

# FLL Programming 101 RoboLab™

*August 2004*

*Version 1.0*





# Legal Stuff

---

© 2002-3 INSciTE in agreement with, and permission from FIRST and the LEGO Group. This document is developed by INSciTE and is not an official FLL document from FIRST and the LEGO Group. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit

<http://creativecommons.org/licenses/by-nc-sa/2.0/>

or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

LEGO®, ROBO LAB, and MINDSTORMS™ are trademarks of the LEGO Group used here with special permission. FIRST™ LEGO® League is a trademark owned by FIRST (For Inspiration and Recognition of Science and Technology) and the LEGO Group used here with special permission. INSciTE™ is a trademark of Innovations in Science and Technology Education.

INSciTE

PO Box 41221

Plymouth, MN 55441

[www.hightechkids.org](http://www.hightechkids.org)



# Creative Commons License

---

- High Tech Kids is committed to making the best possible training material. Since HTK has such a dynamic and talented global community, the best training material and processes, will naturally come from a team effort.
- Professionally, the open source software movement has shown that far flung software developers can cooperate to create robust and widely used software. The open source process is a model High Tech Kids wants to emulate for much of the material we develop. The open source software license is a key enabler in this process. That is why we have chosen to make this work available via a Creative Commons license. Your usage rights are summarized below, but please check the complete license at: <http://creativecommons.org/licenses/by-nc-sa/2.0/>.

# Creative Commons License

---

## Attribution-NonCommercial-ShareAlike 2.0

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works

Under the following conditions:



**Attribution.** You must give the original author credit.



**Noncommercial.** You may not use this work for commercial purposes.



**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.



# Credits

---

This presentation was developed by Fred Rose. The accompanying labs were originally done in RCX Code by Joel Stone and converted to ROBOLAB by Doug Frevert. A portion of the material is taken from “*Building LEGO Robots for FIRST LEGO League*” by Dean Hystad. Amy Harris defined the 10 programming steps. Eric Engstrom, Jen Reichow, and Ted Cochran reviewed ongoing drafts. Eric taught the first class and helped modify the content accordingly.

# Computer Programming 101

---

- Objective
  - Develop a basic approach to, and understanding of, programming the RCX
- Structure
  - Theory
  - Examples specific to language
  - Hands-on
- What this class **is**
  - Teach an approach to programming
- What this class **is not**
  - Exhaustive reference on every language command



# Class Agenda

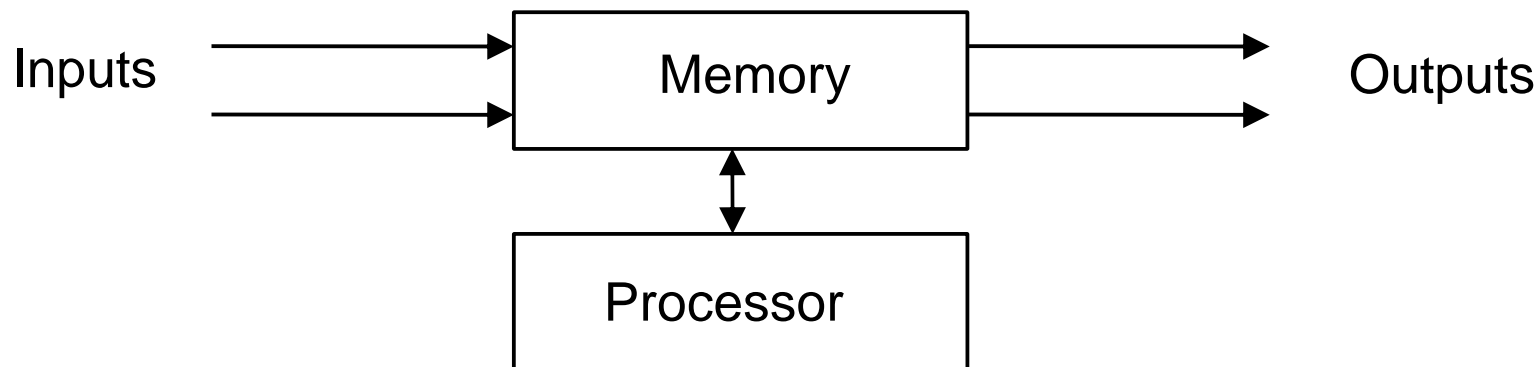
---

- Computer Basics
- The Programming Environment
- Simple Commands
  - Lab #1
- Problem Solving
- Keep It Simple
  - Lab #2
- Sensors
  - Lab #3
- Program Structures
  - Lab #4
- Advanced Topics
- Putting It Altogether

# Computer Basics

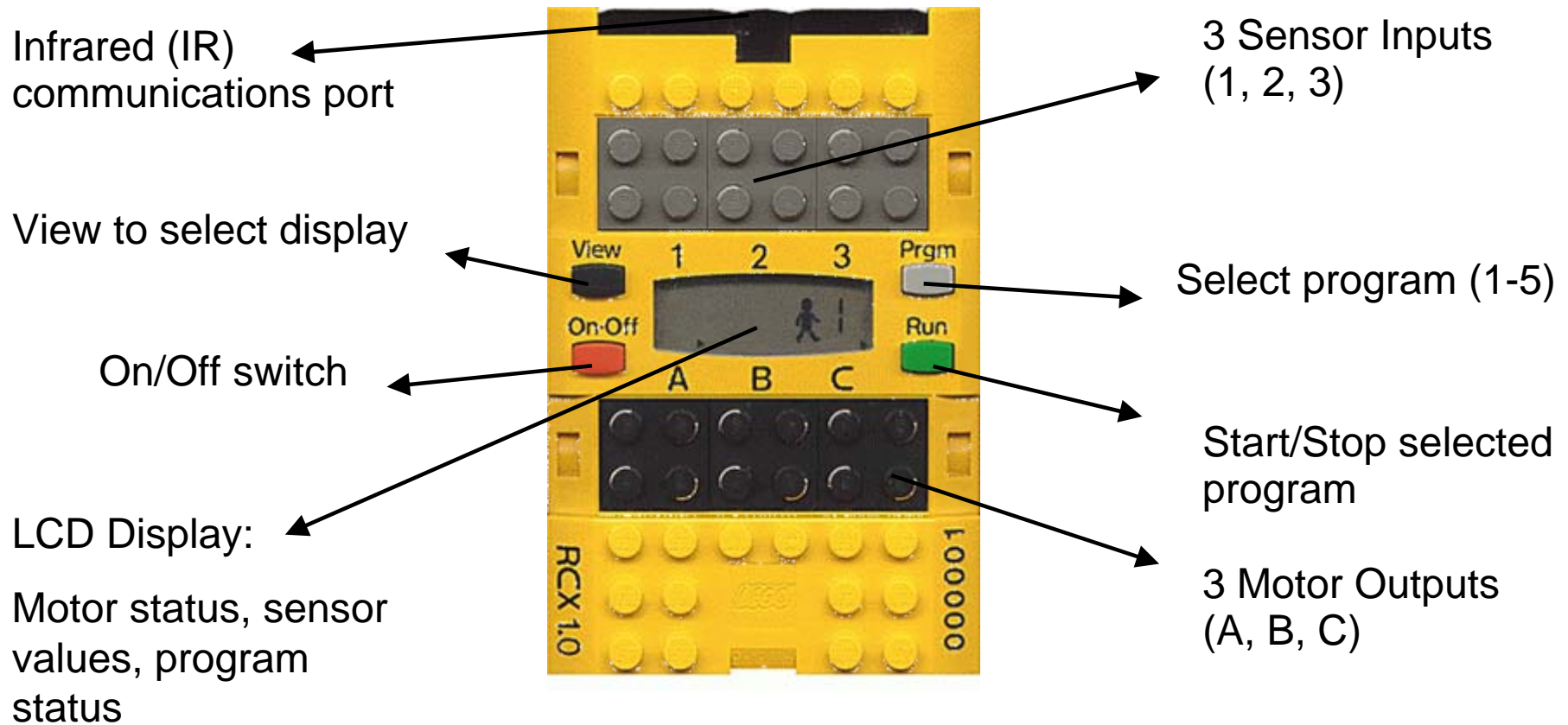
# The Computer (Generic)

---



- Processor executes commands.
- Memory stores program and data.
- Input devices transfer information from outside world into computer.
- Output devices are vice versa.

# RCX

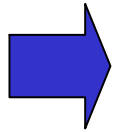
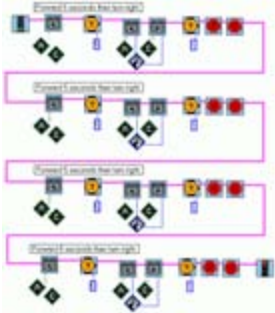


**Processor:** Hitachi H8 8 bit microcontroller running at 5 to 20Mhz

**Memory:** 32K of RAM

# RCX Firmware

## Software



Bytecodes  
{SetPower(A,C,8)}

## Firmware Setup



Your PC ↑

Download ↓

Your RCX ↓

**RAM (Random Access Memory) \***

\* RAM loses its data with no power! Careful changing batteries!!

**Your Programs  
Firmware**

**ROM (read only memory)  
Processor (Hitachi)**



**For FLL purposes, think of firmware as the operating system (like Windows XP or Max OSX) for the RCX**

# Firmware Loaded?

- Firmware must be downloaded to your RCX so that the RCX can understand your programs.
- Only required to be loaded
  - When the RCX is new,
  - Has lost it's firmware for some reason (such as changing batteries too slowly),
  - Or RCX starts behaving badly.



# Computer Programs

---

- Model a real or mental process
- Intricate in detail
- Only partially understood
- Rarely modeled to our satisfaction
- Thus, **programs continually evolve**
  
- The computer is a harsh taskmaster, its programs must be correct and things must be accurate in every detail.

# Writing a Computer Program

---

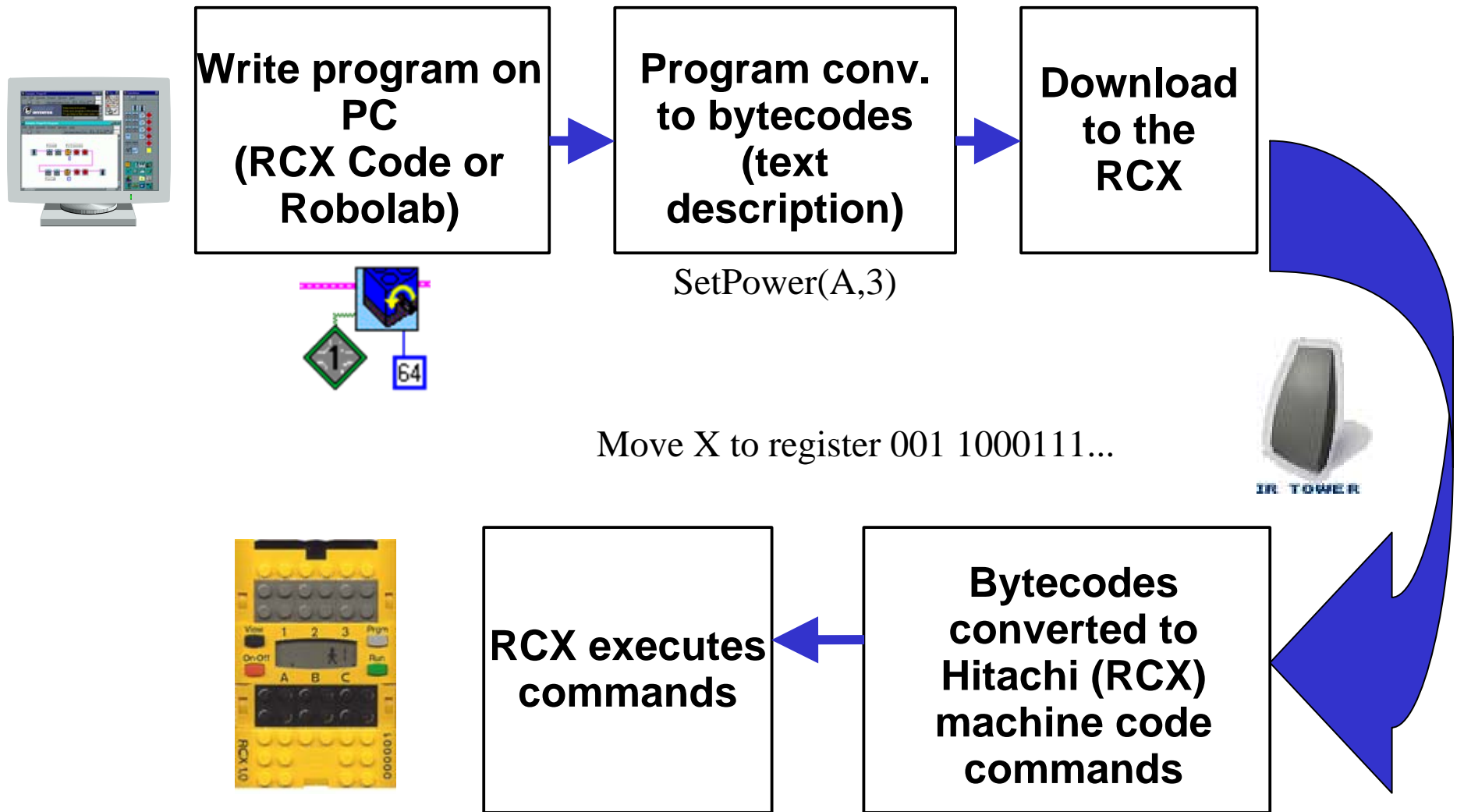
- Specify the task
  - Inputs to be supplied
  - Outputs to be produced
- Devise an algorithm
- Express that algorithm in a computer language

# Language Choices

---

- In FLL, you have two choices for computer language
  - RCX Code (also called RIS - Robotics Invention System).
    - Comes with the commercial version of LEGO Mindstorms
    - Only runs on a PC
    - Use Version 2.0 if at all possible
  - ROBOLAB
    - Runs on MAC or PC.
    - Built on a commercial engineering program called LabVIEW.
  - At the FLL level, there is no advantage to either one (we've been tracking this for 3 years). RCX Code is easier to learn, while ROBOLAB has more growth potential.
- For FLL at the High School level, no language restrictions.

# Running a computer program (RCX)



# Tips and Tricks (1)

---

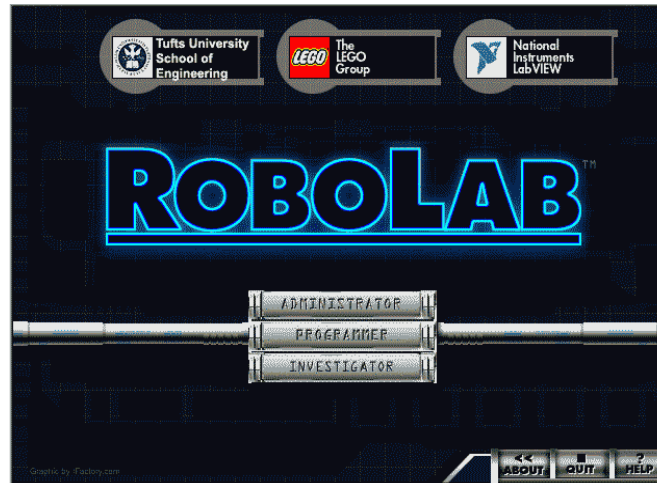
- The RCX has 5 program slots
  - Slots 1 and 2 are locked. Unlocked in Administration.
  - Later we'll show how to use a touch sensor to get you more slots (virtually).
- The RCX automatically powers down.
- Shielding IR port
  - The IR tower has a significant range. Use short range setting.
  - Shield your RCX IR Port with LEGO bricks.

# Tips and Tricks (2)

---

- Direction of connecting wires
  - You can change motor direction by turning the connection to the RCX 180°.
- Batteries
  - Change one at a time to reduce chance of losing firmware.
  - Alkaline rechargeable batteries work.
  - Use a power plug if you can on your RCX.
  - Source of batteries. If you buy alkalines, buy them in bulk from a store like BatteriesPlus or Digikey ([www.digikey.com](http://www.digikey.com)).
  - Avoid stalling the motors, it drains batteries.

