



Cardinal

Cardinal Scale Manufacturing Co.

SmartWeigh
778 Programming
and Simulator Manual
Rev 1.5

8545-M095-O1 Rev D
10/00

PO BOX 151 • WEBB CITY, MO 64870
PH (417) 673-4631 • FAX (417) 673-5001
<http://www.cardinalscale.com>

Printed in USA

INTRODUCTION

Thank you for purchasing our Cardinal SmartWeigh Programmer and Simulator program for the 778 Programmable Weight Indicating Instrument. This program was written with the same Cardinal quality and reliability at our factory in Webb City, Missouri as our scales and indicators. This manual will guide you through installation and operation of your software. Please read it thoroughly before attempting to install or operate the software and keep it handy for future reference.



Trademarks

Any terms or designations used in this manual are used as generalizations unless otherwise noted. Cardinal Scale Manufacturing Company in no way claims rights or obligations from any company that may be referred to directly, by logo or trademark. Use of a term or trademark in this manual should not be regarded as affecting the validity of any trademark or service name.

Windows 95, 98, NT 4.0 and 2000 are registered trademarks of Microsoft Corporation.


All rights reserved. Reproduction or use, without expressed written permission, of editorial or pictorial content, in any manner, is prohibited. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this manual, the Seller assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from use of the information contained herein. All instructions and diagrams have been checked for accuracy and ease of application; however, success and safety in working with tools depend to a great extent upon the individual accuracy, skill and caution. For this reason the Seller is not able to guarantee the result of any procedure contained herein. Nor can they assume responsibility for any damage to property or injury to persons occasioned from the procedures. Persons engaging the procedures do so entirely at their own risk.



REGISTRATION NUMBER _____
DATE OF PURCHASE _____
PURCHASED FROM _____

RETAIN THIS INFORMATION FOR FUTURE USE

PRECAUTIONS
Before using this program, read this manual and pay special attention to all "WARNING" symbols:



IMPORTANT

COMMERCIAL USE OF MODEL 778 APPLICATION PROGRAMS

One of the major advantages of a Cardinal Model 778 Programmable Weight Indicator is its ability to accept special application software programs written by the scale dealer or user. If the application, for which the software program is written, involves the sale or purchase of goods or commodities by weight, it must meet certain requirements. These requirements are established by the National Conference on Weights and Measures and are contained in documents published by the National Institute of Standards and Technology. Enforcement of these requirements is accomplished by local and state weights and measures departments. The purpose of this chapter is to inform the 778 programmer of the existence of these requirements and to identify common mistakes that would prevent your application program from being accepted for commercial use.

What is a NTEP Certificate of Conformance and do I need one?

A National Type Evaluation Program (NTEP) Certificate of Conformance (CC) is a document issued by NTEP that indicates that the device (scale, weight indicator or software program) has been evaluated by an NTEP laboratory and found to comply with the current requirements. If the software program you write is to only be used in a single application, you do not need an NTEP CC. You must, however, have your system evaluated by your local weights and measures inspector before it is placed in use. If your software program will be used in two or more installations, you must secure an NTEP CC for your program

Where do I find the requirements for commercial use of my 778 software program?

The specifications, tolerances and technical requirements for weighing devices (which includes software) are found in the National Institute of Standards and Technology (NIST) Handbook 44 which is published every fall. A complete copy of the handbook can be ordered from the Government Printing Office or can be ordered directly from NIST. An order form for the handbook can be obtained by using the NIST fax on demand system which can be reached by calling the 24-hour fax line 1-800-925-2453. The handbook can also be obtained by joining the National Conference on Weights and Measures (NCWM). More information can be obtained by writing to the National Conference on Weights and Measures at P.O. Box 4025, Gaithersburg, Maryland 20885. A copy of the handbook sections dealing with weighing devices can be obtained from Cardinal Scale Manufacturing Company on a limited basis.

How do I submit my software program to NTEP for type evaluation?

The evaluation process begins with the submission of a completed application for evaluation and a \$175 filing fee (this fee is in effect at the time this was written). The application form is found in the NCWM Publication 14 which may be purchased directly from NCWM. This Publication 14 contains information on the administrative policies, technical policy, test procedures and checklist of NTEP. It is recommended that you purchase Publication 14 because it is a valuable resource when dealing with NTEP. If you choose not to purchase Publication 14, the application form can be obtained directly from the NIST fax on demand line by calling 1-800-925-2453. After submitting the completed application and filing fee, you will be assigned a control number and told the NTEP lab that has been assigned to evaluate your software program. Note that you will have to include a Model 778 and any other hardware peripheral needed to execute your program. Owner's or instruction manuals should also be submitted to the lab along with the equipment.

What happens when the evaluation of my software is completed?

Assuming that your software program has met all the requirements, the NTEP lab will contact the office in Gaithersburg, Maryland and a certificate number will be assigned to your program. The certificate number allows you to sell your software program for commercial use. The formal NTEP Certificate of Conformance will be sent to you in about 90 days from assignment of the certificate number. You will also be billed for another \$175 for drafting and issuance of the certificate plus the charges for the time spent at the lab in the evaluation process. Different labs have different rates and, since more complex programs require more time to evaluate, it is difficult to estimate these costs.

If your software program does not pass the evaluation, the examiner will normally contact you and advise you of the shortcomings. You may then correct these problems and continue with the evaluation. This, however, must be done in a timely manner. If not, your file will be closed and you will be forced to start the process from the beginning.

What do I need to do to ensure that my software program meets the requirements?

To detail all the requirements necessary for commercial use is beyond the scope of this document. There are, however, some things to remember when writing the program that will make it more likely to pass the NTEP evaluation:

Display of Weight

It is of utmost importance that the weight value obtained from the 778 indicator not be altered in any way before it is shown on the display screen. The weight must be displayed and properly identified at all times. Note, however, that it is not necessary to

display the weight when in setup screens, while entering data (including tare weights) or operational parameters, or when performing diagnostics. During normal operation, the weight must be displayed and should be accompanied with the units of measure (pounds, kilograms, tons, etc.), the polarity, the scale number (if more than one scale is in use) *and* the mode (gross or net). The status conditions (like motion, center of zero and over capacity) should also be displayed. Note that the 778 weight indicator program (Standard.bas) has been evaluated and presents the weight display in an acceptable manner. You may wish to use it as a guide in displaying weight.

Identification of Program

The name of your program, it's author, and revision level can be displayed when a specific key is pressed. Regardless of how it is accomplished, the local weights and measures inspector must be able to view this information.

Recording of Weight Data

If your software program transmits weight data to a printer, it is important to note that the same criteria applies to it as it does to the display of weight data. That is, the units of measure should be included along with the mode (gross, tare or net) and the scale number (if more than one scale is used). Further, the weight must not be printed when the gross weight is below zero (negative), when the gross weight exceeds the scale capacity (over capacity) or when the weight is unstable (motion). These conditions are indicated by the appropriate status within the 778 weight indicator variables. Printing weight under one or more of these conditions is not allowed in commercial devices. Weight data transmitted to a computing device is not under these restrictions and can take place regardless of these conditions provided the weight value is accompanied by the unit of measure and status information. Weight data transmitted to a remote display can also take place at anytime. Remember, however, that the remote display must be capable of displaying these same conditions (polarity, over capacity, motion, scale number, units of measure and weight mode).

Facilitation of Fraud

Special care must be taken in the design of the software program to ensure that fraud cannot take place, That is, provisions must be made to prevent the scale operator from manipulating the displayed and recorded weight value. This is normally accomplished through protecting access to the calibration switch with a lead-wire security seal. The 778 adds another measure of protection by allowing the weights and measures inspector to return to the basic weight indicator program which permanently resides in the 778. In so doing, the inspector can compare the weight value displayed in the previously approved basic weight indicator with that displayed by the special application software to ensure that the two are identical. If a difference exists between the two, it could indicate that the value has been modified in the application software program. Finally, the SmartWeigh Development Software has a feature that identifies the particular development system used to develop the application software. This information can help identify the source of the application program.

Other Considerations

When writing 778 application software programs that will be submitted for NTEP evaluation or checked by local weights and measures inspectors, it is important to keep in mind other requirements. Perhaps the best single source of information that identifies these requirements is the applicable checklist contained in Publication 14. This checklist consists of a series of simple tests or items to verify to ensure compliance with Handbook 44. By stepping through this checklist prior to submittal of your software, you can ensure that the evaluation process will result in a NTEP Certificate of Conformance.

CARDINAL SmartWeigh

778 Programmable Weight Indicator Programmer/Simulator

The Cardinal SmartWeigh is a versatile program used to program and simulate the powerful features of the 778 Programmable Weight Indicator. Programming is accomplished by using the Cardinal BASIC program language. The Cardinal BASIC language is simple to use and easy to learn. It's similar to the BASIC language, but with more powerful commands and features.

INSTALLING

SmartWeigh can be installed on a computer with Windows 95, 98, NT 4.0 or 2000. Before you can use SmartWeigh you have to run the setup program from Windows so it will work properly on your computer. You cannot just copy the files from the SmartWeigh disks to your hard drive. The files on the distribution disks are compressed in a special way to save space. The setup program de-compresses those files and places them on your hard drive.

To install the SmartWeigh program, follow these steps:

1. Click on the "Start" box on the tool bar at the bottom left of the screen.
2. Select "Settings", then click on "Control Panel".
3. When the control panel window appears, double-click on the "Add/Remove Programs" icon.
4. The "Add/Remove Programs Properties" window will appear. Click on the "Install..." button.
5. The "Install Program From Floppy Disk or CD-ROM" window will appear. Insert the SmartWeigh Disk 1 into the floppy drive and then click on the "Next >" button.
6. The computer will search for the Setup program. When located, the "Run Installation Program" window will appear and "A:\SETUP.EXE" or "B:\SETUP.EXE" will show in the command line window.
7. Click on the "Finish" button to begin the installation process.
8. The Setup program will appear showing you the percentage completed in preparing the InstallShield Wizard to guide you through the rest of the setup process. When the InstallShield Wizard is at 100%, the setup "Welcome" window will appear.
9. Read the instructions on the screen, then click on the "Next >" button.

INSTALLING, Cont.

10. The "Choose Destination Location" window will appear.

This is the location that you want your SmartWeigh files to be placed. Setup will suggest a drive and directory of "C:\WIN778N".

If you want the SmartWeigh program files installed on a different drive or into a different directory, click on the "Browse" button and select the drive and directory name you would like to install to or type in a new one in the "Path" command line window. Click on the "OK" button to proceed. If the directory does not exist, a "Setup" window will appear asking if you want to create the directory. Click on the "Yes" button. The "Setup" window will close and the "Choose Destination Location" window will be in full view again.

12. Click on the "Next >" button.

13. The "Setup Type" window will appear. There are three types of Setup, click on the type of Setup you prefer, then click on the "Next >" button. The three types are as follows:

Typical: The program will be installed with the most common options.
This type is recommended for most users.

Compact: The program will be installed with minimum required options.

Custom: You may choose the options you want to install.
This is recommended for "advanced" users.

14. Setup will begin de-compressing the necessary files and placing them on your hard drive. The status of the operation will be displayed as the setup program performs this.

15. When the setup program has installed the files from Disk 1, it will prompt you to insert the next disk, Disk 2 or to input the location of the files from Disk 2, if they can be found in another location.

16. After inserting Disk 2 or inputting the location of the files, click on the "OK " button.

17. Setup will begin de-compressing the files from Disk 2 and placing them on your hard drive. Again, the status of the operation will be displayed as the setup program performs this.

18. When the installation of Disk 2 is finished, a "Setup Complete" window will appear. If you want to run the SmartWeigh program at this time, click on the "Yes, Launch the program file" and then the "Finish" button. Otherwise, click on the "Finish" button by itself to return to Windows 95.

STARTING

When SmartWeigh is started the first time, it will look for a configuration file that hasn't been created yet and will create a "default" file. See Figure No. 1. This "default" file will allow you to run all of the features of the program **except** the Transfer options, which allows you to send files to and receive files from the 778. To activate the Transfer options of your SmartWeigh program, you must register your program. Click on the "OK" button to continue.



Figure No. 1

Next, you will see a window indicating the default project file "DEFAULT.PRJ" can't be found. See Figure No. 2. This is **not** an error, only an indication that the "default" file that SmartWeigh loads and runs each time it is started has not been created yet. The default project file will be created when you exit the SmartWeigh program. It will also be updated each time thereafter when you exit the program. Click on the "OK" button to continue.

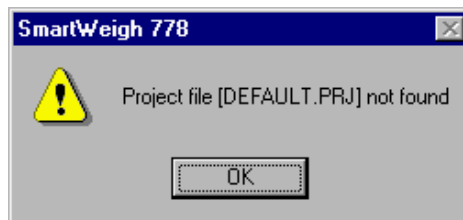


Figure No. 2

REGISTERING

To register your SmartWeigh program and activate the Transfer option, you must complete the program registration form located in the "Help (or Help, About)" menu bar selection. See Figure No. 3. This registration form requires you to enter the Cardinal Scale sales order number, a contact (your name), your company name, the city, state, and country your company is in and a registration number. See Figure No. 4.

To obtain your "Registration Number", contact Cardinal Scale sales customer service at 1-800-441-4237. *You do not have to register immediately, but product support is only available to registered users.*

REGISTERING, Cont.

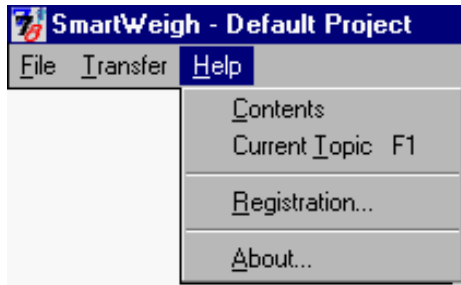


Figure No. 3

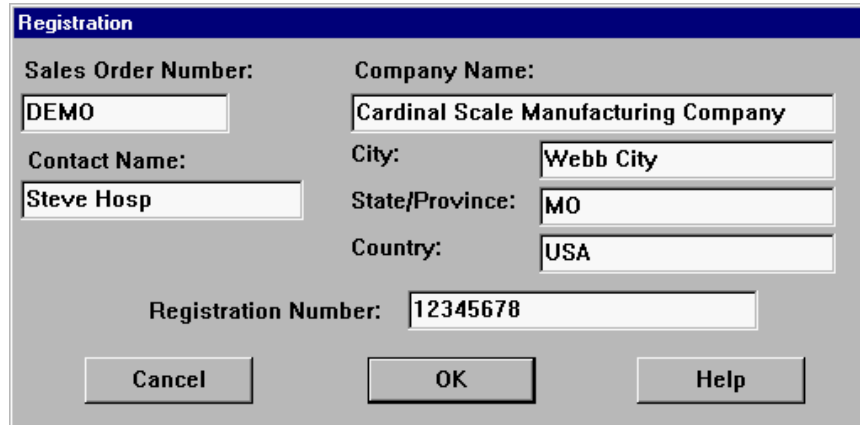
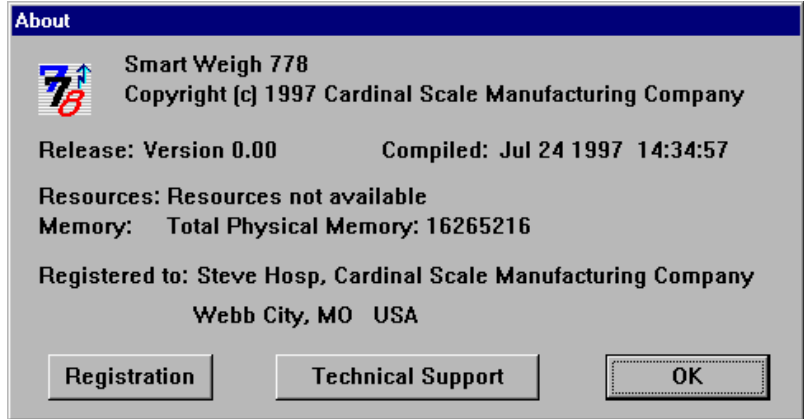


Figure No. 4

REGISTRATION REMINDER

A registration reminder screen (see Figure No. 5) will be displayed on an un-registered version of SmartWeigh each time the program is closed and when opening a New 778 Simulator. This is to remind you that the file transfer options and product support is only available to registered users.

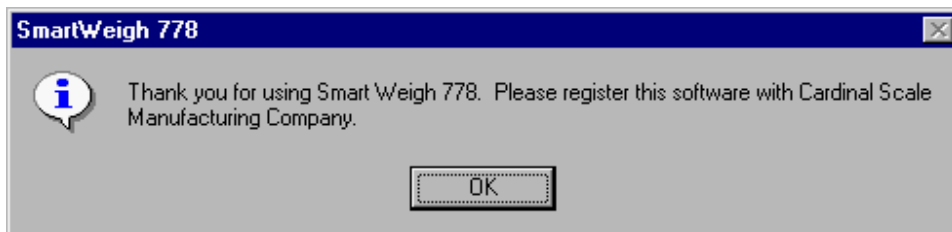


Figure No. 5

RUNNING SMART-WEIGH

File Menu

SmartWeigh will start with a intro screen, then after a few seconds, proceed to a blank display with only the menu bar headings.



