTO 800
720 IF C(N)=2 THEN PRINT"▄";:GOTO 800
740 IF C(N)=3 THEN PRINT"■";:GOTO 800
760 IF C(N)=4 THEN PRINT"■";:GOTO 800
770 PRINT"";
800 NEXT N
810 PRINT G$;
815 V$=""
820 FOR J=0 TO V:V$=V$+"■":NEXT J
823 IF LEN(STR$(HS))>4 THEN 0=1:GOTO830
825 0=5-LEN(STR$(HS))
830 FOR I=0 TO 20
834 IF HS>1000 THEN HS$=STR$(HS*I/V/1000):SX=3:GOTO 850
836 IF HS<0.01 THEN HS$=STR$(INT((HS*I/V)*1000+.5)):SX=-3:GOTO 850
840 HS$=STR$(HS*I/V)
```

```
850 IF I=0 THEN PRINT TAB(Q);"0":GOTO 870
860 IF I/U=INT(I/U) THEN PRINT U$;TAB(Q);HS$
862 IF (I=20)*(SX=3) THEN PRINT"■■■■■";"K"
864 IF (I=20)*(SX=-3) THEN PRINT"■■■■";"MILIMETRS"
870 NEXT I
880 PRINT"■";G$;
890 FOR I=1 TO A
900 I$=STR$(I)
910 IF A<=16 THEN T=5+I*2:P=15:GOTO 930
915 IF I/2<>INT(I/2) THEN 960
920 T=6+I:P=T
930 IF I=>10 THEN I1$=LEFT$(I$,1):I2$=RIGHT$(I$,1):PRINT"■";:GOTO 950
935 IF I$="2" THEN PRINT TAB(P);"■■";KN$;"■■■"
940 PRINT TAB(T);"■";I$;"■";:GOTO 960
950 PRINT TAB(T);"■";I1$;"■■";I2$;"■■";
960 NEXT I
970 PRINT"■";"■■■";
980 FOR D=1 TO L5:PRINT Z1$(D):NEXT D
1000 PRINT"■";G$;"■■■TO BE FILED? ..... Y OR N ";:USR(62)
1010 A$=""
1020 GET A$:IF A$="" THEN 1020
1030 IF A$="Y" THEN 1070
1040 IF A$="N" THEN 1300
1050 GOTO 1020
1070 CURSOR 0,24:PRINT"FILE READING!       WAIT A MINUTE!!";
1180 ON ERROR GOTO 5000
1200 CHAIN,"DSPTO ?+F"
1300 PRINT"■";G$;"■■■GRAPH CHANGED? ......... Y OR N ";:USR(62)
1310 A$=""
1320 GET A$:IF A$="" THEN 1320
1330 IF A$="Y" THEN GX=4:GOTO 1480
1340 IF A$="N" THEN GX=2:GOTO 1480
1350 GOTO 1310
1480 ON ERROR GOTO 5005
1500 CHAIN,"TABULATION TX"
1960 END
2000 REM * DECIMAL POINT *
2100 FOR J=1 TO LD
2140 IF MID$(SD$,J,1)="," THEN P=J:GOTO 2160
2150 IF VAL(MID$(SD$,J,1))<>0 THEN S=J:LM1$=MID$(SD$,J,1)
2160 NEXT J
2170 SX=100:LD=P-S+1:LM=VAL(LM1$)
2180 RETURN
5000 CURSOR 0,24:PRINT"■■ INSERT THE DISKETTE ......PUSH X";:GOTO 5020
5010 PRINT"■";:CURSOR 3,8:PRINT"■■ INSERT THE DISKETTE ......PUSH X"
5020 A$=""
5030 GET A$:IF A$="" THEN 5030
5040 IF A$="X" THEN 5060
5050 GOTO 5020
5060 RESUME
```

```
10 REM* COPY ON PRINTER OR DISK *
20 USR(62)
25 IF A=0 THEN PRINT"■"::CURSOR 10,10:PRINT"SHIFTING TO TABULATION!!":GOTO 430
30 Z=53248:DIM Y(222),X$(24)
40 FOR I=0 TO 221:READ Y(I):NEXT I
50 P$="":FOR N=0 TO 23:FOR M=1 TO 40
60 ZZ=PEEK(Z):B=Y(ZZ)
70 IF B=32 THEN P$=P$+" ":Z=Z+1:NEXT M:GOTO 90
80 P$=P$+CHR$(B):Z=Z+1:NEXT M
90 X$(N)=P$:P$="":NEXT N
100 USR(62):PRINT"■"::CURSOR 3,7:PRINT"■■■ OUTPUT ON PRINTER?     Y OR N"
110 GET A$:IF A$="" THEN 110
120 IF A$="Y" THEN USR(62):GOTO 142
130 IF A$="N" THEN USR(62):GOTO 190
140 GOTO 110
142 ON ERROR GOTO 4000
145 PRINT#P"■";
150 FOR N=0 TO 23
160 PRINT#P"■";X$(N)
170 NEXT N
180 FOR K=1 TO 5:PRINT#P:NEXT K
190 PRINT"■"::CURSOR 3,7:PRINT"◆◆ TO BE FILED ?       Y OR N"
200 GET A$:IF A$="" THEN 200
210 IF A$="Y" THEN USR(62):GOTO 240
220 IF A$="N" THEN USR(62):GOTO 320
230 GOTO 200
240 CURSOR 3,9:PRINT"FILE NAME? (UP TO 16 CHARS)":INPUT "    ";N$
250 L2=LEN(N$):IF L>16 THEN PRINT"ERROR":N$="":GOTO 240
260 CURSOR 3,14:PRINT"◆◆◆ UNDER FILING!!   WAIT A MINUTE!!"
265 ON ERROR GOTO 3000
270 WOPEN #1,N$
280 FOR I=0 TO 23
290 A$=X$(I)
300 PRINT#1;A$
310 NEXT I:CLOSE#1
320 IF NW=5 GOTO 450
330 PRINT"■";
340 USR(62):CURSOR 3,8:PRINT"◆◆ REPORT TO BE MADE?..... Y OR N"
350 A$=""
360 GET A$:IF A$="" THEN 360
370 IF A$="Y" THEN USR(62):GOTO 400
380 IF A$="N" THEN USR(62):GOTO 430
390 GOTO 360
400 PRINT"■"::CURSOR 3,12:PRINT"■■■ REPORT GENERATOR TEXT CALLING!":PRINT
410 CURSOR 8,14:PRINT"WAIT A MINUTE  !! "
415 ON ERROR GOTO 3500
420 CHAIN,"REPORT TX"
430 GX=2
435 ON ERROR GOTO 3500
440 CHAIN,"TABULATION TX"
450 FOR I=0 TO 24:X$(I)="":NEXT I
490 ON ERROR GOTO 3500
500 END
600 DATA 32,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83
700 DATA 84,85,86,87,88,89,90,251,205,221,203,209,48,49,50,51,52,53
800 DATA 54,55,56,57,45,61,59,47,46,44,229,244,236,218,227,226,215,212
900 DATA 230,232,194,193,196,199,207,202,0,225,254,200,250,95,248,241,247
1000 DATA 63,204,219,220,233,245,58,94,60,91,243,93,64,201,62,252,92,198
1100 DATA 223,208,206,211,210,255,33,34,35,36,37,38,39,40,41,43,42,222
1200 DATA 246,235,234,195,197,239,240,228,231,238,237,224,253,216
1300 DATA 213,242,249,217,214,0,161,154,159,156,146,170,151,152,166,175
1400 DATA 169,184,179,176,183,158,160,157,164,150,165,171,163,155,189
1500 DATA 162,187,153,130,135,140,188,167,172,145,147,148,149,180,181
1600 DATA 182,174,173,186,178,185,168,177,131,136,141,134
1700 DATA 132,137,142,191,133,138,143,190,129,139,144
1800 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,112,113,114,115,116
```

```
1900 DATA 117,118,119,120,121,122,123,125,126,"END"
3000 IF ERN=50 THEN F=2:GOTO 3500
3005 PRINT"■"::CURSOR 5,5:PRINT"CHANGE FILE NAME.......PUSH X"
3010 GET A$:IF A$="" THEN 3010
3020 IF A$="X" THEN 3100
3030 GOTO 3010
3100 N$="":KILL #1:RESUME 190
3500 PRINT"■"::CURSOR 5,5:PRINT"INSERT A DISKETTE......PUSH X"
3510 GET A$:IF A$="" THEN 3510
3520 IF A$="X" THEN 3600
3530 GOTO 3510
3600 IF NW=5 THEN RESUME 500
3610 IF F=2 THEN RESUME 190
3620 RESUME 330
4000 PRINT"■"::CURSOR 5,5:PRINT"SWITCH ON PRINTER......PUSH X"
4010 GET A$:IF A$="" THEN 4010
4020 IF A$="X" THEN 4100
4030 GOTO 4010
4100 RESUME 100
```

```
5 REM * REPORT TX*
10 PRINT"■":DIM R$(23) :I$="":L=1
20 A$="                    DATE :"
30 B$="                    NAME :"
40 C$="                                  "
50 D$="      TITLE:"
60 PRINT TAB(10):"******************"
65 PRINT TAB(10):"*                  *"
70 PRINT TAB(10):"* REPORT MAKING *"
75 PRINT TAB(10):"*                  *"
77 PRINT TAB(10):"******************":PRINT:PRINT
80 PRINT" MAKE BY THE FOLLOWING PROCEDURE":PRINT
90 INPUT"DATE (UP TO 11 CHARS):  ":I$:R$(1)=A$+I$:I$="":PRINT
100 INPUT"NAME (UP TO 11 CHARS):  ":I$:R$(2)=B$+I$:I$="":PRINT
110 R$(3)=C$
120 INPUT"TITLE(UP TO 20 CHARS):  ":I$:R$(4)=D$+I$:I$="":PRINT
130 R$(5)=C$
135 R$(6)="1. PURPOSE ":PRINT"■":PRINT"*  PURPOSE "
140 PRINT:PRINT"PRESS 'CR' KEY EVERY LINE.":PRINT"(WITHIN 3 LINES)"
145 PRINT:PRINT"WHEN ENDING HALFWAY, INPUT 'END' AFTER  'CR' KEY."
150 Z=7 :GOSUB 1150
160 R$(10)="2.METHOD":PRINT"■":PRINT"*  METHOD "
170 PRINT:PRINT"PRESS 'CR' KEY EVERY LINE.":PRINT"(WITHIN 3 LINES)"
175 PRINT:PRINT"WHEN ENDING HALFWAY, INPUT 'END' AFTER  'CR' KEY."
180 Z=11:GOSUB 1150
190 R$(14)="3.RESULT":PRINT"■":PRINT"*  RESULT"
200 PRINT:PRINT"PRESS 'CR' KEY EVERY LINE.":PRINT"(WITHIN 3 LINES)"
210 PRINT:PRINT"WHEN ENDING HALFWAY, INPUT 'END' AFTER  'CR' KEY."
220 Z=15:GOSUB 1150
230 R$(18)="4.EXAMINATION":PRINT"■":PRINT"*  EXAMINATION "
240 PRINT:PRINT"PRESS 'CR' KEY EVERY LINE.":PRINT"(WITHIN 3 LINES)"
250 PRINT:PRINT"WHEN ENDING HALFWAY, INPUT 'END' AFTER  'CR' KEY."
260 Z=19:GOSUB 1150
300 PRINT"■":FOR I=1 TO 21:PRINT R$(I):NEXT I
310 REM** CORRECTION **
320 CURSOR 0,23:PRINT"CORRECT: 'E' KEY       RIGHT: 'R' KEY";
325 GET A$:IF A$="" THEN 325
330 IF A$="E" THEN 350
335 IF A$="R" THEN 440
340 GOTO 325
350 CURSOR 0,23:PRINT"INPUT LINE NO. OF WRONG LINE          ":
355 CURSOR 36,23:INPUT"":A$:N=VAL(A$)
360 CURSOR 0,23:PRINT"INPUT CURSOR ANEW                 ":PRINT R$(N):R$(N)=""
370 CURSOR 0,23:INPUT"  ":I$
380 IF N=1 THEN R$(1)=A$+I$:GOTO 300
382 IF N=2 THEN R$(2)=B$+I$:GOTO 300
384 IF N=4 THEN R$(4)=D$+I$:GOTO 300
385 IF (N=14)+(N=18)+(N=6)+(N=10) THEN R$(N)=I$:GOTO 300
386 L=LEN(I$)
387 IF L>37 THEN PRINT"■";" × ERROR (UP TO 37-CHARAS)":I$="":GOTO 370
390 R$(N)="  "+I$:GOTO 300
440 REM** OUTPUT RESULTS ON PRINTER **"
445 REM** OUTPUT RESULTS ON DISK **"
450 CURSOR 0,23:PRINT"*OUTPUT OF RESULTS:PRINTER-P   DISK-D":A$=""
460 GET A$:IF A$="" THEN 460
470 IF A$="P" THEN 482
475 IF A$="D" THEN 505
480 GOTO 460
482 ON ERROR GOTO 4000
485 PRINT#P"■";
490 FOR I=1 TO 21:PRINT#P R$(I):NEXT I
492 FOR I=1 TO 15:PRINT#P:NEXT I
494 CURSOR0,23:PRINT" * TO BE FILED ON DISK  Y OR N *      ":A$=""
496 GET A$:IF A$="" THEN 496
498 IF A$="Y" THEN 506
500 IF A$="N" THEN 900
```

```
502 GOTO 496
505 REM ** DATA INPUT TO DISK **
506 PRINT"■":"■■":"** MAY PREVIOUS DATA BE DELETE? Y OR N":PRINT:PRINT:A$=""
507 GET A$:IF A$="" THEN 507
508 IF A$="Y" THEN PRINT"PREVIOUS FILE NAME: ":GOTO 512
509 IF A$="N" THEN  N$="":GOTO 516
510 GOTO 507
512 INPUT" ";NR$
513 ON ERROR GOTO 2000
514 DELETE NR$:NR$=""
516 INPUT"FILE NAME..(UP TO 16 CHARS)";NR$:L1=LEN(NR$)
518 IF L1>16 THEN PRINT" X  ERROR":GOTO 512
520 ON ERROR GOTO 2000
525 WOPEN #9,NR$
530 FOR N=1 TO 21
540 P1$=R$(N)
550 PRINT #9,P1$
560 NEXT:CLOSE #9
900 CX=10
1000 CHAIN,"TABULATION TX"
1140 END
1150 REM *ERROR ROUTINE*
1155 S=0:PRINT:PRINT
1160 INPUT"   ";I$:L=LEN(I$)
1170 IF L>37 THEN PRINT" X ERROR (UP TO 37 CHARS)":I$="":GOTO 1160
1180 IF I$="END" THEN I$="":R$(Z)=C$:GOTO 1210
1190 R$(Z)="  "+I$:I$="":S=S+1:IF S>2 THEN 1210
1200 Z=Z+1:GOTO 1160
1210 RETURN
2000 PRINT"■";
2004 IF ERN=50 THEN CURSOR 5,5:PRINT"INSERT THE DISKETTE ....PUSH X":GOTO 2010
2008 CURSOR 5,5:PRINT"CHECK FILE NAME AGAIN..PUSH X"
2010 A$=""
2020 GET A$:IF A$="" THEN 2020
2030 IF A$="X" THEN 2050
2040 GOTO 2020
2050 IF ERN=40 THEN 2070
2060 NR$="":KILL #9:RESUME 506
2070 NR$="":RESUME 506
4000 PRINT"■"::CURSOR 5,5:PRINT"SWITCH ON PRINTER...PUSH X"
4010 A$=""
4020 GET A$:IF A$="" THEN 4020
4030 IF A$="X" THEN 4050
4040 GOTO 4020
4050 RESUME 300
```

```
10 REM * REGRESSION LINE *
20 IF GX=6 THEN GOSUB 1000:GOTO 280
100 DIM A(32),B(32)
110 PRINT"@";
120 XA=0:YB=0:S2=0:T2=0
130 CURSOR 5,5:INPUT"NUMBER OF CALCULATIONS?";A
140 FOR I=1 TO A
150 CURSOR 5,7:INPUT"VALUE OF X-AXIS ";A(I)
160 CURSOR 5,9:INPUT"VALUE OF Y-AXIS ";B(I)
170 NEXT I
280 FOR I=1 TO A
290 XA=A(I)+XA:YB=B(I)+YB
300 S2=A(I)*A(I)+S2:T2=A(I)*B(I)+T2
310 NEXT I
320 A2=(T2-XA*YB/A)/(S2-XA*XA/A)
330 B2=YB/A-A2*XA/A
335 A2=INT(A2*1000+.5)/1000:B2=INT(B2*1000+.5)/1000
340 CURSOR 0,24:PRINT"                                                    ";
345 CURSOR 0,24:PRINT"Y=AX+B";
350 CURSOR 10,24:PRINT"A=";A2;
360 CURSOR 25,24:PRINT"B=";B2;
365 ON ERROR GOTO 2000
370 PRINT/P "Y=AX+B"
375 PRINT/P
380 PRINT/P "A=";A2,"B=";B2
385 FOR I=1 TO 3:PRINT/P:NEXT I
390 PRINT/P "A(I)=","B(I)="
400 FOR I=1 TO A
410 PRINT/P A(I),B(I)
420 NEXT I
490 FOR I=1 TO 15:PRINT/P:NEXT I
500 END
1000 DIM A(32)
1005 XA=0:YB=0:S2=0:T2=0
1010 FOR I=1 TO A
1020 T$=LEFT$(A$(I),1):T=ASC(T$)
1030 IF (T=>48)+(T=<57) THEN A(I)=VAL(A$(I)):GOTO 1050
1040 CURSOR 0,24:PRINT"CALCULATION IMPOSSIBLE-NOT NUMERIC!!":GOTO 1100
1050 NEXT I:RETURN
1100 FOR I=1 TO 3:USR(62):NEXT I
1110 CURSOR 0,24:PRINT"SHIFTING TO ORIGINAL TEXT. WAIT PLEASE!"
1120 CHAIN,"GRAPH TX"
2000 PRINT"@";
2010 CURSOR 3,8:PRINT"■ SWITCH ON PRINTER ...PUSH X"
2020 A$=""
2030 GET A$:IF A$="Y" THEN 2030
2040 IF A$="X" THEN 2060
2050 GOTO 2020
2060 RESUME 365
```

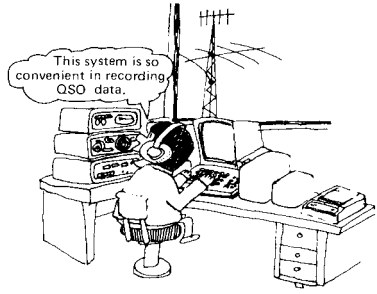
```
10 REM * REGRESSION CURVE *
20 IF GX=7 THEN GOSUB 1000:GOTO 280
100 DIM A(32),B(32),Z(32),X1(32),Y1(32)
110 PRINT"■":
120 XX=0:YY=0:S3=0:T3=0
130 CURSOR 5,5:INPUT"NUMBER OF CALCULATIONS ?";A
140 FOR I=1 TO A
150 CURSOR 5,7:INPUT"VALUE OF X-AXIS";A(I)
160 CURSOR 5,9:INPUT"VALUE OF Y-AXIS";B(I)
170 NEXT I
280 FOR I=1 TO A
290 X1(I)=A(I)-A(1):Y1(I)=B(I)-B(1)
300 IF I=1 THEN 340
310 Z(I)=Y1(I)/X1(I)
320 XX=X1(I)+XX:YY=Z(I)+YY
330 S3=X1(I)*X1(I)+S3:T3=X1(I)*Z(I)+T3
340 NEXT I
350 AA=(T3-XX*YY/(A-1))/(S3-XX*XX/(A-1))
360 BB=YY/(A-1)-AA*XX/(A-1)
365 CC=AA*A(1)*A(1)-BB*A(1)+B(1)
370 BB=BB-2*AA*A(1)
375 AA=INT(AA*1000+.5)/1000:BB=INT(BB*1000+.5)/1000
377 CC=INT(CC*1000+.5)/1000
380 CURSOR 0,24:PRINT"                                                    ";
385 CURSOR 0,24:PRINT"Y=AX↑2+BX+C";
390 CURSOR 13,24:PRINT"A=";AA;
395 CURSOR 22,24:PRINT"B=";BB;
400 CURSOR 31,24:PRINT"C=";CC;
405 ON ERROR GOTO 2000
410 PRINT/P "Y=AX↑2+BX+C"
420 PRINT/P
430 PRINT/P"A=";AA,"B=";BB,"C=";CC
440 FOR I=1 TO 3:PRINT/P:NEXT I
450 PRINT/P"X-VALUE ","Y-VALUE"
460 FOR I=1 TO A
470 PRINT/P A(I),B(I)
480 NEXT I
490 FOR I=1 TO 15:PRINT/P:NEXT I
500 END
1000 DIM A(32),Z(32),X1(32),Y1(32)
1005 XX=0:YY=0:S3=0:T3=0
1010 FOR I=1 TO A
1020 T$=LEFT$(A$(I),1):T=ASC(T$)
1030 IF (T=>48)+(T=<57) THEN A(I)=VAL(A$(I)):GOTO 1050
1040 CURSOR 0,24:PRINT"CALUCULATION IMPOSSIBLE-NOT NUMERIC":GOTO 1100
1050 NEXT I:RETURN
1100 FOR I=1 TO 3:USR(62):NEXT I
1110 CURSOR 0,24:PRINT"SIFTING TO ORIGINAL TEXT!  WAIT PLEASE!"
1120 CHAIN,"GRAPH TX"
2000 PRINT"■":
2010 CURSOR 3,8:PRINT"■ SWITCH ON PRINTER...PUSH X"
2020 A$=""
2030 GET A$:IF A$="" THEN 2030
2040 IF A$="X" THEN 2060
2050 GOTO 2020
2060 RESUME 405
```

Let's computerize your amateur radio log

He, a great guy, enters his shack swiftly and switches his rig on. Then his shack is suddenly flooded with familiar and strange call signs that have come in one after another. He looked then so pleased.

His shack is equipped with the Series MZ-80 log processing system, and his disks are filled with lots of QSO data.

He switches on this time his MZ-80 system and inputs the RUN command. Then the log processing system is activated and the computer display shows the information shown in Photo 1, asking him for the next input command.



He turns again to his rig and tunes the 2 m Band, when he hears then a call sign of G6ABC at 144.40 MHz. The S-meter needle swings over just when he changes the direction of his antenna toward the east a little bit from the south. It is a long time since we had has such a good QSO.

He taps 1 and CR at once to input QSO data. His computer comes to ask him in succession for data in order to make a long.

