FLL Programming 101
RCX Code 2.0*

August 9, 2004
Version 1.0

* Also Known As: RIS : Robotics Invention System
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 document is released under the terms of the GNU General Public License (GPL), Version 2. You may redistribute it, or modify it, only under terms of this license. For purposes of this document, the Microsoft Word document should be considered source code and Adobe PDF should be considered object code. The GNU GPL should be available with this document or is available at: http://www.opensource.org/licenses.

• LEGO®, ROBOLAB, and MINDSTORMSTM 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
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
  – Meant to teach an approach to programming

• What this class is not
  – Not an exhaustive reference on every language command
Class Agenda

- Computer basics
- Programming environment
- Simple commands
- Subroutines
- Sensors
- Repetitive statements
- Putting it altogether
- Advanced topics
Format of Each Section

- Theory
- Language Specific Commands
  - Robolab or RCX Code
- Examples
- Frequently Encountered Situations
- Tips and Tricks
- Lab
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 but ROBOLAB has more growth potential.
  • For FLL at the High School level, no language restrictions.
The 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 basics

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

Memory: 32K of RAM
Firmware on the RCX

Software

Firmware Setup

Your PC

Your RCX

Download

Set Direction A–C

Set Power AC

On AC for 5.0 sec.

Set Direction A–C

On AC for 2.0 sec.

Bytecodes

\{\text{SetPower(A,C,8)}\}

RAM (Random Access Memory) *

Your Programs

Firmware

Processor (Hitachi)

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

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

©2002 INSciTE
Firmware Loaded?

• Firmware must be installed onto your RCX so that you can program the RCX brick.

• 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

• Every computer program is a model, of a real or mental process.
• These processes are intricate in detail, and usually only partially understood.
• They are rarely modeled to our satisfaction
• Thus, they 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 the computer is to carry out
  - Inputs to be supplied
  - Outputs to be produced
- Devise an algorithm, or sequence of steps, by which the computer can produce the required outputs from the inputs
- Express the algorithm as a program in a computer language

From: Introduction to Pascal, Welsh and Elder,
Writing a Computer Program

• Specify the task the computer is to carry out
  – Move forward 5 seconds
  – Inputs: Time, power, direction {In an RCX some “inputs” are set in the program}
  – Outputs: Motor drive
• Devise an algorithm, or sequence of steps,
  – Class example
• Express the algorithm as a program in a computer language
  – Class example
Running a computer program (RCX)

1. Write program on PC (RCX Code or Robolab)
2. Program conv. to bytecodes (text description)
3. Download to the RCX
4. Bytecodes converted to Hitachi (RCX) machine code commands
   - Move X to register 001
   - 1000111...
5. RCX executes commands
Tips and Tricks (1)

• The RCX has 5 program slots
  – Slots 1 and 2 are locked, must be unlocked in administration page.
  – Later we’ll show how to use a touch sensor to get you more slots (virtually)

• You can also change the time delay before the RCX auto powers down.

• Shielding IR port
  – The IR tower has a significant range, be careful in an environment with downloading going on
  – Build a shield with LEGO bricks for your RCX during the competition
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 program.
  - Rechargeable batteries work fine.
  - 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 from Digikey (www.digikey.com)
  - Properly used, batteries should last awhile
The Programming Environment
Main Menu (RCX Code 2.0)

Click to enter programming work area

Click to enter SETTINGS area
Settings (RCX Code)

Click to set up IR tower

Is firmware version 2.0 installed? If not, click Download Firmware and follow instructions.
Work Space (RCX Code)
Debugging Screen

Step by Step debugging by checking the actual values on the RCX
Simple Commands or Small Blocks
Simple Commands

- Simple commands are basic actions
  - Like English statements
    - Turn motor on
    - Stop motor
    - Reverse motor direction
- Setting parameter values
  - Many commands have parameters that can be set
    - Motor power
    - Time
Small Blocks

Select the ON FOR command and drag it.

Turn On Motors ABC for 1 second

Click the tab to view command options (or right click green block)
Driving Motors

• Making the motors move and do something is the point (the output) of your program. That’s what makes your creation a robot!

