Arduino BASIC for the Smart Response XE

A User Manual

Index

Chapter 1: What is BASIC? 6

- *History of BASIC*
- *Why learn BASIC?*

Chapter 2: Arduino BASIC 8

- *What is an Arduino?.....9*

Chapter 3: SMART Response XE 11

- *Using The Smart Response XE.....12*
- *Device Keypad Information*

Chapter 4: Using Arduino BASIC 14

- *Device Commands*
  - ***BYE***
  - ***CLS***
  - ***PRINT***
  - ***RUN*****.....15**
  - ***MSAVE***
  - ***MLOAD***
  - ***LIST***
  - ***BATT*****.....16**
  - ***NEW*****.....17**

- *BASIC Commands*
  - ***GOTO***.....18
  - ***LET***.....19
  - ***GOSUB***.....20
  - ***RETURN***.....21
  - ***REM***
  - ***INPUT***.....22
  - ***IF***.....23
  - ***FOR***.....24
  - ***NEXT***.....25
  - ***DIM***
  - ***PAUSE***.....26
  - ***POSITION***
  - ***RND***.....27
  - ***LEN***
  - ***VAL***

Chapter 5: Data Types 28

- *Integers*.....29
- *Floats*.....31
- *Strings*.....32
- *Variables*.....34
- *Arrays*.....37

Chapter 6: Operators 43

- *Arithmetic Operators*
  - Operator: **+**.....44

- Operator: **-**
- Operator: ***.....45**
- Operator: **/**
- Operator: **MOD**..... .....*46*

- *Comparison Operators*
  - Operator: **=**
  - Operator: **<>**.....*47*
  - Operator: **>**
  - Operator: **>=**.....*48*
  - Operator: **<**
  - Operator: **<=**
- *Logical Operators*.....*49*
  - Operator: **AND**
  - Operator: **OR**

Chapter 7: Example Programs 50

1. The BASIC's..... .....*51*
2. Count to 100.....*54*
3. Number Guessing Game.....*55*
4. Text Animation.....*56*
5. Move A Character On Screen.....*57*
6. Calculator.....*59*
7. Roll The Dice.....*60*
8. Hangman.....*61*
9. *Simple Text Based Game*.....*64*
10. Search An Array.....*70*

Chapter 8: Your Programs 73

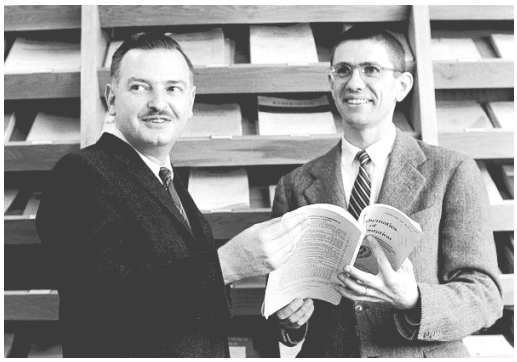
- Program 0: _____.....73
- Program 1: _____.....78
- Program 2: _____.....82
- Program 3: _____.....87
- Program 4: _____.....91
- Program 5: _____.....96
- Program 6: _____.....100
- Program 7: _____.....105
- Program 8: _____.....109
- Program 9: _____.....114

Chapter 1: What is BASIC?

Yeah what is it BASICALLY?

History of BASIC

BASIC (Beginners' All-purpose Symbolic Instruction Code) is a family of general-purpose, high-level programming languages designed for ease of use.



John G. Kemeny and Thomas E. Kurtz

The original version was created by John G. Kemeny and Thomas E. Kurtz at Dartmouth College in 1963. They wanted to enable students in non-scientific fields to use computers. At the time, nearly all computers required writing custom software, which only scientists and mathematicians tended to learn.

Why learn BASIC?

Because of its simplicity, BASIC has frequently been used in teaching the

introductory concepts of programming with a working language.

Learning to program in BASIC is a good stepping stone to fundamental programming concepts such as storing, accessing, and changing variables, and working with different data types. Although BASIC is not used much in the real world of programming in 2023, the concepts you can learn while coding in BASIC can and will transfer to other programming languages you learn later.

Chapter 2: What is Arduino BASIC?

And what in the blazes is the Arduino?!

Arduino BASIC is a version of BASIC that is designed to run on an Arduino.

Most types of old school BASIC including Arduino BASIC use a line numbering format for programs. Each program is made up of individual numbered lines. Each line MUST start with a number, then a space before the line code.

The CPU uses the line numbers for knowing where it's at in the program, and it also allows the programmer to guide the program along its way.

It's good to write your line numbers spaced out, e.g.: 10,20,30… Or: 20,40,60. This is mainly because if you want to add something between lines, and you wrote your program with lines 1,2,3,4,5… There would be no place to put a new line.

Below is an example of line number usage in an Arduino BASIC program.

```
10 int = 1
20 string$ = “Hi!”
30 PRINT string$
40 IF int > 0 THEN PRINT “Yes”
50 int = int + 1
60 PAUSE 1000
70 GOTO 30
```

Pretty cool Right? This book will cover what all that stuff does in more detail.

What is an Arduino anyway?

An Arduino is a type of microcontroller board with an ATMEL chip installed.

An Arduino UNO Developement Board

Due to the low power consumption of these microcontrollers, and affordable prices, they are used for many things. From simple sensor devices to robots!

Because the ATmega328p is one of the most widely used ATMEL chips, the specs are included below.

An Atmega328p IC Chip

The Smart Response XE device has a variant of the "Arduino" chip called ATmega128RFA1, which is much more capable in terms of memory size than the ATmega328p.

	ATmega128RFA1	ATmega328p
Max Processor Speed	16 Mhz	16 Mhz
SRAM	16 Kb	2 Kb
Flash Storage Memory	128 Kb	32 Kb
EEPROM Storage Memory	4 Kb	1024 Bytes

Now that we have a basic understanding of the Arduino, and what Arduino BASIC is, we can move onto the specifics of the Smart Response XE device in the next Chapter.

Chapter 3: About the SMART Response XE

Is it really smart?

The Smart Response XE was originally created for use in classrooms, for test taking, etc. Which means there were a lot of these made, they were made to be very durable. They feature a full keyboard, a monochrome LCD, and can run on 4 AAA batteries for days.

A Computer Nerd

It wasn't long before some computer nerds discovered that the device contained an Arduino compatible chip. Challenge accepted! The nerds started talking and soon they hacked the device to run their own software.