```
AMATEUR RADIO LOG

◆ INDEX ◆
    1.  INPUT OF QSO DATA
    2.  ADDITION OR CORRECTION
    3.  VARIOUS LISTS
    4.  END

SET YOUR REQUEST AT NUMBER : ■
```

Photo 1

Input of QSO data

1. Call sign (up to 10 characters) ? **G6ABC** CR
2. Date of QSO (6 digits) ? **800111** CR
   Today is 11 January 1980.
3. Time of QSO start (Y OR N) ? **Y** CR
   (Enter time data after depression of **Y** and CR when inputting at the present, and enter after pressing **N** CR in other cases.)
4. The QSO band list is displayed.
   He input **9** CR as the code of 144 MHz band.
5. Transmitting mode (up to 3 characters) ? **SSB** CR
6. RST (up to 3 characters) ? **59** CR
7. QRA (up to 8 characters) ? **JONES** CR
8. CITY OR COUNTRY (up to 8 characters) ? **ENGLAND** CR
9. QSL card RECEIVE (Y OR N) ? **Y** CR
10. QSL issuance NO (up to 4 characters) ? **N** CR
11. Classification comments (up to 6 characters) ? **RSGB** CR
    G6ABC is a member of RSGB.
12. Remarks data (up to 9 characters) ? **N** CR

Is the computer display awaiting commands ?
Do you continue to input?
(If you continue, input Y, and if not, input N) ? **Y** CR

"I will continue," he says. His computer continues to store QSO data on the disk with continuous clicking sounds of head movement. Before long he QSY's to the band where club members are waiting.

Addition or correction of data

If you press 2 and [CR] while the computer is awaiting commands as shown in Photo 1, it asks as follows.

◆ ADDITION OR CORRECTION FOR QSO DATA
CALL SIGN PLEASE

If you input a certain call sign, the computer display will show filed data as shown in Photo 2. To correct the data, input a proper correction item number through the keyboard and follow the same procedure as for inputting data.

On the completion of correction, press [O] [CR] to store corrected data in the file. If the data does not need to be corrected, it can be passed by inputting [O] [CR]. After the completion of data outputting. The computer clears its buffer area and returns to the state shown in Photo 1.

Photo 2.

Outputting various classification lists

The computer, when staying in the state shown in Photo 1, displays the message of Photo 3 if its 3 and [CR] are pressed, and it awaits again the next command. Then, input the identification number of a desired list to choose any one of the following seven sorts of classification lists.

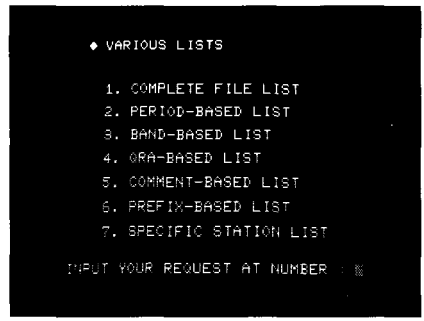


Photo 3.

1) Complete file list

If you press [1] and [CR] while the display is in the state shown in Photo 3, the computer asks you as follows.

◆◆ COMPLETE FILE LIST ◆◆
DATA OUTPUT TO PRINTER (Y OR ANY KEY) ?

If you want data to be printed out, press Y and if you want to obtain data on the display, input any other key. In the latter case, the computer stops after having displayed data of 15 stations and asks you whether or not you wish to advance to the following lists. If some key is then pushed, the subsequent data appears on the display. When data is completely output, the computer asks again showing the following message.

ADVANCE TO THE NEXT ?

And the system returns to the state shown in Photo 1 the instant any key is depressed.
(When you use the printer to output data, the same message as above appears as above appears on the CRT screen as soon as all data has been printed out.)

2) Period-based list (This list shows QSO data within a specified period.)

If [2] and [CR] are pressed in the state shown in Photo 3, the computer asks displaying like

"FROM YEAR, MONTH, DAY ?"

Input then the start date (6 digits) in the same manner as for inputting data, and the computer displays

"UNTIL YEAR, MONTH, DAY ?"

Input the date of termination in the same manner as above in response to the question. If the computer asks you

"DATA OUTPUT TO PRINTER (Y OR ANY KEY) ? "

choose either mode in the same way as given in 1) above.

3) QSO-band-based list

Press [3] and [CR] in the state shown in Photo 3, then the QSO band list appears on the computer screen. Now, you have only to input your desired band number through the keyboard. The computer then asks you to choose either output mode--printer or display.

Choose the one you want in the same manner as in **1)** above.

4) QRA-based retrieval list

Press [4] and [CR] in the state shown in Photo 3, and the computer shows as follows.

"QRA (UP TO 8 CHARACTERS) ?"

Input then QRA of station you wish to retrieve. **(Needless to say, no data comes out unless the input and the corresponding filed data coincide in format. When filed data has to the format [J] [O] [N] [E] [S], it will not come out if the input has the format [J] [ ] [O] [ ] [N] [ ] [E] [ ] [S]).** The computer asks you if your data should be output to the printer. Take your choice in the same manner as in **1)** above.

5) Classification-comment-based retrieval list

Press [5] and [CR] in the state shown in Photo 3. The computer displays as follows.

COMMENT (UP TO 6 CHARACTERS) ?

Specify the club call recorded as classification comment then the listed club members and their data are output. The following operation is the same as in **1).**

6) Prefix-based list

Depress [6] and [CR] in the state shown in Photo 3, and the prefix-based list is displayed. Input the prefix of the call sign you need. Thereafter, take your choice of the printer and CRT screen as an output device by the same procedure as given in **1)** above.

7) Specific station list

Depress [7] and [CR] in the state shown in Photo 3, then the computer displays as follows.

CALL SIGN PLEASE ?

Input then the call sign of your desired station through the keyboard. If the specified call sign is recorded on your disk, its data is output. If you have had many QSO's with the same station, all its data will be output.

So far we have described the way to use this sample program. Next, we will show examples of output to the printer that can be made by this program.

◆◆ COMPLETE FILE LIST ◆◆

P - 1

CALL SIGN	DATE	TIME	BAND	EM.	RST	QRA	QTH	Q/NO	R	COMME.	REMARKS
DL5SOS/AM	21/7, 60	12:33	21	SSB	53	CURT	GERMANY	--	*	------	TS520D
JA3AER	1/12, 60	9:38	7	AM	59	ARAKAWA	JAPAN	342	R	SHARP	6JS6
G3QRN	3/8, 73	5:14	21	SSB	55	JOHN	LONDON	--	*	ISWL	FT101E
K6BX	3/1, 75	5:34	14	CW	599	CLIF	USA	45	*	CHC	FT401
K5ZMS	4/3, 75	17: 5	50	CW	589	RAY	USA	100	*	SMIRK	TS-600
AA6AA	2/5, 75	15:31	14	CW	589	JOE	USA	--	*	------	CONTEST
DL5QRK	3/9, 75	15:28	14	CW	599	KEN	GERMANY	--	*	------	QRP
HB9QRO	10/1, 76	23: 1	144	SSB	51	STEVE	SWISS	--	*	OTC	---------
WB8QRH	30/1, 77	22: 1	50	SSB	59	MARTY	USA	--	*	------	IC-502
G3QRZ	1/2, 77	19:23	21	SSB	55	BILL	UK	--	*	ISSB	---------
HB7QRP	1/11, 77	16:21	14	CW	589	FRED	SWISS	1000	R	AHC	---------
W6XJ	2/1, 78	13:32	50	SSB	57	CARY	USA	101	*	------	7ELE
VP9L	1/2, 78	15:39	50	CW	519	TOM	RUSIA	1101	R	------	---------
KG6RO	5/3, 78	18:18	3.6	SSB	51	BILL	SAIPAN	999	R	------	CONTEST
ZL2VHF	7/3, 78	18:44	50	CW	599	ROY	NEW ZEA	123	*	------	---------
HH2PR	1/4, 78	14:31	28	SSB	51	BOB	HAITI	321	R	------	---------
I3QSV	5/11, 78	19:37	21	SSB	53	MARV	ROME	--	*	I-YL	---------
JA2APC	21/3, 79	23: 1	50	AM	51	TAKAI	JAPAN	777	R	SHARP	RJX-601
W1BB	1/5, 79	16:39	1.9	CW	599	STEW	BOSTON	1234	R	DXCC	CONTEST
JH3LTX	11/5, 79	5:44	144	FM	59	NAKANISI	JAPAN	2221	R	SHARP	OSAKA
G3QSB/MM	15/7, 79	5: 5	14	SSB	59	HARRY	SWEDEN	2346	R	QCWB	---------
DL1QSO	15/8, 79	1: 5	21	CW	599	GENE	GERMANY	987	R	------	FT301
JA3YHU	10/9, 79	22:22	3.6	CW	419	SHARP	JAPAN	--	*	SHARP	---------
JF3MXU	1/10, 79	17:33	14	CW	519	TED	JAPAN	--	*	SHARP	MZ-80K
JA3DUG	31/12,79	12:59	50	CW	599	NAKA	JAPAN	--	*	JARL	---------
N2ATT	1/1, 80	18:22	14	CW	599	ARAKAWA	USA	345	R	A1-OP	---------
I2QRT	1/2, 80	22:19	7	CW	559	HANK	ITALY	--	*	RCC	---------

◆◆ QSO BAND LIST < 14 MHZ> ◆◆

P - 1

NO	CALL SIGN	DATE	TIME	EM.	RST	QRA	QTH	Q/NO	COMME.	REMARKS
1	K6BX	3/1, 75	5:34	CW	599	CLIF	USA	45	CHC	FT401
2	AA6AA	2/5, 75	15:31	CW	589	JOE	USA	--	------	CONTEST
3	DL5QRK	3/9, 75	15:28	CW	599	KEN	GERMANY	--	------	QRP
4	HB7QRP	1/11, 77	16:21	CW	589	FRED	SWISS	1000	AHC	---------
5	G3QSB/MM	15/7, 79	5: 5	SSB	59	HARRY	SWEDEN	2346	QCWB	---------
6	JF3MXU	1/10, 79	17:33	CW	519	TED	JAPAN	--	SHARP	MZ-80K
7	N2ATT	1/1, 80	18:22	CW	599	ARAKAWA	USA	345	A1-OP	---------

◆◆ PREFIX-BASED LIST <G3> ◆◆

P - 1

CALL SIGN	DATE	TIME	BAND	EM.	RST	QRA	QTH	Q/NO	R	COMME.	REMARKS
G3ORN	3/8, 73	5:14	21	SSB	55	JOHN	LONDON	--	*	ISWL	FT101E
G3QRZ	1/2, 77	19:23	21	SSB	55	BILL	UK	--	*	ISSB	---------
G3QSB/MM	15/7, 79	5: 5	14	SSB	59	HARRY	SWEDEN	2346	R	QCWB	---------

◆◆ PERIOD-BASED LIST ◆◆
< 1/5, 79 - 1/8, 79 >

P - 1

CALL SIGN	DATE	TIME	BAND	EM.	RST	QRA	QTH	Q/NO	R	COMME.	REMARKS
W1BB	1/5, 79	16:39	1.9	CW	599	STEW	BOSTON	1234	R	DXCC	CONTEST
JH3LTX	11/5, 79	5:44	144	FM	59	NAKANISI	JAPAN	2221	R	SHARP	OSAKA
G3QSB/MM	15/7, 79	5: 5	14	SSB	59	HARRY	SWEDEN	2346	R	QCWB	---------

◆◆ QRA-BASED LIST <ARAKAWA > ◆◆

P - 1

NO	CALL SIGN	DATE	TIME	BAND	EM.	RST	QTH	Q/NO	R	COMME.	REMARKS
1	JA3AER	1/12, 60	9: 38	7	AM	59	JAPAN	342	R	SHARP	6JS6
2	N2ATT	1/1, 80	18: 22	14	CW	599	USA	345	R	A1-OP	---------

◆◆ COMMENT-BASED LIST <SHARP > ◆◆

P - 1

NO	CALL SIGN	DATE	TIME	BAND	EM.	RST	QRA	QTH	Q/NO	R	REMARKS
1	JA3AER	1/12, 60	9:38	7	AM	59	ARAKAWA	JAPAN	342	R	6JS6
2	JA2APC	21/3, 79	23: 1	50	AM	51	TAKAI	JAPAN	777	R	RJX-601
3	JH3LTX	11/5, 79	5:44	144	FM	59	NAKANISI	JAPAN	2221	R	OSAKA
4	JA3YHU	10/9, 79	22:22	3.6	CW	419	SHARP	JAPAN	--	*	---------
5	JF3MXU	1/10, 79	17:33	14	CW	519	TED	JAPAN	--	*	MZ-80K

◆◆ QSO DATA FOR JH3LTX STATION ◆◆

P - 1

NO	DATE	TIME	BAND	EM.	RST	QRA	QTH	Q/NO	R	COMME.	REMARKS
1	11/5, 79	5: 44	144	FM	59	NAKANISI	JAPAN	2221	R	SHARP	OSAKA

Program description

Now we will describe this sample program. Why don't you modify this program, make more practical variations or apply it to information control in other fields?

1) Data configuration

Data entered through the keyboard – call sign, and the other 12 items including QSO date – are divided based on ASCII code into four blocks, each block being composed of 16 bytes.

Block 1 Data A$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
*	G	6	A	B	C	SP	SP	SP	SP	SP	9	S	S	B	R

1st byte (I1$): Reserve (filled with "*" in this program.)
2nd to 11th bytes (I1$): Call sign
12th byte (I4$): Band data (Number of band you want)
(0 thru 9; 10 to 14 correspond to A to E.)
13th to 15th bytes (15$): Transmitting mode
16th byte (I9$): Data indicating whether or not QSL card is received.

Block 2 Data B$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5	9	+	1	9	5	2	O	U	T		1	0	0	W	

1st to 3rd bytes (I6$): RST data
4th to 7th bytes (I3$): Data of QSO start time
8th to 16th bytes (J3$): Remarks

Block Data C$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
J	O	N	E	S	SP	SP	SP	L	O	N	D	O	N	SP	SP

1st to 8th bytes (I7$): QRA data
9th to 16th bytes (I8$): CITY or COUNTRY

Block Data C$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
7	9	1	2	1	0	1	2	3	4	R	S	G	B	SP	SP

1st to 6th bytes (I2$): QSO data
7th to 10th bytes (J1$): QSL No.
11th to 16th bytes (J2$): Data of classification comment

2) File architecture

The first page of a file is filled with page no. that identifies the head location of the final data block. The illustration at right reveals that data for 20 stations is stored in file and the final data–the data of the 20th station– is written over four pages, 78th to 81st pages. The data of every station is stored on the 2nd and subsequent pages in order of date and time with four pages as one block.

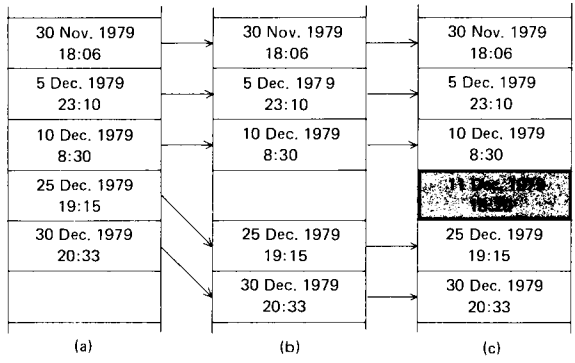
Rearrangement of the order of date and time for data is made if a change occurs in date and time when inputting or correcting data.

1	78
2	A$
3	B$
4	C$
5	D$
6	A$
7	B$
8	C$
78	A$
79	B$
80	C$
81	D$

Final data in file

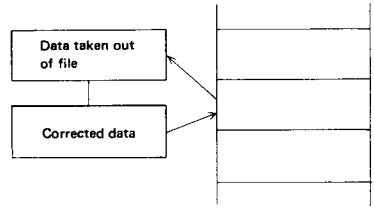
3) Rearrangement of data

Suppose that the final data on the file be of 20:30 on 30 Sec. 1979 as shown in illustration (a) below. If the data of 18:20 on 11 Dec. 1979 is input anew, the last two items dated later than the newly input one are displaced downward by one block (see illustration (b)). Then the new data is placed in the resulting empty block (see illustration (c)).

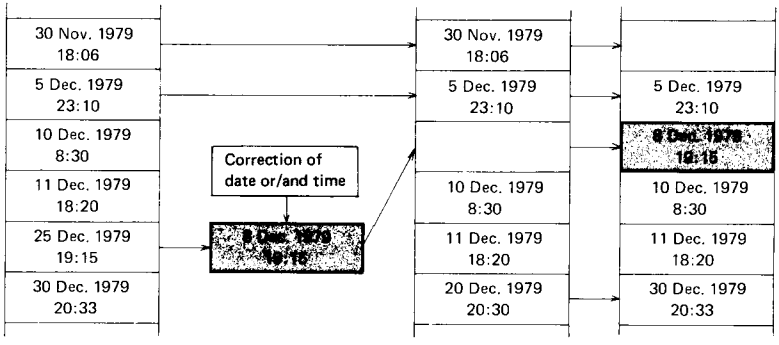


4) Correction of data

(1) Correction of data other than those date or/and time causes no positional change in data block as illustrated at right.



(2) If certain data – the data of 19:15 on 25 Dec. 1979 in this example – is corrected in date or/and time – date is changed from 25 Dec. 1979 to 8 Dec. 1979 in this example – all data having a later date than the new date –8 Dec. 1979 – go down by one block as illustrated below. Then, the corrected date is placed in the resulting empty block.



The rearrangement of dates is made by comparing them with each other with 1 minute as minimum unit. Meanwhile, the rearrangement and correction of dates are made based on random access (XOPEN).

5) Major variables used in the program

Arrays

A$ (60)	Data buffer for 1st block	These buffers temporarily retain data road from the file.
	Data buffer for 2nd block	
	Data buffer for 3rd block	
	Data buffer for 4th block	
F (60)	Retains pages each block of data occupies on the file.	
F$ (14)	QSO band table	

Character variables

A$	1st block Data		W$	1st comparison block Data	
B$	2nd block Data		X$	2nd comparison block Data	
C$	3rd block Data		Y$	3rd comparison block Data	
D$	4th block Data		Z$	4th comparison block Data	
I1$	Call sign	Input data	A1$	Call sign	Output data
I2$	QSO date	Input data	D1$	QSO data	Output data
I3$	QSO start date	Input data	B1$	QSO start date	Output data
I4$	QSO band	Input data	A2$	QSO band	Output data
I5$	Mode	Input data	A2$	Mode	Output data
I6$	RST	Input data	B1$	RST	Output data
I7$	QRA	Input data	C1$	QRA	Output data
I8$	QTH	Input data	C2$	QTH	Output data
I9$	QSL card reception	Input data	A4$	QSL card reception	Output data
J1$	QSL card No.	Input data	D2$	QSL card No.	Output data
J2$	Classification comment	Input data	D3$	Classification comment	Output data
J3$	Remarks	Input data	B3$	Remarks	Output data

Numeric variables

A	INDEX command
I	File page counter
S	File page counter
T	File page counter
M	Block counter for read data
V	Final page of data block on file
F1	Correction flags for data and time
Z1	File data correction flag
U8	To-printer output flag
U9	Counter of the number of stations displayed on CRT screen
B4	Flag for decision of presence or absence of following pages
B5	Flag for decision of presence or absence of objective data
P1	Page counter in printer output mode
X	Counter for data block serial number in printer output mode

—Amateur radio terminology—

Call sign A call sign is given to every amateur radio station. There is no other station with the same call sign in the world. The call sign tells us of the nationality and sometimes the region its owner belongs to.

Ex. G3ABC

Nation

The characters G3 are called the "prefix" of the call sign.

QSO Radio communication (Q sign, not abbreviation)

Ex. QSO date denotes the date when radio communication was made.

QRA Name of each amateur radio station (Q sign)

Ex. G3BRS---QRA = Bury Radio Society

If the license holder is an individual, the QRA indicates the operator's name.

Ex. G4ZZZ--QRA = Mr. ABS

QSY To change transmitting frequency or transmission mode (for example, telegraph, telephone, television, teletype, etc.)

QTH Position of each radio station, that is, address or location

QSL card A kind of greeting card the radio station that has communicated with a certain station sends there as a proof of communication.
QSL number refers to the issuance number of the QSL.

RST Every station, when having communication with some station, must report to it its signal strength or the degree of receiving sensitivity. The RST, the criterion of the report, is evaluated by numeric values.

R = Receiving sensitivity	1-5	The evaluation is made according to a certain criterion.
S = Signal strength	1-9	
T = Tone	1-9	T is necessary only for telegraphy (Morse Code).

Ex. RST = 599 is the best in the case of telegraph communication.

Log Radio log, a notebook to record communication data in. This is one of the operation records that must be maintained by in every amateur radio station.

Shack Every amateur radio operator calls the room equipped with his own rig his "shack" with friendly sentiments.

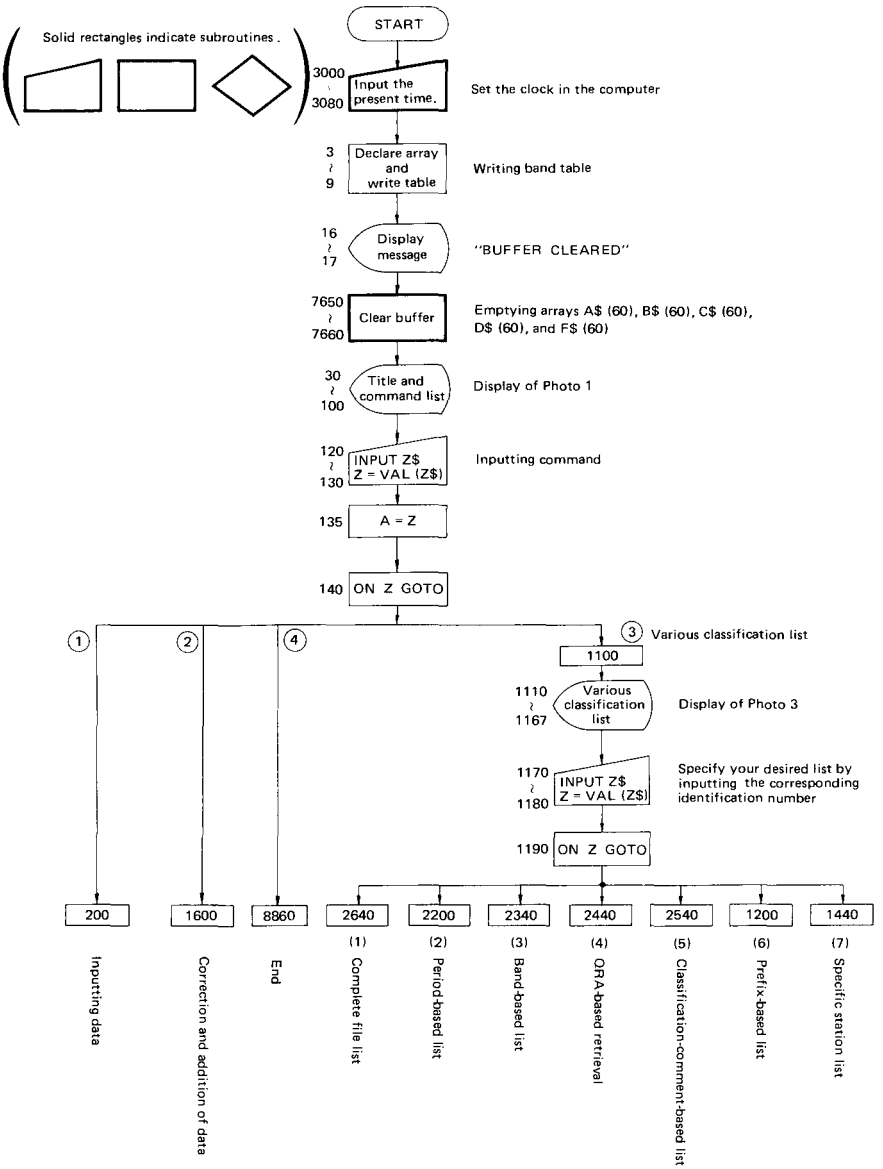
Band Many frequency bands are allowed for amateur radio communication. Accordingly, every operator must record the frequency bands he has used to communicate with other station. In the U.K. the following frequency bands are available for amateur radio communication.

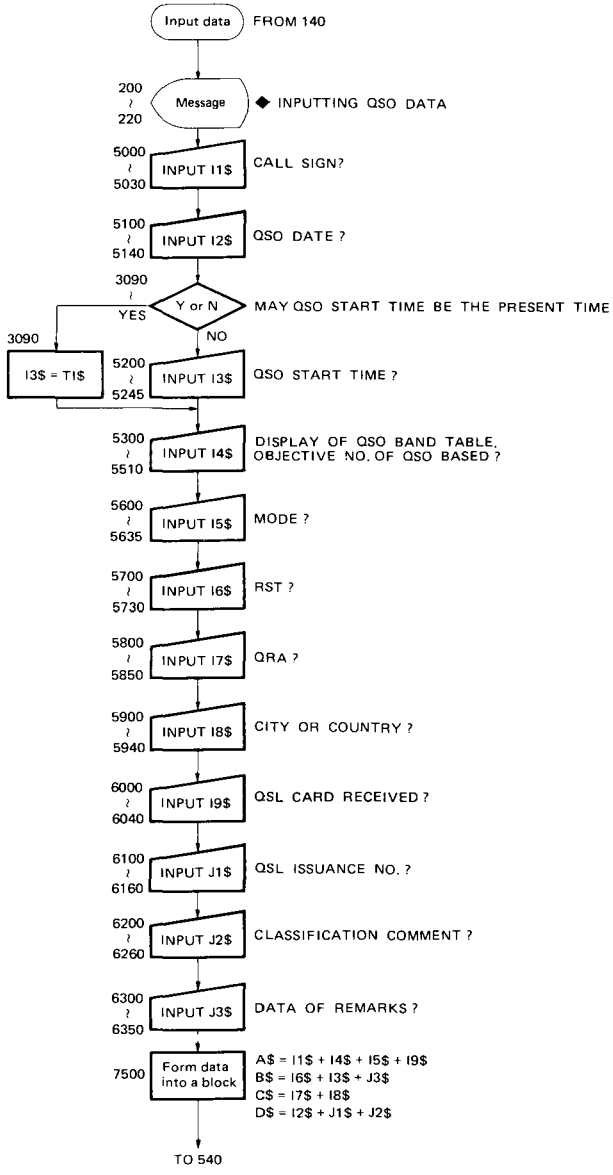
1.9, 3.6, 7, 14, 21, 28, 70, 144, 432, and 1200 MHz (Bands of over 1200 MHz are omitted.)

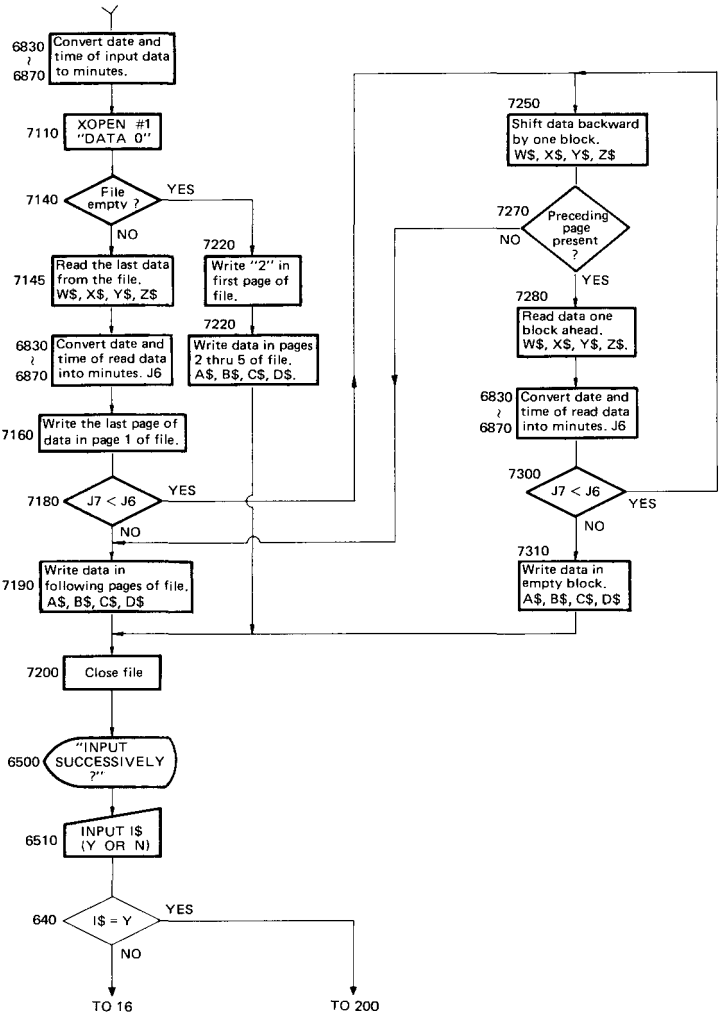
Also, the frequency band is in some cases represented by wavelength.

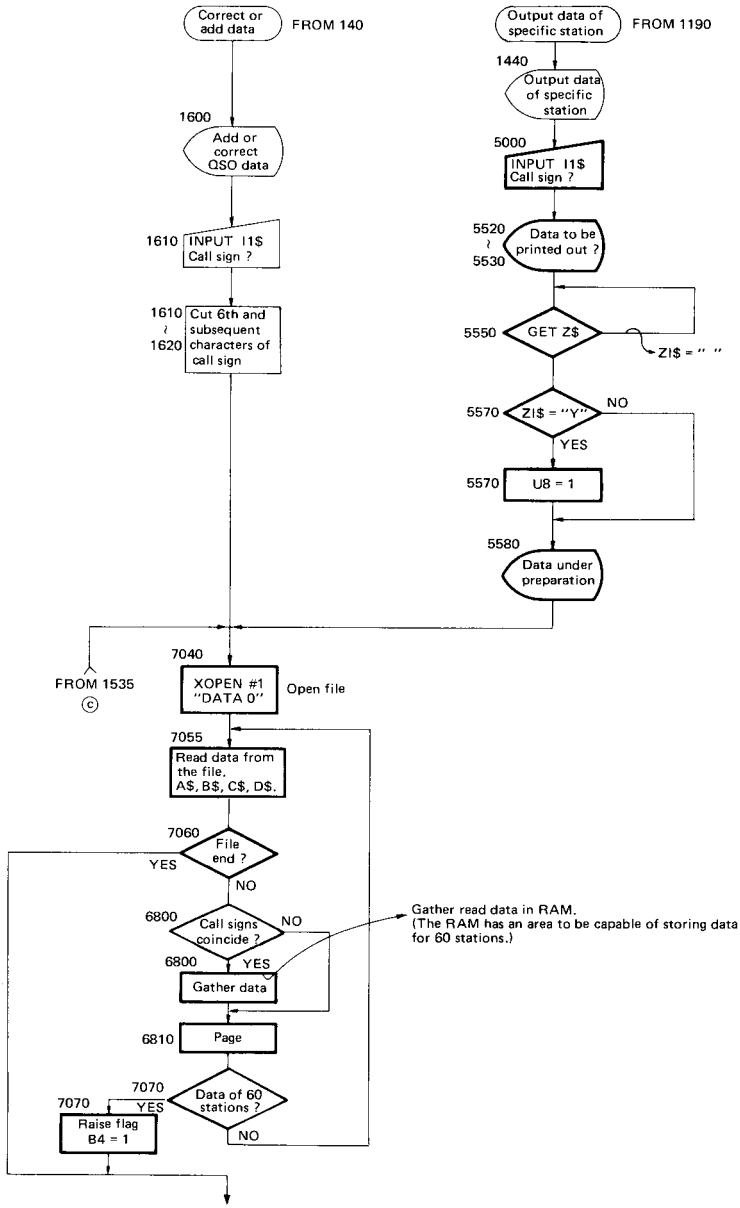
Ex. 4m band = 70 MHz band

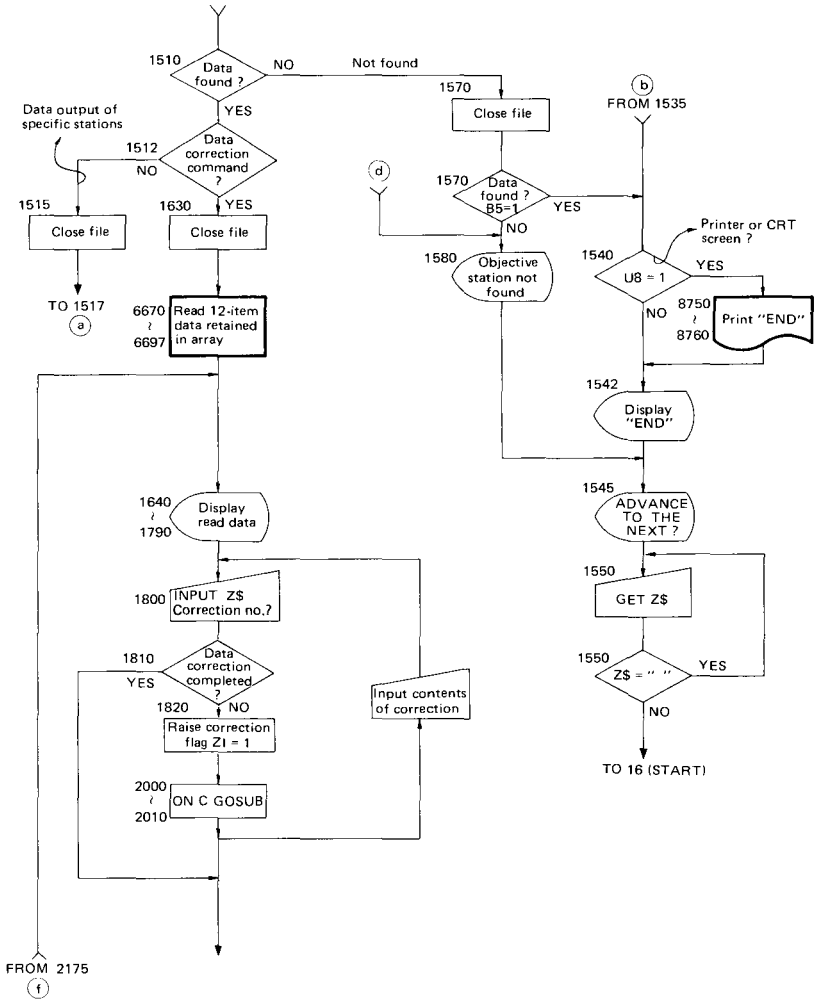
Log program flowchart

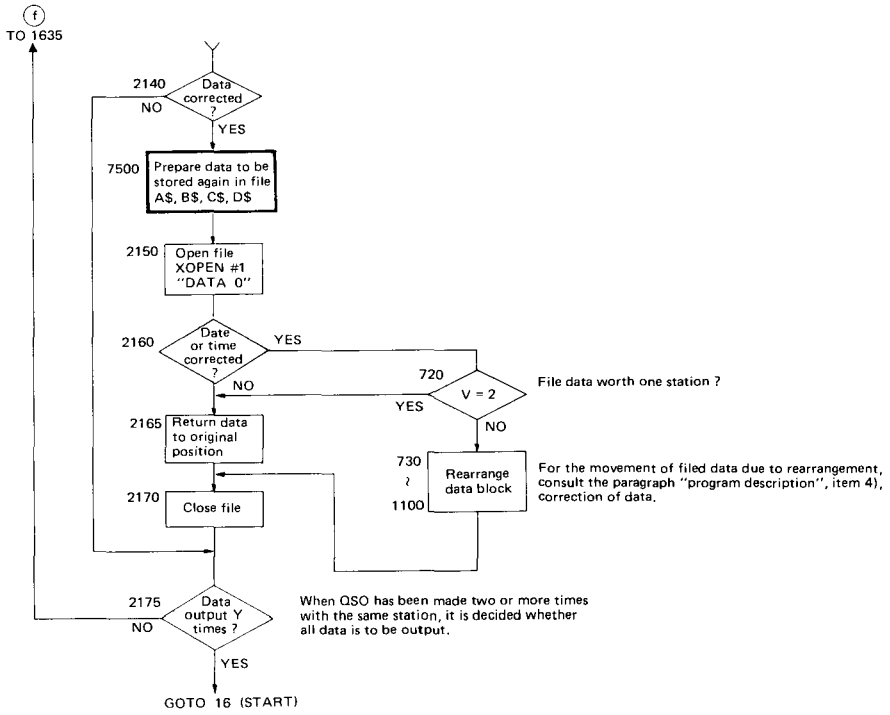


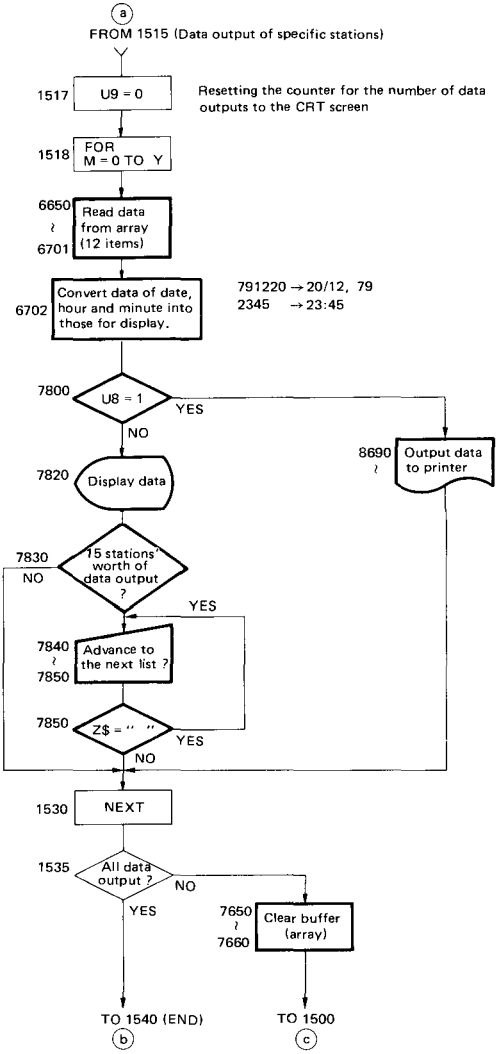

Input data
FROM 140
200
~
220
Message
◆ INPUTTING QSO DATA
5000
~
5030
INPUT I1$
CALL SIGN?
5100
~
5140
INPUT I2$
QSO DATE ?
3090
~
Y or N
MAY QSO START TIME BE THE PRESENT TIME ?
YES
NO
3090
I3$ = T1$
5200
~
5245
INPUT I3$
QSO START TIME ?
5300
~
5510
INPUT I4$
DISPLAY OF QSO BAND TABLE,
OBJECTIVE NO. OF QSO BASED ?
5600
~
5635
INPUT I5$
MODE ?
5700
~
5730
INPUT I6$
RST ?
5800
~
5850
INPUT I7$
QRA ?
5900
~
5940
INPUT I8$
CITY OR COUNTRY ?
6000
~
6040
INPUT I9$
QSL CARD RECEIVED ?
6100
~
6160
INPUT J1$
QSL ISSUANCE NO. ?
6200
~
6260
INPUT J2$
CLASSIFICATION COMMENT ?
6300
~
6350
INPUT J3$
DATA OF REMARKS ?
7500
Form data into a block
A$ = I1$ + I4$ + I5$ + I9$
B$ = I6$ + I3$ + J3$
C$ = I7$ + I8$
D$ = I2$ + J1$ + J2$
TO 540

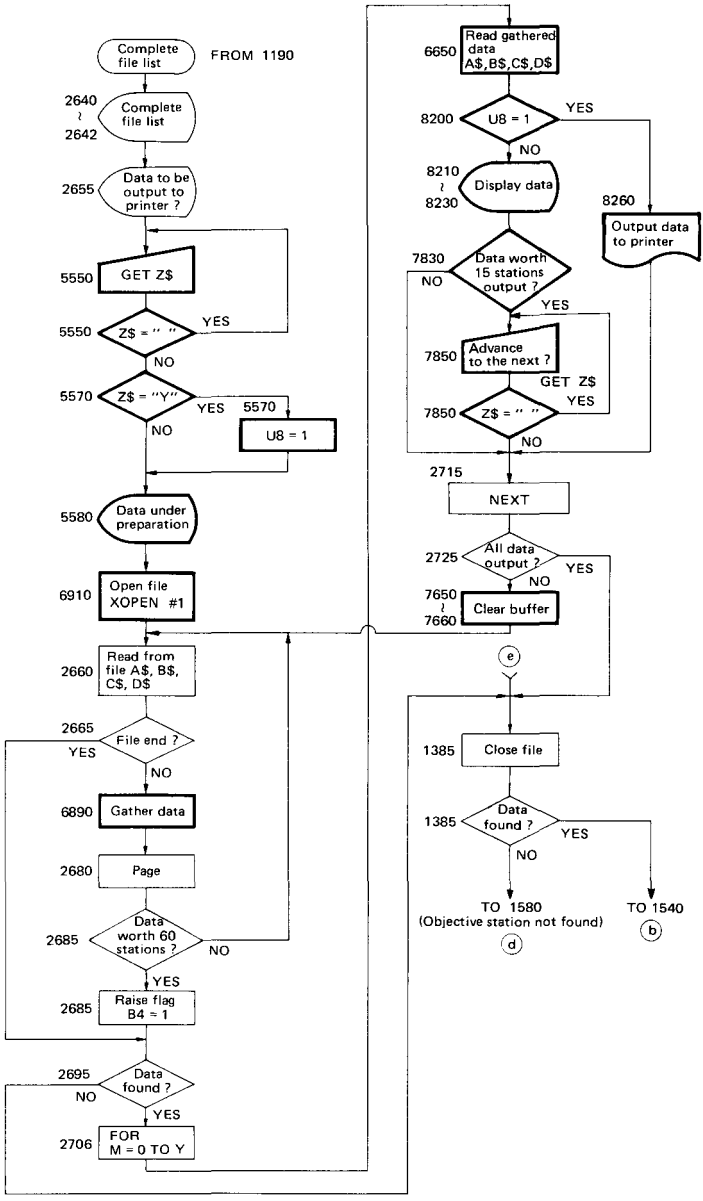

6830 ~ 6870
Convert date and time of input data to minutes.
7110
XOPEN #1 "DATA 0"
7140
File empty ?
YES
NO
7145
Read the last data from the file. W$, X$, Y$, Z$
6830 ~ 6870
Convert date and time of read data into minutes. J6
7160
Write the last page of data in page 1 of file.
7180
J7 < J6
YES
NO
7190
Write data in following pages of file. A$, B$, C$, D$
7220
Write "2" in first page of file.
7220
Write data in pages 2 thru 5 of file. A$, B$, C$, D$.
7200
Close file
6500
"INPUT SUCCESSIVELY ?"
6510
INPUT I$ (Y OR N)
640
I$ = Y
YES
NO
TO 16
TO 200
7250
Shift data backward by one block. W$, X$, Y$, Z$
7270
Preceding page present ?
NO
YES
7280
Read data one block ahead. W$, X$, Y$, Z$.
6830 ~ 6870
Convert date and time of read data into minutes. J6
7300
J7 < J6
YES
NO
7310
Write data in empty block. A$, B$, C$, D$

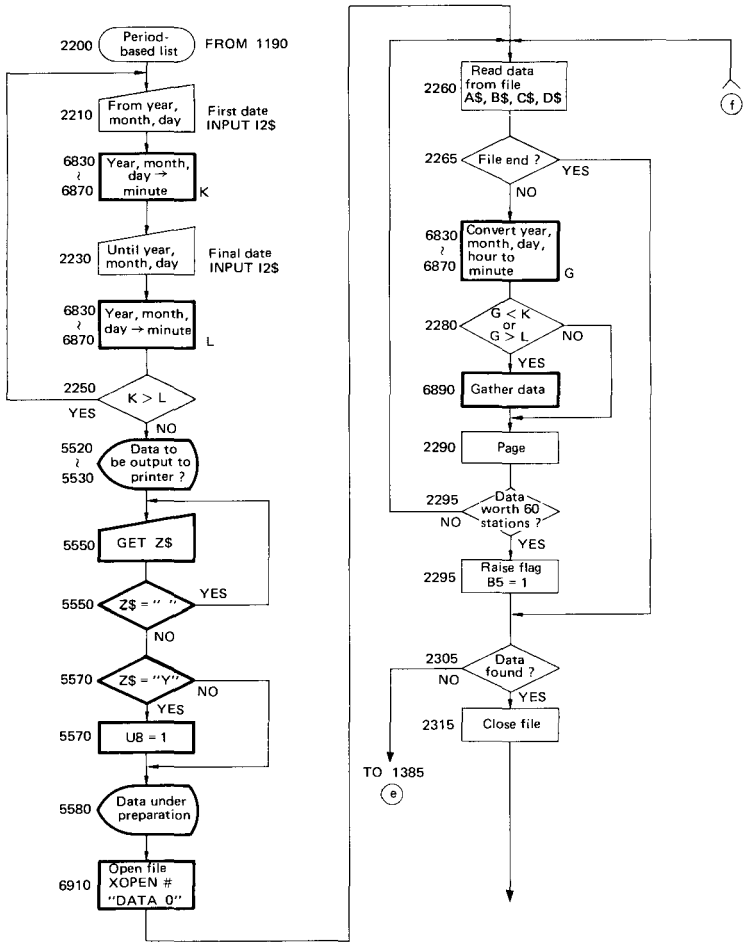

Correct or add data
FROM 140
1600
Add or correct QSO data
1610
INPUT I1$ Call sign ?
1610 ~ 1620
Cut 6th and subsequent characters of call sign
Output data of specific station
FROM 1190
1440
Output data of specific station
5000
INPUT I1$ Call sign ?
5520 ~ 5530
Data to be printed out ?
5550
GET Z$
Z1$ = " "
5570
Z1$ = "Y"
NO
YES
5570
U8 = 1
5580
Data under preparation
FROM 1535
ⓒ
7040
XOPEN #1 "DATA 0"
Open file
7055
Read data from the file, A$, B$, C$, D$.
7060
File end ?
YES
NO
6800
Call signs coincide ?
NO
6800
YES
Gather data
Gather read data in RAM. (The RAM has an area to be capable of storing data for 60 stations.)
6810
Page
7070
Data of 60 stations ?
7070
YES
Raise flag B4 = 1
NO

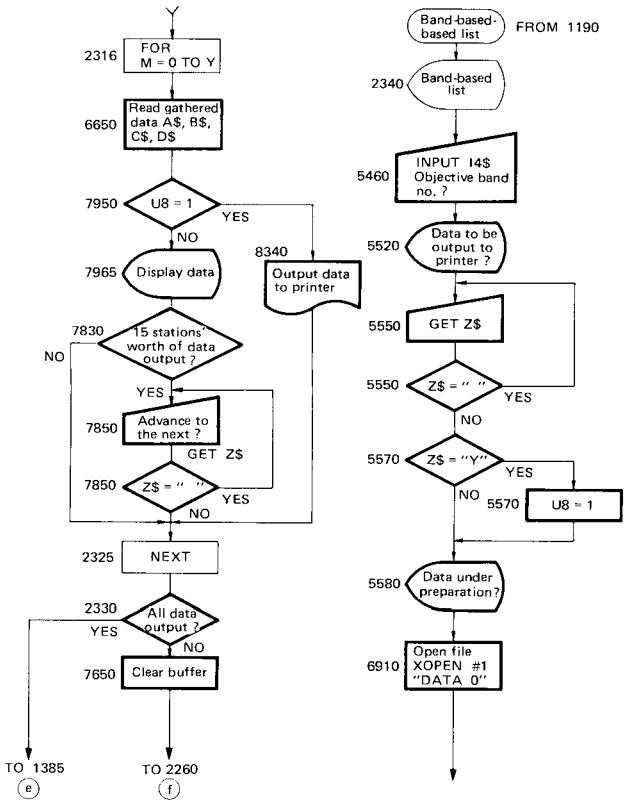

1510
Data found ?
NO
Not found
YES
1570
Close file
b
FROM 1535
Data output of specific stations
1512
Data correction command ?
NO
YES
d
1570
Data found ? B5=1
YES
NO
1515
Close file
1630
Close file
1580
Objective station not found
1540
U8 = 1
YES
Printer or CRT screen ?
NO
8750 ≀ 8760
Print "END"
TO 1517
a
6670 ≀ 6697
Read 12-item data retained in array
1542
Display "END"
1640 ≀ 1790
Display read data
1545
ADVANCE TO THE NEXT ?
1800
INPUT Z$ Correction no.?
1550
GET Z$
1810
Data correction completed ?
YES
1820
NO
Input contents of correction
1550
Z$ = " "
YES
NO
Raise correction flag ZI = 1
2000 ≀ 2010
ON C GOSUB
TO 16 (START)
FROM 2175
f

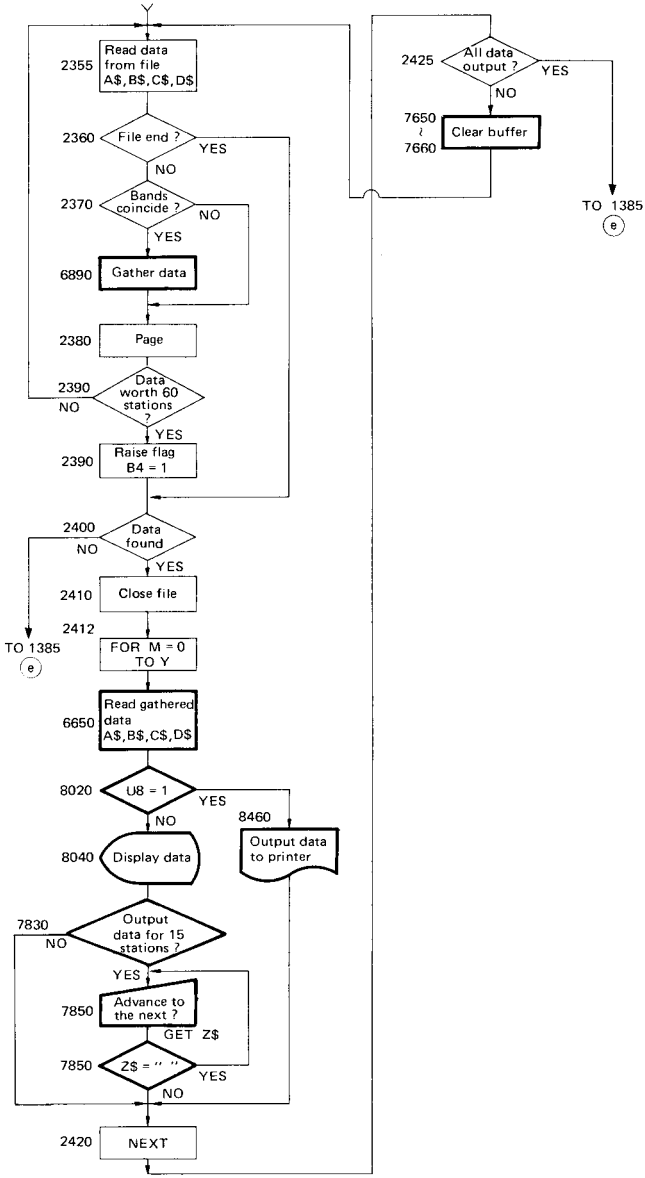

f
TO 1635
2140
Data corrected ?
NO
YES
7500
Prepare data to be stored again in file A$, B$, C$, D$
2150
Open file XOPEN #1 "DATA 0"
2160
Date or time corrected ?
YES
NO
720
V = 2
YES
NO
File data worth one station ?
2165
Return data to original position
730
?
1100
Rearrange data block
For the movement of filed data due to rearrangement, consult the paragraph "program description", item 4), correction of data.
2170
Close file
2175
Data output Y times ?
NO
YES
When QSO has been made two or more times with the same station, it is decided whether all data is to be output.
GOTO 16 (START)

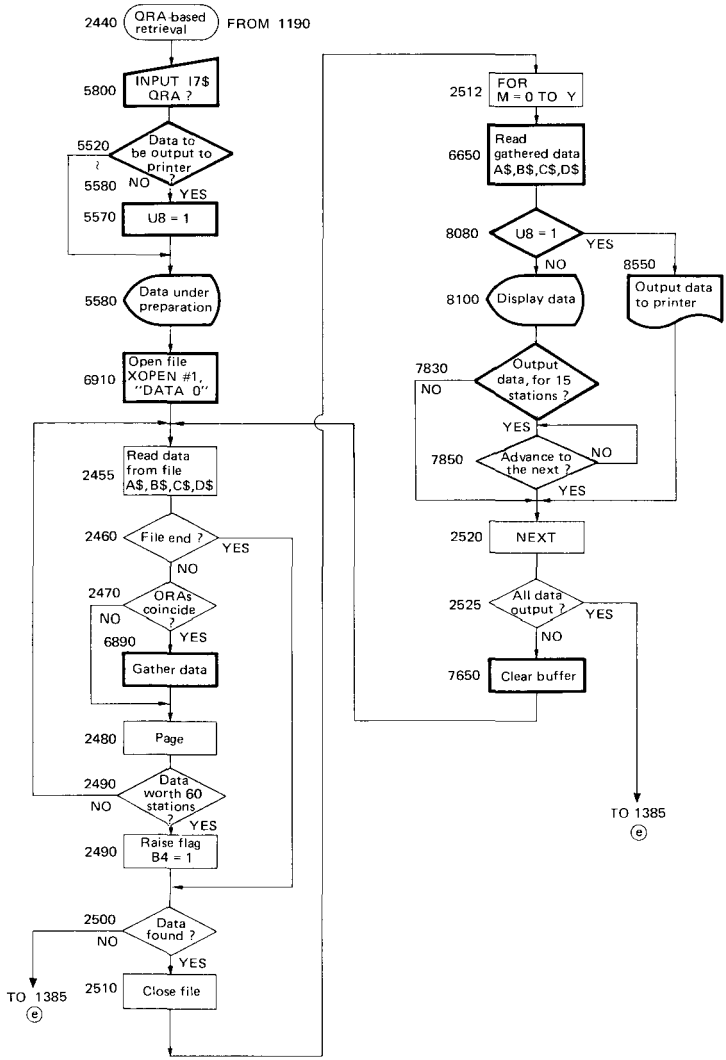

a
FROM 1515 (Data output of specific stations)
1517
U9 = 0
Resetting the counter for the number of data outputs to the CRT screen
1518
FOR M = 0 TO Y
6650 ~ 6701
Read data from array (12 items)
6702
Convert data of date, hour and minute into those for display.
791220 → 20/12, 79
2345 → 23:45
7800
U8 = 1
YES
NO
8690 ~
Output data to printer
7820
Display data
7830
15 stations' worth of data output ?
NO
YES
7840 ~ 7850
Advance to the next list ?
7850
Z$ = " "
YES
NO
1530
NEXT
1535
All data output ?
NO
YES
7650 ~ 7660
Clear buffer (array)
TO 1540 (END)
b
TO 1500
c

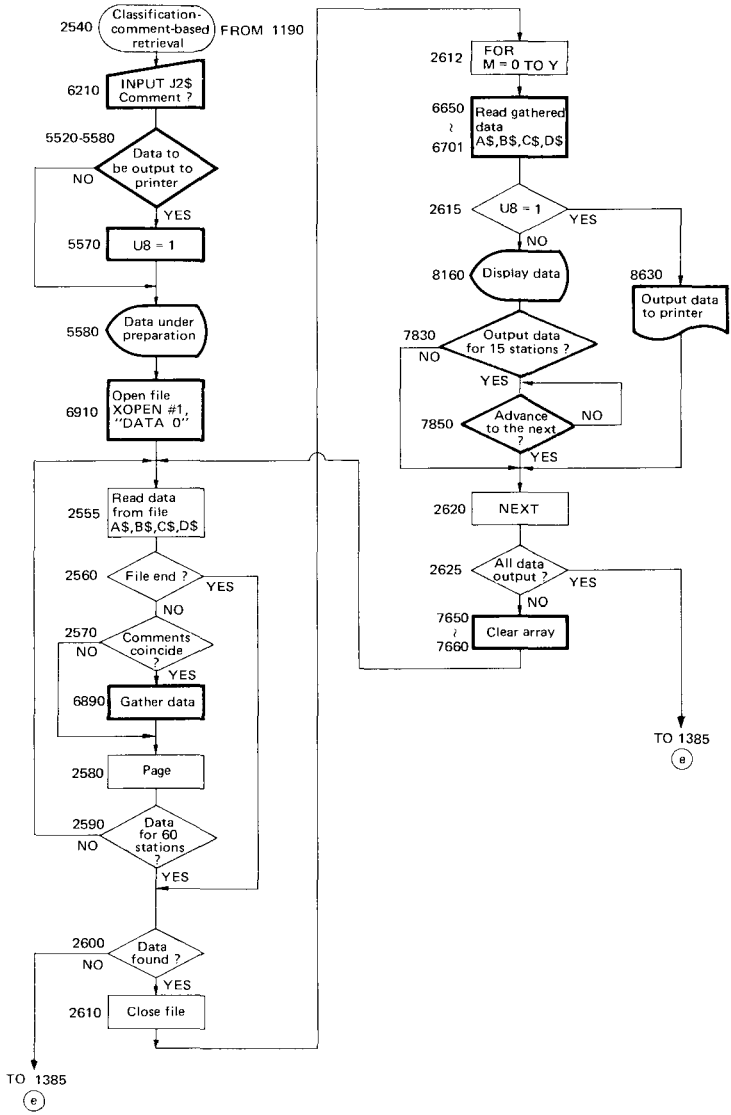

Complete file list
FROM 1190
2640 ~ 2642
Complete file list
2655
Data to be output to printer ?
5550
GET Z$
5550
Z$ = " "
YES
NO
5570
Z$ = "Y"
YES
5570
U8 = 1
NO
5580
Data under preparation
6910
Open file XOPEN #1
2660
Read from file A$, B$, C$, D$
2665
File end ?
YES
NO
6890
Gather data
2680
Page
2685
Data worth 60 stations ?
NO
YES
2685
Raise flag B4 = 1
2695
Data found ?
NO
YES
2706
FOR M = 0 TO Y
6650
Read gathered data A$,B$,C$,D$
8200
U8 = 1
YES
NO
8210 ~ 8230
Display data
8260
Output data to printer
7830
Data worth 15 stations output ?
NO
YES
7850
Advance to the next ?
GET Z$
7850
Z$ = " "
YES
NO
2715
NEXT
2725
All data output ?
YES
NO
7650 ~ 7660
Clear buffer
e
1385
Close file
1385
Data found ?
YES
NO
TO 1580
(Objective station not found)
d
TO 1540
b

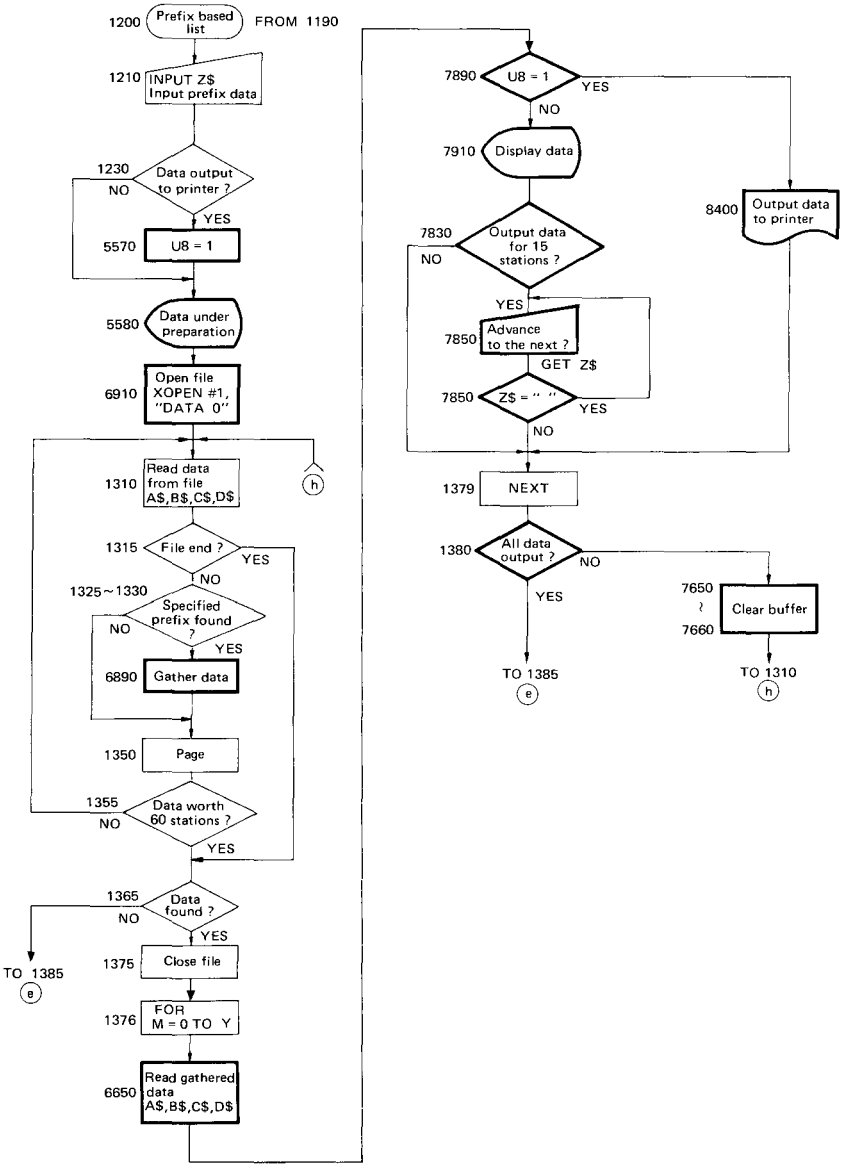

2200
Period-based list
FROM 1190
2210
From year, month, day
First date INPUT I2$
6830 ≀ 6870
Year, month, day → minute
K
2230
Until year, month, day
Final date INPUT I2$
6830 ≀ 6870
Year, month, day → minute
L
2250
K > L
YES
NO
5520 ≀ 5530
Data to be output to printer ?
5550
GET Z$
5550
Z$ = " "
YES
NO
5570
Z$ = "Y"
NO
YES
5570
U8 = 1
5580
Data under preparation
6910
Open file XOPEN # "DATA_0"
2260
Read data from file A$, B$, C$, D$
2265
File end ?
YES
NO
6830 ≀ 6870
Convert year, month, day, hour to minute
G
2280
G < K or G > L
NO
YES
6890
Gather data
2290
Page
2295
Data worth 60 stations ?
NO
YES
2295
Raise flag B5 = 1
2305
Data found ?
NO
YES
2315
Close file
TO 1385
e
f


Y
2316
FOR M = 0 TO Y
6650
Read gathered data A$, B$, C$, D$
7950
U8 = 1
YES
NO
7965
Display data
8340
Output data to printer
7830
15 stations' worth of data output ?
NO
YES
7850
Advance to the next ?
GET Z$
7850
Z$ = " "
YES
NO
2325
NEXT
2330
All data output ?
YES
NO
7650
Clear buffer
TO 1385
e
TO 2260
f
Band-based-based list
FROM 1190
2340
Band-based list
5460
INPUT I4$ Objective band no. ?
5520
Data to be output to printer ?
5550
GET Z$
5550
Z$ = " "
YES
NO
5570
Z$ = "Y"
YES
NO
5570
U8 = 1
5580
Data under preparation?
6910
Open file XOPEN #1 "DATA 0"


2355
Read data from file A$,B$,C$,D$
2360
File end ?
YES
NO
2370
Bands coincide ?
NO
YES
6890
Gather data
2380
Page
2390
Data worth 60 stations ?
NO
YES
2390
Raise flag B4 = 1
2400
Data found
NO
YES
2410
Close file
2412
FOR M = 0 TO Y
TO 1385
e
6650
Read gathered data A$,B$,C$,D$
8020
U8 = 1
YES
NO
8460
Output data to printer
8040
Display data
7830
Output data for 15 stations ?
NO
YES
7850
Advance to the next ?
GET Z$
7850
Z$ = " "
YES
NO
2420
NEXT
2425
All data output ?
YES
NO
7650
7660
Clear buffer
TO 1385
e


2440
QRA-based retrieval
FROM 1190
5800
INPUT I7$ QRA ?
5520
Data to be output to printer ?
NO
YES
5580
5570
U8 = 1
5580
Data under preparation
6910
Open file XOPEN #1, "DATA 0"
2455
Read data from file A$,B$,C$,D$
2460
File end ?
YES
NO
2470
ORAs coincide ?
NO
YES
6890
Gather data
2480
Page
2490
Data worth 60 stations ?
NO
YES
2490
Raise flag B4 = 1
2500
Data found ?
NO
YES
TO 1385
e
2510
Close file
2512
FOR M = 0 TO Y
6650
Read gathered data A$,B$,C$,D$
8080
U8 = 1
YES
NO
8550
Output data to printer
8100
Display data
7830
Output data, for 15 stations ?
NO
YES
7850
Advance to the next ?
NO
YES
2520
NEXT
2525
All data output ?
YES
NO
7650
Clear buffer
TO 1385
e


2540
Classification-comment-based retrieval
FROM 1190
6210
INPUT J2$ Comment ?
5520-5580
Data to be output to printer
NO
YES
5570
U8 = 1
5580
Data under preparation
6910
Open file XOPEN #1, "DATA 0"
2555
Read data from file A$,B$,C$,D$
2560
File end ?
YES
NO
2570
Comments coincide ?
NO
YES
6890
Gather data
2580
Page
2590
Data for 60 stations ?
NO
YES
2600
Data found ?
NO
YES
2610
Close file
TO 1385
e
2612
FOR M = 0 TO Y
6650 ≀ 6701
Read gathered data A$,B$,C$,D$
2615
U8 = 1
YES
NO
8160
Display data
8630
Output data to printer
7830
Output data for 15 stations ?
NO
YES
7850
Advance to the next ?
NO
YES
2620
NEXT
2625
All data output ?
YES
NO
7650 ≀ 7660
Clear array
TO 1385
e


1200 Prefix based list
FROM 1190
1210 INPUT Z$ Input prefix data
1230 Data output to printer ?
NO
YES
5570 U8 = 1
5580 Data under preparation
6910 Open file XOPEN #1, "DATA 0"
1310 Read data from file A$,B$,C$,D$
h
1315 File end ?
YES
NO
1325~1330 Specified prefix found ?
NO
YES
6890 Gather data
1350 Page
1355 Data worth 60 stations ?
NO
YES
1365 Data found ?
NO
YES
TO 1385
e
1375 Close file
1376 FOR M = 0 TO Y
6650 Read gathered data A$,B$,C$,D$
7890 U8 = 1
YES
NO
7910 Display data
8400 Output data to printer
7830 Output data for 15 stations ?
NO
YES
7850 Advance to the next ?
GET Z$
7850 Z$ = " "
YES
NO
1379 NEXT
1380 All data output ?
NO
YES
TO 1385
e
7650 ~ 7660 Clear buffer
TO 1310
h

