# Robot C Tutorial

ENGR 120/121

January 2017

# Introduction to Computer Programming

- A computer program is a set of instructions that precisely specify the operations and logic needed to complete a defined task.
  - Specification of data involved in task.
  - Operations required to complete task.
  - Logic to control operations.

# Introduction to C

- Robot C is the programming language used to program your VEX controllers.
- Robot C is a variation of the C programming language developed by AT&T in 1972.
  - C is a popular language for programming microcontrollers.
  - A very good language to learn to help obtaining work terms.
- Inspiration for Objective C, C++, and Java languages.

# An example program

```
int Sum(int highestValue)
{
  int sum;          // summation variable
  int counter;      // counter variable

  sum = 0;          // Reset sum to zero
  // Counter will take every value from 0 to
  // highestValue.
  for (counter=0;counter<highestValue;counter++) {
    sum = sum + counter;  // Increase value of sum
  }
  //  Return calculated value as output.
  return(sum);
}
```

# Programming Robot C

- C is a strongly typed programming language.
- If you need to store a value in your program:
  - Storage of value is called a variable.
  - Type of each data variable must be defined.
- Types available:
  - Integers (e.g. -2,-1,0,1,2,…)
  - Booleans (e.g true or false)
  - Enumerated types (see later)
  - Floating point values (e.g. 0.98785)
  - Characters (e.g. ‘a’)

# Basic Robot C Syntax

- Most statements end with semicolon “;”
  - Exception function and task headers.
- Names are case sensitive.
  - “fred” is not the same as “Fred”.
- Statements are grouped together with curly brackets “{“ and “}”
- Logical tests are placed in parenthesis “(“ and “)”

# Types of Data in Robot C

- Integer value variables
  - Declared as type `int`
- Floating point variables
  - Declared as type `float`
- Boolean variables
  - Declared as type `bool`
  - Can take values of `true` or `false`.
- Enumerated variables
  - Set of values of variable must be defined.
  - Defined as type `enum`

# Declaring an enumerated type

- Desire something to hold state of a fisherman:
  - States considered: Rod down, Rod up, and Drinking beer.
  - Declaration in Robot C:

```
typedef enum {
    ROD_DOWN,
    ROD_UP,
    DRINKING_BEER
} T_FISHERMAN;
T_FISHERMAN fishermanState = ROD_UP;
```

# Using Variables

- You can assign a variable a value with "="
- e.g. `x=3;`
  - Assigns a value of 3 to `int` or `float` variable "x"
- Boolean variables can be assigned:
  - `true` or `false` directly.
  - Another `bool` variable.
  - A logical expression:
  - e.g. `bool check = ( x > 3 );`
    - check assigned value of `true` if x greater than 3.
    - check assigned value of `false` if x less than or equal to 3.

# Logical “Gotcha” in C

- Use the “==“ operator to create a `bool` value to check if a variable is equal to a certain value:
  - `( x == 3 )` is `true` if x equal to 3.
  - `( x == 3 )` is `false` if x not equal to 3.
- Common mistake when writing C:
  - `( x = 3 )` assigns a value of 3 to x and evaluates to `true` no matter what the initial value of x was.
  - Watch for this!!!

# Logical AND/OR/NOT in Robot C

- Two `bool` variables: `bool1` and `bool2`
- Logical NOT: Calculate NOT of `bool1`:

```
!bool1
```

- Logical AND: Calculate `bool1` AND `bool2`:

```
bool1 && bool2
```

- Logical OR: Calculate `bool1` OR `bool2`:

```
bool1 || bool2
```

# Logical Flows

- Most programs have conditional execution.
- If statement structure:

```
if ( x == 3 ) {
    // If x equal to 3,
    // this code is executed.
    …
} else {
    // If x not equal to 3,
    // this block is executed.
    …
}
```

# Switch Statements 1

- Switch statement allows multiple logical branching.
- In the following example x is an `int`

```
switch(x){
   case 0:    // Jump to here if x == 0
     y = 0;
     break;   // Jump out of switch.
   case 1:    // jump to here if x == 1
     z = 35;
   default:   // jump to here if no match
     v = 29;
}
```

# Switch Statements 2

- When `break` keyword encountered, execution jumps past end of switch statement.
- In example above:
  - If x is 0, then y is assigned 0.
  - If x is 1, then z is assigned 35 and v is assigned 29.
  - If x is any other value then v is assigned 29.

