The logo for Team Ford First is a shield-shaped emblem. The top half is pink and contains the word "TEAM" in white, dotted letters. The bottom half is light blue and contains the word "FIRST" in white, dotted letters. The word "FORD" is written in large, white, solid letters across the middle of the shield. The text "C Programming and the DEFAULT Code" is overlaid on the top half of the shield.

C Programming  
and  
the DEFAULT Code

4 and 11 November 2006

Mr. Bob Koehl

[ace4bert@sbcglobal.net](mailto:ace4bert@sbcglobal.net)

[www.teamfordfirst.org/Events](http://www.teamfordfirst.org/Events)

# Preparing to program a FIRST Robot in C

- Describe the software needed for robot programming
- What is the DEFAULT Program?
- Learn the default names of items connected to the robot controller and operator interface
- **Learn critical C Program Keywords, Operators, and Functions.**
- Change the DEFAULT Program
- Use the Dashboard Viewer to diagnose programs.

# C Program Variables

- A Variable is exactly what the name implies: the value can change as the program is executed.
- Variables can be of three main types: integer, float, and character.
- We are going to focus on integer and float.
- But first we need to know how to name variables.

# C Program Variable Names

Variable names can be assigned to any data type that will be used in the program.

Variable names can be any number of characters long.

The first character **MUST** be from the alphabet.

The remaining characters can be any valid keyboard character.

Names are sensitive to upper and lower case; **MyName** is NOT the same as **MYNAME**, **myname**, or **mYnAmE**.

# C Program Variable Names

Good programmers use short names that apply to what is being programmed.

Most programmers follow guidelines for names:

- all lower case names are program variables;

- all UPPER case names are program constants that are macros assigned by the *#define* directive;

- multiple word names are either separated by the underscore or by starting each additional word with an upper case letter.

For example, *my\_date* and *thisIsJune26* are names that follow this guide line for multi word variables.

These guidelines make it easier to visually scan programs.

# Names used in Robot Program

- Before we try to program the robot from scratch, we should look at the program that is given to us by Innovation First Robotics.
- The program is called the DEFAULT Code or Program.
- [2006\\_DEFAULT\\_Code-zip file](#)
- [user\\_routines.c](#) program file

## user\_routines.c file

- Look at the file [user\\_routines.c](#)
- Notice the variable names used throughout the file.
- Do they look familiar?
- What does `pwm01 = p1_y;` mean to you?
- What would it mean to running the robot?

A large, semi-transparent watermark of the FRC Team logo is centered on the slide. The logo consists of a shield shape with a red top half and a blue bottom half. The word 'TEAM' is written in white, dotted letters across the top half, and 'FIRST' is written in white, dotted letters across the bottom half.

`pwm01 = p1_y;`

- What does `pwm01 = p1_y;` mean to you?
- (Answer:) Give the Victor connected to pwm01 the y-value of the joystick connected to Port 1.
- What would it mean to running the robot?
- (Answer:) Assuming a motor is connected to Victor 1, the motor should go forward when the Joystick is moved forward and go backwards when the Joystick is moved backwards.

