

UNIVERSITY OF FLORIDA
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

EEL 5666C
INTELLIGENT MACHINES DESIGN LAB

Final Report

Date:	08/10/04
Student Name:	Robert Lee
TAs:	William Dubel Max Koessick
Instructors:	A.A. Arroyo E. Schwartz

TABLE OF CONTENTS

<u>Subject</u>	<u>Page</u>
I. Abstract -----	2
II. Executive Summary -----	3
III. Introduction -----	4
IV. Integrated System -----	5
a. Figure 1.1 -----	6
V. Mobile Platform -----	7
a. Figure 1.2a -----	7
b. Figure 1.2b -----	8
VI. Actuation -----	8
VII. Sensors -----	9
VIII. Behaviors -----	10
IX. Experimental Layout and Results -----	11
a. Table 1.1 -----	12
b. Table 1.2 -----	12
c. Figure 1.3 -----	12
X. Conclusions -----	13
XI. Documentation -----	13
XII. Appendices -----	14
a. Code -----	14
I. ABSTRACT	

The importance of having a good robot design is whether or not it can be practically feasible, while at the same time resourceful. In the course of determining basic functions for my robot, first, I had to figure out all of the possible limitations that could obstruct my goals. However, the biggest constraints for most peoples' projects always seem to arrive in the forms of time, energy and/or money; three of which I have neither of. Therefore, reusing old parts and implementing pre-built parts seemed the most cost/time effective method.

The idea of a "Fire-fighting" robot, with the capability of intelligent decision-making, sparked an interest in me as early as a year ago. Over the course of this summer semester, my goal is to make this "idea" a practical reality. Many attempts have been previously made upon this similar concept, and I plan to conduct improvements where I see fit. My overall goal is to basically develop a more flexible model that will conduct the same old routines in a more effective manner.

II. EXECUTIVE SUMMARY

Over the summer (2004), I was part of the IMDL group, directed by Drs. Eric Schwartz and Antonio Arroyo. I was actually able to design and create an autonomously moving and “fire-detecting” robot that doused flames (...or anything in its path). Throughout the summer, I encountered plenty of setbacks and suffered from quite a few “explosions” (human-error of course). But, that was all done in the process of learning and at a price (\$\$\$). Additionally, I got the chance to interface several different hardware platforms, and integrated them into one package. Prior to this experiment, an application of this type seemed a little difficult. Now, however, I feel I should and will improve upon this design in the months ahead. Overall, my robot design functioned “mediocre”. I think it basically did the job, but nothing else. There was room for several improvements to be made. First and foremost, was the lack of power and calibration from my “driving” motors. They were simply not designed for the purpose I had in mind. Second, the chassis could not provide enough cabinet space for all of the necessary components to work “error-free”. After these two, there were several others to follow. In contrast, there were actually some positive things that came out of this whole project. First, my programming code is pretty much universal, and I had the opportunity to create some very efficient and useful functions (assembly language). These can be used in future applications and/or versions of this project. Also, I got the opportunity to learn about PICs. Actually, these microprocessors are incredible, and I highly recommend becoming familiar in their broad range of uses, and their simple register space and instruction set.

III. INTRODUCTION

A problem that exists to this day is that people directly interact with fires in the process of extinguishing them. A large number of injuries and fatalities occur every year because of this interaction. A solution to this problem is to allow an intelligent robot to complete this risky task for them. However, due to practical reasons and particular time constraints (being a student and employed), I will be limited to a less flexible robot (in terms of its intelligence and ability to navigate through “rough” environments).

My “Fire-fighting” robot is dubbed “FLAME”; an acronym that stands for: *Fire Locator And Mechanical Extinguisher*. The very nature of this robot is pretty much self-explanatory. It will roam around a small-sized arena and avoid obstructions in its path, while simultaneously reading its environment for an open flame. If and when an open flame has been detected, it will try to align itself so an extinguishing mechanism can be activated. Then, (business as usual) the robot will continue its search for another flame.

In the course of reading this report, I definitely plan to inform the viewer of the details in my designing of this autonomous “Fire-fighting” robot. The report is broken down into several sections regarding the required hardware and software (code) and experimental results that concluded. A “walk-through” of the steps and building process will be fully provided to allow for easy duplication and/or upgrading for future projects. Specifically, these steps include sections from: Integrated System, Mobile Platform, Actuation, Sensors, Behaviors, and