```
1 REM ***  AMATEUR RADIO LOG
3 TEMPO 7:M$="C3EGC‾":M1$="B2‾":N$="DATA 0":GOSUB 3000
5 DIM A$(60),F(60),E$(14)
6 PRINT"■":E$(0)="----":E$(1)=" 1.9":E$(2)=" 3.6":E$(3)=" 3.8":E$(4)="  7 "
8 E$(5)="  14":E$(6)="  21":E$(7)="  28":E$(8)="  50":E$(9)=" 144"
9 E$(10)=" 432":E$(11)="1200":E$(12)="2300":E$(13)="5700":E$(14)="ETC."
16 PRINT"■":MUSIC M1$:CURSOR 3,9
17 PRINT"BUFFER CLEARED NOW■■■■■■■WAIT A MINUTE":GOSUB 7650
30 PRINT"■":MUSIC M1$:CURSOR 7,3:PRINT"| ■■──────────────────────────┐ ■■ "
35 PRINT TAB(7);"└──────────────────────────┘ "
40 PRINT"■■";TAB(9);"AMATUER RADIO LOG":PRINT"■■":PRINT"  ♦ INDEX ♦"
70 PRINT"■        1.  INPUT OF QSO DATA"
80 PRINT"■        2.  ADDITION OR CORRECTION"
100 PRINT"■        3.  VARIOUS LISTS"
110 PRINT"■        4.  END"
120 INPUT"■■ SET YOUR REQUEST AT NUMBER : ";Z$:Z=VAL(Z$)
130 IF (Z<1)+(Z>4) THEN PRINT"■■■■":MUSIC M$:GOTO 120
135 MUSIC M1$:A=Z
140 ON Z GOTO 200,1600,1110,8860
200 PRINT"■  ♦♦  INPUT OF QRS DATA  ♦♦"
210 PRINT"■■  * PRESS 'N' WITH NO DATA"
220 PRINT"  * PRESS 'B' WHEN CANCELING■■"
230 GOSUB 5000:PRINT:IF I1$="B" THEN 30
250 GOSUB 5100:PRINT:IF I2$="B" THEN 30
270 GOSUB 3090:PRINT:IF I3$="B" THEN 30
280 GOSUB 5300:PRINT:IF BK=1 THEN 30
290 GOSUB 5600:PRINT:IF I5$="B" THEN 30
310 GOSUB 5700:PRINT:IF I6$="B" THEN 30
330 GOSUB 5800:PRINT:IF I7$="B" THEN 30
350 GOSUB 5900:PRINT:IF I8$="B" THEN 30
370 GOSUB 6000:PRINT:IF I9$="B" THEN 30
390 GOSUB 6100:PRINT:IF J1$="B" THEN 30
410 GOSUB 6200:PRINT:IF J2$="B" THEN 30
430 GOSUB 6300:PRINT:IF J3$="B" THEN 30
450 REM *** PREPARATION OF INPUT DATA
460 GOSUB 7500:I=2
555 REM *** INPUT DATA TO FILE
557 GOSUB 7100:GOSUB 6500
640 IF I$="Y" GOTO 200
650 GOTO 16
690 REM *** DATE & TIME SETTING
720 IF V=2 THEN 2165
730 E$=B$:F$=D$:GOSUB 6830
740 J7=J6
750 IF S=2 THEN S=S+4:GOTO 910
760 S=S-4
770 GOSUB 1090
790 IF J6=J7 THEN 1060
800 IF J6>J7 THEN 1000
810 S=S+4
820 IF S=V THEN 1070
830 T=T+4
840 INPUT #1(T),W$,X$,Y$,Z$
850 E$=X$:F$=Z$:GOSUB 6830
860 IF J6>=J7 THEN 1070
870 PRINT #1(S),W$,X$,Y$,Z$
890 IF T=V THEN 1060
900 S=S+4:T=T+4:GOTO 840
910 GOSUB 1090
930 IF J6>=J7 THEN 1060
940 PRINT #1(T),W$,X$,Y$,Z$
950 IF S=V THEN 1070
960 T=T+4:S=S+4
970 GOSUB 1090
990 GOTO 930
1000 PRINT #1(T),W$,X$,Y$,Z$
```

```
1010 IF S=2 THEN 1070
1020 S=S-4:T=T-4:GOSUB 1090
1050 IF J6>J7 GOTO 1000
1060 PRINT #1(T),A$,B$,C$,D$:GOTO 1080
1070 PRINT #1(S),A$,B$,C$,D$
1080 F1=0:Z1=0:GOTO 2170
1090 INPUT #1(S),W$,X$,Y$,Z$
1100 E$=X$:F$=Z$:GOSUB 6830:RETURN
1110 PRINT"■■■   ♦ VARIOUS LISTS■■■"
1120 PRINT"      1. COMPLETE FILE LIST■"
1130 PRINT"      2. PERIOD-BASED LIST■"
1140 PRINT"      3. BAND-BASED LIST■"
1150 PRINT"      4. ORA-BASED LIST■"
1160 PRINT"      5. COMMENT-BASED LIST■"
1165 PRINT"      6. PREFIX-BASED LIST■"
1167 PRINT"      7. SPECIFIC STATION LIST■■"
1170 INPUT" INPUT YOUR REQUEST AT NUMBER : ";Z$:Z=VAL(Z$)
1180 IF (Z<1)+(Z>7) THEN PRINT"■■":MUSIC M$:GOTO 1170
1185 MUSIC M1$:U=0
1190 ON Z GOTO 2640,2200,2340,2440,2540,1200,1440
1195 GOTO 1110
1200 PRINT "■":CURSOR 2,2:PRINT" ♦♦  PREFIX-BASED LIST  ♦♦ ■■■■"
1210 INPUT"■INPUT PREFIX DATA : ";Z$:Z3=LEN(Z$)
1220 IF Z3>3 THEN MUSIC M$:PRINT"■■":GOTO 1210
1230 GOSUB 5520
1300 GOSUB 6900
1310 INPUT #1(I),A$,B$,C$,D$
1315 IF EOF(#1) THEN 1365
1325 GOSUB 7350
1330 IF Z$=MID$(A$,2,Z3) GOSUB 6890
1350 I=I+4
1355 IF M=60 THEN B4=1:GOTO 1365
1360 GOTO 1310
1365 Y=M-1:IF M=0 THEN 1385
1375 U9=0:CLOSE
1376 FOR M=0 TO Y
1377 GOSUB 6650:IF U8=1 GOSUB 6721:GOTO 1379
1378 GOSUB 6702
1379 GOSUB 7890:NEXT
1380 XOPEN #1,N$:IF B4=1 THEN B4=0:M=0:GOSUB 7650:GOTO 1310
1385 CLOSE:IF B5=1 THEN B5=0:GOTO 1540
1390 GOTO 1580
1430 REM *** OUTPUT DATA
1440 PRINT"■":CURSOR 9,8:PRINT"♦♦ SPECIFIC STATION ♦♦■■■■":PRINT
1450 GOSUB 5000:I1$=MID$(I1$,2,6):IF I1$="B" THEN 30
1451 O$=I1$:GOSUB 5520
1465 I=2:M=0:X=1:B4=0:B5=0
1500 GOSUB 7040
1510 Y=M-1:IF M=0 GOTO 1570
1512 IF A=2 GOTO 1630
1515 CLOSE #1:PRINT:PRINT
1517 U9=0
1518 FOR M=0 TO Y
1520 GOSUB 6650:GOSUB 6702:GOSUB 7800
1530 NEXT M
1535 IF B4=1 THEN B4=0:M=0:GOSUB 7650:GOTO 1500
1540 IF U8=1 GOSUB 8750
1542 PRINT TAB(28):"■* END *■"
1545 PRINT"■   ♦  ADVANCE TO THE NEXT( ANY KEY )"
1550 GET Z$:IF Z$="" THEN 1550
1555 GOTO 16
1560 REM ***  NO DATA
1570 CLOSE #1:IF B5=1 THEN B5=0:GOTO 1540
1580 PRINT"■":CURSOR 6,8:MUSIC M$
1590 PRINT "OBJECTIVE STATION NOT FOUND!!■■■":GOTO 1545
```

```
1600 PRINT"■◆ ADDITION OR CORRECTION FOR QSO DATA ◆":B=1
1610 INPUT"■      CALL SIGN PLEASE : ";I1$:MUSIC M1$:I1$=LEFT$(I1$,6)
1615 IF (I1$="B")+(I1$="N") THEN 30
1620 AX$=I1$:H=6:GOSUB 6400:I1$=AX$:GOTO 1465
1630 CLOSE
1631 Y=M:M=0:Z1=0:F1=0
1635 GOSUB 6650:GOSUB 6702
1640 PRINT"■*HAD QSO WITH THIS STATION B'Y";V;" TIMES"
1650 PRINT"■   DATA ";B;"  ( CORRECT OR ADD )":PRINT
1660 PRINT" 1. CALL SIGN          : ";A1$:I1$=A1$:GOSUB 5020
1670 PRINT" 2. QSO DATE           : ";D1$:I2$=D4$
1680 PRINT" 3. QSO START TIME     : ";B2$:I3$=B4$
1690 GOSUB 7520
1700 PRINT" 4. QSO BAND           : ";A2$;" MHZ":I4$=A5$
1710 PRINT" 5. MODE               : ";A3$:I5$=A3$:GOSUB 5620
1720 PRINT" 6. RST                : ";B1$:I6$=B1$:GOSUB 5710
1730 PRINT" 7. QRA                : ";C1$:I7$=C1$:GOSUB 5810
1740 PRINT" 8. CITY OR COUNTRY    : ";C2$:I8$=C2$:GOSUB 5920
1750 PRINT" 9. QSL CARD RECEIVED: ";A4$:I9$=A4$
1760 PRINT"10. QSL ISSUANCE NO. : ";D2$:J1$=D2$:GOSUB 6120
1770 PRINT"11. COMMENT            : ";D3$:J2$=D3$:GOSUB 6220
1780 PRINT"12. REMARKS            : ";B3$:J3$=B3$:GOSUB 6320
1790 GOSUB 7560
1795 CURSOR 1,19:PRINT"CORRECTION NUMBER PLEASE !!"
1800 INPUT"■ ( IF NOT CORRECT, PRESS '0' ) : ";Z$:C=VAL(Z$)
1810 IF C=0 THEN 2140
1820 Z1=1:GOSUB 7560
1830 IF (C<0)+(C>12) GOTO 1790
1835 GOTO 2000
1840 GOSUB 5520:GOSUB 6900
1850 INPUT #1(I),A$,B$,C$,D$
1855 IF EOF(#1) THEN 1895
1865 GOSUB 7350:IF B1<>74 THEN 1870
1866 IF (B2>47)*(B2<58) THEN 1870
1867 IF (B3>47)*(B3<58) THEN 1880
1870 GOSUB 6890
1880 I=I+4
1885 IF M=60 THEN B4=1:GOTO 1895
1890 GOTO 1850
1895 Y=M-1:IF M=0 THEN 1385
1905 U9=0:CLOSE
1908 FOR M=0 TO Y
1910 GOTO 1377
1912 GOSUB 6702
1920 IF B4=1 THEN B4=0:M=0:GOSUB 7650:GOTO 1850
1925 GOTO 1385
2000 PRINT"■■■■■■■"
2005 ON C GOSUB 5000,5100,5200,7600,5600,5700,5800,5900,6000,6100,6200,6300
2010 ON C GOTO 2020,2015,2030,2040,2050,2060,2070,2080,2090,2100,2110,2120
2015 CURSOR 23,5:GOSUB 7550:GOTO 1790
2020 CURSOR 23,4:PRINT MID$(I1$,2,10);"    ":GOTO 1790
2030 CURSOR 23,6:GOSUB 7590:GOTO 1790
2040 CURSOR 23,7:PRINT E$(I4);"  ":GOTO 1790
2050 CURSOR 23,8:PRINT I5$:GOTO 1790
2060 CURSOR 23,9:PRINT I6$:GOTO 1790
2070 CURSOR 23,10:PRINT I7$:GOTO 1790
2080 CURSOR 23,11:PRINT I8$:GOTO 1790
2090 CURSOR 23,12:PRINT I9$:GOTO 1790
2100 CURSOR 23,13:PRINT J1$:GOTO 1790
2110 CURSOR 23,14:PRINT J2$:GOTO 1790
2120 CURSOR 23,15:PRINT J3$
2130 GOTO 1790
2140 IF Z1=0 THEN 2175
2145 GOSUB 7500:I=F(M):S=I:T=I
2150 XOPEN #1,N$:INPUT #1(1),U
2160 IF F1=1 THEN 720
```

```
2165 PRINT #1(I),A$,B$,C$,D$
2170 CLOSE #1
2175 M=M+1:B=B+1:IF B>V GOTO 16
2180 GOTO 1635
2200 PRINT"■":MUSIC M1$:CURSOR 3,3:PRINT"◆◆ QSO (PERIOD-BASED LIST) ◆◆■"
2205 PRINT"■     ( INPUT IN 6 DIGITS )■■"
2210 INPUT"■■  FROM YEAR,MONTH,DAY?: ";I2$:U1$=I2$
2215 IF LEN(I2$)<>6 THEN MUSIC M$:PRINT "■■":GOTO 2210
2220 P1=1:F$=I2$:E$="0000":GOSUB 6830:K=J6
2223 D1$=I2$:GOSUB 6700:GOSUB 6702:I2$=D1$:U1$=I2$
2225 PRINT"■":TAB(26);I2$:PRINT:PRINT:PRINT
2230 INPUT"■■  UNTIL YEAR,MONTH,DAY?: ";I2$
2235 IF LEN(I2$)<>6 THEN MUSIC M$:PRINT"■■":GOTO 2230
2240 U2$=I2$:F$=I2$:E$="2359":GOSUB 6830:L=J6
2243 D1$=I2$:GOSUB 6700:GOSUB 6702:I2$=D1$
2245 PRINT"■";TAB(27);I2$:U2$=I2$
2250 IF K>L THEN MUSIC M$:GOTO 2200
2255 FOR U0=1 TO 1000:NEXT:GOSUB 5520:GOSUB 6900
2260 INPUT #1(I),A$,B$,C$,D$
2265 IF EOF(#1) THEN 2305
2275 F$=LEFT$(D$,6):E$=MID$(B$,4,4):GOSUB 6830:G=J6
2280 IF (G<K)+(G>L) THEN 2290
2285 GOSUB 6890
2290 I=I+4:IF G>L THEN 2305
2295 IF M=60 THEN B4=1:GOTO 2305
2300 GOTO 2260
2305 V=M-1:IF M=0 THEN 1385
2315 U9=0:CLOSE
2316 FOR M=0 TO V
2320 GOSUB 6650:IF U8=1 GOSUB 6721:GOTO 2325
2323 GOSUB 6702
2325 GOSUB 7950:NEXT
2330 XOPEN #1,N$:IF B4=1 THEN B4=0:M=0:GOSUB 7650:GOTO 2260
2335 GOTO 1385
2340 PRINT"■":CURSOR 3,4
2345 PRINT"◆◆ QSO BAND-BASED LIST ◆◆■■■■":GOSUB 7600:IF BK=1 THEN 30
2346 O$=E$(I4):IF I4=0 THEN 30
2350 GOSUB 5520:GOSUB 6900
2355 INPUT #1(I),A$,B$,C$,D$
2360 IF EOF(#1) THEN 2400
2370 IF I4$=MID$(A$,12,1) GOSUB 6890
2380 I=I+4
2390 IF M=60 THEN B4=1:GOTO 2400
2395 GOTO 2355
2400 V=M-1:IF M=0 THEN 1385
2410 U9=0:CLOSE
2412 FOR M=0 TO V
2415 GOSUB 6650:IF U8=1 GOSUB 6721:GOTO 2420
2418 GOSUB 6702
2420 GOSUB 8020:NEXT
2425 XOPEN #1,N$:IF B4=1 THEN B4=0:M=0:GOSUB 7650:GOTO 2355
2427 GOTO 1385
2440 PRINT"■":CURSOR 5,6
2443 PRINT"◆◆ QRA-BASED LIST ◆◆":PRINT"■■■■■■"
2445 GOSUB 5800:IF I7$="B" THEN 30
2446 O$=I7$:GOSUB 5520
2450 GOSUB 6900
2455 INPUT #1(I),A$,B$,C$,D$
2460 IF EOF(#1) THEN 2500
2470 IF I7$=LEFT$(C$,8) GOSUB 6890
2480 I=I+4
2490 IF M=60 THEN B4=1:GOTO 2500
2495 GOTO 2455
2500 V=M-1:IF M=0 THEN 1385
2510 U9=0:CLOSE
2512 FOR M=0 TO V
```

```
2515 GOSUB 6650:GOSUB 6702:GOSUB 8080
2520 NEXT
2525 XOPEN #1,N$:IF B4=1 THEN B4=0:M=0:GOSUB 7650:GOTO 2455
2530 GOTO 1385
2540 PRINT"■":CURSOR 3,8
2542 PRINT"◆◆  COMMENT-BASED LIST  ◆◆■■■■":GOSUB 6200
2543 IF J2$="B" THEN 30
2545 GOSUB 5520:O$=J2$:GOSUB 6900
2555 INPUT #1(I),A$,B$,C$,D$
2560 IF EOF(#1) THEN 2600
2570 IF J2$=RIGHT$(D$,6) GOSUB 6890
2580 I=I+4
2590 IF M=60 THEN B4=1:GOTO 2600
2595 GOTO 2555
2600 Y=M-1:IF M=0 THEN 1385
2610 U9=0:CLOSE
2612 FOR M=0 TO Y
2615 GOSUB 6650:IF U8=1 GOSUB 6721:GOTO 2620
2618 GOSUB 6702
2620 GOSUB 8140:NEXT
2625 XOPEN #1,N$:IF B4=1 THEN B4=0:M=0:GOSUB 7650:GOTO 2555
2630 GOTO 1385
2640 PRINT"■":CURSOR 7,10
2642 PRINT"◆◆◆ COMPLETE FILE LIST ◆◆◆"
2650 U8=0:O$=" ◆◆ COMPLETE FILE LIST ◆◆"
2655 CURSOR 5,13:GOSUB 5525:GOSUB 6900
2660 INPUT #1(I),A$,B$,C$,D$
2665 IF EOF(#1) THEN 2695
2680 GOSUB 6890:I=I+4
2685 IF M=60 THEN B4=1:GOTO 2695
2690 GOTO 2660
2695 Y=M-1:IF M=0 THEN 1385
2705 U9=0:CLOSE
2706 FOR M=0 TO Y
2710 GOSUB 6650:IF U8=1 GOSUB 6721:GOTO 2713
2712 GOSUB 6702
2713 GOSUB 8200
2715 NEXT
2725 XOPEN #1,N$:IF B4=1 THEN B4=0:M=0:GOSUB 7650:GOTO 2660
2730 GOTO 1385
3000 PRINT"■":CURSOR 2,7:PRINT "┌──────────────────────────────────┐"
3010 PRINT" │      WHAT TIME IS IT NOW ?       │"
3020 PRINT" └──────────────────────────────────┘":MUSIC M1$
3030 INPUT"■■■■(INPUT DATE IN 3-4 DIGITS) : ";I3$:H=LEN(I3$):Z=VAL(I3$)
3040 IF (H<3)+(H>4)+(Z>2400) THEN 3000
3060 IF H=3 THEN TI$="0"+I3$+"00":GOTO 3080
3070 TI$=I3$+"00"
3080 RETURN
3090 I3$=TI$:IF LEFT$(I3$,1)="0" THEN I3$=MID$(I3$,2,3):I3$=" "+I3$:GOTO 3110
3100 I3$=LEFT$(I3$,4)
3110 PRINT" 3.  QSO START TIME      ";:GOSUB 7590
3120 PRINT"  ( Y OR N )"
3140 GET Z1$:IF Z1$="" THEN PRINT"■■■":GOTO 3090
3150 IF Z1$="Y" THEN F1=1:MUSIC M1$:RETURN
3155 IF Z1$="B" THEN I3$=Z1$:RETURN
3160 PRINT"■■";SPC(38);"■":GOTO 5200
5000 PRINT" 1.  CALL SIGN (UP TO 10 CHARS)":PRINT TAB(12)::INPUT": ";I1$
5005 IF I1$="B" THEN RETURN
5010 IF I1$="N" THEN PRINT"■";SPC(38);"■":MUSIC M$:GOTO 5000
5020 IF LEN(I1$)>10 THEN I1$=LEFT$(I1$,10)
5030 I1$="*"+I1$:AX$=I1$:H=11:GOSUB 6400:I1$=AX$:MUSIC M1$:RETURN
5100 PRINT" 2.  DATE OF QSO (6 DIGITS)"
5110 PRINT TAB(12)::INPUT": ";I2$
5115 IF I2$="B" THEN RETURN
5120 IF I2$="N" THEN I2$="000000"
5130 IF LEN(I2$)<>6 THEN PRINT"■■";:MUSIC M$:GOTO5100
```

