



Machine Intelligence Laboratory

Machine Intelligence Laboratory

MIL

<http://www.mil.ufl.edu>

ICC11 Primer

Director

A. Antonio Arroyo, PE, Ph.D.

Associate Professor in

Electrical and Computer Engineering

University of Florida

Gainesville, Florida

Email: arroyo@mil.ufl.edu

Telephone: (352) 392-2639

Introduction to the Image Craft 68HC11 C Compiler (ICC11)

This document is not meant to be an all-inclusive reference manual for ICC11. Instead, it is meant to show a few of the features that might not be obvious or intuitive. The definitive document on ICC11 is the *ICC11: ImageCraft 68HC11 Compiler and Development Environment Version 5.0* manual located in the IMDL Lab in Benton 335B. This document assumes that the reader has had some basic experience programming in C.

ICC11 Installation Cautions

The lab has customized versions of ICC11 version 5.0 for each of the 68HC11 boards available in MIL or those sold by Mekatronix {e.g., ME11, MSCC11, etc.}. If you are restoring ICC11 from the backup floppies make sure you unzip the zipped backup file into the proper directory name {i.e., “*directory_name*” is one of the following names: ICCTJ, ICCME, ICCTJP, ICCTK, etc.}. Edit your AUTOEXEC.BAT to include the *directory_name*\BIN in the path and reboot the machine. The ICC11 file ICC.INI should reside in the *directory_name*\BIN directory and not in the WINDOWS directory (if so delete the copy in the WINDOWS directory).

For example, suppose you wish to restore the ICC11 version used with the ME11 daughter board to the M68HC11 EVBU. First, insert the 3.5” floppy marked ICCME in the floppy in the floppy drive, open it using the WINDOWS EXPLORER and double click on the file ICCME.ZIP to start WINZIP. Press the EXTRACT icon in the toolbar. Now type the directory name C:\ICCME in the “Extract To:” box and hit the EXTRACT button. WINZIP will then create the directory ICCME in drive C of the computer. Next, edit the AUTOEXEC.BAT file in the root directory of drive C and add C:\ICCME\BIN to the path. Now you must restart the computer in order for the path statement to take effect. Finally double click on the ICC11IDE.EXE program icon in C:\ICCME\BIN to start ICC.

ICC11 Data Types

The following data types are supported in ICC11 (among others):

int	2-byte integers	
	signed	-32768 to 32767
	unsigned	0 to 65535
char	1-byte	
	signed	-128 to 127
	unsigned	0 to 255
char*	pointer to a string	
int*	pointer to an integer	

The long, float and double data types are 4 bytes each. You can use ARRAYS, SETS and STRUCTS.

ICC11 Syntax

ICC11 syntax is standard (ANSI compatible). All declarations need to be at the top. After regular instructions you cannot make additional declarations.

```
/* COMMENTS */
#include <filename> // file includes & headers //
void foo(int); // function prototypes //
int x,y; // variables //
void main()
{
    int a;
    a++;
    foo(a);
}
void foo(int x);
{
    ...
}
```

ICC11 Libraries

Before using any of the library functions below, the appropriate initialization programs must be executed. The *init_<program>* for the files below are:

For the ME-11	For the TJPRO	For the TJ	For the Talrik
<code>init_analog();</code>	<code>init_analog();</code>	<code>init_analog();</code>	<code>init_analog();</code>
<code>init_clocktjp();</code>	<code>init_clocktjp();</code>	<code>init_clocktj();</code>	<code>init_clocktk();</code>
<code>init_motorme();</code>	<code>init_motortjp();</code>	<code>init_motjdr();</code>	<code>init_motork();</code>
<code>init_servome();</code>	<code>init_servotjp();</code>		<code>init_servotk();</code>
<code>init_serialtp();</code>	<code>init_serialtp();</code>	<code>init_serialtj();</code>	<code>init_serialtk();</code>

The following library functions and a brief description of these appears below.

analog.c

The functions for using the analog port on the M68HC11. Defines the *init_analog()* and *analog()* functions.

init_analog();

➤ Call once before using any analog port inputs.

int analog(portE_bit);

This function returns the 8-bit number that is the value of the analog input port designated by the parameter *portE_bit* (0 .. 7) for the analog value you want to read.

clocktjp, clocktj and clocktk

Generates msec, seconds, minutes, hours, days. Also, variables *timertj*, *timertj* and *timertk*, respectively, are msec unit free running counters which can be used. The function *wait()* permits your program to execute time waits.

