1.6.2 Editing a program

You can edit your programs quickly and easily by using the screen editor. To do this, display the program to be edited on the screen, by using the LIST or EDIT command. Then move the cursor to the location you want to edit your program text. You must edit only one program line at a time, and press the $\boxed{\text{RETURN}}$ key after completing the editing. If you forget to press the $\boxed{\text{RETURN}}$ key, the line in the program memory will remain unchanged even though the line on the screen appears to be altered.

Chapter 2

PROGRAMMING

2.1 Program Lines and Line Numbers

BASIC regards a program as comprising a collection of lines. Each line consists of a line number and a statement. For example, when you key in

$$\underbrace{10}_{\text{Line number}} \quad \underbrace{A=10}_{\text{Statement}} \quad \boxed{\text{RETURN}}$$

from the keyboard, BASIC stores the statement "A = 10" in program memory with a line number of 10 as its label. When you execute the RUN command, BASIC interprets and executes the program lines in program memory sequentially in numerical order.

You must specify the destination of the GOTO or GOSUB statement with a line number. You must also specify the line number when modifying, deleting, or printing a portion of a program.

NOTE:
BASIC executes all program lines in numerical order unless the program flow is changed by the GOTO, GOSUB, or IF...THEN...ELSE statement.

A line can be a maximum of 255 characters long including the line number. Within this limit, a line can contain two or more statements, which must be separated by colons (:). Line numbers must be integers between 0 and 65529.

The use of the AUTO command is convenient when entering two or more program lines. When the AUTO command is executed, BASIC will automatically generate a new line number each time you press the $\boxed{\text{RETURN}}$ key, saving you from having to enter line numbers. To terminate the AUTO command, press the $\boxed{\text{CTRL}}$ + $\boxed{\text{C}}$ or $\boxed{\text{STOP}}$ (see the description of the AUTO command).

2.2 Constants and Variables

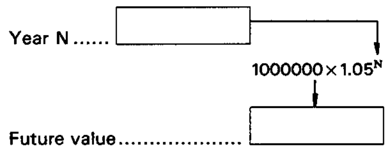
BASIC handles two types of data: constants and variables. Constants represent with numeric or alphabetic characters the actual values of some object (e.g., length, weight, amount of money, etc.). Variables are labels used to represent values.

Consider an example of depositing one million in a bank at an annual interest rate of 5 percent over a period of three years. The future value of the investment can be calculated using the following formula:

$$\textbf{(Total value after 3 years)} = \textbf{(Principal)} \times \textbf{(1 + Interest/100)\^{}(Year)}$$
$$= \mathbf{1{,}000{,}000 \times 1.05^3}$$

Let us use the variable N for the compounding year and S for the future value after N years. The above formula can be expressed as shown below where the principal 1000000 and interest rate 1.05 are constants.

$$\mathbf{S = 1{,}000{,}000 \times 1.05^N}$$



Using the above formula, you can find not only the future value after three years but also future values after any years by assigning appropriate values to N. The program will look like this:

`10 INPUT N`	Gets the year.
`20 S=1000000*1.05^N`	Calculates future value after N years and assigns the result to S.
`30 PRINT S`	Displays the value of S on the screen.
`40 END`	Terminates the program.

After entering the above program, key in:

```
RUN
```

BASIC will then run the program and display the prompt:

```
? ■
```

Eter a year.

```
? 3■
```

Press the $\boxed{\text{RETURN}}$ key, and the program evaluates the expression then displays the result on the screen as follows:

```
1.15762E+06
```

NOTE:
1.15762E+06 represents 1.15762×10^6.

You can apply the above program to various situations by assigning the variable F to the principal and R to the interest rate. To do so, modify lines 10 and 20 as follows:

```
10 INPUT N,F,R
20 S=F*(1+R/100)^N
```

Run the program using the RUN command. Enter variable values in the order N (year), F (principal), and R (interest rate in %), separated by commas. For example:

```
? 3,5000000,5■
```

The use of constants and variables are described on the following pages.

2.2.1 Constants

Constants are classified as follows:

- String constants
- Numeric constants
  - Integer type
    - Decimal
    - Octal
    - Hexadecimal
  - Real type
    - Single precision
    - Double precision

(1) String constants

String constants are enclosed in double quotation marks when they are entered into a program. The CHR$ function must be used to handle double quotation marks as string constants. String constants must not exceed 255 characters.