# The Programming Environment

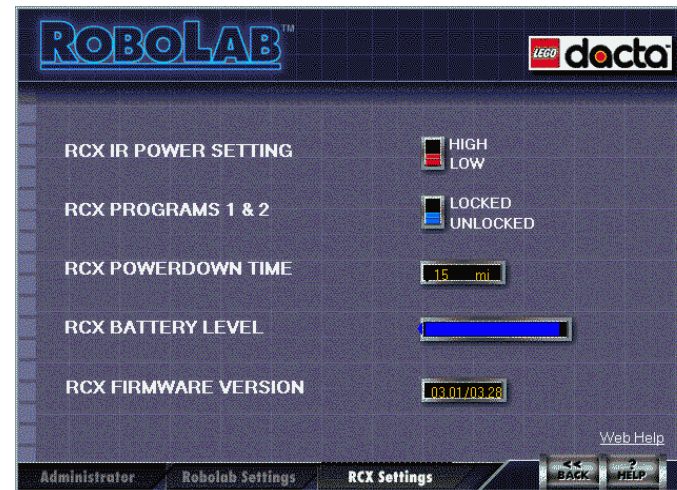


# Administrator Settings

## Administrator



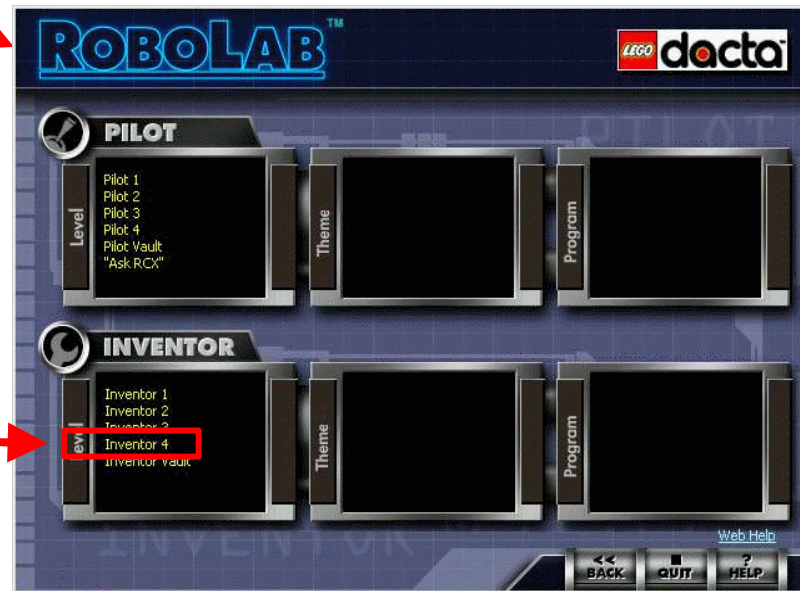
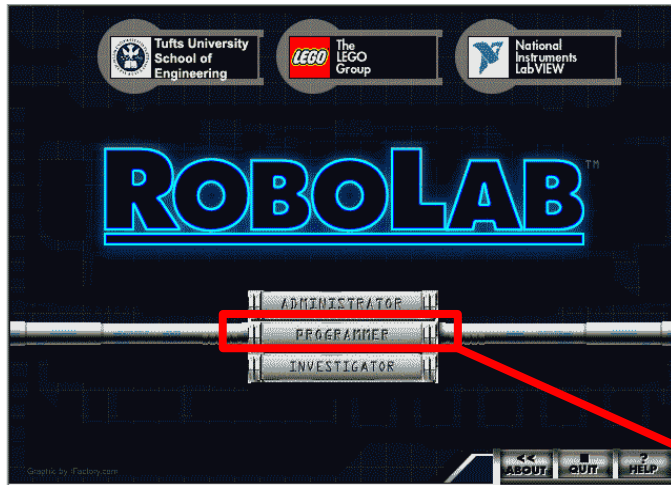
## RCX Settings



**The “RCX Settings” menu options should be set as shown.**

- |                      |                                |   |
|----------------------|--------------------------------|---|
| RCX IR Power Setting | <b>Low</b>                     |   |
| RCX Programs 1 & 2   | <b>Unlocked</b>                |   |
| RCX Powerdown Time   | 15 minutes                     | <i>(5 min. to save batteries, 30 min. to lower aggravation)</i> |
| RCX Battery Level    | <i>should be about 9 volts</i> |   |
| RCX Firmware Version | 03.01/03.09 (RoboLab 2.0)      | 03.01/03.28 (RoboLab 2.5)                                       |

# The Programming Environment



Double click  
Inventor4

# RoboLab Work Space

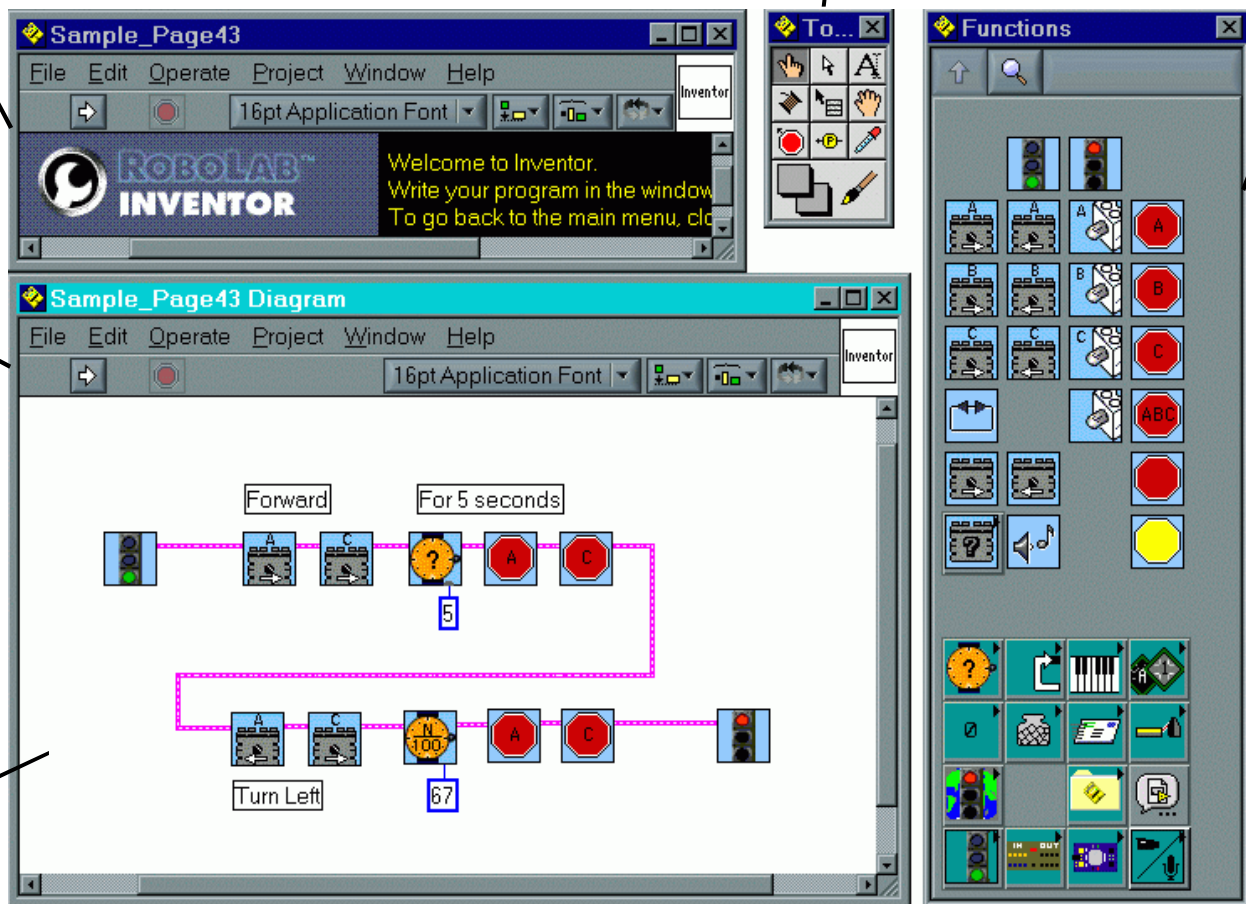
Panel Window

Diagram Window

Program goes here

Toolbox

Functions Window

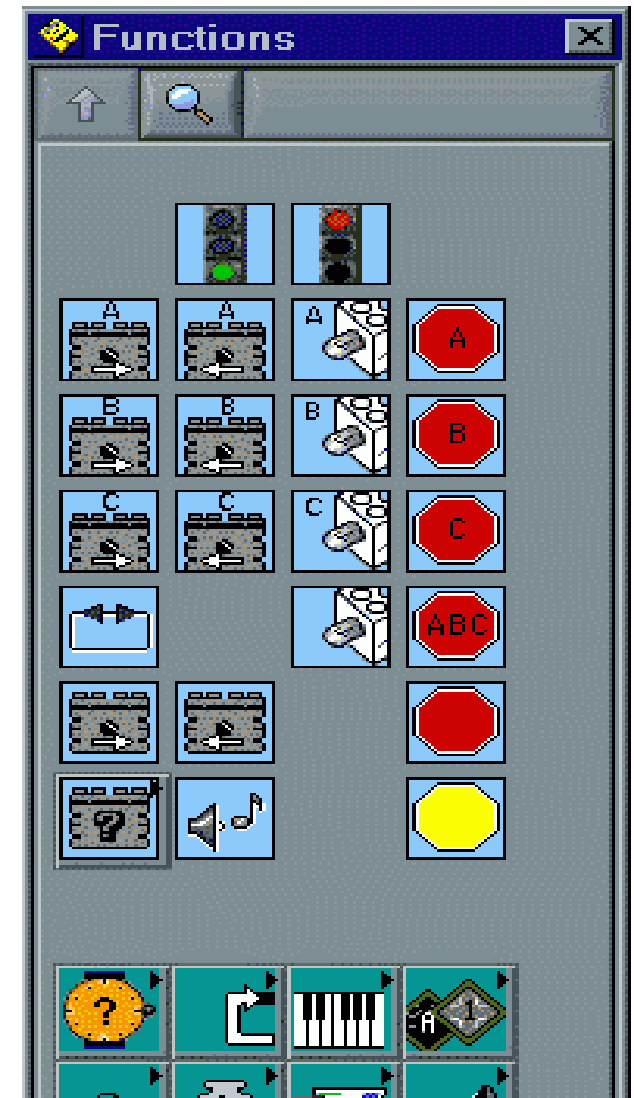


**Remember to use Context Sensitive Help!**

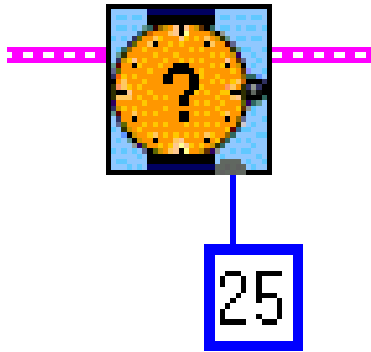
# Simple Commands

# Simple Commands

- Simple commands are basic actions
  - Like English statements
    - Turn motor on
    - Stop motor
    - Reverse motor direction
- Setting parameter values
  - Many commands have modifiers
    - Motor power
    - Time

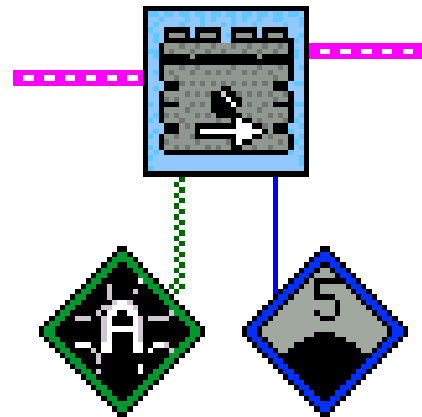


# Simple Commands with Modifiers



Wait for **25** Seconds

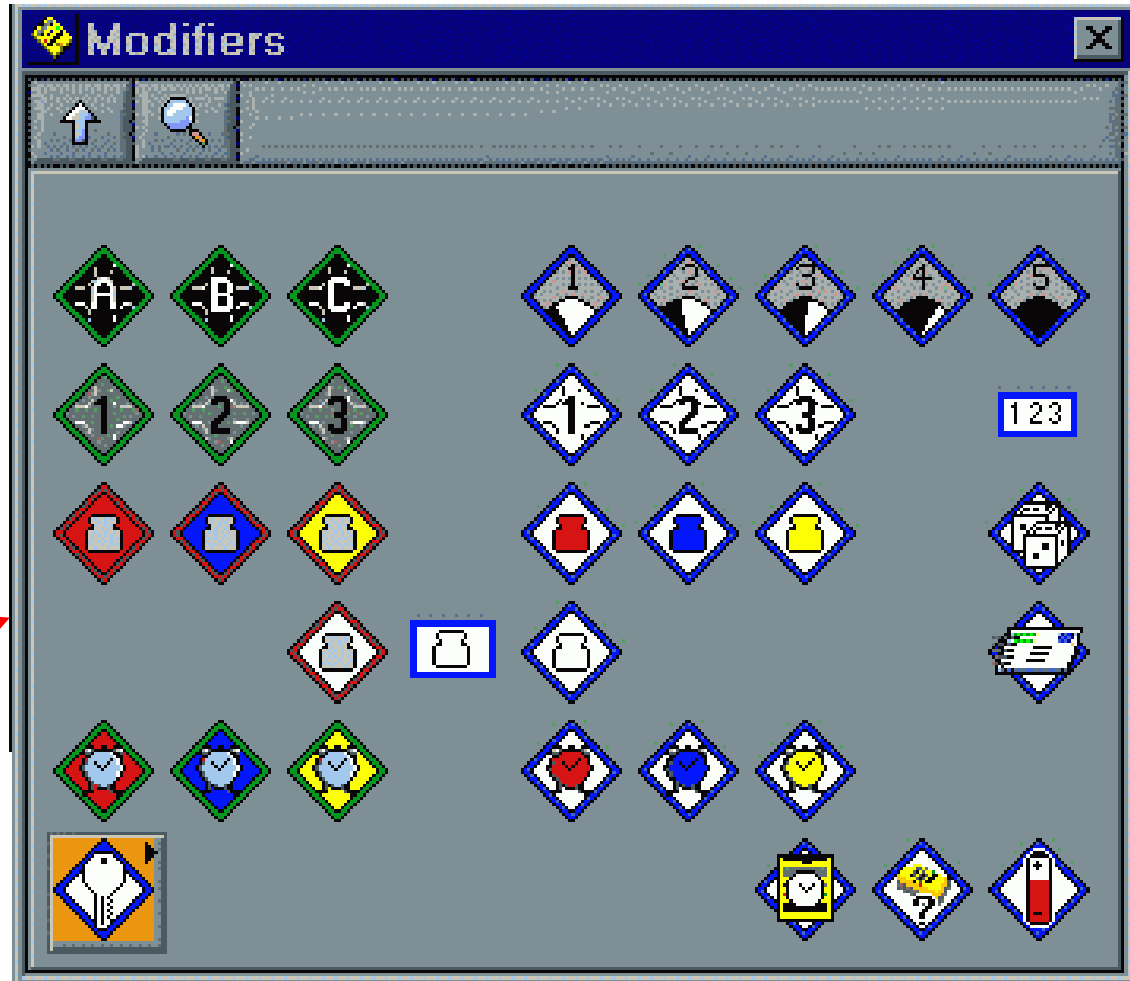
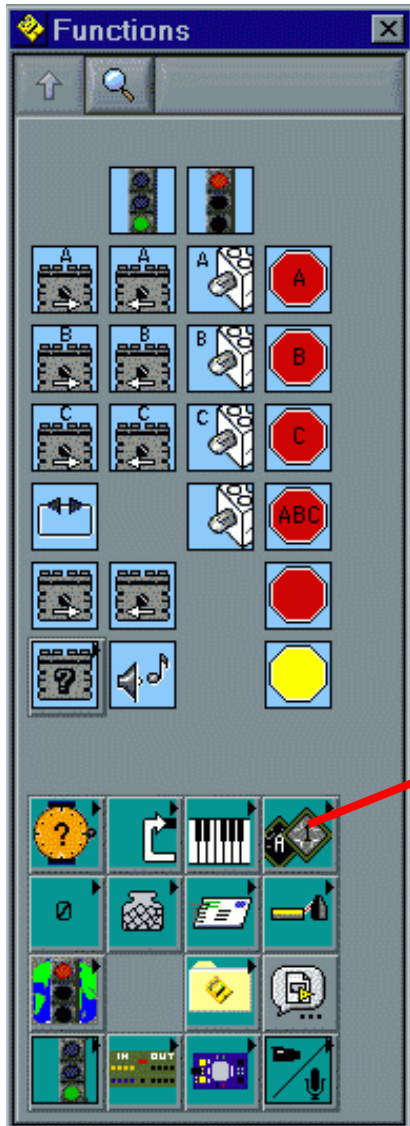
Wiring Tool