motorme, motortjp, motortj, motork

These are driver functions for the motors attached to the ME, TJP, TJ and TK boards, respectively.

init_motorme(); init_motortjp(); init_motortj(); init_motork()

➤ Call once at the beginning of the program before using any DC motors.

motorme(index, speed); motorp(index, speed); motortj(index, speed); motork(index, speed);

➤ This function(s) will set a new motor speed. The first parameter, *index*, specifies the motor whose direction you want to change (0 or 1), and the speed parameter is a value between -100 and +100.

servome, servotjp, servotk

These are driver functions for the servomotors attached to the ME, TJP and TK boards, respectively.

init_servome(); init_servotjp(); init_servos()

➤ Call once at the beginning of the program before using any servo motors.

servo(index,position);

➤ This function(s) will set a new servo position. The first parameter, *index*, specifies the servo motor you want to change (0, 1, ... no of servos), and position is an integer 0 or 2000 to 4000. You'll have to play around with the servos to figure out which position values move the servo where you want it to go.

serialtp, serialtj, serialtk

These are driver functions for the using the serial port on the ME, TJP, TJ and TK boards, respectively. Use `<stdio.h>` for most input functions.

init_serial();

➤ Call once at the beginning of the program before using the serial port.

read_int();

➤ This function(s) will return an integer from the input stream.

Character I/O routines require special handling and implementation. See the ICC11 manual pp. 27.

The following program will hang until you receive something on the serial port.

```
int c;  
c = get_char( ); /* Hang until something is received */
```

If the user types an 'A' the character received will have value 65.

ICC11 General Header Files

hc11.h

This file contains ICC11's predefined pointers to all of the M68HC11 internal registers. These pointers have the register names that have been defined in the HC11 reference manual (e.g., TMSK1, PACTL, HPRI0, etc.). They are accessed like variables, both for reading and writing. For example:

```
counter = TCNT; /* puts the value of TCNT into a variable named 'counter' */  
PORTB=0xFF; /* Change the value of PORB to FF hex */
```

The ICC11 macros `INTR_ON()`; and `INTR_OFF()`; are equivalent to the CLI and SEI instructions, respectively and are used to set or clear the M68HC11 interrupt flag (i-bit).

To use, include the file *hc11.h* at the beginning of your program.

mil.h

This file contains some useful functions for bit manipulation.

```
CLEAR_BIT(x,y); /* Clear all bits in x corresponding to the bits set in y, other bits in x are not affected */
```

```
SET_BIT(x,y); /* Set all bits in x corresponding to the bits set in y. The other bits in x are not affected. */
```

```
CLEAR_FLAG(x,y); /* USE this routine ONLY for clearing flags. Clears flags in x corresponding  
to the bits set in y. Other flags in x are not affected. */
```

vectors.h

This file defines all the M68HC11 interrupt vectors generically. User interrupt service routines must equate to the predefined interrupt generic names.

Miscellaneous ICC11 Features

You can embed assembly code into your C code, write interrupt drivers, build libraries, and manage code in ways discussed in university C courses. See the ICC11 manual in Benton 335B for further information.

Creating, Compiling and Downloading Code in ICC11 Version 5

When you start up ICC11 version 5, you see a blank Status Window. Select NEW or OPEN from the file menu to bring up a context sensitive text editor. You can use this text editor to generate or edit your source code. Once you are satisfied with your source code, select COMPILE TO EXECUTABLE from the compile menu. You should see periods and the "SUCCEED" message if your program is correct. This process generates the *filename.s19* which is ultimately downloaded to the target computer. To download, select the TERMINAL option of the target menu and press the BOOTSTRAP DOWNLOAD button (or the ASCII DOWNLOAD button for those loading into a BUFFALO-enabled system) to download the file to the target computer. You may also use the high speed down loader, PCBUG11, or even BUFFALO to accomplish the download as your application dictates. There are many other very useful features in ICC11 which you are encouraged to try and become familiar with. Double clicking an error message with the format: `!E<filename>(<line #>):...` brings up an editor with the file loaded and the cursor pointing to the offending line.

Files in ICC11

Input Files-the following file types are understood by ICC11:

- ❖ *filename.c*
a C source file (text).
- ❖ *filename.i*
a preprocessed C source file.
- ❖ *filename.s*
an assembly source file.