`relay3_fwd = p3_sw_trig;`

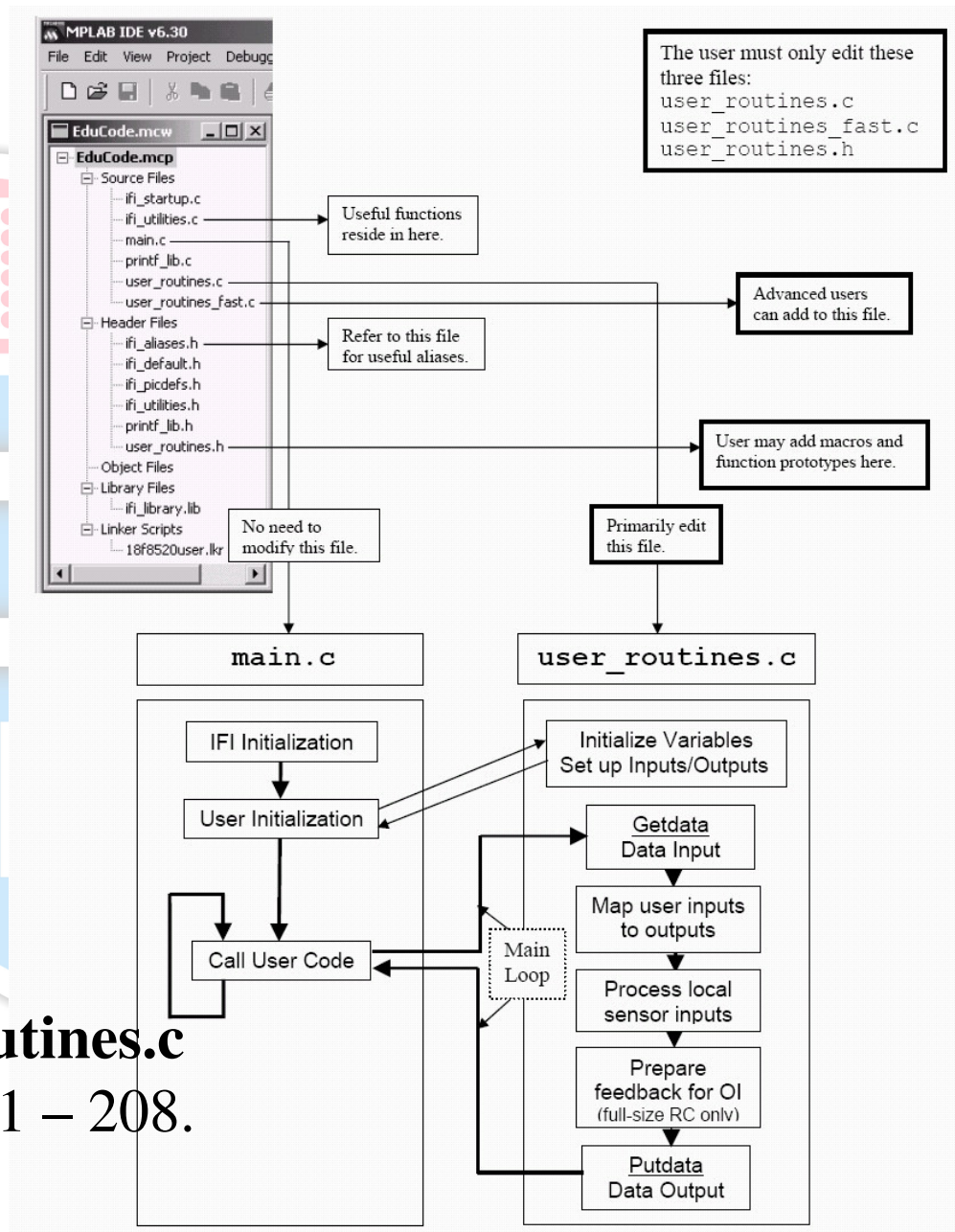
- What does `relay3_fwd = p3_sw_trig;` mean to you?
- (Answer:) Give the Spike Relay connected to relay1 the value of the joystick trigger connected to Port 1.
- What would it mean to running the robot?
- (Answer:) Assuming something is connected to Spike 1, it would turn ON when the Joystick trigger is pressed and turn OFF when the Joystick trigger is NOT pressed.

# DEFAULT Program flow

Review: There is a "Main Loop" in `user_routines.c` that expects to execute every 0.026 seconds (38.5 times per second).

Let's look through `user_routines.c` on the computer lines 181 – 208.

