Embedded Software Essentials

C-programming Review

C1 M1 V4

Be Boulder.

University of Colorado Boulder

Declaring Variables

Variable Declaration Format:

```
type-qualifier(s) type-modifier data-type variable-name = initial-value;
```

Example Declaration and Assignment:

```
const unsigned char foo = 12;
long int foo;
   …
foo = 400;
```

Declaring Variables

Variable Declaration Format:

```
type-qualifier(s) type-modifier data-type variable-name = initial-value;
```

Example Declaration and Assignment:

```
const unsigned char foo = 12;
long int foo;
   …
foo = 400;
```

Data Types:

- Integer
- Floating Point
- Enumerated
- Derived
- Void

Declaring Variables

Variable Declaration Format:

```
type-qualifier(s) type-modifier data-type variable-name = initial-value;
```

Example Declaration and Assignment:

```
const unsigned char foo = 12;
long int foo;
    …
foo = 400;
```

Modifiers:

- Short
- Long
- Unsigned
- Signed

Declaring Variables

Variable Declaration Format:

```
type-qualifier(s) type-modifier data-type variable-name = initial-value;
```

Example Declaration and Assignment:

```
const unsigned char foo = 12;
long int foo;
   …
foo = 400;
```

Qualifiers:

- Const
- Volatile
- Restrict

C-Data Types

1 Byte = 8 Bits

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$

Most Significant Bit (MSb)

Least Significant Bit (LSb)

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Binary to Hexadecimal

- **Binary** – Base 2
  - 0b – Indicates Binary
- **Hexadecimal** – Base 16 (0-9, A-F)
  - 0x – Indicates Hexadecimal

```
0xA49E = 0b1010010010011110
```

Hex	A	4	9	E
Binary	1010	0100	1001	1110

Hex	Binary
0x0	0b0000
0x1	0b0001
0x2	0b0010
0x3	0b0011
0x4	0b0100
0x5	0b0101
0x6	0b0110
0x7	0b0111
0x8	0b1000
0x9	0b1001
0xA	0b1010
0xB	0b1011
0xC	0b1100
0xD	0b1101
0xE	0b1110
0xF	0b1111

Bits and Numbers

Given an n-bit Binary Number:

$$\boldsymbol{b_{n-1} b_{n-2} \ldots b_1 b_0}$$

MSb LSb

$$Decimal\_Value(unsigned) = b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_1 2^1 + b_0 2^0$$

$$Decimal\ Value\ (signed\ -2's) = -(b_{n-1}2^{n-1}) + b_{n-2}2^{n-2} + \cdots + b_1 2^1 + b_0 2^0$$

Raw Binary	Unsigned	Two's Complement	Hexadecimal
0b0111_1111	127	127	0x7F
0b0111_1110	126	126	0x7E
0b0111_1101	125	125	0x7D
0b0000_0010	2	2	0x02
0b0000_0001	1	1	0x01
0b0000_0000	0	0	0x00
0b1111_1111	255	-1	0xFF
0b1111_1110	254	-2	0xFE
0b1000_0010	130	-126	0x82
0b1000_0001	129	-127	0x81
0b1000_0000	128	-128	0x80

Bits and Numbers

Given an 8-bit Binary Number:

$$\mathbf{0b10010101}$$

$$\boldsymbol{Decimal\ Value\ unsigned}\ (\mathbf{0b10010101})$$
$$= 1 * 2^7 + 0 * 2^6 + 0 * 2^6 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$$

$= (128 + 16 + 4 + 1)_{10}$

$= (149)_{10}$

$$\boldsymbol{Decimal\ Value\ signed\ 2's}\ (\mathbf{0b10010101})$$
$$= -(1 * 2^7) + 0 * 2^6 + 0 * 2^6 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$$

$= (-128 + 16 + 4 + 1)_{10}$

$= (-107)_{10}$

Raw Binary	Unsigned	Two's Complement	Hexadecimal
0b0111_1111	127	127	0x7F
0b0111_1110	126	126	0x7E
0b0111_1101	125	125	0x7D
0b0000_0010	2	2	0x02
0b0000_0001	1	1	0x01
0b0000_0000	0	0	0x00
0b1111_1111	255	-1	0xFF
0b1111_1110	254	-2	0xFE
0b1000_0010	130	-126	0x82
0b1000_0001	129	-127	0x81
0b1000_0000	128	-128	0x80

Operators

Type	Operators
Logical	||, &&, !
Bitwise	<<, >>, |, &, ^, ~
Arithmetic	+, -, /, *, ++, --, %
Relational	<, <=, >, =>, ==, !=