```
5140 PRINT"■":TAB(25)::GOSUB 8800:F1=1:MUSIC M1$:RETURN
5200 PRINT" 3.  QSO START TIME (UP TO 3-4 DIGITS)"
5210 PRINT SPC(12)::INPUT": ";I3$
5215 IF I3$="B" THEN RETURN
5220 IF I3$="N" THEN I3$="0000":MUSIC M1$:RETURN
5230 IF (LEN(I3$)>4)+(LEN(I3$)<3) THEN PRINT"■■":MUSIC M$:GOTO 5200
5240 IF LEN(I3$)=3 THEN I3$=" "+I3$
5245 PRINT"■";TAB(24)::GOSUB 7590:F1=1:MUSIC M1$:RETURN
5300 PRINT"■ 4.  QSO BAND":PRINT
5307 PRINT TAB(8);" 0.      NO DATA"
5308 PRINT
5310 PRINT TAB(8);" 1.      1.9 MHZ  BAND"
5320 PRINT TAB(8);" 2.      3.6 MHZ  BAND"
5330 PRINT TAB(8);" 3.      3.8 MHZ  BAND"
5340 PRINT TAB(8);" 4.       7  MHZ  BAND"
5350 PRINT TAB(8);" 5.      14  MHZ  BAND"
5360 PRINT TAB(8);" 6.      21  MHZ  BAND"
5370 PRINT TAB(8);" 7.      28  MHZ  BAND"
5380 PRINT TAB(8);" 8.      50  MHZ  BAND"
5390 PRINT TAB(8);" 9.     144  MHZ  BAND"
5400 PRINT TAB(8);"10.     432  MHZ  BAND"
5410 PRINT TAB(8);"11.    1200  MHZ  BAND"
5420 PRINT TAB(8);"12.    2300  MHZ  BAND"
5430 PRINT TAB(8);"13.    5700  MHZ  BAND"
5440 PRINT TAB(8);"14.    OTHERS"
5445 PRINT
5460 INPUT"OBJECTIVE NUMBER : ";I4$:I4=VAL(I4$)
5470 IF I4$="B" THEN BK=1:RETURN
5490 IF (I4<0)+(I4>14) GOSUB 7770:MUSIC M$:PRINT"■■":GOTO 5460
5500 IF I4<10 THEN I4$=STR$(I4):MUSIC M1$:RETURN
5505 IF I4=0 THEN I4$="0"
5510 I4$=CHR$(I4+55):MUSIC M1$:RETURN
5520 PRINT"■":CURSOR 3,10
5525 PRINT"◆   OUTPUT DATA ON PRINTER   ◆"
5530 PRINT"■■                ( Y OR ANY KEY )■":Z1$=""
5550 GET Z1$:IF Z1$="" THEN 5550
5570 U8=0:P1=1:IF Z1$="Y" THEN U8=1:PRINT
5580 PRINT"■":CURSOR 6,11:PRINT"DATA  UNDER  PREPARATION !!":RETURN
5600 PRINT" 5.  MODE (UP TO 3 CHARS)"
5610 PRINT TAB(12)::INPUT": ";I5$
5615 IF I5$="B" THEN RETURN
5620 IF I5$="N" THEN I5$="---":MUSIC M1$:RETURN
5630 IF LEN(I5$)>3 THEN I5$=LEFT$(I5$,3)
5635 AX$=I5$:H=3:GOSUB 6400:I5$=AX$:MUSIC M1$:RETURN
5700 PRINT" 6.  RST (UP TO 3 CHARS)":PRINT TAB(12)::INPUT": ";I6$
5705 IF I6$="B" THEN RETURN
5710 IF I6$="N" THEN I6$="---":MUSIC M1$:RETURN
5720 IF LEN(I6$)>3 THEN I6$=LEFT$(I6$,3)
5730 AX$=I6$:H=3:GOSUB 6400:I6$=AX$:MUSIC M1$:RETURN
5800 PRINT" 7.  QRA (UP TO 8 CHARS)":PRINT TAB(12)::INPUT": ";I7$
5805 IF I7$="B" THEN RETURN
5810 IF I7$="N" THEN I7$="--------":MUSIC M1$:RETURN
5820 IF LEN(I7$)>8 THEN I7$=LEFT$(I7$,8)
5830 AX$=I7$:H=8:GOSUB 6400
5840 I7$=AX$
5850 MUSIC M1$:RETURN
5900 PRINT" 8.  CITY OR COUNTRY (UP TO 8 CHARS)"
5910 PRINT TAB(12)::INPUT": ";I8$
5915 IF I8$="B" THEN RETURN
5920 IF I8$="N" THEN I8$="--------":MUSIC M1$:RETURN
5930 IF LEN(I8$)>8 THEN I8$=LEFT$(I8$,8)
5940 AX$=I8$:H=8:GOSUB 6400:I8$=AX$:MUSIC M1$:RETURN
6000 PRINT" 9.  QSL CARD RECEIVED (Y OR N)"
6010 PRINT TAB(12)::INPUT": ";I9$
6015 IF I9$="B" THEN RETURN
6020 IF LEFT$(I9$,1)="Y" THEN I9$="R":MUSIC M1$:RETURN
```

```
6030 I9$="*"
6040 MUSIC M1$:RETURN
6100 PRINT"10.  QSL ISSUANCE NO.(UP TO 4 CHARS)"
6110 PRINT TAB(12)::INPUT": ";J1$
6115 IF J1$="B" THEN RETURN
6120 IF J1$="N" THEN J1$=" -- ":MUSIC M1$:RETURN
6125 K1=LEN(J1$)
6130 IF K1>4 THEN J1$=LEFT$(J1$,4)
6135 IF K1=4 THEN 6160
6140 FOR N=K1 TO 3:J1$=" "+J1$:NEXT
6160 MUSIC M1$:RETURN
6200 PRINT"11.  COMMENT (UP TO 6 CHARS)"
6210 PRINT TAB(12)::INPUT": ";J2$
6215 IF J2$="B" THEN RETURN
6220 IF J2$="N" THEN J2$="------":MUSIC M1$:RETURN
6230 IF LEN(J2$)>6 THEN J2$=LEFT$(J2$,6)
6240 AX$=J2$:H=6:GOSUB 6400
6250 J2$=AX$
6260 MUSIC M1$:RETURN
6300 PRINT"12.  REMARKS (UP TO 9 CHARS)"
6310 PRINT TAB(12)::INPUT": ";J3$
6315 IF J3$="B" THEN RETURN
6320 IF J3$="N" THEN J3$="---------":MUSIC M1$:RETURN
6330 IF LEN(J3$)>9 THEN J3$=LEFT$(J3$,9)
6340 AX$=J3$:H=9:GOSUB 6400
6350 J3$=AX$:MUSIC M1$:RETURN
6400 AX=H-LEN(AX$)
6405 IF AX=<0 THEN RETURN
6410 FOR N=1 TO AX:AX$=AX$+" ":NEXT
6460 MUSIC M1$:RETURN
6500 PRINT"INPUT IN  SUCCESSION ?"
6510 INPUT"IF SO, PRESS 'Y', IF NOT, 'N': ";I$:RETURN
6640 REM ♦♦♦  OUT DATA FROM DIM.
6650 A1$=MID$(A$(M),2,10):A6$=MID$(A$(M),12,1):A2=ASC(A6$)-48
6660 IF A2>9 THEN A2=A2-7
6670 A2$=E$(A2):A3$=MID$(A$(M),13,3):A4$=MID$(A$(M),16,1):A5$=MID$(A$(M),12,1)
6680 B1$=MID$(A$(M),17,3):B3$=MID$(A$(M),24,9)
6690 C1$=MID$(A$(M),33,8):C2$=MID$(A$(M),41,8):D2$=MID$(A$(M),55,4)
6695 D3$=RIGHT$(A$(M),6):B2$=MID$(A$(M),20,4):B4$=B2$
6697 D1$=MID$(A$(M),49,6):D4$=D1$
6700 S$=LEFT$(D1$,2):T$=MID$(D1$,3,1):U$=MID$(D1$,4,1):V$=MID$(D1$,5,1)
6701 W$=RIGHT$(D1$,1):RETURN
6702 IF (T$="0")*(U$="0") THEN D1$=" "+W$+"/"+U$+", "+S$+" ":GOTO 6708
6703 IF T$="0" THEN D1$=U$+W$+"/"+U$+", "+S$+" ":GOTO 6708
6704 IF V$="0" THEN D1$=W$+"/"+T$+U$+", "+S$+" ":GOTO 6708
6705 D1$=U$+W$+"/"+T$+U$+","+S$+" "
6708 IF MID$(B2$,3,1)="0" THEN B2$=LEFT$(B2$,2)+": "+RIGHT$(B2$,1)+" ":RETURN
6715 B2$=LEFT$(B2$,2)+": "+RIGHT$(B2$,2):RETURN
6721 IF (T$="0")*(U$="0") THEN D1$=" "+W$+"/"+U$+", "+S$:GOTO 6725
6722 IF T$="0" THEN D1$=U$+W$+"/"+U$+", "+S$:GOTO 6725
6723 IF V$="0" THEN D1$=W$+"/"+T$+U$+", "+S$:GOTO 6725
6724 D1$=U$+W$+"/"+T$+U$+","+S$
6725 IF MID$(B2$,3,1)="0" THEN B2$=LEFT$(B2$,2)+": "+RIGHT$(B2$,1):RETURN
6727 B2$=LEFT$(B2$,2)+":"+RIGHT$(B2$,2):RETURN
6800 IF I1$=MID$(A$,2,6) THEN A$(M)=A$+B$+C$+D$:F(M)=I:M=M+1
6810 I=I+4:RETURN
6830 J1=VAL(LEFT$(F$,2))*535680:J2=VAL(MID$(F$,3,2))*44640
6850 J3=VAL(MID$(F$,5,2))*1440
6855 E$=MID$(E$,4,4)
6856 IF LEFT$(E$,1)=" " THEN E$="0"+MID$(E$,2,3)
6860 J4=VAL(LEFT$(E$,2))*60
6870 J5=VAL(RIGHT$(E$,2)):J6=J1+J2+J3+J4+J5:RETURN
6890 A$(M)=A$+B$+C$+D$
6895 F(M)=I:M=M+1:B5=1:RETURN
6900 M=0:I=2:X=1:B4=0:B5=0
6910 XOPEN #1,N$:RETURN
```

```
7040 XOPEN #1,N$
7050 F(M)=I
7055 INPUT #1(I),A$,B$,C$,D$
7060 IF EOF(#1) THEN RETURN
7070 GOSUB 6800:IF M=60 THEN B4=1:RETURN
7075 GOTO 7050
7095 REM ***  PREPARATION DATA
7100 E$=B$:F$=D$:GOSUB 6830:J7=J6
7110 XOPEN #1,N$:INPUT #1(1),S:INPUT #1(2),W$
7140 IF EOF(#1) THEN 7220
7145 INPUT #1(S),W$,X$,Y$,Z$
7150 E$=X$:F$=Z$:GOSUB 6830
7160 T=S:S=S+4:PRINT #1(1),S
7180 IF J7<J6 THEN 7250
7190 PRINT #1(S),A$,B$,C$,D$
7200 CLOSE #1:RETURN
7220 PRINT #1(1),2:PRINT #1(2),A$,B$,C$,D$:GOTO 7200
7250 PRINT #1(S),W$,X$,Y$,Z$:S=S-4:T=T-4
7270 IF T<2 THEN 7190
7280 INPUT #1(T),W$,X$,Y$,Z$
7290 E$=X$:F$=Z$:GOSUB 6830
7300 IF J7<J6 THEN 7250
7310 PRINT #1(S),A$,B$,C$,D$:GOTO 7200
7320 X$="   "+STR$(X):X$=RIGHT$(X$,4):RETURN
7350 B1=ASC(MID$(A$,2,1)):B2=ASC(MID$(A$,3,1)):B3=ASC(MID$(A$,4,1)):RETURN
7500 A$=I1$+I4$+I5$+I9$:B$=I6$+I3$+J3$:C$=I7$+I8$:D$=I2$+J1$+J2$:RETURN
7520 Z1=0:I4=ASC(A6$)-48
7530 IF I4>9 THEN I4=I4-7
7540 I4$=E$(I4):RETURN
7550 PRINT RIGHT$(I2$,2);"/";MID$(I2$,3,2);", ";LEFT$(I2$,2):RETURN
7560 CURSOR 0,16
7563 FOR N=1 TO 7:PRINT SPC(38):NEXT N:RETURN
7590 PRINT LEFT$(I3$,2);" : ";RIGHT$(I3$,2):RETURN
7600 PRINT"■  1)  1.9       2)  3.6       3)  3.8"
7605 PRINT"  4)    7       5)   14       6)   21"
7610 PRINT" 7)   28       8)   50       9)  144"
7615 PRINT" 10)  432      11) 1200      12) 2300"
7620 PRINT" 13) 5700      14) OTHERS"
7625 PRINT"     QSO BAND":GOSUB 5460
7630 RETURN
7650 FOR N=0 TO 60
7655 A$(N)=""
7660 NEXT N:RETURN
7770 PRINT"■";SPC(39):PRINT"";:RETURN
7800 IF U8=1 THEN 8690
7810 IF U9=0 THEN U9=U9+1:GOSUB 7875
7820 PRINT D1$;" ";B2$;" ";A2$;" ";A3$;" ";C1$;" ";D2$:U9=U9+1
7830 IF U9<=15 THEN RETURN
7840 U9=0: PRINT"■◆ ADVANCE TO THE NEXT (PRESS ANY KEY )"
7850 GET Z$:IF Z$="" THEN 7850
7860 RETURN
7875 PRINT"■  ◆◆ SPECIFIC STATIONS ( ";O$;" )■"
7880 PRINT"■   DATE    TIME  BAND EM. QRA    QSL-NO■":RETURN
7890 IF U8=1 THEN 8400
7900 IF U9=0 THEN U9=U9+1:GOSUB 7930
7910 PRINT A1$;D1$;" ";A2$;" ";A3$;" ";C2$;" ";A4$:U9=U9+1
7920 GOTO 7830
7930 PRINT"■ ◆ PRIFIX-BASED LIST ( ";Z$;" )■"
7940 PRINT"CALL SIGN    DATE   BAND EM. QTH  QSL-R■":RETURN
7950 IF U8=1 THEN 8340
7960 IF U9=0 THEN U9=U9+1:GOSUB 7980
7965 PRINT A1$;" ";D1$;" ";B2$;"  ";A2$;" ";A3$:U9=U9+1
7970 GOTO 7830
7980 PRINT"■ ◆◆ PRIOD-BASED LIST ◆◆■"
7990 I2$=U2$:PRINT TAB(8);U1$;SPC(3);"-";SPC(4);U2$
8010 PRINT"■CALL SIGN     DATE    TIME   BAND EM.■":RETURN
```

```
8020 IF U8=1 THEN 8460
8030 IF U9=0 THEN U9=U9+1:GOSUB 8060
8040 PRINT A1$;D1$;" ";A2$;" ";D2$;" ";A4$;" ";C1$:U9=U9+1
8050 GOTO 7830
8060 PRINT"■ ♦ BAND-BASED LIST  ";O$;" MHZ BAND"
8070 PRINT"■CALL SIGN    DATE   EM.  QSL-R QRA■":RETURN
8080 IF U8=1 THEN 8550
8090 IF U9=0 THEN U9=U9+1:GOSUB 8120
8100 PRINT A1$;D1$;" ";A2$;" ";A3$;" ";C2$;" ";A4$:U9=U9+1
8110 GOTO 7830
8120 PRINT"■ ♦QRA-BASED LIST ( ";O$;" )■"
8130 PRINT"CALL SIGN    DATE   BAND EM. QTH  QSL-R■":RETURN
8140 IF U8=1 THEN 8630
8150 IF U9=0 THEN U9=U9+1:GOSUB 8180
8160 PRINT A1$;" ";D1$;" ";A2$;" ";A3$;" ";C1$:U9=U9+1
8170 GOTO 7830
8180 PRINT"■ ♦COMMENT-BASED LIST ( ";O$;" )■"
8190 PRINT"CALL SIGN     DATE   BAND EM. QRA■":RETURN
8200 IF U8=1 THEN 8260
8210 IF U9=0 THEN U9=U9+1:GOSUB 8240
8220 PRINT A1$;" ";D1$;" ";B2$;" ";A2$;" ";A3$;"  ";A4$:U9=U9+1
8230 GOTO 7830
8240 PRINT"■   ";O$;"■"
8250 PRINT"CALL SIGN     DATE    TIME  BAND EM. Q/R■":RETURN
8260 IF U9=0 GOSUB 8300
8280 PRINT/P A1$;"|";D1$;"|";B2$;"|";A2$;"|";A3$;"|";B1$;"|";C1$;"|";
8282 PRINT/P C2$;"|";D2$;"|";A4$;"|";D3$;"|";B3$:U9=U9+1:X=X+1
8285 IF M=V THEN 8287
8286 IF U9<30 GOTO 8289
8287 PRINT/P"-----------------------------------------------------------------";
8288 PRINT/P"-+-+---------+---------■":U9=0:RETURN
8289 PRINT/P"-----------------------------------------------------------------";
8290 PRINT/P"-+-+---------+---------":RETURN
8300 GOSUB 8840:PRINT/P"  ♦♦ COMPLETE FILE LIST ♦♦■"
8315 GOSUB 8850
8316 PRINT/P"-----------------------------------------------------------------";
8317 PRINT/P"-+-+---------+---------"
8320 PRINT/P"CALL SIGN |  DATE  |TIME |BAND|EM.|RST|  QRA   |  QTH   |Q/NO";
8325 PRINT/P" R|COMME.|REMARKS"
8326 PRINT/P"-----------------------------------------------------------------";
8330 PRINT/P"-+-+---------+---------":RETURN
8340 IF U9=0 GOSUB 8350
8345 GOTO 8280
8350 GOSUB 8840
8355 PRINT/P" ♦♦ PERIOD-BASED LIST ♦♦":PRINT/P"   (";U1$;" - ";U2$;")■"
8360 GOTO 8315
8400 IF U9=0 GOSUB 8430
8410 GOTO 8280
8430 GOSUB 8840:PRINT/P" ♦♦ PREFIX-BASED LIST (";Z$;") ♦♦■"
8450 GOTO 8315
8460 IF U9=0 GOSUB 8500
8470 GOSUB 7320
8480 PRINT/P X$;"|";A1$;"|";D1$;"|";B2$;"|";A3$;"|";B1$;"|";C1$;"|";
8485 PRINT/P C2$;"|";D2$;"|";D3$;"|";B3$:U9=U9+1:X=X+1
8486 IF M=V THEN 8488
8487 IF U9<30 GOTO 8490
8488 PRINT/P"-----------------------------------------------------------------";
8489 PRINT/P"-+-+---------+---------■":U9=0:RETURN
8490 PRINT/P"-----------------------------------------------------------------";
8491 PRINT/P"-+-+---------+---------":RETURN
8500 GOSUB 8840:PRINT/P"  ♦♦ QSO BAND LIST (";O$;" MHZ) ♦♦■":GOSUB 8850
8525 PRINT/P"-----------------------------------------------------------------";
8530 PRINT/P"-+-+---------+---------"
8540 PRINT/P" NO|CALL SIGN |  DATE  |TIME |EM.|RST| QRA    |  QTH   |";
8545 PRINT/P "Q/NO|COMME.|REMARKS"
8546 PRINT/P"-----------------------------------------------------------------";
```

```
8547 PRINT/P"─┴──────┴────────────":RETURN
8550 IF U9=0 GOSUB 8600
8560 GOSUB 7320
8570 PRINT/P X$;"|";A1$;"|";D1$;"|";B2$;"|";A2$;"|";A3$;"|";B1$;"|";
8572 PRINT/P C2$;"|";D2$;"|";A4$;"|";D3$;"|";B3$;:PRINT/P:U9=U9+1:X=X+1
8575 IF M=Y THEN 8577
8576 IF U9<30 GOTO 8579
8577 PRINT/P"────┴──────────┴──────────┴──────┴────┴──────┴──────────┘";
8578 PRINT/P"────┴─┴───────┴──────────────":U9=0:RETURN
8579 PRINT/P"────┼──────────┼──────────┼──────┼────┼──────┼──────────┤";
8580 PRINT/P"────┼─┼───────┼──────────────":RETURN
8600 GOSUB 8840:PRINT/P" ♦♦ QRA-BASED LIST (";O$;") ♦♦▣":GOSUB 8850
8610 PRINT/P"────┬──────────┬──────────┬──────┬────┬──────┬──────────┬────";
8612 PRINT/P"───┬─────┬──────────────"
8615 PRINT/P" NO|CALL SIGN |  DATE   | TIME |BAND|EM.|RST|  QTH   |Q/N";
8620 PRINT/P"O R|COMME.|REMARKS"
8622 PRINT/P"────┼──────────┼──────────┼──────┼────┼──────┼──────────┼────";
8623 PRINT/P"─┼─┼──────┼────────────":RETURN
8630 IF U9=0 GOSUB 8660
8635 GOSUB 7320
8640 PRINT/P X$;"|";A1$;"|";D1$;"|";B2$;"|";A2$;"|";A3$;"|";B1$;"|";C1$;
8645 PRINT/P"|";C2$;"|";D2$;"|";A4$;"|";B3$:U9=U9+1:X=X+1
8646 IF Y=M THEN 8648
8647 IF U9<30 GOTO 8650
8648 PRINT/P"────┴──────────┴──────────┴──────┴────┴──────┴──────────┴────";
8649 PRINT/P"──────┴─────┴─┴────────────▣":U9=0:RETURN
8650 PRINT/P"────┼──────────┼──────────┼──────┼────┼──────┼──────────┼────";
8651 PRINT/P"──────┼──────┼─┼────────────":RETURN
8660 GOSUB 8840:PRINT/P" ♦♦ COMMENT-BASED LIST (";O$;") ♦♦▣":GOSUB 8850
8670 PRINT/P"────┬──────────┬──────────┬──────┬────┬──────┬──────────┬────";
8672 PRINT/P"──────┬──────┬────────────"
8675 PRINT/P" NO|CALL SIGN |  DATE  |TIME |BAND|EM.|RST| QRA    | QTH";
8676 PRINT/P"    |Q/NO R|REMARKS"
8677 PRINT/P"────┼──────────┼──────────┼──────┼────┼──────┼──────────┼────";
8678 PRINT/P"──────┼──────┼─┼────────────":RETURN
8690 IF U9=0 GOSUB 8725
8695 GOSUB 7320
8700 PRINT/P X$;"|";D1$;"|";B2$;"|";A2$;"|";A3$;"|";B1$;"|";
8705 PRINT/P C1$;"|";C2$;"|";D2$;"|";A4$;"|";D3$;"|";B3$:U9=U9+1:X=X+1
8706 IF Y=M GOTO 8710
8707 IF U9<30 THEN 8713
8710 PRINT/P"────┴──────────┴──────┴────┴──────┴──────────┴──────────┴────┘";
8712 PRINT/P"─┴──────┴──────────────":U9=0:RETURN
8713 PRINT/P"────┼──────────┼──────┼────┼──────┼──────────┼──────────┼────┤";
8714 PRINT/P"─┼──────┼──────────────":RETURN
8725 GOSUB 8840:PRINT/P" ♦♦ QSO DATA FOR ";O$;" STATION ♦♦▣":GOSUB 8850
8727 PRINT/P"────┬──────────┬──────┬────┬──────┬──────────┬──────────┬────";
8728 PRINT/P"─┬──────┬──────────────"
8735 PRINT/P" NO|  DATE   | TIME |BAND|EM.|RST| QRA    | QTH    |Q/NO R|";
8740 PRINT/P"COMME.|REMARKS"
8741 PRINT/P"────┼──────────┼──────┼────┼──────┼──────────┼──────────┼────┬";
8742 PRINT/P"─┼──────┼──────────────":RETURN
8750 PRINT/P:PRINT/P:PRINT/P TAB(60);"* END *"
8760 PRINT"▣▣▣▣▣▣▣▣▣":RETURN
8800 S$=LEFT$(I2$,2):T$=MID$(I2$,3,1):U$=MID$(I2$,4,1):V$=MID$(I2$,5,1)
8805 W$=RIGHT$(I2$,1)
8810 IF (T$="0")*(V$="0") THEN O1$=" "+W$+"/"+U$+", "+S$+" ":GOTO 8835
8820 IF T$="0" THEN O1$=V$+W$+"/"+U$+", "+S$+" ":GOTO 8835
8825 IF V$="0" THEN O1$=W$+"/"+T$+U$+", "+W$+" ":GOTO 8835
8830 O1$=V$+W$+"/"+T$+U$+","+S$+" "
8835 PRINT O1$:RETURN
8840 PRINT"▣":PRINT/P"▣":PRINT/P:PRINT/P:PRINT/P:PRINT/P"▣":RETURN
8850 PRINT/P TAB(70);"P - ";P1;"▣":P1=P1+1:RETURN
8860 PRINT"▣":CURSOR 17,10:PRINT"E N D":CURSOR 0,23:END
```

Hop, Step, Jump from a heavy stock book
(INVENTORY CONTROL PROGRAM)

Now, we wonder how the stock book is being updated by those who run their own business or small company. Using pen and paper ? You have to take out a heavy stock book to look for a transaction item every time a transaction is done and adjust the number of inventory and its value, so far as inventory control is concerned.

Why don't you give a rest to pen and paper for the time being, even if you are an expert ? Instead, let us try a computer with floppy disks and printer. Here, we provide you with a computer-controlled inventory control program for your own use. All inventory data is stored in this floppy diskette. And, you can take out any data at any time you want. Use of the printer allows you to prepare an inventory list automatically. This will enable you to register stock data as many as 1000 items at the most.

Before running the program

Just one word before running this program. Prepare one extra diskette for storing data. This extra diskette must be loaded in the drive unit assigned for present DIR execution. As soon as the extra diskette is at hand, it is now ready to run the program. This program, however, never be interrupted by using the BREAK key.

Date entry and job list

Immediately after commencement of program execution, the disk drive unit is used for initializing the computer. Then, input guidance is developed on the CRT calling for the input of the date (Photo 1).

```
********  INVENTORY CONTROL  **********

WHAT'S THE DATE TODAY?
[YEAR]   80
[MONTH]  1
[DAY]    19
```

Photo 1

First, enter "80" if the year is 1980, "1" for January, and "19" for the present day. When the date entry is over, a message is displayed on the CRT screen (Photo 2). This information shows the contents of the inventory management programs, which is called a job list. Eight kinds of job are prepared for inventory management, each one having different type of job.

```
**********     JOB LIST     **********

 [1]  REGISTRATION OF MASTER FILE
 [2]  CORRECTION OF MASTER FILE
 [3]  INFORMATION OF EACH FILE
 [4]  INCOMING OR OUTCOMING STOCK
 [5]  FILE COPY
 [6]  SIZE-MODEL REFERENCE TABLE
 [7]  DELETION OF FILE
 [8]  END OF ALL JOB

