

Variable Types and Operations



ME430: MECHATRONICS

Everything is stored in binary

0	1	1	1	1	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	1	1	1	1	0	1
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	1



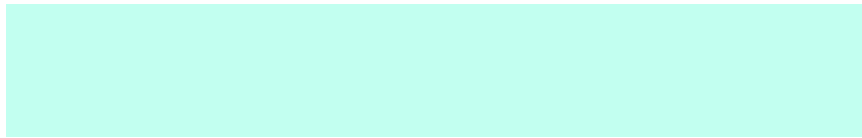
ME430: MECHATRONICS

Two Primary Storage Types

Integer



Floating Point



ME430: MECHATRONICS

Everything is stored in binary


	0	1	1	1	1	0	0	1
char x = 1;	0	0	0	0	0	0	0	1
char y = 2;	0	0	0	0	0	0	1	0
char z = 3;	0	0	0	0	0	0	1	1
	0	0	0	0	0	0	0	0
int Dave = 1;	0	0	0	0	0	0	0	1
	0	0	1	1	1	1	0	1
	1	1	0	0	1	1	0	0
float pt1x = 0.1;	1	1	0	0	1	1	0	0
	1	1	0	0	1	1	0	1



ME430: MECHATRONICS


Integer Ranges

	Unsigned		Signed	
	Min	Max	Min	Max
char (8 bits)				
int (16 bits)				
long (32 bits)				

 ME430: MECHATRONICS

Integer Overflow

	0	1	1	1	1	0	0	1	<p>What if...</p> <p>x = 200;</p> <p>y = 200;</p> <p>x = 260;</p> <p>Dave=100+100+100;</p>
unsigned char x=1;	0	0	0	0	0	0	0	1	
char y = 2;	0	0	0	0	0	0	1	0	
	0	0	0	0	0	0	1	1	
int Dave = 1;	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	1	
	0	0	1	1	1	1	0	1	
	1	1	0	0	1	1	0	0	
	1	1	0	0	1	1	0	0	
	1	1	0	0	1	1	0	1	

 ME430: MECHATRONICS

Floating Point

```
float pt1x = 0.1;
```

0	1	1	1	1	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	1	1	1	1	0	1
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	1

Sign	8-bit biased exponent E	23-bit unsigned fraction f
\pm	$e_7e_6e_5e_4e_3e_2e_1e_0$	$d_0d_1d_2d_3 \dots d_{23}$



ME430: MECHATRONICS

Floats are approximated

```
float x = 0.1;
x = 0 0111 1011 100 1100 1100 1100 1100 1101
```

Actually get's stored as: $x = 0.1000000005$

```
float f = 0.1;
float f2 = 0.1000000005;

if (f == f2)
    printf("The two are equal");
else
    printf("The two are different");
```



ME430: MECHATRONICS

Operator	Symbol	Associativity
Address	&	L to R
Dereferencing	*	
Logical NOT	!	R to L
Bit inverse	~	
Increment	++	
Decrement	--	
Negation	-	
Cast (see note**below)	(<type>)	
Multiplication	*	L to R
Division	/	
Modulus	%	
Addition	+	L to R
Subtraction	-	
Shift Bits Left	<<	L to R
Shift Bits Right	>>	
Less Than	<	L to R
Less Than or Equal To	<=	
Greater Than	>	
Greater Than or Equal To	>=	
Equal To	==	
Not Equal To	!=	
Bit AND	&	L to R
Exponentiation (for float)	^	L to R
Exclusive OR (for int)		
Bit OR		L to R
Logical AND	&&	L to R
Logical OR		L to R
Assignment	=	R to L
Additive Assignment	+=	
Multiplicative Assignment	*= etc.	

Integer math issues: + - * /

Truncates fractional results

```
char y;
y = 3/4 * 20;
```

Order of operations

```
y = 100*8/16;
```

ME430: MECHATRONICS

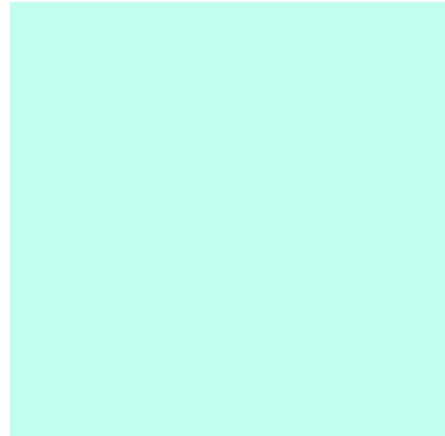
Shortcuts: +=, -=, *=, /=, ++, --

- Quick variable modifications
- Examples

```
myInt = 40;  
myInt += 10;  
myInt *= 2;  
myInt -= 1;
```

