**WINTER CAMP XLV ROBOT PROGRAMMING GUIDE**

# Introduction

This is a guide for programming in BASIC, a programming language developed in the 1960’s and widely used in the first generation of home computers in the 1970’s and 80’s. It was originally designed to help people other than scientists, engineers and mathematicians use computers.

I picked BASIC for the robots because it’s not a very complicated language to learn, works well on small systems without a lot of memory or processing power, and is self-contained (no need to install anything or use a separate compiler).

Also worth noting, there are many different implementations of BASIC. The one I’m using here is based on <https://github.com/BleuLlama/TinyBasicPlus>

# Lines, Line Numbers, and Editing

Basic programs are written as lines of text. Here’s an example:

5 PRINT "Remote Control Demo Program"

10 I = JOY

20 IF I = 255 GOTO 10

25 PRINT I

30 IF I = 254 DRIVE(5)

40 IF I = 253 DRIVE(15)

50 IF I = 251 DRIVE(7)

60 IF I = 247 DRIVE(13)

70 IF I = 239 CLAW(0)

80 IF I = 223 CLAW(100)

1000 IF JOY <> 255 GOTO 1000

1005 DRIVE(0)

1010 GOTO 10

Each line needs a number at the beginning. When the program runs, BASIC will execute each line in numerical order (some exceptions later).

Entering lines is done at the basic prompt (the **>**). When typing in the program, the lines can be entered in any order. If you type in another line with the same line number as an existing line in the program, the existing line will be replaced with the new one. You can also erase lines by entering in a line number with no further text.

Note that often, you will see the line numbers between each line spaced apart by 10 {10, 20, 30, etc.}. This is done in case you might need to put in another line between 2 existing lines.

Two important commands for editing in BASIC are:

**LIST** - which shows all lines of the basic program on screen, and

**NEW -** which erases all lines of a program

Also of note at this point – commands in this BASIC can be entered in CAPS or small letters. This isn’t true of all basic variants, though.

Here’s an example of entering a program into BASIC:

>new previous program is erased

>10 ‘this is line 10 entering some lines

>20 ‘this is line 20

>30 ‘this is line 30

>40 ‘this is line 40

>30 line 30 is erased

>25 ‘this is line 25 line 25 inserted between 2 lines

>20 ‘this is the new line 20 line 20 is replaced

>list

Result:

10 ‘this is line 10

20 ‘this is the new line 20

25 ‘this is line 25

40 ‘this is line 40

# Comments

The commands I entered in the above example are just comments, which are completely ignored when the program is running. You can make comment lines by using an apostrophe as above, or by using the **REM** keyword, as in:

10 REM this is a comment

# Loading, Saving, and FILES

Basic programs can be saved, and they are stored on the robot’s internal memory.

A program can be saved by using the **SAVE** keyword, and loaded using **LOAD.** To do this, use SAVE or LOAD, followed by the name of the program, in quotation marks.

The names of all saved program can be seen with the **FILES** keyword.

Here’s an example:

>new create new program

>10 ‘this is a comment

>list list program I just entered

10 ‘this is a comment

>files show the list of saved programs

there aren’t any

>save “program” save the program

>files

program file “program” now exists

>new erase the current program

>list show that the program is gone

>load “program” load the saved program again

>list now, it’s back

10 ‘this is a comment

>

# A first program and PRINT

Let’s try writing our first program:

>new

>10 print “hello”; does not move onto a newline

>20 print “ Winter Camp”

Now, we can run it:

>run

hello Winter Camp

OK

>

Our program is just 2 lines, and uses the **PRINT** statement to display the text “hello” on the screen. PRINT is how you can make your program display things. It can also display variables, which I’ll get to in a minute.

Each line of PRINT will display on it’s own line, unless you put a semicolon at the end of the line. In that case, PRINT will not go onto the next line, and further PRINT statements will continue on the same line.

The program is started with **RUN**. When this command is entered, BASIC starts executing the program at the lowest line number. With this version of BASIC, RUN always starts the program at the beginning

When the program is finished, BASIC returns to the prompt (>).

# Variables

Now, we can get into how our program make decisions.

The first concept is variables. These are used to store numbers; you might recognize them as the letters from your math class. This version of BASIC has 26 of them, the letters A to Z. They can store any number from -32678 and 32767, and are limited to integers (no decimal points).

**PRINT** can be used to display variables as well, and you can do multiplication, division, addition and subtraction on them:

>list

10 A = 3

20 B = 2

30 C = A + B

40 PRINT C a and b are still the values from earlier

50 C = A – B but c gets overwritten

60 PRINT C

70 C = A \* B

80 PRINT C

90 C = A / B

100 PRINT C

OK

>run

5

1

6

1 yes, 3/2 = 1 here since integers only

# INPUT

In the above example, we put lines in the program to set the value of some variables. If we wanted to change them, we would have to re-enter the line with the new value, and run the program again. Alternately, we can use INPUT to ask the user for a value when the program is running.