Figure No. 6

NOTE: If a project or program was displayed when SmartWeigh was closed, that program or project will be shown (instead of a blank display) the next time SmartWeigh is started.

Click on File (see Figure No. 6) to activate the File window.

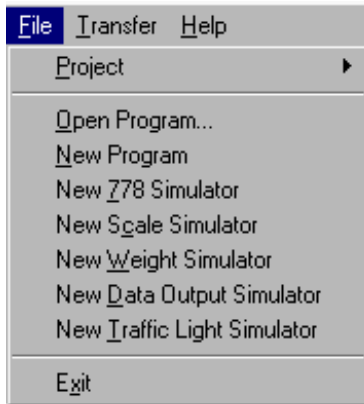


Figure No. 7

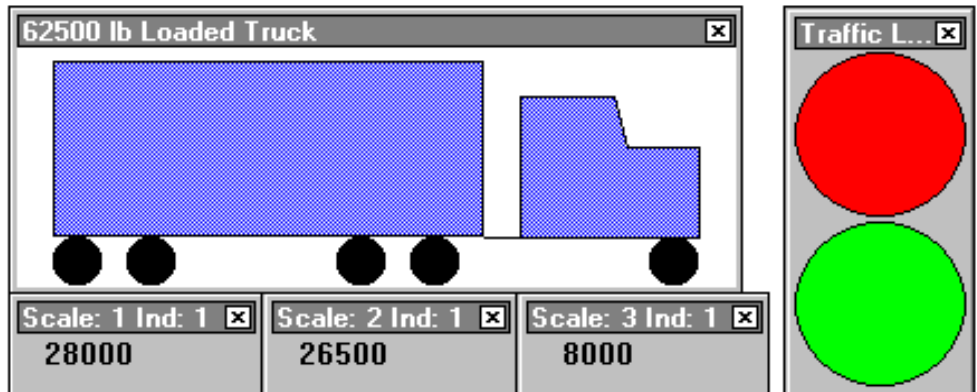


Figure No. 8

Click on one or more of the following File selections (see Figure No. 7):

- Project -- This command (see Figure No. 9) allows you to create a New project, Open an existing project (.prj file extension) to edit or to Save the project as... (*project name*).



Figure No. 9

- Open Program . . . -- This command allows you to open an existing program (.bas file extension) to list, edit or to run
- New Program -- This command allows you to create a new Cardinal BASIC program for the 778 indicator.
- New 778 Simulator -- This command loads the self running SmartWeigh Simulator program. Refer to SmartWeigh Simulator section for operation of self running program.

File Menu, Cont.

Examples of the following File selections can be seen in Figure No. 8.

- New Scale Simulator -- This command opens a window to represent a scale platform. The scale simulator can be positioned anywhere within the main window as well as reduced or enlarged in size. Multiple simulators can be opened and placed together to represent a sectional scale. You can make connections to indicators and scales (assign numbers) from the Options selection on the menu bar after the Scale Simulator window is opened.
- New Weight Simulator -- This command opens a window which looks like a multiple axle truck. Like the Scale Simulator, it can be reduced or enlarged in size as needed. When positioned on top of the Scale Simulator(s), it will cause the scale to display a weight reading. The weight for up to five (5) axles can be configured from the Options selection on the menu bar after the Weight Simulator window is opened.
- New Data Output Simulator -- This command will start the data output simulator. It can be configured to represent the data output of one (1) or all eight (8) serial ports by selecting Port on the menu bar. The example shown in Figure No. 10 is on port 2 of the simulator and represents a Ticket printer.
- New Traffic Light Simulator -- This command opens a window which looks like a traffic light. It also can be reduced or enlarged as needed. After opened, from the Options selection on the menu bar, you can configure the number of lights, their color and position as well as make connections to an indicator, board or relay. In addition, the connections can be configured as Normally Closed or Normally Open.
- Exit -- This command will close Smart-Weigh and exit to Windows.

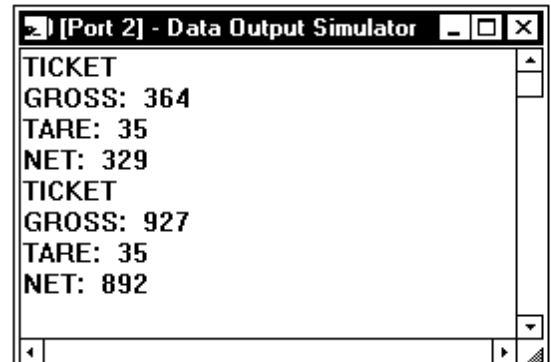


Figure No. 10

Transfer Menu

The Transfer menu (Figure No. 11) allows you to send (download) a program from the computer to a 778 indicator or to receive (upload) a program from a 778 to the computer. The transfer is by way of the RS232 serial port on the 778 and the computer's Com 1, serial port.

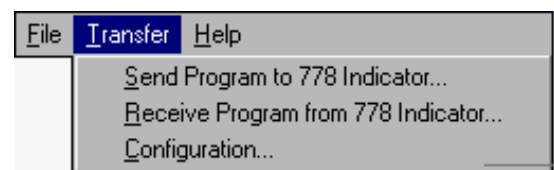


Figure No. 11



If your SmartWeigh program has not been registered, a reminder (see Figure No. 12) will appear when attempting to use the file transfers options. To activate the file Transfer options you must register your program.

Transfer Menu, Cont.

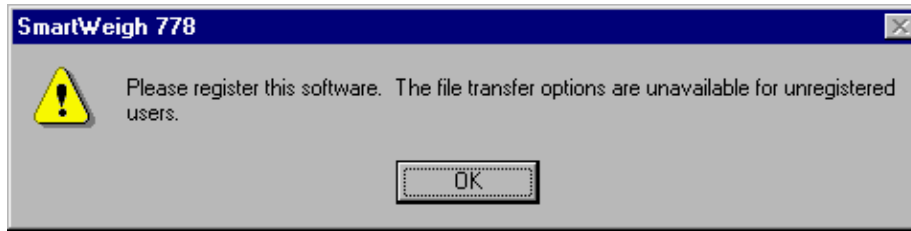


Figure No. 12

Send Program

To send a program from the computer to the 778, click on Transfer on the main menu bar. Next, click on Send Program to 778 Indicator. A window (see Figure No. 13) will open prompting for the filename. Type in or click on the file name, then click on the "OK" button. That window will close and another window (see Figure No. 14) will open to display the status of the file transfer. When the transfer is complete, the display will clear and a small window (Figure No. 15) showing the file transfer was successful will open. Click on the "OK" button to continue.

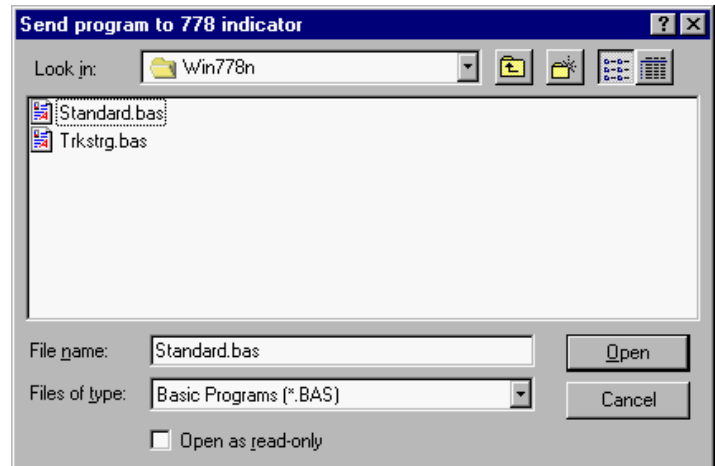


Figure No. 13

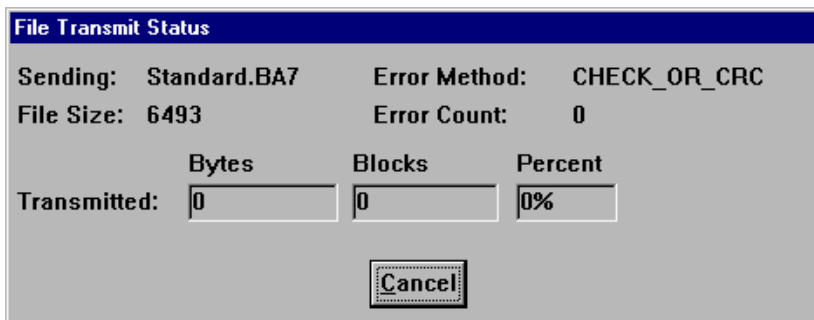


Figure No. 14

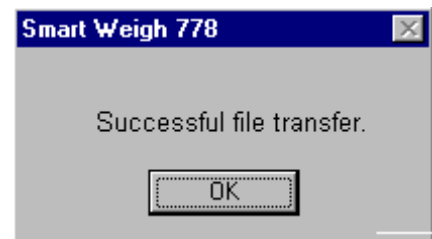


Figure No. 15

Receive Program

To transfer (receive) the program from the 778 to the computer, click on Transfer on the main menu bar. Next, click on Receive Program from 778 Indicator. A window (see Figure No. 16) will open to display the status of the file transfer. When the transfer is complete, the display will clear and a small window (Figure No. 17) showing the file transfer was successful will open. Click on the "OK" button. That window will close and a new window (see Figure No. 18) will open prompting for the filename. Type in the file name, then click on the "OK" button.

Transfer Menu, Cont.

Receive, Cont.

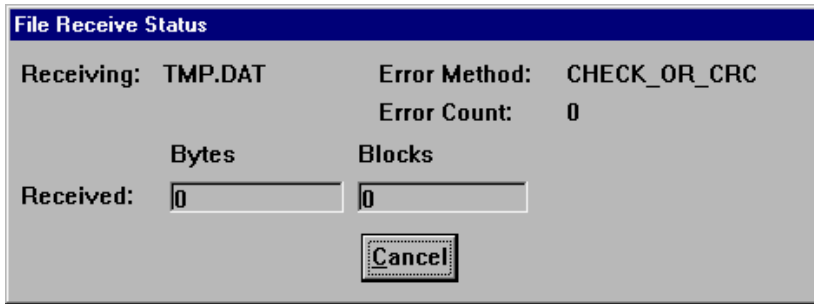


Figure No. 16

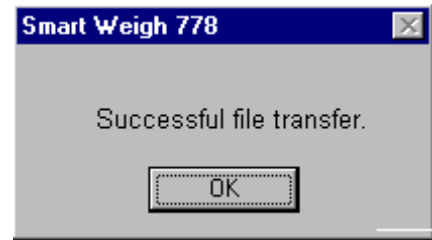


Figure No. 17

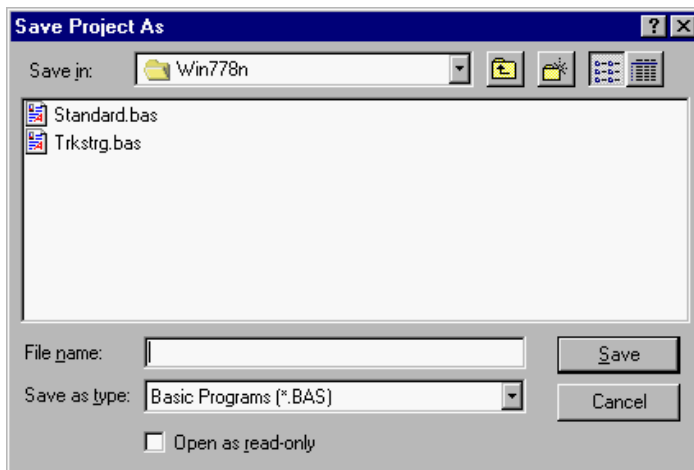


Figure No. 18

Configuration

To configure the serial communications between the 778 and the computer SmartWeigh is running on, click on **T**ransfer on the main menu bar, then click on **C**onfiguration. This option allows you to select the serial port on your computer you will be using to transfer files to and from the 778. It also allows you to set the baud rate, parity, data bits and stop bits on the computer to match those settings in the 778 indicator. See Figure No. 20.

Program Password

SmartWeigh also provides a layer of security for your custom program to prevent un-authorized duplication. This is accomplished by allowing you to enter a password that will be attached to the file sent from SmartWeigh to the 778 indicator. The password can be up to eight (8) numeric characters. With a password protected indicator, the 778 will prompt for a Authorization Code (which must be entered) before it can send a file to the computer.

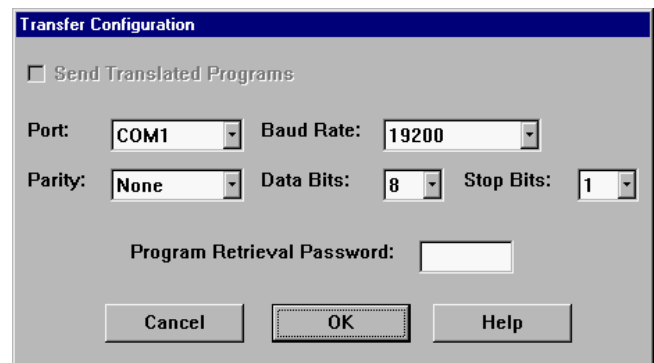


Figure No. 20

Transfer Menu, Cont.

Program Password, Cont.



CAUTION: The password is not visible on the simulator when entering it, only asterisks (*) are displayed. Use a password that you can easily remember. In the event the program in the 778 must be retrieved, due to loss of the file on the computer, and the password is not available, contact 778 Technical Support at (417) 673-7835.

SMARTWEIGH SIMULATOR

778 Simulator

From the SmartWeigh Main Menu click on File, then click on New Z78 Simulator. This will load the 778 Programmable Weight Indicator Simulator. Refer to Figure No. 21.

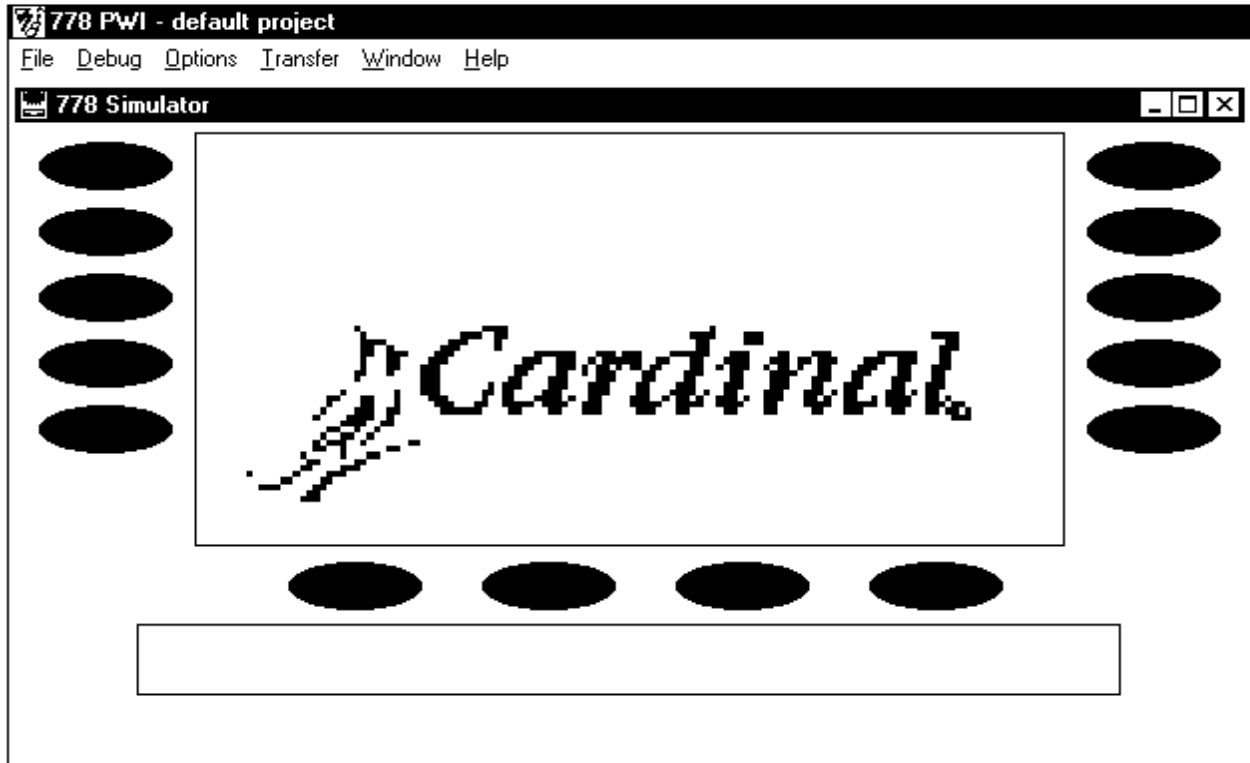


Figure No. 21

File Menu

With the 778 Simulator screen showing, click on File on the main menu bar. A File Menu window will open (see Figure No. 22) displaying additional options than the Main Menu File menu. Click on Open Program into Simulator.

