Expressions (2E)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Based on Embedded Software in C for an ARM Cortex M
http://users.ece.utexas.edu/~valvano/Volume1/

Precedence

Precedence	Operators	Associativity
highest	() [] . -> ++(postfix) --(postfix)	left to right
	++(prefix) --(prefix) ! ~ sizeof(type) +(unary) -(unary) &(address) *(dereference)	right to left
	* / %	left to right
	+ -	left to right
	<< >>	left to right
	< <= > >=	left to right
	== !=	left to right
	&	left to right
	^	left to right
	|	left to right
	&&	left to right
	||	left to right
	? :	right to left
	= += -= *= /= %= <<= >>= |= &= ^=	right to left
lowest	,	left to right

Unary Operators

```
short data;          /* -32767 to +32767 */
short *pt;           /* pointer to memory */
short flag;          /* 0 is false, not zero is true */
```

	meaning	example	result
~	binary complement	~0x1234	0xEDCB
!	logical complement	!flag	flip 0 to 1 and nonzero to 0
&	address of	&var	address in memory where a variable is stored
-	negate	-100	negative 100
+	positive	+100	100
++	preincrement	++var	var=var+1, then result is var
--	predecrement	--var	var=var-1, then result is var
++	postincrement	var++	result is var, then var=var+1
--	postdecrement	var--	result is var, then var=var+1
*	reference	*pt	16 bit information pointed to by pt

Binary Operators

	meaning	example	result
+	addition	100+300	400
-	subtraction	100-300	-200
*	multiplication	10*300	3000
/	division	123/10	12
%	remainder	123%10	3
<<	shift left	102<<2	408
>>	shift right	102>>2	25

Bit-wise Operators

	meaning	example	result
&	bitwise and	0x1234 & 0x00FF	0x0034
|	bitwise or	0x1234 | 0x00FF	0x12FF
^	bitwise exclusive or	0x1234 ^ 0x00FF	0x12CB

Logical Operators

operator

	meaning	example	result
&&	and	0 && 1	0 (false)
||	or	0 || 1	1 (true)

Comparisons

```
short a,b,c,d;

int main(void) {
  a = 0x0F0F;
  b = F0F0;
  c = a&b;          /* boolean result c will be 0x0000 */
  d = a&&b;         /* logical result d will be 1 (true) */
  return 1;
}
```

Relational Operators

	meaning	example	result
==	equal	100 == 200	0 (false)
!=	not equal	100 != 200	1 (true)
<	less than	100 < 200	1 (true)
<=	less than or equal	100 <= 200	1 (true)
>	greater than	100 > 200	0 (false)
>=	greater than or equal	100 >= 200	0 (false)

```
short a,b;

void program(void) {
  if (a==0) subfunction();          /* execute subfunction if a is zero */
  if (b=0) subfunction();           /* set b to zero, never execute subfunction */
}
```

Compound Assignment Operators

```
short a,b;

void initialize(void) {
    a += b;          /* a=a+b    */
    a -= b;          /* a=a-b    */
    a *= b;          /* a=a*b    */
    a /= b;          /* a=a/b    */
    a %= b;          /* a=a%b    */
    a <<= b;         /* a=a<<b   */
    a <<= b;         /* a=a<<b   */
    a >>= b;         /* a=a>>b   */
    a |= b;          /* a=a|b    */
    a &= b;          /* a=a&b    */
    a ^= b;          /* a=a^b    */
}
```

Setting and Clearing Bits

```
void function(void) {
  PORTA |= 0x01;        /* set PA0 high */
  PORTB &= ~0x80;       /* clear PB7 low */
  PORTC ^= 0x40;         /* toggle PC6 */
}
```

Ranges

type	range	precision	example variable
unsigned char	0 to 255	8 bits	unsigned char uc;
char	-128 to 127	8 bits	char sc;
unsigned int	0 to 4294967295	32 bits	unsigned int ui;
int	-2147483648 to 2147483647	32 bits	int si;
unsigned short	0 to 65535U	16 bits	unsigned short us;
short	-32768 to 32767	16 bits	short ss;
long	-2147483648 to 2147483647	32 bits	long sl;
unsigned long	0 to 4294967295	32 bits	unsigned long ui;

Simple Type Conversion

type1		type2	example
unsigned char	fits inside	unsigned short	uc+us is of type unsigned short
unsigned char	fits inside	short	uc+ss is of type short
unsigned char	fits inside	long	uc+sl is of type long
char	fits inside	short	sc+ss is of type short
char	fits inside	long	sc+sl is of type long
unsigned short	fits inside	long	us+sl is of type long
short	fits inside	long	ss+sl is of type long

Explicit cast for higher precision arithmetic

```
char y;                 // output of the filter
unsigned char x;        // input of the filter

void filter(void) {
  y = (12 * (short) x + 56 * (short) y) / 100;
}

// 16-bit short type arithmetic
// then assign the result to 8-bit unsigned char type
```

Explicit cast for intermediate results

```
// y(n) = [113*x(n) + 113*x(n-2) – 98*y(n-2)]/128
// channel specifies the A/D channel

short x[3],y[3];                    // MACQs containing current and previous

void SysTick_Handler(void) {
  y[2]=y[1]; y[1]=y[0];             // shift MACQ
  x[2]=x[1]; x[1]=x[0];
  x[0] = A2D(channel);              // new data
  y[0] = (113*((long)x[0]+(long)x[2])-98*(long)y[2]) >> 7;
}


// 113*(x[0]+x[2]) – 98*y[2] might result in overflow and underflow
// (113*((long)x[0]+(long)x[2]) – 98*(long)y[2] thus used explicit type cast (long)
```

Selection Operators

```
short a,b;

void sub1(void) {
  a = (b==1) ? 10 : 1;
}

void sub2(void) {
  if (b==1)
     a=10;
  else
     a=1;
}
```

Condition Codes

bit	name	meaning after addition or subtraction
N	negative	result is negative
Z	zero	result is zero
V	overflow	signed overflow
C	carry	unsigned overflow

References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun
[5] “A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux“
http://cseweb.ucsd.edu/~ricko/CSE131/teensyELF.htm
[6] http://en.wikipedia.org
[7] http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html
[8] http://csapp.cs.cmu.edu/public/ch7-preview.pdf