Used to assign, manipulate and compare data

Operators

Type	Operators
Logical	||, &&, !
Bitwise	<<, >>, |, &, ^, ~
Arithmetic	+, -, /, *, ++, --, %
Relational	<, <=, >, =>, ==, !=

```
|| = Logical OR
&& = Logical AND
!  = Logical NOT
```

```
if ( condition1 || condition2 )
if ( condition1 && condition2 )
while ( ! condition )
```

Boolean Conditions

- **False: Any condition that is Zero**
- **True: Any non-zero condition**

```
int foo1 = 1;
int foo2 = 0;

if ( foo1 || foo2 ){…}

if ( foo1 && foo2 ){…}

while ( ! foo2 ){…}
```

Operators

Type	Operators
Logical	||, &&, !
Bitwise	<<, >>, |, &, ^, ~
Arithmetic	+, -, /, *, ++, --, %
Relational	<, <=, >, =>, ==, !=

```
|| = Logical OR
&& = Logical AND
!  = Logical NOT
```

```
if ( condition1 || condition2 )
if ( condition1 && condition2 )
while ( ! condition )
```

Boolean Conditions

- **False: Any condition that is Zero**
- **True: Any non-zero condition**

```
int foo1 = 1;
int foo2 = 0;

if ( foo1 || foo2 ){…} => TRUE

if ( foo1 && foo2 ){…} => FALSE

while ( ! foo2 ){…} => TRUE
```

Operators

Type	Operators
Logical	||, &&, !
Bitwise	<<, >>, |, &, ^, ~
Arithmetic	+, -, /, *, ++, --, %
Relational	<, <=, >, =>, ==, !=

```
<< = Left Shift
>> = Right Shift
|  = Bitwise OR
&  = Bitwise AND
^  = Bitwise EXOR
~  = One's Complement
```

Examples:

```
foo = varA >> 4;
foo = varA | varB;
foo = varA & varB;
foo = varA ^ varB;
foo = ~varA
```

Right-Shift (logical)

varA **>>** 4 → Right Shift 4 bits

Ex: 0b10010110 **>>** 4

= 0b00001001

AND

foo = varA **&** varB;

varA = 0xF5; →	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**1**
varB = 0x5B; →	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**

foo = 0x51

Operators

Type	Operators
Logical	||, &&, !
Bitwise	<<, >>, |, &, ^, ~
Arithmetic	+, -, /, *, ++, --, %
Relational	<, <=, >, =>, ==, !=

```
<< = Left Shift
>> = Right Shift
|  = Bitwise OR
&  = Bitwise AND
^  = Bitwise EXOR
~  = One's Complement
```

Examples:

```
foo = varA >> 4;
foo = varA | varB;
foo = varA & varB;
foo = varA ^ varB;
foo = ~varA
```

Right-Shift (logical)

varA **>>** 4 → Right Shift 4 bits

Ex: 0b10010110 **>>** 4

= 0b00001001 **(zeros shifted in)**
(lower bits truncated)

AND

foo = varA **&** varB;

varA = 0xF5; →	1	1	1	1	0	1	0	1
varB = 0x5B; →	0	1	0	1	1	0	1	1
foo = 0x51 →	0	1	0	1	0	0	0	1

Operators

Type	Operators
Logical	||, &&, !
Bitwise	<<, >>, |, &, ^, ~
Arithmetic	+, -, /, *, ++, --, %
Relational	<, <=, >, =>, ==, !=

```
+  = Addition
-  = Subtraction
/  = Divide
*  = Multiple
++ = Increment
-- = Decrement
%  = Modulus
```

Examples:

```
foo = varA + varB;
foo = varA - varB;
foo = varA * varB;
foo = varA % 2;
foo = varA++;
```

Modulus Operator (Remainder)

15 % 4 → 3

Post-Increment/Pre-Decrement

varA++ → varA = varA + 1;
varA-- → varA = varA - 1;

Be Boulder.
University of Colorado Boulder

Operators

Type	Operators
Logical	||, &&, !
Bitwise	<<, >>, |, &, ^, ~
Arithmetic	+, -, /, *, ++, --, %
Relational	<, <=, >, =>, ==, !=

Examples:

```
foo += varA;
foo -= 5;
foo |= varA;
foo ^= varB;
foo >>= 2;
```

Combination with Assignment

```
varB += varA; → varB = varB + varA;
varB |= varA; → varB = varB | varA;
```

Be Boulder.