The first window will close and another will open (see Figure No. 23) prompting for the filename to open and load into the simulator. Type in a filename or click on a filename already shown and then click on the "OK" button.

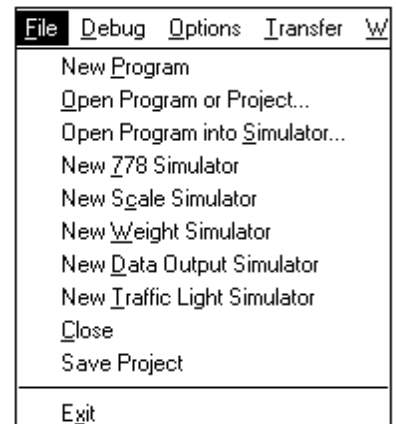


Figure No. 22

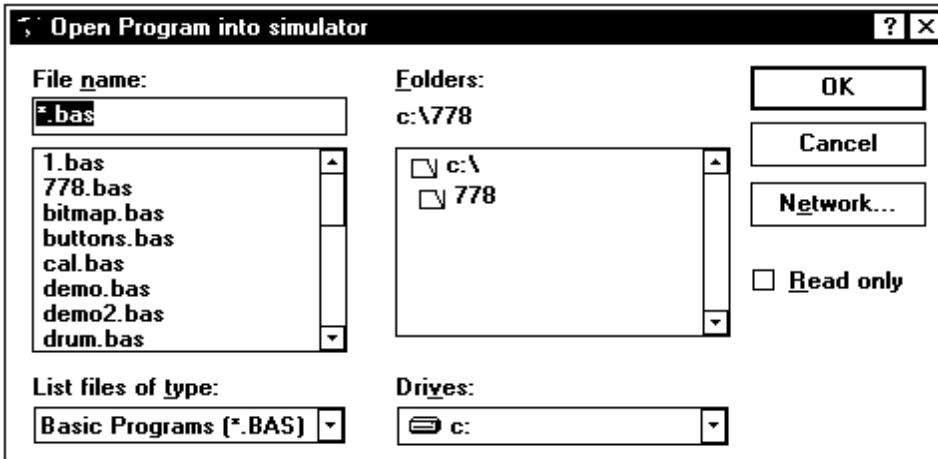


Figure No. 23

After the filename has been entered and the program loaded, click Debug on the menu bar to display a window (see Figure No. 24) allowing you to Run or Step Run the program loaded into the simulator.

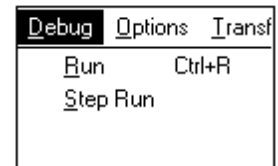


Figure No. 24

With the program running, if you click on Debug, the window will have changed (see Figure No. 25 to allow you to Stop the program or if in the Step mode, to advance to the Next step of the program.



Figure No. 25

Options Menu

The Option Menu (see Figure No. 26) of the 778 Simulator allows you to show the sliding bar Weight Simulator, add and show up to four (4) I/O boards, and to show the display in black and white.

The sliding bar Weight Simulator allows you to vary the simulated weight by moving the mouse pointer along the rectangle centered directly under the four (4) softkeys below the simulated display (see Figure No. 27).

The four (4) I/O boards represent the 778 Relay Output Boards. Each board has 8 outputs and 8 inputs. The outputs can be used to simulate status indicators. An example would be for Preset Weight Comparator presets. The inputs can represent AC inputs such as remote start and stop switches.



Figure No. 26

The B/W mode displays the 778 simulated display with black characters on a white background instead of red characters on a black background.

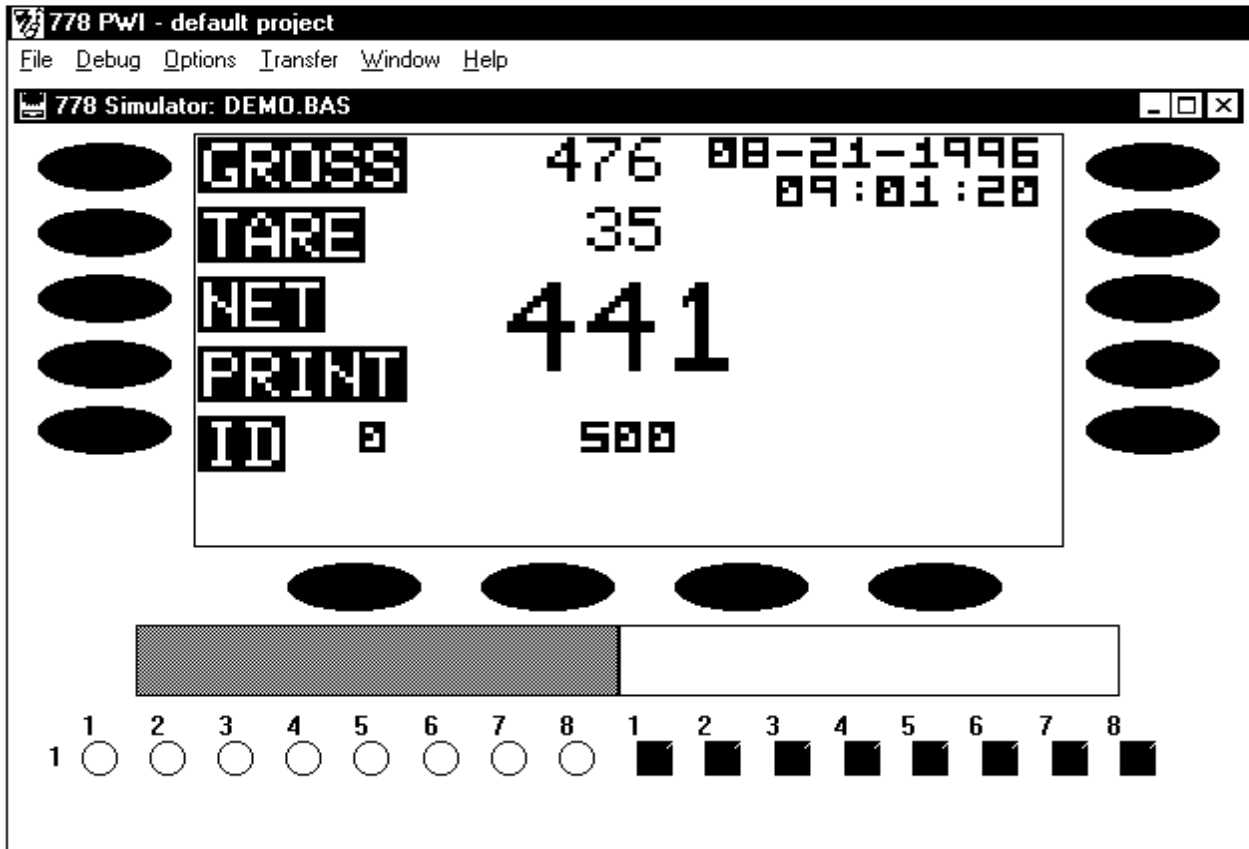


Figure No. 27

Pages 18 left out for future use.

CHARACTER SET

The Cardinal BASIC character set includes all characters that are legal in Cardinal BASIC commands, functions, system variables, labels and variables. The set is comprised of alphabetic characters (A-Z), numeric characters (0-9 and A-F for hexadecimal numbers), and special characters. The following special characters are recognized by Cardinal BASIC:

<u>Character</u>	<u>Meaning</u>
	Blank
=	Equals sign or assignment symbol
+	Plus sign
-	Minus sign
*	Asterisk or multiplication symbol
/	Slash or division symbol
^	Up arrow or caret or exponentiation symbol
(Left parenthesis
)	Right parenthesis
%	Percent
#	Number or pound sign
\$	Dollar sign or string data type suffix
[Left bracket
]	Right bracket
,	Comma
" "	Double quotation marks or string delimiter
.	Period or decimal point
;	Semicolon
:	Colon
&	Ampersand or descriptor for hexadecimal and octal number
?	Question mark
<	Less than
>	Greater than
\	Integer division symbol (backslash)
Enter	Terminates input of a line

USING CARDINAL BASIC

The information in this section will help you learn about the use of constants, variables, labels, expressions, and operators in Cardinal BASIC.

Constants

Constants are static values Cardinal BASIC uses during execution of your program. There are two types of constants: *string* and *numeric*.

A *string constant* is a sequence of 0 to 255 alphanumeric characters enclosed in double quotation marks. These alphanumeric characters can be any of the characters whose ASCII codes fall within the range 0-255 (except the double quote character (") and carriage return, line-feed sequences). The following are examples of string constants:

"TRUCKS" "ENTER TARE WEIGHT" "\$2,195.00"

Numeric constants can be positive or negative. When entering a numeric constant in Cardinal BASIC, you should not type the commas. For instance, the number 10,000 would be entered as 10000 if used as a constant. The four types of numeric constants are listed below:

- | | |
|----------------|--|
| Integer | Whole numbers between -32768 and +32767. They do not contain decimal points, but may have an optional sign prefix (+ or -). |
| Floating-Point | Positive or negative numbers represented in exponential form (similar to scientific notation). A floating-point constant is an optionally signed integer (the mantissa) followed by the letter E and an optionally signed integer (the exponent). The constant's value is the mantissa multiplied by the power of ten represented by the exponent. Single-precision constants have a range of -3.37E+38 to 3.37E+38. |
| Hexadecimal | One or more hexadecimal digits (0-9, or A-F) with the prefix &H. The range for integer hexadecimal constants is &H0 to &HFFFF. For example: &H2A. |
| Octal | One or more octal digits (0-7) with the prefix &O, or &. The range for integer octal constants is &O0 to &O177777. For example: &O347 or &1234. |

Constants in Cardinal BASIC are *Single-Precision* numbers and are stored as Floating-Point numbers with 7 digits accuracy.

USING CARDINAL BASIC, Cont.

Variables

A variable is a name that refers to an object, a particular number or string (\$). Simple variables refer to a single number or string. Array variables refer to a group of objects, all of the same type.

A numeric variable, whether simple or array, can be assigned only a numeric value (single precision). A string variable can be assigned only a character-string value. You can assign one record variable to another only if both variables are the same type. **Note:** Before a variable is assigned a value, its value is assumed to be zero (for numeric variables) or null (for string variables).

Variable Names

A Cardinal BASIC variable name may contain up to 40 characters. The characters allowed in a variable name are letters, numbers, the period (.), and the declaration character (\$). The first character in a variable name must be a letter.

A variable name cannot be a reserved word, but embedded reserved words are allowed. For example, the following statement is illegal because LOG is a reserved word.

```
LOG = 12
```

However, the following statement is legal:

```
TIMELOG = 12
```

Reserved words include all Cardinal BASIC commands, functions, system variables and expression and operator names. **Note:** Variable names must be unique and cannot duplicate procedure (SUB and FUNCTION) names or label names.

Labels

A label is an identifying string that denotes a specific location in a program. It may consist of a single letter or combination of letters, digits and periods. The label string must be terminated by a colon (:) and on a line by itself (no other characters including spaces). Labels are used by the GOTO command to branch to another section of the program and execute the commands following the label. **NOTE:** Within a program or sub-routine a label must be unique and cannot duplicate the name of a variable or a SUB or FUNCTION procedure.

USING CARDINAL BASIC, Cont.

Expressions

An expression can be a string or numeric constant, a variable, or a combination of constants, variables, and other expressions with operators that produce a single value. An "expression" can be thought of as a formula. Expressions may be as complex as needed, with the results of one expression passed to a function or combined with other expression, and so on.



IMPORTANT! Cardinal BASIC keywords, (commands, function and system variables), labels, variable names, expressions and operators must be in upper case characters (capital letters).

Operators

Operators perform mathematical or logical operations on values. The operators recognized by Cardinal Basic are divided into four categories:

- Arithmetic
- Relational
- Logical
- Functional

The following operators are listed in order of precedence, with 1 equal to the highest.

Arithmetic Operators

Arithmetic operators are used to perform calculations on values. Operations within parentheses are performed first. Inside the parentheses, the usual order of precedence is maintained.

<u>Operator</u>	<u>Operation</u>	<u>Precedence</u>	<u>Expression</u>
^	Exponentiation	1	X^Y
*	Multiplication	2	X * Y
/	Floating-point Division	2	X / Y
\	Integer Division	3	X \ Y
MOD	modulus (remainder)	4	X MOD Y
+	Addition	5	X + Y
-	Subtraction	5	X - Y
=	Equal to or assignment	6	X = Y

USING CARDINAL BASIC, Cont.

Operators, Cont.

Relational Operators

Relational operators are used to compare two values. The result of the comparison is either "true" (nonzero) or "false" (zero). This result can then be used to make a decision regarding program flow. Although Cardinal BASIC treats any nonzero value as true, true is usually represented by -1.

<u>Operator</u>	<u>Relation</u>	<u>Precedence</u>	<u>Expression</u>
<>	Unequal	7	X <> Y
<	Less than	8	X < Y
>	Greater than	9	X > Y
<=	Less than or equal to	10	X <= Y
>=	Greater than or equal to	10	X >= Y

Logical Operators

Logical operators perform tests on multiple relations, bit manipulations, or Boolean operations and return a true (nonzero) or false (zero) value to be used in making a decision. In an expression, logical operations (also known as Boolean operations) are performed after arithmetic and relational operations. The operands of logical operators must be in the range of -2,147,483,648 to +2,147,483,647. Operands are converted to integers before the logical operation is done. (If the operands are not in this range, an error results.) If the operands are either 0 or -1, logical operators return 0 or -1 as the result. **NOTE:** Since logical operators perform boolean operations, they will not properly evaluate operands other than 0 (false) or -1 (true). Listed, in order of precedence are the logical operators in Cardinal BASIC:

<u>Operator</u>	<u>Meaning</u>	<u>Precedence</u>	<u>Expression</u>
NOT	Negation (complement)	12	NOT Y
AND	Conjunction	13	X AND Y
OR	Disjunction (inclusive "or")	14	X OR Y
XOR	Exclusive "or"	15	X XOR Y
IMP	Implication	16	X IMP Y
EQV	Equivalence	17	X EQV Y

USING CARDINAL BASIC, Cont.

Operators, Cont.

Functional Operators

A function is used in an expression to call a predetermined operation that is to be performed on an operand. Cardinal BASIC has intrinsic functions such as SQR (square root) or SIN (sine). Cardinal BASIC also allows "user-defined" functions that are written by the programmer. The following are Cardinal BASIC functional operators:

ABS	EXTINP*	LOG	SPACE\$
ASC	GROSS*	MID\$	SPC
ATN	HEX\$	MKD\$	SQR
BLOZERO*	INP*	MKI\$	STR\$
CAPACITY*	INP\$*	MKS\$	STRING\$
CHR\$	INSTR	MOTION*	TAN
CINT	INT	NET	TARETYPE*
COS	INTERVAL*	OCT\$	TIMER
CVD	LEFT\$	OVERCAP*	UNITS*
CVI	LEN	RIGHT\$	VAL
CVS	LINE INPUT	RND	WTMODE*
CZERO*	LOC	SGN	
EXP	LOF	SIN	

** Denotes functions exclusive to the 778 and Cardinal BASIC.*

Cardinal System Variables

CURSCALE*	DSPFONT*	DSPY*	TARE*
CURUNIT*	DSPLOGIC*	ERRNO*	TIME\$
DATE\$	DSPX*	RELAY*	

** Denotes system variables exclusive to the 778 and Cardinal BASIC.*

USING CARDINAL BASIC, Cont.

Cardinal BASIC Commands

ARC*	DO...LOOP UNTIL	IF...THEN...ELSE	PUT
BIRD*	DOTARE*	INPUT	RANDOMIZE
BOX*	DOZERO*	INPUT#	READ
CALL	DRAW	KEY	READY*
CLEAR	ELSE	KILL	REM
CLOSE	ELSEIF	LET	RESTORE
CLS	END	LOCATE	RSET
COMMON	EOF	LSET	RTRIM\$
DATA	ERASE	LTRIM\$	SELECT CASE
DEF FN	EVENT*	NAME	SUB
DEFSNG	EXIT	NEXT	SWAP
DEFSTR	FIELD	OPEN	WHILE..WEND
DIM	FOR	OPEN...FOR...AS	WRITE
DO	FUNCTION	PLOT*	
DO...LOOP	GET	PRINT	
DO WHILE...LOOP	GOTO	PRINT USING	

** Denotes commands exclusive to the 778 and Cardinal BASIC.*

CARDINAL BASIC

This section summarizes the Commands, Functions, and System Variables of the Cardinal BASIC program language. Each Function, Command or System Variable is shown exactly as it must be entered to use it, followed by a description and example. In addition, several lines of a program are listed to show you a working sample of the particular part of Cardinal BASIC.