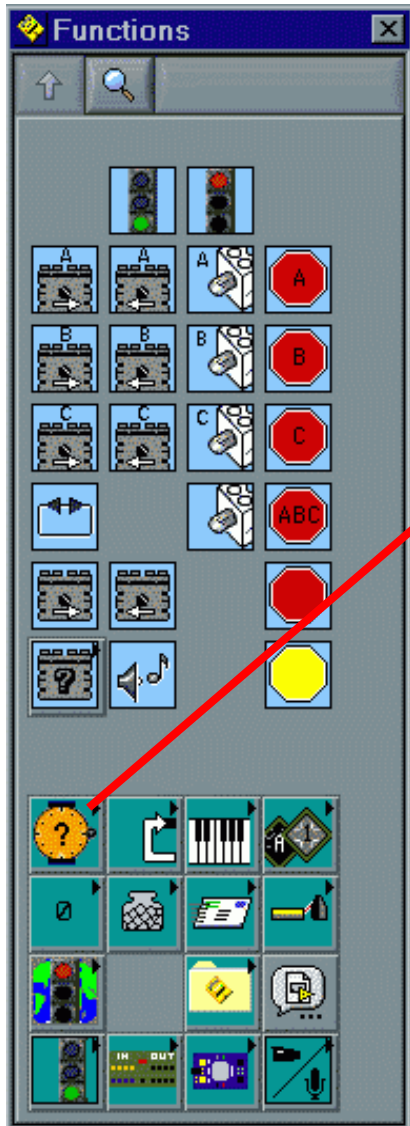
Forward Motor **A**  
Power Level **5**

- Select Command
- Click in Diagram Window to Place Command
- Wire Command Connections

# Modifiers Palette



# WaitFor Palette

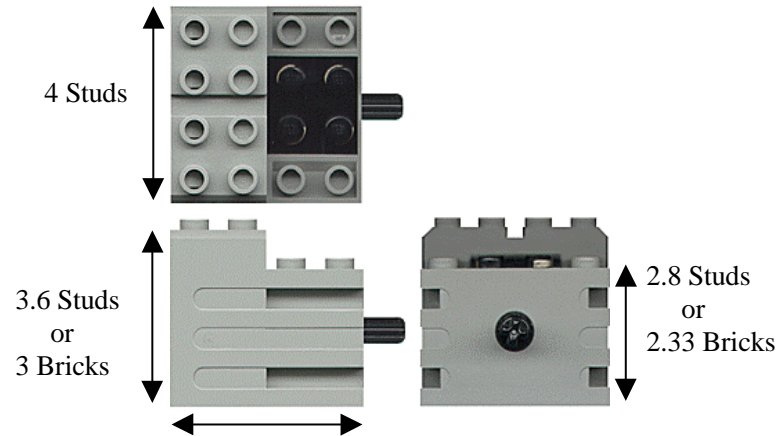


# Motors

---

- Making the motors turn is the **output** of your program. It makes your creation a robot!
- Formally called the 9 volt geared motor
  - Without load, motor shaft turns at about 350 rpm.
  - With a typical robot, 3-4 hours on a set of batteries.
- FLL allows up to 3 motors.

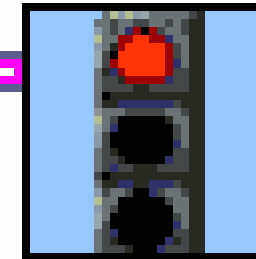
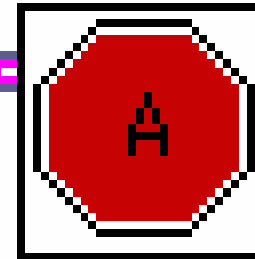
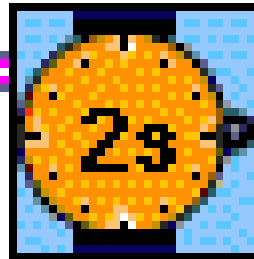
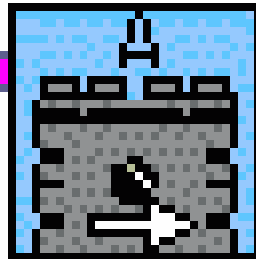
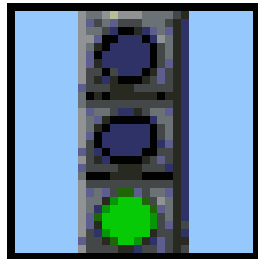
# Motor Details



- Motor can be set to different power settings
  - 5 settings in ROBOLAB (1/8, 2/8, 4/8, 6/8, 8/8)
  - 8 settings in RCX code
  - Changing power settings is usually a poor substitute for gearing.
- Turning the power setting up higher essentially makes the shaft turn faster.

# Using Motor A

---



Turn On  
Motor A -  
Forward  
Full Power

Wait for 2  
seconds

Stop Motor A

**Motors run until stopped!**

# Lab One

## Task:

Move forward for 5 seconds and return

Then try:

Move forward for 5 seconds, turn right  $90^{\circ}$

# Problem Solving

# Generic Problem Solving Process

---

- Define the problem
- Brainstorm solutions
- Evaluate solutions Pick one
- Try (implement) best solution
- Evaluate results
  
- Express the solution as an algorithm, then convert it into a computer program.

# What's an Algorithm?

---

- An algorithm (pronounced AL-go-rith-um) is a procedure for solving a problem. The word derives from Mohammed ibn-Musa **Al-Khowarizmi**, a mathematician of the royal court in Baghdad. He lived from about 780 to 850. Al-Khowarizmi's work is the likely source for the word **algebra** as well.
- A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small **procedure that solves a recurrent problem.**

# Example Algorithm

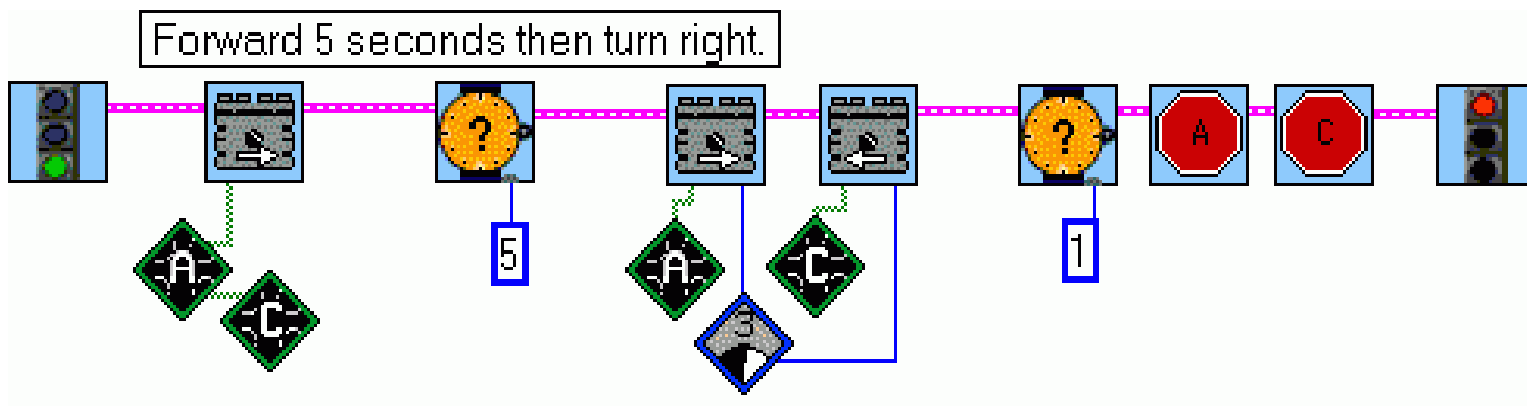
---

**Make robot go forward 5 seconds and then turn right 90°**

- Set direction and power of motors
- Turn motors on and start timing
- Wait 5 seconds,
- Stop motors, turn right.
  - Turn right by reversing right-side motor
  - Turn motors on for ? Seconds.
- Stop all motors

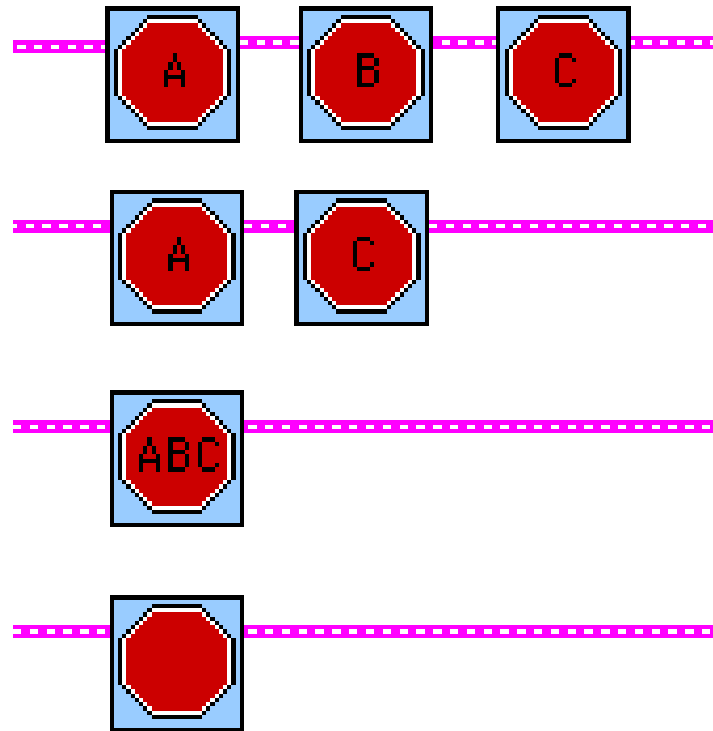
# Conversion to a Program

- Motors A and C forward
- Wait 5 seconds
- Motor A forward, C backward  
Power Level 3
- Wait ? Seconds
- Stop Motors A and C



# Optimizing Code

- Which is faster?  
more reliable?  
best?



- Use the one that makes sense to you, the programmer.

# Debugging and Analysis

---

- Literally walk through it
- Ask lots of questions
  - What ifs
- Do little pieces at a time
  - For example, get the robot to where it needs to be first, then work on getting it to do something
- Reuse pieces that work
  - For example, you know how to turn 90°
- Feel confident in your algorithm before starting to code it.

# Common Situations

---

- Not sure which solution is better
  - Try them both, or at least the primary element of each
  - Which is easiest for your team to do?
- Can't think of all the steps needed for the algorithm
  - Get out your robot, and walk through it.
  - Program and test the steps you understand.
- Give programs descriptive names
  - ClearSoccerField *not* Csf\_amy\_3a

# Keep It Simple Strategies

KISS #1: Subroutines  
#2: Comments

# KISS #1: Subroutines

---

- Wrap a complicated process into a neat and tidy package.
- Once wrapped, just worry about the package.
  
- a.k.a. Black Box, function, macro, SubVI
  - So simple . . . So powerful.



# Subroutines: When to Use

---

- To do the same thing from different places.
  - Turning Left
  - Move arm up
- To divide a task into meaningful pieces.
  - Modules
- To hide complex details.
- Real world example: Clock
  - Complicated parts are hidden inside.
  - User tasks are divided into meaningful pieces:  
Time, Set Time, Set Alarm, Turn alarm on/off

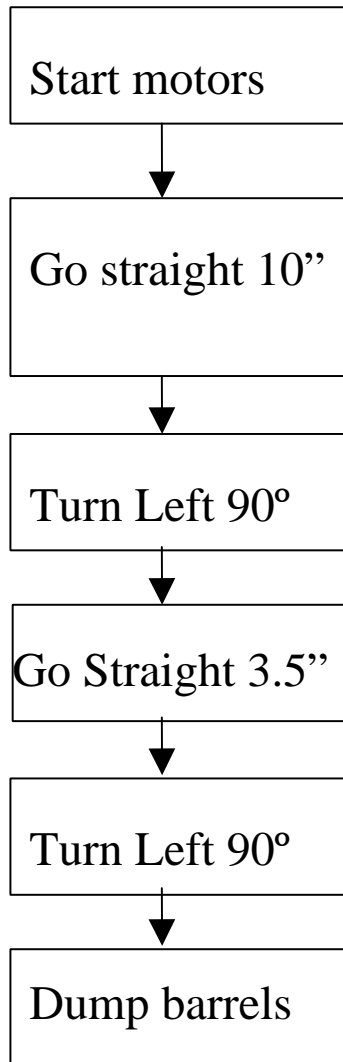
# Subroutine Use

---

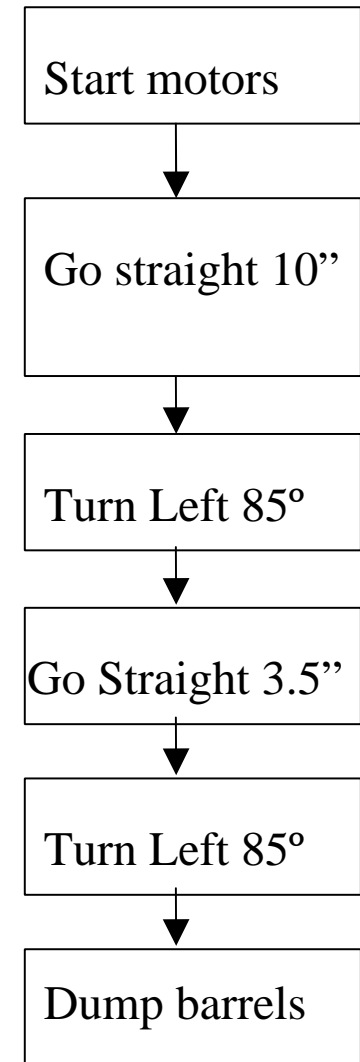
- If you have a motion or action you can do reliably, and will use a lot, make it a subroutine.
- It's simpler to see one "Turn left 90°" command than decipher a long set of commands that do the same thing.
- A subroutine is a **module**. It holds the code for turning.
  - See next slide for example