• Formally called the 9 volt geared motor
  – Without load, motor shaft turns at about 350 rpm
  – With typical robots the power usage should allow 3-4 hours of use on a set of batteries

• FLL currently allows up to 3 motors on your robot
Motor Details

- Motor can be set to different power settings
  - 5 settings in Robolab
  - 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
Setting Motor Parameters

Turn On Motor A and C - on at Full Power

Motor direction can also be reversed by reversing the connector on the RCX. Useful during early stages of design.
Turning the Motor Off

Turn On Motors ABC for 1 sec. Then they stop.

Turn Off Motors ABC. More on next slide

End Program will stop the motors otherwise they will keep running

Motors run Until Stopped!
Motor Off

- Braking turns motor off and applies “brakes” to stop motor
- Coasting turns motor off but motors are “freewheeling”. Your robot will keep rolling until friction or an object stops it.
Other Commands

Always use an End Program command (otherwise your motors will continue to run)

Inserting Beep or Tone commands are useful debugging tools as they let you know where in the program things are happening.

Stops the program and waits for a period of time
Lab One

Task:
Move forward for 5 seconds and return

Then try:
Move forward for 5 seconds, turn right 90°
Problem Solving and Programming Process
Generic Problem Solving Process

- What’s the problem?
- Brainstorm solutions
- Evaluate solutions
- Try (implement) best solution
- Evaluate results

- In our current discussion, the solution is expressed as an algorithm, then converted to a computer program
What’s an Algorithm?

• The term algorithm (pronounced AL-go-rith-um) is a procedure or formula for solving a problem. The word derives from the name of the mathematician, Mohammed ibn-Musa Al-Khowarizmi, who was part of the royal court in Baghdad and who 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
## An Algorithm is Like a Recipe

### Recipe for French Toast

<table>
<thead>
<tr>
<th>Ingredients</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 slices of bread</td>
<td>Mix together eggs, milk and flour and pass through a</td>
</tr>
<tr>
<td>2 eggs</td>
<td>strainer. Dip slices of bread into the mixture and fry</td>
</tr>
<tr>
<td>1 cup milk</td>
<td>in the fat or butter on both sides in a frying pan.</td>
</tr>
<tr>
<td>1/4 cup flour</td>
<td>Before serving, sprinkle with powdered sugar.</td>
</tr>
<tr>
<td>fat or butter</td>
<td></td>
</tr>
<tr>
<td>powdered sugar</td>
<td></td>
</tr>
</tbody>
</table>

How can we make this more “algorithm-like”?

Count eggs, mix eggs, for every 2 eggs add 1 cup milk and 1/4 cup flour, and pass through a strainer. For every 2 eggs, dip 8 slices of bread….
Ways to Express Algorithms

• In the real programming world there are many ways to do this

• In the FLL world, probably the two best ways are:
  – Draw block diagrams
  – Literally act it out

• Always talk it out and test it using a team member to walk through it acting like the robot.
  – Keep actions at lowest possible level
    • Go Forward 3 steps
    • Stop
    • Turn right 90°
FLL Ace Programmer in 5 Steps

• Map the generic problem solving process to programming

• Create a map of where the robot goes and what it does
• Write what the program should do (your algorithm).
• Code it
• Test, and fix, little pieces at a time

We’ll be adding to this process as the class progresses
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 motor
  – Turn left motor on for ? Seconds.
• Stop all motors
Conversion to a Program

Set directions of motors

Set power of motors

Turn motors on and start timing

After 5 seconds, turn left.
Debugging and Analysis

- Literally walk through it
- Ask lots of questions
  - What if...s
- 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
- Use pieces that you know already work
  - For example, you know how to turn $90^\circ$
- Feel confident in your algorithm before starting to code it.
Tricks and Tips

• Don’t try to knock off too much at once
  – Stick to one or two missions
• Remember to KISS (Keep It Simple)
• Develop a little library of algorithms and programs you know work with your robot.
  – Going straight, turning, etc.
• Keep things modular
  – This helps a great deal when debugging
• It’s faster fixing problems at the algorithm level than coding something and seeing what happens
• Give programs descriptive names
  – Mission1V2 not Amy3
Frequently Encountered Situations (FES)