ABS

Function: ABS(argument)

This a math function that returns the absolute value of its argument. The 'argument' can be a number or numeric expression and its unsigned magnitude is returned by the ABS function. For example, ABS(-20) and ABS(20) are both 20.

Example: CLS 0,0,127,63
 PRINT ABS(10*(-5))

Output: This example will clear the display, then print to the display 50, the ABS of 10 multiplied by (-5).

ARC

Command: ARC m ,w ,x ,y ,t ,p

This command will plot an arc specified by:

m = major axis, the radius of the length of the arc,
w = minor axis, the radius of the height of the arc,
 (The major and minor axis's is measured in pixels from the center point.)
x,y = center point, the display coordinates where the axis's intersect (meet)
t = theta, the starting point of the arc in degrees
p = phi, the ending point of the arc in degrees
 (A complete rotation from t= 1 to p = 360 would plot a circle).

NOTE: All coordinates must be within the range of the display, or an error will be reported.

Example: CLS 0,0,127,63
 ARC 25, 25, 64, 32, 1, 180

Output: This example will clear the display, then draw an arc, (which looks like the upper half of a circle) in the upper center of the display.

ASC

Function: ASC(string\$)

This is a string processing function that will return the numeric value of the ASCII code for the first character in a string or string expression. If the 'string\$' is null, an "Illegal Function Call" error will be returned. If the 'string\$' begins with an uppercase letter, the value returned will be in the range of 65 to 90. If the 'string\$' begins with a lowercase letter, the value returned will be in the range of 97 to 122. Numbers 0 to 9 return a value of 48 to 57 respectively.

Example: CLS 0,0,127,63
X\$ = "Weight"
PRINT ASC(X\$)

Output: This example will clear the display, then print to the display 87, the ASCII code for the letter *W*.

ATN

Function: ATN(number)

This is a math function that will return the arctangent of a 'number' with the result given in radians in the range $-\pi/2$ to $\pi/2$ radians, where $\pi = 3.141593$. You can convert from degrees to radians by multiplying the degrees by $\pi/180$. For example, $\pi/2$ radians equals 90 degrees. To convert a radian value to degrees, multiply it by 57.2958.

Example: CLS 0,0,127,63
PRINT ATN(5)

Output: This example will clear the display, then print to the display, 1.3734008, the ATN of 5 radians.

BIRD

Command: BIRD

This command will draw the Cardinal Scale logo on the display.

Example: CLS 0,0,127,63
BIRD

Output: This example will clear the display, then draw the Cardinal Scale logo.

BLOZERO

Function: BLOZERO ([scalenumber])

This is a function that returns the state of below zero status for the default scale or the selected scale. A zero (0) will be returned if the scale is not below zero.

Example: CLS 0,0,127,63
IF BLOZERO = 0 THEN
 PRINT "ZERO OR ABOVE"
ELSE
END IF

Output: This example will clear the display, then read the BeLOW ZERO status of the scale. If the scale is at zero or above, the message "ZERO OR ABOVE" will be printed in the upper left corner of the display.

BOX

Command: BOX X1,Y1,X2,Y2,(fill)

This command will draw a box from the coordinates specified by X1, Y1 to X2, Y2. If (fill) is a "1", the box will be solid or if (fill) is a "0", the box will be lines only. **NOTE:** All coordinates must be within the range of the display, or an error will be reported.

Example: CLS 0,0,126,63
BOX 10,10,40,40,1

Output: This example will clear the display, then draw a solid box from position 10,10 to 40,40.

CALL

Command: CALL subroutine name

This is a command that transfers control of the main program to another part of the same program, called a subroutine. Subroutines are used for repetitive operations such as reading the keyboard to check if a key has been depressed. (See SUB and END SUB).

Example: DO
 CALL INPUTVALUE("TIME","(HHMM)",TIME\$,VALUE\$)
 IF ERRNO=1 THEN EXIT DO
 IF VAL(VALUE\$)>2459 THEN ERRNO=1
 IF LEN(VALUE\$)<>4 THEN ERRNO=2
LOOP UNTIL ERRNO=0

CALL, Cont.

Output: This example will call a subroutine named INPUTVALUE, which prompts for the time on the display and waits for the correct input on the keyboard.

CAPACITY

Function: CAPACITY([scalenumber])

This is a function that returns (reads) the capacity for default scale or selected scale if followed by [scalenumber].

Example: CLS 0,0,127,63
PRINT CAPACITY

Output: This example clears the display, then prints the capacity of the default scale to the display.

CHR\$

Function: CHR\$(number)

This is a function that returns a one-character string with the character being the ASCII equivalent of the 'number'. The 'number' must be a value between 0 and 255. CHR\$ is commonly used to send special characters to the display or printer.

Example: CLS 0,0,127,63
PRINT CHR\$(42)

Output: This example will clear the display, then print to the display an (*) asterisk.

CINT

Function: CINT(number)

This is a math function that is used to truncate a 'number'. If the 'number' is not in the range of -32,768 to 32,767, the function produces a error message that reads "Overflow."

Example: CLS 0,0,127,63
PRINT CINT(-37.2831)

Output: This example clears the display, then prints 37 to the display.

CLEAR

Command: CLEAR

This is a memory management command that reinitializes all program variables. It sets all numerical variables to zero (0), and all string variables to null.

Example: CLEAR

Output: This example zeroes (clears) all COMMON and user variables.

CLOSE

Command: CLOSE #file number

This command is used within a program to conclude the input/output to the file indicated by the '#file number'. Closing a file writes the data associated with the file and frees the file number to be used by another OPEN command. The 'file number' must be specified to close only that file. A CLOSE command without a 'file number' will close all open files.

Example: (For an example, see the OPEN command.)

CLS

Command: CLS [X1,Y1,X2,Y2]

This is a command that is used to clear all or part of the display. By specifying display coordinates, [X1,Y1, X2, Y2] only the region of the display within those coordinates will be cleared. If the CLS command is used without coordinates, the entire display will be cleared and any defined key labels will be redrawn.

Example: CLS 0,0,63,31

Output: This example will clear the display from the the upper left corner (X1,Y1 = 0,0) to approximately the middle (X2,Y2 = 63,31) of the display.



NOTE: When CLS is used, the display position is set to the upper left corner of the CLS region. For example: CLS (without coordinates) will set the display position to the upper left corner of the display, while CLS 20,20,60,40 will clear only that region of the display and set the display position to the upper left corner (20,20) of the region.

COMMON

Command: COMMON variablelist

This is a command that defines global variables that can be shared throughout a program. A variable is a Basic variable name. Variable names can consist of up to 40 characters and must begin with a letter. Valid characters are A-Z, 0-9, and the period (.). The 'variablelist' can be one or more variables, ex. variable[()], variable[()]...

Example: COMMON TRKID\$,TRKSTORED,TRKACCUM,TFLAG\$

Output: This example defines the string variables TRKID\$ and TFLAG\$ and the numeric variables TRKSTORED and TRKACCUM as common and shared throughout the program.



NOTE: COMMON statements should be placed at the beginning of the program before any operational code (statements).

COS

Function: COS(number[numeric expression])

This is a math function that will return the cosine of a 'number' or 'numeric expression' with the result given in radians. The numeric-expression, can be any numeric type. You can convert from degrees to radians by multiplying the degrees by $\pi/180$, where $\pi = 3.141593$. To convert a radian value to degrees, multiply it by 57.2958.

Example: CLS 0,0,127,63
X = COS(4)
PRINT X

Output: This example will clear the display, then print to the display, -0.65364, the cosine of 4 radians.

CURSCALE

System Variable: Set: CURSCALE=(scale number)
Read: CURSCALE ()

This is a system variable that can be used to select (Set) the current default scale or return (Read) the number of the currently selected scale.

Example: CLS 0,0,127,63
CURSCALE=1
X=CURSCALE
PRINT "CURRENT SCALE IS: ";X

CURSCALE, Cont.

Output: This example will clear the display, then set the current scale variable to scale number 1. Next it will read the value of CURSCALE and then print to the display "CURRENT SCALE IS: " and the value of CURSCALE.

CURUNIT

System Variable: Set: CURUNIT [scale number,] unit number
Read: variable = CURUNIT(scale number)

This is a system variable that can be used to select (Set) the weight units for the default scale, or a selected scale or return (Read) the current weight units for the default or selected scale. The unit numbers are (0) for Pounds or 1 for Kilograms.

Example: CLS 0,0,127,63
CURUNIT 1,0
X=CURUNIT (1)
PRINT "CURRENT WEIGHT UNIT IS: "X

Output: This example will clear the display, then set the weight units for the current scale to pounds (0). Next it will read the value of CURUNIT and print to the display "CURRENT WEIGHT UNIT IS:" and the value of CURUNIT.

CVS, CVI, CVD

Function: CVS(string\$) CVI(string\$) CVD(string\$)

The function CVS (convert to a single precision number) converts the argument 'string\$' into a Cardinal BASIC number, a floating point representation of the argument 'string\$'. String\$ variables read from a file must be converted into a number format before Cardinal Basic can use them. CVI (convert to a integer precision number) and CVD (convert to a double precision number), are provided to ease converting programs written in other forms of BASIC to Cardinal BASIC. Cardinal BASIC is single precision, therefore the functions CVS, CVI and CVD are effectively equivalent. (Also see the MKS\$, MKI\$, MKD\$ function).

Example: FIELD #1,4 AS N\$, 12 AS B\$
GET #1
Y=CVS(N\$)

Output: In this example, the first line defines the fields in file 1. The second line reads a record from file 1 and the third line converts the first four bytes into a Cardinal BASIC number assigned to variable Y.

CZERO

Function: CZERO([scale number])

This is a function that returns the state of center of zero status for the default scale or a selected scale. A zero '0' (false) will be returned if the scale is not at zero. A '-1' (true) will be returned if the scale is at zero.

Example: CLS 0,0,127,63
IF CZERO = -1 THEN
PRINT "CENTER OF ZERO"
ELSE
END IF

Output: This example will clear the display, then read the Center of ZERO status of the scale. If the scale is at zero, the message "CENTER OF ZERO" will be printed to the display.

DATA

Command: DATA constant [,constant]...

This is a non-executable command that stores the numeric and string constants used by a program's READ statements. If a string constant contains commas, colons, or leading or trailing spaces you want in your program, you must enclose the string in double quotes.

Example: PRINT "CITY", "STATE", " ZIP"
READ C\$, S\$, Z
PRINT C\$, S\$, Z
DATA "WEBB CITY,", "MO", 64870

Output: This example will print to the display the headings CITY, STATE and ZIP. Next it will read the strings (C\$, S\$) and the numeric (Z) data from the DATA statement. After reading the data, the data will print to the display.

DATE\$

System Variable: Set: DATE\$ = "MM-DD-YYYY"
Read: X = DATE\$

This is a system variable that can be used to Set or return (Read) the date in the 778's internal clock chip. The date will be returned as a string in the form "MM-DD-YYYY".

Example: CLS 0,0,127,63
DATE\$ = 04-12-1997
X\$=DATE\$
PRINT "DATE = ";X\$

DATE\$, Cont.

Output: This example will clear the display, then set the date in the 778's clock chip as 04-12-1997. Next it will read the current value of DATE\$ and print to the display "DATE = " and the value of DATE\$.



NOTE: The date and time used in the SmartWeigh simulator is from the system clock in the PC. Therefore, if you change the date and time in a simulator BASIC program, the change will occur for the PC clock also and may affect other operations on the PC.

DEF FN

Command: DEF FNname(argument) = expression

This is a command that defines and names a function written by the user. In Cardinal BASIC, DEF is a working equivalent of the FUNCTION command.

Example: CLS
DEF FNarea (radius) = 3.14159 * (radius ^ 2)
RAD = 12
PRINT "Radius = "; FNarea(rad)

Output: The example will clear the screen, define a function, and print to the display the results of the function, Radius = 452.16.

DEFSNG

Command: DEFSNG letter[-letter](,letter[-letter])...

This is a command that defines variables with single letter names (or a range of letter names) as numerical variables.

Example: DEFSNG L-P

Output: In this example, all variables beginning with the letters L, M, N, O and P will be defined as numeric variables.

DEFSTR

Command: DEFSTR letter[-letter](,letter[-letter])...

This is a command that declares all variables with single letter names (or a range of letter names) as string variables.

Example: DEFSTR A
A="120#"

Output: In this example, all variables beginning with the letter A will be defined as string variables.

DIM

Command: variable(elements...)[variable(elements...)]...

This is a command that specifies variables that have more than one element in a single dimension (arrayed variables). An array is a list of values all represented by the same variable name. **NOTE:** Only parentheses () are allowed for variable field delimiters.

Example: DIM TMP\$(10)

Output: This example will dimension a string array named Tmp\$ (\$ for a type string) with 11 elements, (0 ~ 10 = 11).

DO

Command: DO[WHILE expression]

The DO command implements a number of forms of program loops. A program loop is a statement that repeats a block of statements while a condition is true or until a condition becomes true. The DO commands are:

DO...LOOP, which simply loops. The only way out of the loop is by the EXIT command.
DO WHILE...LOOP, loops while the "expression" is true (*same as WHILE-WEND loop*).
DO...LOOP UNTIL, loops until the "expression" following UNTIL is true.

DO...LOOP

Example: CLS
I = 0
DO
 I = I + 1
 PRINT "I IS";I
 IF I > 6 THEN EXIT DO
LOOP
PRINT "END"

Output: This example will clear the display, the I is 1 ... I is 7 and the word END will print on the display.

DO WHILE...LOOP

Example: CLS
PRINT "START"
LET X = 0
DO WHILE X < 5
 PRINT "X is ";X
 LET X = X + 1
LOOP
PRINT "END"

DO, Cont.

Output: This example will clear the display, then the word START, I is 1 ... I is 7 and the word END will print on the display.

DO... LOOP UNTIL

Example: CLS
I = 0
DO
 I = I + 1
 PRINT "VALUE OF I IS";I
LOOP UNTIL I > 6

Output: This example will clear the display, then print I is 1... I IS 6 to the display.

DOTARE

Command: DOTARE [scalenumber]

This is a command that sets the current gross weight as the tare weight for the default scale or a selected scale.

Example: DOTARE
PRINT TARE

Output: This example will print to the display the new tare weight.

DOZERO

Command: DOZERO [scalenumber]

This is a command that zeros the default scale or a selected scale.

Example: CLS
DOZERO
PRINT GROSS

Output: This example will clear the display, then the current gross weight, 0 (zero) will print to the display.

DRAW

Command: DRAW X1,Y1,X2,Y2

This command will draw a line from the coordinates specified by X1, Y1 to X2, Y2. All coordinates must be within the range of the display, or error will be reported.

DRAW, Cont.



NOTE: The DRAW command uses the system variable DSPLOGIC settings 0 = SET (on), 1 = XOR (invert) and 2 = RESET (off) to draw. If the system DSPLOGIC is currently set to 3 = OVERWRITE (turn on foreground and turn off background), DRAW will default to DSPLOGIC setting 0 to perform the draw command. This is done only to perform the draw command and does not permanently change the system setting.

Example: CLS 0,0,127,63
 DRAW 14,0,14,50

Output: This example will clear the display, then draw a line from position 14, 0 to 14,50.

DSPFONT

System Variable: Set: DSPFONT = X
 Read: X = DSPFONT

This is a system variable that is used to select (Set) the current font for text output on the 778 display or when used in a expression, it can return (Read) the current value of the active font. There are three (3) fonts available and they are as follows:

- 1 = font 1, a 4 x 5 character in a 5 x 6 block
- 2 = font 2, a 5 x 7 character in a 6 x 8 block
- 3 = font 3, a 10 x 14 character in a 12 x 16 block.



NOTE: The 778 can only display upper case characters (capital letters) with font 1 selected. If font 2 or font 3 are selected, the 778 can display both upper (capital letters) and lower (small letters) case characters.

Example: CLS 0,0,127,63
 DSPFONT=1
 PRINT "PRESS ENTER KEY"
 X=DSPFONT
 PRINT "CURRENT FONT IS: ";X

Output: This example will clear the display, then print PRESS ENTER KEY in a 5 x 6 block in the upper left corner of the display. Next it will read the value current DSPFONT setting and print to the display "CURRENT FONT IS:" and the value of DSPFONT.