Output Files-the following file types are created by ICC11 with the following extensions:

- ❖ *filename.o*
a relocatable object file.
- ❖ *filename.s19*
a Motorola S record file. This is the executable file that you load into your target system.
- ❖ *filename.ihx*
an INTEL hex record file (optional).
- ❖ *filename.a*
a library archive file.
- ❖ *filename.lis*
an assembler-generated listing file.
- ❖ *filename.mp*
a linker-generated map file.
- ❖ *filename.lst*
a linker-generated listing file with all your program's .lis files concatenated with final addresses.
- ❖ *filename.map*
a linker-generated map file compatible with the P&E Microcomputer's Inc. debug format.

Library Files

- ❖ *libc11.a*
contains a subset of the ANSI C library functions and HC11 specific functions
- ❖ *libfp11.a*
contains a version of the *printf* that can print out floating point numbers

Assembly Code Interface

The compiler prepends an underscore to all global functions and variables. Thus, to access such objects in assembly code, you must prepend an underscore to the name of the object. For example, if you have a global variable, say `LEFT_MOTOR_SPEED`, you can use:

```
LDD    _LEFT_MOTOR_SPEED
```

Arguments are promoted to their natural sizes, and pushed from right to left on the stack, **except** for the first argument which is passed in the D register unless the first argument is a structure, floating point, or long. The natural size for integer types, including char and short, is int. The natural size for floating point type is double, and the natural size for a structure is simply the size of the structure. See pp. 34 of the ICC11 manual for further details.

To embed arbitrary assembly statements in your C program use the following format:

```
asm( "asm string");          asm("ldaa BOO\n" "adda CHU\n" "staa ON_YOU")
```

You must be careful to preserve the X register in an embedded asm since it is the frame pointer. To access on chip peripherals efficiently you can use pointer (memory) indirection because on chip ROM, RAM and I/O registers are addressed like all memory addresses. For example to access PORTA at address 0x1000 you can use:

```
*(volatile unsigned char *)0x1000=1;          /* pointer indirection used to write a 1 to PORT A */  
-or-  
PORTA=1;                                       /* if you included <hc11.h> */
```

```
/* Read the content of PORT A */  
i = *(volatile unsigned char *)0x1000; /* uses pointer indirection to read PORT A */  
-or-  
i = PORTA; /* if you included <hc11.h> */
```

Some languages (C, for instance) require you to use routines such as POKE and PEEK to access memory. These are not necessary in ICC11 since C allows memory indirection as shown above.

ICC11 Bit Twiddling

Accessing and using M68HC11 on-chip resources require manipulation of bits in the internal register set of the micro-controller. In most cases, you can directly write these functions using standard C constructs implemented in ICC11. For example:

```
#include <hc11.h>  
.  
.  
DDRC = 0x0F; /* Port C bits {0,1,2,3} are set to output; bits {4,5,6,7} are input */  
DDRC &= ~0x0F; /* Port C, turn off bits {0,1,2,3} as outputs */  
TFLG2 = 0x80; /* Clear the TOF bit */  
TMSK2 |= 0x80; /* Enable TOI bit */
```

Writing Interrupt Routines with ICC11

To implement an interrupt handler as a C function, you must do the following:

1. You must declare the function as an interrupt handler so that the compiler will generate an rti (return from interrupt) instead of an rts (return from subroutine) instruction. You do this by using the following pragma:
#pragma interrupt_handler<name>
2. You must assign the function address to the interrupt vector in vectors.c and include this version in your program.

An example using OC5 to generate a PWM waveform is in the ICC11 manual on pp. 37-38 or see, for example, the file motorme.c in the ICCME\libsrtj directory.

Volatile Type Qualifier

In C, you use the volatile type qualifier to tell the compiler that a data item may change in ways that are not apparent to the compiler. For example, memory-mapped peripheral registers are good examples. Normally, the compiler may optimize certain memory addresses away. For example, if you are reading a location immediately after writing to it, the compiler optimization routine may reuse the temporary value in a register instead of reloading it. For memory mapped registers, the effect is incorrect if the read is not done. Therefore, peripheral registers must be accessed using the volatile type qualifier, which is the case if you use the “defines” provided in the header files.

ICC11 Simple Code Examples

The following very simple programs illustrate the ideas discussed in this primer and give examples of ICC11 code in action.