• You are really 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? To build? To program?
• You can’t think of all the steps needed for the algorithm
  – Get out your robot, and walk through it.
Subroutines and Comments
Subroutines

• Many times in your programming you will want to do similar things
  – Turning 90° Left
  – Moving an arm to do something
• A subroutine is a small piece of a program that you can reuse and insert in your bigger program as a single command
  – Called My Blocks in RCX Code 2.0
• Subroutines are a very important aspect of a good, structured program
  – Allows you to create a small program fragment that works and reuse it without thinking about its details
Subroutine Use

• If you have a motion or action you can do reliably, and will use a lot, that’s a good candidate for a subroutine.
• It’s easier in an algorithm for a solving a mission to think “Turn left here at 90º” than always repeat the set of commands you need to do that.
• Another key part of a subroutine is “Modularity”. It allows you to isolate in one place the code for turning.
  – See next slide for example
Modular Advantages

Algorithm - Mission 1

Start motors
Go straight 10"
Turn Left 90°
Go Straight 3.5"
Turn Left 90°
Dump barrels

Algorithm Updated - Mission 1

Start motors
Go straight 10"
Turn Left
Go Straight 3.5"
Turn Left
Dump barrels

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.

Change twice

Subroutine (only change once)
Subroutine Commands

Decide on piece of code

Copy and paste into MyBlock that you name

MyBlock now available for use
My Block Names

• Use 10 or less characters to keep the font size large. After 10 characters it gets too small to read.

• Suggest using “action + to + target”, but this can be varied. Some examples:
  – Fwd2Wall
  – Fwd2Line
  – FwdDist
  – TurnRight
Comments

• Comments are notes in your program explaining what and why something is being done
• **Very important**, as the program writer quickly forgets why something is done in a certain way
• In a team process like FLL, comments are especially important as more than one person will be working on the program.
• Add things like who made changes, how battery power affects change, etc.
Comment Use

Drag the comment icon to the program block where you want it and it snaps on.

Click on the tab and write the comment.
Debugging and analysis

- Make sure your subroutine code works well before making it a subroutine!
- Always check the "Download to Slot" option on the settings screen before each download!!
- Remember: Use subroutines to hide complexity and use comments liberally!!
Lab Two

Task:
Make the program from Lab One into a subroutine
Data Input

Sensors
Sensors

- Sensors allow your robot to sense something in the real, physical world
- Timer
  - Internal sensor, keeps track of time
- Touch Sensor
  - Has your robot made contact with something?
- Light
  - Is the surface light or dark?
- Rotation
  - How many times has an axle turned?
- Sensors are input to your program
Touch Sensor

- Allows robot to detect touching or bumping into something
- Good to detect robot arm movements (the sensor is activated when the arm moves far enough to push in the touch sensor). This is called a limit switch.
Sensor Watcher is “always active” as it runs in parallel to the main program. In other words, other commands may be executed while the Watcher is waiting and looking.
Touch Sensor Wait Until

- Wait for a touch sensor to be pushed or let go
- More commands in repetitive commands section
- Doesn’t look at sensor until this command is reached, unlike the Watcher on previous slide.
Timer

Wait until Timer reaches 1 second

• Time in tenths of a second
• Timer must be zeroed before it’s used
• RCX Code 2.0 has 3 timers, 1, 2, and 3

Important!: Reset before you use
Timer Watcher

• When Timer reaches 1 second, do command string
  • Time in tenths of a second
  • Timer must be zeroed before it’s used
  • RCX Code 2.0 has 3 timers, 1, 2, and 3

• Timer Watcher is “always active” as it runs in parallel to the main program. In other words, other commands may be executed while the Timer Watcher is waiting. Unlike the wait command which occurs in the main program. No other commands will execute until the wait is over.
Light Sensor

• A light sensor detects reflected light from its red light emitting diode.

• Light sensors operate in "percent" mode, anywhere from 0 to 100 in value.
  