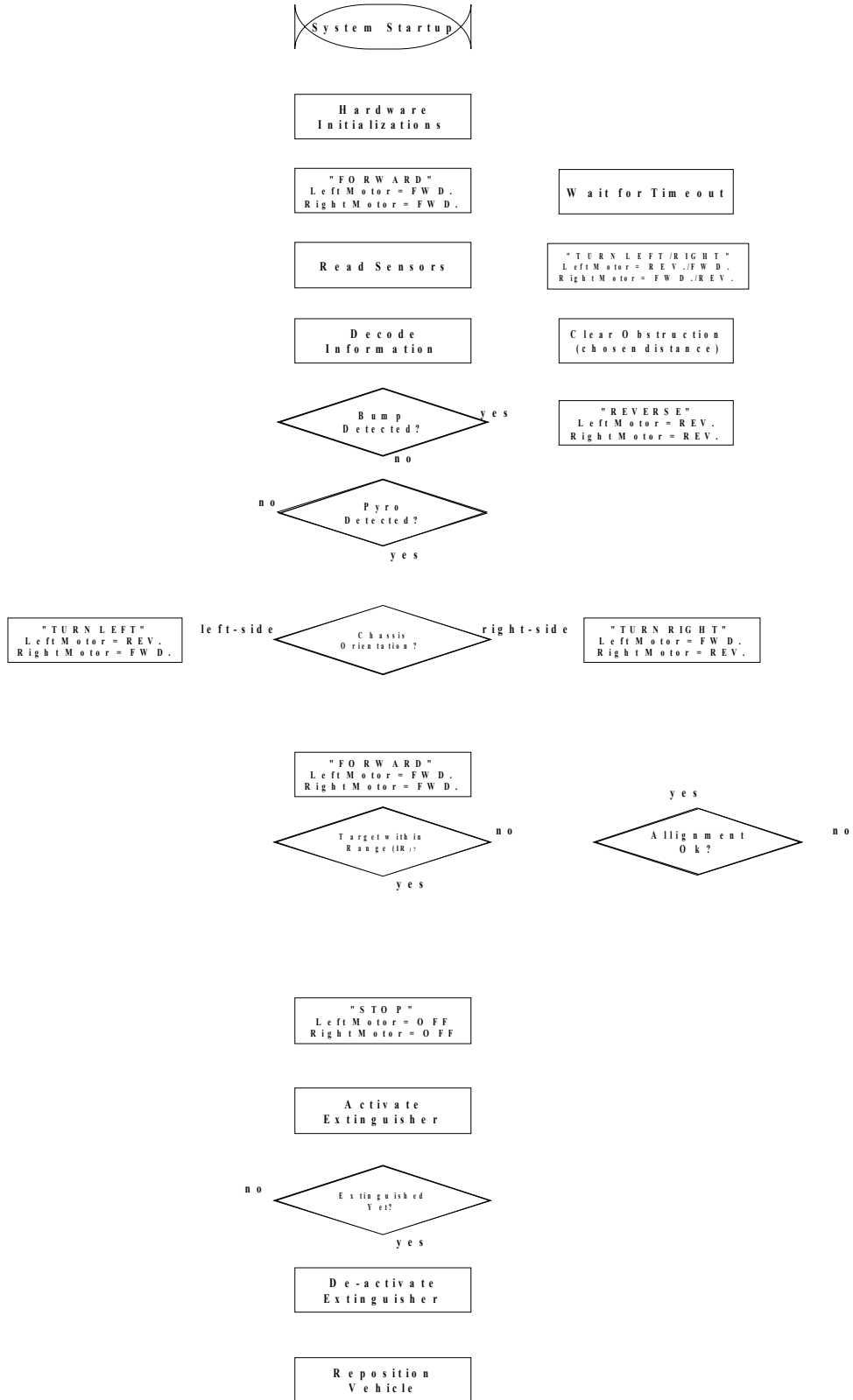
Experimental Layout/Results. Most likely though, this report will only serve one purpose: as a reference to the “work-arounds” for many of the problems I encountered throughout the experiment.

IV. INTEGRATED SYSTEM

The system consists of a set number of routines or patterns to carry out the tasks of a simple “Fire-fighting” robot. Figure 1.1 shows a general flowchart of the entire procedure. A run-through of the procedure illustrates a maximum of seven decision-making blocks to test different types of sensors and drive motors.

Depending on each of their outcomes, different parameters will be set. As a result, the robot should be able to accomplish its task, and continue in an endless loop cycle.

Figure 1.1



V. MOBILE PLATFORM

The chassis to “FLAME” has been built several times, through the use of the T-tek machine and CAD software. The framework is built from 1/8th inch wood, and the motors, wheels and tracks can be mounted easily. The entire chassis is actually quite small, and the guts of it are packed (stuffed) with wires. Basically, it could have been slightly larger to aid better debugging/testing. At the end of the semester (3rd build), the chassis cabinet had all sorts of wires dangling and interfering with each other. It did not look very cosmetic at all! Figures 1.2a and 1.2b below, will give you the dimensions of the chassis.

Figure 1.2a

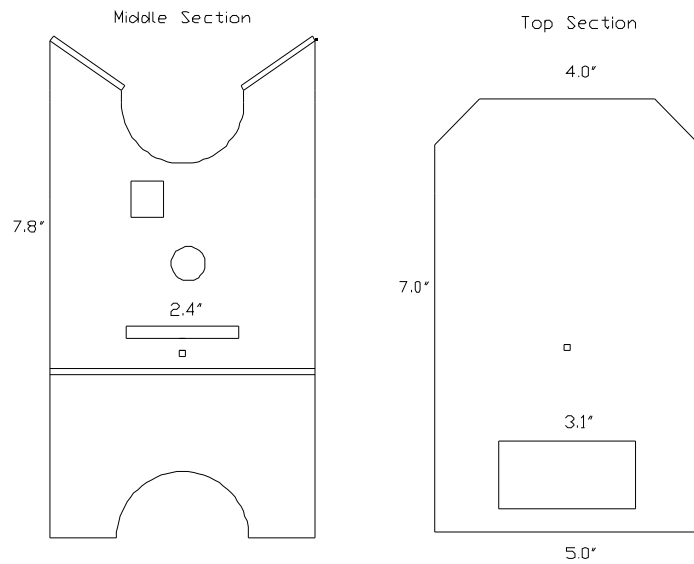
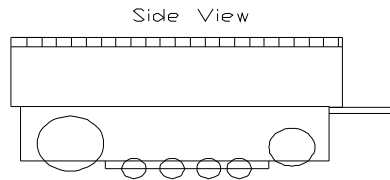


Figure 1.2b



VI. ACTUATION

F.L.A.M.E.'s internals consist of a many components: the chassis, microcontroller, battery packs, water reservoir and crank. The chassis will have two tracks for mobility (similar to a tank's design). Two motors mounted in the rear will control both tracks independently (to allow for all possible movement). These motors will drive two wheels, and those wheels will drive their corresponding tracks. Stationary wheels (placed slightly lower than the "driving wheels") should make clean traction with the surface. The microcontroller and battery packs are physically located on the rear or the top to prevent water damage. The water reservoir and crank are fitted to the center of the chassis (located more towards the front). Three IR cans were mounted on the front. The risk of water damage to electrical components must definitely be considered.

The majority of my design has probably been in implementing control over the motors. Originally software control in the microcontroller's memory would make it possible to move the chassis, via Pulse Width Modulation. However, after several motors were stripped due to current overloads, that design had to be

scrapped. Basically, the H-Bridge (originally intended for use with these motors), had to be supplied with at least (10V to 12V) for those chips (LMD18200) to even operate. A duty cycle could reduce the voltage requirement on the load.

However, the problem was not that the voltage could not be stepped-down. The problem stemmed from the current that was generating from my power source.

As a result, an alternate “relay” system (not recommended) was improvised.