4 and 11 Nov 2006

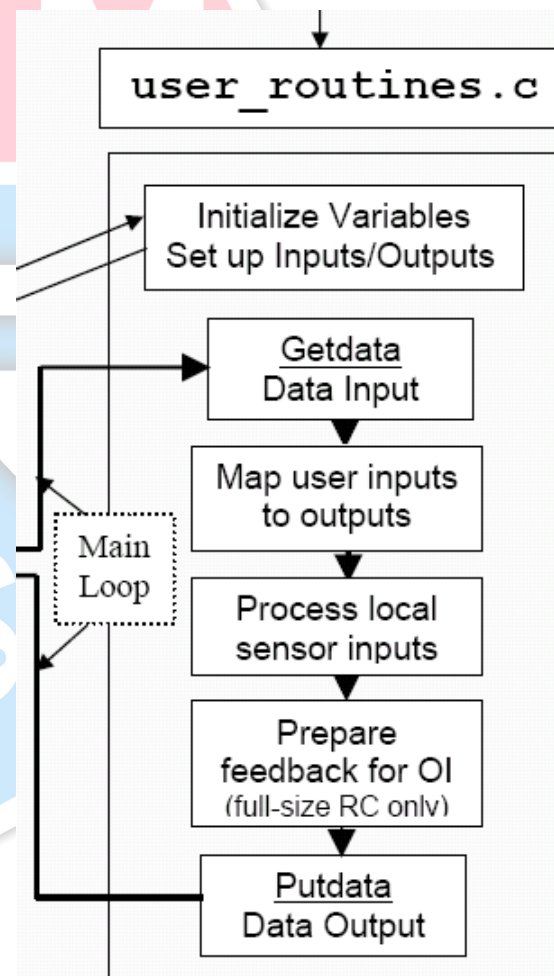


# Program Scan Concept

The program is the repeated loop that is marked with the words "Main Loop" in the illustration.

This loop reads data, processes the data by calculating new output values, then runs the robot by outputting the new values, before doing the loop again and again

....



# C Program Comments

- Comments should be liberally used to explain what is being done AND WHY.
- Lines that begin with a `/*` are green and stay green until a `*/` and can cover multiple lines.
- A comment that is a single line begins with `//`

# C Program Statements

- The most common program statement uses the “=” (equal sign).
- Most program statements end with a “;” (semi colon).
- The statement **a = b;** means to assign the value of b to the variable a.
- What other kinds of statements do you see in user\_routines.c?

# C Program Function Statements

- Look at lines 238 through 250 in `user_routines.c`
- Two concepts are shown:
  - Use of the Function “Limit\_Mix”
  - Multiple equal signs
- Let’s look at each item.

# Functions

- **Limit\_Mix(2000 + p1\_y + p1\_x - 127);**
- Somewhere in the program the function **LimitMix** must be defined.
- In user\_routines.c this is in line 86.
- The function is a group of program statements that can return a value or perform some kind of operation.

# Multiple equal signs

- **pwm13 = pwm14 = Limit\_Mix(a variable);**
- Multiple equal signs are handled from right to left when the equation is calculated.
- That means **pwm14** is set to the value returned by the function.
- **pwm13** is then set to the value of **pwm14**.

# Equations order of operation

C programs may include many calculations that use **+**, **-**, **\***, and **/** to **add, subtract, multiply** and **divide** variables and/or numbers.

The order of operation of any equation is:

1. Perform all **\*** **and** **/** from left to right.
2. Perform all **+** **and** **-** from left to right.
3. Parenthesis can alter the order, causing the calculations inside the parenthesis to be performed from left to right before calculating the rest of the equation.

# Sample Equation solution

C would solve the equation:

$$x = 3 + 2 * 6$$

$$x = 3 + 12$$

$$x = 15$$

If the equation was to multiply the sum of  $3 + 2$  by  $6$ , the equation and solution would be:

$$x = (3 + 2) * 6$$

$$x = 5 * 6$$

$$x = 30$$

Be sure the equation is written in the format to get the solution that YOU need. IF in doubt, use parentheses.

