

# Great Cow BASIC

Beginner's Tutorial

By Hugh Considine

# Introduction

Great Cow BASIC is a simple yet powerful language for making programs that run on Microchip PIC microcontrollers. It is simple and easy to use enough to make the beginner comfortable, and yet has enough power to satisfy the experienced programmer.

Often when we think of programming a computer, we imagine having to type in thousands of lines of weird commands that take years to memorise. This is not so! With GCBASIC, you only have to type in a few easy to remember commands like “if”, “wait” and “set”. GCBASIC is called a compiler, and its job is to translate the simple words that we type in into a series of numbers that the robot can understand.

With this tutorial, we will start out by spending a few minutes learning the basic principles of programming. Then in tutorial 1, we shall put on our robot expert hats, and start programming our robots. At first, we’ll just make them do simple things like flash lights, and drive around. Then, we’ll teach them to think, by responding to the world around them. Finally, we’ll find out how to make the robots do something over and over again.

In tutorial 2, we’ll learn how to make our programs neater, so that we can understand what they do in a year’s time. Once our programs are neat, we can move on to tutorial 3, where we will do fun things like send messages to other robots, and read sensors with higher precision.

Finally in tutorial 4, we’ll learn some of the technical aspects of programming, so that we can program the robots that we design ourselves.

## **To complete this tutorial, you will need:**

- One of these robots:
  - eLabtronics eRacer
- A computer with GCBASIC installed correctly
- Crimson Editor, set up to work with GCBASIC (This is included in the installer)
- A suitable programmer for your robot, set up for use with GCBASIC.
- About 15 mins for the introductory tutorial, and approx. 45-90 minutes for each of the following tutorials.

# Contents

<a href="#">Beginner's Tutorial.....</a>	<a href="#">1</a>
<a href="#">Introductory Tutorial: What is programming?.....</a>	<a href="#">4</a>
<a href="#">Tutorial 1: Basics of GCBASIC.....</a>	<a href="#">5</a>
<a href="#">Exercise 1-1: Introducing your robot to GCBASIC.....</a>	<a href="#">6</a>
<a href="#">Exercise 1-3: Waiting.....</a>	<a href="#">9</a>
<a href="#">.....</a>	<a href="#">11</a>
<a href="#">Exercise 1-4: Repeating things over and over.....</a>	<a href="#">12</a>
<a href="#">Exercise 1-5: Making decisions.....</a>	<a href="#">13</a>
<a href="#">Application 1: Following a line.....</a>	<a href="#">14</a>
<a href="#">Tutorial 2: Thinking Better.....</a>	<a href="#">15</a>
<a href="#">Exercise 2-1: Using variables.....</a>	<a href="#">16</a>
<a href="#">Exercise 2-2: Repeating things a set number of times.....</a>	<a href="#">17</a>
<a href="#">Exercise 2-3: Repeating things until something happens.....</a>	<a href="#">18</a>
<a href="#">Exercise 2-4: Using subroutines.....</a>	<a href="#">19</a>
<a href="#">Exercise 2-5: Using constants.....</a>	<a href="#">20</a>

# Introductory Tutorial: What is programming?

When programming, it is important to remember one key thing: Computers are not smart. They cannot think without a lot of help from the programmer. **The programmer must break everything down into little steps.**

A human can see an instruction that says, “move to a wall”, and follow it without having to think about it. A robot, on the other hand, needs a lot more help. How does it know how to move? How does it know when it is at the wall? What does it do once it is at the wall? In this example, we would have to program the robot to do this:

- Start moving
- Keep moving until the wall is hit
- Stop moving when the wall is hit

**We call these simple steps “pseudo code”,** because the steps are very similar to what is actually programmed into the robot. Once you have become accustomed to having to spell everything out to the computer, it becomes easy. To practice, try filling out this table: (the first two have been done for you)

Task	Instructions for a robot
Move to a wall	<i>Start driving forwards</i> <i>Wait until the sensors say that a wall has been hit</i> <i>Stop moving</i>
Drive in a zigzag pattern	<i>Turn left</i> <i>Wait a little while</i> <i>Turn right</i> <i>Wait a little while</i> <i>Repeat the previous instructions</i>
Turn on a light when a switch is pressed	
Drive forwards for 12 seconds	

Once you have completed the table, please check your answers with those in the back of this tutorial, and then complete this statement:

**Programming is:**

---

---

# Tutorial 1: Basics of GCBASIC

In the introductory tutorial, we found that programs are just lists of simple commands that a robot can understand. Now, we will learn the actual commands that the robot can recognize. We'll start with the simple commands in order to make ourselves familiar with GCBASIC, and then we'll move on to some of the decision making commands that allow robots to carry out useful tasks.