Each motor would be connected to a bus-line, where several relay outputs would be fed into. Voltages on those relays were actually +3.0V (fast-forward), +1.5V (slow-forward) and -1.5V (slow-reverse). Then, it was basically up to software to set/clear the necessary port-C bits to enable directions and speeds. Speeds were basically preset, and only the power-sources could change them. Also, it would probably be good to include a transition delay period to allow any relays to be shut down before turning on any others (power-source collision).

VII. SENSORS

The main sensor in detecting fire in a small room will be a “Pyro-Electric” sensor. Hamamatsu’s “UVTron” is a recommended choice for this purpose. It comes complete with a bulb and a processor board for filtering pulses. I got the idea to use this device from a previous students’ “Fire-fighting” robot design. For application purposes, an interrupt system will sample and record the number of pulses generated during a period of time (Timer-0). If the number of pulses exceeds the desired level, then a special register bit will be set (cleared if pulses generated is insufficient).

The only other sensors in my system are the IR cans (Sharp GP2Y0A02YK). These must be configured with the A/D subsystem on Port-A of my processor. The analog values can be recorded once the initializations (delay-time needed) are complete. Then, it is just up to software to enable/disable channels to read from. In my particular design, I am using three IR cans. Therefore, in code I must set the correct configuration (for each channel) before reading from any of them.

VIII. BEHAVIORS

The central “initial” behavior for my “Fire-fighting” robot is to circle a path while detecting fire. Then, when a fire has been detected, it should maneuver itself toward the target (repositioning in the case of going off-course). Finally, when the target has reached certain proximity, the robot should come to a halt and the extinguishing system should take over. The robot can accomplish this by setting or clearing flags in memory with an interrupt system. Significant overhead can be reduced from this method by limiting the bulk of the code to subroutines from “main()”. A particular register in memory (Pyro-bit0) was actually used for my fire detection process. Setting or clearing this register-bit communicates an “on/off” state, which will allow selected subroutines to be called-up. The entire code (roughly 1500 including several unused functions) can be viewed in the Appendix of this report.

IX. EXPERIMENTAL LAYOUT AND RESULTS

Typical interface data for my pyro-sensor (Hamamatsu UVTron) consists of a set number of pulses, given the proximity of an open flame to the sensor. For example, a higher number of pulses suggest an open flame is nearer to the sensor, and a lower number for a farther distance. Actual data is given below for this device (Table 1.1). Note that the actual number of counted pulses will depend on the Timer-0 delay time (threshold). The IRs (Sharp GP2Y0A02YK) operated differently than the pyro-sensor, in that the interrupt system was not relied upon. The IRs utilized the A/D subsystem (input capture) and gave the proximity of an object relative to a magnitude. For example, a higher magnitude represented a “closer” object, and a small magnitude for a “farther” object. This information is available in Table 1.2. Note that a linear representation of (magnitude vs. distance) varies, and may become unstable at very close proximities (resulting in a non-linear curve). This can be viewed in Figure 1.3.

Table 1.1

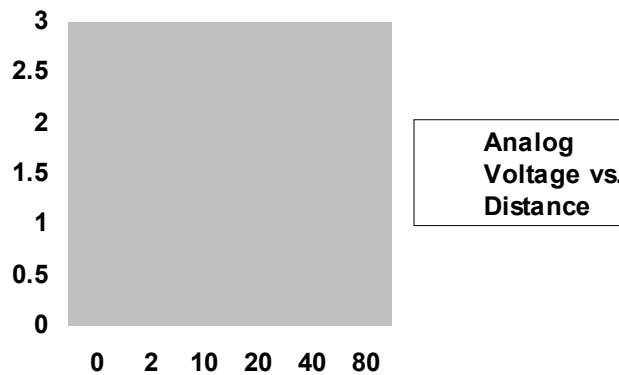
Degrees from Center	UVTron (# pulses)	Timer-0 Delay time	Proximity
0	12	1.0 s	10”
+/- 45	12	1.0 s	10”
+/- 90	11	1.0 s	10”
0	12	1.0 s	24”
+/- 45	11	1.0 s	24”
+/- 90	8	1.0 s	24”
0	12	1.0 s	72”
+/- 45	11	1.0 s	72”

+/- 90	7	1.0 s	72"
--------	---	-------	-----

Table 1.2

Distance (cm)	Voltage (V)	Hex value
2	0.70	h'7C'
10	2.60	h'209'
20	1.40	h'1A7'
40	0.75	h'97'
80	0.50	h'5F'

Figure 1.3



X. CONCLUSIONS

Over the course of this semester I have reached two conclusions. One is that my robot did work. Second is that there is much needed room for improvement.

Much of my problems resulted from incompatibilities with parts. The motors on my vehicle could definitely be upgraded. They were just not acceptable in order

to provide a sustained and accurate means of mobility. On the other end, there was some careless human-error that could have saved me some extra money in the long run. I burned quite a few PIC chips, simply by connecting ports incorrectly. A good bit of advice is to spend some extra time just reviewing a circuit. It will save time and money in the long run. On a different note, future work on this application is a most likely consideration. I think that in the spring of 2005, I could have a much better design functioning to show off at the next media day for IMDL.

XI. DOCUMENTATION

Programming and Customizing PICmicro Microcontrollers. Predko, Mike. 2nd Edition. McGraw-Hill. c.2001.

Online Website: <<http://www.microchip.com>> . "PIC (16f877/16f877a) documentation and code examples".