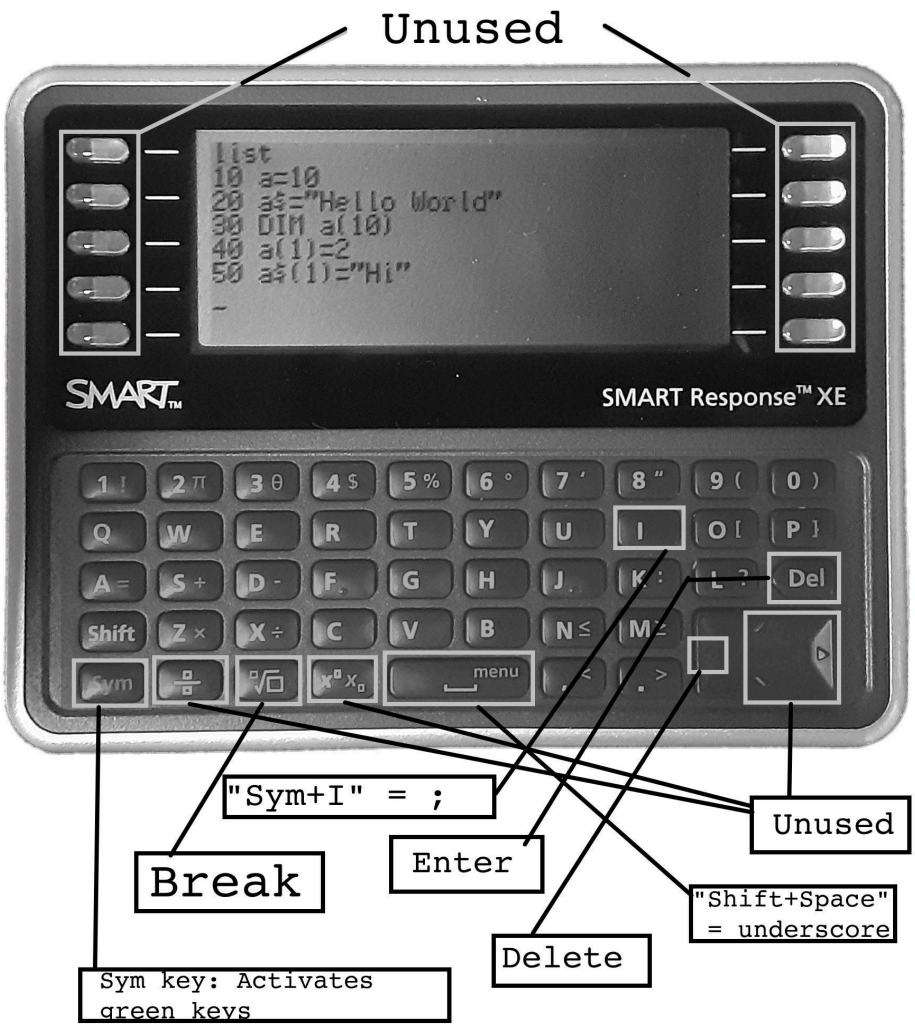
Because they are no longer used in classrooms as of 2023, they are easy to find on Ebay, at a good price.

Using The Smart Response XE with Arduino Basic

In this section we will go over the various buttons on the device, as some of them have been modified for easier use with Arduino BASIC.

Device Keypad Information

Something to remember is that there is no “Enter” key. The “Del” key is used



as the Enter key. Use the “Left arrow” key on the keypad for the “Delete” key.

The “Sym” key is the shift key to type in symbols which are green on the keypad. As there is no “;” key on this keyboard, pressing “Sym+I” will print the “;” key.

Like on a regular keyboard, pressing and holding the “Shift” key before typing a letter will make it print upper case.

At this point the keys to the left and right of the screen are not used for anything.

The square root key is used as a program “break” key. Meaning, if a program is running, and you press this key, it will end the program.

At the top of the device is a power button, pressing and holding this button will power on the device after it’s been powered off.

Chapter 4: Using Arduino BASIC

Learn all the 133t skillz!

Now let's go over the essential Arduino BASIC commands for the Smart Response XE device.

Device Commands:

BYE: To turn the device on, the power button at the top is pressed. To turn the device off all you have to do is type "**bye**" and hit enter. The device will then power off.

CLS: To clear the screen, type "**cls**" then hit enter. The screen will clear.

PRINT: This command is used to display text to the screen. For example, if you type (**print** "hello!"), the screen will display "hello!"

You can also combine things you want to print on one line like this:

```
print "hello";" ";"there";"!"
```

Output: "hello there!".

RUN: After you have written or loaded a program, typing "**run**" then hitting enter, will run the program. Remember, to end a running program, you can hit the "break" button which will end it.

MSAVE: After you have written a program and want to save it, you can use this command to save the program to any of the 10 memory slots on the flash storage chip. For example typing "**msave** 0" then pressing enter, will save the program to slot 0 on the flash chip. You can use slots 0 to 9 for saving programs.

But beware! If you have a program stored in a slot, and save a new program there, it will overwrite the previously saved program.

MLOAD: Typing "**mload** 0" will load the program from slot 0 if you have one saved there. You can use numbers 0 to 9 for loading programs.

LIST: This command is used to view the program that is loaded in working

memory. If you type “**list**” then hit enter, it will list the entire program! If it is a long program, it will show as much as that fills the screen, then scroll down one page at a time when you press any button, until you reach the end of the program. If you press the spacebar, it will scroll to the end of the program.

If you want to just look at one line of the program you can specify that line number. For example, let's say we want to look at line 60, we would type “**list 60,60**” and it will only output that line from the program. If we want to look at lines 60 to 90, we would type “**list 60,90**” and all the lines from 60 to 90 will be displayed on the screen.

BATT: This command is used to read the battery voltage in millivolts. If you type “**print batt**” it will display the voltage on the screen. For example, if the number output is 5412, it means the battery voltage is at 5.412 volts!

NEW: This command will erase everything from working memory (not what is saved on flash memory, slots 0 to 9). If you have not saved a program you want to keep, it's better not to do this lol. But if you are wanting to start a new program from scratch, and you have already saved the program you have loaded this is the way to go. Typing "**new**" then pressing enter will launch this magic.

BASIC Commands:

Now we will get into more BASIC commands that you will be using to write your programs. Something to keep in mind is that you can use the ":" symbol to add more than one command on a single line. For example, look at the program below.

```
10 print "hello":pause 1000:cls:goto 50
50 print "hello again!":pause 5000:cls
```

If we were to write this program without using the ":" symbols, it would take up more lines like in the example below.

```
10 print "hello"
20 pause 1000
30 cls
40 goto 50
50 print "hello again!"
60 pause 5000
70 cls
```

Either way works, it's just a matter of how you want to write it!

GOTO: is used to "jump" from one line to the line specified, for example "**goto** 60" will make the program jump to line 60 and run from there.

On the next page is an example of a simple BASIC program demonstrating a **GOTO** command being used.

```
10 print “hello!”       (outputs “hello!)
20 cls                  (clears the screen)
30 GOTO 60              (jumps to 60)
40 print “hi!”          (will not print as we
jumped to line 60)
50 print “whats up?”    (this will also not print)
60 print “whaaat?”      (this WILL print)
```

This example is just to show what a **GOTO** command does. This program is not a good one because lines 40 and 50 are just useless code. But later as you learn more BASIC, you will discover that the **GOTO** command is used a lot, and is very useful.

LET: can be used to declare a variable. We will look at variables more in depth in Chapter 5. With Arduino BASIC, using the **LET** command is not necessary to declare a variable, but we will include it in the manual as it is used in many other types of BASIC.

On the next page is an example of creating a variable named “a” with a value of 10 using the **LET** command, as

well as creating a variable named "b" without using it.

```
10 LET a = 10
20 b = 20
30 print a        (prints 10)
40 print b        (prints 20)
```

GOSUB: is very similar to GOTO, but it is quite useful as you can have the program jump to a specified line, then return back to where you called the **GOSUB** command. Below is an example of this being used.

```
10 a=10        (make variable a, value of 10)
20 print a     (prints 10)
30 GOSUB 80    (jump to line 80)
40 print a     (a is now 15, so it prints 15)
50 a = 20      (we change a to 20)
60 print a     (prints 20)
70 stop        (ends the program)
80 a=15        (jumped from 40, a is now 15)
90 RETURN      (used to return to line 30)
```

GOSUB is a useful command to create blocks of code that can be used again and again.

RETURN: As described above, this command is used with GOSUB to return from the line you jumped to with GOSUB.

REM: is used to make a comment in your program code. When the processor reads the program, if the line begins with REM, it skips over it. Example below.

```
10 print “hello!”
20 REM this will do nothing
30 print “howdy”
```

This program will print “hello!” then “howdy”. The line beginning with **REM** is skipped by the CPU when executing the code. If you need to make a note in your program code, this command is useful.

INPUT: is used to allow the program's user to enter text or numbers. Below we will see a simple example of this being used.

```
10 INPUT a (assigns user input to a)
20 cls     (clear the screen)
30 print a (if the user types 10, then
presses enter the screen will display 10,
and so forth)
```

Where we may end up with an issue here, is if the user types “hello” or anything that is not a number, and presses enter. The variable “a” can only be a number, so we will end up with an error.