## **In this tutorial:**

<a href="#">Exercise 1-1: Introducing your robot to GCBASIC.....</a>	<a href="#">6</a>
<a href="#">Exercise 1-3: Waiting.....</a>	<a href="#">9</a>
<a href="#">.....</a>	<a href="#">11</a>
<a href="#">Exercise 1-4: Repeating things over and over.....</a>	<a href="#">12</a>
<a href="#">Exercise 1-5: Making decisions.....</a>	<a href="#">13</a>
<a href="#">Application 1: Following a line.....</a>	<a href="#">14</a>
<a href="#">Exercise 2-1: Using variables.....</a>	<a href="#">16</a>
<a href="#">Exercise 2-2: Repeating things a set number of times.....</a>	<a href="#">17</a>
<a href="#">Exercise 2-3: Repeating things until something happens.....</a>	<a href="#">18</a>
<a href="#">Exercise 2-4: Using subroutines.....</a>	<a href="#">19</a>
<a href="#">Exercise 2-5: Using constants.....</a>	<a href="#">20</a>

### Exercise 1-1: Introducing your robot to GCBASIC

One of the first things that we must do when writing a program with GCBASIC is tell GCBASIC what type of robot the program is for. When we put our program into GCBASIC, our program is converted into a series of numbers that the robot can understand. Different robots, however, require different sets of numbers to achieve a given task. If we do not tell GCBASIC the type of robot it is dealing with, it cannot produce the correct set of numbers and therefore our program will not work.

To tell GCBASIC what type of robot we have is easy. We just need to use the `#include` command, to tell it to read a file that has information on our robot, such as where we have connected motors, lights, and other things.

We are now going to make our first program to download to the robot. Open up Crimson Editor, and click New. Then, click Save and type a name for our program, so that if something goes wrong, we will not lose any of our work.

To help make the program easier to read, we can set Crimson Editor to colour code our program. Click the Document menu, then Syntax Type, and then select GCBASIC.

**Now, it is time to actually write the program. Just follow these two simple steps:**

1. **Tell GCBASIC what type of robot it is dealing with.** In the editor, type:

```
#include <eracer.h>
```

2. **Add a command, so that the program actually does something we can see.** For now, lets just turn on the green light under the robot. Under InitBot, type:

```
Set Green on
```

That's all there is to it! Press F9, and click write on the programmer. Wait a few seconds, and the green light will come on! Don't worry about capitals! GCBASIC doesn't care, so why should we? Congratulations, we've just made our first program using GCBASIC!

**Before we move on to Exercise 1-2, let's make a note of what we have learnt:**

When we make a new program, we must first tell GCBASIC what sort of robot it is dealing with, using the command \_\_\_\_\_. Next, we must set up the robot, using the command \_\_\_\_\_.

For an example of the file created here, please see the file Exercise 1-1.txt in the Tutorial Programs folder.

**Exercise 1-2: Moving, and turning things on and off**

In the last exercise, we turned on the green light under the robot. However, there are several other things we can set on and off. To turn things on and off, we use the SET command. When using the SET command, we must type SET, and then the name of the thing we want to set, and then ON or OFF, depending on what we want to do.

Here is a list of the things that we can turn on and off. Don't worry, there is no need to memorise these (although they are so simple, we probably will anyway!)