Examples:

```
PRINT "HELLO"····HELLO
PRINT "I SAID ";CHR$(34);"HELLO";CHR$(34)
      ···I SAID "HELLO"
```

(2) Numeric type constants

Numeric constants are either positive or negative numbers or 0. Negative numbers must always be preceded by a minus (−) sign but positive numbers can be preceded by a plus (+) sign simply when desired.

①Integer constants

Integer type constants are integers from −32768 to +32767. Use one of the following forms to represent integers:

- Decimal form: Decimal numbers from 0 to 9 followed by a % mark.
- Octal form: Octal numbers from 0 to 7 preceded by & or &O. The range of values that BASIC can handle is from &0 to &177777.
- Hexadecimal form: Integers represented by characters 0 to 9 and A to F preceded by &H. A, B, C, D, E, and F represent 10, 11, 12, 13, 14, and 15 in decimal, respectively. The range of values that BASIC can handle is from &H0 to &HFFFF.

②Real type constants

Real type constants can be subdivided into single- and double-precision constants.

(i) Single-precision constants have three forms. Single-precision constants are rounded to 6 digits for display or printout.
 (a) Numbers with not more than seven significant digits
  Example: 3489.0
 (b) Numbers represented using E (the exponent can take values from −38 to 37).
  Example: $\underbrace{235.988}_{\text{Mantissa}}\text{E}\underbrace{-7}_{\text{Exponent}} = 235.988 \times 10^{-7} = 0.0000235988$
 (c) Numbers followed by a !
  Example: 279.9!

(ii) Double-precision constants have three forms. BASIC displays or prints double-precision constants as 16 digits.
 (a) Numbers with 8 to 16 significant digits
  Example: 345692811
 (b) Numbers represented using D (the exponent can take values from −38 to 37).
  Example: $-1.09432\text{D}-6 = -(1.09432 \times 10^{-6}) = -0.00000109432$
 (c) Numbers followed by a #
  Example: 3489.0#

Precision of numeric data

The range of values that can be internally represented inside the computer is limited. The precision of numeric values increases as the range of values to be taken increases. PX-4 BASIC supports the following types of precision:

- Integer type
- Single precision
- Double precision

Among numbers of the above precision types, double-precision real numbers provide the highest precision but occupy the most memory space and take the most processing time. If you know in advance that your program uses only integers, you need not use single- or double-precision representation. Use the most appropriate data representation according to the processing to be performed.

2.2.2 Variables

(1) Variable names

A variable name is a sequence of 1 to 40 alphanumeric characters which begins with an alphabetic character. BASIC does not distinguish between upper- and lower-case alphabetic characters. Reserved words cannot be used as variable names. Reserved words are keywords which are used to represent BASIC statements and functions (see Appendix I "RESERVED WORDS"). If a reserved word is used as a variable name, BASIC displays an error message "SN Error " on the screen and terminates the execution. The first two letters of variable names must not be FN. If a variable name beginning with FN is used, BASIC will call a user-defined function.

(2) Variable types

You must specify types of variables as with constants. BASIC treats variables having the same variable name but with and without a type declaration character as different variables. The type of a variable is identified by the type declaration character at the end of each variable. Variables with no type declaration character are assumed to be of single precision type.

You can declare a variable type without a type declaration character, by using type declaration statements such as DEFINT, DEFSTR, DEFSNG, and DEFDBL. For details on these statements, see Chapter 3 "COMMANDS AND STATEMENTS."

The BASIC type declaration characters for variables are described below.

① $: String variable
  Used to identify variables which are to be loaded with character strings.
  Example: A$ = "ABC"

② %: Integer variable
  Used to identify variables which are to be loaded with integers.
  Example: A% = 300

③ !: Single-precision variable
  Used to identify variables which are to be loaded with single-precision real numbers.
  Example: A! = 12.34

④ #: Double-precision variables
  Used to identify variables which are to be loaded with double-precision real number.
  Example: A# = 12.3400001525

(3) Array variable

When data items are to be processed in a program, as many variables as the data items must be prepared. When handling data items of similar type, it is often convenient to give a single name to the group of data items by using variable to refer to them collectively. In this case, each data item can be identified using one or more index expressions. A variable used to refer to such a group or table of data items is called an array variable. The number of data items (elements) of an array variable is specified by placing the number (for a single-dimension table) enclosed in parentheses after the variable name. Array variables are declared by using the DIM statement. For example, a group of ten string variables can be collectively allocated by declaring an array variable of ten elements by using the DIM A$(9) statement. This statement has the same effect as declaring ten separately named string variables A$(0) to A$(9).