# Modular Advantages

## Algorithm - Mission 1



## Algorithm Updated- Mission 1

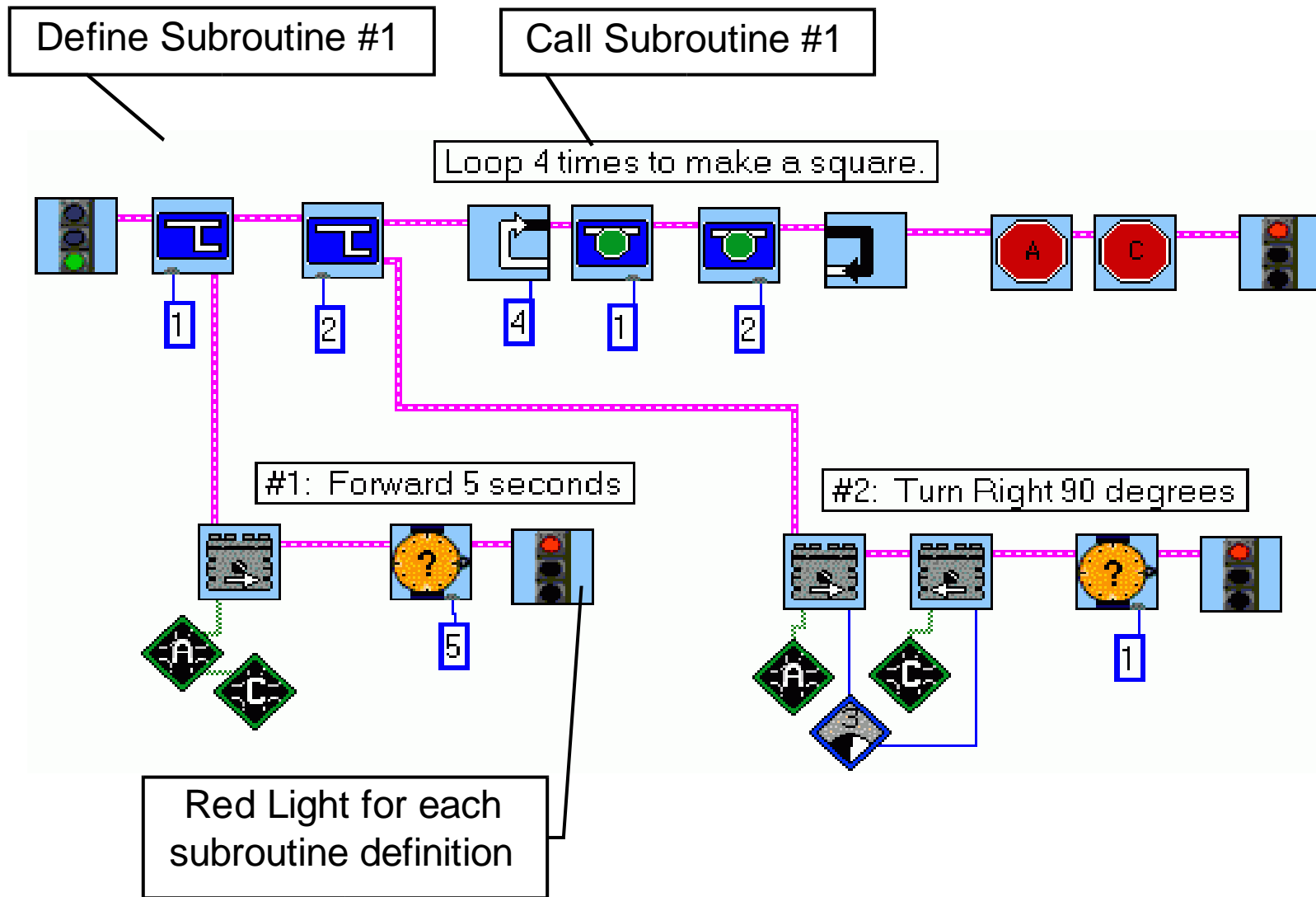


If in the process of testing your program, you realize it is an 85° turn, you only have to change your program in one place, in your subroutine.

Subroutine

Turn Left ~~90°~~ 85°

# Subroutine Commands (local)



# Subroutine Names

---

- Useful and informative names.
- Suggest using “action + to + target”:
  - Fwd2Wall or ForwardToWall or Forward\_To\_Wall
  - Fwd2Line, etc.
  - FwdDist
  - TurnRight
- Name the task accomplished, not how it was done.
  - FollowLine *not FollowLine1LightSensor*



# Subroutines: SubVI

---

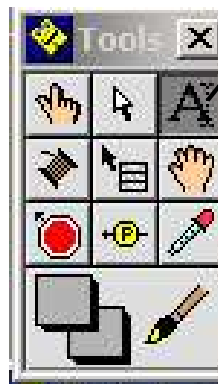
Demo

## KISS #2: Comments

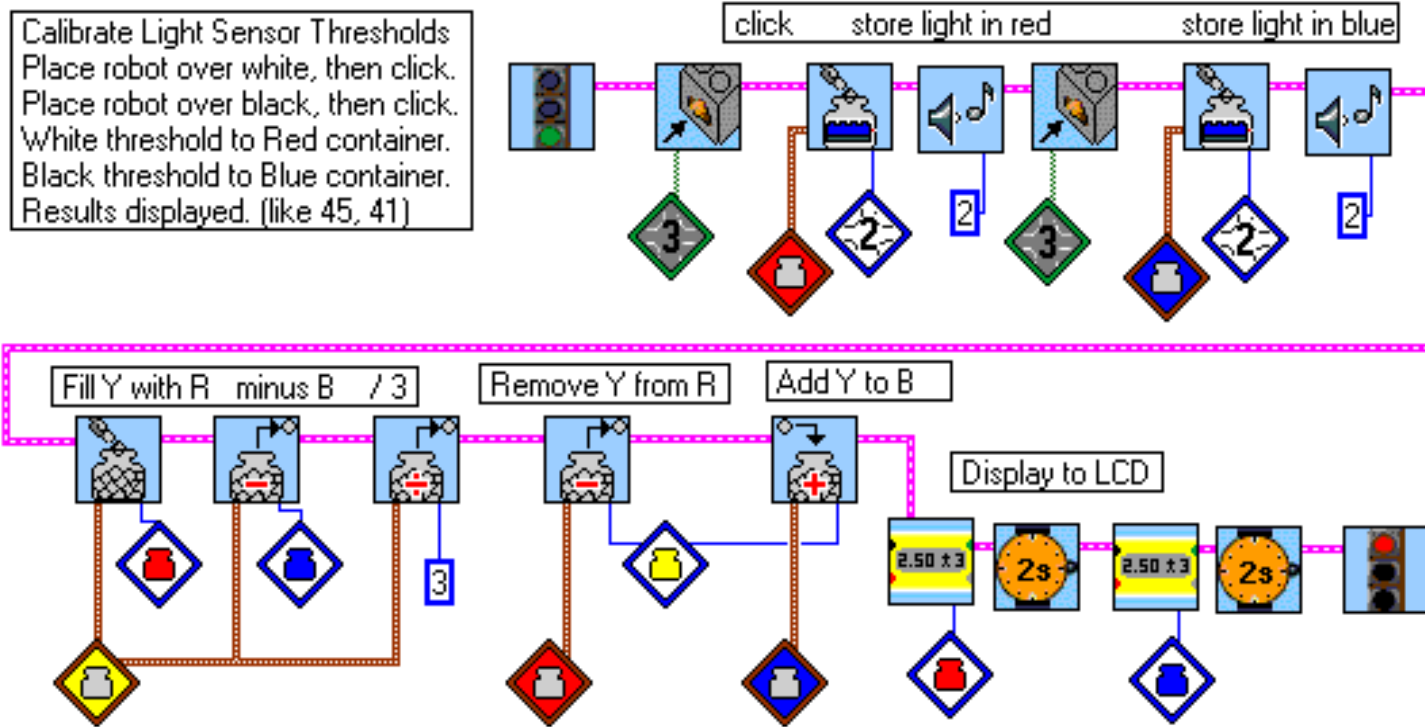
---

- Comments explain the program to other programmers.
- **Very** important. Programmers forget.
- In a team process like FLL, comments are especially important as more than one person will be working on the program.

Enter comments with  
the text tool.



# Comment Use



- Add things like who made changes, when, how to use, assumptions, expected results, etc.
- Use color to highlight things

## Lab Two

### Task:

Make the program from Lab One  
(Move forward 5 seconds and turn right 90 degrees)  
into a subroutine

# Data Input

## *Sensors*

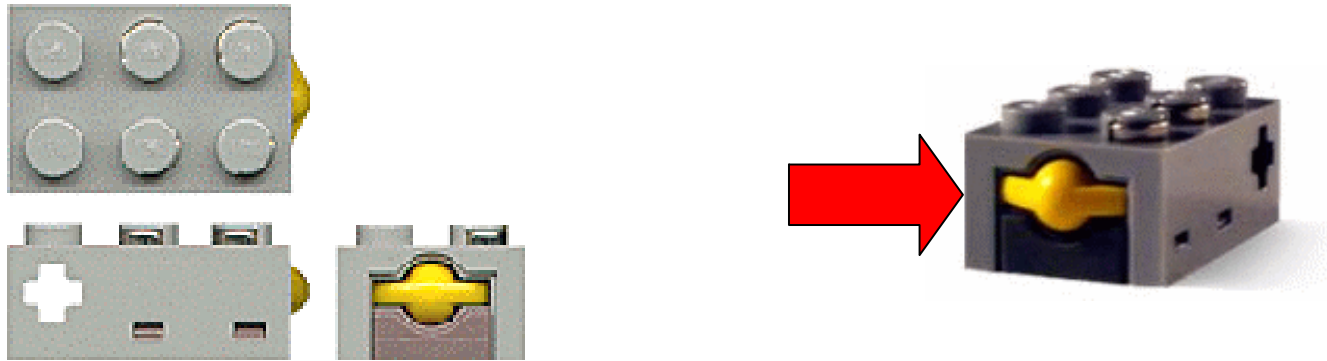
# Sensors

---

- Sensors allow your robot to detect the real world.
  - Touch
    - Has your robot made contact with something?
  - Light
    - Is the surface light or dark?
  - Rotation
    - How many times has an axle turned?
  - Timer
    - Internal sensor, keeps track of time
  - Battery Voltage

# Sensor #1: Touch

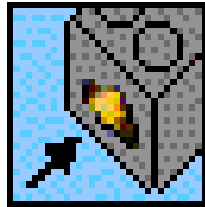
---



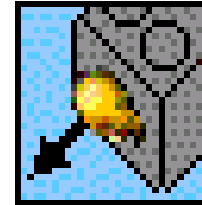
- To detect touching or bumping into something.
- Good for detecting robot arm movements.  
The sensor activates when the arm moves far enough to push in the touch sensor.

# Touch Sensor WaitFor Commands

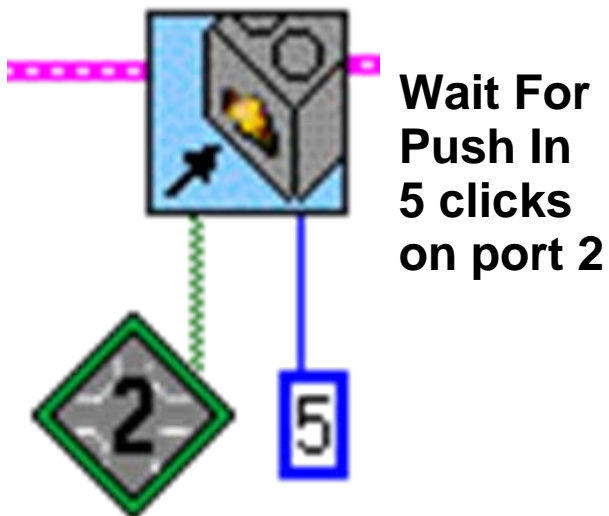
---



**Wait For Push In**  
(default: 1 click on port 1)

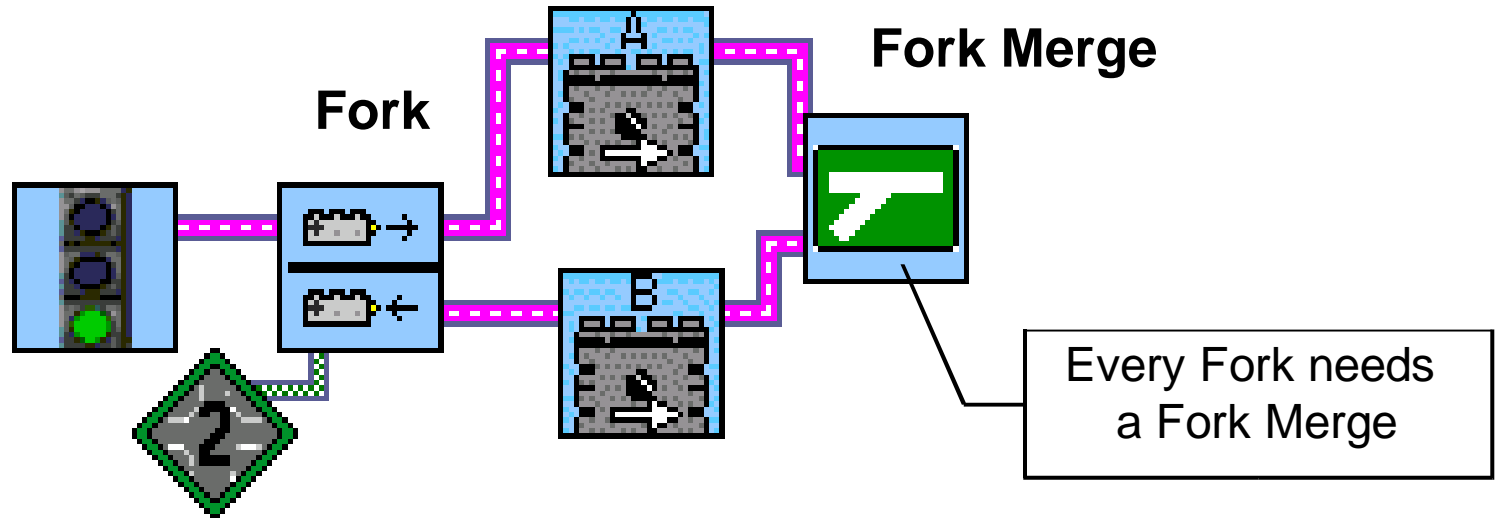


**Wait For Let Go**  
(default: port 1)



# Touch Sensor Fork Commands

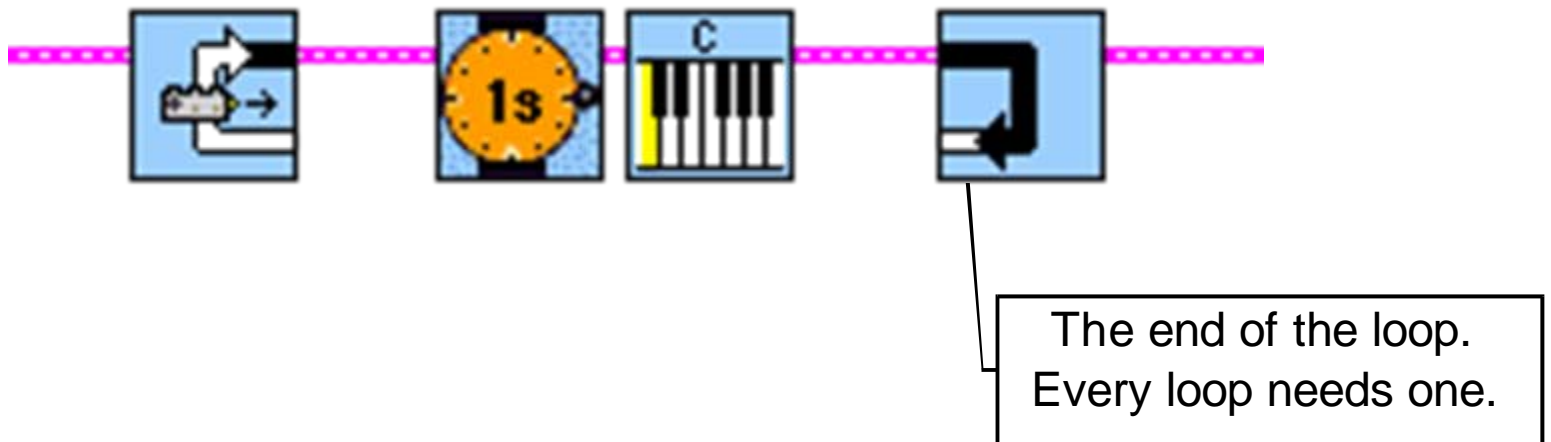
- Waiting for a touch sensor to be pushed or released can be useful, but many times you want to do different things based on the value (is it pushed or released?)