If we want to allow text to be input such as “qwertyuiop” then we would need to specify the **INPUT** as being a STRING variable. On the next page is an example of storing user input in a string variable. You can read more about data types and variables in Chapter 5.

```
10 INPUT a$      (the INPUT assigns whatever the
user types to a)
20 cls           (clear the screen)
30 print a$      (if the user types 10, then
presses enter the screen will display 10, and so
forth)
```

Now anything can be typed into this input without any errors.

IF: is used to compare data. Below is an example of this being used.

```
10 print "What animal meows?"
20 input a$
30 IF a$ = "cat" THEN goto 100
40 cls
50 goto 10
100 print "You are correct!"
```

If the user types in "cat", then the program will jump to line 100". Otherwise, it will jump back to the beginning.

We can also combine multiple conditions like in the example below. The

symbols =, <>, <, >, >=, <= can all be used in these **IF** comparisons.

```
8 print “Enter a number:”
10 input a
20 IF a>0 THEN print “Number is a Positive Number”
30 IF a<0 THEN print “Number is a Negative Number”
40 IF a=0 THEN print “Number is Zero”
50 IF a<>10 THEN print “Number is not 10”
60 IF a>=10 THEN print “Number is either 10 or greater”
70 IF a<=10 THEN print “Number is either 10 or lesser”
80 IF a>=10 and a<=20 THEN print “Number is between 10 and 20”
90 IF a=10 OR a=20 THEN print “The number is 10 or 20”
100 print “Enter another number?”
120 print “Type: yes or no”
130 input a$
140 cls
150 IF a$=”yes” THEN goto 8
160 print “Goodbye!”
```

FOR: is used to iterate through a range of numbers, or through an array of data. We will cover arrays in Chapter 5, with some more examples of using the **FOR** command. Below is an example of iterating through the numbers 1-10.

```
10 FOR i=1 TO 10 (sets the range of numbers)
20 print i       (prints current value of i)
30 NEXT i        (increments i by 1, until
we reach 10)
```

NEXT: is the command used in a FOR loop to go to the next increment of the range. As shown above.

DIM: is the command used to create an array. An array is a method of storing data like we have learned about variables, but unlike a single variable, we can specify multiple values we want to store in it. In other programming languages, an array is created like this:

```
a=[1,3,5,9,3,1]
```

This array "a" would be created with 6 values, and normally selecting the first value would be done by saying a[0].
A multidimensional array would look like this:

```
a=[[1,2,3],
   [4,5,6],
   [7,8,9]]
```

Selecting the array location where the 6 is stored would be done like this a[1][2]. I am showing this form of arrays so it is easier to visualize what an array is structured like.

In Arduino BASIC we can also create multidimensional arrays. We will discuss that more in Chapter 5. Below is an example of creating an array and assigning some data to it.

```
10 DIM a(3)          (create array "a" with a size for 3 values)
20 a(1)=10           (in Arduino basic, the first array index is 1)
30 a(2)=20:a(3)=30   (now we have given values to them all)
40 print a(1)        (outputs 10)
```

PAUSE: This command is very simple. Using "**PAUSE 1000**", will pause the running program for 1000 milliseconds (or 1 second).

POSITION: This command can be used to print at a specific location of the screen. If the author of this book got around to it there should be an example of using the **POSITION** command to create a text animation in Chapter 7. The LCD

display of the SMART Response XE has 32 columns and 7 rows of text. Below is an example of printing the letter "A" on the 25th column, and 3rd row of the screen.

```
10 POSITION 25,3
20 print "A"
```

RND: This command is used to generate a random number between 0 and 1. Returns a decimal. Using **INT(RND*10)** will generate a number between 0 and 9.

LEN: This command returns the length of a string. Below is an example.

```
10 a$="hi"
20 print LEN(a$)          (returns 2)
30 print LEN("howdy")     (returns 5)
```

VAL: This command returns the value of a string. Example below.

```
10 print VAL("4+2+2+2")     (returns 10)
```

Chapter 5: Data Types

Where do you put your data?

There are several different types of data that we will work with in Arduino Basic. Data types are a fundamental concept to any computer language.

Each of these data types are stored differently in the chip's memory, which is why they need to be defined. The only exception is the float data type, which Arduino BASIC handles automatically.

Data Types:

- Integer
- Float
- String

Method of Storing Data:

- Variables
- Arrays

Integers

Definition: *Integers include positive numbers, negative numbers, and zero. 'Integer' is a Latin word which means 'whole' or 'intact'. This means integers do not include fractions or decimals.*

In Arduino BASIC, we can store integers in variables like this:

a = 1

Or we can store them in an array like this:

dim a(5) (create an array with a space for 5 integer variables)

a(1) = 1
a(2) = 10
a(3) = 100
a(4) = 1000
PRINT a(3) (outputs 100)

A string variable that contains a single integer can be converted to an integer datatype like this:

a$ = "1"

a$ + 1 (This will return an error, as a$ is a string, and cannot be added to the integer 1)

b = INT(a$)
print b (Outputs 1)
print b + 1 (Outputs 2)

A float is a decimal number, eg. 4.2, 0.76, 200.1, and so on.

Using the **INT()** function, we can convert a float to a whole number. In Arduino BASIC, the **INT()** function will always round down. So if you want it to round up you must add 1 to whatever you are converting. An example:

```
10 a = 3.14
20 a = INT(a) (convert float(a) to INT)
30 print a    (outputs 3)
```

Let's take another look at the **RND** function we briefly discussed in the previous chapter.

If we type "**PRINT RND**", we will get a float between 0.0000001 and 0.9999999. That can be useful in many applications, but if we want to get a number between 1 and 10 every time, we can multiply the result of the RND function by 10 to move

the decimal point over once, then convert it to an integer, which rounds down, giving us a possible number from 0-9, so we then add 1.

```
10 num = INT(RND*10)+1
20 print num
```

Below is a program that will keep printing out random numbers between 1 and 100 until the program is terminated.

```
10 num=INT(RND*100)+1 (generate number)
20 print num           (print the number)
30 pause 200          (add a delay)
40 goto 10            (jump to beginning)
```

Floats

We won't discuss floats any more here, as we pretty much covered them when we were talking about integers. They don't need to be stored in any special way like in other programming languages, as Arduino BASIC automatically handles them.

Strings

In Arduino BASIC, a string is one or more characters that are stored in a manner where they can be combined with many different characters. There are some simple rules for strings.

A string variable name must end with a "$" sign, and the data stored in that variable must have quotation marks at the beginning and end. Below are some examples of some strings.

```
10 x$ = "abc123!@#,&QWERTY"
20 y$ = "The quick brown fox jumped over the lazy dog"
30 z$ = "1"
```

On line 30 above we are storing a single number in a string, even though the number itself is an integer in theory, it is still stored in a string data type, so must be treated as a string. On line 10 above, we can see that

all sorts of characters can be stored in a string. In Arduino basic, the only character that cannot be stored in a string is the quotation mark itself: ", so if you want to store a quote in a string, you would have to put it in single quotes, example:

```
10 a$ = "He said, 'Wow cool!'"
```

Strings can be joined together:

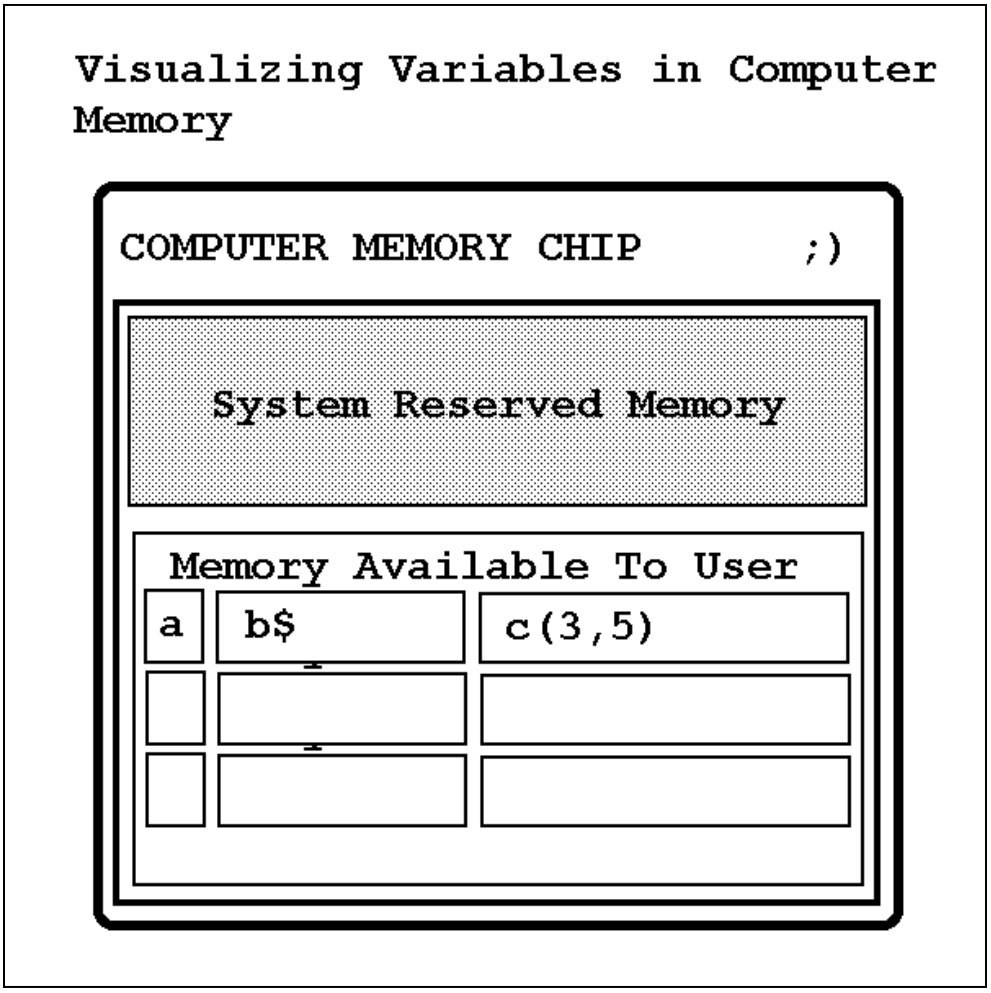
```
10 a$ = "Hello there, "
20 b$ = "my name is: "
30 c$ = "Bob."
40 d$ = a$+b$+c$
50 print d$
(Output:Hello there, my name is: Bob.)
```

Variables

We have already been talking about and using variables a lot in this book. In case there is any confusion about

what a variable is, let's define it right now.

A variable is a name that you can put on a spot in the computer's memory, allowing you to store and access data from that spot in memory. There are



maximum sizes to how much data you can store in one variable. The author of this book was too lazy to find out exactly what the maximum size is, so

maybe you can experiment and find out for yourself! On the previous page is an illustration showing the general idea of what a variable is doing.

When we name a variable, it's like we are putting a label on a box, we can put things in that box, and take things out, and because we labeled it, we know which box to look in.

In Arduino BASIC, variable names can be up to 8 characters long, and must start with a letter.

It does not matter if the variable name has uppercase or lowercase letters in it. Our variable name can be "VAR" or "Var" or "var" and it would still point to the same place in memory.

If you try to name a variable "123var", it will not work, as it must start with a letter. But you can name a variable "var123" and it will work.

Below we can see how variables are named and data is stored in them. "name" can be substituted for the

variable names of your choice and need for that particular variable.

INTEGER	name = 123
STRING	name$ = “this is a string”
INT DIM	DIM name(2,3) a(1,1) = 1 a(1,2) = 2 a(1,3) = 3
STR DIM	DIM name$(2,3) a$(1,1) = “this is” a$(1,2) = “a” a$(1,3) = “string”

At any time you can change what is in a variable:

```
10 a=1
20 print a      (outputs 1)
30 a=2
40 print a      (outputs 2)
```

String variables take up more space in memory than integer variables, and

array variables take up more memory than strings and integers.

Arrays

In Chapter 4 we discussed arrays in some detail. They are a good way to store groups of data. An array can either be a group of string variables, or a group of integer variables:

```
10 DIM a(10)   (create an integer array 10 long)
20 DIM a$(10)  (create a string array 10 long)
```

Array "a" above can hold 10 integers, and array "a$" can hold 10 strings. By default, when the array is created, they are empty. If we try to access one of the elements in the array before saving any data to it, this is what happens:

```
10 DIM a(10)   (create an integer array
10 long)
20 DIM a$(10)  (create a string array 10
long)
30 PRINT a(1)  (try to access 1st element
of array "a", it outputs "0")
40 PRINT a$(1) (try to access 1st
element of array "a$", it outputs
```

```
nothing)
```

We need to add some data to our arrays, continuing from the above code, let’s do this:

```
50 a(1) = 2023
60 a$(1) = “It’s “
70 a$(2) = “!”
```

Now we have added data to both of our arrays. Let’s try to use it.

```
80 PRINT a$(1)+a(1)+a$(2)
```

This will result in an error, because we are trying to join an integer a(1) with string elements from a$. We need to change the integer from the array “a” to a string type:

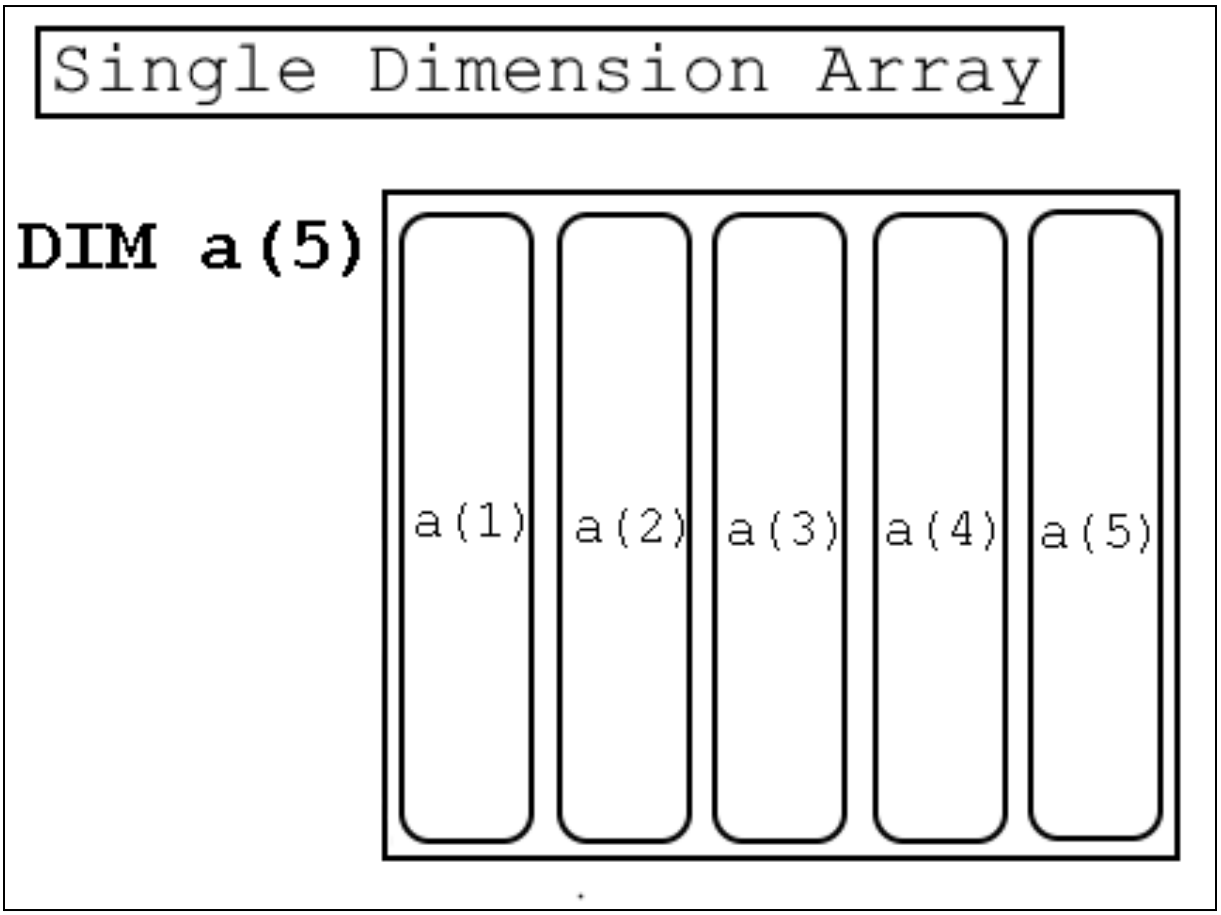
```
80 PRINT a$(1)+STR$(a(1))+a$(2)
```

This outputs “It’s 2023!”

The keyword **DIM** is used to define them, relating to “Dimensional”.

Let’s examine these dimensions in more detail. We will think of the dimensions as boxes. In the above example, we created an array “DIM a(10)”, so we will look at that as a box, with slots for 10 items in it. But what if we

decided to put 10 boxes inside that big box? Below is an example of a single dimensional array with 5 “slots” to store data, in this case integers.



The elements inside the array are shown as rounded rectangles, the larger rectangle is the array itself, the container we are storing these elements in.

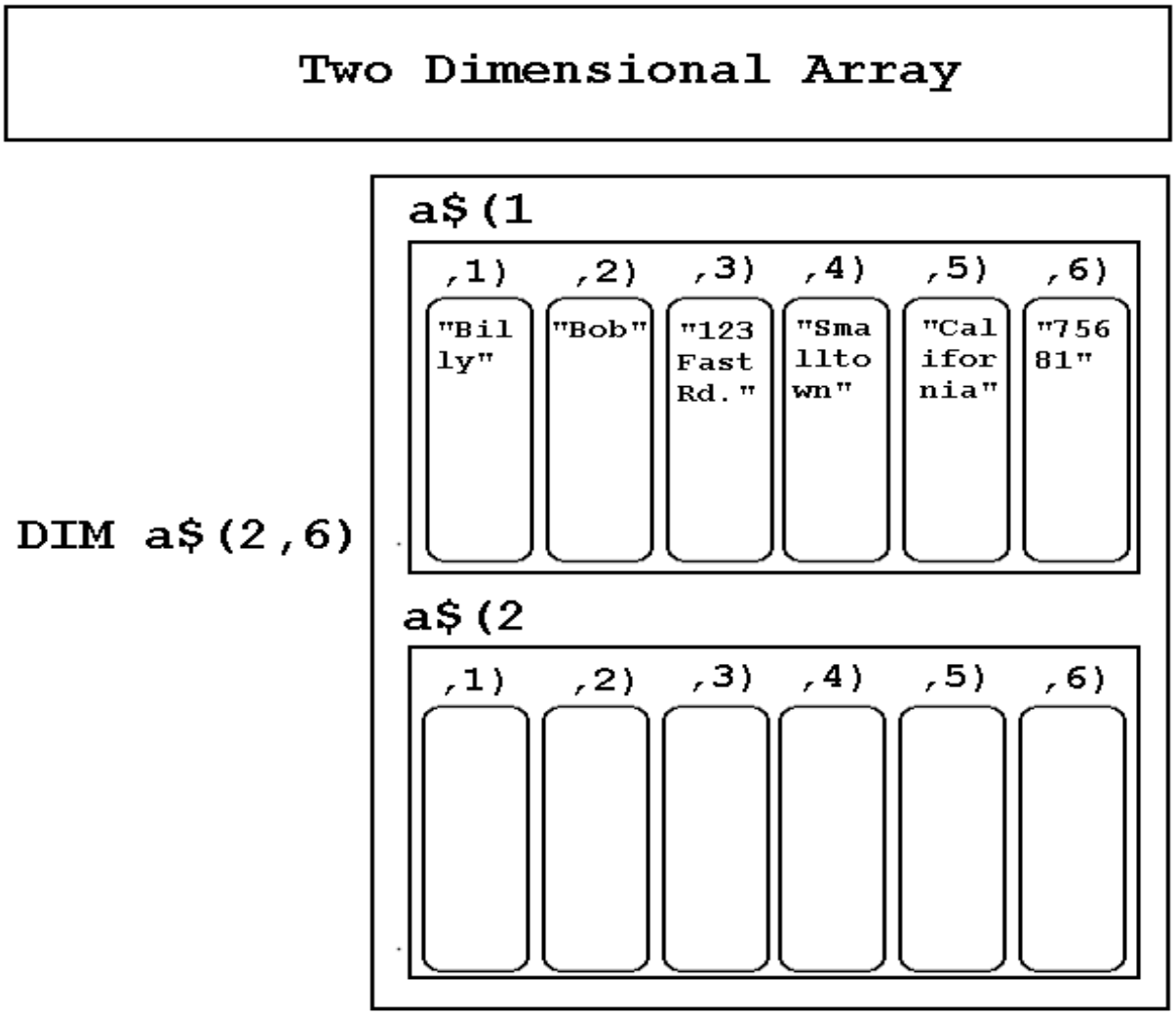
Let’s make an array with more dimensions! In this example we will make a string type array, storing useful

information, in this case an address book.

```
10 DIM a$(2,6)
20 a$(1,1)=“Billy”
30 a$(1,2)=”Bob”
40 a$(1,3)=”1234 Fast Rd.”
50 a$(1,4)=”Smalltown”
60 a$(1,5)=”California”
70 a$(1,6)=”75681”
```

Here we made an array containing two separate arrays. We populated one of those arrays with Billy Bob’s name and address. Each of those arrays can contain 6 string variables. The second array we created is empty as we did not add any data to it.

Arrays can have even more than two dimensions, for example “Dim a(5,5,5)” Will create a box (array), with 5 boxes inside it, and each of those boxes will have 5 boxes, and each of those boxes will have 5 slots for storing data. Usually 1 or 2 dimensions is all you will need when using an array.



Arrays really come in handy when you want to store a set of data that needs to be interacted with more than a regular variable, and needs to have relation to each other. In our address storing example, we could make an array that could store many addresses, and add more array fields for things such as telephone

numbers and email addresses, etc. A good reason to use arrays is we can search them using the input function.

The FOR and NEXT functions discussed earlier in this book are the main tools that we can use to loop through an array.

In the **Examples** section near the end of this book, there will be an example program for searching for data in an array.

CHAPTER 6: Operators

They said it was an operator error…

Operators are used for combining or comparing data. Three main kinds of operators are used in Arduino BASIC as well as many other programming languages.

If you think of programming as merely a means to sort, compare, and modify data, you will discover that operators are very important to use even in the most simple of programs.