# While Loops

- Execution of code in loop is repeated while given logical statement evaluates to true.
  - Condition checked at beginning of each loop.
- Example:

```
int c = 0;
while ( c < 3 ) {
    c = c + 1;
}
```

# For Loops

- Compact form for specifying loop.
- For statement specifies:
  - Initialization condition before starting loop.
  - Termination logical condition.
  - Statement to be evaluated at the end of each loop.

```
int c;
for (c=0; c<3;c=c+1) {
    // This for statement reproduces
    // behaviour of while loop
    // example on previous slide.
}
```

# Exiting a Loop Early

- There are two special operations for loops.
- The `continue` statement causes the current loop iteration to be terminated and then start the next loop iteration.
- The `break` statement immediately terminates the loop and execution starts after the end of the loop.
  - The `break` statement also is used in `switch` blocks as described above.

# Functions in Robot C 1

- To facilitate code reuse, Robot C allows you to define functions that can be called from elsewhere in your code.
- A function is definition has a
  - Return data type.
  - Function name.
  - List of input arguments.

# Functions in Robot C 2

- Look at our example code on Slide 4 which computes the sum.
- Properties of Robot C functions:
  - Input arguments can be read but not modified inside of a function.
    - If you change the value of an argument inside of that function, that change will not be seen by the code that called the function.

# Built-In Operatons in Robot C

- The `SensorValue[]` operator is used
  - to read the value of a VEX sensor port:

    ```
    value = SensorValue[sensor_name];
    ```

    - `sensor_name` is configured in “Motors and Sensors Setup”
      - Port connection and type of sensor must be configured.
  - To write a value to a VEX sensor port.
  - `SensorValue[sensor_name] = value;`
    - `sensor_name` is configured in “Motors and Sensors Setup”
      - Port must be configured as a “Digital Output” sensor.
      - Only values of 1 and 0 can be written to the port.
      - Writing a ‘0’ causes the signal line to be set at 0 Volts.
      - Writing a ‘1’ causes the signal line to be set at 3.3 Volts.

# Motor Control in RobotC

- Motors are controlled by assignments:

```
motor[motor_label] = motor_value;
```

  - `Motor_label` configured in "Motors and Sensor Setup".
  - If `motor_value > 0`, motor tries to rotates forward
  - If `motor_value < 0`, motor tries to rotates backward
  - If `motor_value == 0`, motor stops.

Gotcha: The magnitude of the control value must be above a threshold value for motor to rotate. Threshold value is determined by load on the motor.

# Quadrature Sensors in RobotC

- Quadrature encoder is a special sensor:
  - Read with:

```
rotation = getMotorEncoder(motor_label);
```

    - `motor_label` refers to the motor that the encoder is connected to.
    - Configured in “Motors and Sensor Setup”.
  - Reset the encoder to zero with:

```
resetMotorEncoder(motor_label);
```

  - Minor gotcha: About 627.2 ticks per rotation

1 tick ≠ 1 degree

# Even More Built-In Functions in RobotC

- You have 4 timers available in RobotC:
  - `T1`,`T2`, `T3`, and `T4`.
- Reset timer `T1` back to zero:
  `clearTimer(T1);`
- Read time since last reset of `T1` in milliseconds:

```
time_elapsed = time1[T1];
```

- Stop program execution for `time_msec` milliseconds: `wait1Msec(time_msec);`

# Program Constants

- If you need a constant in your code you use the special `const` keyword.
- Example to set an integer constant called motorLevel to 50:

```
const int motorLevel = 50;
```

- Robot C will generate an error if you try to assign a new value to constant variable.

# Code Structure

- The sensors and motor names are defined by the programmer.
- The code execution is started in a special function denoted as

```
task main()
{
    …
}
```

# Example of Writing Code 1

- Here we would like to write some code to control a simple robot.
- The robot has two touch sensors.
  - Two touch sensor called `button1` and `button2`.
- The robot has a motor connected.
  - Motor is denoted as `motor1`.