# Keywords

**auto**

*double*

*int*

**struc**

**break**

*else*

*long*

**switch**

**case**

**enum**

**register**

**typedef**

*char*

**extern**

**return**

**union**

**const**

*float*

*short*

*unsigned*

**continue**

**for**

*signed*

**void**

**default**

**goto**

**sizeof**

**volatile**

**do**

*if*

**static**

**while**

*Blue italic font* indicates Keywords we will use **SOON**

## if Statement

```
if (condition is TRUE) //NO semi colon here
```

```
{ //{ starts a series of commands  
  command; //executed if the condition is true  
  command; //executed if the condition is true  
} //} end of if condition is true
```

- if the condition is **NOT TRUE** the commands within the braces { ... } are skipped.

# Conditional Operators

- In C there are a number of operators which give **CONDITIONAL**, that is logical, results.
- These operators return a 1 for true, or a 0 for false.
- C always takes a 0 value to be false and a nonzero value to be true.
- The largest use of Conditional Operators is with the **if** instruction.

## if Statement

```
if (CONDITION is TRUE) //NO semi colon here
```

```
{ //{ starts a series of commands  
  command; //executed if the condition is true  
  command; //executed if the condition is true  
} //} end of if condition is true
```

- if the **CONDITION** is **NOT TRUE** the commands within the braces { ... } are skipped.

# condition

Conditions: if ( condition is TRUE)

<b>==</b> (double =) equal	<b>if( p1_x == 127)</b>
<b>!=</b> not equal	<b>if( p1_y != p1_y_old)</b>
<b>&gt;</b> greater than	<b>if( p1_y &gt; 107)</b>
<b>&gt;=</b> greater than or equal	<b>if( p1_x &gt;= 108)</b>
<b>&lt;</b> less than	<b>if( p1_y &lt; 137)</b>
<b>&lt;=</b> less than or equal	<b>if( p1_x &lt;= 138)</b>

# multiple *conditions*

**OR** conditions use the symbols **||**

```
if( p1_sw_trig == 1 || p1_sw_top == 1)
```

If *either* the trig is equal to 1 *or* the top is equal to 1  
the condition is TRUE

**AND** conditions use the symbols **&&**

```
if( p1_sw_trig == 1 && p1_sw_top == 1)
```

If *both* the trig is equal to 1 *and* the top is equal to 1  
the condition is TRUE

## **else** keyword

**else** is used with the **if** instruction

```
if(condition is True)
{ commands if condition is True
}
else
{ commands if condition is False
}
```

# if .. else .. if combinations

multiple **else** may be used with multiple **if** instructions

```
if(condition 1 is True)
{
  commands if condition 1 is True
}
else if(condition 2 is True)
{
  commands if condition 2 is True
}
else
{
  commands if neither condition is True
}
```

An **if .. else if .. else** only executes the first True condition that is found. The remainder of the else .. if tests are ignored.

## if .. else if .. else example

```
if(p1_sw_trig == 1) //if trigger is held
{ pwm01 = 254; } //run motor full speed forward
else if(p1_sw_top == 1) //if top is held
{ pwm01 = 0; } //run motor full speed reverse
else //if neither trigger nor top is held
{ pwm01 = 127; } //stop the motor
```

What would happen if the last else was left out?

(Answer): The motor would continue to run at the speed set by the last **if** that had a True condition.

## if .. else if example

```
if(p1_sw_trig == 1) //if trigger is pressed
{ relay1_fwd = 1; } //turn high speed ON
else if(p1_sw_top == 1) //if top is pressed
{ relay1_fwd = 0; } //turn high speed OFF
relay1_rev = 1 - relay1_fwd; //set the rev
```

What would be different if the **else** was **not** used??

Would it work the same when BOTH the trig and the top were held at the same time? ..and if the **else** was **not** used??