- Super shortcuts

```
myInt++;  
myInt--;
```



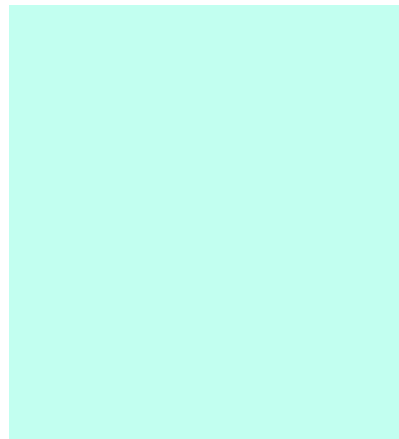
ME430: MECHATRONICS

Modulus: %

- Long division remainder

- Examples

```
int myInt;  
char quo, remain  
char hunds, tens, ones;  
quo    = 23 / 7;  
remain = 23 % 7;  
  
myInt = 345;  
hunds = (myInt / 100) % 10;  
tens  = (myInt / 10) % 10;  
ones  = (myInt / 1) % 10;
```



ME430: MECHATRONICS

Cast: (<type>)

- Converting from one type to another

- Examples

```
float myFloat = 14.5;
int myInt;
char myChar = 100;
myInt = (int) myFloat + 6;
myFloat = (float) myInt - 5.5;
myInt = (int) myChar +
        myChar + myChar;
```



ME430: MECHATRONICS

Bit Shifts: >> <<

- Moving the bits within a variable
 - Effectively a divide by 2 or multiply by 2

- Examples:

```
int a;
a = 0b0110 >> 2;
a = 0b0110 << 1;
```



ME430: MECHATRONICS

Bit Inverse: ~

- Bitwise NOT (inverse) operation

A	Y
0	1
1	0

- Examples:

```
char a;
a = ~0b00000001;
```



ME430: MECHATRONICS

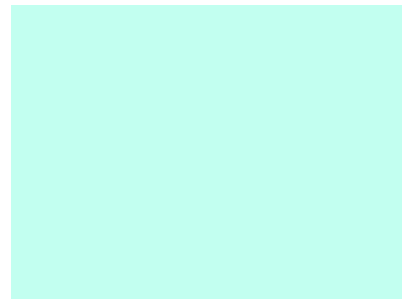
Bit AND: &

- Bitwise AND operation

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

- Examples:

```
char a;
a = 0b0111 & 0b1101;
a = a & 0b1011;
```



- Common bit clear:

```
a &= 0x01;
```



ME430: MECHATRONICS

Bit OR: |

- Bitwise OR operation

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

- Examples:

```
char a;
a = 0b1100 & 0b1000;
a = a | 0b0010;
```

- Common bit set:

```
a |= 0x01;
```



ME430: MECHATRONICS

Bit XOR: ^

- Bitwise Exclusive Or operation

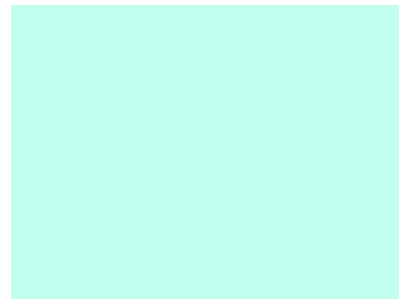
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

- Examples:

```
char a;
a = 0b0110 ^ 0b0100;
a = a ^ 0b1000;
```

- Common bit toggle:

```
a ^= 0x01;
```



ME430: MECHATRONICS

Logical Operations

- Standard in conditional statements
- &&, ||, ==, !=, <, >, <=, >=

- Examples:

```
char myChar = 1;
char isTrue = 1;
if( 3<4 && myChar != 5)
while(myChar>0 || myChar == -5)
while(!isTrue)
```



Bitwise vs. Logical Summary

	Bitwise	Logical
AND	&	&&
OR		
NOT	~	!



