



Software Fundamentals I

C Language Review

main(), variables, arrays, loops, function calls,
program control with switch versus if statements

DISCLAIMERS

- C programmers can be **rabid** (def: “Extremely zealous or enthusiastic; fanatical”) about certain coding conventions. If your company does not define one, pick one and use it faithfully.
- Common Rabid Topics
 - Placement of curly braces (new line or not)
 - Variable naming (Hungarian, `_`, pre and post cryptics)
 - Tab spacing (3 space, tab, etc.)

Philosophy

- “Code should be written to be read by humans first, and machines second.” ~ *Don Laabs*
- My preference
 - Coding for clarity
 - Can I “read” the code like a novel and quickly understand the logic?
 - Bug detection
 - Using common blocks, similar layouts, consistency
 - Coding for maintainability and fast upgrades
 - Avoid duplication, collect user configurations together
 - Coding to short term memory
 - Two weeks later, you’ll forget what you coded. Make it self-descriptive, include comments.
 - Document any “strangeness” or dependencies

Identifiers, Naming Conventions

- Keywords – reserved for the language

auto	double	int	struct
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

- Conventions

- All capitals - #defines (fixed numbers) **TRUE**
- Leading caps – global scope **Status_flag**
- No leading caps – local scope **index**

Identifiers, Naming Conventions

- Readability - underscore, descriptive names

Variable for motor position in counts

Bad: mp mpos

Good: motor_position_counts
motor_position_cts

Variable for surgical blanket temperature

Bad: temp (to be confused with "temporary")

Good: blanket_temperature
blanketTemperature

Loop counter for accessing array of ECG data

Bad: i

Good: i
ECG_array_ptr

Word Sizes

- Take care to use appropriate variable size
 - Limited memory in embedded devices (we get 4k!)
 - Speed related to bus size
 - Try to match register size for more efficient operation/code size
- Sizes – some are MACHINE DEPENDENT!

Type	Size	Signed	Unsigned
char	1 byte = 8 bits	-128 to 127	0 to 255
int	2 bytes = 16 bits	-32,768 to 32,767	0 to 65,535
long	4 bytes = 32 bits	-2,147,483,648 to 2,147,483,647	0 to 4,294,967,295
float	4 bytes = 32 bits	3.4E-38 to 3.4E+38	N/A
double	8 bytes = 64 bits	1.7E-308 to 1.7E+308	N/A

Many embedded systems do not have floating point capabilities! You must write your own or deal with it!

Operators

- Arithmetic: + - / * %
 - fred = fred + 5;
 - foo = 1000 / 43;
 - counter = (counter + 1) % 100;
- Increment ++, decrement --
 - i++
- Shorthand: +=
 - fred += 5;
- Binary: ==, >=, &&, ||
 - if (fred == 5 && barney != 17)

Precedence

Operator Type	Operator	Associativity
Primary Expr. Operators	() [] . -> <i>expr</i> ++ <i>expr</i> --	left-to-right
Unary Operators	* & + - ! ~ ++ <i>expr</i> -- <i>expr</i> (<i>typecast</i>) sizeof()	right-to-left
Binary Operators	* / %	left-to-right
	+ -	
	>> <<	
	< > <= >=	
	== !=	
	&	
	^	
	&&	
Ternary Operator	?:	right-to-left
Assignment Operators	= += -= *= /= %= >>= <<= &= ^= =	right-to-left
Comma	,	left-to-right

What Does This Code Do?

```
#include "msp430x20x3.h"
unsigned int read_ADC_1(void);
unsigned int temp_val;

void main(void)
{
    do
    {
        temp_val = read_ADC_1();
        if (temp_val > 130)
            P1OUT |= 0x01;
        else if (temp_val < 90)
            P1OUT &= ~0x01;
    }
    while (1);
}
```

C File Structure

- Title Block
 - Info about the file – name, function, author, version history, CR numbers, reason for changes, etc.
- Include files
 - Headers for standard .h and your .h files
- #defines
 - Constants (ON, OFF, port declarations, etc.)
- Function Redeclarations
 - List of all functions in the file in proper format
- Global Variables
- Main()
- Subroutines, interrupts, etc
- Comments (spread throughout the code appropriately)

```
/** -----  
// Name: Temperature control device, heater_control.c  
// Function: Read temperature.  If too low, turn on heater.  
// -----  
#include "msp430x20x3.h"  
  
#define HEATER_ON 0x01  
#define MAX_TEMPERATURE 130  
#define MIN_TEMPERATURE 90  
  
unsigned int read_heater_ADC(void);  
void turn_heater_on(void);  
void turn_heater_off(void);  
  
unsigned int Temperature;  
  
void main(void)  
{  
    do  
    {  
        Temperature = read_heater_ADC();  
        if (Temperature > MAX_TEMPERATURE)  
            turn_heater_off();  
        else if (Temperature < MIN_TEMPERATURE)  
            turn_heater_on();  
    }  
    while (1);  
}  
void turn_header_on(void)  
{  
    P1OUT |= HEATER_ON;  
}  
void turn_header_off(void)  
{  
    P1OUT &= ~HEATER_ON;  
}
```

Arrays, Loops, Curly Braces

```
#define      TEMPERATURE_SAMPLES      100
unsigned int Temperatures[TEMPERATURE_SAMPLES];
unsigned int read_heater_ADC(void);

void main(void)
{
    unsigned char    temperature_ctr = 0;
    unsigned int     ave_temperature;
    unsigned char    i;

    do
    {
        Temperatures[temperature_ctr] = read_heater_ADC();
        temperature_ctr++;

        for (i=0; i<temperature_ctr; i++)
        {
            ave_temperature += Temperatures[i];
        }
        ave_temperature /= temperature_ctr;

        if (ave_temperature > MAX_TEMPERATURE)
            !! Do something with heater
    }
    while (1);
}
```

initializations, braces, global vs local, bugs???

Typical Program Control

```
void main(void)
{
    while (1)
    {
        !!! (stuff)
    };
}
```

Run forever – power switch controls program "exit"

```
void main(void)
{
    do
    {
        !!! (stuff)
    }
    while (User_finished_f == NO);
}
```

Run until some external condition is set.

Switches versus If-Statements

- Both used to **conditionally execute code based on various conditions** (external inputs, math operations, interrupts, timing, etc.)
- Switches
 - Used for discrete, discontinuous, categorical or logical conditions
 - e.g., ON/OFF, different states, user selections, keypad inputs
- If-Then-Else Statements
 - Used for continuous variables, ranges
 - e.g., temperature, class grades, speeds
 - Order of operation counts!

If-Then-Else

```
void main(void)
{
    while (1)
    {
        !! (get temperature)
        if (temperature > 150)
            !! (DISABLE_HEATER)
        else if (temperature > 130)
            !! (LOWER_HEAT)
        else if (temperature < 90)
            !! (RAISE_HEAT)
        else
            !! (do nothing)
    };
}
```

Switch Statement

```
void main(void)
{
    while (1)
    {
        !! (get user_selection)
        switch(user_selection)
        {
            case RAISE_HEAT :
                !! (do something)
                break;
            case LOWER_HEAT :
                !! (do something)
                break;
            case DISABLE_HEATER :
                !! (do something)
                break;
            default:
                !! (whatever gets done otherwise);
        }
    }
}
```