University of Colorado Boulder

Operators

Type	Operators
Logical	||, &&, !
Bitwise	<<, >>, |, &, ^, ~
Arithmetic	+, -, /, *, ++, --, %
Relational	<, <=, >, =>, ==, !=

```
<  = Less-Than
<= = Less-Than Or Equal to
>  = Greater-Than
>= = Greater-Than or Equal to
== = Equal to
!= = Not Equal To
```

Examples:

```
if ( varA <= varB )
if ( varA == varB )
while ( varA != 0 )
while ( varA < 10 )
```

```
int foo1 = 1;
int foo2 = 0;

if ( foo1 > foo2 ){…} => TRUE

if ( foo1 <= foo2 ){…} => FALSE

If ( foo1 != foo2 ){…} => TRUE
```

Be Boulder.

University of Colorado Boulder

Control Program Flow

```
if ( condition ) {
   //code
}

if ( condition )
    //code
}
else {
   //code
}
```

```
if ( condition-1 ) {
   //code
}
else if ( condition-2 ) {
   //code
}
else {
   //code
}
```

```
switch ( expression ) {
  case const-exp1:
    //code
    break;
  case const-exp2:
    //code
    break;
  …
  default:
    //code
    break;
}
```

Be Boulder.

University of Colorado Boulder

Loops

```
for( variable-initialize; condition; variable-expression ) {
   // Code
}
```

Preloop check

```
while ( condition ) {
    // Code
}
```

Preloop check

```
do {
    // Code
} while( condition );
```

Postloop check

Break & Continue

```
for( i = 0 ; i < 100; i++ ) {
  if ( array[i] < 0 ) {
    break;
  }
  if ( array[i] == 0 ) {
    continue;
  }

  sum += ( varA / array[i] );
}
```

Exits loop immediately

Moves to next iteration (i+1), checks condition

Functions

Function Declaration/Prototype

```
function-type function-name ( param1-type param1, param2-type param2, … );
```

Function Definition

```
function-type function-name( param1-type param1, param2-type param2, … ){
  //code for the function
}
```

Example Declarations:

```
int main();
char foo( char a, char b );
void bar( int * ptr );
int * bar( float data );
```

Example Definition:

```
char foo( char a, char b ) {
  return ( (a + b) / 2 );
}
```

Be Boulder.

University of Colorado Boulder

Functions and Headers

file.c

```
#include "file.h"

/* Function Definition */
void foo( int * a, char b ){
  *a = b % 2
}
```

file.h

```
#ifndef __FILE_H__
#define __FILE_H__

/* Function Declaration/Prototype */
void foo( int * a, char b );

#endif /* __FILE_H__ */
```

Pointers

```
int foo = 0x34;
int * ptr;        // Pointer Declaration Operator
ptr = &foo;       // Address-Of Operator
*ptr = 0x52;      // Dereference Operator
```

=> foo = 0x52

Pointers

```
int foo = 0x34;
int * ptr;     // Pointer Declaration Operator
ptr = &foo;    // Address-Of Operator
*ptr = 0x52;   // Dereference Operator
```

	Memory
Address	**Data**
0x100	0xF0012345
0x104	0x12FF234D
0x108	0x0120AB01

4 Bytes

Garbage Data to start

Declaring foo allocates and initializes 4 bytes in memory at an arbitrary address

	Memory
Address	**Data**
0x100 **(foo)**	**0x00000034**
0x104	0x12FF234D
0x108	0x0120AB01

```
int foo = 0x34;
```

Be Boulder.

University of Colorado Boulder

Pointers

```
int foo = 0x34;
int * ptr;     // Pointer Declaration Operator
ptr = &foo;    // Address-Of Operator
*ptr = 0x52;   // Dereference Operator
```

Defining a pointer allocates memory for the address type

	Memory
Address	**Data**
0x100 (foo)	0x00000034
0x104 **(ptr)**	0x12FF234D
0x108	0x0120AB01

```
int * ptr;
```

Variable ptr, holds the address of foo (address = 0x100)

	Memory
Address	**Data**
0x100 (foo)	0x00000034
0x104 (ptr)	**0x00000100**
0x108	0x0120AB01

```
ptr = &foo;
```

Dereferencing ptr and assigning a new value, changes foo

	Memory
Address	**Data**
0x100 (foo)	**0x00000052**
0x104 (ptr)	0x00000100
0x108	0x0120AB01

```
*ptr = 0x52;
```

Be Boulder.

University of Colorado Boulder