DSPLOGIC

System Variable: Set: DSPLOGIC = X
Read: X = DSPLOGIC

This is a system variable that is used to select (Set) the current display logic for output on the 778 display. It can also be used to return (Read) the current value of the active font.

There are four (4) logic settings available and they are as follows:

- 0 = SET pixels (turn on foreground pixels)
- 1 = XOR pixels (invert foreground pixels)
- 2 = RESETpixels (turnoff foreground pixels)
- 3 = OVERWRITE (turn on foreground and turn off background, for image loads and graphic text commands only).

Example: CLS 0,0,127,63
BOX 15,15,45,45,1
DSPLOGIC=1
X=DSPLOGIC
LOCATE 22,27
PRINT X

Output: This example will clear the display, then draw a solid box at the coordinates 15,15,45,45. Next it will invert the foreground pixels, read the current value of DSPLOGIC and using the LOCATE command, print the value of DSPLOGIC approximately in the middle of the solid box.

DSPX, DSPY

System Variable: Set: DSPX = X or DSPY = Y
Read: X = DSPX or Y = DSPY

The system variables DSPX and DSPY are used to specify the current display X and Y coordinates for text and graphics displays. The value of X must be a number between 0 and 127, while the value of Y must be a number between 0 and 63. DSPX and DSPY can also be used in an expression to return (Read) the current display print position.

Example: CLS
DSPX = 48
DSPY = 24
PRINT "HELLO"
X = DSPX
Y = DSPY
PRINT "DSPX=";X
PRINT "DSPY=";Y

DSPX, DSPY, Cont.

Output: This example will clear the display, set the coordinates of DSPX to 48 and DSPY to 24 and print "HELLO," at those coordinates. Next it will read the values of DSPX and DSPY and print the current display print position.

ELSE

Command: ELSE

ELSE is a command that introduces a default condition in a multi-line IF statement, when the IF statement is false.

Example: (For an example, see IF...THEN...ELSE command).

ELSEIF

Command: ELSEIF

ELSEIF is a command that introduces a secondary condition in a multi-line IF statement, when no previous condition is true.

Example: (For an example, see IF...THEN...ELSE command).

END

Command: END IF | FUNCTION | SELECT | SUB

This is a multipurpose command that is used to end a multi-line IF statement (END IF), a multi-line function (END FUNCTION), a select case statement (END SELECT), or a multi-line subroutine (END SUB).

Example: (For an example, see IF...THEN...ELSE command).

EOF

Command: EOF (file number)

This command returns a value of TRUE (-1) if the file associated with the file number is at the End-Of-File, (no more data to read). EOF returns a value of FALSE (0) while reading the file as long as there is still data to read.

Example: (For an example, see INPUT# command).

ERASE

Command: ERASE variable[, variable]...

This command eliminates arrayed variables from a program. After the arrayed variables have been erased, they may be redimensioned. An error will occur if an array is redimensioned without first erasing it.

Example: DIM A(100, 100)
ERASE A
DIM A(5)

Output: This example will first dimension the array "A", then erase the array "A" and then re-dimension it.

ERRNO

System Variable: Set: ERRNO = 0
Read: IF ERRNO = X THEN...

This system variable returns the error number (X) of the most recent error. A zero (0) will be returned if there is no error or a -1 will be returned if the ESC key has been pressed. As a system variable, ERRNO can also be used to set the error number to a known value which can then be checked to see if it has changed.

Example: CLS
ERRNO = 0
DO
 PRINT ERRNO
 INPUT "ENTER TIME (HHMM)",VALUE\$
 IF ERRNO=1 THEN EXIT DO
 IF VAL(VALUE\$)>2459 THEN ERRNO=1
 IF LEN(VALUE\$)<>4 THEN ERRNO=2
LOOP UNTIL ERRNO=0
END

Output: This example will set the ERRNO to the known value of 0 (zero). It will then perform a loop that prints the error number and prompt to the operator to enter the time. It will check the data entered for the correct range and number of digits and return an error number accordingly. When both checks result in no error, a (0) zero returned, the program will end.

EVENT

Command: EVENT number GROSS(scalenum) | NET(scalenum) OP var | literal value RELAY bdnm, relnum, cond

EVENT, Cont.

This is a command that sets up an EVENT for Cardinal BASIC (the operating system) to monitor in the background. When the condition is true and the specified RELAY is not in the state specified, the RELAY will be changed to the specified state. The 'number' is a unique number used to identify the event. Repeated EVENT statements with the same number replaces previous EVENT with the same number. The GROSS(scalenum) or NET(scalenum) where 'scalenum' is the scale number to monitor must be specified. The OP may be < (less than), <= (less than or equal to), > (greater than), or >= (greater than or equal to). A variable name 'var' or literal value may be specified. If a variable name is used, the condition will operate dynamically, that is, changes in the variable value will affect the EVENT in real time. The RELAY bdnnum, relnum, cond should be specified as in the RELAY command.

EVENT number OFF will turn off a specified event. If the 'number' is 0 (zero) all events will be turned off. **NOTE:** If no events are on, EVENT number OFF will produce an error.

Example: PRESET = 10000
 EVENT 1 NET(1) >= PRESET RELAY 1,1,1

Output: This example set up EVENT 1 to monitor the net weight. It will turn the first relay on the first board *on* when the NET weight is equal to or exceeds the preset amount of 10,000 pounds.

EXIT

Command: EXIT [FOR],[DO]

This command when used with DO will exit from a DO...LOOP loop. When combined with FOR (EXIT FOR), it will exit from a FOR...NEXT loop.

Example: (For an example, see the CALL command).

EXP

Function: EXP(number)

This is a math function that returns the exponential value of the 'number' in the argument.
(*e*, the base of natural logarithms raised to the power of number).

Example: CLS
 PRINT EXP (4)

Output: This example will clear the display, then print to the display, *e* to the 4th power, 54.59815.

EXTINP

Function: X=EXTINP(bdnum, inpnum)

This function will return the status of the external inputs. The 'bdnum' designates which board (1 to 4) and the 'inpnum' designates which input (1 to 8) on the board. The status returned will be a 1 (on) or a 0 (off).

Example: CLS 0,0,127,63
X=EXTINP (1,1)
PRINT X

Output: This example will clear the display, read the status of the first input of I/O board 1 and store that status value in "X". It will then print the value of "X" in the upper left corner of the display.

FIELD

Command: FIELD #file number, number AS string-variable\$. . .

This command allocates space in a random file buffer for the file indicated by the 'file number', allocating 'number' bytes and assigning the bytes at this position to the variable string-variable\$. (Also see the OPEN and PUT commands).

Example: OPEN "R", #1, "TRUCKS.DAT", 128
FIELD #1, 20 AS TRKID\$, 8 AS TRKWT\$, 10 AS TRKAC\$

Output: This example will open a file and then define the fields within the file.



NOTE: String variables referenced in the FIELD statement can not be used as variables anywhere else in the program. They can only be used by the FIELD and the LSET commands.

Example: NORMALVAR\$="ABCDEFGG"
FIELD #NUM, 10 as A\$
LSET A\$=NORMALVAR\$
PUT #NUM, RECNUM

FOR

Command: FOR counter = start TO finish [STEP increment]

This command initiates a FOR-NEXT loop with the variable counter initially set to start and incrementing in increment steps (default is 1) until counter equals finish.

FOR, Cont.

Example: FOR X = 1 TO 10 STEP 2
 PRINT X
 NEXT X

Output: This example will print to the display the numbers from 1 to 10 stepping by 2. The actual numbers printed are: 1, 3, 5, 7, 9.

FUNCTION

Command: FUNCTION

This command introduces a function definition, normally ending with END FUNCTION. In Cardinal BASIC, FUNCTION and DEF are working equivalents, so either can be used with single-line function definitions or with multi-line definitions terminated by END FUNCTION. When using the Function command, the code for a Function must be at the end of the mainline code (last instructions in program).

Example: END
 FUNCTION MULTICHARGE (SCALEWT, NUMTRUCKS, RATE)
 TOTALWT = SCALEWT * NUMTRUCKS
 TOTALRATE = TOTALWT * RATE
 MULTICHARGE=TOTALRATE
 END FUNCTION

Output: This FUNCTION example will compute the charge for multiple trucks crossing a scale, given the weight of each truck (SCALEWT), the number of trucks crossing (NUMTRUCKS) and the rate per pound (RATE).



NOTE: Variables created in a function are local to that function. Only variables defined with the COMMON command can be seen in the function and shared throughout the program.

GET

Command: GET #file number[,record number]

This command reads the string representation of variables stored in a file. If the file is a random access file and a record number is specified, the GET command reads the specified record.

GET, Cont.

Example: COMMON F.TRKID\$,F.TRKWT\$
COMMON TRKID\$,TRKWT
"R",#1,"TRUCK.DAT",24
FIELD #1,20 AS F.TRKID\$,4 AS F.TRKWT\$
GET #1,1
TRKID\$ = F.TRKID\$
TRKID\$ = RTRIM\$(TRKID\$)
TRKWT = CVS(F.TRKWT\$)
CLOSE #1

Output: This example will return the variables defined in the FIELD statement from the random access "R" file "TRUCK,DAT". The RTRIM\$ and CVS converts the variables from the format they are stored in, into a format the Cardinal BASIC program can use.

GOTO

Command: GOTO label

This command transfers the control of the program execution by branching unconditionally to a specified label. A 'label' is an identifying string that denotes a specific location in a program. At the specific location in the program that the 'label' is identifying, the 'label' string must be followed by a colon (:) and on a line by itself (no other characters including spaces).

Example: START:
CLS
LOCATE 10,30
PRINT "GROSS WEIGHT=";GROSS
GOTO START

Output: This example sets up an endless loop that will display the gross weight in the middle of the display.



NOTE: It is good programming practice to use structured control statements (CALL, FUNCTION, DO...LOOP, FOR...NEXT, IF..THEN...ELSE, or SELECT CASE) instead of GOTO statements. Especially avoid using the GOTO to branch into or out of a subprogram, multiline function, an IF THEN ... END IF or to end a DO ... LOOP or a FOR ... NEXT loop.

GROSS

Function: GROSS([scalenumber])

This is a function that returns (reads) the gross weight of the default scale or selected scale if followed by [scalenumber].

Example: CLS 0,0,127,63
PRINT GROSS

Output: This example will clear the display, then print to the display the gross weight of the default scale.

HEX\$

Function: HEX\$(number)

This is a math function that returns a string in hexadecimal (base 16) for the 'number'.

Example: CLS 0,0,127,63
DSPLOGIC=3
INPUT "ENTER A NUMBER", X
A\$ = HEX\$(X)
PRINT X; " DECIMAL IS EQUAL TO " ; A\$; " IN HEXADECIMAL"

Output: If the number 12 is entered, the result from this example will print to the display, 12 DECIMAL IS EQUAL TO C IN HEXADECIMAL

IF...THEN...ELSE

Command: IF expression THEN [statement [ELSE statement]]

This command evaluates the 'expression' and performs the THEN statement if it is true or (optionally) the ELSE statement if the expression is false. Cardinal BASIC allows multi-line IF statements with ELSE and ELSE IF cases, ending with END IF.

Example: INPUT "ENTER A NUMBER", X
IF X = 1 THEN
 PRINT "one"
ELSEIF X = 2 THEN
 PRINT "two"
ELSEIF X = 3 THEN
 PRINT "three"
ELSE
 PRINT "must be integer from 1-3"
END IF

IF...THEN...ELSE, Cont.

Output: In this example, a prompt is displayed to enter a number. The number entered (X) will be checked to see if it is between 1 and 3. If X is between 1 and 3 it will be printed to the display. If X isn't between 1 and 3, a error message will print to the display and the program will end.

INP

Function: INP(prompt1\$, prompt2\$, default [, prec, minval, maxval])

This is a function that prompts for an input and returns a number. It allows 778 display inputs to be performed with a minimum of programming code. For example, one method the INP function reduces code is to automatically display numeric key labels during the input and to return to your defined key labels after the input.

The 'prompt1\$' and 'prompt2\$' are the prompts displayed to the operator and *must* be in upper case (capital) letters. If a carriage return, CHR\$(13) is present in 'prompt2\$', it will make the prompt display on two (2) lines. The 'default' is the current value and will be used if nothing is input. The optional arguments 'prec', 'minval', and 'maxval' can be used to specify the precision and the minimum and maximum value (a numeric range) that will be allowed for the input. For example a 'prec' 20, a 'minval' 100 and a 'maxval' 1000, would require the input to be in an even increment of 20 between 100 and 1000 i.e. 260.

Example: TARWT = 400
TARWT = INP("TARE", "ENTER" +CHR\$(13)+" WEIGHT", TARWT)
PRINT "Tare is now: "; TARWT

Output: This example will first set the variable TARWT to 400. Next, it will display "TARE", in a large font, and then in a small font on two lines, prompt the operator to "ENTER WEIGHT", while displaying the current tare setting. A dash will be displayed below the current setting, indicating the indicator is ready to accept the input. After the value has been input, the display will clear, then print to the display "Tare is now: " and the value input. If a value is not entered at the "ENTER WEIGHT" prompt, and the ENTER key is depressed, the current ('default') TARWT value of 400 will be used.

INP\$

Function: INP\$(prompt1\$, prompt2\$, default\$ [, code\$, minlen, maxlen])

This is a function that prompts for an input and returns a string. It allows 778 display inputs to be performed with a minimum of programming code. For example, one method the INP\$ function reduces code is to automatically display numeric key labels during the input and to return to your defined key labels after the input.

The 'prompt1\$' and 'prompt2\$' are the prompts displayed to the operator and *must* be in upper case (capital) letters. The 'default' is the current value and will be used if nothing is input. The optional parameter 'code\$' may be used to specify what characters may be entered. The 'minlen' and 'maxlen' are optional arguments that may be used to specify the minimum and maximum length for data input.

The following are the 'code\$' allowed and their functions:

- ★ Any character allowed
- A Alphabetic characters only
- N Numeric 0 - 9 and decimal point characters only
- P Alphanumeric password input
- Q Numeric only password input only
(Asterisks ★ will be shown as password is entered on keypad)
- U Any character allowed, however lower case alphabetic characters are converted to upper case characters (a to A, b to B etc...)
- Y Yes or No input only
- Z Yes or No input only *with* a automatic ENTER key depression after selection.



NOTE: The ★, A, P and U 'code\$' all require the optional 101 key keyboard for alphabetic character input.

Example: A\$ = "TEST"
A\$ = INP\$("ID", "ENTER STRING", A\$)
PRINT "The ID is now: " ;A\$

Output: This example will first set the string variable A\$ to "TEST". Next, it will display "ID", in a large font, and then in a small font, the prompt to the operator "ENTER STRING", and the current ID setting. A dash will be displayed below the current setting, indicating the indicator is ready to accept the input. After the value has been input, the display will clear, then print to the display "The ID is now: " and the value input. **NOTE:** If a value is not entered at the "ENTER STRING" prompt, and the ENTER key is depressed, the current ('default') A\$ value of "TEST" will be used.

INPUT

Command: INPUT [;]["prompt string";][,]list of variables

This command allows input from the 778 during program execution. When an INPUT statement is encountered, the program pauses and displays a question mark to indicate it is waiting for data. If the "prompt string" is included, the string is displayed before the question mark. To suppress the question mark after the "prompt string", use a comma instead of a semicolon. The data that is entered is assigned to the variables specified in the 'list of variables'. The number of data items entered must be the same as the number of variables in the list. To eliminate the underlines beneath the characters being typed in, set DSPLOGIC=3 before the INPUT command.

Example: (For an example, see the IF...THEN...ELSE command).



NOTE: During an INPUT statement, the 14 soft keys on the front of the 778 are assigned to their standard numeric values and any pre-defined KEY events are suspended. The INPUT statement will not change the display except for the prompt text and the cursor. To display the numeric key labels, you will need to include a series of KEY statements, (without subroutines), to show the key labels before the INPUT. After the INPUT the KEY 0 statement can be used to remove these labels. Additional KEY statements may then be used to return the key labels for any KEY events that were previously defined. **NOTE:** The INP or INP\$ functions are a better method of obtaining a user input of data, as they perform the prompt, keylabels etc.. with one line of program code.

INPUT#

Command: INPUT #file number,variable list

This is a command that is used to read data from a file or device and assign them to program variables. The 'file number' is the number used when the file or device was OPENed for input. The 'variable list' is the variable names that will be assigned to the items in the file. The variable type must match the type specified by the variable name.

Example:

```
CLS
OPEN "TRUCKS.DAT" FOR INPUT AS #1
DO WHILE EOF(1)=0
    COUNT = COUNT + 1
    INPUT #1, WEIGHT
    TOTAL = TOTAL + WEIGHT
    PRINT "TRUCK";COUNT;" WEIGHS";WEIGHT
LOOP
PRINT
PRINT "TOTAL WEIGHT:";TOTAL
PRINT USING " AVG. WEIGHT: #####.#";TOTAL/COUNT
END
```

INPUT#, Cont.