The number enclosed in parentheses is called a subscript of the array variable. You can specify the base of the subscript, i.e., whether the subscript begins with 0 or 1, by using the OPTION BASE command. The default base value is 0 (see the description of the OPTION BASE command).

A$(0)	A$(1)	A$(2)	A$(3)	A$(4)	A$(5)	A$(6)	A$(7)	A$(8)	A$(9)

DIM statements for declaring array variables should normally be placed at the beginning of the program.

The ERASE command is used to clear the memory area allocated for array variables.

In the above example, a one-dimensional array variable is shown. You can declare a two-dimensional array variables by specifying two subscripts enclosed in parentheses in the DIM statement. For example, the statement DIM A$(20, 20) causes BASIC to reserve a memory area for 21 × 21 string variables assuming that an OPTION BASE 0 statement has been executed.

A$(0.0)	A$(0.1)	A$(0.2)	....	A$(0.20)
A$(1.0)	A$(1.1)	"	....	
		⋮		
⋮	⋮	⋮	⋮	

You can reserve as large an area for array variables as memory permits. You may also use integer type variables as array subscripts.

BASIC assumes a maximum subscript value of 10 when an array variable is used without the use of a DIM statement.

(4) Type conversion

BASIC will perform type conversion as required when loading numeric values into numeric variables, except between string and numeric type data. If an attempt is made to convert between string and numeric type variables, BASIC displays an error message "TM Error" on the screen and terminates processing.

① When loading numeric variables with constants of another type

Example 1:

```
10 A%=55.8
20 PRINT A%
RUN
 56
Ok
```

When a real type constant is loaded into an integer type variable, it is converted to an integer type by being rounded to the tenth place.

Example 2:

```
10 A#=12.34
20 PRINT A#
run
 12.34000015258789
Ok
```

② Arithmetic operations

Example 1:

```
10 ? 10/3
20 ? 10/3*3#
RUN
 3.33333
 9.999999761581421
Ok
```

The equation 10/3 is performed in single precision. Its result is converted to double precision before being multiplied by 3, that is, the result is converted to the higher degree of precision.

Example 2:

```
10 PRINT 10#/3
20 PRINT 10#/3*3

RUN
 3.333333333333333
 10
Ok
```

The arithmetic 10/3 is performed in double precision. The precision of the result is compared with that of the multiplier 3; consequently, the multiplication is carried out at the higher precision, i.e., double precision.

Example 3:

```
10 A!=10/3
20 B#=10#/3
30 IF A!>B# THEN 50 ———— Comparison is performed at the higher precision,
40 PRINT "10#/3=";B# : END   i.e., double precision
50 PRINT "10/3= ";A!

RUN
10#/3= 3.333333333333333
Ok

PRINT A!
 3.33333
Ok
A#=A!
PRINT A#
 3.333333253860474
Ok
```

③ Type conversion in logical operations

When a noninteger numeric value is issued in an operand of a logical operation, it is converted to integer type by being rounded to the tenth place before the operation begins.

```
10 A=10.5 AND 12.3
20 PRINT A
RUN
 8
Ok
```

2.3 Arithmetic Operations

(1) Expressions
Expressions are defined as variables or constants, or combinations of variables and constants combined with operators to yield a single value.

BASIC provides the following five operators:

- Arithmetic operators
- Relational operators
- Logical operators
- Functional operators
- String operators

(2) Operator types
①Arithmetic

Operator	Operation	Sample expression	Explanation
∧	Exponentiation	A∧B	Places A to the power of B. See (i).
–	Negation	–A	
*	Multiplication	A*B	
/	Division	A/B	See (i).
+	Addition	A+B	
–	Subtraction	A–B	
MOD	Remainder of division	A MOD B	Gives the remainder of A divided by B. See (ii).
\	Integer division	A\B	Gives the quotient of a division, with a truncated fraction. See (iii).

The precedence of the arithmetic operators are as follows:
1. Parentheses
2. Function
3. Exponentiation
4. Signs (unary +, unary –)
5. *, /, \, MOD
6. +, – (binary)

(i) When a division by 0 is performed, BASIC displays a "/0 Error" message on the screen and continues processing. It returns the maximum value that the computer can handle. The same result will be returned if an exponentiation operation results in zero being raised to a negative power.