- If touch sensor released, turn on motor A
- If touch sensor pushed in, turn on motor B

# Touch Sensor Loop Commands

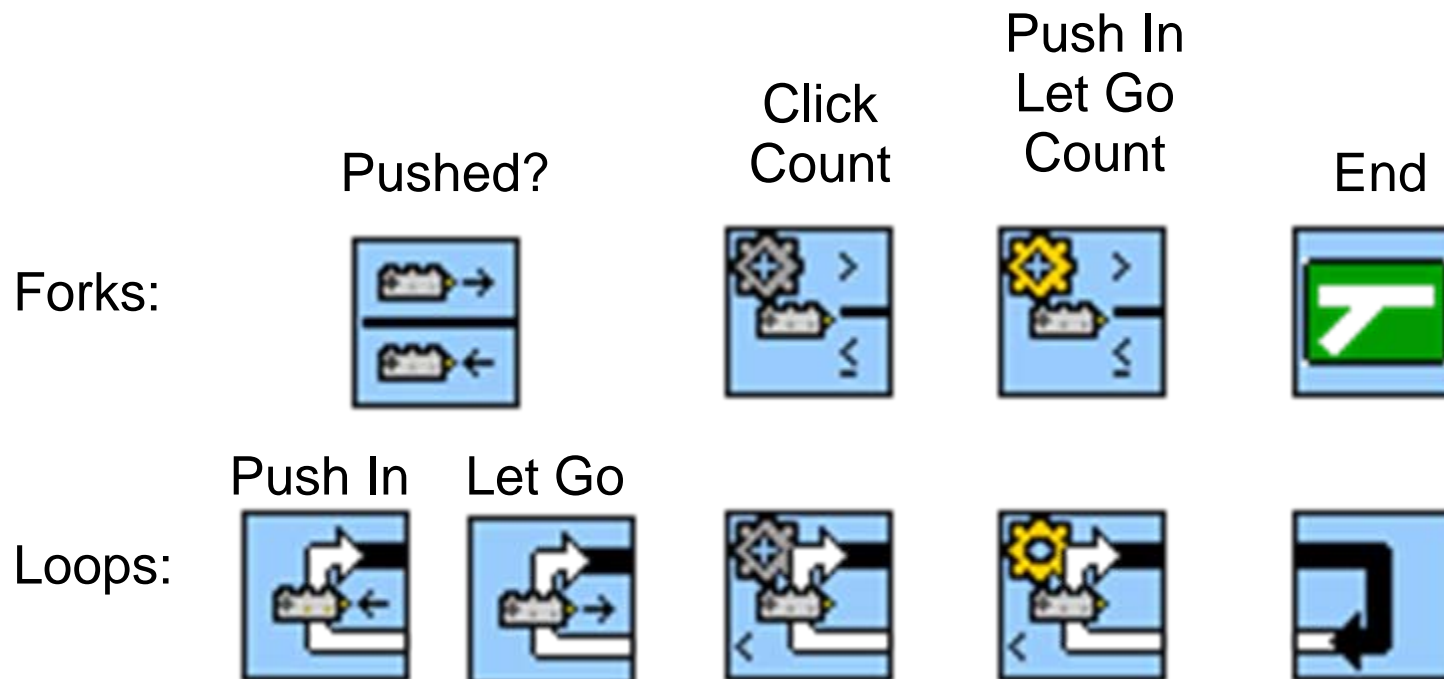
- Loop until a touch sensor is released. Useful if the loop contains commands that must be repeated.
  - For instance, a routine that beeps until a bumper hits something.



# Push In, Clicks and Let Go

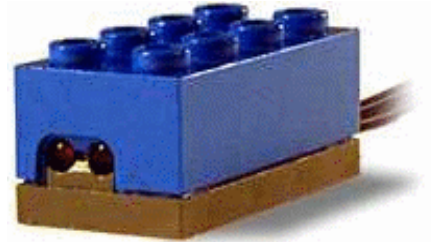
Two additional functions involving the Touch Sensor:

- Click Counter: a counter of PushIn/LetGo cycles.
- Touch and Release Counter: count changes (twice as big as the click count).



## Sensor #2: Light

---



- A light sensor shines a red light and detects light. It is most sensitive to red light.
- Light sensors operate in "percent" mode, anywhere from 0 to 100 in value.
  - Higher numbers mean more light. A lighter surface reflects more light.
- Light sensor is useful and frustrating.

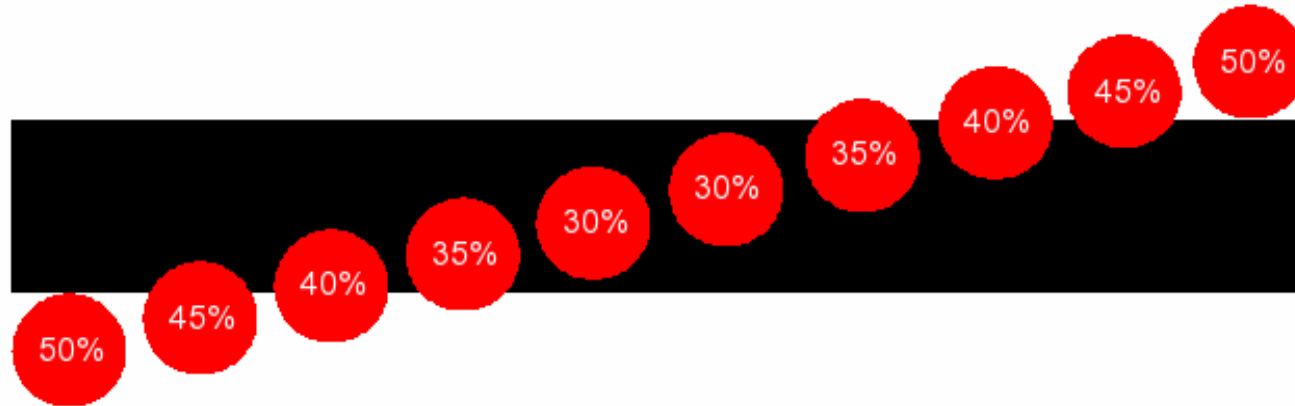
# Light Sensor Readings

---

- Lowest likely reading 20% (in very dark room)
- Highest likely reading 100% (pointing at a bright light)
- Normally between 30-60%
- Readings also depend on the color of the surface
  - See **“Building LEGO Robots for FIRST LEGO League”** by Hystad for extensive discussion of this.
  - **Results vary. You should experiment with your surfaces/colors.**
- Light sensor is sensitive to the distance between the sensor and the reflecting surface. Variations can make the readings unusable. Keep the sensor close to the surface and shielded.

# Light Sensor Readings

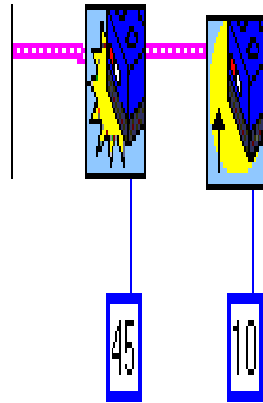
---



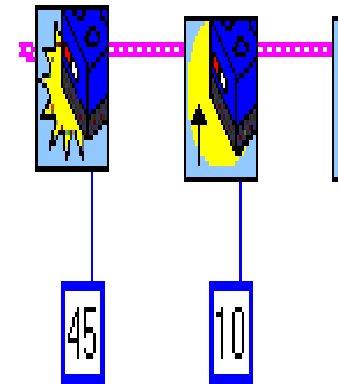
- The light sensor averages its readings over roughly a circular area. Don't drive too fast or you will get inaccurate readings.
- Shield the sensor from ambient light.
- Try various conditions. Test it on competition day.

# Light Sensor WaitFor Commands

---



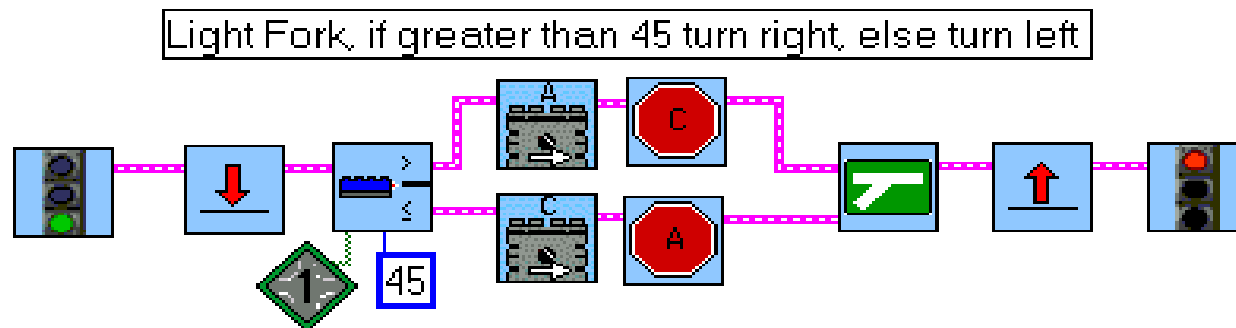
**Wait for Light  
(greater than 45%)**



**Wait for Brighter  
(increase by 10%)**

**Also Wait for Dark  
and Wait for Darker  
commands**

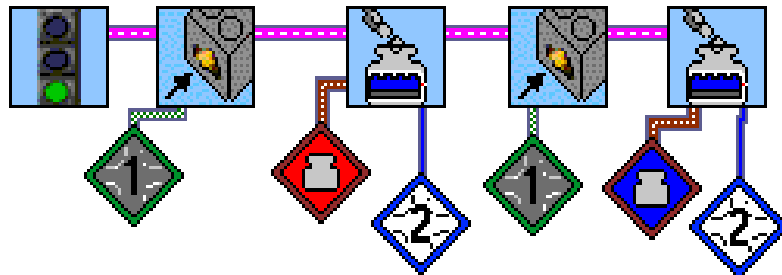
# Light Sensor Fork Command



Follows the thick black oval line of the Mindstorm Test Pad

- If light sensor greater than 45, turn motor A on in reverse direction. Stop C.
- If light sensor less than or equal to 45, turn motor C on in reverse direction. Stop A.
- The light sensor fork is very useful.

# Calibrate Light Sensor



Position light sensor over white area and push touch sensor.

Set container **Red** to light sensor value.

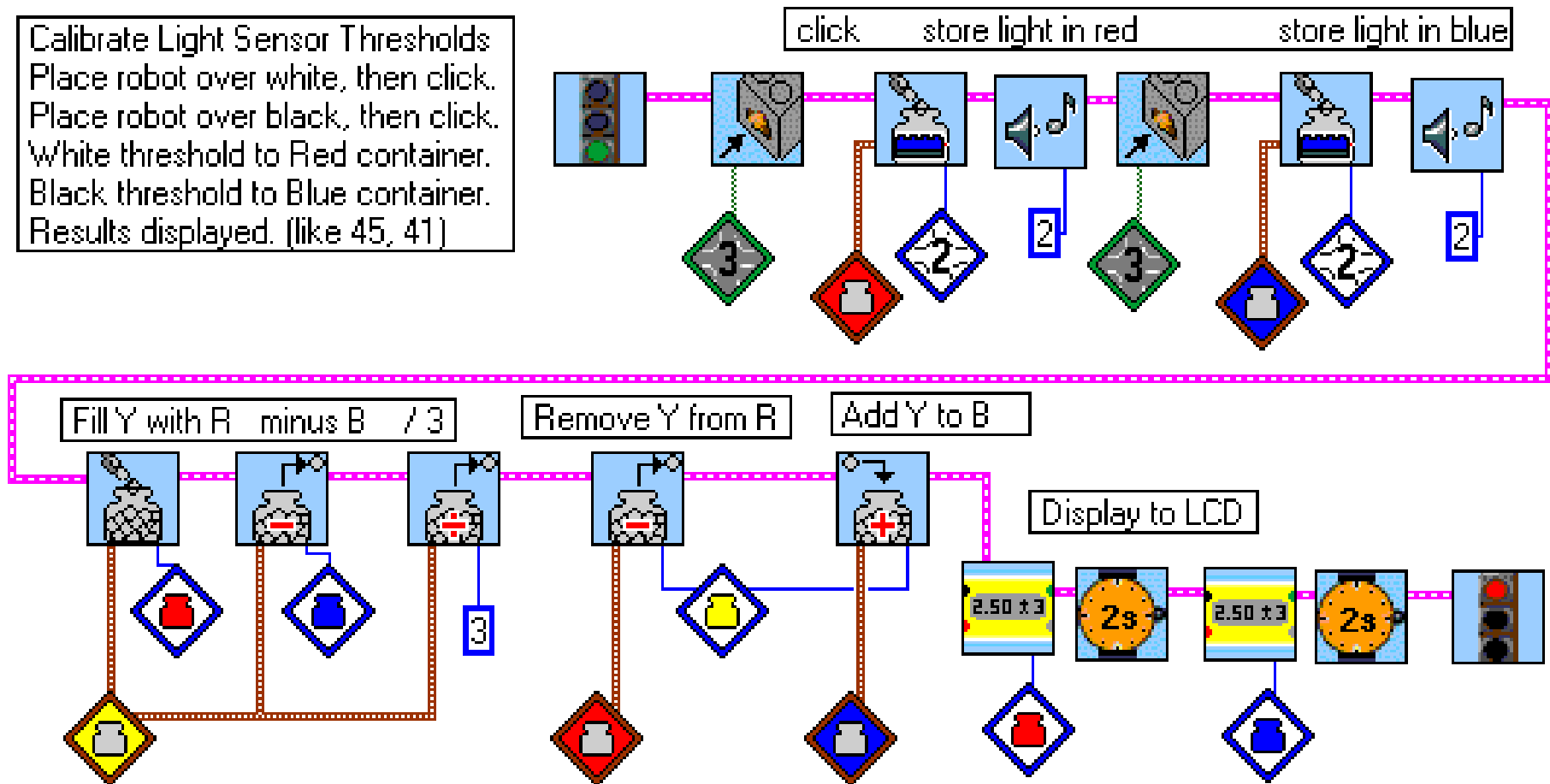
Position light sensor over dark area and push touch sensor.

Set container **Blue** to light sensor value.

- This program assumes you move the robot over light and dark areas, and then use a touch sensor to trigger a reading.
- The containers Red and Blue now contain the calibrated values of light and dark. These containers can then be used as command modifiers.

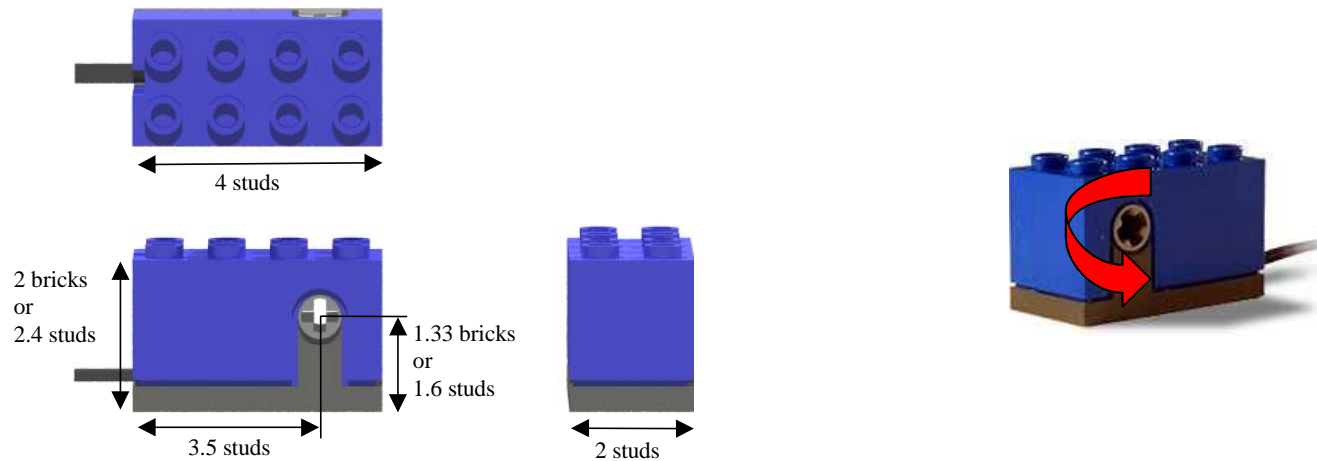
# Refining the Calibration for Edge Following

Calibrate Light Sensor Thresholds  
 Place robot over white, then click.  
 Place robot over black, then click.  
 White threshold to Red container.  
 Black threshold to Blue container.  
 Results displayed. (like 45, 41)



The previous example set thresholds for white and black.  
 This example sets them at light grey and dark grey.