Output: This example reads a series of truck weights from a sequential file, (TRUCKS.DAT). Next, it will print the number and the weight of each truck and calculate the total weight of the trucks. When finished reading the file, it calculates the average weight of the trucks and prints the total and average weight of the trucks.

INSTR

Function: INSTR([start position,] search string, substring)

This is a function that searches a string for a substring and returns an integer which represents the position in which the substring occurs in the string. The start position allows you to specify where in the string to begin searching and is optional.

NOTE: In Cardinal BASIC, INSTR cannot be used for assignments.

Example:

```
CLS
X$="PLEASE INPUT THE TARE WEIGHT"
PRINT X$
Y$="TARE"
PRINT INSTR(10,X$,Y$)
```

Output: This example will search the string PLEASE INPUT THE TARE WEIGHT for the substring TARE, starting at the tenth (10) position, then print to the display "18", the position where the match begins. If no match is found, INSTR returns a value of zero (0).

INT

Function: INT(number)

This is a math function that returns the largest integer less than or equal to the argument 'number'. This is not a "truncated" integer function, refer to CINT if a "truncated integer function is desired.

Example:

```
CLS 0,0,127,63
PRINT INT (7.8)
```

Output: This example will clear the display, then print a "7" to the display.

INTERVAL

Function: INTERVAL([scalenumbers])

This is a function that returns (reads) the interval (division size) for default scale or selected scale if followed by [scalenumbers].

INTERVAL, Cont.

Example: CLS 0,0,127,63
 PRINT INTERVAL

Output: This example will clear the display, then print to the display the interval value of the default scale.



NOTE: In the simulator, INTERVAL is set by selecting Options, Configure Weight Simulator... from the menubar. In the Indicator, INTERVAL is set in SETUP and CALIBRATION.

KEY

Command: KEY n, "sub-routine", "label"

This is a command that defines the function of the softkey positions KEY n (n = 1-14). Each time the key is pressed, the "sub-routine" is called to perform. The "label" is the description of the key displayed to the operator surrounded by a box. Setting a key sub-routine without a "label" will make the key active, but no label or box will show on the display. After a key has been defined, it can be disabled by setting the key sub-routine to an empty string " ", " ". To disable all previously defined keys use the command KEY 0.

Example: KEY 6,"PRINTKEY","PRINT"
 KEY 7,"LBKGKEY","LB/KG"
 KEY 8,"ZEROKEY","ZERO"
 KEY 9,"TAREKEY","TARE"
 KEY 10, " ", " "

Output: This example will define each of the softkeys, 6 to 10, and display their label. Key 6 is defined as the PRINT key, key 7 as the LB/KG key, key 8 as the ZERO key, key 9 as the TARE key and key 10 as "disabled".



NOTE: Since a KEY sub-routine may be called anywhere in a program, it is recommended to save the DSPLOGIC and DSP() coordinates if the KEY sub-routine will be writing or drawing to the display. The following example could be used to preserve the DSPLOGIC and DSP() anytime writing or drawing to the display is performed. The SUB label and variable names (X, Y) are given only as a suggestion.

KEY, Cont.

```
Example:  SUB XYZ()  
          LGC = DSPLOGIC  
          X = DSPX  
          Y = DSPY  
          .  
          . (your KEY sub-routine code would go here)  
          .  
          LOCATE X,Y  
          DSPLOGIC = LGC  
        END SUB
```



NOTE: Key sub-routines should not be called by the CALL sub-routine command, but if you find it necessary to call a key sub-routine, disable the key (KEY x, " ", " ") before the CALL and re-enable the key after the CALL command.

KILL

Command: KILL "filename"

This is a command that removes (deletes) a file. When executed on the 778 indicator, it will delete the file from the RAM directory in the indicator memory. When executed from the SmartWeigh simulator, it will delete the file on the computer's hard drive.

Example: KILL "TRUCKS.DAT"

Output: This example will delete the file named "TRUCKS.DAT".



NOTE: You must specify the full "filename" (include an extension if any) when using the KILL command. If a KILL command is given to a file that is currently opened, a "File already open" error will occur. Also, an error will occur if you execute the KILL command to a file that doesn't exist.

LEFT\$

Function: LEFT\$(string\$, number-of-spaces)

This is a function that returns a substring from a 'string\$' with 'number-of-spaces' from the left (beginning) of the string\$. LEFT\$ cannot be used for assignment.

```
Example:  INPUT "PRINT THE TICKET";YESNO$  
          IF LEFT$(YESNO$, 1) = "Y" THEN PRINT-TICKET
```

LEFT\$, Cont.

Output: This example will prompt the user to "PRINT THE TICKET" and store the response in the string YESNO\$. If the character is Y, the program calls the PRINT-TICKET procedure to print a ticket. If the character is anything else, the PRINT-TICKET procedure is skipped.

LEN

Function: LEN(string\$)

This is a function that returns an integer that is the number of characters in a 'string\$'.

Example: CLS
 INPUT "ENTER THE TICKET NUMBER"; TKTNUM\$
 NUMCHAR = LEN(TKTNUM\$)
 IF NUMCHAR = 6 THEN
 PRINT NUMCHAR
 ELSE PRINT "NUMBER OUT OF RANGE"
 END IF

Output: This example will prompt the user to enter the ticket number, set the variable NUMCHAR to the length of the ticket number, verify that six (6) digits were entered, then print to the display, the value of NUMCHAR. If NUMCHAR is not six (6) digits, then NUMBER OUT OF RANGE will print to the display.

LET

Command: LET variable = expression

This is a command that assigns the value of 'expression' to the variable. Cardinal BASIC supports implied LET statements (e.g., "X = 4.5678", but does not support assignment to multiple variables (e.g., "x, y, z = 3.141596").

Example: LET A = 5

Output: This example assigns the value of 5 to the variable A.

LINE INPUT

Function: LINE INPUT [#] device-number,["prompt string";] string variable

This is a command that reads an entire line from the keyboard or a file or device into a string variable. If input is from the keyboard, then "prompt string" will be printed first. Unlike INPUT, LINE INPUT reads a whole line, ignoring delimiters.

LINE INPUT, Cont.

Example: PRINT "ENTER TRUCK ID. TO STOP, PRESS <ENTER> WITHOUT ";
 PRINT "ENTERING AN ID" : PRINT
 OPEN "TRUCKID.DAT" FOR OUTPUT AS #1
 DO
 LINE INPUT "->";ID\$
 IF ID\$ <> "" THEN PRINT #1, ID\$
 LOOP WHILE ID\$ <> ""
 CLOSE #1

Output: This example enables the user to enter a trucks ID into a file in memory. The LINE INPUT statement allows you to enter any characters, including those (such as a comma) that are delimiters in a regular INPUT statement. It will continue until the ENTER key is press by itself.

LOC

Function: LOC(device-number)

This is a function that returns the next record that GET or PUT statements will use.

Example: IF LOC(1) > 50 THEN STOP

Output: This example stops the program if the current file position is beyond 50.

LOCATE

Command: LOCATE X,Y

This is a command that sets the current X,Y coordinates for output to the 778 display. Another method to set the X,Y coordinates of the 778 display is to use the DSP? system variables. (See DSPX, DSPY).

Example: CLS
 LOCATE 10,55
 PRINT "WEIGHT ="

Output: This example will clear the display, then print to the display at coordinates X=10 and Y=55, "WEIGHT ="

LOF

Function: LOF(file number)

This is a function that returns the length of a file (specified by file number) in bytes.

LOF, Cont.

Example: CLS
OPEN "R", #TRKS, "TRUCKS.DAT", 128
NUMRECS = (LOF(TRKS)/128)
PRINT NUMRECS

Output: This example will clear the display, then calculate the number of records in the file TRUCKS.DAT, by dividing the length of the file by the record length, and print the answer to the display.

LOG

Function: LOG(number)

This is a math function that returns the natural logarithm of the argument 'number'.

Example: CLS
PRINT LOG(10)

Output: The example will clear the display, then print to the display 2.302585, the logarithm of ten (10).

LSET

Command: LSET string-variable\$ = expression

This is a command that left justifies the data from 'expression' in a string variable or random access buffer field. **NOTE:** In the Example below, " _ " represents a space.

Example: A\$ = " _ _ _ _ _ _ _ _ _ _"
LSET A\$ = "LEFT"

Output: This example will left justify the string LEFT in a ten (10) character field.

LTRIM\$

Command: LTRIM\$(stringexpression\$)

This is a command that removes leading spaces from a string. The 'stringexpression\$' can be any string expression. **NOTE:** In the Example below, " _ " represents a space.

Example: CLS
A\$ = " _ _ _ BASIC _ _ _"
PRINT "*" + LTRIM\$(A\$) + "*"

Output: This example clears the display, then prints *BASIC_ _ _* to the display.

MID\$

Function: MID\$(string\$, start-position-in-string[, number-of-spaces])

This is a function that searches a 'string\$' for a substring and returns the substring, beginning at the 'start-position-in-string' and continuing for the number of spaces specified. The [number-of-spaces] is optional. If omitted, MID\$ returns a substring consisting of all the characters from the start position to the end of the string\$.

Example: CLS
A\$ = "PRESS"
B\$ = "CLEAR ZERO ENTER"
PRINT A\$; MID\$ (B\$,6,5)

Output: This example will clear the display, then print to the display PRESS ZERO. This is the result of A\$ and the MID\$ of B\$, (starting at position six (6) for five (5) spaces).

MKS\$, MKI\$, MKD\$

Function: MKS\$(number) MKI\$(number) MKD\$(number)

The function MKS\$ (convert to a single precision string) converts the numerical value 'number' into a string\$ which can be stored in a more compressed form in a file (especially for random file access). MKI\$ (convert to a integer precision string) and MKD\$ (convert to a double precision string) are provided to ease converting programs written in other forms of BASIC to Cardinal BASIC. Cardinal BASIC is single precision, therefore the functions MKS\$, MKI\$ and MKD\$ are effectively equivalent. (Also see the CVS, CVI, CVD function).

Example: AMT=(K+T)
FIELD #1, 4 AS D\$, 20 AS N\$
LSET D\$=MKS\$(AMT)
PUT #1

Output: In this example, the first line defines AMT as the sum of K and T. The second line defines the fields in file 1. The third line converts AMT into a left justified string (D\$). The PUT #1 in the last line writes the data to the file.

MOTION

Function: MOTION([scalenumbers])

This is a function that returns (reads) the state of motion status for default scale or selected scale. A '-1' (false) will be returned if the scale is in motion.

MOTION, Cont.

Example: CLS 0,0,127,63
 IF MOTION = -1 THEN
 PRINT "IN MOTION"
 ELSE
 END IF

Output: This example will clear the display, then read the MOTION status of the scale. If the scale is unstable (in motion), the message "IN MOTION" will be printed in the upper left corner of the display.

NAME

Command: NAME "old filename" AS "new filename"

This is a command that is used to change the name of a file. The "old filename" must exist and the "new filename" must not exist, otherwise an error will occur.

Example: NAME "TRUCKS.DAT" AS "OLDTRKS.DAT"

Output: In this example, the file formerly named "TRUCKS.DAT" will be changed to the new file name of "OLDTRKS.DAT". This will be performed without changing the content of the file.

NET

Function: NET([scalenumbr])

This is a function that returns (reads) the net weight of default scale or selected scale if followed by [scalenumbr].

Example: CLS 0,0,127,63
 PRINT NET

Output: This example will clear the display, then print to the display the net weight of the default scale.

NEXT

Command: NEXT [counter-variable]

This is a command that comes at the end of a FOR-NEXT loop.

Example: (For an example, see the FOR command).

OCT\$

Function: OCT\$(number)

This is a math function that returns a string in octal (base 8) for the 'number'.

Example: CLS
INPUT "ENTER A NUMBER", X
A\$ = OCT\$(X)
PRINT X; " DECIMAL IS EQUAL TO " ; A\$; " IN OCTAL"

Output: If the number 27 is entered, the result from this example will print to the display, 27 DECIMAL IS EQUAL TO 33 IN OCTAL

OPEN

Command: OPEN mode, #file number, "filename", [record length]

This is a command that enables I/O (input/output) to a file.

The 'mode' is a string expression with one of the following characters (which must be enclosed in quotation marks):

- "O" denotes sequential Output (writes to a file)
- "I" denotes sequential Input (reads from a file)
- "R" denotes Random-access input and output

The '#file number' is a number that is associated with a file. This association remains opened until a CLOSE or CLOSE # filename command is executed. The "filename" is the name of the file. It is always a string value and must be enclosed in quotation marks. The [record length] is an optional integer that sets the record length to be used for files. If omitted the record length defaults to 128-byte records. A file must be opened before it can be used. With different file numbers used, more than one file can be opened at one time.

Example: COMMON F.TRKID\$,F.TRKWT\$
COMMON TRKID\$,TRKWT
TRKID\$ = "B4549"
TRKWT = GROSS(1)
OPEN "R",#1,"TRUCK.DAT",24
FIELD #1,20 AS F.TRKID\$,4 AS F.TRKWT\$
LSET F.TRKID\$ = TRKID\$
LSET F.TRKWT\$ = MKS\$(TRKWT)
PUT #1,1
CLOSE #1

OPEN, Cont.

Output: In this example, the variables are declared COMMON and values set for the variables. Next a file "TRUCKS.DAT" is opened as a "random-access file, with a record length of 24 bytes. The FIELD statement defines the various fields (and their length) in the file. The values in each variable are left justified and written (PUT) to the file. The last statement will close the file. (Also see COMMON, CLOSE, FIELD, LSET, MKS\$, and PUT commands).



NOTE: Opening a file as "R", RANDOM access, allows both read (GET) and writes (PUT) of data, without the need to close the file and re-open it. If a file is opened as "O", OUTPUT, you can only PUT (write) data to it. If a file is opened as "I", INPUT, you can only GET (read) data from it.

OPEN ... FOR ... AS

Command: OPEN "filename" FOR [mode] AS #file number [record length]

This is a command that enables I/O (input/output) to a file or device. The "filename" is the name of the file. It is always a string value and must be enclosed in quotation marks. The [mode] is the operation to perform on the file, INPUT or OUTPUT. The '#file number' is a number that is associated with the file and remains opened until a CLOSE or CLOSE# filenumber command is executed. The [record length] is an optional integer that sets the record length to be used for files. If omitted the record length defaults to 128-byte records.

A common use of this form of the OPEN command, is to open the serial (COMn:) or parallel (LPTn:) ports for communication to a printer or other device. The ports can be configured for either INPUT or OUTPUT (mode) and are referenced by the file number once opened.

The following is a list of parameters (and their meaning) needed to open a port:

OPEN "COMn:[baud, parity, data, stop][,b]" FOR [mode] AS #file number[record length]
OPEN "LPTn:" FOR [mode] AS #file number [record length]

COMn: n=1 to 34

baud = 1200, 2400, 4800, 9600, 19200

parity = E (even), O (odd), S (space), M (mark), or N (none)

data = 7 or 8 bits

stop = 1 or 2 bits

b (binary) = sends a <CRLF > (carriage return-line feed)

LPTn: n=1 to 32

OPEN ... FOR ... AS, Cont.



NOTE: The default mode for COM ports on the 778 is text mode, which on input, translates a <CRLF> (carriage return-line feed) into a <LF> (line feed) only. If the input data ends with a <CR> only, the input won't be terminated until another character is received after the <CR>. To avoid this, open the COM port in binary mode [,b] which doesn't do any translation.

Example: OPEN "COM1: 9600, n,8,1,b" FOR OUTPUT AS #1
 PRINT #1,"TICKET"; TKTNO
 PRINT #1,"GROSS"; GRWT
 PRINT #1,"TARE"; TRWT
 PRINT #1,"NET: ";NTWT
 CLOSE #1

Output: This example will open COM1: in the binary mode (no translation) for output as file 1 (#1). It will output (print #1) several headings and the data from the specified variables, then close the file. (Also see the CLOSE command).

OVERCAP

Function: OVERCAP ([scale number])

This is a function that returns (reads) the state of the overcapacity status for default scale or selected scale. A '-1' (true) will be returned if the scale is not overcapacity.

Example: CLS 0,0,127,63
 IF OVERCAP = -1 THEN
 PRINT "OVER CAPACITY"
 ELSE
 END IF

Output: This example will clear the display, then read the OVER CAPACITY status of the scale. If the weight on the scale is over the capacity of the scale, the message "OVER CAPACITY" will be printed in the upper left corner of the display.