Example 1:

```
10 A=10/0
20 PRINT A

RUN
/0 Error
 1.70141E+38
Ok
```

Example 2:

```
10 A=0^-2
20 PRINT A

Ok
RUN
/0 Error
 1.70141E+38
Ok
```

(ii) When the divisor or dividend is noninteger, it is rounded to the tenth place, to form an integer before the division is carried out.

```
10 A=10.5 MOD 2.5
20 PRINT A

RUN
 2
Ok
```

(iii) When the divisor or dividend of an integer division is a noninteger, it is rounded to the tenth place to form an integer before the division is carried out, and only the integer part of the quotient is returned.

Example 3:

```
10 A=10¥2.5
20 PRINT A

run
 3
Ok
```

(iv) When the result of an operation cannot fit in the memory area reserved for the variable storing the result, BASIC displays an "OV Error" message on the screen, and continues or terminates processing depending on the type of the operation.

```
10 A=5^100
20 PRINT A

RUN
OV Error
 1.70141E+38
Ok
```

② Relational operators

Operator	Sample expression	Explanation
=	A=B	A is equal to B.
< > or ><	A< > or A><B	6 A is not equal to B.
<	A<B	A is smaller than B
>	A>B	A is greater than B.
<= or <	A<= or A=<B	A is smaller than or equal to B.
>= or =<	A>= or A=>B	A is greater than or equal to B.

The result of relational operations is either true (–1) or false (0). Relational operations are used to alter the program flow depending on various conditions (see the descriptions of the IF...THEN...ELSE and IF...GOTO statements).

For example, in evaluating the expression

X + Y < (T – 1)/Z,

BASIC calculates X + Y and (T-1)/Z then compares the results of the preceding calculations. If the result of the former is smaller than that of the latter, the true value (–1) is returned.

NOTE:
The equal sign is used as a relational operation only in the conditions clause of the IF statement. In addition to being used to indicate equality, the equal sign is also used to indicate the assignment of a value to a variable (see the LET statement).

Example:

```
10 B=7 : C=5
20 A=B>C
30 PRINT A
```

When the above program is executed, –1 is displayed on the screen. On line number 20, the result (–1 or 0) of the relational operation B>C is placed in A. Since B>C is true, when executed this program displays –1 on the screen. Change line number 0 to:

```
10 B=5 : C=7
```

BASIC will display 0 because B<C is false.

③ Logical operators

Logical operators are used in two forms:

(i) <operand> logical operator <operand>

- When the <operand> is either a numeric constant or a variable, the logical operator performs bit manipulation on two or more operands according to the logical relation determined by the operator and returns either 1 or 0 for each bit.

  The logical operators in this form are:

NOT (Negation)

A	NOT A
1	0
0	1

AND (Logical product)

A	B	A AND B
1	1	1
1	0	0
0	1	0
0	0	0

OR (Logical sum)

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

XOR (Exclusive or)

A	B	A XOR B
1	1	0
1	0	1
0	1	1
0	0	0

IMP (Implication)

A	B	A IMP B
1	1	1
1	0	0
0	1	1
0	0	1

EQV (Equivalence)

A	B	A EQV B
1	1	1
1	0	0
0	1	0
0	0	1

<operand> can take values from −32768 to 32767. Any values other than integers are rounded to integers.

The numeric value specified by the <operand> is converted to a 16-bit signed binary number in two's complement form before being submitted to the logical operation. Positive integers from 0 to 32767 are expressed as 0000000000000000 to 0111111111111111. Negative numbers from −1 to −32768 are expressed as 1111111111111111 to 1000000000000000. The bit in the leftmost position indicates the sign of the number. A 0 in this bit identifies a positive number and a 1 a negative number.

```
PRINT NOT 3                    3=0000 0000 0000 0011
-4                      -----------------------------
Ok                      NOT    3=1111 1111 1111 1100

PRINT 1 AND 2                  1=0000 0000 0000 0001
 0                             2=0000 0000 0000 0010
Ok                      -----------------------------
                        1 AND  2=0000 0000 0000 0000

PRINT 3 AND 2                  3=0000 0000 0000 1111
 2                             2=0000 0000 0000 0010
Ok                      -----------------------------
                        3 AND  2=0000 0000 0000 0010

PRINT 6 OR 5                   6=0000 0000 0000 0110
 7                             5=0000 0000 0000 0101
Ok                      -----------------------------
                        6 OR   5=0000 0000 0000 0111

PRINT 2 XOR 5                  2=0000 0000 0000 0010
 7                             5=0000 0000 0000 0101
Ok                      -----------------------------
                        2 XOR  5=0000 0000 0000 0111

PRINT 3 IMP 0                  3=0000 0000 0000 0011
-4                             0=0000 0000 0000 0000
Ok                      -----------------------------
                        3 IMP  0=1111 1111 1111 1100

PRINT 2 EQV 4                  2=0000 0000 0000 0010
-7                             0=0000 0000 0000 0000
Ok                      -----------------------------
                        2 EQV  4=1111 1111 1111 1001
```

(ii) <operand> logical operator <operand>

- When the expressions in the <operand> are connected by relational operators, the logical operator performs logical operations on the results (true or false) of the relational operations and returns a true (−1) or false (0) result. The logical operators in this form are the same as those given in (i).