XII. APPENDICES

Code for this experiment (below):

```
*****
;PROG:          "FLAME.asm"
;VERSION:       1.0
;AUTHOR:        Robert Lee
;DATE:          08/09/04

;Function:
; This code will provide drivers + a set routine for "FLAME ver.1.0".
; IRs, an LCD, 1.5-3V motors, and pyro-sensor are included to allow
; for a "fire-fighting" robot design. Presently, the code only
```



```

                                goto    DEBUG1      ;Bingo! "REPEAT ALLIGNMENT PHASE"
                                goto    RESPONSE_1    ;Try Again "KEEP ALLIGNING"

RESPONSE_2:
                                call    LEFT_1
                                call    DELAY_TICK
                                call    STOP
                                call    AD_CH0
                                call    AD0_READ_FAR
                                btfsc   AD_BIT0,0
                                goto    DEBUG1      ;Bingo! "REPEAT ALLIGNMENT PHASE"
                                goto    RESPONSE_2    ;Try Again "KEEP ALLIGNING"

```

```

DEBUG2:
                                call    STOP        ;stop motors
                                call    SPLASH_FOUND ;print result to display
                                call    DC_MOTOR     ;start "extinguisher" motor
                                goto    here        ;done!....(for this project)

;))))))))))))))))))))))))))))))))))))))))))
;))))))))))))))))))))))))))))))))))))))))))
;))))))))))))))))))))))))))))))))))))))))))

```

```

here:      goto    here        ;Safety Loop (no entry into subroutines)

```

```

;@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
;@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

```

```

;-----
;* * *INT. SERVICE ROUTINES* * *
;-----

```

```

ISR:
                                btfsc   INTCON,T0IF  ;time-out has occurred for Timer-0 int.
                                goto    ISR_TIMER0

                                btfsc   INTCON,RBIF  ;pulse has been detected for Port-B (change) int.
                                goto    ISR_PULSE

                                goto    end_isr      ;Nevermind...It was nothing!

```

```

;-----

```

```

ISR_PULSE:
                                bcf     INTCON,RBIF  ;clears RB.interrupt flag
                                incf    PULSE_L     ;increments the counter (pulses)
                                btfsc   STATUS,Z
                                goto    inc_pul2
done_pb:   goto    end_isr
inc_pul2:  incf    PULSE_H
                                goto    done_pb

```

```

;-----

```

```

ISR_TIMER0:
                                tstf   TIME_L     ;check if TIME1 is 'zero' or not
                                skpz
                                goto    dec1        ;if not zero, then decrement
                                goto    test2      ;otherwise skip the 'goto'

dec1:
                                decf   TIME_L,F
                                goto    end_timer

test2:
                                tstf   TIME_H
                                skpz
                                goto    dec2
                                goto    pulse_out ;if both registers are 'clear', then pulse!

dec2:

```



```

;- -----
STOP:    ;0V to motors (no movement)
        banksel PORTC
;*****
        bCf      PORTC,L_30      ;turn everything "off"
        bCf      PORTC,R_30
        bCf      PORTC,L_15
        bCf      PORTC,R_15
;*****
        return

;- -----
LEFT_1: ;+1.5V to motors (left direction)
        banksel PORTC
;*****
        bCf      PORTC,R_30      ;turn-off +3.0V
        bCf      PORTC,L_30

        call     DELAY_1SEC      ;TRANSITION...(needed!!!)

        bSf      PORTC,R_15      ;turn-on +1.5V
        bCf      PORTC,L_15
;*****
        return

;- -----
RIGHT_1: ;+1.5V to motors (right direction)
        banksel PORTC
;*****
        bCf      PORTC,R_30      ;turn-off +3.0V
        bCf      PORTC,L_30

        call     DELAY_1SEC      ;TRANSITION...(needed!!!)

        bCf      PORTC,R_15      ;turn-on +1.5V
        bSf      PORTC,L_15
;*****
        return

;- -----
LEFT_2: ;+3.0V to motors (left direction)
        banksel PORTC
;*****
        bCf      PORTC,R_15      ;turn-on +1.5V
        bCf      PORTC,L_15

        call     DELAY_1SEC      ;TRANSITION...(needed!!!)

        bSf      PORTC,R_30      ;turn-on +1.5V
        bCf      PORTC,L_30
;*****
        return

;- -----
RIGHT_2: ;+3.0V to motors (right direction)
        banksel PORTC
;*****
        bCf      PORTC,R_15      ;turn-on +1.5V
        bCf      PORTC,L_15

        call     DELAY_1SEC      ;TRANSITION...(needed!!!)

        bCf      PORTC,R_30      ;turn-on +1.5V
        bSf      PORTC,L_30
;*****
        return

```



```

; "USE FOR DEBUGGING PURPOSES - ONLY!"
;
;*****
;               call    WRITE_AD           ;write result to LCD
;*****
;
;               movlw   h'1'
;               subwf   AD_H_TEMP
;               skpc
;               goto    next_far
;               goto    good_dist0far
;
next_far:
;               movlw   h'50'               ;DEBUGGER-USE ONLY!!!
;               subwf   AD_L_TEMP
;               skpc
;               goto    bad_dist0far
;               goto    good_dist0far
;
bad_dist0far:
;               bCf     AD_BIT0,0
;               goto    done_a0far
;
good_dist0far:
;               bSf     AD_BIT0,0           ;OUTPUT-BIT IS NOW SET,
;               B/C ALL CONDITIONS ARE TRUE!
;
done_a0far:
;               return
;
;-----
AD1_READ:      ;Reads A/D-bit1 (LEFT) -> sets/clears bit accordingly
;
;               banksel PORTA
;               bsf     ADCON0,GO_DONE      ;set "go-bit" to begin the
;
conversion
ad_wait1:
;               btfsc  ADCON0,GO_DONE      ;wait for "complete-flag" to be set
;               goto   ad_wait1           ;....then proceed.
;
;               BANKSEL TRISC
;               movfw  ADRESL              ;move A/D result into 'WREG'
;               BANKSEL PORTC
;               movwf  AD_L                ;RESULT(LOW)
;               movwf  AD_L_TEMP
;               movfw  ADRESH
;               andlw  b'00000011'
;               movwf  AD_H                ;RESULT(HIGH) for "WRITE_AD"
;
subroutine
;               movwf  AD_H_TEMP           ;result for testing distance of IR to target (2
;               = 4 inches)
;
;*****
; "USE FOR DEBUGGING PURPOSES - ONLY!"
;
;*****
;               call    WRITE_AD           ;write result to LCD
;*****
;
;               movlw   h'1'
;               subwf   AD_H_TEMP
;               skpc
;               goto    next_1
;               goto    good_dist1
;
next_1:
;               movlw   h'10'
;               MOVLW  H'50'               ;DEBUGGER-USE ONLY!!!
;               subwf   AD_L_TEMP
;               skpc
;               goto    bad_dist1
;               goto    good_dist1
;
bad_dist1:

```

```

                                bCf      AD_BIT1,0
                                goto     done_a1
good_dist1:
                                bSf      AD_BIT1,0          ;OUTPUT-BIT IS NOW SET,
                                B/C ALL CONDITIONS ARE TRUE!
done_a1:
                                return

;-----
AD2_READ:      ;Reads A/D-bit2 (RIGHT) -> sets/clears bit accordingly

                                banksel  PORTA
                                bsf      ADCON0,GO_DONE      ;set "go-bit" to begin the
conversion
ad_wait2:
                                btfsc   ADCON0,GO_DONE      ;wait for "complete-flag" to be set
                                goto     ad_wait2          ;....then proceed.

                                BANKSEL  TRISC
                                movfw   ADRESL              ;move A/D result into 'WREG'
                                BANKSEL  PORTC
                                movwf   AD_L                ;RESULT(LOW)
                                movwf   AD_L_TEMP
                                movfw   ADRESH
                                andlw   b'00000011'
                                movwf   AD_H                ;RESULT(HIGH) for "WRITE_AD"
subroutine
= 4 inches)
                                movwf   AD_H_TEMP          ;result for testing distance of IR to target (2

;*****
; "USE FOR DEBUGGING PURPOSES - ONLY!"
;
;                                call    WRITE_AD          ;write result to LCD
;*****

                                movlw   h'1'
                                subwf   AD_H_TEMP
                                skpc
                                goto     next_2
                                goto     good_dist2

next_2:
;
                                movlw   h'10'
                                MOVLW  H'50'              ;DEBUGGER-USE ONLY!!!
                                subwf   AD_L_TEMP
                                skpc
                                goto     bad_dist2
                                goto     good_dist2

bad_dist2:
                                bCf      AD_BIT2,0
                                goto     done_a2
good_dist2:
                                bSf      AD_BIT2,0          ;OUTPUT-BIT IS NOW SET,
                                B/C ALL CONDITIONS ARE TRUE!
done_a2:
                                return

;-----
DC_MOTOR:      ;Drives "Extinguisher" motor for a few seconds

                                banksel  PORTC
                                bsf      PORTC,5
                                call    DELAY_1SEC
                                call    DELAY_1SEC
                                bcf      PORTC,5
                                call    DELAY_1SEC

```

```

return
;-----
MOTOR_INIT:    ;Initialization of PORTC (motor bits)

                banksel  TRISC          ;PortC will be outputs for "motor relays"
                clrf     TRISC
                banksel  PORTC
                movlw    PY_INIT
                movwf    PY_COUNT

                return
;-----

AD_INIT: ;Initialization of PORTA + A/D subsystem (default=channel0)

                banksel  TRISA
                movlw    b'11111111'
                movwf    TRISA          ;configure PortA = "analog input" (low-byte)

                movlw    b'10000010'
                movwf    ADCON1

                banksel  PORTA

                clrf     AD_BIT0
                clrf     AD_BIT1
                clrf     AD_BIT2

                movlw    d'2'          ;initialize the pulse-count for A/D's
                movwf    AD_COUNT0
                movwf    AD_COUNT1
                movwf    AD_COUNT2

                movlw    b'10000001'  ;INITIALIZE A/D PORT-0
                                         ;other A/D ports can be initiated

elsewhere in "main()"
                movwf    ADCON0        ;configure A/D operation
                call     DELAY_TICK    ;short delay (for boot-up)

                return
;-----

AD_CH0:        ;Selects A/D - (channel 0)

                banksel  ADCON0
                bcf      ADCON0,CHS2
                bcf      ADCON0,CHS1
                bcf      ADCON0,CHS0

                call     DELAY_TICK

                return
;-----

AD_CH1:        ;Selects A/D - (channel 1)

                banksel  ADCON0
                bcf      ADCON0,CHS2
                bcf      ADCON0,CHS1
                bsf      ADCON0,CHS0

                call     DELAY_TICK

                return

```

```

;------
AD_CH2:           ;Selects A/D - (channel 2)

                banksel  ADCON0
                bcf      ADCON0,CHS2
                bsf      ADCON0,CHS1
                bcf      ADCON0,CHS0

                call     DELAY_TICK

                return

;------

TIMER_INIT:      ;Initializes the Timer-0 Subsystem

                banksel  PORTC
                movlw   TIME_INIT           ;(140 clocks)
                movwf  TIME_L
                movwf  TIME_H           ;TIME_H * TIME_L = 140^2 = (19600
clocks)

                banksel  OPTION_REG
                bsf      OPTION_REG,PS0    ;assign prescalar value (~2 seconds)
                bsf      OPTION_REG,PS1
                bsf      OPTION_REG,PS2
                bcf      OPTION_REG,T0CS   ;internal clock cycles
                bsf      OPTION_REG,PSA    ;disables prescalar for Timer-0

                bcf      INTCON,T0IF      ;pre-initialize T0-flag to zero
                bsf      INTCON,T0IE      ;enable Timer-0 system

                return

;------

INTR_INIT:       ;Initializes/Enables all interrupt systems

                banksel  TRISB
                movlw   b'10000001'      ;PortB (7:0) = "inputs"
                movwf  TRISB
                bcf      OPTION_REG,INTEDG ;(cleared) ints. to 'falling-edge'

                banksel  PORTB
                clrf   PULSE_L
                clrf   PULSE_H
                clrf   PYRO
                bcf      ON,h'0'          ;clear 'ON' (default)
                bcf      OFF,h'0'        ;clear 'OFF' (default)

                bsf      INTCON,INTE      ;enable the RB0/INT external interrupt
                bcf      INTCON,INTF      ;ensures that no external interrupt occurred
yet

                bsf      INTCON,RBIE      ;enable Port-B's interrupt system
                bcf      INTCON,RBIF      ;ensures that no PORTB interrupts occurred
yet

                bsf      INTCON,GIE      ;ENABLE ALL GLOBAL INTERRUPTS!

                return

;------

LCD_INIT:        ;Initializes the LCD for 4-bit data format

                banksel  TRISD           ;Switch to BANK-1
                clrf    TRISD           ;Initialize Ports (D&C) to "OUTPUT"

                banksel  PORTD           ;Switch to BANK-0
                movlw   h'00'           ;Initialize(RS,RW,ECLK == 0)
                movwf  PORTD

;------