Item	GCBASIC Name
Top Left LED	TL_LED
Top Right LED	TR_LED
Bottom Red LED	RED
Bottom Green LED	GREEN
Buzzer	BUZZER

Let's make a program that will turn on the green led, and the left led on top of the robot. To do this, we must:

1. **Create and save a new program.** Forgotten how? See Exercise 1-1.
2. **Add a comment to the top of the program.** To do this, put an apostrophe (') at the start of a new line, and then type in a comment. For example, we could write:

```
'Program to turn on the top left LED
```

3. **Choose the robot and then set it up.** Remember the first command we used in Exercise 1-1? We need to add it to our new program. The command is:

```
#include <eracer.h>
```

4. **Turn on the green LED.** Remember the command from Exercise 1-1?

```
Set Green on
```

5. **Turn on the top left LED.** The command is very similar to the command for the green LED:

```
Set TL_LED on
```

6. **Download our program.** Press F9, and then click Write

Now, the top left and green LEDs will turn on when we run our program!

How about trying to do some programming on your own now? Try and program in these combinations:

- Bottom red LED and buzzer on
- Both top LEDs on
- All LEDs on
- Just the buzzer on (*careful – this one can be annoying to other people!*)

We haven't covered how to turn things off, but then there's really no need to until the next tutorial. The command to turn something off is the same as the command to turn something on, except we write off instead of on! Confusing, isn't it!

Let's continue with Exercise 1-2. Now, we get to make the robot move! Again, the commands to make the robot move are really very simple. Just refer to this table:

<b>Movement</b>	<b>GCBASIC Command</b>
Forward	Forward
Reverse	Reverse
Turn left	TurnLeft
Turn right	TurnRight
Spin to the left	SpinLeft
Spin to the right	SpinRight
Stop moving	Stop

Now, we shall make the robot drive forwards:

- 1. Create and Save a new program.**
- 2. Choose a robot, and set it up.**

```
#include <eracer.h>
```

- 3. Add the command to make the robot drive forwards.** Refer to the table above to find the command.
- 4. Download the program to the robot (Press F9)**

The robot should now drive forwards! Now try:

- Spinning to the left
- Turning right
- Driving forwards, using one of the LEDs as a headlight.
- Reversing, using the red LEDs as taillights

Make a brief mental note of how the robot moves. It will come in handy later.

For an example of the file created here, please see the file Exercise 1-2.txt in the Tutorial Programs folder.



### Exercise 1-3: Waiting

Having lights turn on when we switch on the robot gets boring quickly. After all, we could just use a simple switch to accomplish the same thing. Wouldn't it be much better if the robot could actually do something useful, like act as a timer, or do a silly yet entertaining dance?

This is where the Wait command is useful. The Wait command can make the robot do nothing for an amount of time, which we specify. To use the Wait command, we write Wait, and then a number between 0 and 255, and then the units of the number that we typed in. The available units are:

Units	GCBASIC units
Microseconds	us
Microseconds * 10	10us
Milliseconds	ms
Milliseconds * 10	10ms
Seconds	s
Minutes	m
Hours	h

To make the robot wait for 5 seconds, we would use the command Wait 5 s. Normally, we can just use the units that we would use most of the time. However, supposing we want to make the robot wait for 500 ms? We cannot use the command Wait 500 ms, as 500 is greater than 255<sup>1</sup>. This is why there are 10us and 10ms units. We can use the command Wait 50 10ms, and the program will work fine.

For a simple demonstration of the Wait command, let's just turn on the green LED for 2 seconds. To do this:

1. **Create a new file in Crimson Editor, and Save it.**
2. **Add the line to set up the robot.**
3. **Add a command to turn on the green LED.** Refer to the previous exercise if you have forgotten.
4. **Below that, type in the necessary wait command:**

Wait 2 s

5. **Add a command to turn the green LED off again.**
6. **Download the program!**

---

<sup>1</sup> This is due to a technical limitation of PIC microprocessors that renders them unable to handle numbers greater than 255. This may be fixed in a future version of GCBASIC, but until then we'll need to find ways to work around this limit.

If we have written our program correctly, the LED will turn on for 2 seconds, and then switch off again.

For an example of the file created here, please see the file Exercise 1-3.txt in the Tutorial Programs folder.

Let's go back to one of the scenarios that we wrote pseudo code for back in the introductory tutorial, where we had to make a robot drive forwards for 12 seconds. With the pseudo code from the introductory tutorial, and all of the things we have learned in the past two exercises, this should be easy. Try writing it yourself!