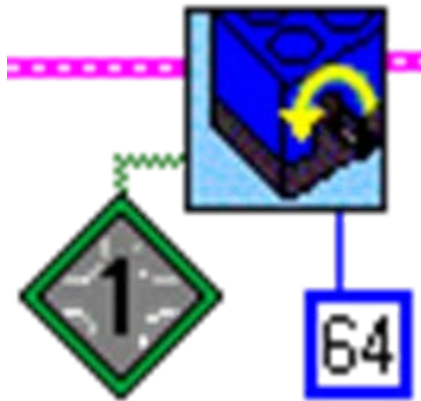
# Sensor #3: Rotation



- Measures how far a rotating axle has turned. As the axle turns, a counter in the RCX is incremented or decremented.
- 16 counts per rotation gives the sensor a resolution of 22.5 degrees ( $360/16$ ). It is sometimes called the angle sensor.

# Rotation Sensor WaitFor Commands

---

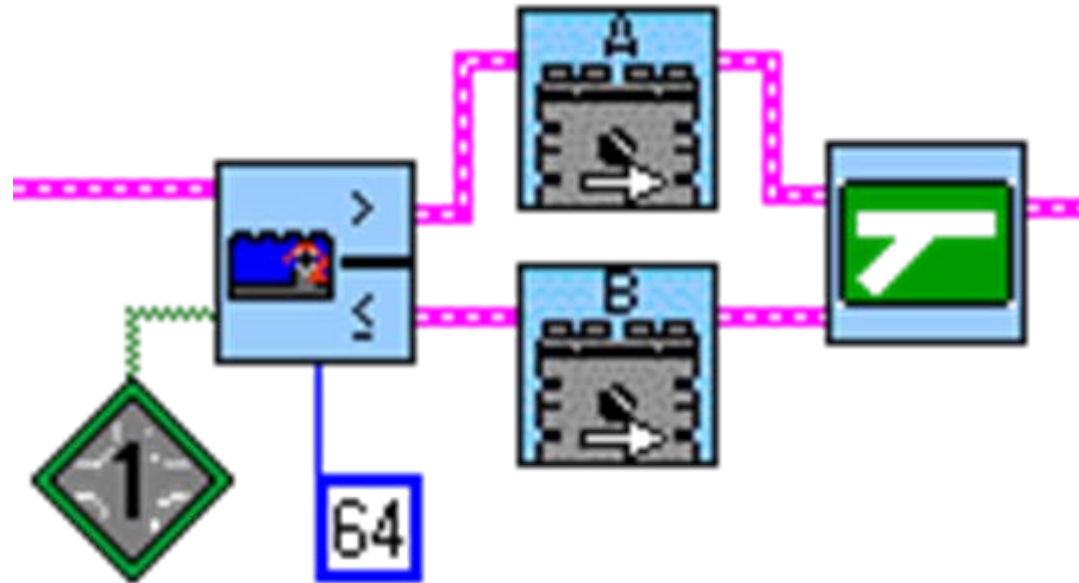


**Wait For Rotation**  
counter greater than 64  
or 4 (64/16) rotations.



**Wait for Rotation Absolute**  
“Absolute” means it is **not**  
**reset to zero** before starting.

# Rotation Sensor Fork Command



- If rotation sensor greater than 64 (4 complete axle rotations), turn motor A on in forward direction
- If rotation sensor less than or equal to 64 , turn motor B on in forward direction

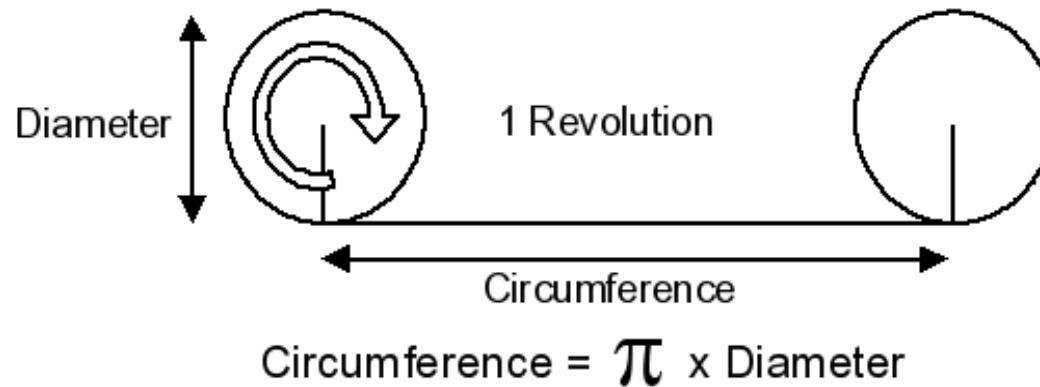
# Using the Rotation Sensor

---

- Use as an odometer (how far have you gone) or as an angle sensor (where does the arm point)
- Turn motors on to go straight
  - Wait until eight complete rotations
  - How far is eight rotations? See next slide.
- Because of momentum, a robot travels beyond the rotation sensor's stopping point. Read the sensor again to find out where the robot stopped.
- Measure rotations: Load a program that floats the motors and uses any rotation sensor function. Run. Push the robot. The LCD shows the rotation value.

# Calculating Distance

- The rotation sensor also brings in the possibility of doing some real math!



- We'll leave that as an exercise for the reader!
- Of course, trial and error also works.
- Sources of error in calculation - dirt on surface, using a skid rather than a wheel, backlash (poor fitting gears).

# More on Rotation Sensor

---

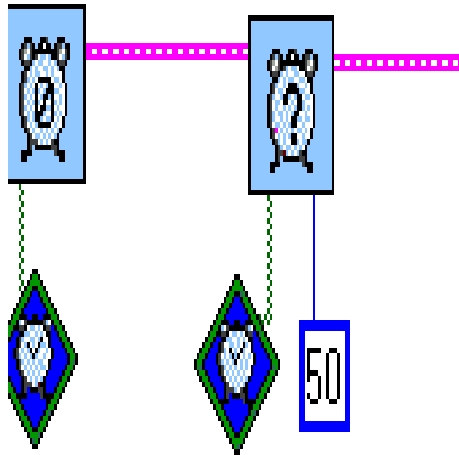
- Rotation sensor **counts forward and backwards.**
  - Up to 32767 and down to -32768
  - Then it rolls over to largest negative from largest positive (or vice versa), however this is unlikely to occur in FLL.
- Increase sensor resolution using gears.
  - Rotation sensor is **reliable in 60-1000 rpm range.** Problems may occur at high or low rpm.
  - Motors typically spin about 200-350 rpm.

# Debugging and Analysis

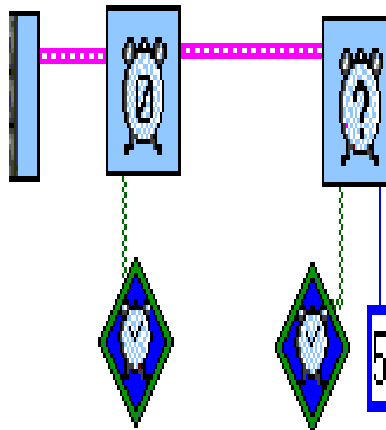
---

- Common problems
  - Programming: **reset the sensor to zero** before use.
  - Design: inadequate sensor resolution (trying to measure something very accurately without using gear reduction).
  - Control: accelerating and turning **too fast**.
  - Variations in the initial conditions: not putting everything in the right place, or at least the same place, before pushing the run button.

# Sensor #4: Timer



- Wait until Blue Timer reaches 5 seconds
  - Time in tenths of a second
  - Blue timer must be zeroed before it's used
  - ROBO LAB has 3 timers, blue, red, yellow



- Zero Blue Timer (reset)

# Tricks and Tips

---

- Use a touch sensor to get more program slots
  - Pushed In: execute one fork branch.
  - Not Pushed In: execute the other fork branch.
  - Do this for every program slot, and you end up with 10 “programs” using 5 slots.
  - Between missions, add a brick to the robot to push the touch sensor.
- Use a light sensor to achieve rotation counting
  - Aim light sensor at a piece of your robot that rotates and count light level changes.
  - Clockwise or Counterclockwise? Does it matter?
  - Touch sensors can also mimic rotation sensors.

# Stacking Sensor Ports

- Attach multiple sensors to the same port.
- Use more than 3 sensors.
- Rotation sensors can **not** be stacked.

Stacked Sensor values			Sensor 1			
			Touch		Light	
			Out	In	White	Black
Sensor 2	Touch	Out	0	1	~50	~35
		In	1	1	100	100
	Light	White	~50	100	~75	~70
		Black	~35	100	~66	~60

# Lab Three

## Task:

Move forward for 50 rotations, turn right  $90^\circ$   
and/or

Use the rotation sensor to move along a 2' x 2' square path

# Keep is Simple Strategies

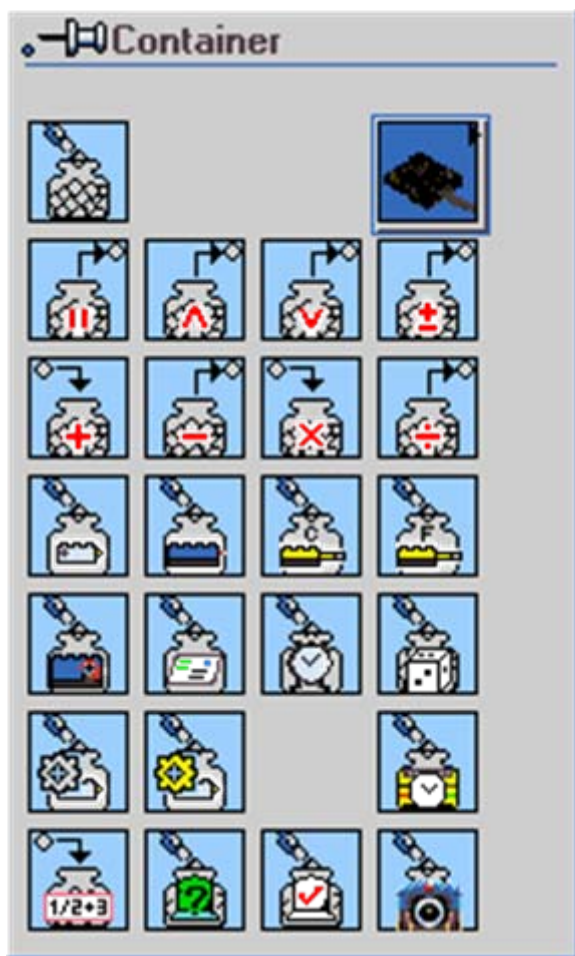
KISS #3: Variables  
#4: Split Task  
#5: Loops

# What's a Variable?

---

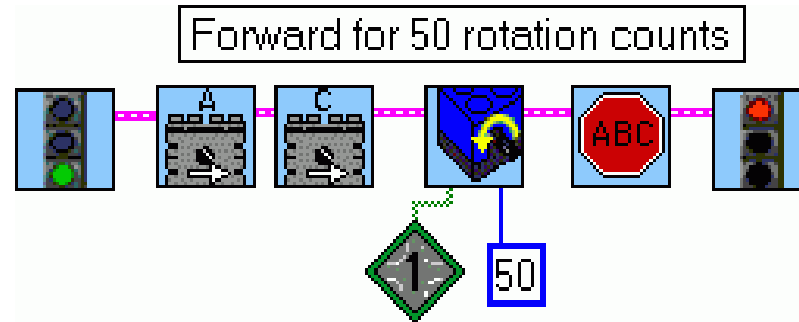
- A value that you can change during your program.
  - This value is “variable”, hence the name.
- For example, the value of a light sensor may be measured during the program and placed in a variable called *RED*.
- Think of a variable as a container containing the value of something.
  - In ROBOLAB, variables are called **containers**.
- Useful to pass values between tasks and SubVIs.
  - For example go straight *N* seconds where *N* is a variable.

# KISS #3: Containers

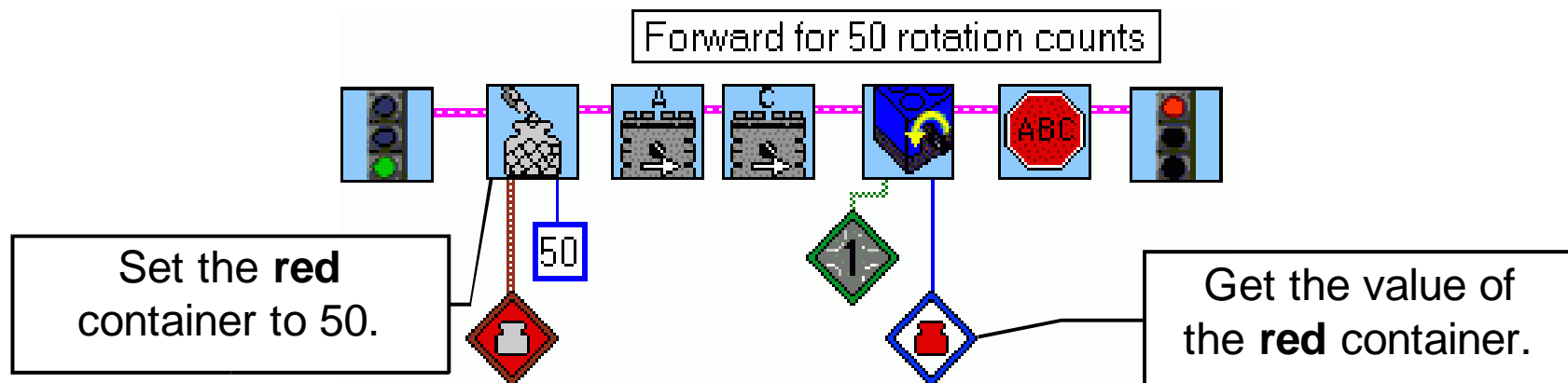


- Fill a Container
  - On the Functions pallet, the container menu has many choices. All of these fill a container with a value.
  - There are 21 containers. 3 are predefined containers: Red, Blue, and Yellow. Use a modifier to identify the rest (numbers 3-20).

# Use of a Container



- Instead of “hardcoding” 50 (above), fill a container with “50”, then get the container (below).
- Once set, containers hold their values (unless the firmware fails.) Fill in Program #1, get in Program #2.