```

```

movlw h'33' ;4-bit mode (enable), part_1
call WRITE_COMM ;COMMAND ($33)
;.....
movlw h'32' ;4-bit mode (enable), part_2
call WRITE_COMM ;COMMAND ($32)
;.....
movlw h'28' ;2 rows for 4-bit data,(small)
call WRITE_COMM ;COMMAND ($28)
;.....
movlw h'0C' ;display(on),cursor and blink(off)
call WRITE_COMM ;COMMAND ($0F)
;.....
movlw h'01' ;clear display, cursor to home
call WRITE_COMM ;COMMAND ($01)
;.....

return

;-----
WRITE_NUM_WORD: ;Writes a "double" number to the LCD

movwf TEMP_L ;save data to RAM
movwf TEMP_H
swapf TEMP_H

movlw h'C0' ;LCD cursor to second line
call WRITE_COMM

movfw TEMP_H ;check which character to output
andlw h'0F'
movwf NUM
call NUM_CHECK

movwf PORTD ;latch data to LCD
bsf PORTD,4
bsf PORTD,6
bcf PORTD,6
call DELAY_X ;short delay

movfw TEMP2 ;return original value to WREG
movwf PORTD ;place data on PORTD bus
bsf PORTD,4
bsf PORTD,6
bcf PORTD,6
call DELAY_X ;short delay

;.....

movfw TEMP_L ;check which character to output
andlw h'0F'
movwf NUM
call NUM_CHECK

movwf PORTD ;latch data to LCD
bsf PORTD,4
bsf PORTD,6
bcf PORTD,6
call DELAY_X ;short delay

movfw TEMP2 ;return original value to WREG
movwf PORTD ;place data on PORTD bus
bsf PORTD,4
bsf PORTD,6
bcf PORTD,6
call DELAY_X ;short delay

movlw h'10' ;cursor shift left (non-destructive)
call WRITE_COMM
movlw h'10' ;cursor shift left (non-destructive)
call WRITE_COMM

```

return

;-----

NUM_CHECK: ;Useful function for deciding the ASCII format for "WRITE_NUM_WORD"

chk_0: bcf STATUS,Z
 movfw NUM
 sublw h'0' ;Check for '0'
 btss STATUS,Z ;Check the Zero-bit for "a zero number"
 goto chk_1

 movlw h'0'
 movwf TEMP3
 goto set_3

chk_1: bcf STATUS,Z
 movfw NUM
 sublw h'1' ;Check for '0'
 btss STATUS,Z ;Check the Zero-bit for "a zero number"
 goto chk_2

 movlw h'1'
 movwf TEMP3
 goto set_3

chk_2: bcf STATUS,Z
 movfw NUM
 sublw h'2' ;Check for '0'
 btss STATUS,Z ;Check the Zero-bit for "a zero number"
 goto chk_3

 movlw h'2'
 movwf TEMP3
 goto set_3

chk_3: bcf STATUS,Z
 movfw NUM
 sublw h'3' ;Check for '0'
 btss STATUS,Z ;Check the Zero-bit for "a zero number"
 goto chk_4

 movlw h'3'
 movwf TEMP3
 goto set_3

chk_4: bcf STATUS,Z
 movfw NUM
 sublw h'4' ;Check for '0'
 btss STATUS,Z ;Check the Zero-bit for "a zero number"
 goto chk_5

 movlw h'4'
 movwf TEMP3
 goto set_3

chk_5: bcf STATUS,Z
 movfw NUM
 sublw h'5' ;Check for '0'
 btss STATUS,Z ;Check the Zero-bit for "a zero number"
 goto chk_6

 movlw h'5'
 movwf TEMP3
 goto set_3

chk_6: bcf STATUS,Z
 movfw NUM
 sublw h'6' ;Check for '0'
 btss STATUS,Z ;Check the Zero-bit for "a zero number"
 goto chk_7

```

movlw h'6'
movwf TEMP3
goto set_3

chk_7:
bcf STATUS,Z
movfw NUM
sublw h'7' ;Check for '0'
btfss STATUS,Z ;Check the Zero-bit for "a zero number"
goto chk_8

movlw h'7'
movwf TEMP3
goto set_3

chk_8:
bcf STATUS,Z
movfw NUM
sublw h'8' ;Check for '0'
btfss STATUS,Z ;Check the Zero-bit for "a zero number"
goto chk_9

movlw h'8'
movwf TEMP3
goto set_3

chk_9:
bcf STATUS,Z
movfw NUM
sublw h'9' ;Check for '0'
btfss STATUS,Z ;Check the Zero-bit for "a zero number"
goto chk_a

movlw h'9'
movwf TEMP3
goto set_3

chk_a:
bcf STATUS,Z
movfw NUM
sublw h'A' ;Check for 'A'
btfss STATUS,Z ;Check the Zero-bit for "a zero number"
goto chk_b

movlw h'1'
movwf TEMP3
goto set_4

chk_b:
bcf STATUS,Z
movfw NUM
sublw h'B' ;Check for 'B'
btfss STATUS,Z
goto chk_c

movlw h'2'
movwf TEMP3
goto set_4

chk_c:
bcf STATUS,Z
movfw NUM
sublw h'C' ;Check for 'C'
btfss STATUS,Z
goto chk_d

movlw h'3'
movwf TEMP3
goto set_4

chk_d:
bcf STATUS,Z
movfw NUM
sublw h'D' ;Check for 'D'