NOT (Negation)

Result of relational operation(A)	NOT A
True (−1)	False (0)
False (0)	True (−1)

AND (Logical product)

Result of relational operation(A)	Result of relational operation(B)	A AND B
True (−1)	True (−1)	True (−1)
True (−1)	False	(0)False (0)
False (0)	True (−1)	False (0)
False (0)	False (0)	False (0)

OR (Logical sum)

Result of relational operation(A)	Result of relational operation(B)	A AND B
True (−1)	True (−1)	True (−1)
True (−1)	False (0)	True (−1)
False (0)	True (−1)	True (−1)
False (0)	False (0)	False (0)

XOR (Exclusive or)

Result of relational operation(A)	Result of relational operation(B)	A AND B
True (−1)	True (−1)	False (0)
True (−1)	False (0)	True (−1)
False (0)	True (−1)	True (−1)
False (0)	False (0)	False (0)

IMP

Result of relational operation(A)	Result of relational operation(B)	A AND B
True (−1)	True (−1)	True (−1)
True (−1)	False (0)	False (0)
False (0)	True (−1)	True (−1)
False (0)	False (0)	True (−1)

EQV (Equivalence)

Result of relational operation(A)	Result of relational operation(B)	A AND B
True (−1)	True (−1)	True (−1)
True (−1)	False (0)	False (0)
False (0)	True (−1)	False (0)
False (0)	False (0)	True (−1)

```
100 IF D<200 AND F<4 THEN 80
```

The above program line causes control to transfer to line 80 because the results of relational operations D<200 and F<4 are both true and the logical operation performed on these results also gives a true value.

```
100 IF J<10 OR K<0 THEN 50
```

The above program line causes control to transfer to line 50 because the result of the OR operation is true if either J<4 or K<0 is true.

④ Functions

A function performs a predefined operation on the given parameters and returns the result of the operation. For example, SIN(X) returns the sine of numeric value X in radians. PX-4 BASIC provides a number of functions. The BASIC functions are described in detail in Chapter 3.

You can define your own functions by using the DEF FN statement. See Chapter 3 for further information.

⑤ String operators

(i) The "+" string operator concatenates character strings together.

```
10 A$="FILE" : B$="NAME"
20 PRINT A$+B$
30 PRINT "NEW "+A$+" "+B$

RUN
FILENAME
NEW FILE NAME
Ok
```

(ii) Character strings can also be compared in the same way as numeric values are, by using relational operators. BASIC compares two strings by matching them one character at a time starting at the leftmost character position. It regards two characters which match in all character positions as equal. If a mismatch is found in a character position, BASIC terminates the comparison and regards the character string with the higher character code value in that character position as the larger string. When one character string is phsically shorter than the other, and the character positions of both strings match, the longer character string is regarded as larger string. Blanks are also compared in the magnitude of their code (see Appendix J, "CHARACTER CODES").

Example 1:

"AA"<"AB"
"FILENAME"="FILENAME"
"X$">"X#"
"kg">"KG"
"SMYTH"<"SMYHE"

Example 2:

```
10 A$="alpha"
20 B$="beta"
30 IF A$>B$ THEN 60
40 PRINT A$;" is lower than ";B$
50 END
60 PRINT B$;" is lower than ";A$

RUN
alpha is lower than beta
Ok
```

2.4 Files

The PX-4 computer treats its input/output devices as files. This section briefly describes the input/output devices that the PX-4 can handle and also shows how BASIC identifies them.

2.4.1 File Specification

A file is identified by means of a file specification which consists of a drive name, a file name, and a file extension.

(1) Drive name
The drive name identifies the input/output device on which a file is to be mounted for processing. The default drive is the logged in drive.

The relationship between the input/output devices and the drive names is shown below.