            JOB
```

Photo 2

Now, look at the CRT screen, the cursor is flickering at the lower side of the screen, asking you which job you wish to select. Pick up one with its job number, as there are jobs from Job 1 to Job 8. If Job 5 is to be assigned, depress the key "5." As a detailed flow of the job will be discussed step by step, understand it while executing the program by yourself.

JOB 1: Preparation of the stock book

In order to release you from the heavy stock book, you have to exert a little bit of labour. We are going to transfer the inventory data from the stock book into the diskette. It is mandatory to execute the Job 1 before utilizing this program. Now, hit the key "1" to select the program.

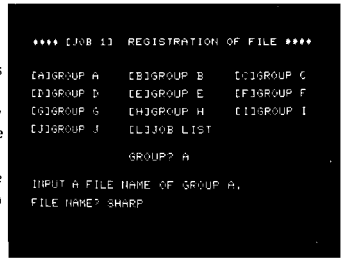


Photo 3

Don't omit the file name

Photo 3 shows that the Job 1 is now under way. You can use 10 files from the Group A to J. Up to 100 items can be registered per group.

The flickering cursor indicates that the computer is awaiting your key entry. If Group A is to be picked, depress the key "A." However, the computer will request through the CRT screen "INPUT A FILE NAME OF GROUP A," if you are to use that group for the first time. So, make an entry of the file name up to 16 characters. Once the file name has been input, this process is then omitted.

P [P] for print mode, and code generation by the computer

Now, the CRT screen goes to change display instruction (Photo 4). At this stage, you must select the print mode, whether the data to be entered next is to be printed out or not. Notation "P [P]" shows the print mode and "P [X]" shows the non-print mode. Depressing the key "P" while "P [P]" is shown, it changes to "P [X]" and vice versa. However, depression of any other key causes no function at all.

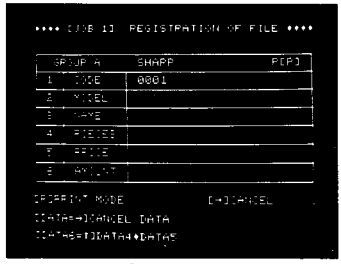


Photo 4

As soon as the print mode is selected, a 4 digits are shown in column 1, which is the merchandize code allocated particularly for that one items. The first digit of the code corresponds with the group the merchandize belongs, Group A to "0," Group B to "1," and so on. The remainder of the code represents the item number in the group, ranging from "001" to "100."

Input; successive merchandize data entry, "$" for the total

From here on you will have to exert a little bit of effort. Successive key entry has to be carried out. Total 6 items have to be filled in for each item of merchandize; CODE (merchandize code), MODEL (model name), NAME (merchandize name), PIECES (number of stock), PRICE (unit price), and AMOUNT (total value); each one having digit limit, 4, 8, 8, 12, 12, and 16 digits. This digital capacity must be observed as absolute. Now, try to enter the data step by step.

Photo 5 shows the contents of the merchandize code "0003" with the model name of "RG-2800", merchandize name "CASSETTE", inventory number "1500" and unit price "48", except the total column showing abnormal "$," seemingly an input error.

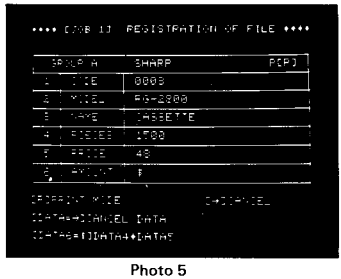


Photo 5

$ = (inventory number) x (unit price)

That's it. It is a multiplying symbol for the inventory number and the unit price. Using this means brings out the result up to 16 digits. Of course, you can calculate the result by yourself. But, avoid using exponential and formula for the numerical input. No guaranee will be made. Overflow beyond 17 digits by "$" is not accepted by the computer.

" → " for cancel, and to repeat if nothing has been done

Immediately after the entry of data for one item of merchandize, the data is then stored in the disk. And, if the print mode is assigned, the contents of the data is printed on the printer. Then, the CRT screen restores the display condition to Photo 4. The printing mode retains the previous state. Further, " → " input is effective at all key entries to return to the top of the Job 1.

JOB 2: Is there any error in data entry?

Now, you seem kind of tried of continuous key input at the Job 1. But, are you sure all the data was input correctly ?

You would't say "Ah" after storing the data in the disk. You wouldn't be able to correct disk data with a pencil, would you ?

Well, anyone can make mistake. Don't worry, you can make correction in easily manner with Job 2. Now, hit the key "2" at JOB LIST.

"1": Correction by merchandize code

At the computer is ready for your key input, depress the key "1" when you know the merchandize code of the merchandize to be corrected. (Depression of the key "L" causes a jump to the JOB LIST.) Now, enter 4 digits of merchandize code.

In Photo 6, the merchandize code is "0003," implying to take out the 3th merchandize in the Group A from the disk contents. Depression of the "CR" key causes the computer to take it out on the spot. This means is the most effective as it takes out the desired data without looking it up.

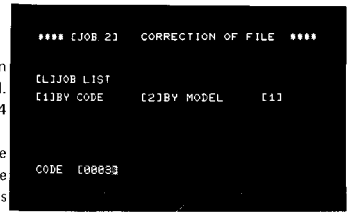


Photo 6

"2": Correction by model name

The data can be retrieved by the model name, as well. Depress the key "2" for this operation. Next, indicate which group is to be selected. If it is Group A, depress the key "A." As the computer successively asks for the model name, enter the data within 8 characters.

In the case of Photo 7, it shows that the merchandize of which name is "RG-2800" of the Group A is to be taken out.

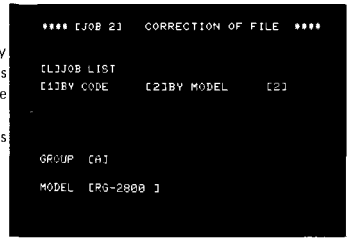


Photo 7

"2" through "6": Correction items

When the data is taken out after the code or model name designation, the CRT displays the information as in Photo 8. Now, you must be ready to correct relevant data. Key assignment ranging from "2" to "6" corresponds with the model name, merchandize name, inventory number, unit price, and total value.

If the model name is to be correct, depress the key "2." As the cursor appears in the model name column, enter correct information and depress the "CR" key. The "$" key is operative at the total column. You must note that it is not allowed to change the merchandize code. In Photo 8 is shown the correction procedure for inventory number.

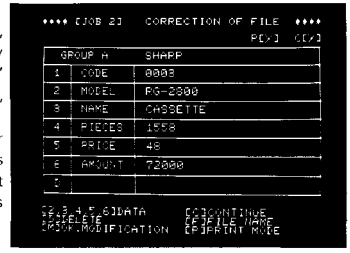


Photo 8

"F" for changing the file name, and "M" for the completion of correction operation

Even after the completion of corrections, the cursor stays flickering at the upper corner of the screen, awaiting for further correction allowing successive correction, among which is included file name correction. Depression of the key "F" allows the file name to be corrected within 16 characters. However, the file name already assigned to another file can not be used again. It will rejected by the computer as an error.

When all corrections are done, depress the key "M," which will enter the corrected information into the relevant data storage. But, with respect to the file name, it will be corrected right after the entry of the corrected data.

In Photo 9 is shown the state that the file name "SHARP" is changed to "SHARP CASSETTE."

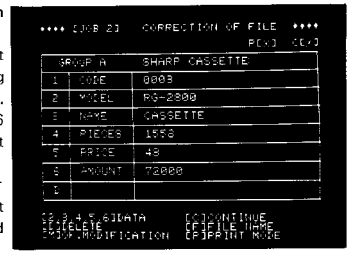


Photo 9

C [C] for continuous retrieval mode

If successive data is to be read upon completing correction of the currently displayed data, depression of the key "C" will enable you to take out the successive data on the display. The initial state "C [X]" will be changed to "C [C]" when the key "C" is depressed. The indication "C [X] is that the computer is out of the continuous retrieval mode.

The indication "P [ ]" also shows the print mode in the same manner as in the Job 1 and alteration of the printing mode is achieved by the depression of the key "P." The printing mode can be changed any time when the cursor is flickering. Photo 10 shows when both the printing mode and the continuous retrieval mode are on.

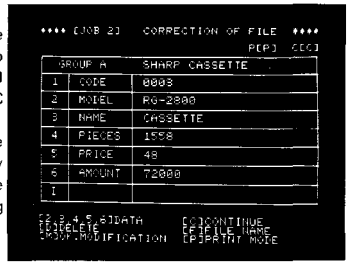


Photo 10

"D" for deletion

When the contents of an item of merchandize has become unnecessary, depression of the key "D" causes the display "DELETE." as in Photo 11 along with appearance of the cursor for the guidance of input items to be deleted. Deletion must be commanded with a positive integer.

As the displayed code is "0003" and the number of deletion is "5" in Photo 11, it means to delete merchandize items having the code from "0003" to "0007." If the number of deletion is "1," only the item of merchandize having the code "0003" is deleted.

In all key-in points of the Job 2, the cancel key "→" is also effective and depression of the key causes a return to the initial step.

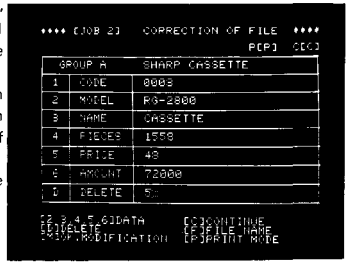


Photo 11

JOB 3: Informer

Because there are 10 groups of files, you are not confused about file names, are you ? Do you still remeber which file has been given to what group and how many items of merchandize registered in that file ?

What gives clear recognition of all file registration is Job 3. So, hit key "3" in the JOB LIST.

Photo 12 exhibits an example. The Group A is given with the file name "SHARP CASSETTE," comprising 11 registered merchandize items, and the Group B was given with the file name "LONDON" comprising 100 registered merchandize items. Symbols "*" and "***" after "NO" express that the respective file name is in registration or that the particular file is in full occupation with the complete file name registration. Because of this, there may be a registered file without item registration even if the file name has been registered. Those which are blank in the file name column are not registered yet.

It will be possible to jump from this Job to another Job. In such a case, the desired Job has to be indicated by the job number. If it were to jump to the Job 1, depression of the key "1" will make it jump to Job 1. Of course, depression of the key "L" causes a jump to the JOB LIST.

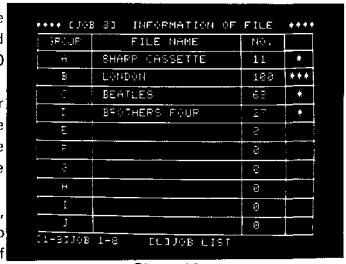


Photo 12

JOB 4: Bookkeeping

Assume that we have accepted shipment of 100 cassettes and sold out 10 car cassettes. Well, we have to update the stock book now. Make a correction with Job 2 ? No, you don't. There is a job specifically arranged for that purpose. Job 4 is the one. So, let's hit the key "4" at the JOB LIST.

Take a good look at Photo 13. We have seen that before somewhere in this text. Right, it is the one that appeared at Job 2. Though this you must designate the merchandize by the code or the model name in order to fetch the data from the disk. If you have forgotten the procedure, you have to refer to that section describing Job 2.

When the desired information is on the CRT screen, it looks something like Photo 14. Indications "P [ ]" and "C [ ]" are also shown on the screen as in Job 2, indicating the given printing mode and continuous retrieval mode. But, you have to make selection of each mode before executing transaction.

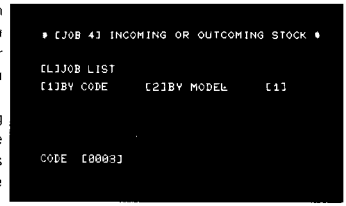


Photo 13

"I" for incoming stock

When the key "I" is depressed while the cursor is flickering in the left upper corner of the screen, it will become ready for accepting data for incoming transaction (Photo 14). Those which indicated with asterisk in columns, 4, 5, and 6, are the data presently stored in the disk. Incoming transaction data have to be entered in the columns indicated with "I." Input capacity is the same as in Job 1.

If the unit price of the incoming merchandize is the same as the previous unit price, you can use "*" for the unit price (see Photo 14). As for the total amount of the incoming merchandize, it will be much more effective to use "$," if it is "(number of incoming merchandize) x (unit price of incoming merchandize)." Photo 14 shows that 100 units of the cassette named "RG-2200" are received with the unit price of 48.

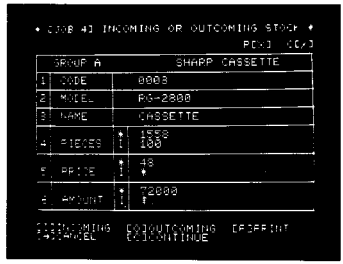


Photo 14

"O" for outgoing stock

In the case of outgoing stock, the key "O" is used. Sequence of key input is the same except the data is related to shipment. Photo 15 shows how the data for outgoing transactions are entered. It shows that 10 units of the car radio called "AR-947" are shipped out with a unit price of 43.

When all the data is entered, number of incoming transaction is added to the balance as well as adding to the total, to be stored in the disk, in the case of incoming stock. In the case of outgoing transaction, new balance of the stock number and total are calculated and stored in the disk.

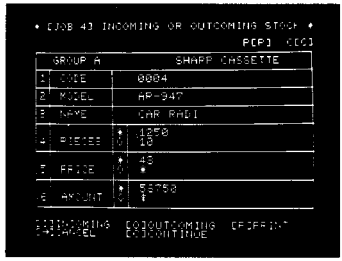


Photo 15

Only the new transactions are printed on the printer. When the successive data is read in the continuous retrieval mode, both the printing mode and the continous retrieval mode hold the previous state. Even in this job, it is possible to use the " → " key during all key entries, which causes a return to the top of Job 4.

JOB 5: Summarizing the disk data

Even if much labour has been exerted in making entry of data, it is not possible to see what has been written in a floppy disk. We wonder if the data is contained in such a thin disk.

Let Job 5 take care of looking into the contents of the disk. So, hit key "5" at the JOB LIST.

"A": Full list

When Job 5 is selected, the CRT screen turns into the state shown in Photo 16. First of all, the name of the desired group has to be indicated. If it is the group F, depress the key "F." Then, the computer will come to ask whether you wish the entire list or a partial list. If you want to see all of merchandizes registered in the group, depress the key "A." Then, the computer will start reading data from the disk and print it out on the printer one after another.

In Photo 16 is designated the Group A and so instructed to list all the contents of the respective file. An example of output is shown in the table below. Numerals are separated by comma at each 3 digits and the grand total is shown in the last colume.

Photo 16

INVENTORY LIST

<GROUP A> SAMPLE CATALOGUE 10.JUL. 80

CODE	MODEL	NAME	PIECES	UNIT PRICE	AMOUNT
0001	AA-80K	COMPUTER	1,500	198,000	297,000,000
0002	AA-80FD	FLOPPY	1,250	298,000	372,500,000
0003	AA-80FDK	E-FLOPPY	1,250	325,000	406,250,000
0004	AA-80P3	PRINTER	25,000	168,000	4,200,000,000
0005	AA-80I/O	I/O BOX	32,000	29,800	953,600,000
0006	AA-80FIO	FIO-CARD	540	27,000	14,580,000
0007	AA-80FMD	M.DISK	500	10,000	5,000,000
0008	AA-80F15	F-CABLE	540	4,300	2,322,000
0009	AA80T20	M.LANG.	2,000	6,000	12,000,000
0010	AA-80TU	S.PRO.	2,000	20,000	40,000,000
0011	AA-80FBD	B.DISKET	3,000	2,400	7,200,000
0012	AA-80T	APP.TAPE	6,000	3,000	18,000,000
0013	XX-730	FAX	10	720,000	7,200,000
0014	XX-810	FAX	15	898,000	13,470,000
0015	XX-205	FAX	35	238,000	8,330,000
0016	XX-740	FAX	15	495,000	7,425,000
0017	XX-725	FAX	10	546,000	5,460,000
0018	ZZ-5100	CAL.	2,000	16,800	33,600,000
0019	ZZ-5101	CAL.	2,500	13,800	34,500,000
0020	ZZ-8245	CAL.	15,000	10,800	162,000,000
0021	MM-1111	CAL.	1,500	29,800	44,700,000
0022	MM-999	CAL.	5,000	15,600	78,000,000
0023	MM-727	CAL.	10,000	59,800	598,000,000
				TOTAL AMOUNT	7,321,137,000

"P": Specifying listing range

When the contents of a group is to be seen partially, the key "P" must be used. Depression of the key causes the computer to ask where to begin (FROM) and end (TO). They have to be responded by a 4-digit code. What is to be cared for is that the 1st digit of the code be corresponding with the particular group code. If it is the Group A, the 1st digit of the code must be "0."

Photo 17 exhibits that the inventory list is requested for the merchandize number starting from "0003" to "0005" of the Group A. So, the merchandize numbers starting from "0003" and ending "0005" are listed, out of the table shown in the previous page. The cancel key "→" is also effective at this job.

Photo 17

INVENTORY LIST

<GROUP A> SAMPLE CATALOGUE 10.JUL. 80

CODE	MODEL	NAME	PIECES	UNIT PRICE	AMOUNT
0003	AA-80FDK	E-FLOPPY	1,250	325,000	406,250,000
0004	AA-80P3	PRINTER	25,000	168,000	4,200,000,000
0005	AA-80I/O	I/O BOX	32,000	29,800	953,600,000
				TOTAL AMOUNT	5,559,850,000

JOB 6: Code-model reference

In Job 2 and Job 4, it had been more effective to use code reference method in reading a desired merchandize item onto the screen. For this purpose, Job 6 is employed. The Job 6 will provide a reference table composed of the model name and code. So, hit the key "6" at the JOB LIST.

When this job is selected, the display indicates the information as in Photo 18. What has to be done at this stage is to designate the desired group code.

As soon as the group is selected, all model names and their merchandize codes registered in the disk are printed out on the printer.

With Photo 18 the Group A is selected. Its example is exhibited below.

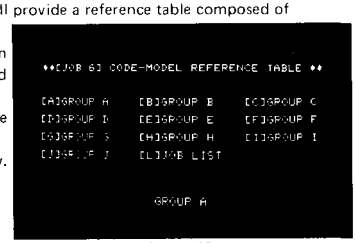


Photo 18

CODE-MODEL REFERENCE TABLE

<GROUP A> SAMPLE CATALOGUE 10.JUL. 80

CODE	MODEL	CODE	MODEL	CODE	MODEL	CODE	MODEL
0001	AA-80K	0002	AA-80FD	0003	AA-80FDK	0004	AA-80P3
0005	AA-80I/O	0006	AA-80FIO	0007	AA-80FMD	0008	AA-80F15
0009	AA80T20	0010	AA-80TU	0011	AA-80FBD	0012	AA-80T
0013	XX-730	0014	XX-810	0015	XX-205	0016	XX-740
0017	XX-725	0018	ZZ-5100	0019	ZZ-5101	0020	ZZ-8245
0021	MM-1111	0022	MM-999	0023	MM-727		

JOB 7: Are there any unnecessary files?

Are there any files that are no more necessary? Job 7 is assigned for the deletion of a file. So, hit the key "7" at the JOB LIST.

You can designate the group to be deleted in the state of Photo 19. If Group C is to be deleted, just depress the key "C."

Photo 19 shows when deletion is carried out for the Group H. In this case, it is essential to depress the key with care, otherwise you may be destroying necessary file contents, wasting all the labour you have exerted in Job 1.

It is recommended to cancel such a file name that has been registered in Job 3 without the contents. Job 7 will do this for you.

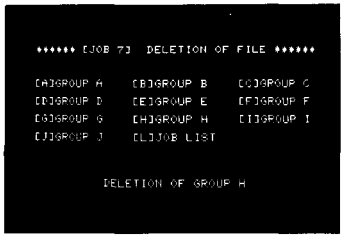


Photo 19

JOB 8: Good night for the job

With this job you will be closing the day's work. Hit the key "8" at the JOB LIST.

With this job the program execution is terminated, but the data necessary for tomorrow's work is kept stored in the disk. Nothing extra should be done by you, just good night to both the computer and you.

It might sound funny to talk of the first thing at the last stage, but remember that the disk used at the termination of the Job 8 has to be used again at the beginning of the next program execution, except the very first time the program is run for the first time.

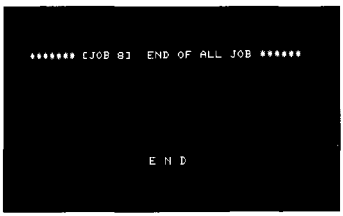


Photo 20

When DIR is executed, it becomes apparent that there is an extraordinary file named "INITIAL SET" other than you have generated (Photo 21). It was made by the computer itself, containing the data showing the registration status of each file.

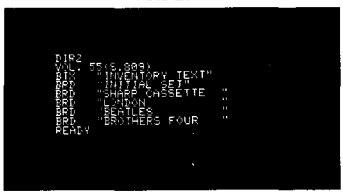


Photo 21

***About the item deletion in the Job 2**

Assume now that the merchandize items were registered from "0001" to "0010" for the Group A first time, but deletion was made to the merchandize code "0003" and "0007" at the Job 2. In such a case that additional merchandize is to be registered in that file next time, the merchandize code "0003" will be assigned to the merchandize first applied, and "0007" to the next one, then numbers after "0011" will be assigned to successive merchandize items. In other words, any additional data will replace the deleted location in the first place before assigning a new code.

Error message

When anything illogical is found by the computer, attention is called for by means of the buzzer alarm or outputting an error message on the CRT screen. Typical error and its warning method are shown below.

(1) Buzzer only

Specific key operation	○ Depression of key other than specified.
Entry of the model name or merchandize name (during registration or correction)	○ Exceeded 8 digits.
Entry of number of stock (incoming and outcoming)	○ Exeeded 12 digits. ○ Existence of character other than numerical figure. ○ Number of outgoing merchandizes exceeded the number of stock.
Entry of unit price	○ Exceeded 12 digits. ○ Existence of character other than numerical figure.
Entry of the total amount	○ Data entry or arithmetical result has exceeded 16 digits. ○ Existence of character other than numerical figure.
Entry of file name	○ Already existing file name is put into use.

(2) Display of an error message on the CRT screen together with buzzer alarm

○ DON'T USE!! ..... The file name used for another file is used again.
○ NO FILE!! ..... The specified group is not registered as a file yet.
○ NO DATA!! ..... No data is available.
○ NO MODEL!! ..... The specified model name is not in existence.
○ CODE OF OTHER FILE!! ..... The code number used is for another group.
○ GROUP A IS FILL!! ..... Choose another group as the Group A is full.
ANOTHER GROUP, PLEASE!!

(3) Special error message

When any of the following error is indicated, depress any opted key after performing appropriate disposition. This will cause the program to restore.

○ UNCONNECTED PRINTER!! ..... The printer is not unconnected.
○ TROUBLED PRINTER!! ..... Trouble with the printer
○ PAPER EMPTY!! ..... Printer paper had run out.
○ UNCONNECTED FLOPPY!! ..... The diskette drive unit is isolated from the system.
○ UNINITIALIZED DISKETTE!! ..... Diskette not initialized.
○ CAN'T DISPOSE!! ..... The computer has encountered an illogical condition and
JOB LIST IS EXECUTED. return to the JOB LIST.
(No guarantee is provided for this case.)

Main variables used in program.

Main variables which will be necessary for the interpretation of the program are shown below. They may be changed according to your need.

1) Variables common to each job

Variable	Description
F$ ( )	File name to each group. "F$(1)" stands for a file of Group A.
IM$ ( )	Items in a merchandize code such as CODE, MODEL, PIECE, etc.
D$ ( )	Data of each item, corresponding with IM$ ( ).
K ( )	Number of registrations in a group.
G$, G	Designation of a group. G$ = "A" designates the Group A, G = 1 the Group A.
CP	Printing mode. CP = 1 stands for printer output.
CZ	Continuous retrieval mode. CZ = 1 stands for continuous retrieval.
PR	To be used for print processing at the time of concelling.
JT ( )	Maximum number of characters in each item.

2) Variables for the arithmetical routine

Variable	Description
D$ (3)	String of operand.
D$ (4)	String of operator.
D$ (5)	String of arithmetical result.
ML	Operational mode: ML = 0 for multiplication, ML = 1 for addition, ML = 2 for subtraction.
W, WW	"WW" notation numerical expression. WW = 10^W
M, N	Number of digits of two figures in decimal notation.
MM, NN	Corresponds with the number of digits in "WW" numerical notation.
A ( )	Stores figures in each digit when the operand is expressed in the WW numerical notation.
B ( )	Stores figures in each digit when the operator is expressed in the WW numerical notation.
C ( )	Stores the arithmetical result of numbers in each digit which is expressed in WW numerical notation.
C$ ( )	String of C ( ) after string conversion. C$ ( ) = STR$ (C ( ))

3) Basic data configuration of the disk (random data file by XOPEN)

Data registration of one merchandize item

	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
5N-4	Code No. | Model name | Not used
5N-3	Merchandize name | Not used
5N-2	Number of stock | Not used
5N-1	Unit price | Not used
5N	Total amount

Deletion of merchandize item

	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
5N-4	✳ | Data to be deleted | Not used

✳: Means deletion data

Deletion data = (lower 3 digits of the deletion item following this merchandize item) + (two spaces) + (lower 3 digits of the deletion item preceding this merchandize item)

To calculate extraordinary large number

Computers can compute large number of operand with large number of operator in a simple manner, but its result is rounded with BASIC when effective figure exceeds beyond 8 digits.

```
? 1 2 3 4 5 6 7 8 9 * 9 8 7 6 5 4 3 2 1
  1 2 1 9 3 2 6 3 E + 1 8
```

This causes trouble so far as stock inventory is concerned. Therefore, a special arithmetical routine is provided for allowing addition, substraction, and multiplication up to 16 digits. So, let's make a quick explanation how the exact result is obtained up to 16 digits.

Multiplication by thousand notation system

If there is a possibility of causing overflow in a single computational procedure in the case of multiplication, the operator and the operand are disassembled into several numbers so as to compute each operation separately. Considering the thousand notation system, a number like 12345678 can be expressed as follows:

$$(12345678)_{10} = 12 \times (10^3)^2 + 345 \times (10^3)^1 + 678 \times (10^3)^0$$
$$= (a_0 a_1 a_2)_{1000}$$
$$a_0 = (12)_{10},\ a_1 = (345)_{10},\ a_2 = (678)_{10}$$

In this manner it can be expressed as a 3 digits of the thousand notation. And, each digit is arranged in the thousand notation (0~999) can equal with that of the decimal notation when they are arranged in order. When two numbers ($A = (a_0 a_1 a_2)_{1000}$, and $B = (b_0 b_1 b_2)_{1000}$) are multiplied in the form of thousand notation by utilizing this nature, the following development can be seen:

$$\begin{aligned} A \times B = {} & (a_0 a_1 a_2)_{1000} \times (b_0 b_1 b_2)_{1000} \\ = {} & (a_0 \times b_0) \times (10^3)^4 \\ + {} & (a_0 \times b_1 + a_1 \times b_0) \times (10^3)^3 \\ + {} & (a_0 \times b_2 + a_1 \times b_1 + a_2 \times b_0) \times (10^3)^2 \\ + {} & (a_1 \times b_2 + a_2 \times b_1) \times (10^3)^1 \\ + {} & (a_2 \times b_2) \times (10^3)^0 \\ = {} & C_0 \times (10^3)^4 + C_1 \times (10^3)^3 + C_2 \times (10^3)^2 + C_3 \times (10^3)^1 + C_4 \times (10^3)^0 \end{aligned}$$

In this way of multiplication, the number each digit in the thousand notation can be obtained by the multiplication of two 3-digit numbers and the summary of that product. As it is the multiplication in decimal system by 3 digits each, its result never exeeds 8 digits, so as to effectively utilize the computational capacity of the computer. Since the summary of the products can not exceeds 8 digits, the sum of digits of two numbers in the decimal notation must be within 100.

Here, there is a need for more processing. Adjustment has to be done since the number of each digit must be within 999 in the thousand notation system. For instance, C_4 in the above expression is to be $(56088)_{10}$, it must be $C_4 = 088$.

$$C_4 = \underbrace{(56)_{10}} \times (10^3)^1 + \underbrace{(088)_{10}} \times (10^3)^0 \quad \cdots \text{ Carry over treatment}$$
$$C_3 = C_3 + 56 \qquad C_4 = 008$$

This procedure is applied to each digit. That is, the number in each digit is divided by 1000 and its quotient is added to the number in higher order and its remainder is taken for the number in its digit. It should never be expressed as $C_4 = 88$, because a number in each order must be one of 3 digits, except the number in highest order.

e.x. 0 → 000 5 → 005, 58 → 058

To unify the divided number, they must be handled by string treatment. You know, a string does not overflow unless it exceeds 256 digits.

Million notation system is used for addition and subtraction

Although the thousand notation system is used for the multiplication, 10^6 notation system is here used (it may be 10^7 notation system). Basic concept is the same as for multiplication. Two numbers are converted into 10^6 notation to be divided into 6 digits each of numbers, so as to compute by each digit. For instance, when A = $(9999999999)_{10}$ is to be subtracted by B = $(3333991111)_{10}$, the following process

$$\begin{aligned} A - B &= [(9999)_{10} \times (10^6)^1 + (999999)_{10} \times (10^6)^0] - [(3333)_{10} \times (10^6)^1 + (991111)_{10} \times (10^6)^0] \\ &= [(9999)_{10} - (3333)_{10}] \times (10^6)^1 + [(999999)_{10} - (991111)_{10}] \times (10^6)^0 \\ &= (6666)_{10} \times (10^6)^1 + (008888) \times (10^6)^0 \\ &= 6666008888 \end{aligned}$$

Addition and subtraction in 10^6 notation system never exceeds 8 digits of effective number and thus allows you to utilize the arithmetic capacity of the computer. Next, as it is handled with the carry over treatment for the multiplication, it will execute borrowing for subtraction and carrying for addition. It must be adjusted to a number of 6 digits, except the number in highest order.

In the actual program

The statement numbers 9000 through 9890 are the subroutine for the three operations. Using this subroutine may allow you to perform many applications. Before the execution of this subroutine, it will be necessary to substitute two operating numbers into D$ (3) and D$ (4) as numerical strings and command ML = 0 is used for multiplication, ML = 1 for addition and ML = 2 for subtraction. When GOSUB 9000, RETURN is executed, its result will be registered in D$ (5) as numerical string.

You will be able to grasp the flow of the program if you look at the flowchart given in page 147. Array variables, A ( ), B ( ), C ( ), C$ ( ), are used as work variables in the operation. To each element of the array variables, A ( ) and B ( ), those which the numerical string of D$ (3) and D$ (4) are divided into 3 or 6 digit partition from the right are substituted after numerical conversion (VAL). That is, they will be handled as the number in each digit in the 10^3 notation system or 10^6 notation system. To each element of the array variable C ( ), arithmetical result in a 10^3 notation system or 10^6 notation system is substituted. To the string array variable C$ ( ), those which each element of the variable C ( ) are string converted (STR$) are substituted.

With the flowchart description and the explanation this time, you will nod and say "Naturally." As sizes of array of A ( ), B ( ), C ( ), and C$ ( ), are assigned to 6, 6, 12, and 12 in this program, it allows you to perform multiplication of up to 18 digits in the decimal system and operation up to 36 digits maximum can be performed for addition and subtraction. However, the arithmetical result is limited up to a maximum of 16 digits for the inventory management program due to limits given to listing format.

The statement numbers 9040 and 9470 correspond with it. When these two statement are deleted, arithmetic operation appropriate to the above array variables can be executed. In case larger capacity is to be required, you can expand the size of array. But in the cause of multiplication care must be exercised to determine the size of C ( ) and C$ ( ) on the basis of the sum of A ( ) and B ( ).

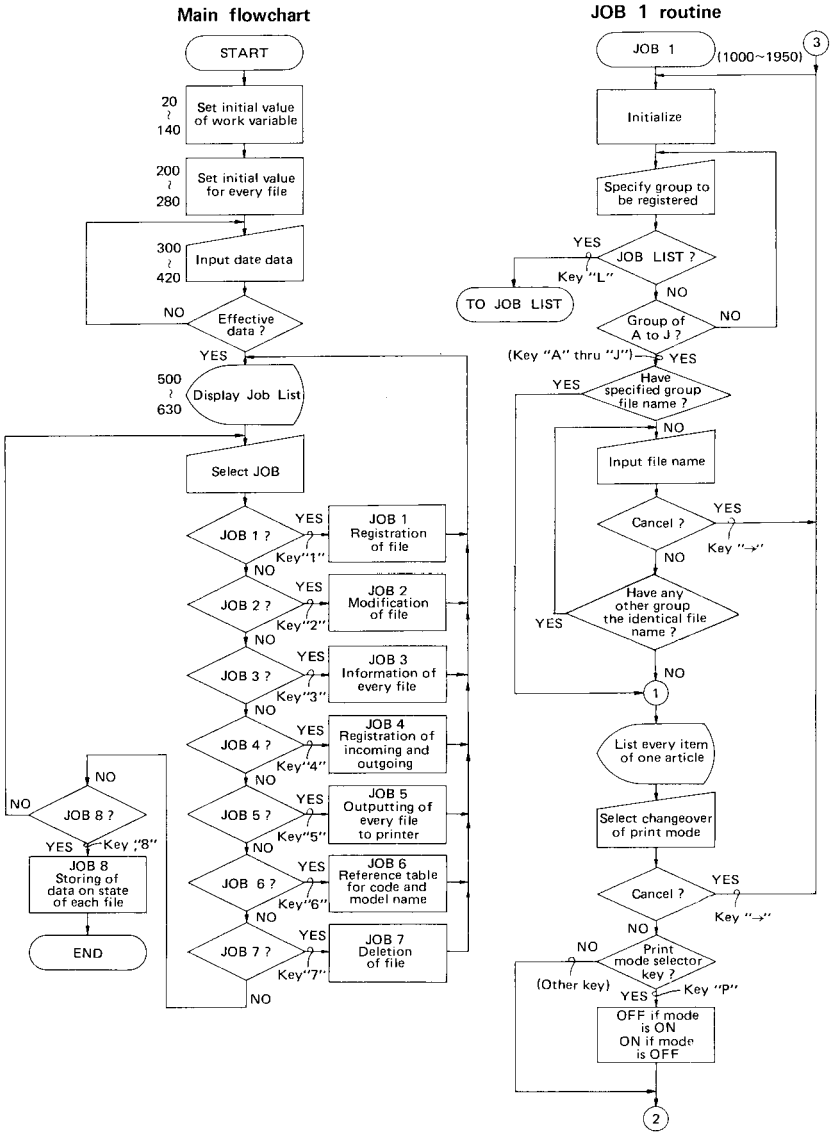
Although negative numbers are not handled in this program, addition of the following processing can then allow such function, if necessary.

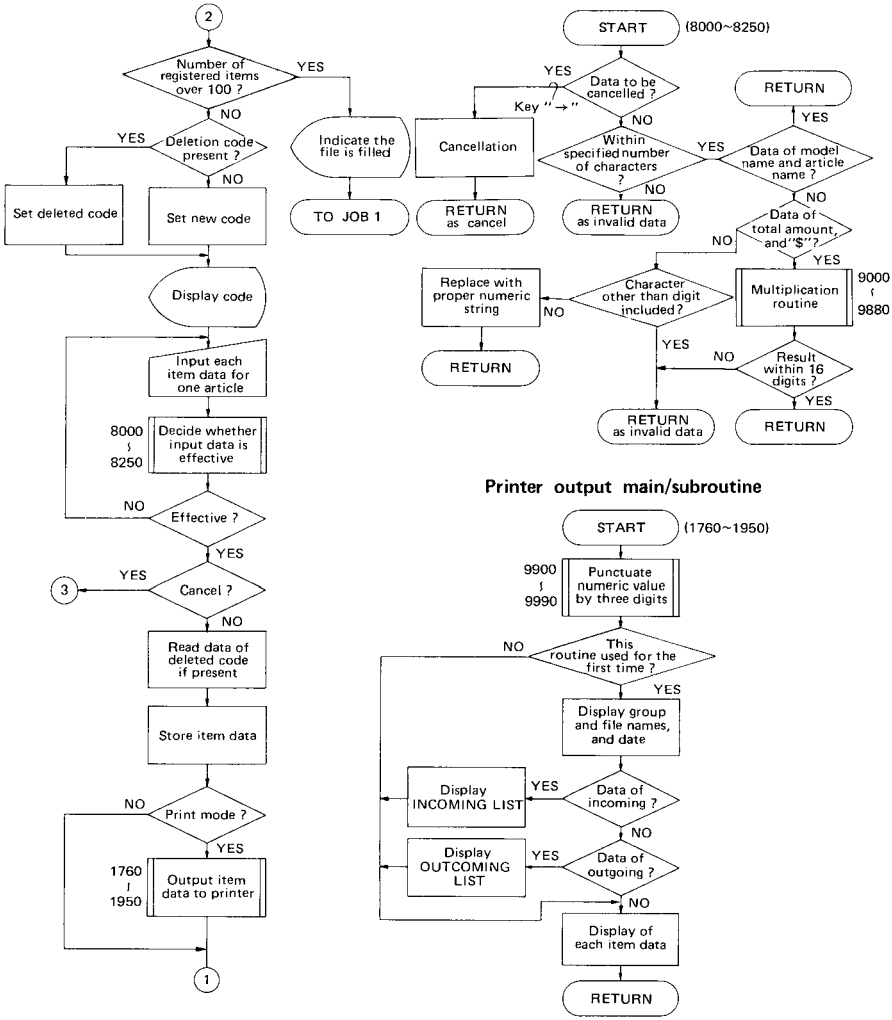
(1) In the case of multiplication, operation by opposing symbols must be handled as "D$ (5) = "−" + D$ (5)" and RETURN.

(2) In the case of addition/subtraction, it must be set to the adding mode (ML = 1) before calling the program. If operation involves calculation of opposing symbols, it should set to the subtraction mode (ML = 2), while deciding which is the larger.
If both are negative and in the adding mode, it should set to "D$ (5) = "−" + D$ (5)."
If the operand D$ (3) with opposing symbol is smaller than the operator D$ (4), it should be set to "D$ (5) = "−" + D$ (5)."

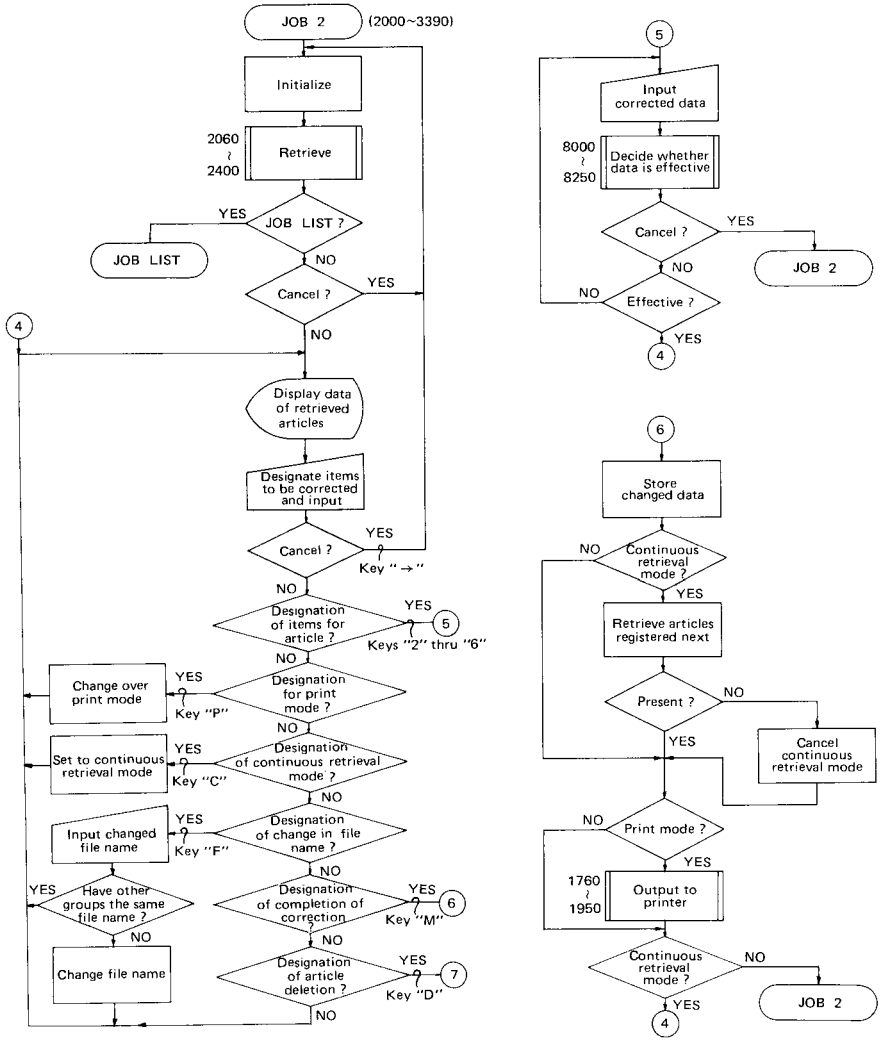
What should be done in a case of division? Combination of addition and subtraction will allow you to easily perform division of a large number.

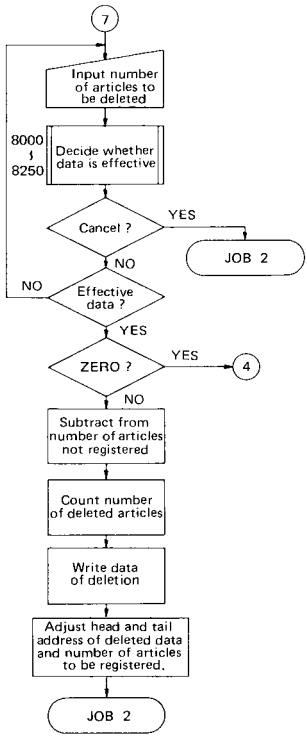
Inventory control program flowchart



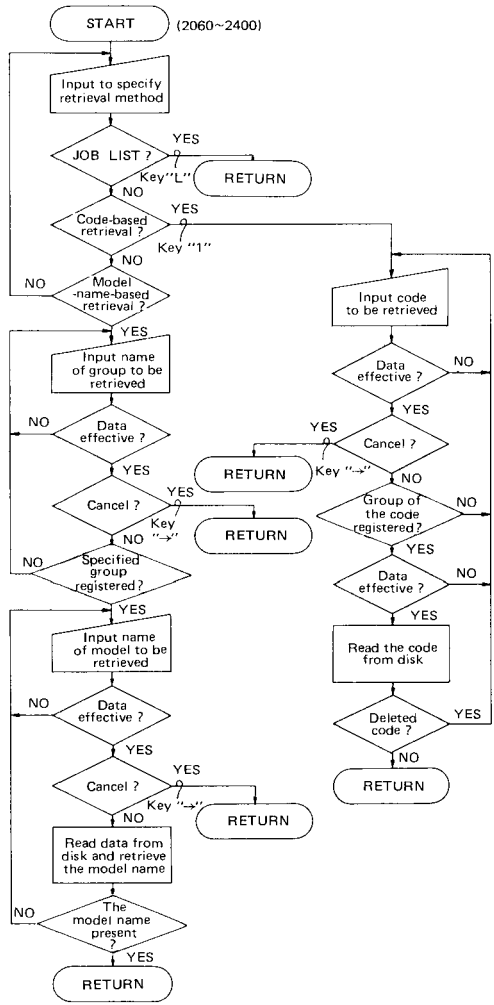

Input data decision subroutine
START
(8000~8250)
Data to be cancelled ?
YES
Key "→"
NO
Cancellation
RETURN as cancel
Within specified number of characters ?
YES
NO
RETURN as invalid data
Data of model name and article name ?
YES
RETURN
NO
Data of total amount, and "$" ?
NO
YES
Multiplication routine
9000 ~ 9880
Character other than digit included ?
YES
NO
Replace with proper numeric string
RETURN
Result within 16 digits ?
NO
YES
RETURN as invalid data
RETURN
2
Number of registered items over 100 ?
YES
NO
Indicate the file is filled
TO JOB 1
Deletion code present ?
YES
NO
Set deleted code
Set new code
Display code
Input each item data for one article
8000 ~ 8250
Decide whether input data is effective
Effective ?
NO
YES
3
Cancel ?
YES
NO
Read data of deleted code if present
Store item data
Print mode ?
NO
YES
1760 ~ 1950
Output item data to printer
1
Printer output main/subroutine
START
(1760~1950)
9900 ~ 9990
Punctuate numeric value by three digits
This routine used for the first time ?
NO
YES
Display group and file names, and date
Data of incoming ?
YES
Display INCOMING LIST
NO
Data of outgoing ?
YES
Display OUTCOMING LIST
NO
Display of each item data
RETURN

JOB 2 routine

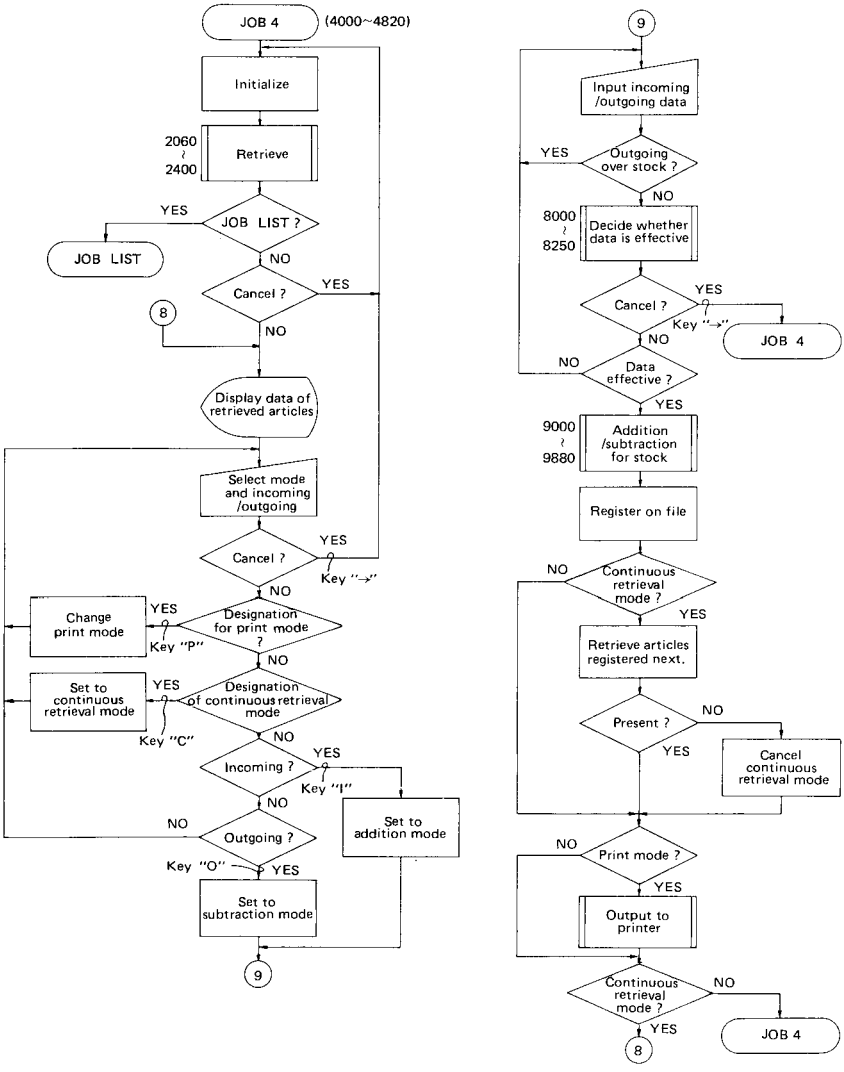




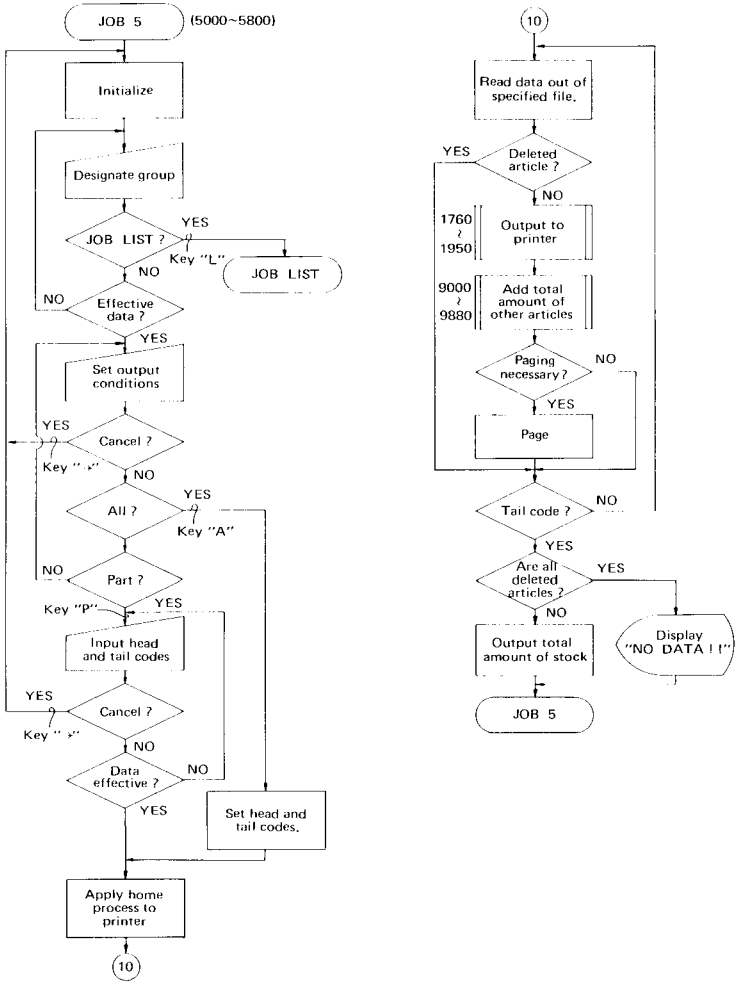
Retrieval subroutine



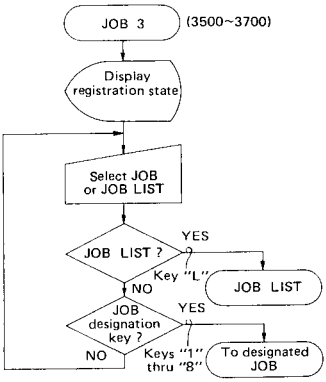
JOB 4 routine



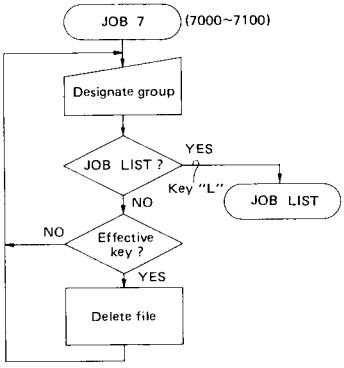
JOB 5 routine



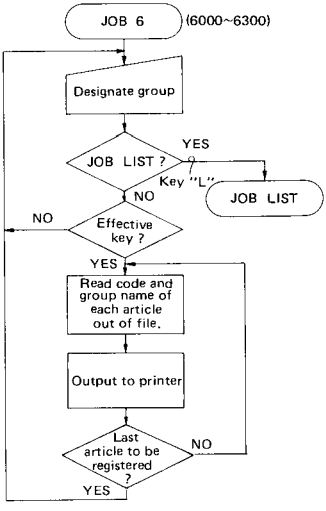
JOB 3 routine



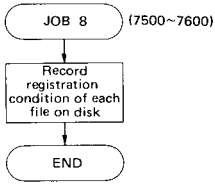
JOB 7 routine



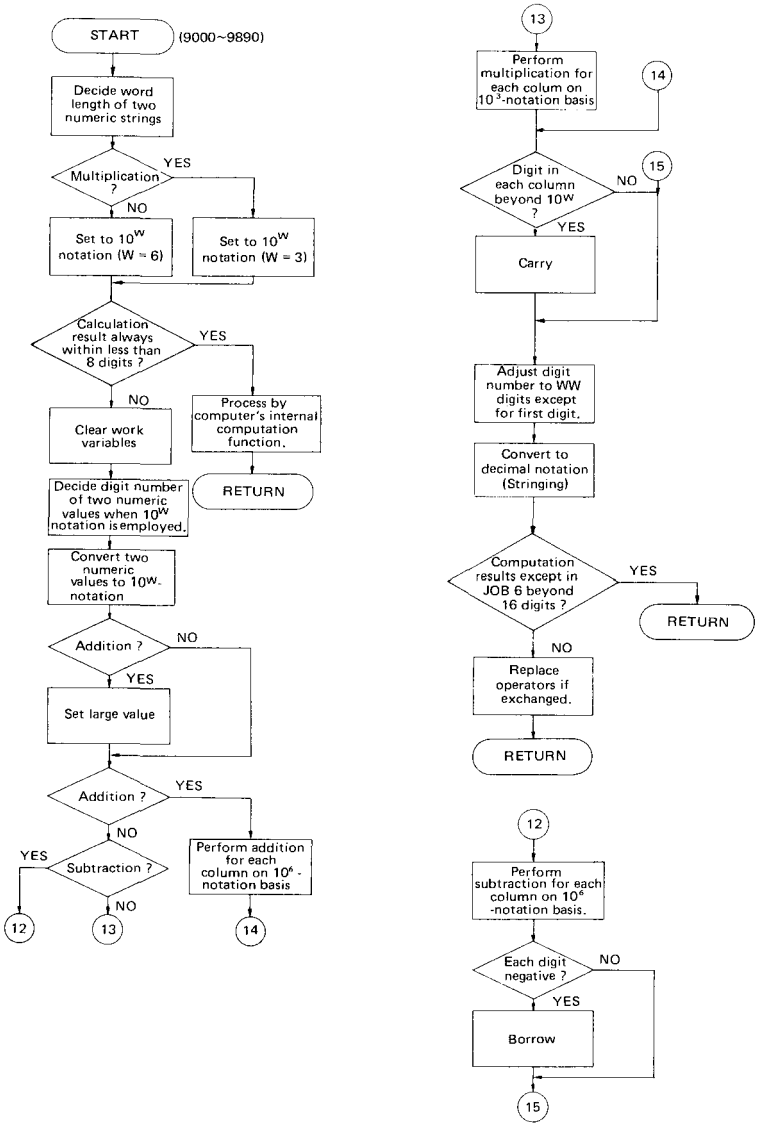
JOB 6 routine



JOB 8 routine



Operational subroutine