So far, we have made the robot wait for a set amount of time. This is useful sometimes, but what if we only want it to wait until something happens? Supposing, say, we want to make a robot alarm, which we can turn off. This is where the Wait Until command is useful. When we use the Wait until command, we must type Wait Until, and then the name of an input, and then the state of the input:

Input	GCBASIC name	States
Button	BUTTON	Pressed, Released
Left light sensor	LDR_Left	Light, Dark
Right light sensor	LDR_Right	Light, Dark

Say we want the program to wait until the button is pressed. We use the command:

Wait Until Button Pressed

It doesn't seem too hard to understand, does it? And once again, don't worry about the capitalisation: We could make every third letter a capital, and our program would still work.

Let's make a program that can act as an alarm. To do this, we must make the robot wait until someone shines a light at it. When the thief shines their light on the robot, the robot can beep, flash some lights, and spin, to scare away the thief. Here are the steps:

1. **Create and save a new program in Crimson Editor.**
2. **Add the line to introduce the robot to GCBASIC, and set it up.**
3. **Add the command that waits for the light sensor to detect light:**

Wait Until LDR\_Left Light

4. **Add a few commands to scare the thief.** Why not make your program unique, and think up these yourself? Perhaps you could take some of the commands that we used in Exercise 1-2, and use them!
5. **Download your program.**
6. **Place the robot in a dark room.** Bend the left light sensor so that it faces towards the door. If you have programmed the robot to move, make sure it is not going to fall anywhere.
7. **Creep into the room like a thief, and "accidentally" shine a torch at the robot.** It should start doing whatever you programmed it to.

That's probably the first useful thing we've done with GCBASIC. Hopefully it is by no means the last, though!

## Great Cow BASIC Beginners Tutorial

For an example of the file created here, please see the file Exercise 1-3b.txt in the Tutorial Programs folder.

### Exercise 1-4: Repeating things over and over

In the last program, there was one major flaw: The alarm only worked once! A thief could set it off, leave, and then come back later without any fear. Wouldn't it be great if there were a way to make the alarm reset itself?

This is where the GOTO command is useful. The GOTO command will make the robot stop reading the part of the program it is up to, and move to another spot. This is a very useful command for repeating things over and over again.

But how do we mark the spot we want the program to go to? The answer is that we used "line labels". A line label is a word (or words) that are on a line on their own, with just a colon (:) for company. For example, these are all valid labels:

- Main:
- Start:
- RepeatAlarm:

We can make up as many line labels as we want, and we can call them almost anything we want. **There are only three rules for line labels:**

1. **Do not use the same name for several places! The robot will get confused.**
2. **Do not use the names of commands as labels, unless they are with another word. For example, do not call a label "Set". (SetLightOn: and similar are fine, since other things accompany the command.)**
3. **No Spaces!**

Let's try using GOTO now! We can modify the program from the previous example here:

1. **Open the file that we created in the last example in Crimson Editor.**
2. **We need to add a line label at the start of the main program.** To do this, click just before the "Wait Until" command.
3. **Type in the label.** Let's call this label "AlarmStart"

AlarmStart:

4. **Now, we need to add a GOTO command to jump back to the label.** This needs to go at the end of the program, so that the robot gets to it just after it's scared away the thief:

goto AlarmStart

Notice how we left off the colon (:) in the goto. We only add the colon at the end of the actual label.

5. **Save changes, and download the program to the robot!**
6. **Now, creep into the room like a thief over and over.** The robot will see every time!

## Great Cow BASIC Beginners Tutorial

For an example of the file created here, please see the file Exercise 1-4.txt in the Tutorial Programs folder.

### Exercise 1-5: Making decisions

Up to now, each program has been list of things the robot has to do. This is useful sometimes, but to be really useful a robot has to be able to understand, at least partially, what is going on around it.

This is where the IF command comes in handy. The IF command allows the robot to only run a certain piece of code if certain conditions are met. Perhaps, say, the robot should drive forwards when a button is released, and backwards when a button is pressed. Another more practical example could be a night-light that only turns on when it is dark.