>new

>10 input J get the user to input a new value for j

>20 print "you entered:";

>30 print j

>run

?10

you entered:10

OK

>

When an INPUT statement is encountered in the program, it will print a question mark (?), then pause and wait for the user to type in a number and press Enter.

# Comparisons and IF

This is probably the most important concept in BASIC, because we can use it to have the program make decisions based on certain conditions.

The general format of IF works like this:

IF <some number> <comparison> <another number> <something to do>

The comparisons you can use are these:

= are equal

> is greater than

< is less than

>= is greater than or equal to

<= is less than or equal to

<> are not equal

And here is an example:

>list

10 A = 10

20 IF A = 10 PRINT "is ten"

30 IF A > 15 PRINT "more than 15"

40 IF A > 10 PRINT "more than ten"

50 IF A < 20 PRINT "less than 20"

60 IF A >= 8 PRINT "more or equal to 8"

70 IF A <> 30 PRINT "is not 30"

OK

>run

is ten

less than 20

more or equal to 8

is not 30

OK

>

The “something to do” here is a print command in this case, though most any basic command can be used.

# GOTO

Another key concept in basic, this is how you can direct the program flow. It allows you to make the program move to any line number. Once there, it will continue executing in increasing line number order as usual.

>new

>10 PRINT "start"

>20 GOTO 100

>50 PRINT "this is line 50"

>60 J = 10

>70 IF J = 10 GOTO 200

>100 PRINT "this is line 100"

>110 GOTO 50

>200 PRINT "this is an infinite loop"

>210 GOTO 200

>run

start

this is line 100

this is line 50

this is an infinite loop

this is an infinite loop

this is an infinite loop

this is an infinite loop

this is an infinite loop

this is an infinite loop

this is an infinite loop

GOTO can also be used to create a loop, as shown above. To get out of an infinite loop, use the **Break** button at the top of the window.

As shown in the above example, GOTO can be used with IF to jump to a line if some condition is met.

Here’s another example of using GOTO to make a loop:

>10 print "stuck in a loop"

>20 input a

>30 if a = 20 goto 100

>40 goto 10

>100 print "escaped the loop!"

>run

stuck in a loop

?12

stuck in a loop

?4

stuck in a loop

?23

stuck in a loop

?10

stuck in a loop

?20

escaped the loop!

OK

>

Here, the program is stuck in a loop, until variable A is equal to 20.

# GOSUB and RETURN

**GOSUB** is a lot like GOTO, but lets you come back to the line after where you left from. This is useful if you have a section of code that you would like to use multiple times. At the end of the section you GOSUB to, use the **RETURN** keyword to go back.

Here’s an example without GOSUB:

>new

>10 a = 12

>20 print "I have ";

>30 print a;

>40 print " bananas"

>50 a = 23

>60 print "I have ";

>70 print a;

>80 print " bananas"

>90 a = 11

>100 print "I have ";

>110 print a;

>120 print " bananas"

>130 a = 9

>140 print "I have ";

>150 print a;

>160 print " bananas"

>run

I have 12 bananas

I have 23 bananas

I have 11 bananas

I have 9 bananas

OK

>

and here is with gosub. You only need to write the text printing part once, then refer to it using the line number.

>new

>10 a = 12

>20 gosub 100 jump to line 100

>30 a = 23 the return on line 130 goes back here

>40 gosub 100 and so on

>50 a = 11

>60 gosub 100

>70 a = 9

>80 gosub 100

>90 stop

>100 print "I have ";

>110 print a;

>120 print " bananas"

>130 return

>run

I have 12 bananas

I have 23 bananas

I have 11 bananas

I have 9 bananas

OK

>

# STOP

This is used to stop a program and return to the BASIC prompt. It isn’t always needed, as if the program reaches the end of the program on its own and there is not a further line to move onto, it will end by itself. But, it can be used to end a program from the middle.

Using STOP does also turn off both drive motors on the robot.

>new

>10 print "hello"

>20 stop

>30 print "this line never gets printed"

>run

hello

OK

>

# FOR Loops

This is another way to make a loop, with a certain number of steps. You can do the same with GOTO and IF, but this might look easier to understand. Here’s an example:

>10 for i = 1 to 10

>20 print i

>30 next i

>run

1

2

3

4

5

6

7

8

9

10

OK

>

The FOR part of the loop sets up a variable to use, and the start and end values it will use in the loop.At the end of the loop NEXT <variable> is used to jump back to the start of the loop.