A:	RAM disk
B:	ROM capsule-1
C:	ROM capsule-2
D:,E:,F:,G:	Floppy disk
H:	Microcassette
I:	RAM cartridge
J:	ROM cartridge-1
K:	ROM cartridge-2
SCRN:	LCD display
LPT0:	Printer
COM0:	RS-232C
COM1:	SIO
COM2:	RS232C SIO
COM3:	Cartridge serial
KYBD:	Keyboard
CAS0:	External cassette

(2) File name

The file name identifies a file on a single input/output device. A file name is made up of one to eight alphanumeric characters followed by a period and a 1- to 3-character file extension (or file type). When we refer to a file name, we normally include the file extension.

You need not specify a file name for the keyboard, RS-232C port, and those input/output devices which cannot handle more than one file at a time.

(3) File extension

The file extension is usually used to indicate the use of a file (e.g., for storing programs or data). The file extension can consist of one to three alphanumeric characters. The LOAD and SAVE commands assume a file extension of BAS when no file extension is specified. The file name and extension must be separated by a period.

Example:

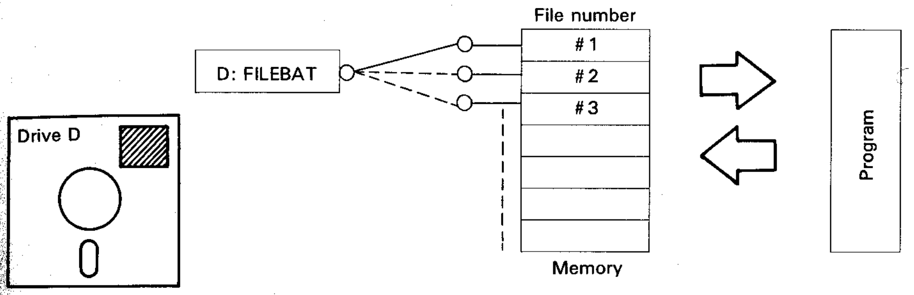
```
SAVE "A:PROG1.BAS"
```

The above command saves the program in the currently selected program area onto the RAM disk with a file name of PROG1 and a file extension of BAS.

2.4.2 File number

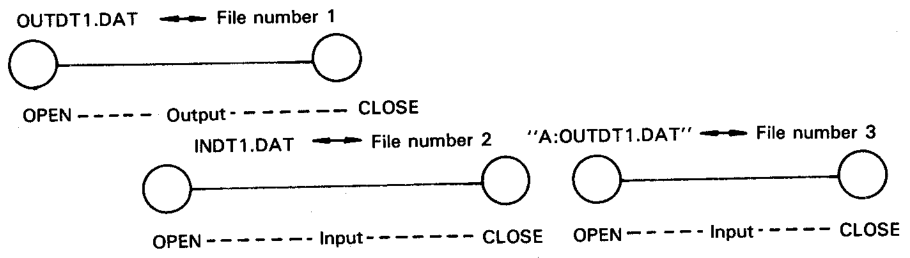
A file must be given a file number by the OPEN statement when it is to be used by a program for input/output processing. The program performs input/output operations to and from the file via the memory area (buffer) associated with the file number that is uniquely assigned to the file. The association between the file and the file number is released by the CLOSE statement.

File numbers begin with 1. The maximum file number is determined by the /F:<number of files> parameter that is specified when BASIC is cold-started. 3 is assumed when the /F: parameter is omitted.



Let us consider an example program for handling input/output. If the /F: parameter (i.e, the number of files that can be opened at a time) is specified as 4 at BASIC start-up time, we can specify integers 1 to 4 as file numbers.

① The first code segment creates a file on the RAM disk with a file name of OUTDT1.DAT, and we have decided to make the file number 1.
② The second code segment reads data from the file INDT1.DATA on the floppy disk. Since file number 1 is already being used for the output file, we must select a second file number for this file from file numbers 2, 3, and 4. Here we have selected file number 2.
③ The third code segment reads data from the file OUTDT1.DAT on the RAM disk. We can use any number from 1 to 4 for this file. We used file number 3.


OUTDT1.DAT ←→ File number 1
OPEN - - - - - Output - - - - - - - CLOSE
INDT1.DAT ←→ File number 2
OPEN - - - - - - - Input - - - - - - - CLOSE
"A:OUTDT1.DAT" ←→ File number 3
OPEN - - - - - Input - - - - - CLOSE

```
100 OPEN "O",#1,"A:OUTDT1.DAT"
 .            .
 .            .
 .            .
 .            .
 .            .
200 OPEN "I",#2,"A:INDT1.DAT"
 .            .
 .            .
250 CLOSE #1
 .            .
 .            .
300 CLOSE #2
 .            .
 .            .
340 OPEN "I",#3,"A:OUTDT1.DAT"
 .            .
 .            .
 .            .
 .            .
 .            .
 .            .
 .            .
 .            .
500 CLOSE #3
```

2.4.3 Input/output devices

The PX-4 computer is furnished with a number of input/output devices. These devices can be accessed by using standardized I/O statements and functions. A brief description of the PX-4 I/O devices follows.