PLOT

Command: PLOT X,Y

This is a command that will turn on one (1) pixel of the display at the coordinates specified by "X" and "Y". All "X" coordinates must be within the range of 0 to 127 and all "Y" coordinates must be within the range of 0 to 63, or error will be reported.

PLOT, Cont.



NOTE: The PLOT command uses the system variable DSPLOGIC settings 0 = SET (on), 1 = XOR (invert) and 2 = RESET (off) to plot. If the system DSPLOGIC is currently set to 3 = OVERWRITE (turn on foreground, turn off background), PLOT will default to DSPLOGIC setting 0 to perform the plot command. This is done only to perform the plot command and does not permanently change the system setting.

Example: CLS 0,0,127,63
 FOR X = 41 TO 86 STEP 5
 FOR Y = 9 TO 54 STEP 5
 PLOT X,Y
 NEXT Y
 NEXT X

Output: This example will clear the display, then turn on pixels, beginning at column forty-one (41) and row nine (9) until it a box made up of one hundred (100) pixels (5 pixels apart) is in the center of the display.

PRINT

Command: PRINT [# device-number,][USING format-string\$:] expressions...

This is a command that outputs (prints) text to the display or a device specified by the device-number. Expressions to be printed must be separated by a comma " , " (begins printing a the next zone (prints like columns)), the semicolon " ; " (prints immediately after the last value) or the plus sign " + " (combines the strings then prints immediately after the last value). **NOTE:** Do not use expressions separated by blanks or tabs.

If PRINT USING is specified, the following formatting marks may appear in the format-string:

!	prints the first character of a string
\\	prints 2+x characters of a string, (where x = the number of spaces between the backslashes)
&	variable-length string field
#	represents a single digit in output format for a number (numbers are rounded as necessary)
.	decimal point in a number
+	sign of a number (will output + or -)
-	trailing minus after a number
**	fill leading spaces with asterisks
\$\$	output dollar sign in front of a number
^^	output number in exponential format
_	(underscore) output next character literally

PRINT, Cont.

Example: A = 1.15
 B\$ = "AAA"
 PRINT 3*A
 PRINT USING "\$#.## \\"; A;B\$;A;B\$

Output: This example first defines the values of A and B\$. The first PRINT will print to the display, the unformatted answer of 3 times A (3.45). The second print uses 8 print formatter commands to display the numeric variable A in a \$X.XX format, followed by a space and two (2) digits of the text in variable B\$. The same formatter commands are also used to print the third and 4th variables.



NOTE: The PRINT command is meant for text output to printers or the 778 display and requires a <CR-LF> be sent every 128 characters. If your data does not contain a <CR-LF> every 128 characters, Cardinal BASIC will send one when 128 characters has been reached. This makes the PRINT command unusable for devices where continuous data output is required and the data does not contain a <CR-LF> (a scoreboard for example). Use the PUT command if continuous data output is required.

Do not use the PRINT command to send a CHR\$(02) "STX" control character to a device. The CHR\$(02) will be interpreted as an ASCII character and spaces equal to its ASCII value will be sent to the device instead of the control character. Use the PUT command to send a CHR\$(02) "STX" control character to a device.

PUT

Command: PUT #file number[,record number]

This is a command that outputs the next available record or the record specified by record-number to the file denoted by file number.

Example: OPEN "R",#DATAOUT,"COM2:9600,N,8,1",15
 DATAOUT\$ = CHR\$(02)+WEIGHT\$+UNIT\$+MO\$+OC\$+CHR\$(13)
 FIELD #DATAOUT,15 AS F.DATAOUT\$
 RSET F.DATAOUT\$=DATAOUT\$
 PUT #DATAOUT
 CLOSE #DATAOUT

Output: This example will open a serial port (COM2:) and output continuous data to a scoreboard or other serial device. Fifteen characters are sent starting which a <STX> (start of transmission), the weight, the units (lb or kg), motion status, over capacity status and ends with a <CR> carriage return.

RANDOMIZE

Command: RANDOMIZE number

This is a command that seeds the random number generator (see the RND function). Under Cardinal BASIC, the TIMER function can be used to supply a (number) seed for the random number generator. This is the only way to insure random numbers each time the program is run.

Example: CLS
RANDOMIZE TIMER
FOR X = 1 TO 5
PRINT RND
NEXT X

Output: This example will clear the display, then print five (5) random numbers to the display.

READ

Command: READ variable[, variable]...

This is a command that reads values from DATA statements and assigns these values to the named variables. Variable types in a READ statement must match the data types in DATA statements. See also DATA and RESTORE commands.

Example: PRINT "CITY", "STATE", " ZIP"
READ C\$, S\$, Z
PRINT C\$, S\$, Z
DATA "WEBB CITY,", "MO", 64870

Output: This example will print to the display the headings CITY, STATE and ZIP. Next it will read the strings (C\$, S\$) and the numeric (Z) data from the DATA statement. After reading the data, it will print it to the display.

READY

Command : READY (device number)

This is a command that reads the status of the SIO (Serial Input/Output) card's DSR (data set ready) input and the PIO (Parallel Input/Output) card's select, paper out and error status'. The 'device number' is the device number used in the OPEN statement for the device. If the device is an SIO board (COM3--34), READY will return a "-1" if DSR is active and a "0" if it is not active. If the device is a PIO board (LPT1--32), READY will return a "-1" if the printer is selected, the paper is in and there is no error. Otherwise it will return a "0". **NOTE:** COM1 and COM2 do not have handshaking (DSR) and will always return a status value of -1 (active).

READY, Cont.

Example: PRINTER2 = 2
 OPEN "LPT2:" FOR OUTPUT AS #PRINTER2
 IF READY (PRINTER2) = 0 THEN
 PRT.STATUS\$ = "BAD"
 ELSE
 PRT.STATUS\$ = "GOOD"
 END IF

Output: This example will open LPT2: for output as device number PRINTER2. It will then check PRINTER2 to determined it's status.

RELAY

System Variable: Set: RELAY bdnun, relnum, cond
 Read: X=RELAY (bdnum, relnum)

This is a system variable that is used to enter (Set) the relays on the optional relay output board or return (Read) the status of the relay selected. The "bdnum" designates which board (1 to 4) , the "relnum" is which relay (1 - 8) and the "cond" is the condition of the relay, 1 = on and 0 = off. The status of the relay read is the same as the "cond" when setting, 1 = on and 0 = off.

Example: CLS 0,0,126,63
 FOR I = 1 TO 8
 RELAY 1,I,1
 PRINT "ON";I
 FOR X=1 TO 150
 NEXT X
 NEXT I
 FOR I = 1 TO 8
 RELAY 1,I,0
 CLS 0,0,126,63
 PRINT "OFF";I
 FOR X=1 TO 150
 NEXT X
 NEXT I

Output: This example will turn on each relay, print to the display, which relay is being turned on and then repeat the sequence turning off each relay and printing to the display which relay is being turned off.

REM

Command: REM string

This is a command that allows remarks to be included in a program. The entire line following REM is ignored.



NOTE: Cardinal BASIC does not support the use of the single quotation mark (') to denote remarks or placing REM statements at the end of a line of code. REM statements should be placed on a line by themselves before the line of code.

Example: REM ** Initialize global variables
GRWT = 500
TRWT = 35

Output: This example explains to the person reading the program listing, that the next program statement(s) will "Initialize global variables".

RESTORE

Command: RESTORE label

This is a command that resets the label position counters for DATA and READ statements to the top of the program file or to the beginning of the specified label.

Example: READ NAME\$, ID\$
RESTORE
READ NAME2\$, ID2\$
DATA "CARDINAL SCALE", "TRUCK 1"

Output: This example will allow the DATA statement to be reread by the second READ statement to assign the string "CARDINAL SCALE" to both NAME\$ and NAME2\$ and "TRUCK 1" to both ID\$ and ID2\$.

RIGHT\$

Function: RIGHT\$(string\$, number-of-spaces)

This is a function that returns a substring a string\$ with number-of-spaces from the right (end) of the string). RIGHT\$ cannot be used for assignment.

Example: DO
INPUT "ENTER THE PASSWORD"; PW\$
IF RIGHT\$(PW\$, 4) = "3579" THEN
PRINT "PROCEED"
CALL MENU
LOOP

RIGHT\$, Cont.

Output: This example will prompt the user to "ENTER THE PASSWORD" and store the response in the string PW\$. If the last four characters entered are "3579" the program will call the sub-routine MENU and continue. If anything else was entered, the program will loop until the correct password is entered.

RND

Function: RND(number)

This is a function that returns a pseudo-random number. The (number) value is ignored by Cardinal BASIC if supplied. The RANDOMIZE command reseeds the random-number generator.

Example:

```
CLS
FOR X = 1 TO 5
PRINT INT (RND * 101);
NEXT X
```

Output: This example will clear the display, then print five (5) random numbers from 0 - 100 to the display.

RSET

Command: RSET string-variable\$ = expression

This is a command that right justifies the data from 'expression' in a string variable or random access buffer field. **NOTE:** In the Example below, " _ " represents a space.

Example:

```
A$ = " _ _ _ _ _ _ _ _ _ _"
RSET A$ = "RIGHT"
```

Output: This example will right justify the string RIGHT in a ten (10) character field.

RTRIM\$

Command: RTRIM\$(stringexpression\$)

This is a command that removes trailing spaces from a string. The 'stringexpression\$' can be any string expression. **NOTE:** In the Example below, " _ " represents a space.

Example:

```
CLS
A$ = " _ _ _ BASIC _ _ _"
PRINT "*" + RTRIM$(A$) + "*"
```

Output: This example clears the display, then print to the display, * _ _ _BASIC*.

SELECT CASE

Command: SELECT CASE expression

This is a command that introduces a multi-line conditional selection statement. The expression given as the argument to SELECT CASE will be evaluated by the CASE statements (CASE, CASE ELSE, CASE IF) following the command. The SELECT CASE statement concludes with an END SELECT statement.



NOTE: CASE statements may be followed by string values, but only simple comparisons (equals, CASE IS = or not equal, CASE IS <>) can be performed.

Example:

```
CLS
PRINT "PLEASE ENTER"
INPUT "GATE NUMBER (1-4): ", GATE
SELECT CASE GATE
    CASE 3 TO 4
        PRINT "DIRECT TRUCK TO"
        PRINT "SCALE 3 FOR"
        PRINT "PAPER & PLASTIC"
    CASE 1 TO 2
        PRINT "DIRECT TRUCK TO"
        PRINT "SCALE 2 FOR"
        PRINT "SCRAP METAL"
        PRINT " & ALUMINUM"
    CASE ELSE
        PRINT "INVALID GATE NUMBER"
        PRINT "PLEASE CHECK #"
        PRINT "AND RE-ENTER"
END SELECT
```

Output: This example will print to the display the input prompt "PLEASE ENTER GATE NUMBER (1-4)". It will then take the value entered, store it in the variable GATE and based on the value of GATE, perform different actions (print instructions on the display to the operator).

SGN

Function: SGN(number)

This is a function that returns the sign of the argument (number). SGN(number) will return a " 1 " for positive numbers, a " 0 " will be returned for 0, and a " -1 " will be returned for negative numbers.

Example: PRINT SGN(10)

Output: This example will print a " 1 " to the display.

SIN

Function: SIN(number)

This is a math function that returns the sine of the argument (number or numeric expression) with the result given in radians. The numeric-expression, can be any numeric type. You can convert an angle measurement from degrees to radians by multiplying the degrees by $\pi/180$, where $\pi = 3.141593$. To convert a radian value to degrees, multiply it by 57.2958.

Example: X = SIN(4)
PRINT X

Output: The result of this SIN example will print to the display, -1.513605, the sine of 4 radians.

SPACE\$

Function: SPACE\$(number)

This is a function that returns a string of blank spaces, the length determined by the argument (number).

Example: PRINT SPACE\$(10); "TICKET # = "

Output: This example will print to the display, _____TICKET # =.
NOTE: In this example the _____ represent 10 blank spaces.

SPC

Function: SPC(number)

This is also a function that returns a string of blank spaces, the length determined by the argument (number).

Example: (For an example, see the SPACE\$ command).

SQR

Function: SQR(number)

This is a math function that returns the square root of the argument (number).

Example: PRINT SQR(12)

Output: This example will print to the display, 3.464102, the square root of 12.

STR\$

Function: STR\$(number)

This is a function that returns a string representation of the argument (number).

Example: A\$ = STR\$(644)
PRINT A\$

Output: This example prints to the display the string\$ 644. **NOTE:** The output from the example A\$ = STR\$(644) is the same as the statement A\$ = "644".

STRING\$

Function: STRING\$(number, ascii-value|string\$)

This is a function that returns a string (with the length determined by the 'number') made from the 'ascii-value' (or another 'string\$'). The STRING\$ function is useful for printing borders on the display or displaying lines to separate blocks of text.

Example: A\$ = STRING\$(10,45)
PRINT A\$; " REPORT "; A\$

Output: This example will print to the display "----- REPORT -----". The 45 in this example is the decimal equivalent of the ASCII symbol for the minus (-) sign or dash. **NOTE:** A\$ = STRING\$(10, "-") produces the same display as the example given.

SUB

Command: SUB subroutine-name()

This is a command that introduces a named, multi-line subroutine (sub program). The subroutine is called by a CALL statement, and concludes with an END SUB statement.

Example: SUB TAREINP()
PRINT "What is the"
PRINT "Tare"
INPUT "Weight"; TRWT
END SUB

Output: When called by the main program, this example will print to the display, the prompt to enter the tare weight, "What is the Tare Weight". It will then store the value in TRWT and end the sub-routine.



NOTE: Variables created in a subroutine are local to that subroutine. Only variables defined with the COMMON command can be seen in the subroutine and shared throughout the program.

SWAP

Command: SWAP variable, variable

This is a command that swaps the values of two variables. The two variables must be of the same type (either numerical or string).

Example: A = 10
B = 25
SWAP A, B
PRINT "A WAS 10 BUT NOW IS: "; A
PRINT "B WAS 25 BUT NOW IS: "; B

Output: This example sets the value of the A and B. Next it swaps the value of A and B, then prints the original and changed values of A and B.

TAN

Function: TAN (number or numeric-expression)

This is a math function that returns the tangent of the (number or numeric-expression) with the result given in radians. The numeric-expression, can be any numeric type. You can convert an angle measurement from degrees to radians by multiplying the degrees by $\pi/180$, where $\pi = 3.141593$. To convert a radian value to degrees, multiply it by 57.2958.

Example: CLS
X = TAN(25)
PRINT X

Output: The example clears the display, then prints -.1335264 to the display.

TARE

System Variable: Set: TARE [scalenumber,] tare
Read: TARE ([scalenumber])

This is a system variable that is used to enter (Set) a keypad tare for the default scale or a selected scale or return (Read) the tare weight of the currently selected scale.

Example: CLS 0,0,127,63
TARE 1, 100
X=TARE (1)
PRINT "TARE WEIGHT IS: "X

Output: This example will clear the display, then set the TARE weight to 100 lbs. Next it will read the current value of TARE and then print to the display "TARE WEIGHT IS: " and the value of TARE.

TARETYPE

Function: TARETYPE ([scalenumber])

This is a function that returns the type of tare stored for the default scale or a selected scale. There are three (3) types of tare available:

0 = no tare stored

1 = pushbutton tare

2 = keypad tare

Example: CLS 0,0,127,63
X=TARETYPE (1)
PRINT "TARE TYPE IS: ";X

Output: This example will clear the display, then read the current value of TARETYPE and print to the display "TARE TYPE IS: " and that value.

TIME\$

System Variable: Set: TIME\$ = "HH:MM[:SS]"
Read: X = TIME\$

This is a system variable that can be used to Set the time for the 778 or return (Read) the 778's internal clock chip. The time will be returned as a string in the form "HH:MM:SS".



NOTE: The time is displayed and entered in a 24 hour format with 12 added to all times after noon, i.e. 9AM would be "09:00", 3 PM would be "15:00" and midnight would be "00:00".

Example: CLS 0,0,127,63
TIME\$ = "10:35:00"
X\$=TIME\$
PRINT "TIME = ";X\$

Output: This example will clear the display, then set the TIME in the 778's clock chip as 10:35:00 am. Next it will read the current value of TIME\$ and print to the display "TIME = " and the value of TIME\$.



NOTE: The date and time used in the SmartWeigh simulator is from the system clock in the PC. Therefore, if you change the date and time in a simulator BASIC program, the change will occur for the PC clock also and may affect other operations on the PC.

TIMER

Function: TIMER

This is a function that returns the time in the system clock in seconds elapsed since midnight. For example, 2:15:02 p.m. will print to the display as 51302.

```
Example:  CLS 0,0,127,63
          PRINT TIMER
          T = TIMER + 10
            WHILE TR < T
              TR = TIMER
              IF TR < 1 THEN T = 0
            WEND
          PRINT TIMER
```

Output: This example will clear the display then read the internal clock of the 778 and print to the display the time (in seconds) past (00:00) midnight. Next it will perform the WHILE WEND loop (which is a 10 second delay) and again print to the display the time (in seconds) past midnight (00:00).