Kinds of Operators:

- Arithmetic
- Comparisons
- Logical

Arithmetic Operators

Operator: +

This is the only operator that can be used on a string, eg. to join two strings together:

```
10 string$ = “String1” + “String2”
```

This operator is also used to add numbers together:

```
10 number = 1+2
20 print number (Outputs 3)
30 number = 1.75 + 2
40 print number (Outputs 3.75)
```

Operator: -

Used to subtract one number from another.

```
10 number = 17 - 7
20 print number      (Outputs 10)
```

Operator: *

Used to multiply numbers.

```
10 number = 2 * 5
20 print number    (Outputs 10)
```

Operator: /

Used to divide numbers. If the number does not divide to a whole number, the number will be converted to a FLOAT.

```
10 number = 10 / 2
20 print number     (Outputs 5)
30 number = 3 / 2
40 print number     (Output: 1.5)
```

Operator: MOD

Used to find the remainder of a divided number.

```
10 num1 = 10
20 num2 = num1 MOD 2
30 print num2          (Output: 0)
40 num1 = 11
50 num2 = num1 MOD 2
60 print num2          (Output: 1)
```

Comparison Operators

Operator: =

Checks to see if two numbers or strings ARE equal.

```
10 a = 1
20 b = 1
30 IF a=b THEN PRINT "Yes" (Output: Yes)
40 a$ = "hi"
50 b$ = "hi"
60 IF a$=b$ THEN PRINT "Yes"
Output: Yes
```

Operator: <>

Checks whether two strings or numbers are NOT equal.

```
10 a=1
20 b=2
30 IF a <> b THEN PRINT "Not Equal."
Output: Not Equal
```

Operator: >

Checks whether one number is greater than another number.

```
10 a=3
20 b=2
30 IF a > b THEN PRINT "Greater"
Output: Greater
```

Operator: >=

Checks whether one number is greater than or equal to another number.

```
10 a=3
20 b=2
30 IF a >= b THEN PRINT "Greater or Equal" (Output: Greater or Equal)
```

Operator: <

Checks if one number is less than another number.

```
10 a=1
20 b=2
30 IF a < b THEN PRINT "Less Than"
Output: Less Than
```

Operator: <=

Checks whether one number is less than or equal to another number.

```
10 a = 2
20 b = 2
30 IF a >= b THEN PRINT "Less or Equal"
Output: Less or Equal
```

Logical Operators

These operators are used to check multiple conditions at once.

Operator: AND

Returns true if both conditions are true.

```
10 a = 2
20 b = 2
30 IF a > 1 AND b = 2 THEN PRINT "True"
Output: True
```

The code after the THEN will run IF both of the conditions are true.

Operator: OR

Returns true if one condition is true.

```
10 a = 0
20 b = 2
```

```
30 IF a > 1 OR b = 2 THEN PRINT "True"
Output: True
```

The code after the THEN will run IF one of the conditions is true.

CHAPTER 7: Example Programs

"Do it right or do it twice" - Gideon

This chapter has 10 example programs you can try out and experiment with. In order to test these out you will have to manually type them into your device line by line before saving or running them. Tedious I know, but it's good practice for becoming familiar with the code!

All example programs have comments in them (using the REM feature) explaining what the code is doing. When you put these programs into your device you may skip those lines if you want to save your thumbs from all that extra typing.

Program #1: The BASIC's

This program is just to showcase all the different commands and functions covered in this book. It doesn't really do anything special, but can be referenced to see use cases for various commands and functions.

Program Data:

```
10 REM create an integer variable
20 a = 1
30 REM create a string variable
40 a$ = "Example Program "
50 REM convert int variable a to string
60 b$ = STR$(a)
70 REM join the strings
80 c$ = a$+b$
90 print c$;": The BASIC's"
100 REM add 19 to variable a
110 a = a + 19
120 REM divide a by 2
130 a = a / 2
140 REM subtract 2 from variable a
150 a = a - 2
160 REM multiply variable a by 7
170 a = a * 7
180 PRINT a
```

```
190 REM print the remainder of a / 5
200 PRINT a MOD 5
210 REM check if a is greater than 2
220 IF a > 2 THEN PRINT "Yes"
230 REM skip the next line
240 GOTO 260
250 a = 100
260 PRINT "line 250 skipped"
270 REM if a>2 set a to 2, goto line 220
280 IF a>2 THEN a = 2:GOTO 220
290 REM make a into float
300 a = 1.234567
310 print a
320 REM convert a to whole number (int)
330 b = INT(a)
340 print b
350 REM convert a to int, round up
360 b = INT(a)+1
370 PRINT b
380 REM create a loop, enter 5 to exit
390 if a = 5 THEN GOTO 430
400 REM get user input
402 PRINT "Enter 5 to exit loop"
410 input a
420 goto 390
430 print "a = ";a
440 pause 3000
450 cls
460 REM gosub jump to 1000
```

```
470 GOSUB 1010
480 print "GOSUB function done"
490 GOTO 1100
500 a = 1 : DIM a$(10)
510 a$(1)="The"
520 a$(2)="quick"
530 a$(3)="brown"
540 a$(4)="fox"
550 a$(5)="jumped"
560 a$(6)="over"
570 a$(7)="the"
580 a$(8)="lazy"
590 a$(9)="dog"
1000 REM gosub function count to 10
1001 REM or if a=1 iterate and
1002 REM print array values
1010 FOR i=1 TO 10
1020 IF a=1 THEN GOTO 1040
1030 print i:pause 500:goto 1070
1040 IF i<10 THEN PRINT a$(i):GOTO 1060
1050 if i=10 THEN PRINT "The End."
1060 PAUSE 500:CLS
1070 NEXT i
1072 IF a = 1 THEN GOTO 1100
1080 RETURN
1100 PRINT "End of Program!"
```

Program #2: Count To 100

This is a program to count from 1 to 100 and print it to screen.

Program Data:

```
10 CLS
30 PRINT "Enter 1 for style 1"
50 PRINT "Enter 2 for style 2"
80 INPUT u
100 REM verify user input
150 IF u<>1 AND u<>2 THEN GOTO 10
160 CLS
170 FOR i=1 TO 100
200 PRINT i
210 IF i=100 THEN GOTO 240
230 IF u=2 THEN PAUSE 500:CLS
240 NEXT i
250 PRINT "Done!"
```

Program #3: Number Guessing Game

This program generates a random number between 1 and 100, the user has to guess it, and the program will say whether their guess is too high or low, until they guess the correct number.

Program Data:

```
10 n = INT(RND*100)+1
20 tries = 0
50 CLS
60 PRINT "Tries: ";tries
90 PRINT "Guess a number 1-100"
130 INPUT g
140 tries = tries + 1
170 IF g = n THEN GOTO 800
180 IF g > n THEN GOTO 400
185 IF g < n THEN PRINT "Too Low!"
190 PAUSE 1000:GOTO 50
400 PRINT "Too High!"
430 PAUSE 1000:GOTO 50
800 PRINT "You Won!":PAUSE 1000
810 PRINT "Enter 1 to play again.
820 PRINT "Enter 2 to exit.
830 INPUT a
840 CLS
850 IF a<>1 AND a<>2 THEN GOTO 810
```

```
860 IF a=1 THEN GOTO 10
870 IF a=2 THEN PRINT "Goodbye"
880 PAUSE 2000
890 CLS
```

Program #4: Text Animation

This program should move a text character or number randomly around the screen using the POSITION function. Remember the device screen is 32 columns wide and 7 rows high.

Program Data:

```
10 REM var c is our character
20 c=0
30 REM y and x are chars screen position
40 x=0
60 y=0
80 REM dy and dx are chars x and y dir
100 dy=1
120 dx=1
140 CLS
160 REM set chars position
200 POSITION x,y
220 REM show char on screen
260 PRINT c
```

```
280 REM random number generator
300 dyr = INT(RND*10)
340 dyx = INT(RND*10)
370 IF dyr <= 3 THEN dy = 0
390 IF dxr <= 3 THEN dx = 0
410 IF dyr > 3 AND dyr < 7 THEN dy = -1
420 IF dxr > 3 AND dxr < 7 THEN dx = -1
430 IF dxr >= 7 THEN dx = 1
440 IF dyr >= 7 THEN dy = 1
470 x = x+dx
490 y = y+dy
510 IF x > 31 THEN x = 31
520 IF y > 6 THEN y = 6
530 IF x < 0 THEN x = 0
540 IF y < 0 THEN y = 0
550 PAUSE 500: GOTO 140
```

Program #5: Move A Character On Screen

We will reuse the code from program #4, but instead of randomly moving our character on the screen, we will get the input from the arrow keys using the "INKEY" function that is not discussed anywhere else in this book.