```
10 REM..................... INVENTORY CONTROL APPLICATION ....................
15 REM
20 DIM F$(10),IM$(6),D$(6),ZB(10),ZF(10),ZE(10),K(10)
30 DIM A(5),B(5),C(11),C$(11),TB(2),DM$(6),IO$(2),JT(4),DT$(11)
40 ON ERROR GOTO 8500
50 TEMPO 6
60 IM$(1)="CODE":IM$(2)="MODEL":IM$(3)="NAME":IM$(4)="PIECES":IM$(5)="PRICE"
70 IM$(6)="AMOUNT":ZZ$="00000":ST$="**********":I1=3:I2=5
80 MS$="■■B0R■■BR■■B":PK$="                "
82 DT$(0)="JAN":DT$(1)="FEB":DT$(2)="MAR":DT$(3)="APR":DT$(4)="MAY"
84 DT$(5)="JUN":DT$(6)="JUL":DT$(7)="AUG":DT$(8)="SEP":DT$(9)="OCT"
86 DT$(10)="NOV":DT$(11)="DEC"
90 JT(0)=8:JT(1)=8:JT(2)=12:JT(3)=12:JT(4)=16
100 PRINT"■":PRINT:FOR I=0 TO 22
110 IF (I>9)*(I<13) THEN PRINT:PRINT TAB(17);"SHARP":PRINT:I=13:GOTO 140
120 II=10-I:IL=I:IF I>12 THEN II=I-12:IL=22-I
130 PRINT TAB(IL);LEFT$(ST$,II);" INVENTORY CONTROL ";LEFT$(ST$,II)
140 NEXT
199 REM************************************************ [INITIAL SET OF FILES]
200 XOPEN #2,"INITIAL SET"
210 FOR I=1 TO 10:H=3*I-2
220 INPUT #2(H),F$(I),K$,ZF$
230 K(I)=VAL(LEFT$(K$,3)):ZB(I)=VAL(MID$(K$,4,5))
240 ZF(I)=VAL(LEFT$(ZF$,3)):ZE(I)=VAL(MID$(ZF$,4,5))
250 IF ZE(I)=0 THEN ZE(I)=1
260 IF F$(I)="" GOTO 280
270 IF ASC(F$(I))=32 THEN F$(I)=""
280 NEXT:CLOSE #2
299 REM*************************************************************** [DATING]
300 PRINT"■********  INVENTORY CONTROL  **********■■■■"
310 PRINT"WHAT'S THE DATE TODAY?■"
320 INPUT "[YEAR]   ";A$
330 IF VAL(A$)=0 THEN PRINT"■";:MUSIC MS$:GOTO 320
340 INPUT "[MONTH]  ";B$
350 B=VAL(LEFT$(B$,2)):IF (B<1)+(B>12) THEN PRINT"■";:MUSIC MS$:GOTO 340
360 INPUT "[DAY]    ";C$
370 C=VAL(LEFT$(C$,2)):IF (C<1)+(C>31) GOTO 410
380 IF (B=2)*(C>29) GOTO 410
390 IF (B<>4)*(B<>6)*(B<>9)*(B<>11) GOTO 420
400 IF C<>31 GOTO 420
410 PRINT"■";:MUSIC MS$:GOTO 360
420 DT$=STR$(C)+"."+DT$(B-1)+", "+A$
499 REM************************************************************** [JOB LIST]
500 PRINT"■**********    JOB LIST     **********■■"
510 PRINT"■[1]  REGISTRATION OF MASTER FILE"
520 PRINT"■[2]  CORRECTION OF MASTER FILE"
530 PRINT"■[3]  INFORMATION OF EACH FILE"
540 PRINT"■[4]  INCOMING OR OUTCOMING STOCK"
550 PRINT"■[5]  FILE COPY"
560 PRINT"■[6]  CODE-MODEL REFERENCE TABLE"
570 PRINT"■[7]  DELETION OF FILE"
580 PRINT"■[8]  END OF ALL JOB"
590 CURSOR 15,22:PRINT"JOB ";
600 GET G$:PRINT"_■ ■ ■";:IF G$="" GOTO 600
610 G=ASC(G$)-48
620 IF (G<1)+(G>8) THEN MUSIC MS$:GOTO 600
630 ON G GOTO 1000,2000,3500,4000,5000,6000,7000,7500
999 REM************************************************************** [JOB 1]
1000 CP=0:PG=0:JP=1:PR=0
1005 GOSUB 1510:GOSUB 1550:CURSOR 14,13:PRINT"GROUP? ";:GOSUB 1650
1007 PRINT G$
1010 IF LEN(F$(G))<>0 GOTO 1045
1020 PRINT"■■INPUT A FILE NAME OF GROUP ";G$;".■"
1030 INPUT "FILE NAME? ";F$(G):IF ASC(F$(G))=198 THEN F$(G)="":GOTO 1000
1040 F$(G)=LEFT$(F$(G)+PK$,16):FOR I=1 TO 10:IF G=I GOTO 1043
1042 IF F$(G)=F$(I) GOTO 1044
```

```
1043 NEXT:GOTO 1045
1044 PRINT"DON'T USE !!":MUSIC MS$:PRINT"■■■":GOTO 1030
1045 IF K(G)<100 GOTO 1050
1047 GOSUB 1500:MUSIC MS$:CURSOR 10,10:PRINT"GROUP ";G$;" IS FULL !!■■"
1048 PRINT TAB(10);"GO TO NEXT GROUP !!":GOSUB 1640:GOSUB 1920:GOTO 1000
1050 GOSUB 1510:GOSUB 1600
1060 PRINT"|  GROUP ";G$;TAB(15);F$(G);TAB(33);
1065 IF CP=0 THEN PRINT"P[X] |":GOTO1070
1068 PRINT"P[P] |"
1070 GOSUB 1610:FOR I=1 TO 6
1080 PRINT"|";I;" |  ";IM$(I);TAB(13);"|";TAB(38);"|"
1085 IF I<>6 GOSUB 1620
1090 NEXT:GOSUB 1630
1110 PRINT"■[P]PRINT MODE            [→]CANCEL"
1120 PRINT"■[DATA=→]CANCEL DATA"
1130 PRINT"■[DATA6=$]DATA4*DATA5"
1140 PRINT"■■■";
1150 GET P$:IF P$="" THEN PRINT"_■ ■ ■";:GOTO 1150
1160 IF P$="→" THEN GOSUB 1920:GOTO 1000
1170 CURSOR 35,4
1180 IF (P$="P")*(CP=1) THEN CP=0:PRINT"X":GOTO 1200
1190 IF (P$="P")*(CP=0) THEN CP=1:PRINT"P":GOTO 1200
1195 PRINT""
1200 IF ZB(G)=0 THEN CF=1:Z=ZE(G):ZE(G)=ZE(G)+1:GOTO 1220
1210 CF=0:Z=ZB(G)
1220 Z$=STR$(Z):GOSUB 1750
1230 IF LEN(Z$)=2 THEN D$(0)=STR$(G-1)+"0"+Z$:GOTO 1260
1240 IF LEN(Z$)=1 THEN D$(0)=STR$(G-1)+"00"+Z$:GOTO 1260
1250 D$(0)=STR$(G-1)+"100"
1260 PRINT" ";D$(0):GOSUB 1750
1270 FOR I=1 TO 5
1280 INPUT" ";D$(I)
1290 GOSUB 8000
1300 ON B GOTO 1000,1330,1320
1310 PRINT"■■■":GOSUB 1750:MUSIC MS$:GOTO 1280
1320 PRINT"■■■":GOSUB 1750:PRINT" ";D$(I)
1330 GOSUB 1750:NEXT
1335 IF ZB(G)=ZF(G) THEN ZB(G)=0:ZF(G)=0
1340 V=VAL(MID$(D$(0),2,3)):D$=D$(0)+D$(1)
1390 XOPEN #1,F$(G)
1400 IF ZB(G)=ZF(G) GOTO 1420
1410 INPUT #1(5*V-4),DM$
1420 PRINT #1(5*V-4),D$,D$(2),D$(3),D$(4),D$(5):CLOSE #1
1430 K(G)=K(G)+1
1440 IF CP=1 GOSUB 1760
1470 IF ZB(G)=ZF(G) GOTO 1045
1480 ZB(G)=VAL(MID$(DM$,2,3))
1490 GOTO 1045
1500 REM*********************************************************** [SUBROUTINE]
1510 PRINT"■**** [JOB 1]  REGISTRATION OF FILE ****■■":RETURN
1550 PRINT"■[A]GROUP A    [B]GROUP B    [C]GROUP C"
1560 PRINT"■[D]GROUP D    [E]GROUP E    [F]GROUP F"
1570 PRINT"■[G]GROUP G    [H]GROUP H    [I]GROUP I"
1580 PRINT"■[J]GROUP J    [L]JOB LIST":RETURN
1600 PRINT"┌──┬──────────────────────────────────┐":RETURN
1610 PRINT"├──┼──────────────────────────────────┤":RETURN
1620 PRINT"├──┼──────────────────────────────────┤":RETURN
1630 PRINT"└──┴──────────────────────────────────┘":RETURN
1640 FOR T=1 TO 2000:NEXT:RETURN
1650 G$="":GET G$:PRINT"_■ ■ ■";
1660 IF G$="" GOTO 1650
1670 G=ASC(G$)-64
1680 IF ((G<1)+(G>10))*(G<>12) GOTO 1710
1690 IF G=12 THEN GOSUB 1920:GOTO 500
1700 RETURN
1710 MUSIC MS$:GOTO 1650
```

```
1750 PRINT"■";TAB(14);:RETURN
1759 REM************************************************ [MAIN PRINT ROUTINE]
1760 PR=1:GOSUB 9900
1765 IF PG=G GOTO 1820
1770 PRINT#P:PRINT#P
1780 PRINT#P"■■(GROUP ";G#;")  ";F$(G);TAB(28);DT$
1790 PRINT#P"■"
1800 IF ML=1 THEN PRINT#P"INCOMING STOCK LIST":PRINT#P:PRINT#P"■":GOTO 1820
1810 IF ML=2 THEN PRINT#P"OUTCOMING STOCK LIST":PRINT#P:PRINT#P"■"
1820 IF PG=G GOSUB 1870:RETURN
1825 GOSUB 1830:GOSUB 1870:RETURN
1830 PRINT#P"■┌──────┬──────┬──────┬──────┬──────┐";
1840 PRINT#P"└──────┴──────┴──────┴──────┴──────┘"
1850 PRINT#P IM$(1);"│   ";IM$(2);"│   ";IM$(3);"  │      ";IM$(4);
1860 PRINT#P"    │    ";"UNIT PRICE";"  │         ";IM$(6):RETURN
1870 PRINT#P"■├──────┼──────┼──────┼──────┼──────┤";
1880 PRINT#P"└──────┴──────┴──────┴──────┴──────┘"
1890 PRINT#P D$(0);"│ ";D$(1);"│ ";D$(2);"│ ";TAB(41-TB(0));D$(3);"│";
1900 PRINT#P TAB(58-TB(1));D$(4);"│";TAB(80-TB(2));D$(5)
1910 PG=G:RETURN
1920 IF (CP=0)+(PR=0) THEN RETURN
1930 PRINT#P"└──────┴──────┴──────┴──────┴──────┘";
1940 PRINT#P"──────────────────────"
1950 RETURN
1999 REM************************************************************** [JOB 2]
2000 PG=0:JP=2:CZ=0:CP=0:PR=0:XV=0
2010 GOSUB 2050:GOSUB 2060
2020 ON B1 GOTO 500,2410
2030 GOSUB 1920:GOTO 2000
2050 PRINT"■**** [JOB 2]   CORRECTION OF FILE  ****■■":RETURN
2059 REM************************************************************ [RETRIEVAL]
2060 PRINT"■■[L]JOB LIST"
2070 PRINT"■[1]BY CODE     [2]BY MODEL     [ ]"
2080 PRINT"■";TAB(32);
2090 GET G$:PRINT"_■ ■ ■";:IF G$="" GOTO 2090
2100 IF G$="L" THEN B1=1:RETURN
2110 IF G$="1" THEN PRINT"1":GOTO 2140
2120 IF G$="2" THEN PRINT"2":GOTO 2270
2130 MUSIC MS$:GOTO 2090
2139 REM************************************************************** [BY CODE]
2140 PRINT"■■■■■■■CODE  [????]"
2150 PRINT"■■■■■■■";:INPUT"[";D$(0):D$(0)=LEFT$(D$(0),4)
2160 G=ASC(D$(0))-47
2170 IF ((G>0)*(G<11))+(G=151) GOTO 2175
2173 MUSIC MS$:GOTO 2150
2175 IF G=151 THEN B1=0:RETURN
2180 IF (XV=0)*(F$(G)="") THEN PRINT"■NO FILE !!":PRINT"■■■":GOTO 2173
2185 G$=CHR$(G+64)
2190 V=VAL(RIGHT$(D$(0),3))
2192 IF (V=0)*(XV=2) THEN PRINT"■";SPC(25):PRINT"■■";TAB(15);:GOTO 2173
2194 IF V=0 THEN PRINT"■";SPC(25);"■■":GOTO 2173
2195 IF XV<>0 THEN B1=1:RETURN
2200 IF (V<1)+(V>ZE(G)-1)) GOTO 2250
2210 XOPEN #1,F$(G)
2220 INPUT #1(5*V-4),D$(1),D$(2),D$(3),D$(4),D$(5)
2230 CLOSE #1
2240 IF ASC(D$(1))<>42 GOTO 2255
2250 PRINT"■NO DATA !!":PRINT"■■■":GOTO 2173
2255 D$(0)=LEFT$(D$(1),4):D$(1)=MID$(D$(1),5,8):D$(2)=LEFT$(D$(2),8)
2260 GOSUB 3200:B1=2:RETURN
2269 REM************************************************************** [BY MODEL]
2270 PRINT"■■■■■■■GROUP  [ ]"
2275 PRINT"■■■■■■■■■"
2280 GET G$:PRINT"_■ ■ ■";
2290 IF G$="" GOTO 2280
2300 G=ASC(G$)-64
```

```
2310 IF ((G>0)*(G<11))+(G=134) GOTO 2312
2311 MUSIC MS$:GOTO 2280
2312 IF G=134 THEN B1=0:RETURN
2315 IF (K(G)=0)+(F$(G)="") THEN PRINT"■■NO FILE,NO DATA■■":MUSICMS$:GOTO 2275
2318 PRINT G#
2319 PRINT "■":SPC(30):PRINT"■"
2320 PRINT"MODEL  [        ]"
2330 PRINT"■■■■■■■■"::INPUT"[":D$(1)
2333 IF LEFT$(D$(1),1)="→" THEN B1=0:RETURN
2334 IF D$(1)="]" THEN MUSIC MS$:GOTO 2330
2335 D$(1)=LEFT$(D$(1),8):FOR J=8 TO 1 STEP -1
2337 R$=MID$(D$(1),J,1):IF (R$=" ")+(R$="]") THEN NEXT:MUSIC MS$:GOTO 2330
2340 D$(1)=LEFT$(D$(1),J)+LEFT$(PK$,8-J)
2350 XOPEN #1,F$(G)
2360 FOR I=1 TO ZE(G)-1
2370 INPUT #1(5*I-4),D$(0),D$(2),D$(3),D$(4),D$(5)
2373 R$=LEFT$(D$(0),1):IF R$="→" THEN NEXT:GOTO 2390
2380 IF MID$(D$(0),5,8)=D$(1) GOTO 2395
2385 NEXT
2390 CLOSE #1:MUSIC MS$:PRINT"■■■■■■■■■NO MODEL !!■■":GOTO 2330
2395 D$(0)=LEFT$(D$(0),4):D$(2)=LEFT$(D$(2),8)
2400 GOSUB 3200:CLOSE #1:B1=2:RETURN
2410 GOSUB 2050:PRINT"■■"
2420 PRINT TAB(29);:IF CP=1 THEN PRINT"P[P]■■"::GOTO 2425
2423 PRINT"P[X]■■";
2425 IF CZ=1 THEN PRINT"C[C]":GOTO 2430
2427 PRINT"C[X]"
2430 GOSUB 1600
2435 PRINT"|  GROUP ";G$:TAB(15);F$(G);TAB(38);"|":GOSUB 1610
2440 FOR I=1 TO 6
2450 PRINT"|";I;" | ";IM$(I);TAB(13);"| ";D$(I-1);TAB(38);"|"
2460 GOSUB 1620:NEXT
2470 PRINT"| D |";TAB(13);"|";TAB(38);"|"
2490 GOSUB 1630
2500 PRINT"■[2,3,4,5,6]DATA     [C]CONTINUE"
2510 PRINT"[D]DELETE           [F]FILE NAME"
2520 PRINT"[M]OK.MODIFICATION  [P]PRINT MODE"
2525 PRINT"■■■■";
2530 GET P$:IF P$="" THEN PRINT"_■ ■ ■"::GOTO 2530
2540 I=ASC(P$)-49
2550 IF I=149 THEN GOSUB 1920:GOTO 2000
2560 IF (I>0)*(I<6) GOTO 2600
2565 IF I=19 GOTO 2730
2570 IF I=21 GOTO 2700
2575 IF I=28 GOTO 3000
2580 IF I=18 THEN CZ=1:CURSOR 37,2:PRINT "C":GOTO 2525
2585 IF I<>31 THEN MUSIC MS$:GOTO 2530
2590 IF CP=1 THEN CP=0:CURSOR 31,2:PRINT "X":GOTO 2525
2595 CP=1:CURSOR 31,2:PRINT "P":GOTO 2525
2600 CURSOR 14,6+I*2
2620 INPUT " ";D$(I)
2630 GOSUB 8000
2640 ON B GOTO 2000,2525,2660
2650 PRINT "■";TAB(14);:MUSIC MS$:GOTO 2620
2660 PRINT "■";TAB(15);D$(I):GOTO 2525
2700 CURSOR 14,4:INPUT " ";F$
2710 IF LEFT$(F$,1)="→" GOTO 2000
2715 F$=LEFT$(F$,16)
2720 FOR JL=1 TO 10:IF F$=F$(JL) THEN MUSIC MS$:GOTO 2525
2723 NEXT
2725 RENAME F$(G),F$:F$(G)=F$:GOTO 2525
2729 REM************************************************************ [DELETION]
2730 CURSOR 5,18:PRINT " DELETE ";TAB(14);
2733 INPUT " ";DN$
2735 I=3:DM$(I)=DN$:D$(I)=DN$:GOSUB 8000
2740 ON B GOTO 2000,2755
```

```
2750 MUSIC MS$:GOTO 2750
2755 DN=VAL(D$(I))
2760 IF DN=0 THEN D$(I)=DM$:GOTO 2525
2770 U=VAL(RIGHT$(D$(0),3))
2780 UU=U+DN-ZE(G)
2790 IF UU>=0 THEN DN=DN-UU
2800 UU=U+DN-1
2810 XOPEN #1,F$(G)
2812 KD=0:IF DN=1 GOTO 2830
2814 FOR I=U+1 TO UU
2816 H=5*I-4
2818 INPUT #1(H),W$
2820 IF LEFT$(W$,1)="*" THEN KD=KD+1:W1$=W$
2822 IF KD=1 THEN W2$=W$
2824 W3$="*"+STR$(I+1)+"  "+STR$(I-1)
2826 PRINT #1(H),W3$:NEXT
2830 K(G)=K(G)+KD-DN:U1=5*U-4:U2=5*UU-4
2831 IF KD=0 GOTO 2844
2832 PRINT #1(U1),"*"+STR$(U+1)+"  "+STR$(VAL(MID$(W2$,5,5)))
2833 PRINT #1(U2),"*"+STR$(VAL(MID$(W1$,2,3)))+"  "+STR$(UU-1)
2834 X=VAL(MID$(W2$,5,5))+5-4:IF (X+4)-5=0 GOTO 2837
2835 INPUT #1(X),W$
2836 PRINT #1(X),"*"+STR$(U)+"  "+STR$(VAL(MID$(W$,5,5)))
2837 X=VAL(MID$(W1$,2,3))+5-4:IF (X+4)-5=ZE(G) GOTO 2840
2838 INPUT #1(X),W$
2839 PRINT #1(X),"*"+STR$(VAL(MID$(W$,2,3)))+"  "+STR$(UU)
2840 IF V<ZB(G) THEN ZB(G)=U
2841 IF VV>ZF(G) THEN ZF(G)=VV
2842 GOTO 2910
2844 IF V>ZF(G) GOTO 2870
2845 IF VV<ZB(G) GOTO 2890
2850 I=ZB(G)
2852 INPUT #1(5*I-4),W$:X=VAL(MID$(W$,2,3)):Y=VAL(MID$(W$,5,5))
2854 IF X<V THEN I=X:GOTO 2852
2855 IF ND<>1 GOTO 2857
2856 PRINT #1(U1),"*"+STR$(X)+"  "+STR$(I):GOTO 2860
2857 PRINT #1(U1),"*"+STR$(U+1)+"  "+STR$(I)
2858 PRINT #1(U2),"*"+STR$(X)+"  "+STR$(UU-1)
2860 PRINT #1(5*I-4),"*"+STR$(U)+"  "+STR$(Y)
2862 IF X=ZE(G) GOTO 2910
2864 INPUT #1(5*X-4),W$
2866 PRINT #1(5*X-4),"*"+STR$(VAL(MID$(W$,2,3)))+"  "+STR$(UU)
2868 GOTO 2910
2870 IF DN=1 GOTO 2874
2872 PRINT #1(U1),"*"+STR$(U+1)+"  "+STR$(ZF(G)):GOTO 2876
2874 PRINT #1(U1),"*"+STR$(ZE(G))+"  "+STR$(ZF(G))
2876 IF ZF(G)=0 GOTO 2882
2878 INPUT #1(5*ZF(G)-4),W$
2880 PRINT #1(5*ZF(G)-4),"*"+STR$(U)+"  "+STR$(VAL(MID$(W$,5,5)))
2882 IF DN=1 THEN ZF(G)=U:GOTO 2887
2884 PRINT #1(U2),"*"+STR$(ZE(G))+"  "+STR$(UU-1)
2886 ZF(G)=UU
2887 IF ZB(G)=0 THEN ZB(G)=U
2888 GOTO 2910
2890 IF DN=1 GOTO 2894
2892 PRINT #1(U1),"*"+STR$(U+1)+"  0":GOTO 2896
2894 PRINT #1(U1),"*"+STR$(ZB(G))+"  0"
2896 IF ZB(G)=0 GOTO 2902
2898 INPUT #1(ZB(G)*5-4),W$
2900 PRINT #1(ZB(G)*5-4),"*"+STR$(VAL(MID$(W$,2,3)))+"  "+STR$(UU)
2902 IF DN=1 THEN ZB(G)=U:GOTO 2910
2904 PRINT #1(U2),"*"+STR$(ZB(G))+"  "+STR$(UU-1)
2906 ZB(G)=U
2910 CLOSE #1:GOTO 2000
3000 GOSUB 3300
3020 IF CP=0 GOTO 3040
```

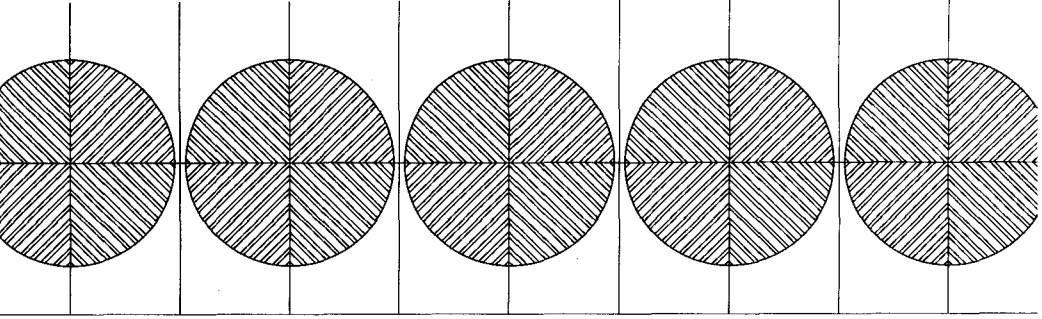
```
3030 GOSUB 1760
3040 IF CZ=0 GOSUB 1920:GOTO 2000
3050 GOSUB 3100:GOSUB 3200:GOTO 2410
3100 D$(2)=LEFT$(D2$,8):D$(3)=D3$:D$(4)=D4$:D$(5)=D5$
3110 D$(0)=LEFT$(D0$,4):D$(1)=MID$(D0$,5,8):RETURN
3200 FOR I=3 TO 5:GOSUB 8300:NEXT:RETURN
3300 XOPEN #1,F$(G)
3310 U=VAL(MID$(D$(0),2,3)):D$=D$(0)+D$(1)
3320 PRINT #1(5*U-4),D$,D$(2),D$(3),D$(4),D$(5)
3330 IF CZ=0 GOTO 3390
3340 U=U+1
3350 IF U>=ZE(G) THEN CZ=0:GOTO 3390
3360 INPUT #1(5*U-4),D0$,D2$,D3$,D4$,D5$
3370 D6$=LEFT$(D0$,1)
3380 IF D6$="*" GOTO 3340
3390 CLOSE #1:RETURN
3499 REM************************************************************** [JOB 3]
3500 PRINT"■**** [JOB 3]  INFORMATION OF FILE  ****"
3510 PRINT"┌──────┬──────────────────┬──────┬──┐"
3520 PRINT"| GROUP |     FILE NAME     | NO. |   |"
3530 FOR I=1 TO 10
3540 GOSUB 3700:0$=CHR$(64+I)
3550 PRINT"|   ";0$;"   | ";F$(I);TAB(28);"|";K(I);TAB(34);
3560 IF F$(I)="" THEN PRINT"|   |":GOTO 3590
3570 IF K(I)<>100 THEN PRINT"| * |":GOTO 3590
3580 PRINT"|***|"
3590 NEXT
3600 PRINT"└──────┴──────────────────┴──────┴──┘"
3610 PRINT"[1-8]JOB 1-8    [L]JOB LIST";TAB(36);
3620 GET G$:PRINT"_■ ■ ■";:IF G$="" GOTO 3620
3630 IF G$="L" GOTO 500
3640 G=ASC(G$)-48
3650 ON G GOTO 1000,2000,3500,4000,5000,6000,7000,7500
3660 MUSIC M$:GOTO 3620
3700 PRINT"├──────┼──────────────────┼──────┼──┤":RETURN
3999 REM************************************************************** [JOB 4]
4000 CZ=0:CP=0:JP=3:PG=0:W=0:GOSUB 4800:PR=0:GOSUB 2060
4030 ON B1 GOTO 500,4040
4035 GOSUB 1920:GOTO 4000
4040 GOSUB 4800
4050 PRINT TAB(29);:IF CP=1 THEN PRINT"P[P]■■";:GOTO 4060
4055 PRINT"P[X]■■";
4060 IF CZ=1 THEN PRINT"C[C]":GOTO 4070
4065 PRINT"C[X]"
4070 GOSUB 1600
4080 PRINT"|  GROUP ";G$;TAB(20);F$(G);TAB(38);"|":GOSUB 4820
4090 FOR I=1 TO 6
4100 IF I>3 GOTO 4120
4110 PRINT"|";STR$(I);"| ";IM$(I);TAB(11);"|";SPC(3);D$(I-1);TAB(38);"|"
4113 IF I<>3 GOSUB 4810:GOTO 4160
4115 PRINT"├──┼──────┼──────────────────────────┤":GOTO 4160
4120 PRINT"| |";TAB(11);"|* ";D$(I-1);TAB(38);"|"
4130 PRINT"|";STR$(I);"| ";IM$(I);TAB(11);"| |";TAB(38);"|"
4140 IF I<>6 THEN PRINT"├──┼──────┼──────────────────────────┤":GOTO 4160
4150 PRINT"└──┴──────┴──────────────────────────┘"
4160 NEXT
4170 PRINT"■[I]INCOMING  [O]OUTCOMING  [P]PRINT"
4180 PRINT"[→]CANCEL    [C]CONTINUE"
4185 PRINT"■■■■";
4190 GET T$:IF T$="" THEN PRINT "_■ ■ ■";:GOTO 4190
4200 IF T$="I" THEN ML=1:GOTO 4280
4210 IF T$="O" THEN ML=2:GOTO 4280
4220 IF T$="P" GOTO 4260
4230 IF T$="→" THEN GOSUB 1920:GOTO 4000
4240 IF T$="C" THEN CZ=1:CURSOR 37,2:PRINT "C":GOTO 4185
4250 MUSIC M$:GOTO 4190
```