Example 1

This example program awaits for the bumper to be pressed in a TJPRO™ robot and then turns both motors, one forward and one in reverse.

```
/******  
*                                                                 *  
* Title:                Mottst1.c                               *  
* Programmer:          A. Antonio Arroyo {from code by Ivan Zapata & K. Doty}*  
* Date:                February 12, 2008                       *  
* Version:             1.0                                     *  
*                                                                 *  
* Description:                                                *  
*   The robot will wait for the bumper to be sensed and turns on *  
*   both motors, the right motor forward, the left motor in reverse *  
******/  
  
/***** Includes *****/  
  
#include <analog.h>  
#include <motortjp.h>  
#include <clocktjp.h>  
#include <vectors.h>  
#include <isrdecl.h>  
#include <stdio.h>  
  
/***** End of Includes *****/  
  
/***** Constants *****/  
  
#define LEFT_MOTOR      0  
#define RIGHT_MOTOR    1  
#define MAX_SPEED      100  
#define ZERO_SPEED     0  
#define BUMPER         analog(0)  
  
/***** End of Constants *****/  
  
/***** Main *****/  
void main(void)  
{  
  
    init_analog();  
    init_motortjp();  
  
    while(BUMPER<120); /*Press the rear bumper to start the program*/  
    while(1)  
    {  
        motorp(RIGHT_MOTOR, MAX_SPEED);  
        motorp(LEFT_MOTOR, -MAX_SPEED);  
    }  
}  
/***** End of Main *****/
```

Example 2

This example program awaits for the bumper to be pressed in a TJPRO™ robot and then turns both motors using speed values supplied by the user.

```
/* **** */
*
* Title:          Mottst2.c
* Programmer:    A. Antonio Arroyo {from code by Ivan Zapata & K. Doty}*
* Date:         February 12,2008
* Version:      1.0
*
* Description:
*   The robot will wait for the bumper to be sensed and turn on
*   both motors after reading a right and a left motor speed value.
* **** */

/***** Includes *****/
#include <analog.h>
#include <motortjp.h>
#include <clocktjp.h>
#include <serialtp.h>
#include <vectors.h>
#include <isrdecl.h>
#include <stdio.h>
/***** End of Includes *****/

/***** Constants *****/
#define LEFT_MOTOR      0
#define RIGHT_MOTOR    1
#define MAX_SPEED      100
#define ZERO_SPEED     0
#define BUMPER         analog(0)
/***** End of Constants *****/

/***** Main *****/
void main(void)
{
    int speedr, speedl;

    init_analog();
    init_motortjp();
    init_serial();

    while(BUMPER<120); /*Press the rear bumper to start the program*/
    while(1)
    {
        printf("Enter Right Motor Value \n");
        speedr=read_int();
        printf("Enter Left Motor Value: \n");
        speedl=read_int();
        motorp(RIGHT_MOTOR, speedr);
        motorp(LEFT_MOTOR, speedl);
    }
}
/***** End of Main *****/
```

Example 3

This example program awaits for the bumper to be pressed in a TJPRO™ robot and then turns on the IR emitters and senses and displays the two IR detector values, the bumper on Analog Port 0 and the value in Analog Port 1.

```

/*****
*
* Title:          CDstst.c
* Programmer:     A. Antonio Arroyo {from code by Ivan Zapata & K. Doty}*
* Date:          February 12, 2008
* Version:       1.0
*
* Description:
*   The robot will wait for the bumper to be sensed and turn on
*   the IRs and display the sensed value.
*****/

/***** Includes *****/
#include <analog.h>
#include <clocktjp.h>
#include <serialtp.h>
#include <stdio.h>
/***** End of Includes *****/

/***** Constants *****/
#define BUMPER          analog(0)
#define CDS             analog(1)
#define RIGHT_IR       analog(2)
#define LEFT_IR        analog(3)
#define TURN_ON_IRS    *(unsigned char *) 0x7000=0x07
/***** End of Constants *****/

void main(void)
/***** Main *****/
{
    int irdr, irdl;
    char clear[]= "\x1b\x5B\x32\x4A\x04"; /* clear screen */
    char place[]= "\x1b[1;1H";          /* Home cursor */

    init_analog(); /* Initialize the A/D System */
    init_clocktjp(); /* Initialize ms Clock */
    init_serial(); /* Initialize Serial Port */

    TURN_ON_IRS; /* Turn on 40KHz IR LED emitters */

    printf("%s", clear); /*clear screen*/
    printf("%s", place); /*home cursor*/

    printf("\tTitle\t\tIRTst.c\n"
           "\tProgrammer\tA. A. Arroyo\n"
           "\tDate\t\tFebruary 12, 2008\n"
           "\tVersion\t\t1\n\n");

    printf("*Hit Rear Bumper to Start*\n\tDisplay IR Sensor Readings\n");
    printf("Left IR    Right IR    Bumper    CDS cell\n");