Arrays

- Array of some existing type
- Example

```
char myChar;
char myArray[5];
myChar = myArray[0];
myChar = myArray[4];
```

0	0	0	0	0	1	0	1	
0	0	0	0	0	0	1	0	
0	0	0	0	0	0	1	1	
0	0	0	0	1	0	0	0	
0	0	0	1	0	0	0	0	



ME430: MECHATRONICS

Arrays

Loading data into an array

```
int i[] = { 1, 2, 3, 4, 5};
```

```
char y[5];
y[0] = 10;
y[1] = 20;
y[2] = 30;
y[3] = 40;
y[4] = 50;
```



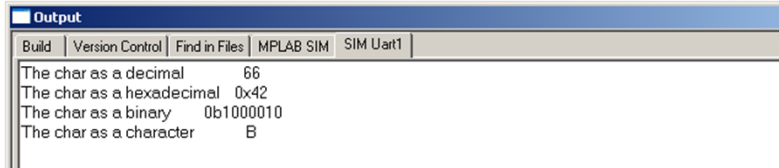
ME430: MECHATRONICS

ASCII			Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
	0	00			Null	32	20	Space	64	40	@	96	60	`
	1	01			Start of heading	33	21	!	65	41	A	97	61	a
	2	02			Start of text	34	22	"	66	42	B	98	62	b
	3	03			End of text	35	23	#	67	43	C	99	63	c
	4	04			End of transmit	36	24	\$	68	44	D	100	64	d
	5	05			Enquiry	37	25	%	69	45	E	101	65	e
	6	06			Acknowledge	38	26	&	70	46	F	102	66	f
	7	07			Audible bell	39	27	'	71	47	G	103	67	g
	8	08			Backspace	40	28	(72	48	H	104	68	h
	9	09			Horizontal tab	41	29)	73	49	I	105	69	i
	10	0A			Line feed	42	2A	*	74	4A	J	106	6A	j
	11	0B			Vertical tab	43	2B	+	75	4B	K	107	6B	k
	12	0C			Form feed	44	2C	,	76	4C	L	108	6C	l
	13	0D			Carriage return	45	2D	-	77	4D	M	109	6D	m
	14	0E			Shift out	46	2E	.	78	4E	N	110	6E	n
	15	0F			Shift in	47	2F	/	79	4F	O	111	6F	o
	16	10			Data link escape	48	30	0	80	50	P	112	70	p
	17	11			Device control 1	49	31	1	81	51	Q	113	71	q
	18	12			Device control 2	50	32	2	82	52	R	114	72	r
	19	13			Device control 3	51	33	3	83	53	S	115	73	s
	20	14			Device control 4	52	34	4	84	54	T	116	74	t
	21	15			Neg. acknowledge	53	35	5	85	55	U	117	75	u
	22	16			Synchronous idle	54	36	6	86	56	V	118	76	v
	23	17			End trans. block	55	37	7	87	57	W	119	77	w
	24	18			Cancel	56	38	8	88	58	X	120	78	x
	25	19			End of medium	57	39	9	89	59	Y	121	79	y
	26	1A			Substitution	58	3A	:	90	5A	Z	122	7A	z
	27	1B			Escape	59	3B	;	91	5B	[123	7B	{
	28	1C			File separator	60	3C	<	92	5C	\	124	7C	
	29	1D			Group separator	61	3D	=	93	5D]	125	7D	~
	30	1E			Record separator	62	3E	>	94	5E	^	126	7E	~
	31	1F			Unit separator	63	3F	?	95	5F	_	127	7F	□

char
often used to hold ASCII values since only 7 bits are needed

Example using ASCII

```
void main() {
    char y = 'B';
    printf("The char as a decimal      %d  \n", y);
    printf("The char as a hexadecimal  0x%x \n", y);
    printf("The char as a binary       0b%b \n", y);
    printf("The char as a character     %c  \n", y);
}
```



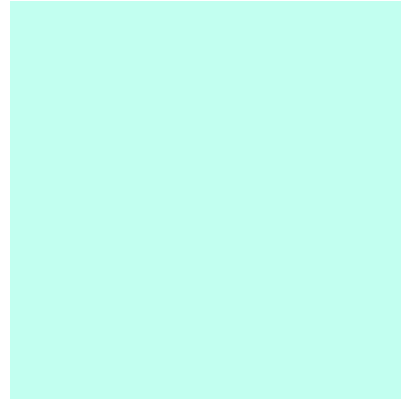
Character arrays (Strings)

```
char listchar[]={ 't','o','y',' ','b','o','a','t','\0' };
char listchar2[] = "toy boat";
```

```
char numchar[5];
numchar[0] = 0x61;
numchar[1] = 98;
numchar[2] = 'c';
numchar[3] = 0b01100100;
numchar[4] = '\0';
```

```
printf(" %c ",listchar[1]);
printf(" %s ",listchar2);
printf(" %s ",numchar);
```

Strings always end in the null character ('\0')



Advanced data types

- Pointers
& *
- Structures
struct
- Enumerated Data Types
enum
- Custom Data Types
typedef