①LCD display
The LCD display is an write-only device which is used to show data and program contents. When using it as a file, specify SCRN: as the drive name.

②Keyboard
The keyboard is an read-only device used to receive data. When using it as a file, specify KYBD: as the drive name.

③RAM disk
The RAM disk is used to store programs and data. The RAM disk is part of the PX-4 main memory which simulates a floppy disk but provides far higher input/output processing speed than normal floppy disk drives. Its drive name is A:. The size of the RAM disk can be specified during system initialization. The contents of the RAM disk are preserved when the PX-4 is powered off.

④RS-232C, SIO, and cartridge serial ports
The PX-4 computer can exchange data with a variety of external devices (e.g., QX-10, PX-8, etc.) via their communications facilities such as the RS-232C, SIO, and cartridge serial interfaces. See Chapter 6, "Data Communications Facilities" for details. The drive names COM0: to COM3: are reserved for these interfaces.

⑤ROM capsule
The ROM capsule is read-only device which reads programs and data stored in the computer. It may also be used as part of main memory. The drive name of the ROM capsule is B: or C:.

2.4.4 Other peripheral devices

The PX-4 computer also includes a buzzer and a clock. These devices cannot be used as files so they are not assigned drive names. The following statements and functions are available for these devices:

Buzzer:	BEEP, SOUND
Clock:	TIME$, DATE$, DAY

2.4.5 Microcassette drive (Optional)

The optional microcassette can be used to store data and programs in the same way as the RAM disk. Not only as a storage device, but also as an input/output device. The microcassette is assigned the drive name of H:.

The PX-4 controls files on the microcassette by using a directory (a table used to store information about the files). It reads this directory into memory from the microcassette tape whenever performing an I/O operation (i.e., read or write) on the microcassette. This procedure is known as mounting. Once the directory is loaded, the PX-4 accesses the files on the microcassette based on the directory information in memory. It alters the contents of the directory in memory each time it writes data onto the microcassette tape. This means that you have to save the contents of the directory back onto the tape before removing the microcassette from the cassette deck following an I/O operation. This procedure is known as remove. The mount and remove procedures can be carried out in the system display mode; however, they may also be performed from within a program by using the BASIC REMOVE and MOUNT commands.

The directory of a new microcassette tape must be initialized (DIRINIT) before use. For detailed handling procedures of the microcassettes, refer to Section 2.8, "Microcassette Handling" in the PX-4 Operating Manual.

Although microcassettes can be opened in the random mode, they do not allow accesss. I/O operations on microcassette random data files must be carried out in record number sequence. The PX-4 can access only one microcassette file at a time. For example, if one microcassette file is opened in the input mode and then another microcassette file is opened, the PX-4 can only access the latter file. Once a file is opened for output, no other files can be opened until the opened file is closed.

The file specification for microcassette files has the following format:

[H:][(<options>)][<file name>][.<extension>]

You can specify two types of options for microcassette files:

① S: Stop mode
   N: Nonstop mode

When this option is omitted, the microcassette file is processed as follows:

(i) When the file is opened in the input ("I") mode
Data is read from the file in the mode in which the data was written.
(ii) When the file is opened in the output ("O") mode
Data is written to the file in the stop mode.
(iii) When the SAVE command is used
Data is written to the file in the nonstop mode.
(iv) When the random ("R") mode is specified in the OPEN statement
If the file name specified in the OPEN statement exists on the microcassette, data is read from that file in the mode in which the data was written. If the OPEN statement is used to create a new file, data is written to the file in the stop mode.

② V: Specifies that when the file is closed, a CRC check is to be made after rewinding the tape (verify mode).
   N: Specifies that no CRC check is to be made (nonverify mode). When this option is omitted, the mode specified in the system display mode is becomes active.

Example:

```
SAVE "H:(SV)TEST.BAS"
```

When specifying only option ②, place a blank in the position where option ① is to be specified.

NOTE:
When an item keyboard is installed on the PX-4, the system display cannot be obtained, so the microcassette drive cannot be controlled manually.