UNIT\$

Function: UNIT\$([scalenum])

This is a function that returns a string for the weight units of the default scale or a selected scale. The string returned will be "LB" for pounds or "KG" for kilograms.

```
Example:  CLS 0,0,127,63
          X$=UNIT$
          PRINT "UNITS = ";X$
```

Output: This example will clear the display, then read the current value of UNIT\$ and print to the display "UNITS = " and the value of UNIT\$.

VAL

Function: VAL(string\$)

This is a function that returns the numerical value of the string\$. The VAL function stops reading the string at the first character that it cannot recognize as part of a number. The VAL function also strips leading blanks, tabs, and line feeds from the argument string.

```
Example:  INPUT "ENTER THE DATE (MM-DD-YY)"; X$
          PRINT VAL(X$)
```

VAL, Cont.

Output: This example will prompt the operator to enter the date in the format MM-DD-YY. Next it will print to the display, the VAL of the date entered. For example, if you entered 12-22-96 at the prompt, PRINT VAL(X\$) will print to the display "12". This is because VAL reads everything up to the first non-numeric character ("- " in this case).

WHILE ... WEND

Command: WHILE expression

This is a command that initiates a WHILE-WEND loop. A WHILE-WEND loop is used to execute a series of statements as long as the "expression" is TRUE (-1).

Example: X = 0
WHILE X < 20
 PRINT "X = ";X
 X = X + 1
WEND

Output: This example will set the value of X to 0, then check and print the value of X to the display while X is less than 20. When the value of X equals 20 the loop will end.

WRITE

Command: WRITE [# device-number,] element [, element]....

This is a command that outputs variables to the display, or a device specified by the device-number, a COM port for example. WRITE inserts commas between the expressions as they are written and delimits (encloses) strings with quotation marks.

Example: CLS 0,0,127,63
A\$ = "TRUCK ID"
B = 45123
WRITE A\$,B

Output: This example will clear the display, then define the string variable A\$ and the integer variable B. Next, it will write the value of A\$ and B to the display.

WTMODE

Function: WTMODE([scalenumber])

This is a function that returns (reads) the weight mode for the default scale or selected scale if followed by [scalenumber]. The weight modes are:

1 = Pounds only

2 = Kilograms only

3 = Pounds/Kilograms

Example: CLS 0,0,127,63
 PRINT WTMODE

Output: This example will clear the display, then print to the display the weight mode (wtmode) of the default scale.

SUMMARY OF CAUTIONS

ARC

All coordinates must be within the range of the display, or an error will be reported.

BOX

All coordinates must be within the range of the display, or an error will be reported.

CLS

When CLS is used, the display position is set to the upper left corner of the CLS region. For example: CLS (without coordinates) will set the display position to the upper left corner of the display, while CLS 20,20,60,40 will clear only that region of the display and set the display position to the upper left corner (20,20) of the region.

COMMON

COMMON statements should be placed at the beginning of the program before any operational code (statements).

DATE\$

The date and time used in the SmartWeigh simulator is from the system clock in the PC. Therefore, if you change the date and time in a simulator BASIC program, the change will occur for the PC clock also and may affect other operations on the PC.

DIM

Only parentheses () are allowed as delimiters for variable fields.

DRAW

The DRAW command uses the system variable DSPLOGIC settings 0 = SET (on), 1 = XOR (invert) and 2 = RESET (off) to work. If the system DSPLOGIC is currently set to 3 = OVERWRITE (turn on foreground and turn off background), DRAW will default to DSPLOGIC setting 0 to perform the command. This is done only to perform the command and does not permanently change the system setting.

DSPFONT

The 778 can only display upper case characters (capital letters) with font 1 selected. If font 2 or font 3 are selected, the 778 can display both upper (capital letters) and lower (small letters) case characters.

EVENT

If no events are on, EVENT 'number' OFF will produce an error.

SUMMARY OF CAUTIONS,Cont.

FIELD

For string\$ file variables (referenced in the FIELD statement) you must use LSET everytime referencing these variables.

FUNCTION

When using the Function command, the code for a Function must be at the end of the mainline code (last instructions in program). Do not use the same name for a FUNCTION, as a 'label', or a SUB routine, or a variable. Variables created in a function are local to that function. Only variables defined with the COMMON command can be seen in the function and shared throughout the program.

GOTO

Do not use 'GOTO' to exit DO and FOR loops or to branch out of an 'IF...THEN...ENDIF'. If you must prematurely end a loop, use EXIT DO... for a DO loop and use EXIT FOR to end a FOR...NEXT.

INP\$

The *, A, P and U 'code\$' all require the optional 101 key keyboard for alphabetic character input. If a value is not entered at the "prompt" and the ENTER key is depressed, the current ('default') value will be used.

INPUT

During an INPUT statement, the 14 soft keys on the front of the 778 are assigned to their standard numeric values and any pre-defined KEY events are suspended. The INPUT statement will not change the display except for the display of the prompt text and the cursor. To display the numeric key labels, you will need to include a series of KEY statements, (without subroutines), to show the key labels before the INPUT. After the INPUT the KEY 0 statement can be used to remove these labels. Additional KEY statements may then be used to return the key labels for any KEY events that were previously defined.

INSTR

In Cardinal BASIC, INSTR cannot be used for assignments.

INTERVAL

In the simulator, INTERVAL is set by selecting Options, Configure Weight Simulator... from the menubar. In the Indicator, INTERVAL is set in SETUP and CALIBRATION.

SUMMARY OF CAUTIONS,Cont.

KEY

Setting a key to an empty string " ", " " disables that key. To disable all previously defined keys use the command KEY 0 (zero).

Since a KEY sub-routine may be called anywhere in a program, it is recommended to save the DSPLOGIC and DSP() coordinates if the KEY sub-routine will be writing or drawing to the display.

Key sub-routines should not be called by the CALL sub-routine command, but if you find it necessary to call a key sub-routine, disable the key (KEY x, " ", " ") before the CALL and re-enable the key after the CALL command.

KILL

You must specify the full "filename" (include an extension if any) when using the KILL command. If a KILL command is given to a file that is currently opened, a "File already open" error will occur. Also, an error will occur if you execute the KILL command to a file that doesn't exist.

Labels

At the specific location in a program that a 'label' is identifying, the 'label' string must be followed by a colon (:) and on a line by itself (no other characters including spaces). Do not use the same name for a 'label', as a FUNCTION, or a SUB routine, or a variable.

LEFT\$

In Cardinal BASIC, LEFT\$ cannot be used for assignments.

LET

Cardinal BASIC supports implied LET statements (e.g., "X = 4.5678", but does not support assignment to multiple variables (e.g., "x, y, z = 3.141596").

LOCATE

The system variables DSPX and DSPY could alternately be used instead of the LOCATE command to display X,Y coordinates.

OPEN

Opening a file as "R", RANDOM access, allows both read (GET) and writes (PUT) of data, without the need to close the file and re-open it. If a file is opened as "O", "OUTPUT, you can only PUT (write) data to it. The same applies to a file opened as "I", INPUT, you can only GET (read) data from it.

SUMMARY OF CAUTIONS,Cont.

OPEN...FOR...AS

The default mode for COM ports on the 778 is text mode, which on input, translates a <CRLF> (carriage return-line feed) into a <LF> (line feed) only. If the input data ends with a <CR> only, the input won't be terminated until another character is received after the <CR>. To avoid this, open the COM port in binary mode [,b] which doesn't do any translation.

PLOT

The PLOT command uses the system variable DSPLOGIC settings 0 = SET (on), 1 = XOR (invert) and 2 = RESET (off) to work. If the system DSPLOGIC is currently set to 3 = OVERWRITE (turn on foreground and turn off background), PLOT will default to DSPLOGIC setting 0 to perform the command. This is done only to perform the command and does not permanently change the system setting.

PRINT

Expressions separated by blanks or tabs are not supported.

The PRINT command is meant for text output to printers or the 778 display and requires a <CR-LF> be sent every 128 characters. If your data does not contain a <CR-LF> every 128 characters, Cardinal BASIC will send one when 128 characters has been reached. This makes the PRINT command unusable for devices where continuous data output is required (a scoreboard for example). Use the PUT command if continuous data output is required.

The PRINT command should not be used to send control characters to a device. The control character will be interpreted as an ASCII character and spaces equivalent to its ASCII value will be sent to the device instead of the control character. Use the PUT command to send control characters to a device.

PUT

When writing a new record to a RANDOM file it is important to use the next sequential record number beyond the current last record number. With Cardinal Basic it is possible to write to any record number. It will work if the new record number is way beyond the current number of records, (your data will be at the new record number), but all records in between the last record and your new record will be filled in with random bits of data from memory.

REM

Cardinal BASIC does not support the use of the single quotation mark (') to denote remarks or placing REM statements at the end of a line of code. REM statements should be placed on a line by themselves before the line of code.

SUMMARY OF CAUTIONS,Cont.

RIGHT\$

In Cardinal BASIC, RIGHT\$ cannot be used for assignments.

SELECT CASE

CASE statements may be followed by string values, but only simple comparisons (equals, CASE IS = or not equal, CASE IS <>) can be performed.

SUB

Variables created in a subroutine are local to that subroutine. Only variables defined with the COMMON command can be seen in the subroutine and shared throughout the program.

TIME\$

The time is displayed and entered in a 24 hour format with 12 added to all times after noon, i.e. 9AM would be "09:00", 3 PM would be "15:00" and midnight would be "00:00".

The date and time used in the SmartWeigh simulator is from the system clock in the PC. Therefore, if you change the date and time in a simulator BASIC program, the change will occur for the PC clock also and may affect other operations on the PC.

WRITE

When writing a new record to a RANDOM file it is important to use the next sequential record number beyond the current last record number. With Cardinal Basic it is possible to write to any record number. It will work if the new record number is way beyond the current number of records, (your data will be at the new record number), but all records in between the last record and your new record will be filled in with random bits of data from memory.

New Commands for 778 REV 00.9

Command: COM2CTL ctl

This command controls operation of the COM2 serial port. 'Ctl' is a number whose value determines whether or not RS-485 output is enabled and selects either the RS-232 or RS-485 pins for input.

Select the value of 'ctl' as follows:

VALUE	FUNCTION
1	disable RS-485 output
2	enable RS-485 output
4	input from RS-232 pins (RS-485 pins will be ignored)
8	input from RS-485 pins (RS-232 pins will be ignored)

Note that the input and output functions can be set simultaneously by adding the appropriate values together. For example, if 'ctl' is 10, then RS-485 output and input will both be selected ($10 = 8 + 2$).

Example: COM2CTL 1

Output: This example will enable the RS-485 output on COM2.
The input source will be unchanged.

Function: COM2CTL

This function will return the current setting of COM2 serial port controls. The value returned will be as follows:

- 0 - RS-485 output disabled, RS-232 input enabled
- 1 - RS-485 output enabled, RS-232 input enabled
- 2 - RS-485 output disabled, RS-485 input enabled
- 3 - RS-485 output enabled, RS-485 input enabled

Example: COM2CTL 10
PRINT COM2CTL

Output: This example will print 3 on the display.

Command: COUNTER bdnm,counter,command[,value]

This command is used to control the counters on the dual counter input card. 'Bdnm' specifies the board (1 through the number of boards installed). 'Counter' specifies the counter ("A" for counter A, "B" for counter B, and "AB" for both counters A and B). 'Command' is one of the following:

- 0 to reset the counter
- 1 to set prescale value for the counter
- 2 to enable the counter
- 3 to disable the counter
- 4 to preset the counter
- 5 to count up
- 6 to count down

If 'command' is 1, then 'value' will be the prescale value for the counter (0 to 255). If 'command' is 4, then 'value' will be the preset value for the counter (0 to 65535). None of the other commands have a value.

Example: COUNTER 1,"A",4,1000
 COUNTER 1,"A",6
 COUNTER 1,"A",2

Output: This example will preset counter A of board 1 to 1000, select down counting, and enable the counter.

Function: COUNTER(bdnm,function)

This function will return information about selected counter. 'Bdnm' specifies the board (1 through the number of counter boards installed). 'Function' specifies the information to be returned as follows:

FUNCTION	RETURN VALUE
"A"	counter A
"B"	counter B
"AB"	counter card inputs
"STATUS"	counter card status

The inputs are mapped to individual "bits" of the return value, which have numeric values as follows:

BIT	NUMERIC VALUE	INPUT
0	1	Counter A connector, pin 4
1	2	Counter A connector, pin 5
2	4	Counter A connector, pin 6
3	8	Counter A connector, pin 7
4	16	Counter B connector, pin 4
5	32	Counter B connector, pin 5
6	64	Counter B connector, pin 6
7	128	Counter B connector, pin 7

The return value will be the sum of the corresponding numeric values for all inputs which have a '1' value (i.e., contact open). To get the value for an individual input, AND the return value with the numeric value for that input. For example,

```
IF COUNTER(1,"AB") AND 16 THEN
  PRINT "COUNTER B PIN 4 IS 1"
ELSE
  PRINT "COUNTER B PIN 4 IS 0"
END IF
```

will display "COUNTER B PIN 4 IS 0" if pin 4 of counter B connector is at '0' (i.e., connected to ground).

The card status will be the sum of one or more of the following values:

VALUE	STATUS
1	counter A enabled
2	counter B enabled
4	counter A is counting down
8	counter B is counting down

when the corresponding status is true. For example, if A is enabled, B is disabled, A is counting down, and B is counting up, then status will be 5 (1 + 0 + 4 + 0).

Addition to operator's manual for analog output card setup:

To calibrate the analog output card, power the indicator on and press the SETUP button on the main menu screen. Setup the scale interface board, if necessary, or press the EXIT button to bypass scale setup. The display will show:

```
CAL DAC?  
  
CURRENT=NO  
  
?
```

A "0/NO" response will return to main menu. A "5/YES" response will continue with analog output calibration. If there is more than one analog output card, the display will show:

```
DAC ID?  
(1 TO N)  
CURRENT=1  
  
?
```

where N is the number of analog output cards. Enter the number of the card to be calibrated.

If there is more than one scale the display will show:

```
SCALE?  
(1 TO N)  
CURRENT=M  
  
?
```

where N is the number of scales and M is the scale the analog card was last assigned to. Enter the number of the scale whose weight will be tracked by the analog output card.

The display will show:

```
BEHIND 0?  
  
CURRENT=XXXXXX  
  
?
```

where XXXXX is the last entered 'BEHIND ZERO' weight. Enter the number representing the weight, below zero, which will produce the lowest analog output (0 volts or 4 mA).

The display will show:

```
VOLTS?  
(YES=VOLT,NO=MA)  
CURRENT=YES
```

?

Enter "5/YES" if the next item is to be entered in volts. Enter "0/NO" if it is to be entered in milliamperes.

If volts were selected, the display will show:

```
DAC OUT?  
(0.001 TO 10V)  
CURRENT=XXXXXX
```

?

otherwise, the display will show:

```
DAC OUT?  
(4.002 TO 20MA)  
CURRENT=XXXXXX
```

?

XXXXXX is the last entered DAC output (voltage or current). Enter a number representing the maximum voltage (or current) to be output by the DAC at scale capacity, not to exceed either 10V or 20 mA.

The display will show:

```
DAC  N  
0.000 V  
4.000 MA
```

where N is the number of the analog output card being calibrated. The buttons have the following functions:

ESC - return to previous display screen
 ZERO - set DAC output to 0V (4 mA)
 MAX - set DAC output to maximum (as entered on DAC OUT screen)
 EXIT - returns to main menu or DAC ID screen, if there is more
 than one analog output card
 V-/V+ - trim DAC voltage
 SPAN-/SPAN+ - trim DAC current span
 OFFSET-/OFFSET+ - trim DAC current offset

Adjust the DAC output as required by pressing the appropriate buttons. When finished, press EXIT to return to main menu or to select another analog output card, if there is more than one.

During normal indicator operation, the analog output will follow the NET weight of the scale to which it is assigned. Analog output is unpredictable when changing from the primary weight units to a secondary weight unit. Therefore, it is not recommended.

Addition to operator's manual for 101 keyboard option operation:

When running a BASIC application program, the 101 keyboard's function keys (F1-F12) are mapped to the indicator's soft keys. That is, pressing the function key will have the same effect as pressing the corresponding soft key. The mapping is shown in the following figure:

F1 < >	< > F9
F2 < >	< > F10
F3 < >	< > F11
F4 < >	< > F12
Shift-F4 < >	< > Shift-F12
< > < > < > < >	
F5 F6 F7 F8	

778 keypad overlay showing 101 function key to soft key mapping.