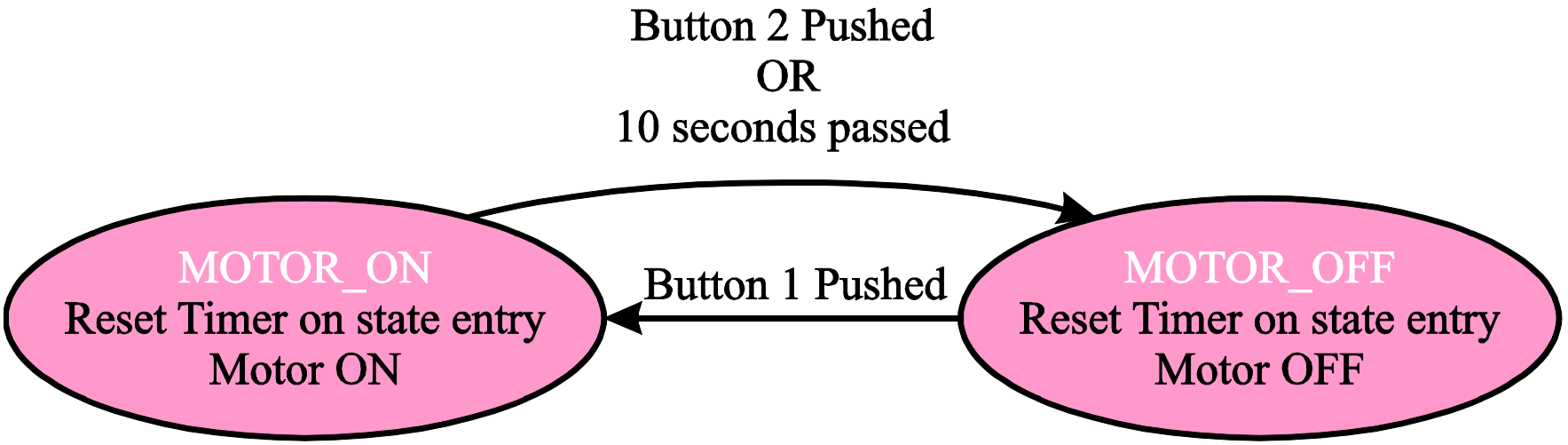
# Example of Writing Code 2

- Robot behaviour is given as
  - Initially robot starts with motor off.
  - When a button1 is pushed the Robot runs the motor for 10 seconds.
    - If button2 is pushed the motor is stopped immediately.

# Define Finite State Machine

- Robot has two states: MOTOR_OFF, and MOTOR_ON.
- When in MOTOR_ON:
  - Run motor at desired level
  - Transition to MOTOR_OFF state when button2 pushed.
  - Transition to MOTOR_OFF state when timer passes 10 second count.
- When in MOTOR_OFF:
  - Motor is off.
  - Transition to MOTOR_ON state when button1 pushed after resetting timer.

# Finite State Machine Diagram



# Recommended Code Structure

- Define constants at top of code.
- Provide a function for each state.
  - Each state checks inputs and timers to see if transition to new state is needed.
- Run a finite state machine in the main task.
  - An infinite loop runs in the main task.
  - In each iteration, check current state and call the indicate state function.
  - State functions return the state for the next iteration.

# Constants for Code

```
// Motor power level
const int motorLevel = 50;
// Motor run time in 1 msec ticks.
const int timeMotorRun = 10000;

// Enumerated type for holding state.
typedef enum {
  MOTOR_OFF = 0,
  MOTOR_ON
} T_state;
```

# Motor Running State Function

```
T_state MotorOnState()
{
    if ( time1[T1] > timeMotorRun ||
         SensorValue(button2) == 1 ) {
         // Turn off motor
         motor[motor1] = 0;
         // Transition to motor off state.
         return(MOTOR_OFF);
    } else {
         // Keep in motor on state.
         return(MOTOR_ON);
    }
}
```

# Motor Off State Function

```
T_state MotorOffState()
{
    if ( SensorValue(button1) == 1 ) {
        // Turn on motor
        motor[motor1] = motorLevel;

        // Transition to motor on state.
        return(MOTOR_ON);
    } else {
        // Keep in motor off state.
        return(MOTOR_OFF);
    }
}
```

# All of Main Code

```
task main() {
  // Storage for system state
  T_state systemState = MOTOR_OFF;
  T_state newState;

  // main loop
  while (true) {
    switch(systemState) {
      case(MOTOR_ON):
        newState = MotorOnState();  // Run Motor Run State function.
        break;
      case(MOTOR_OFF):
        newState = MotorOffState(); // Run Motor Off State function.
        break;
      default:
        // This should never be run.
    }  // matched to switch(systemState)
    if ( newState != systemState )
      clearTimer(T1); // Reset timer when state changes.
    systemState = newState;  // Set new state for next loop.
  }  // matched to while(true)
}
```

# Main Code Initialization

- Allocates variables to store the state.
- Sets first state.

```
// Storage for system state
T_state systemState = MOTOR_OFF;
T_state newState;
```

# Main Code Loop

```
// main loop
while (true) {
  switch(systemState) {
    case(MOTOR_ON):
      newState = MotorOnState();  // Run Motor Run State
   function.
      break;
    case(MOTOR_OFF):
      newState = MotorOffState(); // Run Motor Off State
   function.
      break;
    default:
      // This should never be run.
  }  // matched to switch(systemState)
  if ( newState != systemState )
    clearTimer(T1); // Reset timer when state changes.
  systemState = newState;  // Set new state for next loop.
}  // matched to while(true)
```

# Notes about Main Loop

- Switch statement for states.
  - Clauses for each state call separate functions.
- We use timer `T1` to record how long we have been in the current state.
  - Reset the timer when the state has changed.
- This is used in the `MOTOR_ON` state.
  - When the timer goes over 10 seconds in this state, we transition to the `MOTOR_OFF` state.