# Keywords introduced so far



<b>auto</b>	<i>double</i>	<i>int</i>	<b>struc</b>
<b>break</b>	<i>else</i>	<i>long</i>	<b>switch</b>
<b>case</b>	<b>enum</b>	<b>register</b>	<b>typedef</b>
<i>char</i>	<b>extern</b>	<b>return</b>	<b>union</b>
<b>const</b>	<i>float</i>	<i>short</i>	<i>unsigned</i>
<b>continue</b>	<b>for</b>	<i>signed</i>	<b>void</b>
<b>default</b>	<b>goto</b>	<b>sizeof</b>	<b>volatile</b>
<b>do</b>	<i>if</i>	<b>static</b>	<b>while</b>

*Blue italic font* indicates **Keywords introduced.**

# Adding Variables



- As modifications are made to the DEFAULT Program, there will be the need to add new variable names.
- Variables are added in the section of the program where they are needed

# Integer Variable types

*char* -128 to +127

*unsigned char* 0 to 255

*int* -32,768 to +32,767

*unsigned int* 0 to 65,535

*long* -2,147,483,648 to + 2,147,483,647

*unsigned long* 0 to 4,294,967,295

*short long* -8,388,608 to +8,388,607

*unsigned short long* 0 to 16,777,215

# Integer Math “oddities”

*int*, *long* and *char* (whether *signed* or *unsigned*) are Integers.

## *Differences in integer math from regular math*

1. *int* division: the remainder is dropped.  
Examples:  $10 / 3 = 3$  and  $2 / 5 = 0$  ! **Take note!!**
2. *int* math in general: if the answer is beyond the range of the integer type, the answer will not be what is expected.

Example:  $x = (\text{unsigned char})(25 * 25 / 25);$

$25 * 25 / 25$  is NOT 25

Example:

$x = (\text{unsigned char})(25 * 25 / 25);$

The expected answer of  $25 * 25 / 25$  is 25.

An **unsigned char** can be no more than 255; therefore as the equation is solved, there will be a problem.  $25 * 25$  is greater than 255 so the answer will be 113 that is divided by 25 which gives 4 as the answer ( $113/25=4.52$  drop decimals)

**Take Note: be sure to define variables of a type large enough to handle the largest part of the equation being solved**

# Preparing to program a FIRST Robot in C

- Describe the software needed for robot programming
- What is the DEFAULT Program?
- Learn the default names of items connected to the robot controller and operator interface
- Learn critical C Program Keywords, Operators, and Functions.
- **Change the DEFAULT Program**
- Use the Dashboard Viewer to diagnose programs.

# Declaring Variables

- In C it is necessary to tell the compiler about all the variables before using them. This enables the compiler to know what sort of variable it is, and how much space it takes. This process is called declaring variables.
- Here is an example of how to declare two global variables : x and y :

**unsigned char x;**

**unsigned char y;**

**unsigned char x;  
unsigned char y;**

- **unsigned char** keyword is the type of the variable
- A char type is actually 8 bits, and can store numbers from 0 to 255
- As x and y are the same type they could also be declared as follows:  
**unsigned char x,y;**
- The statement that declares a variable always ends with a **;**

# Use these in an application

- Suppose we wanted to make the **Switch1\_LED** light turn on the first time we pressed the **p1\_sw\_trig** and turn off the next time we pressed the **p1\_sw\_trig**.
- This is called making an output “*toggle*” and could be used to control parts of the robot using one button, for example high and low gear.

# Create a “toggling” output

- Can you think of how to make a toggling output?
- One approach:  
Each time the trig changes from being OFF to being ON, change the condition of the output.
- The only way to know if something has changed is to create a variable to store the “old” value of the switch.

# Create the toggle

Define a new variable and initialize it to OFF (0):

```
unsigned char old_p1_sw_trig=0;
```

Check to see if the trig has changed from OFF to ON:

```
if(old_p1_sw_trig == 0 && p1_sw_trig == 1)
```

IF the trigger has changed, reverse the state of the output

```
Switch1_LED = 1 - Switch1_LED;
```

Update the old\_trig to the current trig value:

```
old_p1_sw_trig = p1_sw_trig;
```

**Remember the program scans from top to bottom.**

**That's why this works.**

# The complete “toggle”

```
unsigned char old_p1_sw_trig=0;

if(old_p1_sw_trig == 0 && p1_sw_trig == 1)
{ Switch1_LED = 1 - Switch1_LED;
}

old_p1_sw_trig = p1_sw_trig;
```

Can you think of another way to do the “toggle”?