```

```

        btfss    STATUS,Z
        goto     chk_e

        movlw   h'4'
        movwf   TEMP3
        goto    set_4

chk_e:

        bcf     STATUS,Z
        movfw   NUM
        sublw   h'E'           ;Check for 'E'
        btfss   STATUS,Z
        goto    chk_f

        movlw   h'5'
        movwf   TEMP3
        goto    set_4

chk_f:

        bcf     STATUS,Z
        movfw   NUM
        sublw   h'F'           ;Check for 'F'
        btfss   STATUS,Z
        goto    set_3

        movlw   h'6'
        movwf   TEMP3
        goto    set_4

set_3:
        movlw   h'3'
        goto    done_num
set_4:
        movlw   h'4'
        goto    done_num

done_num:
        return

;-----

WRITE_COMM:    ;Writes commands to the LCD

        banksel PORTD
        movwf   TEMP1           ;saved in location #1

        swapf   TEMP1,W

        andlw   h'0F'
        movwf   PORTD
        bsf     PORTD,6
        bcf     PORTD,6         ;toggle PORTD-7; (E-clk)
        call    DELAY_X         ;short delay

        movfw   TEMP1
        andlw   h'0F'
        movwf   PORTD
        bsf     PORTD,6
        bcf     PORTD,6
        call    DELAY_X         ;short delay

        return

;-----

WRITE_DATA:    ;Writes data to the LCD

        banksel PORTD
        movwf   TEMP1           ;saved in location #1

        swapf   TEMP1,W
        andlw   h'0F'
        movwf   PORTD

```



```

movwf    TEMP_L
swapf    TEMP_H

movfw    TEMP_H
andlw    h'0F'
movwf    NUM
call     NUM_CHECK

movwf    PORTD
bsf      PORTD,4
bsf      PORTD,6
bcf      PORTD,6
call     DELAY_X           ;short delay

movfw    TEMP3           ;return original value to WREG
movwf    PORTD           ;place data on PORTD bus
bsf      PORTD,4
bsf      PORTD,6
bcf      PORTD,6
call     DELAY_X           ;short delay

;.....

movfw    TEMP_L
andlw    h'0F'
movwf    NUM
call     NUM_CHECK

movwf    PORTD
bsf      PORTD,4
bsf      PORTD,6
bcf      PORTD,6
call     DELAY_X           ;short delay

movfw    TEMP3           ;return original value to WREG
movwf    PORTD           ;place data on PORTD bus
bsf      PORTD,4
bsf      PORTD,6
bcf      PORTD,6
call     DELAY_X           ;short delay

movlw    h'10'           ;cursor shift left (non-destructive)
call     WRITE_COMM
movlw    h'10'           ;cursor shift left (non-destructive)
call     WRITE_COMM
movlw    h'10'           ;cursor shift left (non-destructive)
call     WRITE_COMM
movlw    h'10'           ;cursor shift left (non-destructive)
call     WRITE_COMM

return

;-----

DELAY_X:           ;short delay function for "hardware boot-ups"

;*****
; LOOP TIME = (50*50*30) = 75,000 clock cycles (15ms)
; ECLK = [20 MHz]
; MCLK = [5 MHz]
;*****
movlw    d'50'
movwf    COUNT1           ;initialize Counter #1 to d'255'
movwf    COUNT2           ;initialize Counter #2 to d'255'
movlw    d'30'
movwf    COUNT3           ;initialize Counter #3 to (USER DEFINED)

C1:      decfsz    COUNT1           ;decrement Counter #1 until "time out"
        goto     C1
        movlw    d'50'
        movwf    COUNT1           ;(Refill Counter #1)

```

```

C2:                decfsz  COUNT2        ;decrement Counter #2 until "time out" occurs
                   goto    C1            ;Loop back to "START" until "time out"
occurs

                   movlw   d'50'
                   movwf   COUNT2        ;(Refill Counter #2)
C3:                decfsz  COUNT3        ;decrement Counter #3 until "time out" occurs
                   goto    C1            ;Loop back to "START" until "time out"
occurs

                   return

;-----
DELAY_1SEC:        ;delay function = "roughly 1 second"....(or so!)

;*****
; LOOP TIME = (136^3) = 5/2 Million clock cycles (1s)
; ECLK = [20 MHz]
; MCLK = [5 MHz]
;*****
                   movlw   d'136'
                   movwf   COUNT1        ;initialize Counter #1 to d'255'
                   movwf   COUNT2        ;initialize Counter #2 to d'255'
                   movlw   d'136'
                   movwf   COUNT3        ;initialize Counter #3 to (USER DEFINED)
C4:                decfsz  COUNT1        ;decrement Counter #1 until "time out"
                   goto    C4
                   movlw   d'136'
                   movwf   COUNT1        ;(Refill Counter #1)
C5:                decfsz  COUNT2        ;decrement Counter #2 until "time out" occurs
                   goto    C4            ;Loop back to "START" until "time out"
occurs

                   movlw   d'136'
                   movwf   COUNT2        ;(Refill Counter #2)
C6:                decfsz  COUNT3        ;decrement Counter #3 until "time out" occurs
                   goto    C4            ;Loop back to "START" until "time out"
occurs

                   return

;-----
DELAY_TICK:        ;short delay function

;*****
; LOOP TIME = (y=x^3) = 250,000 clock cycles (10ms)
; ECLK = [20 MHz]
; MCLK = [5 MHz]*[50ms] = 250,000
;*****
                   movlw   d'60'
                   movwf   COUNT1        ;initialize Counter #1 to d'255'
                   movwf   COUNT2        ;initialize Counter #2 to d'255'
                   movwf   COUNT3        ;initialize Counter #3 to (USER DEFINED)
C7:                decfsz  COUNT1        ;decrement Counter #1 until "time out"
                   goto    C7
                   movlw   d'60'
                   movwf   COUNT1        ;(Refill Counter #1)
C8:                decfsz  COUNT2        ;decrement Counter #2 until "time out" occurs
                   goto    C7            ;Loop back to "START" until "time out"
occurs

                   movlw   d'60'
                   movwf   COUNT2        ;(Refill Counter #2)
C9:                decfsz  COUNT3        ;decrement Counter #3 until "time out" occurs
                   goto    C7            ;Loop back to "START" until "time out"
occurs

                   return

;-----