  • Higher the number, the brighter the light, hence the lighter the surface as more light is reflected back

• Light sensor can be the most frustrating sensor due to it’s variability
Light Sensor Readings

- Lowest reading likely 20% (in very dark room)
- Highest reading likely 100% (pointing directly at sun or incandescent light bulb)
- Normal operation values tend to be 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
  - You can’t always predict what this will be, you should experiment with your surfaces/colors
- Light sensor is extremely sensitive to the distance between the sensor and the reflecting surface. Even small variations can make the readings unusable. Try to 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.

• It is very sensitive to ambient light. Shield the sensor as much as possible and try it out under the conditions that you will compete in. Test it on competition day.
Light Sensor Watcher

Wait for a bright event, a dark event, a blink, or a value within a certain range.

Automatic setting checks the light value when you hit run and calls that “neutral”. A value higher is then called “brighter”, etc.
Light Sensor, Wait Until

- If you want the robot to continue doing something while the light is a certain value, use the Repeat While, covered next section
Calibrate Light Sensor

Position light sensor over white area and push touch sensor to take a reading

Set variable *Bright* to this value

Position light sensor over dark area and push touch sensor to take a reading

Set variable *Dark* to this value

- This program assumes you physically move the robot over the light and dark areas, and then use a touch sensor to trigger a reading.

- The variables Bright and Dark now contain the calibrated values of light and dark. These variables can be used elsewhere in your program.
Refining the Calibration for Edge Following

Position light sensor over white area and push touch sensor to take a reading

Set variable *Bright* to this value

Position light sensor over dark area and push touch sensor to take a reading

Set variable *Dark* to this value

Determine the difference between light and dark, and set variable *delta* to this value.

Increase *Dark* by 30% (and decrease *Bright* by 30%) to follow the line edge. Remember page #63 which shows an average light reading
• The rotation sensor is used to measure how far a rotating axle has turned. As the axle turns, a counter in the RCX is incremented or decremented.

• Each full rotation registers as 16 counts giving the sensor a resolution of 22.5 degrees (360/16). Hence it is sometimes called the angle sensor.
Turn the Rotation Sensor On

By Default, it is turned off. You must go into the Advanced Setting Menu to turn it on.
Sensor Watcher

When Rotation (count equal to 64 or 4 complete rotations), then execute stack.

Also can wait for rotation value greater than, less than, or in a range.
Other Rotation Commands

Wait Until works just like other Sensor Wait Until Commands

Reset Rotation counter to zero
Using the Rotation Sensor

- Extremely powerful for navigation (how far have you gone) or how much you have turned an arm

- Turn motors on to go straight
  - Wait until eight complete rotations
  - {How far is eight rotations? See next slide}

- To do this well requires some of the Repeat commands covered in next section
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)

\[ \text{Circumference} = \pi \times \text{Diameter} \]
More on Rotation Sensor

- Rotation sensor can count forwards or back, up to Absolute Value (32,767), then it rolls over to largest negative from largest positive (or vice versa), however this is really unlikely to occur in FLL.
- Increase sensor resolution with gear reduction
- Rotation sensor is quite reliable if shaft speeds are kept in 60-1000 rpm range, some counting problems may occur at very high or very slow speeds.
  - Motors typically around 200 rpm.
Debugging and Analysis

• Common sources of repeatability problems for the rotation sensor
  – Programming: forgetting to reset the sensor 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.
Remember Different Sensor Commands

Program commands execute in order

Program will wait here until sensor condition is true. Sensor value is NOT checked elsewhere in the program.

Program continues executing commands in order

This stack of commands will execute once EVERY time this sensor condition becomes true
Tricks and Tips

• Use a touch sensor to get more program slots
  – If touch sensor is in, execute one program stack
  – If touch sensor is out, execute another program stack
  – Do this for every program slot, and you end up with 10 program slots instead of 5.
  – Add a brick to robot between missions to push the touch sensor in, to achieve this.

• Use a light sensor to achieve rotation counting
  – Aim light sensor at a piece of your robot that rotates and measure color differences (for example, have two colors on a piece that rotates), and count these.
Stacking Sensor Ports

- Attaching multiple sensors to the same port
- Allows you to use more than 3 sensors
- Touch sensors work best for this, rotation sensors cannot be stacked.