To use the IF command, we type IF, then the condition that has to be true (See Exercise 1-3), THEN, and then the command to carry out if the condition is true. If we wanted the buzzer to turn on when we press the button, we would type:

```
IF button pressed THEN Set buzzer on
```

Let's try programming a night-light now:

1. **Open Crimson Editor, and create and save a new file.**
2. **Add the usual commands to set up the robot.** Glance back at exercise 1 if you have forgotten what these are.
3. **Add a line label at the start of the program.** We want the night-light to last more than one night, don't we? Let's call this label "Start:" since it's easy to remember.

```
Start:
```

4. **Now we need to add the IF commands to turn the light on and off:**

```
IF LDR_Right Light THEN Set Green OFF  
IF LDR_Right Dark THEN Set Green ON
```

5. **Finally, we add the goto command to make the robot check over and over again:**

```
goto Start
```

6. **Now download to the robot, and try out the program!**

That concludes the first part of Tutorial 1. The time has now come to apply the knowledge gained. Turn over for the first Application!

## Great Cow BASIC Beginners Tutorial

For an example of the file created here, please see the file Exercise 1-5.txt in the Tutorial Programs folder.

## **Application 1: Following a line**

### **The Task:**

To make the robot follow a line.

### **Why is this relevant?**

Many robots in the real world have to follow lines. There are:

- Industrial robots that drive around factories, picking up and dropping off things that would otherwise have to be delivered by hand.
- Lawn mowing robots that follow a wire buried under the lawn, in order to achieve an even cut, without running into the roses or chasing after people!

While this application uses a coloured line and light sensor instead of buried wires and a magnetic sensor, the basic principles are the same.

### **Special Requirements:**

- One of these:
  - A white line on a dark floor
  - A large white paper mat with a black line
  - Some other surface with a line, where the line contrasts sharply with the surface.

### **Programming Hints:**

- The robot needs to stay over the line.
- The easiest way to do this is to make it turn one way when it sees the line and the other when it doesn't. There are other ways to follow a line – you may like to try finding and using these as well!
- Make sure that the robot turns, rather than spinning. If it spins, it'll just shake from side to side. If it turns, it'll zigzag its way along the track.

### **I'm stuck! Where can I see an example program?**

Open the file “Application 1.txt” in the Tutorial Programs folder. It is an example of a line following program that you can look at and use.



## Tutorial 2: Thinking Better

Up to this point, the robots we've programmed have all been very simple affairs. They've all waited until something has happened, and then done a particular action. Further to this, all of the actions have been written in a way that would make the program very messy and hard to read if there were more than a few.

In this tutorial, we'll learn how to store information so that it can be accessed later, and we'll find out ways to make our programs neat. Along the way, we'll also discover some more ways to repeat code.

### **In this tutorial:**

<a href="#"><u>Exercise 1-1: Introducing your robot to GCBASIC.....</u></a>	<a href="#"><u>6</u></a>
<a href="#"><u>Exercise 1-3: Waiting.....</u></a>	<a href="#"><u>9</u></a>
<a href="#"><u>.....</u></a>	<a href="#"><u>11</u></a>
<a href="#"><u>Exercise 1-4: Repeating things over and over.....</u></a>	<a href="#"><u>12</u></a>
<a href="#"><u>Exercise 1-5: Making decisions.....</u></a>	<a href="#"><u>13</u></a>
<a href="#"><u>Application 1: Following a line.....</u></a>	<a href="#"><u>14</u></a>
<a href="#"><u>Exercise 2-1: Using variables.....</u></a>	<a href="#"><u>16</u></a>
<a href="#"><u>Exercise 2-2: Repeating things a set number of times.....</u></a>	<a href="#"><u>17</u></a>
<a href="#"><u>Exercise 2-3: Repeating things until something happens.....</u></a>	<a href="#"><u>18</u></a>
<a href="#"><u>Exercise 2-4: Using subroutines.....</u></a>	<a href="#"><u>19</u></a>
<a href="#"><u>Exercise 2-5: Using constants.....</u></a>	<a href="#"><u>20</u></a>

**Exercise 2-1: Using variables**

**Exercise 2-2: Repeating things a set number of times**

**Exercise 2-3: Repeating things until something happens**

**Exercise 2-4: Using subroutines**

**Exercise 2-5: Using constants**