Program Data:

```
10 REM var c is our character
20 c=0
22 REM set key to 0
26 key = 0
30 REM y and x are chars screen position
40 y=1
60 x=1
80 REM dy and dx are chars x and y dir
100 dy=0
120 dx=0
140 CLS
160 REM set chars position
200 POSITION x,y
220 REM show char on screen
260 PRINT c
270 PAUSE 200
280 REM if key is not pressed, get key
290 key=INKEY
300 IF key = 0 THEN GOTO 280
302 IF key = 4 THEN dy = -1
304 IF key = 5 THEN dy = 1
306 IF key = 3 THEN dx = 1
308 IF key = 2 THEN dx = -1
310 key = 0
470 x = x+dx
490 y = y+dy
510 IF x > 30 THEN x = 30
```

```
520 IF y > 6 THEN y = 6
530 IF x < 1 THEN x = 1
540 IF y < 0 THEN y = 0
550 GOTO 140
```

Program #6: Calculator

A simple calculator program using the Arduino BASIC built in arithmetic operators and the VAL function.

Program Data:

```
10 PRINT "Calculator"
20 INPUT a$
30 REM make sure input is not blank
40 IF LEN($a) < 3 THEN CLS:GOTO 10
60 CLS:PRINT "Calculator"
70 PRINT a$;" = ";VAL(a$)
80 GOTO 20
```

Program #7: Roll The Dice

Dice rolling program. Allowing users to select up to 3 6 sided dice.

Program Data:

```
10 PRINT "Dice Game"
12 tmp = 0
30 PRINT "Enter dice: 1-3"
40 INPUT roll
44 PRINT "Rolling ";roll;" dice..."
50 IF roll<>2 AND roll<>3 THEN roll=1
52 DIM dice(roll)
60 FOR i=1 TO roll
62 tmp=0
80 FOR n=1 to 6
82 z=RND
84 if z > 0.4 THEN z = 1
90 tmp = tmp + INT(z)
92 NEXT n
100 dice(i) = tmp
110 PAUSE 200: NEXT i
120 CLS
140 FOR d=1 TO roll
160 PRINT "Dice ";d;": ";dice(d)
170 PAUSE 500
180 NEXT d
200 PRINT "Roll again? y(es), n(o)"
220 INPUT o$
```

```
230 IF o$ = "y" THEN CLS:GOTO 10
260 PRINT "Goodbye":PAUSE 2000:CLS
```

Program #8: Hangman

A rudimentary version of the classic "hangman" game you may remember playing with pencil and paper. We will create an array with a list of words that can be guessed in the game.

Program Data:

```
10 DIM w$(10)
20 w$(1)="cat"
30 w$(2)="mouse"
40 w$(3)="cheese"
50 w$(4)="monkey"
60 w$(5)="squirrel"
70 w$(6)="rabbit"
80 w$(7)="apple"
82 w$(8)="shovel"
84 w$(9)="treehouse"
86 w$(10)="superfluous"
87 word$=w$(INT(RND*10)+1)
88 wl = LEN(word$)
89 DIM rw$(wl)
90 DIM uw$(wl)
```

```
92 g$=""
94 FOR i=1 TO wl
95 rw$(i)=MID$(word$,i,1)
96 uw$(i)="_"
97 g$=g$+" _"
98 NEXT i
99 guessed = 0
112 tries = 0
113 l$=""
114 DIM letters$(26)
120 CLS:PRINT "Hangman"
122 PRINT "Word:";g$
130 PRINT "Guesses: ";tries
150 PRINT "Letters Guessed: ";l$
155 IF guessed = wl THEN GOTO 820
160 INPUT u$
170 IF LEN(u$)<1 THEN GOTO 120
180 IF LEN(u$)>1 THEN GOTO 120
190 FOR i=1 TO wl
200 IF u$=rw$(i) THEN GOTO 240
220 GOTO 300
240 if uw$(i)="_" THEN guessed=guessed+1
260 uw$(i)=rw$(i)
300 NEXT i
310 g$=""
320 FOR i=1 TO wl
330 g$=g$+" "+uw$(i)
340 NEXT i
360 FOR i=1 TO 26
```

```
370 IF letters$(i)=u$ THEN GOTO 120
380 IF letters$(i)="" THEN GOTO 400
390 GOTO 420
400 letters$(i)=u$:tries=tries+1
410 GOTO 422
420 NEXT i
422 l$=""
430 FOR i=1 TO 26
440 if letters$(i)="" THEN GOTO 480
460 l$=l$+letters$(i)+" "
480 NEXT i
580 IF tries-guessed < 1 THEN GOTO 120
600 CLS:PRINT "   (._.)    ":PAUSE 1000
610 IF tries-guessed > 1 THEN GOTO 630
620 GOTO 120
630 PRINT "=---[   ]---=":PAUSE 1000
640 IF tries-guessed > 2 THEN GOTO 660
650 GOTO 120
660 PRINT "    [   ]    ":PAUSE 1000
670 IF tries-guessed > 3 THEN GOTO 690
680 GOTO 120
690 PRINT "    !! !!    ":PAUSE 1000
700 IF tries-guessed > 4 THEN GOTO 720
710 GOTO 120
720 PRINT "   ==   ==   ":PAUSE 1000
730 PRINT "Game Over!":Pause 1000
740 PRINT "The word was: ";word$
760 PRINT "Play Again? y(es), n(o)"
780 INPUT z$
```

```
790 IF z$="y" THEN GOTO 10
800 CLS:PRINT "Goodbye!":Pause 2000:CLS
810 GOTO 840
820 PRINT "You Won!":Pause 1000
830 GOTO 760
```

Program #9: Simple Text Based Game

We will create a relatively simple text based adventure game. In this program we will use a 2 dimensional array to create a game "map" that stores the types of tiles the user can explore by typing (n,s,e,w). The map area will be 10x10 "tiles". We will add a feature to make sure the user can't "go off" the map. We will also add an NPC the user can talk to. As well as an object the user can pick up or drop.

This game code is quite messy, so in that regard it is a good example of how things could be written better. But if you play around with it, eventually you will get an idea of what is going on.