```

SPLASH_VER: ;displays general version/author information

```
banksel PORTD
movlw "F"
call WRITE_DATA
;.....
movlw "L"
call WRITE_DATA
;.....
movlw "A"
call WRITE_DATA
;.....
movlw "M"
call WRITE_DATA
;.....
movlw "E"
call WRITE_DATA
;.....
movlw " "
call WRITE_DATA
;.....
movlw "("
call WRITE_DATA
;.....
movlw "v"
call WRITE_DATA
;.....
movlw "e"
call WRITE_DATA
;.....
movlw "r"
call WRITE_DATA
;.....
movlw "1"
call WRITE_DATA
;.....
movlw "."
call WRITE_DATA
;.....
movlw "0"
call WRITE_DATA
;.....
movlw ")"
call WRITE_DATA
;.....
movlw h'C0' ;move cursor to beginning of second row
call WRITE_COMM
;.....
movlw "b"
call WRITE_DATA
;.....
movlw "y"
call WRITE_DATA
;.....
movlw ":"
call WRITE_DATA
;.....
movlw " "
call WRITE_DATA
;.....
movlw "R"
call WRITE_DATA
;.....
movlw "."
call WRITE_DATA
;.....
movlw "E"
call WRITE_DATA
;.....
movlw "."
```

```

call    WRITE_DATA
;-----
movlw  "L"
call    WRITE_DATA
;-----
movlw  "E"
call    WRITE_DATA
;-----
movlw  "E"
call    WRITE_DATA
;-----

return

;-----
SPLASH_ON:    ;displays "on" to the LCD

banksel  PORTD

btfsc   ON,h'0'    ;skip procedure if output already 'exists'
goto    end_on

;
;
movlw   h'02'      ;move cursor home
call    WRITE_COMM

;
;
movlw   h'C0'     ;move cursor to beginning of second row
call    WRITE_COMM
;-----
movlw   "O"
call    WRITE_DATA
movlw   "n"
call    WRITE_DATA
movlw   " "
call    WRITE_DATA

;
;
movlw   h'10'     ;cursor shift left (non-destructive)
call    WRITE_COMM
;
;
movlw   h'10'     ;cursor shift left (non-destructive)
call    WRITE_COMM
;
;
movlw   h'10'     ;cursor shift left (non-destructive)
call    WRITE_COMM

bsf     ON,h'0'    ;set 'ON'
bcf     OFF,h'0'   ;clear 'CLEAR'

end_on:

return

;-----
SPLASH_OFF:   ;displays "off" to the LCD

banksel  PORTD

btfsc   OFF,h'0'  ;skip procedure if output already 'exists'
goto    end_off

;
;
movlw   h'02'      ;move cursor home
call    WRITE_COMM

;
;
movlw   h'C0'     ;move cursor to beginning of second row
call    WRITE_COMM
;-----
movlw   "O"
call    WRITE_DATA
movlw   "f"
call    WRITE_DATA
movlw   "f"
call    WRITE_DATA

```

```

;           movlw  h'10'           ;cursor shift left (non-destructive)
;           call   WRITE_COMM
;           movlw  h'10'           ;cursor shift left (non-destructive)
;           call   WRITE_COMM
;           movlw  h'10'           ;cursor shift left (non-destructive)
;           call   WRITE_COMM

           bcf     ON,h'0'         ;clear 'ON'
           bsf     OFF,h'0'       ;set 'OFF'

end_off:
           return

;-----
SPLASH_FOUND: ;displays "found" to the LCD

           banksel PORTD

           movlw  h'C0'           ;move cursor to beginning of second row
           call   WRITE_COMM

           movlw  "F"
           call   WRITE_DATA
           movlw  "O"
           call   WRITE_DATA
           movlw  "U"
           call   WRITE_DATA
           movlw  "N"
           call   WRITE_DATA
           movlw  "D"
           call   WRITE_DATA

           bcf     ON,h'0'         ;clear 'ON'
           bcf     OFF,h'0'       ;clear 'OFF'

           return

;-----
SPLASH_CLOSE: ;displays "close" to the LCD

           banksel PORTD

           movlw  h'C0'           ;move cursor to beginning of second row
           call   WRITE_COMM
;.....
           movlw  "C"
           call   WRITE_DATA
;.....
           movlw  " "
           call   WRITE_DATA
;.....
           movlw  " "
           call   WRITE_DATA
;.....
           movlw  " "
           call   WRITE_DATA
;.....
           movlw  " "
           call   WRITE_DATA
;.....

           return

;-----
SPLASH_FAR:   ;displays "far" to the LCD

           banksel PORTD

           movlw  h'C0'           ;move cursor to beginning of second row

```

```

call    WRITE_COMM
;.....
movlw   "F"
call    WRITE_DATA
;.....
movlw   " "
call    WRITE_DATA
;.....
movlw   " "
call    WRITE_DATA
;.....
movlw   " "
call    WRITE_DATA
;.....
movlw   " "
call    WRITE_DATA
;.....
movlw   " "
call    WRITE_DATA
;.....
return

;-----
PRINT_AD0:    ;displays "AD-0" to the LCD

banksel   PORTD

movlw   h'01'    ;move cursor home
call    WRITE_COMM

movlw   "A"
call    WRITE_DATA
movlw   "D"
call    WRITE_DATA
movlw   "-"
call    WRITE_DATA
movlw   "0"
call    WRITE_DATA

;.....
return

;-----
PRINT_AD1:    ;displays "AD-1" to the LCD

banksel   PORTD

movlw   h'01'    ;move cursor home
call    WRITE_COMM

movlw   "A"
call    WRITE_DATA
movlw   "D"
call    WRITE_DATA
movlw   "-"
call    WRITE_DATA
movlw   "1"
call    WRITE_DATA

;.....
return

;-----
PRINT_AD2:    ;displays "AD-2" to the LCD

banksel   PORTD

movlw   h'01'    ;move cursor home
call    WRITE_COMM

movlw   "A"

```

```

        call    WRITE_DATA
        movlw  "D"
        call    WRITE_DATA
        movlw  "-"
        call    WRITE_DATA
        movlw  "2"
        call    WRITE_DATA

;.....
        return

;-----

CLEAR_HOME:    ;clears the LCD and moves the cursor to line-1(to the left)

                banksel  PORTD

                movlw  h'01'    ;move cursor home
                call    WRITE_COMM

;.....
                return

;-----

SPLASH_FIRE:   ;displays "fire" to the LCD

                banksel  PORTD

                movlw  h'01'    ;move cursor home
                call    WRITE_COMM

                movlw  "F"
                call    WRITE_DATA
                movlw  "I"
                call    WRITE_DATA
                movlw  "R"
                call    WRITE_DATA
                movlw  "E"
                call    WRITE_DATA

;.....
                return

;-----

                END                                ;no more code beyond this point!

```