```

```
/* Press the rear bumper to start the program*/  
while(BUMPER<120);  
while(1)  
{  
    irdl = LEFT_IR;  
    irdr = RIGHT_IR;  
    printf("\x1b[10;1H    \b\b\b\b\b%d",irdl);  
    printf("\x1b[10;13H   \b\b\b\b\b%d",irdr);  
    printf("\x1b[10;25H   \b\b\b\b\b%d",BUMPER);  
    printf("\x1b[10;37H   \b\b\b\b\b%d",CDS);  
    wait(50);  
}  
}  
/***** End of Main *****/
```

Example 4

This example program awaits for the bumper to be pressed in a TJPRO™ robot and then performs simple obstacle avoidance by sensing the IR and turning in a random direction if a threshold value is exceeded.

```
/*
 *
 * Title:          obsavtjp.c
 * Programmer:     A. Antonio Arroyo {from code by Ivan Zapata & K. Doty}
 * Date:          February 12, 2008
 * Version:       1.0
 *
 * Description:
 *   A very simple collision avoidance program for EEL-5666 using the
 *   TJ PRO robot. Obstacle avoidance can be obtained by:
 *   (1) reading each IR detector
 *   (2) turning away from any obstacle in its path if the IR value
 *       exceeds a threshold
 *
 *   For illustration purposes, if the TJ PRO bumper is sensed,
 *   the robot will back up, turn, and go on.
 *
 *   The robot will also wait for the bumper to be sensed
 *   to begin operation.
 */

/* Includes */
#include <analog.h>
#include <motortjp.h>
#include <clocktjp.h>
#include <serialtp.h>
#include <isrdecl.h>
#include <vectors.h>
#include <stdio.h>

/* End of Includes */

/* Constants */
#define LEFT_MOTOR      0
#define RIGHT_MOTOR    1
#define MAX_SPEED      100
#define ZERO_SPEED     0
#define AVOID_THRESHOLD 120
#define BUMPER         analog(0)
#define RIGHT_IR        analog(2)
#define LEFT_IR         analog(3)

/* End of Constants */

/* Prototypes */
void turn(void);

/* End of Prototypes */
```

```
void main(void)
/***** Main *****/
{
  int irdr, irdl, speedr, speedl;

  init_analog();
  init_motortjp();
  init_clocktjp();
  init_serial();

  *(unsigned char *) 0x7000=0x07; /* Turn on IR */

  while(BUMPER<120); /*Press the rear bumper to start the program*/

  while(1)
  {
    /* Step 1: read the IR detectors, and decide whether to turn to avoid any
    obstacles */
    irdr = RIGHT_IR;
    irdl = LEFT_IR;

    if (irdl > AVOID_THRESHOLD)
      speedr = -MAX_SPEED;
    else
      speedr = MAX_SPEED;
    if (irdr > AVOID_THRESHOLD)
      speedl = -MAX_SPEED;
    else
      speedl = MAX_SPEED;

    motorp(RIGHT_MOTOR, speedr);

    motorp(LEFT_MOTOR, speedl);

    /* This "if" statement checks the bumper. If the bumper is pressed, */
    /* Tj will back up, and turn. */

    if((BUMPER>10)&&(BUMPER< 120))
    {
      motorp(LEFT_MOTOR, -MAX_SPEED);
      motorp(RIGHT_MOTOR, -MAX_SPEED);
      wait(600);
      turn();
    }

    wait(35);
  }
}
/***** End of Main *****/
```

```
void turn(void)
/*****
 * Function: Will turn in a random direction for a "random" amount of
 *           time, dictated by the fast changine lower bits in
 *           mseconds().
 * Returns:  None
 *
 * Inputs
 *   Parameters: None
 *   Globals:   None
 *   Registers:  TCNT
 * Outputs
 *   Parameters: None
 *   Globals:   None
 *   Registers:  None
 * Functions called: motorp(), wait()
 * Notes:
 *****/
{
  int i;
  unsigned rand;

  rand = TCNT;

  if (rand & 0x0001)
    /*turn left*/
    {
      motorp(RIGHT_MOTOR, MAX_SPEED);
      motorp(LEFT_MOTOR, -MAX_SPEED);
    }
  else
    /*turn right*/
    {
      motorp(RIGHT_MOTOR, -MAX_SPEED);
      motorp(LEFT_MOTOR, MAX_SPEED);
    }

  i=(rand % 1024);
  if(i>250) wait(i); else wait(250);
}

/*****End Function turn *****/
```