<table>
<thead>
<tr>
<th></th>
<th>Touch Open</th>
<th>Touch Closed</th>
<th>Light Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch Open</td>
<td>0</td>
<td>1</td>
<td>light level</td>
</tr>
<tr>
<td>Touch Closed</td>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>Light Sensor</td>
<td>light level</td>
<td>100</td>
<td>???</td>
</tr>
</tbody>
</table>

Sensor readings for 2 stacked sensors
Lab Three

Task:
Move forward for 50 rotations, turn right 90°
or
Experiment with other sensors
Variables and Repetitive Commands

Controlling looping, doing things over and over, and adding structure
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 white for a light sensor may be measured during the program and placed in a variable called Bright.
• Think of a variable as a container containing the value of something.
  – In ROBOLAB, variables are called containers
• Useful in subroutines to pass in parameters
  – For example go straight \( N \) seconds where \( N \) is a variable.
Defining and Using a Variable

Add Variable

Enter name for a new variable

Ok  Cancel

Repeat

for

Place Block Here

RotationCount

Variables

counter1
RotationCount
create new variable

Untitled

File  Edit  Settings

Copy
Add a Variable
Delete a Variable
Add a Comment

©2002 INSciTE
Repetitive 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”.
- Repetitive commands (repeat, control statements, while commands) are used for this purpose.
- These commands allow the repetition of some command, or group of commands, until some condition is met.
- There are always multiple ways to do these, this section will cover some examples.
- RCX Code calls these Repeat Commands.
Four Repeat Commands

Repeat some set of command N times

Repeat some set of commands WHILE a sensor reading is at a certain value

Repeat some set of command forever

Repeat some set of commands UNTIL a sensor reading is at a certain value

The two commands on the right are very useful for navigation and control
Loops

• Loops are the most powerful and useful control structure
  – Also called repetitive commands or repeat or while commands
• 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.
Simple Loop

- Go in a square.
- Comment should describe what is happening.

- Another approach, go forward until rotation sensor is greater than a certain value.
- Note that Repeat Commands are nested.
Yes or No Command

If a sensor reading meets the condition

Then execute Yes string
Else execute No string
More Complex Loop

This is the rest of the light sensor program from previous section.

If over edge to white, then turn left until over edge to dark.

If not over edge to white, then if over edge to dark, then turn right until over edge to white, else go straight.

This results in a zig-zaggy motion along the line edge.
Comparing Different Sensor Commands

- Sensor Watcher
  - Always watching for the event
  - Many times hard to fit into an algorithm
  - Useful for waiting for a discrete event
    “go forward to hut, when touch sensor triggered, then stop, dump barrels and return to base”

- Repeat/Wait
  - Only watches for event at actual command, which means you could miss the event
  - Many times easier to fit into an algorithm
  - Can lead to long strings of code - use subroutines.
  - Be careful when writing it to make sure you will be watching for event at right time
Comparing Different Sensor Commands

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

• The Sensor Watcher may seem more intuitive at first but it is not always as easy to get it to do what you want.

• Repeat
  • Execute something until an event

• Wait
  • Wait for an event, then execute something

• Yes/No extremely useful for making a choice based on sensor value at a given point in time.
Tricks and Tips

• Never wait for a sensor to be an exact value, always greater than or less than
  – You could miss the event due to sampling rate
  – For example, wait until rotation greater than 64 not equal to 64.
Lab Four

Task:
Using a light or rotation sensor and loops
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

4. Think about 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.).
7. Write the code using above information
   - Code little parts and test them
   - Name subroutines and files with descriptive names
     (Right_90_Deg_turn is better than Rturn)
   - Think about how readable the code is. This makes it less confusing.
   - Use a lot of comments
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.
Summary

- Remember the problem solving process
- Remember to create algorithm before writing your code
- Use subroutines to keep things cleaner
- Be careful in using sensor commands, especially sensor watchers.
- Use comments liberally
- Program and test little pieces at a time
- Have fun!!