```
4260 CURSOR 31,2:IF CP=0 THEN PRINT "P":CP=1:GOTO 4185
4270 PRINT "*":CP=0:GOTO 4185
4280 CURSOR 12,13:PRINT T$:"■■■■":T$:"■■■■":T$
4285 CURSOR 14,13
4290 LM=ML:FOR I=3 TO 5
4300 INPUT " ":IO$(I-3)
4305 DM$(I)=D$(I)
4310 IF (I=4)*(ASC(IO$(I-3))=42) GOTO 4340
4320 D$(I)=IO$(I-3):GOSUB 8000
4323 IF (ML=2)*(I=3) THEN IF VAL(D$(3))>VAL(DM$(3)) THEN B=0
4325 ON B GOTO 4000,4340,4335
4330 PRINT"■";TAB(14)::MUSIC MS$:D$(I)=DM$(I):GOTO 4300
4335 PRINT"■";TAB(15);D$(I):GOTO 4350
4340 PRINT"■■";TAB(14)::NEXT
4350 ML=LM:FOR I=3 TO 5:IO$(I-3)=D$(I):NEXT
4360 D$(3)=DM$(3):D$(4)=IO$(0):GOSUB 9000:DM$(3)=D$(5)
4370 D$(3)=DM$(5):D$(4)=IO$(2):GOSUB 9000:D$(3)=DM$(3):D$(4)=DM$(4)
4400 GOSUB 3300
4410 IF B=1 GOTO 4000
4420 IF CP=0 GOTO 4470
4430 IF ASC(IO$(1))=42 GOTO 4450
4440 D$(4)=IO$(1)
4450 D$(3)=IO$(0):D$(5)=IO$(2)
4460 GOSUB 1760
4470 IF CZ=0 GOTO 4000
4480 GOSUB 3100:GOSUB 3200:GOTO 4040
4800 PRINT"■* [JOB 4] INCOMING OR OUTCOMING STOCK *■":RETURN
4810 PRINT"├─┼──────┼──────────────────────────────┤":RETURN
4820 PRINT"├─┼──────┼──────────────────────────────┘":RETURN
4999 REM********************************************************************* [JOB 5]
5000 ML=0:CP=0:D$="0":JP=0:PG=0:XV=1
5005 GOSUB 5800:GOSUB 1550:CURSOR 16,13:PRINT"GROUP ";
5010 GOSUB 1650:PRINT G$
5012 IF F$(G)="" THEN PRINT"■":TAB(20);"NO FILE !!":GOTO 5018
5014 IF K(G)=0 THEN PRINT"■":TAB(20);"NO DATA !!":GOTO 5018
5016 GOTO 5020
5018 MUSIC MS$:FOR I=1 TO 1000:NEXT:GOTO 5000
5020 PRINT"■■":SPC(39):PRINT"[A]ALL";TAB(12);"[P]PART";TAB(36);"[ ]"
5030 PRINT"■";TAB(37);
5040 GET P$:PRINT"_■ ■ ■";:IF P$="" GOTO 5040
5050 IF P$="A" THEN PRINT"A":D=1:V=ZE(G)-1:GOTO 5197
5060 IF P$="P" THEN PRINT"P":GOTO 5090
5070 IF P$="→" GOTO 5000
5080 MUSIC MS$:GOTO 5040
5090 PRINT"■FROM  [    ]";TAB(17);"TO  [    ]":A$=G$
5100 GOSUB 2150:IF B1=0 GOTO 5000
5105 IF A$=G$ GOTO 5120
5110 PRINT"■CODE OF OTHER FILE !!":GOTO 5140
5120 IF V<ZE(G) THEN D=V:GOTO 5145
5130 PRINT"■NO DATA !!";SPC(11)
5140 PRINT"■■■":MUSIC MS$:GOTO 5100
5145 XV=2:PRINT"■";SPC(30);"■■"
5150 PRINT TAB(15)::GOSUB 2150
5155 IF B1=0 GOTO 5000
5160 IF A$=G$ GOTO 5180
5170 PRINT"■CODE OF OTHER FILE !!":MUSIC MS$:PRINT"■■■":GOTO 5150
5180 IF D>V THEN V=D:GOTO 5195
5190 IF V>(ZE(G)-1) THEN V=ZE(G)-1
5195 PRINT"■";SPC(30)
5197 PRINT#P "■":PRINT#P:PRINT#P
5198 PRINT#P "■":TAB(13);"INVENTORY   LIST■"
5200 XOPEN #1,F$(G):CU=0:DD=V
5210 FOR JJ=D TO V:H=5*JJ-4
5220 INPUT #1(H),D$(0),D$(2),D$(3),D$(4),D$(5)
5230 IF ASC(D$(0))=42 THEN NEXT JJ:GOTO 5290
5240 D$(1)=MID$(D$(0),5,8):D$(0)=LEFT$(D$(0),4):D$(2)=LEFT$(D$(2),8)
```

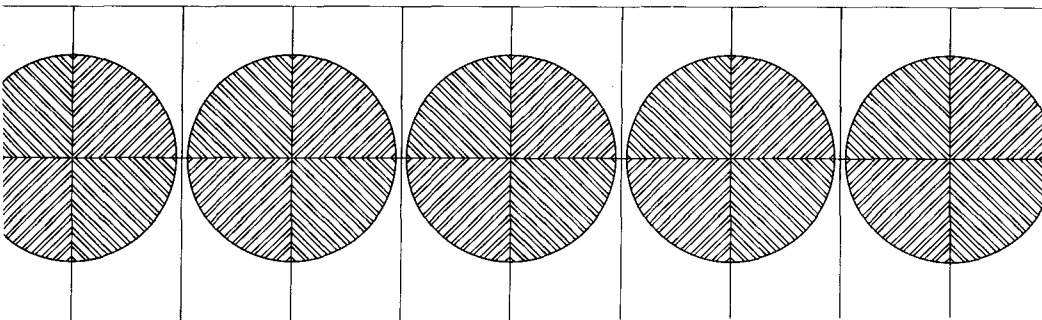
```
5245 GOSUB 3200
5250 D$(6)=D$(5)
5260 GOSUB 1760:CU=CU+1
5270 D$(3)=D$(6):D$(4)=D$:ML=1:GOSUB 9000:ML=0:D$=D$(5)
5275 IF ((CU=36)+(CU=76))*(JJ<DD) GOTO 5279
5278 GOTO 5280
5279 GOSUB 5310:PRINT/P"▮":PRINT/P:PRINT/P:GOSUB 1830
5280 NEXT JJ
5290 CLOSE #1:IF CU=0 GOTO 5360
5300 D$(5)=D$:I1=5:GOSUB 9900:I1=3:GOSUB 5310:GOTO 5330
5310 PRINT/P"▮──────┴──────────┴──────────┴──────────────┴──────────────────────┴─";
5320 PRINT/P"────────────────────":RETURN
5330 PRINT/P:PRINT/P TAB(44);"TOTAL AMOUNT";TAB(80-TB(2));D$(5)
5340 PRINT/P TAB(44);"──────────────────────────────────────"
5350 GOTO 5000
5360 CURSOR 14,23:PRINT"NO DATA !!":MUSIC MS$
5370 FOR JJ=1 TO 1000:NEXT:GOTO 5000
5800 PRINT"▮******** [JOB 5]   FILE COPY  *********▮▮":RETURN
5999 REM************************************************************** [JOB 6]
6000 PRINT"▮**[JOB 6] CODE-MODEL REFERENCE TABLE **▮▮"
6010 JP=0:CP=0:GOSUB 1550
6020 CURSOR 16,15:PRINT"GROUP ";
6030 GOSUB 1650:PRINT G$:CURSOR 16,17
6070 IF LEN(F$(G))<>0 GOTO 6080
6075 PRINT"NO FILE !!":GOTO 6090
6080 IF K(G)<>0 GOTO 6100
6085 PRINT "NO DATA !!"
6090 MUSIC MS$:FOR I=1 TO 1000:NEXT:GOTO 6000
6100 IF K(G)=0 THEN PRINT"NO DATA !!":GOTO 6090
6110 JB=0:XOPEN #1,F$(G)
6120 FOR I=1 TO ZE(G)-1
6130 H=5*I-4
6140 INPUT #1(H),D$(0)
6150 IF ASC(D$(0))=42 GOTO 6260
6160 D$(1)=MID$(D$(0),5,8):D$(0)=LEFT$(D$(0),4)
6170 IF (JB=1)*(L=4) GOTO 6220
6175 IF (JB=1)*(L<>4) GOTO 6250
6177 PRINT/P:PRINT/P:PRINT/P:PRINT/P
6180 PRINT/P"▮";TAB(7);"CODE-MODEL REFERENCE TABLE":PRINT/P:PRINT/P
6190 PRINT/P"(GROUP ";G$;")  ";F$(G);TAB(30);DT$:PRINT/P"▮▮"
6200 FOR K=1 TO 4:PRINT/P "┌───────┬──────────┐ ";:NEXT:PRINT/P ""
6210 FOR K=1 TO 4:PRINT/P "| CODE |  MODEL   | ";:NEXT:PRINT/P ""
6220 FOR K=1 TO 4:PRINT/P "├───────┼──────────┤ ";:NEXT:PRINT/P "":L=0
6250 PRINT/P "| ";D$(0);" | ";D$(1);TAB(17);" | ";:L=L+1:JB=1
6260 NEXT
6270 IF L=4 GOTO 6290
6280 FOR K=L+1 TO 4:PRINT/P "|       |          | ";:NEXT:PRINT/P ""
6290 FOR K=1 TO 4:PRINT/P "└───────┴──────────┘ ";:NEXT:PRINT/P "▮"
6300 CLOSE #1:GOTO 6000
6999 REM************************************************************** [JOB 7]
7000 PRINT"▮****** [JOB 7]  DELETION OF FILE ******▮▮"
7010 GOSUB 1550
7020 CURSOR 10,15
7030 PRINT"DELETION OF GROUP ";
7040 GOSUB 1650:PRINT G$
7050 IF LEN(F$(G))<>0 GOTO 7080
7060 CURSOR 10,17:PRINT"NO FILE !!"
7070 MUSIC MS$:FOR I=1 TO 1000:NEXT:GOTO 7000
7080 DELETE F$(G)
7090 F$(G)="":K(G)=0:ZB(G)=0:ZF(G)=0:ZE(G)=1
7100 GOTO 7000
7499 REM************************************************************** [JOB 8]
7500 PRINT"▮******* [JOB 8]  END OF ALL JOB ******"
7510 CURSOR17,12
7520 PRINT"E N D"
7530 XOPEN #2,"INITIAL SET"
```

```
7540 FOR I=1 TO 10:H=3+I-2
7550 K$=STR$(K(I))+"  "+STR$(ZB(I))+"
7560 F$=STR$(ZF(I))+"  "+STR$(ZE(I))
7570 PRINT #2(H),F$(I),K$,F$
7580 NEXT:CLOSE
7590 PRINT"■"
7600 END
7999 REM************************************************** [DATA JUDGEMENT]
8000 IF ASC(D$(I))=253 THEN B=0:RETURN
8005 IF ASC(D$(I))=198 THEN B=1:GOTO 8200
8010 IF ASC(MID$(D$(I),JT(I-1)+1,1))<>32 THEN B=0:RETURN
8020 D$(I)=LEFT$(D$(I),JT(I-1))
8030 IF I<3 THEN B=2:RETURN
8040 IF (I=5)*(ASC(D$(I))=36) THEN ML=0:B=3:GOSUB 9000:RETURN
8050 X$=LEFT$(D$(I),7):V$=MID$(D$(I),8,5):Z$=MID$(D$(I),13,4)
8060 X1=VAL("1"+X$):V1=VAL("1"+V$):Z1=VAL("1"+Z$)
8070 IF (X1=0)+(V1=0)+(Z1=0) THEN B=0:RETURN
8080 X$=STR$(VAL(X$))
8090 IF V1=1 THEN D$(I)=X$:GOTO 8130
8100 V$=X$+MID$(STR$(V1),2,5)
8110 IF Z1=1 THEN D$(I)=V$:GOTO 8130
8120 D$(I)=V$+MID$(STR$(Z1),2,4)
8130 B=2:RETURN
8200 IF JP=0 THEN RETURN
8210 GOSUB 1920
8220 IF JP<>1 THEN RETURN
8230 IF CF=0 THEN RETURN
8240 IF ZE(G)=1 THEN RETURN
8250 ZE(G)=ZE(G)-1:RETURN
8300 X$=LEFT$(D$(I),8):V$=RIGHT$(D$(I),8)
8310 IF ASC(V$)=32 THEN D$(I)=STR$(VAL(X$)):RETURN
8320 IF ASC(V$)<>48 THEN D$(I)=X$+STR$(VAL(V$)):RETURN
8330 FOR K=2 TO 8:V=ASC(MID$(V$,K,1))
8340 IF V<>32 THEN NEXT K
8350 D$(I)=X$+LEFT$(V$,K-1)
8360 RETURN
8499 REM************************************************** [DISPOSAL OF ERRORS]
8500 IF (ERL=7080)*(ERN=40) THEN RESUME 7090
8510 MUSIC MS$
8520 IF (ERL=2725)*(ERN=42) THEN RESUME 2525
8530 PRINT"■*********   ERROR OCCURS !!   *********":CURSOR 10,12
8540 IF ERN=65 THEN PRINT "UNCONNECTED PRINTER !!":GOTO 8590
8550 IF ERN=66 THEN PRINT "TROUBLED PRINTER !!":GOTO 8590
8560 IF ERN=67 THEN PRINT "PAPER EMPTY !! ":GOTO 8590
8570 IF ERN=50 THEN PRINT "UNCONNECTED FLOPPY !!":GOTO 8590
8580 IF ERN=54 THEN PRINT "UNINITIALIZED DISKETTE !!":GOTO 8590
8585 GOTO 8630
8590 CURSOR 10,14
8600 GET ER$:IF ER$="" THEN PRINT"_■ ■ ■";:GOTO 8600
8610 PRINT"■":CURSOR 15,12:PRINT "WORKING !!"
8615 IF (ERN>64)*(ERN<68) THEN RESUME
8620 IF (ERN=50)+(ERN=54) THEN KILL:RESUME
8630 CURSOR 10,12:PRINT"CAN'T DISPOSE !!"
8640 CURSOR 10,14:PRINT"JOB LIST IS EXECUTED."
8650 FOR I=1 TO 5000:NEXT
8660 IF (ERL>=1000)*(ERL<=1490) GOSUB 8200
8670 RESUME 500
8999 REM************************************************** [ARITHMETIC OPERATION]
9000 M=LEN(D$(3)):N=LEN(D$(4)):W=6:WW=1000000:WS=5
9010 IF ML=0 THEN W=3:WW=1000:WS=2
9020 V0=VAL(D$(3)):V1=VAL(D$(4))
9030 ON ML GOTO 9070,9085
9040 IF (M+N)>17 THEN B=0:RETURN
9050 IF (M+N)<9 THEN D$(5)=STR$(V0+V1):RETURN
9060 GOTO 9090
9070 IF (M<8)*(N<8) THEN D$(5)=STR$(V0+V1):RETURN
```

```
9080 GOTO 9090
9085 IF (M<9)*(N<9) THEN D$(5)=STR$(V0-V1):RETURN
9090 FOR J=0 TO 11:C$(J)="":C(J)=0:NEXT
9100 MM=INT(M/W):IF MM=M/W THEN MM=MM-1
9110 NN=INT(N/W):IF NN=N/W THEN NN=NN-1
9120 ON ML+1 GOTO 9160,9130,9160
9130 IF MM>=NN THEN ZZ=1:GOTO 9160
9140 ZZ=0:V=N:N=M:M=V:V=NN:NN=MM:MM=V
9150 V$=D$(4):D$(4)=D$(3):D$(3)=V$
9160 A(MM)=VAL(LEFT$(D$(3),M-W*MM))
9170 B(NN)=VAL(LEFT$(D$(4),N-W*NN))
9180 IF MM=0 GOTO 9200
9190 FOR J=0 TO MM-1:A(J)=VAL(MID$(D$(3),M-W*J-WS,W)):NEXT
9200 IF NN=0 GOTO 9220
9210 FOR J=0 TO NN-1:B(J)=VAL(MID$(D$(4),N-W*J-WS,W)):NEXT
9220 ON ML GOTO 9600,9800
9299 REM************************************************************ [MULTIPLICATION]
9300 FOR J=0 TO NN
9310 FOR K=0 TO MM
9320 C(K+J)=C(K+J)+A(K)*B(J)
9330 NEXT K,J
9340 L=MM+NN
9350 IF L=0 GOTO 9440
9360 FOR J=0 TO L-1
9370 P=INT(C(J)/WW)
9380 IF P>=1 THEN C(J)=C(J)-P*WW:C(J+1)=C(J+1)+P
9390 NEXT
9400 FOR J=0 TO L-1
9410 C$(J)=STR$(C(J)):LN=LEN(C$(J))
9420 C$(J)=LEFT$(ZZ$,W-LN)+C$(J)
9430 NEXT
9440 C$(L)=STR$(C(L))
9450 D$(5)="":FOR J=L TO 0 STEP -1
9460 D$(5)=D$(5)+C$(J):NEXT
9470 IF (JP<>0)*(LEN(D$(5))>16) THEN B=0:RETURN
9480 IF (ML=1)*(ZZ=0) THEN D$(3)=D$(4):D$(4)=V$
9490 RETURN
9599 REM************************************************************** [ADDITION]
9600 FOR J=0 TO MM
9610 IF J>NN THEN C(J)=A(J):GOTO 9630
9620 C(J)=A(J)+B(J)
9630 NEXT
9640 L=MM
9650 GOTO 9350
9799 REM********************************************************** [SUBSTRUCTION]
9800 FOR J=0 TO MM
9810 IF J>NN THEN C(J)=C(J)+A(J):GOTO 9830
9820 C(J)=C(J)+A(J)-B(J)
9830 IF C(J)<0 THEN C(J)=C(J)+WW:C(J+1)=C(J+1)-1
9840 NEXT
9850 FOR J=MM TO 0 STEP -1
9860 IF C(J)<>0 THEN L=J:GOTO 9880
9870 NEXT:D$(5)="0":RETURN
9880 IF L=0 GOTO 9440
9890 GOTO 9400
9899 REM******************************************************** [COMMA INSERTION]
9900 FOR K=I1 TO I2:FF$=""
9910 M=LEN(D$(K)):MM=INT(M/3)
9920 IF MM=M/3 THEN MM=MM-1
9930 V$=LEFT$(D$(K),M-3*MM)
9940 IF MM=0 GOTO 9980
9950 FOR J=0 TO MM-1
9960 FF$=","+MID$(D$(K),M-3*J-2,3)+FF$
9970 NEXT J
9980 D$(K)=V$+FF$:TB(K-3)=M+MM
9990 NEXT K :RETURN
```



Chapter 4
Summarization of the Disk Basic



List of Disk Basic commands

Following shown below are the list of commands used for DISK BASIC SP-6015. Commands are divided into groups in the sequence of direct execution commands, error processing commands, file control commands, statements, link to machine language commands, string processing commands, functions, arithmetic operators, logical operators, and symbols.

Care must be taken for the structure and contents of commands, as they are subject to change and modification according to revision of the software.

Direct execution commands

DIR	DIR1 DIR DIR2	Directory of the diskette in the drive 1 is displayed. In the case of the drive 1, the number "1" can be omitted as "DIR." Directory of the diskette in the drive 2 is displayed. As the directory display stops once after occupying a full display, it then becomes possible to execute the next command. Depressing the "CR" key will continue to display the directory. Because the system holds that drive number when the DIR command is executed, the drive number can be omitted if the same drive is to be designated by any of the direct execution commands or file access commands.
DIR/P	DIR/P DIR2/P	Directory of the diskette set in the drive 1 is printed on the optional printer. The above operation is executed for the diskette set in the drive 2. In printing out the directory the sector numbers of a file are also printed at the same time.
	File name (BTX file) ↓	
LOAD	LOAD "TEXT A" LOAD FD2@10, "TEXT A"	BASIC text (BTX) named "TEXT A" is loaded. The above operation is executed for an auxiliary diskette with the volume No. 10 which is set in the drive 2.
	LIMIT 49151 READY LOAD "OBJ" ↑ Object file	Shown in the left is an example of using the LOAD command which is used in linking the BASIC text with the machine language program (OBJ). The area is restricted for BASIC usage by the LIMIT command and the machine language program of which file name is "OBJ" is loaded. This can be executed in the midst of the BASIC text as a statement. (Refer to the "Link to Machine Language Command.")
LOAD/T	LOAD/T "TEXT B"	BASIC text of which file name is "TEXT B" is loaded from the cassette tape file.
SAVE	SAVE "TEXT C"	A file name called "TEXT C" is given to the BASIC text currently stored in the computer and written to the diskette.
	SAVE FD1, "TEXT C"	The above operation is executed for the drive 1.
SAVE/T	SAVE/T "TEXT D"	A file name called "TEXT D" is given to the BASIC text currently stored in the computer and written to the cassette tape file.

RUN	RUN	Starts program from the beginning of the text.
	RUN 1000	Starts program from the statement number 1000.
	RUN "TEXT A" ↑ BTX file	Loads the program named "TEXT A" from the diskette and executes the program. (With RUN command all the variables are cleared to zero before the execution of the program.)
	RUN FD3@7, "SYSTEM 1" ↑ Object file	Loads from the drive number 3 the machine language program file named "SYSTEM 1" with volume number 7. In this case the system control disengages the BASIC.
LIST	LIST	Currently loaded BASIC program is shown on the CRT screen.
	LIST 70	Displays only the program of the statement number 70.
	LIST 70–	Displays programs after the statement number 70.
	LIST 70–100	Displays programs from the statement number 70 to 100.
	LIST –100	Displays programs up to the statement number 100.
LIST/P	LIST/P	The same function as the above LIST command but uses the printer instead of the CRT screen.
NEW	NEW	The program currently in the computer is deleted.
CONT	CONT	With this command any interrupted program can be resumed from where it was interrupted (when the program is interrupted by such as the BREAK key operation, the STOP command or END command in the program. However, it is not possible to use the CONT command when a program using the statement number is edited after the interruption. Any direct execution command can be used.)
CLR	CLR 10 CLR	Clears all the variables with number and the string variables.
BYE	BYE	Stops use of BASIC and returns to monitor.

Error processing commands

ON ERROR	10 ON ERROR GOTO 1000	Declarative statement which directs the program jump to the statement number 1000 when an error is encountered during program execution.
IF ERN	1000 IF ERN = 44 THEN 1050	With this command the program jumps to the statement number 1050 when error number is "44."
IF ERL	1010 IF ERL = 350 THEN 1090	With this command the program is directed to jump to the statement number 1090 when the error causing statement number is 350.
	1030 IF (ERN = 53)*(ERL = 700) THEN END	With this command program termination is directed when the error number is 53 with the error causing statement number being 700.
RESUME		With this command the program returns to the main program after the error processing. There are the following types of restorations.
	1080 RESUME	Hands the control to the error causing command again.
	1120 RESUME NEXT	Hands the control to the command following the error causing command.
	1150 RESUME 440	Hands the control to the statement number 440.
	1900 RESUME 0	Hands the control to the top of the program.

File control commands

LOCK	LOCK "ABC" 150 LOCK FD1@1, "ABC"	Locks the file "ABC" contained in the diskette. With this command the above operation is executed in the program to the file "ABC" contained in the diskette of the volume number 1 of the drive 1. For the locked file, an asterisk mark "*" is at attached next to the file mode symbol when the directory is displayed.
UNLOCK	UNLOCK FD2, "DEF"	Releases the lock of the file "DEF" of the diskette in drive 2.
	200 UNLOCK "DEF"	Releases the lock of the file "DEF" in the diskette in the program.
RENAME	RENAME FD2@3, "DEF","GHI"	Renames the file "DEF," in the diskette of the volume number 3 of the drive 2, to "GHI."
	55 RENAME "NAME1","NAME2"	Rename of a file name is executed in the program.
DELETE	DELETE "FILE A" 150 DELETE FD2, "FILE A"	Deletes the file "FILE A" from the diskette. Deletion of a file is executed in the program.
CHAIN	300 CHAIN "TEXT 2"	Chains to the "TEXT 2" of the BASIC text file in the diskette. In other words, the "TEXT 2" is loaded and program execution is started from the top of that program. At this occassion, the previous program is subjected to NEW, but the values in variables are transferred to the chained text. (CHAIN can be taken as GOTO "statement.")
SWAP	500 SWAP FD2@7, "SUBR 1"	Swaps with the BASIC text file "SUBR 1" which is located in the diskette of the volume number 7 of the drive 2. That is, the text in execution is saved once in the diskette, and fetches "SUBR 1" instead, then program execution is started from the top of that program. Upon completion of "SUBR 1," the saved text is then restored and program execution continues from the next command. SWAP must be executed in one level only. (SWAP can be taken as a subroutine call.)
WOPEN#	20 WOPEN #3, FD2@7, "SEQ DATA 1"	In order to write a sequential access file, the file name "SEQ DATA 1"in the diskette of the volume number 7 of the drive 2 is defined to logical number #3, and opens the file.
	20 WOPEN #3, "SEQ DATA 1"	When the drive number can be omitted (refer to DIR command), it can be written as in the left. (As for WOPEN# commands for USR function, description will be given later on.)
PRINT#	30 PRINT #3, A, A$	Writes the contents of the variable A and string variable A$ into the sequential access file defined to the logical number #3. However, they are in temporary registration until the CLOSE# command is executed. (Write data is performed in a block unit of 128.)
CLOSE# (In case WOPEN# is used.)	40 CLOSE #3	The CLOSE# command under the WOPEN# command duly registers the sequential access file. Then, relevant logical number is closed as undefined.
KILL#	35 KILL #3	The KILL# command (or KILL command) under the WOPEN# command cancels the WOPEN# command and suspends registration of the sequential access file. Then, treats the logical numbr as undefined.

ROPEN#	70 ROPEN #5, FD1@8, "SEQ DATA 1"	In order to perform reading of the sequential access file, the file name "SEQ DATA 1" to be read is designated to be loaded into the diskette of the volume number 8 of the drive 1 as logical number #5, and opens the file.
	70 ROPEN #5, "SEQ DATA 1"	If the drive number can be omitted, it can be written as in the left.
INPUT#	80 INPUT #5, B, B$, C, C$	Reads the data from the sequential access file defined to the logical number#5. In the case of this example, variables and string variables, B, B$, C and C$ are substituted with data one after another.
CLOSE# (In case ROPEN# is used.)	90 CLOSE #5	The CLOSE# command under the ROPEN# command terminates read and open, and closes the logical number as undefined.
XOPEN#	25 XOPEN #7, FD2@9, "RND DATA"	In order to perform read and write of the random access file, the file name "RND DATA" is defined and cross open it to the diskette of the volume number 9 of the drive 2 as logical number #7.
	25 XOPEN #7, "RND DATA"	If the drive number can be omitted, it can be expressed as in the left.
PRINT#( )	35 PRINT #7 (5), R5, R5$	Writes the contents of the value in the variable R5 and string variable R5$ into the elements 5 and 6 of the random access file defined to the logical number #7. (The string must be up to 16 bytes.)
INPUT#( )	45 INPUT #7 (5), R (10)	Fetches the numerical data in the element 5 of the random access file defined to the logical number #7 to the superscribed variable R (10).
CLOSE# (In case XOPEN# is used.)	55 CLOSE #7	The CLOSE# command under the XOPEN# command terminates cross open and closes the logical number as undefined.
CLOSE	100 CLOSE	All logical opens are closed.
KILL	90 KILL KILL	Executes KILL to all the logical open. Example in the left shows that KILL is executed as a direct execution command. However, KILL# command can execute without accessing the disk drive unit.
IF EOF (#) ...THEN	200 IF EOF (#5) THEN 300	When an INPUT# statement is executed against the sequential access file or the random access file, the condition turns "true" when a file end is encountered.

Statements

GOTO	100 GOTO 200	Jumps from statement numbers 100 to 200.
GOSUB	100 GOSUB 700	Branches to subroutine at statement number 700.
RETURN	800 RETURN	Ends subroutine execution and returns to next statement assigned by GOSUB statement in the main program.
ON . . . GOTO	10 ON AGOTO 70, 80, 90	Jumps to statement number 70 if value of variable A is 1, statement number 80 if value is 2, and statement number 90 if the value is 3. Executes next statement if A = 0 or A≧4. ON includes INT, and if A is 2.7, A is regarded as 2 jumping to statement number 80.

ON . . . GOSUB

```
100 ON A GOSUB 700, 800
```

Branches to statement number 700 if value of variable A is 1, and statement number 800 if value is 2. Executes next statement if A = 0 or A ≧ 3. ON includes INT.

```
10 IF A>20 THEN 100
```

Jumps to statement number 100 if A is greater than 20. Executes next statement if A is less than 20.

```
20 IF A<> 10 THEN 200
```

Jumps to statement number 200 if variable A is not equal to 10. Executes next statement ifA equals 10.

```
30 IF B<3 THEN B = B + 3
```

Substitues B + 3 value for variable B if value of variable B is less than 3. Executes next statement if B is greater than 3.

```
40 IF B>7 THEN PRINT B
```

Displays B value on CRT screen if value of variable B is greater than 7. Executes next statement is B is less than 7.

IF . . . GOTO

```
100 IF A>= B GOTO 10
```

Jumps to statement number 10 if value of variable A is greater than that of variable B.
Executes next statement if A is less than B.

IF . . . GOSUB

```
100 IF A = B*2 GOSUB 700
```

Branches to statement number 700 if the value of variable A is double that of variable B.
Executes next statement if not. (If multi-statement follows conditional sentence, ON statement is executed when condition is not met, while IF statement shifts to next statement number when condition is not met, and multistatement is omitted.)

FOR . . . NEXT STEP

```
10 FOR A = 1 TO 10
20 PRINT A
30 NEXT A
```

Statement number 10 states that variable A should be changed from 1 to 10, and the first value of A is 1. Statement number 20 states value of A is to be displayed on CRT screen, therefore 1 is on display. At statement number 30, the value of A is 2, when this loop is repeated. Accordingly, 2 is displayed as the value of A. Thus, this loop is repeated until the value of A becomes 10. (When the loop is ended, the value of A is 11.)

```
10 FOR B = 2 TO 8 STEP 3
20 PRINT B↑2
30 NEXT
```

Statement number 10 states that value of B should be increased from 2 to 8 by increments of 3. With the value of STEP made minus, the value of the variable can be decreased. At statement number 20, B squared is displayed. Statement number 30 is not NEXT B. In this case, however, when no variable is given to NEXT statement, it is combined with the nearest FOR loop. In example at left, FOR . . . . NEXT loop about variable B is executed. It is suggested that the variable name is given to the NEXT statement.

```
10 FOR A = 1 TO 3
20 FOR B = 10 TO 30
30 PRINT A, B
40 NEXT B
50 NEXT A
```

Loop B

Loop A

This is an example of double FOR . . . . NEXT loops for variables A and B. Pay attention to the fact that B loop is placed inside A loop. Although double, triple, etc. loops are possible, inner loops must be enclosed by outer loops. The configuration of multi-loop is called "nesting".
In example at left, variable A is first set to 1 and variable B to 10. With A remaining at 1, B is changed from 10 to 11, 12, until it reaches 30 under execution of loop B. At statement number 50, value of A becomes 2 with loop B execution started agian.

```
60 NEXT B: NEXT A
60 NEXT B, A
70 NEXT A, B
```

In substitution for NEXT statements at statement numbers 40 and 50, two statements at number 60 shown at left can be used. However, statement number 70 cannot be used, causing an error to occur.

READ ...DATA

Substitutes constant or string placed in DATA statement for variable shown in READ statement. First READ statement reads the first value of DATA statement, and second READ statement reads the second value of DATA statement. Variable in READ statement and corresponding value in DATA statement should match the form of variable, with numeral for numeric variable and string for string varaible. In addition, no equation can be placed in DATA statement. DATA statement can be placed anywhere in the program, but at the beginning or end for convenience.

```
10 READ A, B, C, D
20 DATA 25, -0.5, 0, 500
```

In READ and DATA statements at left, values of 25, −0.5, and 500 are substituted for variables A, B, C and D.

```
10 READ H$, H, S$, S, S$, D
20 DATA HEART, 3
30 DATA CLUB, 9
40 DATA SPADE, 6
```

In example at left, first value in DATA statement or string "HEART" is substituted for first variable in READ statement or string variable H$. Value 3 is substituted for second variable H, and thus read one after another.

```
10 READ X1$, X2$, Y$
20 DATA "A, B, C, D"
30 DATA "E, F, G"","" : 16"
```

When string includes "," and ":", it is indicated by quotation marks. In example at left, string in quotation marks of DATA statement are substituted for string variables X1$, X2$ and Y$, respectively.

RESTORE

In READ .... DATA statements, values read from DATA statement are shifted as READ statement progresses. However, use of RESTORE statement allows readout value to return to the first DATA statement.

```
10 READ A, B, C
20 RESTORE
30 READ D, E
40 DATA 3, 6, 9, 12, 15
```

In example at left, values 3, 6 and 9 are substituted by READ statement at statement number 10 for variables A, B and C, respectively. Since, however, RESTORE statement is placed next, values to be substituted for variables D and E by READ statement at statement number 30 are not 12 and 15, but 3 and 6 are substituted respectively.

MUSIC TEMPO

This states music to be automatically played. With a tempo assigned by TEMPO statement, a string (equivalent to a group of notes assigned for scale and duration) for a melody between quotation marks in MUSIC statement is converted to sound for delivery through a speaker.

```
MUSIC "CDEFG"
```

Do, re, mi, fa and sol in the mid-range of the scale sound with quarter notes at TEMPO 4.