Example 5

This example program adds a CDs cell “Stop” behavior to the obstacle avoidance behavior in Example 4. The robot awaits for the rear bumper to be pressed and then performs simple obstacle avoidance by sensing the IR and turning in a random direction if a threshold value is exceeded. If he CDs cell on top of the robot, connected via a voltage divider to Analog Port 1 is covered, the robot stops until the cell is uncovered.

```

/*****
*
* Title:          obcdstjp.c
* Programmer:     A. Antonio Arroyo {from code by Ivan Zapata & K. Doty}*
* Date:          February 12, 2008
* Version:       1.0
*
* Description:
*   A very simple collision avoidance program for EEL-5666 using the
*   TJ PRO robot. Obstacle avoidance can be obtained by:
*   (1) reading each IR detector
*   (2) turning away from any obstacle in its path if the IR value
*       exceeds a threshold
*   (3) Uses a CDS cell in Analog Channel 1 to stop the robot
*   For illustration purposes, if the TJ PRO bumper is sensed,
*   the robot will back up, turn, and go on.
*
*   The robot will also wait for the bumper to be sensed
*   to begin operation.
*****/

/***** Includes *****/
#include <analog.h>
#include <motortjp.h>
#include <clocktjp.h>
#include <serialtp.h>
#include <isrdecl.h>
#include <vectors.h>
#include <stdio.h>
/***** End of Includes *****/

/***** Constants *****/
#define LEFT_MOTOR      0
#define RIGHT_MOTOR     1
#define MAX_SPEED      100
#define ZERO_SPEED     0
#define AVOID_THRESHOLD 120
#define BUMPER         analog(0)
#define CDS             analog(1)
#define RIGHT_IR       analog(2)
#define LEFT_IR        analog(3)
/***** End of Constants *****/

/***** Prototypes *****/
void turn(void);
/***** End of Prototypes *****/

```

```
void main(void)
/***** Main *****/
{
  int irdr, irdl, cdsd, speedr, speedl;

  init_analog();
  init_motortjp();
  init_clocktjp();
  init_serial();

  *(unsigned char *) 0x7000=0x07; /* Turn on IR */

  while(BUMPER<120); /*Press the rear bumper to start the program*/

  while(1)
  {
    /* Step 1: read the IRs & CDS, decide whether to stop or turn to avoid
    obstacles */
    irdr = RIGHT_IR;
    irdl = LEFT_IR;
    cdsd = CDS;

    if (cdsd > 100) {speedl=0; speedr=0;}
    else {
      if (irdl > AVOID_THRESHOLD)
        speedr = -MAX_SPEED;
      else
        speedr = MAX_SPEED;
      if (irdr > AVOID_THRESHOLD)
        speedl = -MAX_SPEED;
      else
        speedl = MAX_SPEED;
    }
    motorp(RIGHT_MOTOR, speedr);
    motorp(LEFT_MOTOR, speedl);

    /* This "if" statement checks the bumper. If the bumper is pressed, */
    /* Tj will back up, and turn. */

    if((BUMPER>10)&&(BUMPER< 120))
    {
      motorp(LEFT_MOTOR, -MAX_SPEED);
      motorp(RIGHT_MOTOR, -MAX_SPEED);
      wait(600);
      turn();
    }

    wait(35);
  }
}
/***** End of Main *****/
```

```
void turn(void)
/*****
 * Function: Will turn in a random direction for a "random" amount of *
 *           time, dictated by the fast changine lower bits in *
 *           mseconds(). *
 * Returns: None *
 * *
 * Inputs *
 * Parameters: None *
 * Globals: None *
 * Registers: TCNT *
 * Outputs *
 * Parameters: None *
 * Globals: None *
 * Registers: None *
 * Functions called: motorp(), wait() *
 * Notes: *
 *****/
{
  int i;
  unsigned rand;

  rand = TCNT;

  if (rand & 0x0001)
  /*turn left*/
  {
    motorp(RIGHT_MOTOR, MAX_SPEED);
    motorp(LEFT_MOTOR, -MAX_SPEED);
  }
  else
  /*turn right*/
  {
    motorp(RIGHT_MOTOR, -MAX_SPEED);
    motorp(LEFT_MOTOR, MAX_SPEED);
  }

  i=(rand % 1024);
  if(i>250) wait(i); else wait(250);
}

/*****End Function turn *****/
```