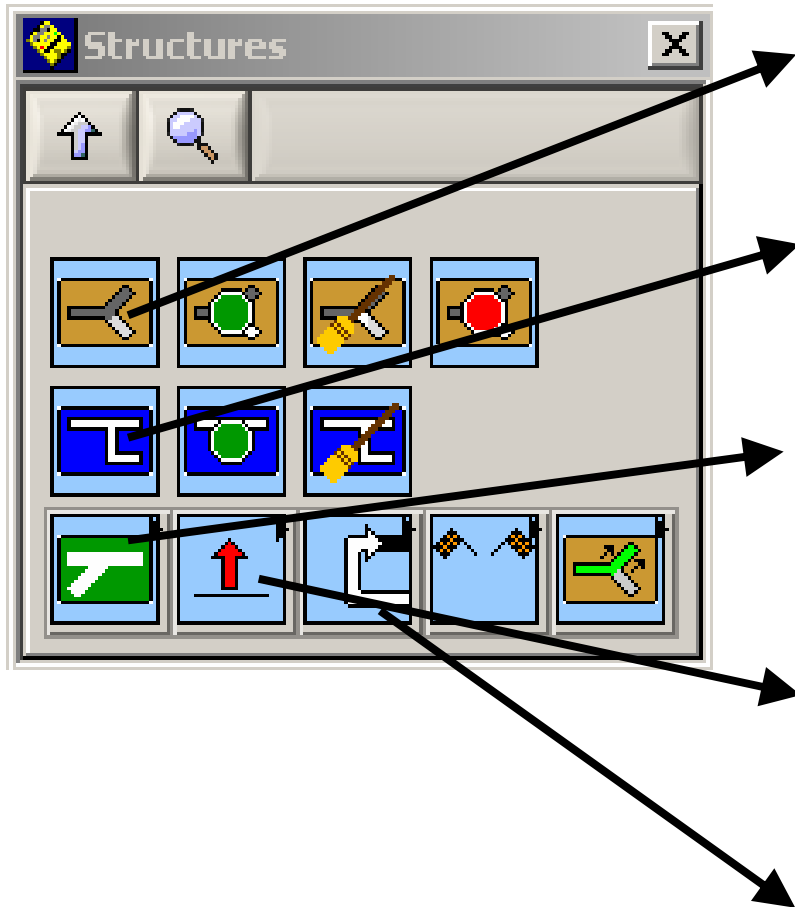


# Structures

---

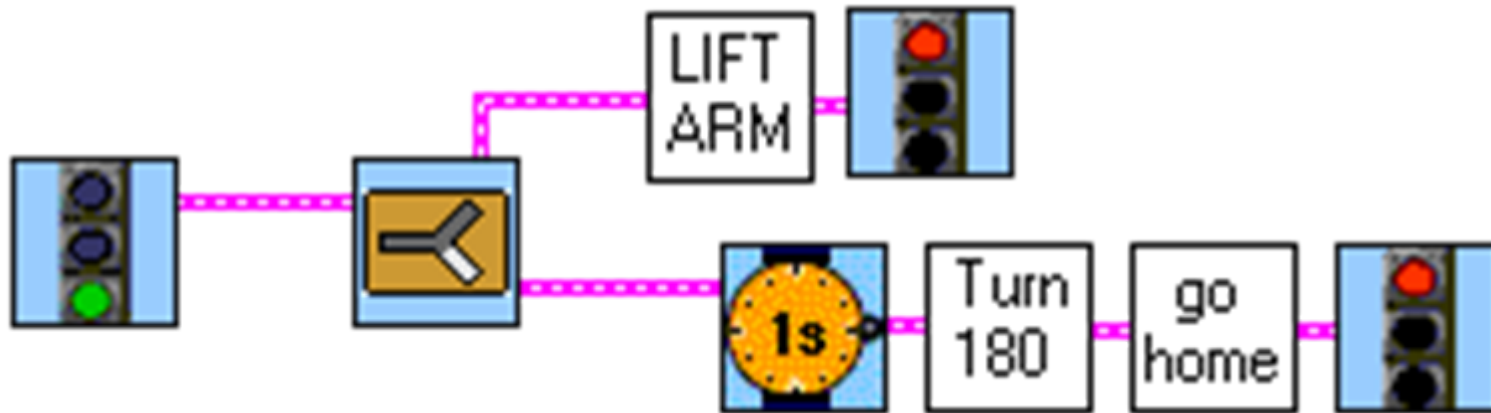
- Structures control groups of commands.
  - In creating an algorithm, many times you want to do something like, “As long as the light sensor reads a dark value, keep going straight.”
  - Repetition, logical choices, subroutines, multitasking are examples.
- A structure can replace a simple command.
  - LoopUntilTimer can replace a WaitForSeconds command.

# Structures



- Split Task
  - Make 2 parallel program tasks
- Subroutines
  - Declare, then call a local subroutine
- Forks
  - Either this way, or that (If, then, else)
- Jumps
  - Jump to another spot in program (GoTo)
- Loops
  - Repeat until something happens (While Do or For Next)

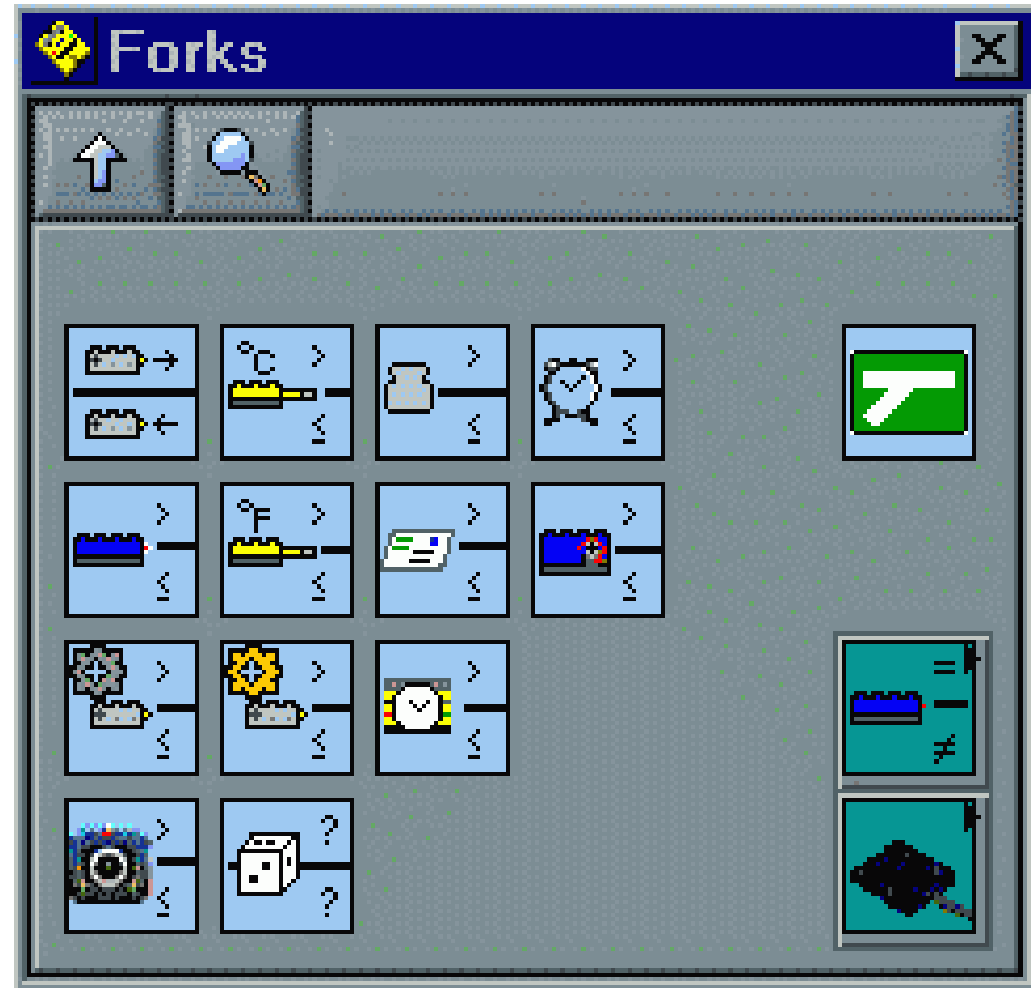
# KISS #4: Split Task



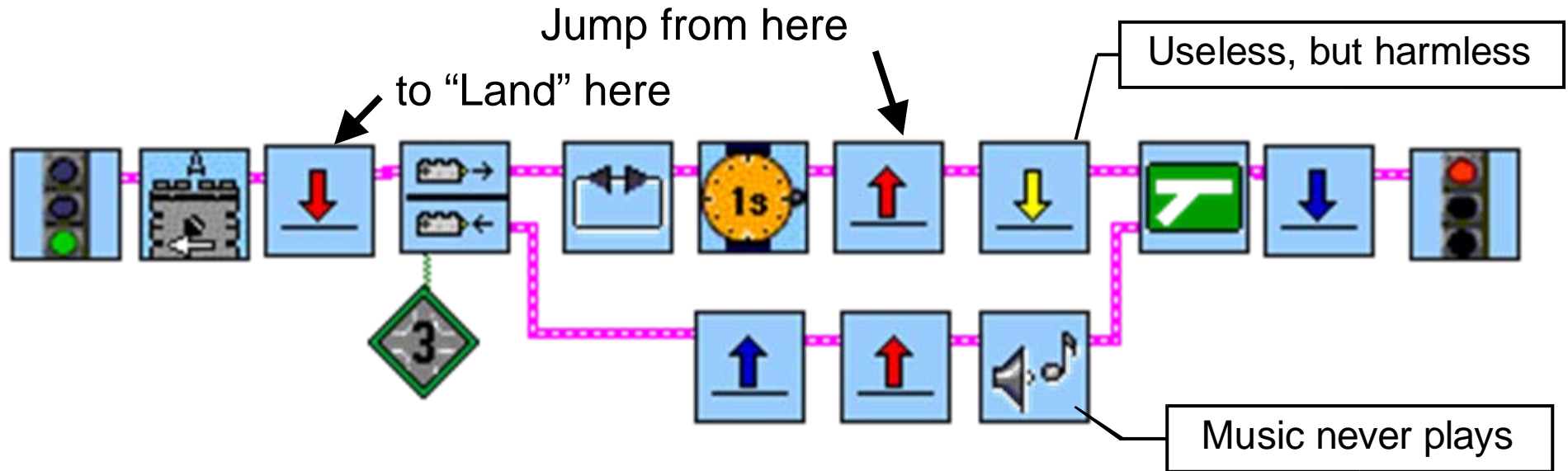
- Make two tasks that run independently.  
(Can you walk and chew gum?)
- Both tasks must stop independently.
- One task lifts the arm.  
The other task waits before turning around and heading home.

# Forks Palette

- Logical test, then pick one of two wires.
- Each fork requires one fork merge.



# Jumps



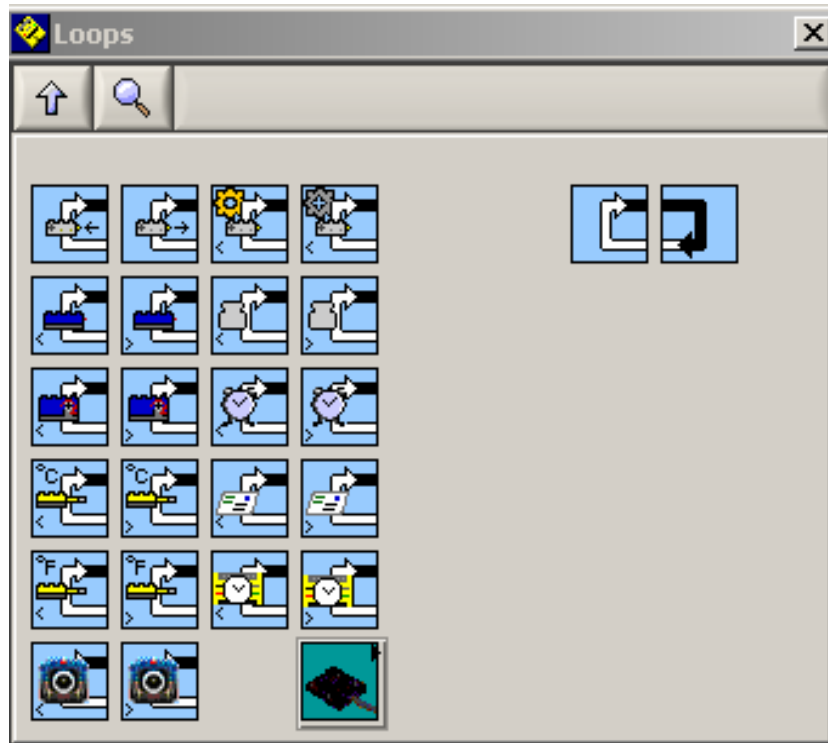
- Jumps can form infinite loops.
- Multiple jumps (like red above) are allowed, but only one "Land".
- ROBOLAB has a variety of jump colors/numbers to allow multiple jumps in your program
- Jumps are a simple control structure.
- Warning: Multiple jumps are difficult to debug.

# KISS #5: Loops

---

- Loops are a powerful and useful control structure
  - In other programming languages:
    - For ... Next Do loop  $n$  times
    - While ... Do Repeat until some test is false
- There are loops for every sensor to allow you to continually do a command or sequence of commands until a sensor condition changes
  - Extremely useful for navigation, moving an arm, etc.
- If your algorithm says something like: “While the sensor reads  $x$ , keep doing this”, then this is a place for a loop.

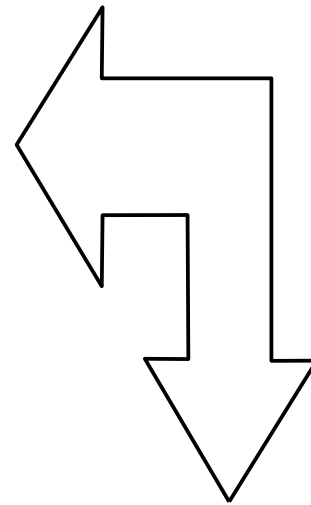
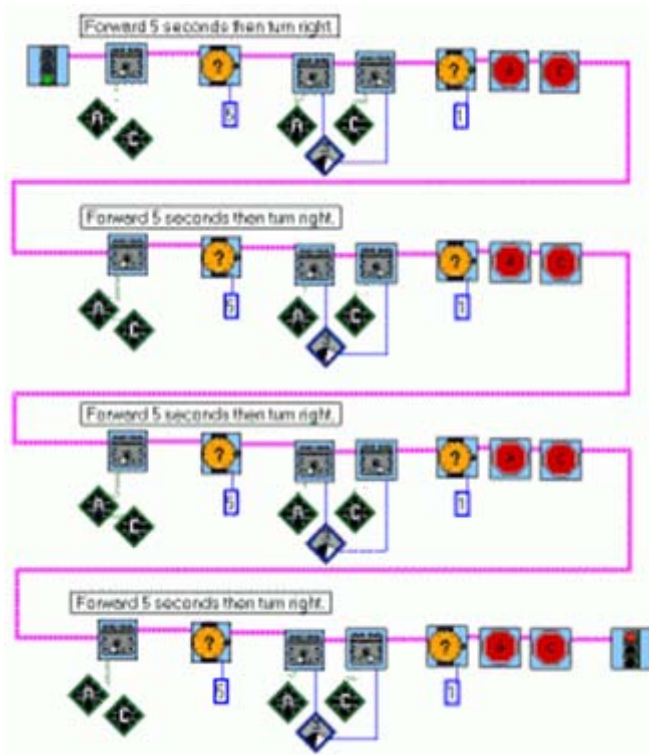
# Loops Palette



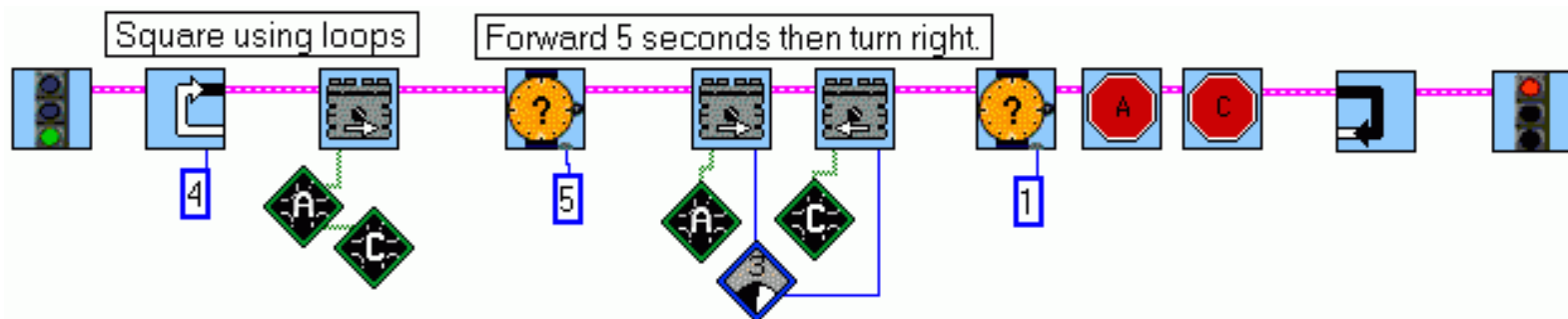
- Play music while the touch sensor is released.
- Additional commands can be put in the loop.