Program Data:

```
10 REM create the game map array
20 DIM m(10,10)
29 REM player direction variables
30 dx = 0
40 dy = 0
49 REM player location variables
50 x = 1
52 y = 1
53 REM player quest progress
54 q = 0
57 REM inventory array
58 DIM i(3)
59 REM game object array
60 DIM i$(5)
61 i$(1)="Coin"
62 i$(2)="Apple"
63 i$(3)="Rock"
64 i$(4)="Bob"
65 i$(5)="Pit"
68 REM create array for help
70 DIM h$(5)
72 h$(1)="List of Commands:"
74 h$(2)="Navigation: n,s,e,w"
76 h$(3)="Items: t,d,i"
78 h$(4)="Quit Game: q"
80 h$(5)="Conversation: c"
90 REM setup the map
100 FOR c=1 to 10
```

```
110 FOR r=1 to 10
120 IF c=5 AND r=5 THEN GOTO 200
140 roll=INT(RND*100)+1
150 IF roll < 88 THEN GOTO 260
154 IF roll >= 88 AND roll < 91 THEN t=5
160 IF roll >= 91 AND roll < 97 THEN t=1
170 IF roll >= 97 AND roll < 99 THEN t=2
180 IF roll >= 99 THEN t=3
182 GOTO 250
200 t=4
250 m(c,r)=t
260 NEXT r
262 NEXT c
267 REM e is var for current tile obj
268 e=0
269 REM game begins here
270 PRINT "-Text Adventure-"
280 PRINT "Tip: 'h' for help"
290 PRINT "Enter CMD:"
300 INPUT u$
319 REM make sure only 1 char can input
320 IF LEN(u$)<>1 THEN GOTO 470
329 REM reset player direction
330 dy=0:dx=0
359 REM 360-450 handles user input
360 IF u$="n" THEN dy=-1:GOTO 600
370 IF u$="s" THEN dy=1:GOTO 600
380 IF u$="e" THEN dx=1:GOTO 600
390 IF u$="w" THEN dx=-1:GOTO 600
```

```
400 IF e>0 AND e<4 AND u$="t" THEN GOTO 1200
410 IF e=0 AND u$="d" THEN GOTO 1100
420 IF u$="i" THEN GOTO 1000
430 IF u$="q" THEN GOTO 9000
440 IF u$="h" THEN GOTO 500
450 IF e=4 AND u$="c" THEN GOTO 700
469 REM catch invalid commands
470 PRINT "Invalid CMD!":GOTO 280
479 REM message for pits
480 PRINT "Can't go that way"
482 PRINT "a pit is there."
484 GOTO 290
499 REM print out help text
500 FOR i=1 TO 5
510 PRINT h$(i)
520 NEXT i
530 GOTO 290
599 REM movement handler code
600 IF x+dx>10 OR x+dx<1 THEN GOTO 610
602 IF y+dy>10 OR y+dy<1 THEN GOTO 610
608 IF m(x+dx,y+dy)=5 THEN GOTO 480
610 y=y+dy:x=x+dx
620 IF y<1 THEN y=1:GOTO 680
630 IF x<1 THEN x=1:GOTO 680
640 IF y>10 THEN y=10:GOTO 680
650 IF x>10 THEN x=10:GOTO 680
651 PRINT "You go ";u$;"."
652 e=m(x,y)
653 IF e=0 THEN GOTO 670
```

```
654 e$=i$(m(x,y))
655 PRINT "A ";e$;" is here."
670 GOTO 290
680 PRINT "Can't go ";u$". Edge of map!"
690 GOTO 290
699 REM NPC interaction code
700 IF q>0 THEN GOTO 739
710 PRINT "Hi, my name is Bob!"
720 PRINT "To win this game,"
730 PRINT "you have 2 options."
732 PRINT "w)hat options? b)ye"
734 INPUT r$
736 IF r$<>"w" AND r$<>"b" THEN GOTO 710
738 IF r$="b" THEN goto 290
739 IF q>1 THEN GOTO 800
740 PRINT "1: Bring me 3 coins"
742 PRINT "and 2 apples.
743 PRINT "Option 2: Bring me 3 rocks."
744 if i(1)>0 or i(3)>0 THEN q=2:GOTO 800
750 q=1:GOTO 290
800 IF i(1)>2 AND i(2)>1 THEN GOTO 900
810 IF i(3)>2 THEN GOTO 940
820 Print "Bob: I need more!"
840 GOTO 290
900 PRINT "Bob: I see you brought"
902 PRINT "me those coin and apple."
904 PRINT "Good work! You win!"
905 PAUSE 3000
906 GOTO 9000
```

```
940 PRINT "Bob: I see you have"
942 PRINT "the rocks I asked for.
944 PRINT "Excellent work!"
946 PRINT "You win! Cya!"
947 PAUSE 3000
948 GOTO 9000
999 REM inventory code
1000 PRINT "Inventory:"
1010 FOR i=1 to 3
1020 PRINT i;")";i$(i);": ";i(i)
1030 NEXT i
1032 IF u$="d" THEN GOTO 1120
1040 goto 290
1099 REM code for drop item
1100 PRINT "What do you want to drop?"
1110 GOTO 1010
1120 PRINT "Enter items number:"
1130 INPUT n
1132 IF n<1 OR n>3 THEN GOTO 1100
1133 IF i(n)=0 THEN GOTO 290
1134 IF i(n)>0 THEN i(n)=i(n)-1
1135 PRINT "Item dropped!"
1140 m(x,y)=n
1150 GOTO 290
1199 REM code for pickup item
1200 PRINT "You take the ";e$
1210 i(e)=i(e)+1
1220 m(x,y)=0
1230 GOTO 290
```

```
8999 REM endgame message
9000 CLS:PRINT "Goodbye!"
9001 PAUSE 2000:CLS
```

Program #10: Search An Array

In this program we develop a primitive search function for searching if an item is in stock in 1 of 3 stores, or we have the option of seeing everything that is in stock at a single store.

The main downside to this program is that all searches ARE case sensitive. If we want to check the inventory of "Store1" but we type "store1" we will not get a result. Or if we search "eggs" instead of "Eggs".

Other than that, this is a good program to learn how to use arrays and the input function.

Program Data:

```
10 DIM a$(3,5)
12 DIM b$(3)
11 REM setup the data:
13 REM Store1 data:
20 a$(1,1)="Store1"
30 a$(1,2)="Bananas"
40 a$(1,3)="Milk"
41 REM Store2 data:
70 a$(2,1)="Store2"
80 a$(2,2)="Eggs"
90 a$(2,3)="Milk"
100 a$(2,4)="Bacon"
120 a$(2,5)="Cheese"
121 REM Store3 data:
130 a$(3,1)="Store3"
140 a$(3,2)="Eggs"
150 a$(3,3)="Bananas"
160 a$(3,4)="Lettuce"
170 a$(3,5)="Chips"
171 REM get user input
180 INPUT ui$
181 REM clear our previous results
182 b$(1)="":b$(2)="":b$(3)=""
181 REM check to see if user is searching
a single stores inventory
190 IF ui$ = "Store1" THEN GOTO 350
200 IF ui$ = "Store2" THEN GOTO 360
210 IF ui$ = "Store3" THEN GOTO 370
211 REM Begin FOR loop to search arrays:
```

```
213 REM Loop through stores:
220 FOR store=1 TO 3
221 REM Loop through store items
230 FOR item=1 TO 5
240 IF a$(store,item)=ui$ THEN GOTO 242
241 GOTO 250
242 b$(store)="Store"+str$(store)
250 NEXT item
260 NEXT store
300 PRINT "Stores that have "+ui$+":"
310 FOR s=1 TO 3:
320 IF LEN(b$(s))>1 THEN PRINT b$(s)
330 NEXT s
331 REM go back to user input:
340 GOTO 180
350 store = 1:GOTO 400
360 store = 2:GOTO 400
370 store = 3:GOTO 400
400 PRINT ui$+" has the following items:"
410 FOR item=2 TO 5
420 PRINT a$(store,item)
430 NEXT item
440 GOTO 180
```

Chapter 8: Your Programs

"Watch out for spaghetti code!"

This chapter has 10 places to save your own programs on paper! Each program has a line to write the program's name, as well as a short description of what it does. Each program can be up to 50 lines.

This section is handy to have if you want to copy a program from your device because you want to free up a memory slot.

It can also be helpful to write programs out by hand as you can see the entire program and evaluate it easier than on a small screen.

Program 0

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500

Program 1

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500

Program 2

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500

Program 3

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500

Program 4

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500

Program 5

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500

Program 6

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500

Program 7

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500

Program 8

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500

Program 9

Name:_____

Description:_____

_____

_____

_____

_____

Program Data:

10

20

30

40

50

60

70

80

90

100

110

120

130

140

150

160

170

180

190

200

210

220

230

240

250

260

270

280

290

300

310

320

330

340

350

360

370

380

390

400

410

420

430

440

450

460

470

480

490

500