2.4.6 RAM cartridge (Optional)

The RAM cartridge is an external storage device which allows both input and output. Like the RAM disk in the main unit, it is used to store data and programs. The RAM cartridge is optional and connected to the main unit via a cartridge interface.

Since the RAM cartridge is battery-backed up, it can preserve data for up to one year when detached from the main unit. The RAM cartridge has a capacity of 16KB and the assigned drive name is I:.

2.4.7 Printer (Optional)

This optional printer is a write-only device which is used to print data and program listings. It is assigned the drive name LPT0: when used as a file. See Appendix N.

2.4.8 Floppy disk (Optional)

Floppy disks are commonly used auxiliary storage devices for storing data and programs. They can be used for both input and output operations. The floppy disk drives for the PX-4 are given the drive names D:, E:, F:, and G:.

2.4.9 ROM cartridges (Optional)

ROM cartridges are read-only external storage devices which are attached to the main unit via the cartridge interface. The drive names J: and K: are reserved for the ROM cartridges.

2.4.10 External audio cassette

The PX-4 can use a commercial audio cassette tape recorder as an external storage device by connecting it with an optional cable (#732). The audio cassette tape recorder is used as a sequential input/output unit and assigned the drive name CAS0:. Its file specification has the format:

CAS0:[<option>][<file name>][.<extension>]

The option must be either the stop or nonstop mode.

S: Stop mode
N: Nonstop mode

The audio cassette recorder allows no manual operation; it is only controlled through BASIC commands. The BASIC commands that support the audio cassette record are: OPEN, CLOSE, INPUT#, LINE INPUT#, PRINT#, PRINT USING, SAVE, LOAD, LOAD?, LIST, RUN, MERGE, MOTOR, BLOAD, BSAVE, EOF, and INPUT$.

Examples:

```
SAVE "CAS0:PROGRAM.BAS"
SAVE "CAS0:(S)PROGRAM.BAS"
```

- **Connecting a cassette recorder to the PX-4**

The REM (remote) terminal need not be used when saving onto or loading from a cassette tape. When handling a data file, however, data cannot be written to the cassette file unless the REM terminal is used and the S (stop mode) option is specified in the file specification. Since the PX-4 reads data from a cassette file one block at a time, it must be able to control the starting and stopping the cassette recorder with the REM terminal in order to position the read/write head at the next block.

When the REM terminal is activated, the PLAY button on the cassette recorder is disabled and tape will not move when the PLAY button is pressed. To start the recorder when the REM terminal is used and the PLAY button is pressed down, execute the MOTOR ON command. To stop the tape, execute the MOTOR OFF command.

- **Saving a program onto cassette tape**

To save a program onto a cassette tape, press down the REC and PLAY buttons on the tape recorder then execute the following command:

SAVE "CAS0:[(<option>)]<file name>[.<file extension>]"[,A | P]

BAS is assumed if <file extension> is omitted and nonstop mode (N) is assumed if <option> is omitted.

• **Program verification**
After a program is saved on a cassette tape, a check should be made to verify whether the program has been saved properly. To do this, rewind the tape up to the point where the program saving started (by visually checking the tape counter), then press down the PLAY button, and execute the command:

LOAD?[“CAS0:[<file name>.<file extension>]”]

Program verification is not achieved by comparing the program in memory with that which has been saved on tape. Instead, the LOAD? command loads the saved program while making CRC checks, and displays the error message “IO Error” on the screen if it detects a CRC check error.

• **Loading a program**
To load a program, rewind the tape up to the point where the program was saved, press down the PLAY button, and execute the command:

LOAD[“CAS0:[<file name>.<file extension>]”]

BASIC will load the program into memory in the mode in which it was saved if <option> is omitted. If <file name>.<file extension> is omitted, BASIC will load the first file found.

• **Input/output to and from a data file**
When <option> is omitted in an I/O statement executed for a data file, if the file is opened in the input (“I”) open mode, it will be read in the mode in which the file was created. The file will be written in the stop (S) mode if the file has been opened in the output (“O”) open mode.

2.5 Error Messages

BASIC displays error messages whenever it detects errors during execution of BASIC statements, commands, and functions. If an error is detected while executing a BASIC program, BASIC immediately interrupts the program execution and returns to the command mode. You can prevent your program from being interrupted by such errors by including in your program some error handling routines which use the ON ERROR statement and the ERROR and ERL functions. See Appendix K, “ERROR CODES AND MESSAGES” for a full description of the BASIC error codes and messages.