```
300 TEMPO 7
310 MUSIC "DE #FGA"
```

At statement number 300, TEMPO 7 is assigned (the fastest). At statement number 310, re, mi and fa with # sol and la in the mid-range are played at quarter note duration. With TEMPO statement not assigned, playing is conducted at TEMPO 4.

```
300 M1$ = "C3EG C"
310 M2$ = "B3GD G"
320 M3$ = "C8R5"
330 MUSIC M1$, M2$, M3$
```

In example at left, a melody is substituted for 3 string variables and MUSIC statement is executed. Melody at right on the scale is played. This has no TEMPO statement assigned, and TEMPO4 is set for playing.

DEF FN

```
100 DEF FNA (X) = X ↑ 2 − X
110 DEF FNB (X) = LOG (X)
120 DEF FNC (Y) = LN (Y)
```

DEF FN defines functions. Statement 100 defines $X^2 - X$ as FNA (X), statement number 110, $\log_{10} X$ as FNB (Y) and statement number 120, logeY as FNC (Y). Function is limited to 1 variable. Function names require the form of "variable ( )" as illustrated A ( ), B ( ) and C ( ).

SET **RESET**	SET X, Y RESET X, Y	Lights the coordinate on CRT screen as assigned by variables X and Y of SET statement. Lights assigned by RESET statement go out. The abscissa is from 0 to 79 starting at the left on CRT screen, while the ordinate is from 0 to 49 starting from the upper to lower part. This means top left corners are 0 and 0, and bottom right corners are 79 and 49. starting from the upper to lower part. This means top left corners are 0 and 0, and bottom right corners are 79 and 49. (Under normal operation, CRT screen display consists of 40 for X and 25 for Y. Please note.)
	SET 0, 0	Lights the top left corner point.
	10 SET 79, 49 20 RESET 79, 49 30 GOTO 10	Flashes the bottom right corner point repeatedly at statement numbers 10 to 30.
		X and Y assignments can be made by variable, equation or constant.
	10 FOR J = 0 TO 49 20 SET J, J 30 NEXT	Program at left is for drawing a slash from top left corner to bottom corner on the CRT screen.
	100 FOR J = 0 TO 79 110 SET J, SQR (J) 120 NEXT	At statement numbers 100 to 120, a parabola is drawn on the CRT screen. SQR(J) does not always produce an integer, but the SET statement includes INT, making it possible to assign any coordinate. What requires attention is that when X and Y exceed 79 and 49, respectively, in the SET and RESET coordinates, X and Y turn to X − 80 and Y −50, respectively. Coordinate assignment by minus value, or X and Y assigned by numerals exceeding 255 characters may cause an error to occur.
CORSOR	80 CURSOR 25, 15 90 PRINT "ABC"	The CURSOR command is utilized to move the cursor to any desired point on the CRT screen. Its position must be directed by a numerical figure or variable; 0 ~ 39 for the X-axis as counted from the left and 0 ~ 24 for the Y-axis as counted from the top. In the example given in the left, the string "ABC" indicates the 26th position from the left and 16th position from the top.
INPUT	10 INPUT A 20 INPUT A$ 30 INPUT "VALUE?"; A	Receives numeral for variable A from keyboard. Receives string for string variable A from keyboard. Before receiving keyin from keyboard, this displays string VALUE?. Separating string from variable uses semicolon ";". However, ? for numeral is not printed again. Numeric variable and string variable can be used in a mixed condition when separated by a comma ",", but the form of variable should be followed when received.
	40 INPUT A, A$, B, B$	
GET	10 GET A	Receives numeral of 1 character for variable A from keyboard. With no key pressed in this case, 0 is received.
	20 GET A$	Receives 1 string for string variable A$ from keyboard. With no key pressed in this case, A$ becomes a empty string.
PRINT	10 PRINT A	Displays value of variable A on CRT screen.
	20 PRINT A$	Displays characters of string variable A on CRT screen.
	30 PRINT A; A$, B; B$	Numeric and string variables can be used in a mixed condition. In addition, when a semicolon is used for separation, continuous display without spaces is possible. If a comma is used, display starts at next position (10 characters per separation). The contents of a string enclosed by quotation marks are displayed as they are. In case of a PRINT statement alone, a line is fed.
	40 PRINT "COST =" ; A	
	50 PRINT	
PRINT/P	10 PRINT/P A 20 PRINT/P A; A$, B ; B$ 30 PRINT/P "COST ="; A	Exectes function identical to PRINT to optional printer. An error occurs if no printer is provided. This statement can be used, irrespective of WOPEN statement.

SIZE	PRINT SIZE	Displays unused memory size (bytes) of the memory presently built into the computer.
TI$	TI$ = "102030" 10 TI$ = "102030"	Sets internal clock to 10 hours 20 minutes and 30 seconds AM. A number of 6 figures is used between quotation marks.
	20 PRINT TI$	Displays the actual time of the internal clock.
TAB	10 PRINT TAB (X); A	Shifts cursor by X character spaces from the left hand on CRT screen, and displays value of variable A.
	20 PRINT TAB (5); A$	Shifts cursor by 5 character spaces from the left hand on CRT screen, and displays character train of string variable A$.
SPC	10 PRINT SPC (X); A 20 PRINT SPC (5); A$	Prints X spaces and displays value of variable A. Prints 5 spaces and displays character train of string variable A$.
DIM		When using variable with a subscript, the maximum of the subscript must be declared by this DIM (abbr. of dimension) statment. Subscript can be used with numerals from 0 to 255 (maximum). Subscript can also be declared by variable and equation in the above range.
	10 DIM A (20)	For variable A with subscript ( ), 21 arrays from A (0) to A (20) are prepared.
	20 DIM B (99, 99)	Variable B with double subscript ( ) can have 10000 arrays from B (0, 0) to B (99,99) are prepared.
	30 DIM A (20), B (99, 99)	Previous statement numbers 10 and 20 are combined for declaration in the form of statement number 30.
	40 DIM C1$ (10)	For string variable C1$ with subscript ( ), 11 arrays from C1$ (0) to C1$ (10) are prepared.
	50 DIM AA (X), AB (X∗2)	For variable AA ( ) with subscript, arrays from AA (0) to AA (X) and for AB ( ), arrays from AB (0) to AB(X∗2) are prepared, respectively.
	60 DIM K$ (7, 5)	String variable K$ with double subscript ( ) can have 48 arrays from K$ (0,0) to K$ (7, 5) are prepared.
STOP	200 STOP	Stops program execution and waits for next instruction. With CONT command given, this continues program execution.
END	999 END	This ends program execution, but with CONT command given, it executes the next program.
REM	100 REM GAME: A = 30	This is a comment and is omited during program execution. The colon ":" used even in REM statement is the separator of multi-statment, and 30 is substituted for variable A.
INP	10 INP @12, A 20 PRINT A	As the I/O port number is designated, data on that port is read. With the statement number 10, the data on the I/O port number 12 (decimal) is input to the variable A.
OUT	10 B = A∗2 20 OUT @13, B	Data is transferred to the external device after designation the I/O port number. With the statement number 20, the value in the variable B is output to the output port of the I/O port number 13. (Connection with an external device is carried out via the interface unit. For more details about INP and OUT commands, refer to the interface unit and universal card manuals.)

Link to machine language commands

LIMIT

```
10 LIMIT 49151
         ↑
         └── Decimal number
```

With this command the area occupied by the BASIC program is limited to the address 49151 (BFFF in hexadecimal system).

```
LIMIT 49151
30 LIMIT A
LIMIT A
```

With this command the area occupied by the BASIC program is limited to the address expressed by the variable A.

```
50 LIMIT $BFFF
          ↑
          └── Hexadecimal system
LIMIT $BFFF
```

With this command the area occupied by the BASIC program is limited to the address BFFF.
The affix "$" must be used when the hexadecimal notation is used.

```
100 LIMIT MAX
LIMIT MAX
70 LIMIT $BFFF
80 LOAD FD2, "S-R 1"
```

Maximum memory area is allocated for the BASIC program area.
When the machine language program (OBJ file) "S-R 1" is to be over the loading address C000, "S-R 1" is loaded into the machine lanugage linking area from drive 2 by means of the program shown in the left.

PEEK

```
100 A = PEEK (49450)
            ↑
Decimal system
110 B = PEEK (C)
```

The data stored in the decimal address 49450 is converted in the decimal number and substituted for the variable A.
The data stored in a decimal address designated by the variable C is converted into the decimal number and substituted for the variable B.
(When the BASIC area address is designated, "32" is always read.)

POKE

```
200 POKE 49450, 175
Decimal system ↑
POKE 49450, 175
210 POKE AD, DA
```

Set the data "175" (decimal number) to the decimal address 49450.

Set the value indicated by the variable DA (within a range of 0 to 255) is set to the address designated by the variable AD.

USR

```
300 USR (49152)
          ↑
Decimal system
```

Program control is handed down to the decimal address 49152. This transfer of the control has the same function as the machine language CALL command. Therefore, if a RET command is in the machine language program (201 by the decimal code), it returns to the BASIC program.

```
330 USR (AD)
```

Executes CALL the decimal address designated by the variable AD.

```
360 USR ($C000)
Hexadecimal system
```

Executes CALL the hexadecimal address C000.

```
40 WOPEN #8, USR ($C000)
50 PRINT #8, A$
60 CLOSE #8
```

Defines USR ($C000) to the logical number #8 and open to write. With the statement number 50 the top address of the buffer, where the contents of the string variable A$ is set, is set to the DE register, sets the data length (max. 255 bytes) to the BC register, and executes USR ($C000).

String Processing Statements

LEFT$

```
300 A$ = LEFT$ (X$, N)
```

Subsitutes the first to Nth character of string variable X$ for string varaible A$. N can be either constant, variable or equation.

RIGHT$

```
600 B$ = RIGHT$ (X$, N)
```

Substitutes the last N characters of string variable X$ for string variable B$. N can be either constant, variable or equation.

MID$

```
900 C$ = MID$ (M$, M, N)
```

Substitutes the N characters from Mth character of string variable X$ for string variable C$. M and N can be either constant, variable or equation.

LEN	100 LX = LEN (X$) 110 LS = LEN (X$ + Y$)	Substitutes character length (No. of characters) of string variable X$ for variable LX. Substitutes the character length sum of string variables X$ and Y$ for variable LS.
ASC	200 A = ASC (X$)	Substitutes value of ASCII code (decimal), for the first character of string variable X$ for variable A.
CHR$	220 X1$ = CHR$ (A)	Contrary to ASC statement, this substitutes ASCII code character equal to value of variable A for string variable X1$. A can be either constant, variable or equation.
VAL	400 K = VAL (N$)	Substitutes numeric string of string variable N$ for variable K as value.
STR$	440 N$ = STR$ (K)	Contrary to VAL statement, this substitutes numeral of variable N$ as string.

Functions

ABS	100 A = ABS (X)	Substitutes absolute value IXI of variable X for variable A. Parenthesis can be either constant or equation. Example: ABS (−3) = 3 ABS (12) = 12
SGN	100 A = SGN (X)	Substitutes −1 if value of variable X is < 0, 0 if X = 0, and 1 if $X > 0$, for variable A. Parenthesis can either be constant or equation. Example: SGN(0.4) = 1 SGN(0) = 0 SGN(−400) = −1
INT	100 A = INT (X)	Determines maximum integer not exceeding X of value of variable X, and substitutes it for variable A. Parenthesis can either be constant or equation. Example: INT(3.87) = 3 INT(0.6) = 0 INT(−3.87) = −4
SIN	100 A = SIN (X) 110 A = SIN(30∗π/180)	Determines sin X value for value of variable X (radian) and substitutes it for variable A. Parenthesis can either be constant or equation. Since the relationship between the radian and degree is, $1 \text{ degree} = \frac{\pi}{180} \text{ radian}$ the substitution of sin 30° for variable A, for example, should be made as at statement number 110.
COS	200 A = COS (X) 210 A = COS (200∗π/180)	Determines cosX value for value of variable X (radian), and substitutes it for variable A. Parenthesis can either be constant or equation. Calculation in degrees can be made in a similar manner to SIN function. Statement number 210 is to substitute value of cos200° for variable A.
TAN	300 A = TAN (X) 310 A = TAN (Y∗π/18)	Determines tanX value for variable X value (radian) and substitutes it for variable A. Parenthesis can either be constant or equation. Calculations in degrees can be made in a similar manner to SIN function. Statement number 310 is a statement for substitution of tanY° value for variable A.

ATN	400 A = ATN (X)	Determines $\tan^{-1}$ X value (radian) for variable X value and substitutes it for variable A. Parenthesis can either be constant or equation. Calculation result is the value between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$.
	410 A = 180/π∗ATN (X)	Statement number 410 is a statement for conversion of $\tan^{-1}$ X value to degrees and substitutes it for variable A.
SQR	100 A = SQR (X)	Determines square root of X for variable X value and substitutes it for variable A. Parenthesis can either be constant or equation, but value must be plus or 0.
EXP	100 A = EXP (X)	Determines value of e^X (e to the Xth power) for variable X value and substitutes it for varaible A. Parenthesis can either be constant to equation.
LOG	100 A = LOG (X)	Determines common logarithms $\log_{10}X$ value for variable X value and substitutes it for variable A. Parenthesis can either be constant or equation, but must be a plus value.
LN	100 A = LN (X) 110 A = LOG (X)/LOG (Y) 120 A = LN (X)/LN (Y)	Determines natural logarithms logeX value for variable X value and substitutes it for variable A. Parenthesis can either be constant or equation, but must be a plus value. To determine logarithms logyX with bottom of Y, statement number 110 or 120 can be used.
RND		This function generates random numbers with values from 0.00000001 to 0.99999999. There are two ways of processing, one with 0 or minus integer given in parenthesis and the other with plus integer given.
	100 A = RND (0) 110 B = RND (−3)	With 0 or minus integer given in parenthesis as at statement number 100 or 110, random number generation is initialized to generate a specified value at all times, and the same value is substituted for both A and B.
	200 A = RND (1) 210 B = RND (10)	With plus integer given in the parenthesis as at statement number 200 or 210, random number with value between 0.00000001 and 0.99999999 is generated, whenever RND function is used. In this case, however, this has nothing to do with the value of plus integer given in the parenthesis.

Arithmetic Operators

Numeral white on black base at left shows calculation priority order. Calculations in parentheses are further given priority.

	=	10 A = X + 3 (Substitution) 20 B = π	Substitutes 3+ variable X value for variable A. Substitutes π (3.1415927) value for variable B.
❶	↑	10 A = X ↑ Y (Power)	Substitutes calculation result of X^Y for variable A. (However, if X ↑ Y with X of minus value and Y of no integer, an error occurs.)
❷	−	10 A = −B (Minus sign)	0 −B is a subtraction, but "−" of −B is a minus sign, so please note.
❸	∗	10 A = X∗Y (Multiplication)	Substitutes product of X and Y values for variable A.
❸	/	10 A = X/Y (Division)	Substitutes quotient of X and Y values for variable A.
❹	+	10 A = X + Y (Addition)	Substitutes total of X and Y values for variable.
❹	−	10 A = X − Y (Subtraction)	Substitutes remainder of X and Y values for variable A.

Logical Operators

=	10 IF A = X THEN . . .	Executes statement after THEN if variable A and X value are equal.
	20 IF A$ = "XYZ" THEN . . . (Equal)	Executes statement after THEN if the contents of string variable A are equal to string XYZ.
<> or ><	10 IF A <> X THEN . . . (Not equal)	Executes statement after THEN if variable A and X value are not equal.
>= or =>	10 IF A >= X THEN . . . (Greater than or equal)	Executes statement after THEN if variable A is greater than or equal to X.
<= or =<	10 IF A <= T THEN . . . (Less than or equal)	Executes statement after THEN if variable A is less than or equal to X.
*	10 IF (A > X) * (B > Y) THEN . . . (Logical multiply)	Executes statement after THEN if variable A is greater than X and variable B is greater than Y.
+	10 IF (A > X) + (B > Y) THEN . . . (Logical add)	Executes statement after THEN if variable A is greater than X or variable B is greater than Y.

Other Symbols

?	200 ? "A + B"; A + B 210 PRINT "A + B ="; A + B	This can be used in substitution for PRINT statement. Therefore, statement numbers 200 and 210 are identical.
:	220 A = X:B = X ↑ 2: PRINT A, B	Symbol representing pause in statement and is used for multi-statement. The multi-statement at statement number 220 includes 3 statements.
;	230 PRINT "AB";"CD";"E"	Executes PRINT statement continuously. At statement number 230, "ABCDE" is displayed on CRT screen without space.
	240 INPUT "X ="; X$	Display "X=" on CRT screen, and waits for string variable X$ to be keyed in.
,	250 PRINT "AB", "CE", "E"	Executes PRINT statement with tabulation. At statement number 250, this displays AB first on CRT screen, then CD at the position 10 characters to the right from A and finally E at the position 10 characters to the right from C.
	260 DIM A (20), B (30)	Example in which the comma is used for variable's separation.
"	270 A$ = "SHARP BASIC" 280 PRINT "*"; AS; "*"	Indicates that a string is between quotation marks " ".
$	290 A1$ = LEFT$ (A$, 5) 300 A2$ = "MZ –80K"	Indicates string variable.
π	400 S = SIN (X * π/180)	Circular constant 3.1415927 is represented by π.

DISK BASIC SP-6015 ERROR LIST

Error No.	Nature	Meaning
1	Statement	Syntax error
2	Numerical data	Overflow in numercial figure or arithmetical result
3	Numerical data	Illegal numerical figure or variable
4	Data, variable	Mismatching variable form
5	String length	String exceeding 255 bytes
6	Memory	Out of memory capacity
7	Dimension	The same array variable is defined larger than before.
8	Statement	Overflow of a BASIC text beyond the given length.
10	GOSUB nesting	GOSUB exceeding 16 levels
11	FOR-NEXT nesting	FOR-NEXT exceeding 16 levels
12	Function nesting	Functional definition over 6 levels
13	FOR-NEXT	Illogical NEXT
14	GOSUB-RETURN	Illogical RETURN
15	Defined function	Undefined function
16	Line number	Designation of non-existing statement number
17	Continue	CONT not executable
18	Memory area	Write request to the BASIC management area
19	Command	Mixed use of the direct command and statement
20	Resumption	RESUME not executable
21	Resumption	Illogical RESUME
24	READ – DATA	Illogical READ
25	SWAP level	SWAP exceeding 1 level
40	File	Reference is made to a non-existing file
41	Data	Occurrence of disk hardware error
42	File	Tried to register under an already existing file name
43	File	Executed OPEN, DELETE, or RENAME for a file already open.
44	File	Made reference to, or executed CLOSE or KILL to an unopened file
46	File, diskette	Write protected file
50	Disk system	Disk not ready for the system
51	File	Tried to register files exceeding 94
52	Disk volume	Error regarding volume number
53	Memory of diskette	Lack of space in the diskette
54	Diskette	Diskette without initialization
56	Data	Data error in FDC routine call
57	Diskette	Unusable medium
58	Diskette	Tried to initialize the master diskette
60	File name	File name error
63	File mode	File mode error
64	Logical number	Illegal logical number
65	Printer	Either the printer is OFF state or disconnected
66	Printer	Mechanical trouble with the printer
67	Printer	Paper has run out of the printer
70	Check sum	Check sum error

ASCII code table

- In the table given below, the upper 4 bits correspond to the collumn while the lower 4 bits correspond to the line. For example, the ASCII code of character "A" is 41H in hexadecimal code, and "Z" is 5AH.
- However, the codes 11H to 16H in the table are the ASCII codes for cursor control. For instance, when the content of ACC is 15H, CALL PRNT (monitor subroutine) causes the cursor to return home (not displaying "H").

ASC II

LSD \ MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000		SP	0	@	P			SP		q	n	SP			
1	0001		!	1	A	Q					a					
2	0010		"	2	B	R				e	z	Ü				
3	0011		#	3	C	S					w	m				
4	0100		$	4	D	T					s					
5	0101		%	5	E	U					u					
6	0110		&	6	F	V				t	i					
7	0111		'	7	G	W				g		o				
8	1000		(	8	H	X				h	Ö					
9	1001		)	9	I	Y					k	Ä				
A	1010		*	:	J	Z					b	f	ö			
B	1011		+	;	K	[				x	v	ä				£
C	1100		,	<	L	\				d						
D	1101	CR	-	=	M	]				r	ü	y				
E	1110		.	>	N					p	ß					
F	1111		/	?	O					c	j					

Display code table

- The display code is the code number to call a character in the character generator.
  The character display on the CRT screen is always made by transferring this code into video RAM.
- In "PRNT (0012H)" routine and "MSG (0015H)" rountine of monitor subroutine, the ASCII code is converted into this display code, and the code is transferred to the position of video RAM indicated by the cursor.
  Codes C1H to C6H in the list are the characters for cursor control.

DISPLAY

LSD \ MSD	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0 0000	SP	P	0		SP		π		SP	p						SP
1 0001	A	Q	1				!		a	q						
2 0010	B	R	2				"		b	r						
3 0011	C	S	3				#		c	s						
4 0100	D	T	4				$		d	t						
5 0101	E	U	5				%		e	u						
6 0110	F	V	6				&		f	v						
7 0111	G	W	7				'		g	w						
8 1000	H	X	8				(		h	x						
9 1001	I	Y	9				)		i	y						
A 1010	J	Z					+		j	z	ß					
B 1011	K	£					*		k	ä	ü					
C 1100	L								l		ö					
D 1101	M								m		Ü					
E 1110	N								n		Ä					
F 1111	O								o		Ö					

How to use utility programs

The master diskette, as we have said earlier, is laden with not only Disk Basic programs but application programs (at BTX mode or at OBJ mode) and utility programs as well: Photo 1 shows what versions the application and utility programs provide — they are also displayed in the directory of the master diskette.

You should be well aware of using the Basic Text (BTX mode), so we shall proceed to the next discussion — on using the utility programs and is the first topic about the "slave diskette initialize".

```
**  MONITOR SP-1002  **
#FD
BOOT DRIVE ?

 * SHARP BASIC SP-6015
 27188 BYTES
READY
DIR
VOL.MASTER(S.673)
OBJ*   "S-DISKETTE INIT"
OBJ*   "FILING & CMT"
OBJ*   "DISKETTE COPY"
OBJ*   "BASIC SP-5025"
BTX    "PROLOGUE"
BTX    "FASHION"
BTX    "CLOCK"
BTX    "TAKE-DOWN GAME"
READY
```

Photo 1

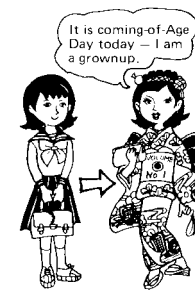
S-DISKETTE INIT (Initialize of Slave Diskette)

This is a utility program that enables an initialize operation of the slave diskette, and is in fact intended to subject the slave diskette to a formation to meet Disk Basic application of the master diskette. This program operation is needed each time the slave diskette is renewed.

Among the commands shown in the master diskette directory, there is:

RUN "S–DISKETTE INIT"

└ The asterisk * here must be erased.



If this command is executed, then its utility program is read entering a new mode of the program control, and the display shown in Photo 2 will arise.

The initialize operation is for the purpose of a formatting the diskette as well as its diskette its volume number setting.

Photo 3 illustrates an example: The slave diskette which has been set in drive No. 1 is initialized to make its volume number 10. As the operation ends, "END" is displayed and the machine returns to its command-ready-to-accept mode: push "!" key to return it to the monitor mode. An error, if caused in the operation, can be cleared by pushing [SHIFT] and [BREAK] keys.

The volume number available ranges from 1 to 127 any of which can be chosen as desired. Note, however, that either diskette shall be given a different number from the other. This means that in the logical open command, for instance, that different number setting can differentiate the diskettes (to be subject to the open command) among each other clearly enough, aided by another command "@V".

Again, this operation is also induced even for the slave diskette that is just in process; at the time all the files, whether locked or not locked, will vanish, however. No initialize operation is possible with the master diskette.

```
* SLAVE-DISK INITIALIZE (79/12/15) *
DRIVE NO.(1-4)[!-MONITOR] ?
```

Photo 2

```
* SLAVE-DISK INITIALIZE (79/12/15) *
DRIVE NO.(1-4)[!-MONITOR] ?1
VOLUME NO.?119
END
DRIVE NO.(1-4)[!-MONITOR] ?
```

Photo 3

FILING←CMT (Filing of Cassette Tape Data onto Diskette)

This is a utility program which causes the data in the cassette tape to be recorded on the slave diskette, and there are two ways of its operation, **"TRANSFER"** and **"CONVERT"**.

Through "TRANSFER" program it is allowed for OBJ and BSD data in the cassette tape to be transferred without any change of them, to the diskette. On the other hand, it is through "CONVERT" program that OBJ data of the cassette tape is once converted into BSD data and then recorded on the diskette — on a converting basis that 1 byte of the data is expressed as 2 byte ASCII code in the hexadecimal notation.

Working with "TRANSFER" program:

Let's consider a machine language "GAME 1" filed in the cassette tape (CMT) which has to be transferred to the slave diskette. The first to do is, of course, to initiate a running of "FILING ← CMT" utility file in the master diskette. The next will be as follows, provided that the slave diskette to do recording onto has been located at the drive No. 1.

```
* TRANSFER FROM CMT TO FD *
(1) FD <- - CMT (!) MONITOR ? 1          ..... "1" is selected.
DRIVE NO. (1-4) ? 1                      ..... Drive No. 1 is designated.
↓PLAY
FILENAME  : GAME 1
FILEMODE  : 1                            | File information of the cassette
BYTE SIZE : 2177                         | tape is available.
DATA ADR. : 4808
EXEC ADR. : 0000
(T) TRANSFER (C) CONVERT (S) SKIP ? T    ..... "TRANSFER" is to be selected.
```

Whether you have to work with "TRANSFER" or with "CONVERT" program can be decided only after you have watched the file information of the cassette tape — on the master diskette's directory. "SKIP" here is the program that makes one cassette tape file skip to another.

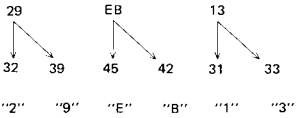
Now everything will go — by merely putting the unit in RUN "GAME 1" mode, then the machine language (OBJ) program "GAME 1" filed in the diskette will be executed.

Working with "CONVERT" program

The example here is to illustrate the case where the machine language (OBJ) program "DISPLAY CONTROL" has to be recorded onto the slave diskette whose drive has been assigned No. 1.

```
* TRANSFER FROM CMT TO FD *
(1) FD <- - CMT (!) MONITOR ? 1          ..... "1" is selected.
DRIVE NO. (1-4) ? 2                      ..... Drive No. 2 is designated.
↓PLAY
FILENAME  : DISPLAY CONTROL
FILEMODE  : 1
BYTE SIZE : 3100
DATA ADR. : 9000
EXEC ADR. : 9000
(T) TRANSFER (C) CONVERT (S) SKIP ? C    ..... "CONVERT" is to be selected.
```

It is through "CONVERT" operation that 1 byte of the machine language code is converted into 2 byte ASCII code in the hexadecimal notation meaning that it further becomes Basic sequencial access data when put into the slave diskette. This means, another way, that the 3 bytes of machine language data is transformed into ASCII code shown below.



·····OBJ data (3 bytes)

·····ASCII code row (6 bytes)
(Sequential access data)

In the first coming BSD block (124 bytes to make up a string of the input # statement) is, however, copied information data for the file.

Application:

The OBJ program which has thus been subjected to the conversion comes to be now BSD program capable of expression in ASCII code, leading to the following program you can perform a list dump in object code.

Input of the file name works in the statement No. 120 and ROPEN does in the statement No. 130. And it is the INPUT # statement (part of the statement No. 140) where the string variable A$ is substituted by the 124 byte string which comprises the file information taking place as the result of the said conversion.

This 124 byte string has a configuration like:

The area ranging from the statement No. 150 to No. 200 represents a combined set of the information display and the hexadecimal notation addresses. In the succeeding area, ASCII data is read in B$ at the intervals of 128 bytes until they reach the file end – with their display as a memory list dump.

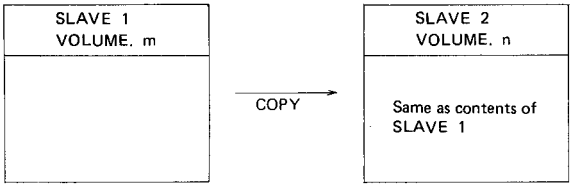
```
100 DIM AD ($) : D1 = 0 : D2 = 8
110 PRINT "**OBJ CODE DUMP LIST**"
120 INPUT "FILENAME ? " ; A$
130 ROPEN #6, A$
140 INPUT #6, A$
150 A1$ = MID$ (A$, 5, 4) : FOR K = 1 TO 4 : AD (K) = ASC (MID$ (A1$, K, 1) ) : NEXT
160 PRINT "■"
170 PRINT "*OBJ CODE DUMP LIST*"
180 PRINT "BYTE $" ; MID$ (A$, 1, 4)
190 PRINT "ADR  $" ; A1$
200 PRINT "EXCE $" ; MID$ (A$, 9, 4)
300 INPUT #6, B$
310 IF EOF (#6) THEN PRINT : CLOSE : END
320 FOR J = 1 TO LEN(B$) – 1 STEP 2
330 IF D2 = 8 GOSUB 600
340 PRINT "  " ; MID$ (B$, J, 2);
350 D2 = D2 + 1 : NEXT : GOTO 300
600 PRINT
610 1F D1 = 1 GOSUB 800
620 D1 = 1 : D2 = O : PRINT "$" ; A1$ ; : RETURN
800 IF AD (4) < 50 THEN AD (4) = AD (4) + 8 : GOTO 840
801 IF AD (4) < 56 THEN AD (4) = AD (4) + 15 : GOTO 840
802 IF AD (4) < 58 THEN AD (4) = AD (4) –8 : GOTO 809
803 AD (4) = AD (4) –15
809 FOR K = 3 TO 1 STEP –1
810 IF AD (K) = ASC ("9") THEN AD (K) = ASC ("A") : GOTO 840
820 IF AD (K) <> ASC ("F") THEN AD (K) = AD (K) + 1: GOTO 840
830 AD (K) = ASC ("0") : NEXT
840 A1$ = "" : FOR K = 1 TO 4:  A1$ = A1$ + CHR$ (AD (K)) : NEXT : RETURN
```

DISKETTE COPY

By using this utility program it is possible to copy the contents of a slave diskette to another diskette: there is no positional change between the two diskettes – any file keeps on the same track and sector position. To cite an example; Contents of a diskette assigned the drive No. 1 are to be copied into another diskette assigned the drive No. 2.

```
* DISKETTE COPY (79/12/15) *
FROM DRIVE NO. (1–4) (! – MONITOR) ? 1
TO DRIVE NO. (1–4) ? 2
```

As the copy operation ends, "END" is displayed.



In this "DISKETTE COPY", take care of that when contents of the SLAVE 1 are copied into the SLAVE 2 all the files if existent in the SLAVE 2 are erased whether they have been locked or not. Still, it is impossible to copy the very Disk Basic of the master diskette into any slave diskette at all.

Meaning of utility program error indication

INHIBIT INITIALIZE.....If the initialize operation is attempted for the master diskette.

NOT READY.....If the drive is not yet READY or if write-in operation is tried for the diskette subjected to a write-protect treatment.

FILE ALREADY EXISTS.....Means, in the filing, that the attempted file is the same as the file already existent in the diskette.

UNFORMAT.....If the diskette which has not yet be initialized is subjected to the filing or copying.

INHIBIT COPY TO MASTER DISKETTE
.....If the master diskette encounters the file-copying.

NO USABLE DISKETTE.....Means the media in use is rejected.

CHECK SUM.....Means a check sum error.

NO FILE SPACE.....Means that no further filing is available.

TOO MANY FILES.....If 94 or more files are entered.

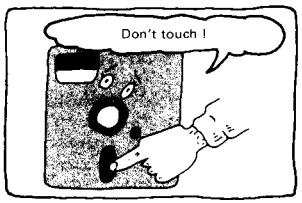
NO MEMORY SPACE.....Means that usable memory area is lacking.

Handling the diskette

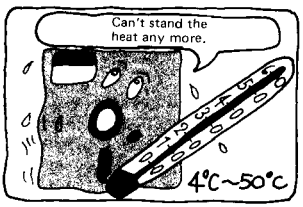
For more of operating the diskette or Floppy Disk, consult the MZ-80FD Instruction Manual.

When handling the master diskette, you should use great care, and when using the SHARP option diskette it should be initialized without fail beforehand.

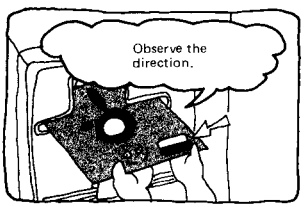
1. Notes on the use of diskette:



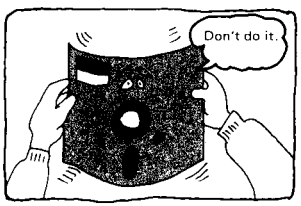
Making fingerprints on the diskette from the head window makes reading and writing difficult. Never soil the diskette surface.



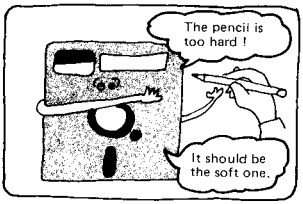
Storage temperature: 4°C to 50°C (ambient temperature).
Do not expose it to the direct sunlight, nor put it in a place whose temperature is over 50°C, or the diskette is deformed and damaged.



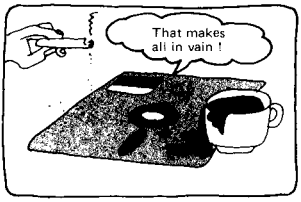
Insert the diskette straight in until it stops. Then close the front door carefully. Rough handling damages the diskette.



Do not bend or fold the diskette. A deformed diskette makes reading and writing difficult.

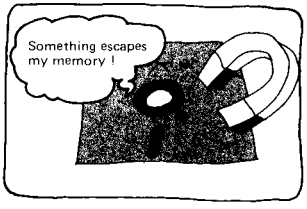


The index label should be filled in before it is attached to the jacket. If it is filled in after it has been attached to the jacket, use a soft-point pen. Do not use a ball-point pen, pencial or other hard-point writing articles.

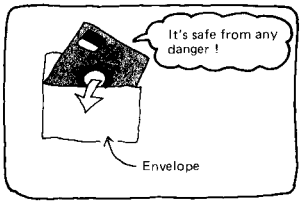


Do not handle the diskette or Floppy Disk while you smoke or drink. In case the diskette is soiled, data program stored in the diskette may be spoiled.

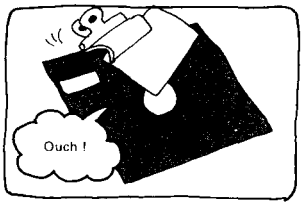
2. Notes on the storage of diskette:



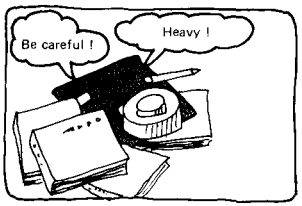
Never put a magnet near the diskette. A magnet ring or magnet necklace may also affect the stored data. Be sure not to use magnetic objects when the diskette is handled. Any apparatus that generates magnetic fields should be also kept from the diskette; for example, a monitor screen of a computer, cassette tape recorder, home television set, etc.



When not in use, be sure to keep the diskette in the envelope. Form a habit of putting the diskette into the envelope immediately after it is taken out of the drive. Such a habit will prevent nearly all problems resulted from wrong diskette handling. The master diskette should be loaded on the Floppy Disk only when it is put in use; and when not in use keep it in the envelope with great care. The envelope is made of special material which works against static electricity, humidity, etc. So, make sure again all the diskettes are in the envelopes.



Do not use a paper holder or paper clip, in a way as shown in the figure.



Do not put heavy object (s) on the diskette. Do not put it on a desk or table carelessly to avoid this. Keep the diskette in its specified place whenever it is not used.

Specifications

Memory capacity	143K bytes/drive (286K bytes/unit)
No. of tracks	70
No. of sectors	16 (per track)
Working conditions	4 to 25°C
	20 to 80% (relative humidity)
Rated voltage	Local voltage, 50Hz
Power consumption	40 W
Outer dimensions	200 mm (W) x 320 mm (D) x 200 mm (H)
Weight	7.9 kg

Specifications subject to change without prior notice for improvement.

MEMO

Printed in Japan
8010-600(E1)

SERIES MZ-80

SHARP CORPORATION