# Simple Loop



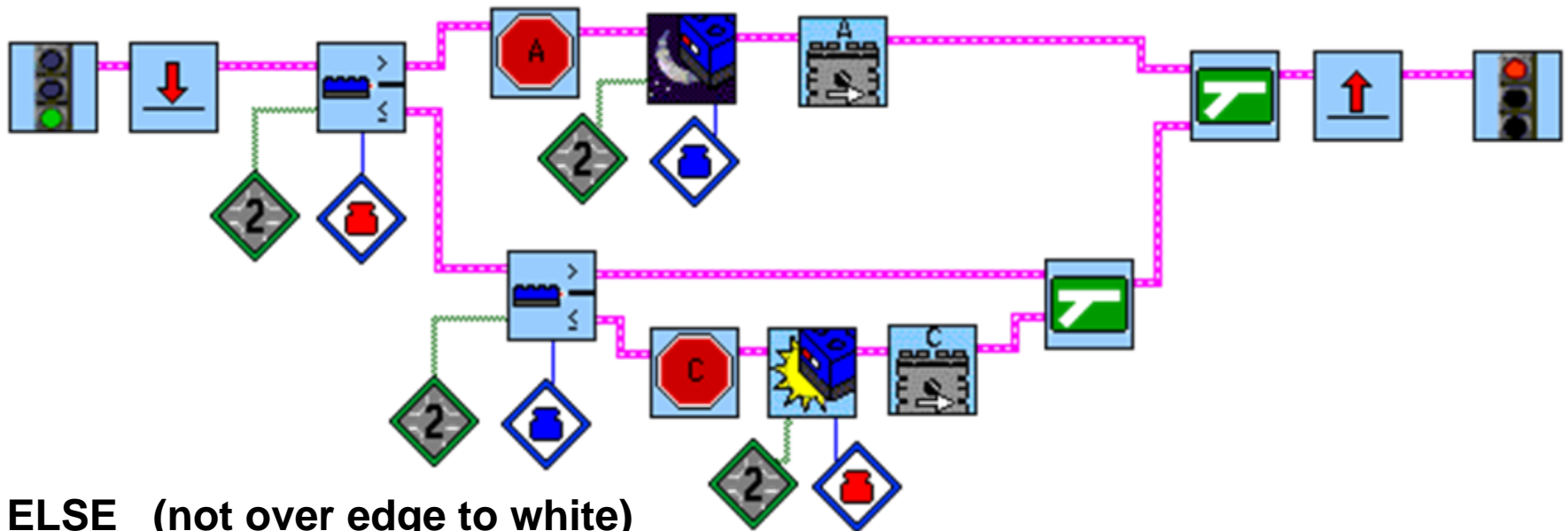
- Convert this set of commands
- to something simpler using loops



# Combining Structures

**Follow lines (using light/dark grey thresholds pg 65).**

**IF over edge to white, THEN turn left until over edge to dark.**

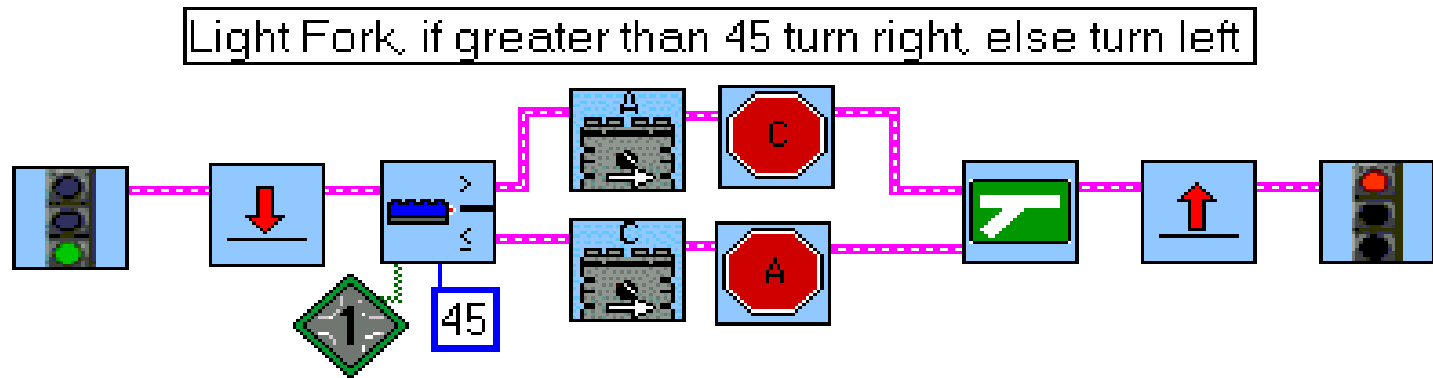


**ELSE (not over edge to white)**  
**IF over edge to dark, THEN**  
     **turn right until over edge to white,**  
**ELSE go straight.**

**This results in a zig-zag motion along the line edge.**

# Comparing Algorithms

- Compare the previous line follower to this one:



- Better**

Follows the thick black oval line of the Mindstorm Test Pad

- Simpler. Doesn't assume motors are running. One threshold.

- Worse**

- Motors only run one at a time.

# Comparing Structure Commands

---

You can generally make your algorithm fit the command you want to use.

- Loop

- Execute something until an event

- WaitFor

- Wait for an event, then execute something

- Forks

- Make a choice based on a sensor value at a given point in time.
- Be careful to make sure you will be watching for the event at the right time

# Lab Four

## Task:

Move exactly one lap around an oval.  
(Black 2cm line on white paper)

# Advanced Topics

***Events***

***Debugging Tools***

***Additional Resources***

# KISS #6: Events

---

A programming style used to handle situations where **any one, of many things, can happen**. It is most useful when more than two sensors are being watched at the same time.

An event is:

- The moment something happens.
- It's also the setup. Knowing what to watch, when, and where.
- And it's what to do after the event happened.

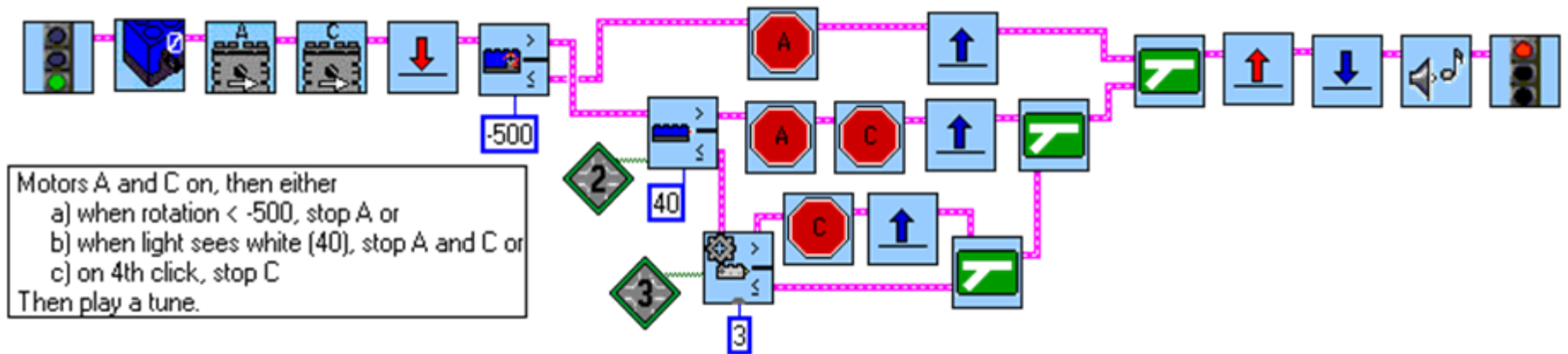
# Events (the logic)

---

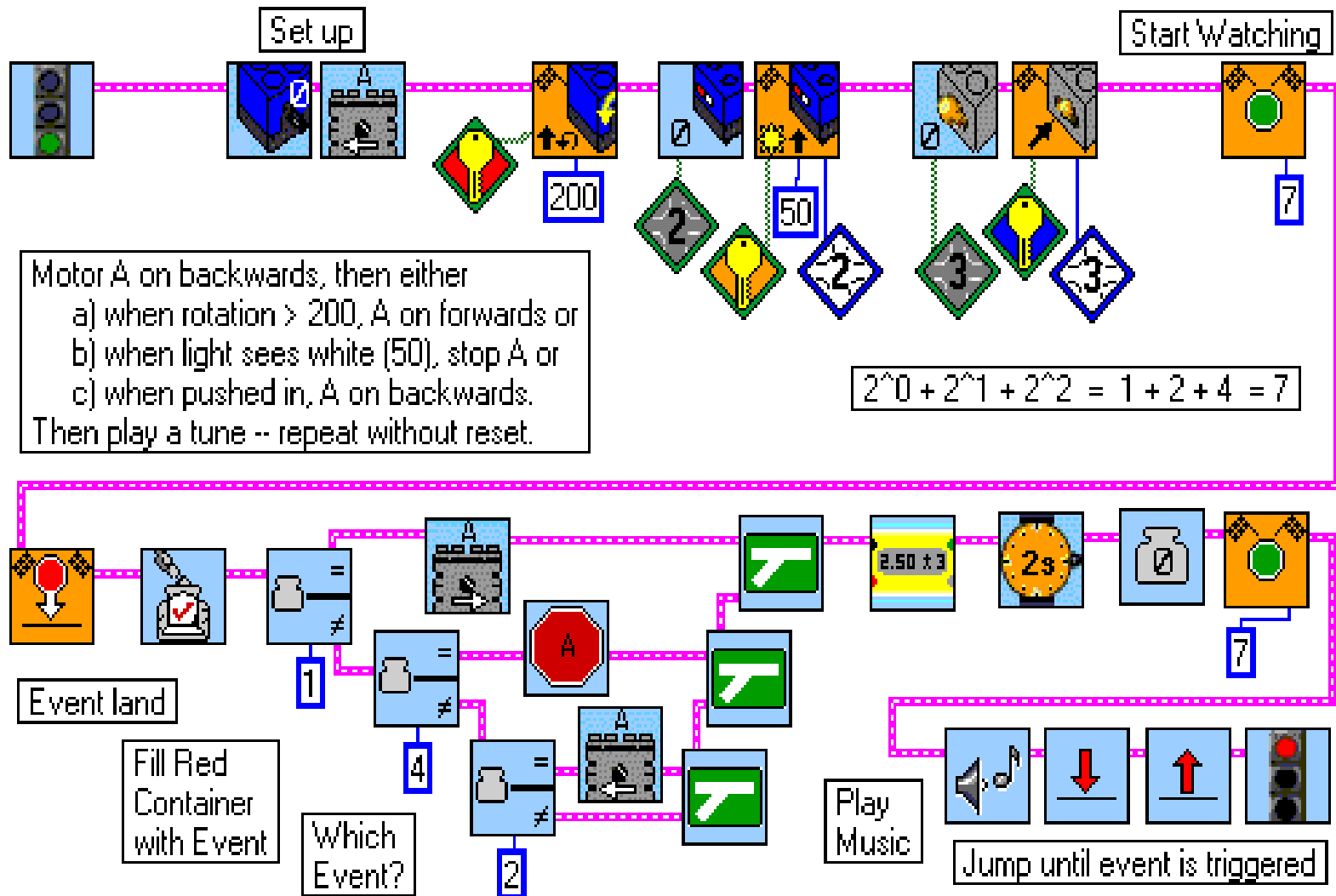
- Coding an Event
  - The setup: declare the sensor to watch, the port, the threshold that triggers the event.
  - Start watching
  - Event Landing
    - All events land in the same place. (ROBOLAB)
    - Determine which event triggered and react.
    - May continue on or restart event watching.
- Events could be recoded
  - Events could be coded as infinite loops containing sensor forks and jumps that land outside the infinite loop.

# Events Using Forks

This example watches 3 sensors using forks.



# Events



ROBOLAB 2.5 introduced event functions.

# Events (Hysteresis)

---

Hysteresis – a lagging between the time a force has an effect and the time it makes a change.

The word was created to describe the process in magnetic regions (like magnetic tape) that allows them to be changed, but then resist change.

Not related to the word “hysterical”, but often thought of as the behavior of sensors near their threshold value.

In robotics, a resistance to change factor added to sensor values so they don’t behave “hysterically”.

# Debugging Tools

- Music

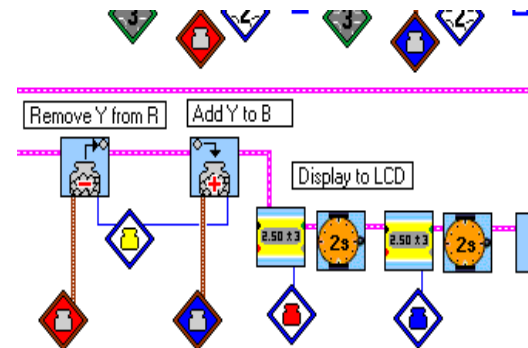
- RoboLab has several functions that play notes on the RCX.
  - Use them to identify sections of code.
  - Identify the end of the task, subprogram, or program.
- It helps to make the notes quick. A good ear can hear the difference between C, E, and G, otherwise use two notes.

- LCD

- ROBOLAB 2.5 introduced a function to write to the RCX's LCD panel.

- RCX Interrogator

- From your PC (or MAC), use the interrogator to view the values of containers and sensors.





# Additional Resources

---

- Internet:
  - [www.hightechkids.org/fll/coaching](http://www.hightechkids.org/fll/coaching) (this presentation)
  - [www.lugnet.org](http://www.lugnet.org)
  - [www.crynwr.com/lego-robotics/](http://www.crynwr.com/lego-robotics/) (firmware decoded)
  - [www.ni.com](http://www.ni.com) (LabVIEW™)

# Robolab Keyboard Shortcuts

---

- <Ctrl-H> help window toggle
- <Ctrl-B> remove broken wires
- <Ctrl-R> download program to RCX
- <Ctrl-E> panel <==> diagram window toggle
- <Ctrl-C> copy
- <Ctrl-X> cut
- <Ctrl-V> paste
- <Ctrl-N> new file
- <Ctrl-S> save file
- Changing the mouse pointer
  - <space> pointer tool
  - <tab> cycle tools



# Robolab Limitations

---

- 8 local subroutines
- 10 tasks
- 32 variables (20 useful containers)
- 4 timers
- 16 events
- 5 program slots

# Putting it All Together

***How to Become an FLL Ace Programmer  
in 10 Easy Steps***

# FLL Ace Programmer in 10 Steps

---

1. Create a map of where the robot goes and what it does. These are your Requirements.
2. Use the Requirements to further examine the problem
  - What tasks can go in the same program?
  - Any actions we do in multiple places? (good candidates for subroutines)
  - Will using variables help?
3. Write out your algorithm

# FLL Ace Programmer in 10 Steps

---

4. How it could fail. How can you recover?

- For example, the robot hits a wall it shouldn't have. What can you do to allow it to recover?

5. How are you going to test and debug it?

- Perhaps use a series of beeps in the program to tell you where the program is.

6. Have a system for versions.

- Put comments at the beginning of the program
- Always save a working version in a file with a name that makes sense (like date, etc.).



# FLL Ace Programmer in 10 Steps

---

## 7. Write the code using above information

- Code little parts and test them.
- Name subroutines and files with descriptive names. Right\_turn is better than Rturn.
- Think about how readable the code is. Make it less confusing.
- Use a lot of comments.

# FLL Ace Programmer in 10 Steps

---

## 8. Fix bugs in a stepwise manner

- Fix the bug, test it, then test other things related to it to make sure they weren't broken by your fix.
- 80% of the bugs come from 20% of the code.

## 9. Don't be afraid to scrap everything and start over if things are getting complex and fragile.

## 10. If coding/testing/bug fixing is driving you insane, go have an ice cream cone! Take a break, have a friend look at your code, come back another day.