# DELAY

A robot specific command, this is used to introduce a time delay within the program. Without it, BASIC will process lines as fast as possible. The value passed to DELAY is in milliseconds. **Break** will not work during DELAY.

>10 print "this line prints now"

>20 delay(1000)

>30 print "this line prints one second later"

>run

this line prints now

this line prints one second later

OK

>

# DRIVE

This robot specific command is used to operate the motors connected to the 2 wheels. Values for DRIVE are based on a 4-bit binary code, as such:

<right motor reverse> <right motor on> <left motor reverse> <left motor on>

Which gives the following useful values for DRIVE:

|  |  |  |
| --- | --- | --- |
| Binary | Decimal | Action |
| 0000 | 0 | Both motors off |
| 0001 | 1 | Left motor forwards, right motor off |
| 0011 | 3 | Left motor reverse, right motor off |
| 0100 | 4 | Right motor forwards, left motor off |
| 0101 | 5 | Both motors forwards |
| 0111 | 7 | Left motor reverse, right motor forwards |
| 1100 | 12 | Right motor reverse, left motor off |
| 1101 | 13 | Left motor forwards, right motor reverse |
| 1111 | 15 | Both motors reverse |

>10 'demo - drives forwards, then backwards

>20 drive(5)

>30 delay(1000)

>40 drive(15)

>50 delay(1000)

>60 drive(0)

>run

# CLAW

Operates the claw on the front of the robot. Use with caution; if restrained, the claw can draw too much power and reset the chip, and you lose your program. Be sure to SAVE beforehand.

For reference, CLAW(0) opens the claw fully; CLAW(55) will capture a golf ball.

>5 'Open and close the claw 10 times

>10 for i = 1 to 10

>20 claw(0)

>30 delay(1000)

>40 claw(55)

>50 delay(1000)

>60 next i

>run

# Another Timing thing

In addition to DELAY(), I have included a second way to deal with time. DELAY() will cause the program to pause until the delay is finished. These clock functions allow you to set up a clock that runs in the background.

Note that CLK updates in increments of 10 ms (1/100 seconds), unlike DELAY which works using 1 ms increments.

**CLKRUN** allows the clock to run, much like a start button on a stopwatch

**CLKSTOP** stops the clock, like a stop button on a stopwatch

**CLKSET(value)** sets the clock to the value specified. Use CLKSET(0) to reset the clock. This will set the clock whether the clock is running or stopped.

**CLK** is the current value of the clock. Counts upwards. Use it like a read only variable.

>10 PRINT "clock start"

>20 CLKSTOP

>30 CLKSET(1000)

>40 CLKRUN

>45 ‘do something here if you want

>50 IF CLK < 2000 GOTO 45

>60 PRINT "this prints after 10 seconds"

>70 STOP

>run

clock start

this prints after 10 seconds

# Light Sensor

There is a light sensor on the bottom of the robot. It’s value can be read at any time in BASIC by using the **LIGHT** keyword. Use it like a read only variable.

In the below example, the program is run and the robot quickly placed onto a white surface.

>10 print LIGHT

>20 goto 10

>run

0

0

0

0

1

1

1

1

# Other User Input

On the TRS-80 and most other BASIC computers, it was possible to read from the keyboard or joystick directly from BASIC, without pausing the program to use INPUT.

For the robot, I implemented this using some keyboard keys on the web browser terminal. While the program is running, it is possible to read the state of the arrow keys and keys 1,2,3, and 4.

To do this, read the **JOY** keyword, used like a read only variable.

The values from JOY are an inverted 8-bit value. Bits go to 0 when the key is pressed, and are 1 when not pressed. So, when no keys are pressed, JOY will be set to 255.

If you don’t feel like doing the math, you can just print out the JOY value and press different keys to see what values you might need. This example shows how it is possible to use the JOY keyword to make the robot a remote control car.

5 PRINT "Remote Control Demo Program"

10 I = JOY

20 IF I = 255 GOTO 10

25 PRINT I

30 IF I = 254 DRIVE(5)

40 IF I = 253 DRIVE(15)

50 IF I = 251 DRIVE(7)

60 IF I = 247 DRIVE(13)

70 IF I = 239 CLAW(0)

80 IF I = 223 CLAW(100)

1000 IF JOY <> 255 GOTO 1000

1005 DRIVE(0)

1010 GOTO 10

And the bits for each key are:

|  |  |
| --- | --- |
| Bit (0 = Least Significant [ones place]) | Key |
| 0 | Up Arrow |
| 1 | Down Arrow |
| 2 | Left Arrow |
| 3 | Right Arrow |
| 4 | “One” (1) key |
| 5 | “Two” (2) key |
| 6 | “Three” (3) key |
| 7 | “Four” (4) key |