

## 3<sup>rd</sup> C Class with Mr. Czik

Our Robovation controller is based on the PIC18F8520 microprocessor. The MPLAB IDE compiles the C code to run specifically on this processor. C itself is mostly platform independent (except for machine-specific hardware calls).

Must declare type of variables in C – char, int, long, float

Type	Min Value	Max Value	Memory Size	Speed for basic operations
char	-128	128	1 byte	Very Fast
int	-32768	32767	2 bytes	Fast
long	-2147483648	2147483647	4 bytes	Average
float	Negatively Virtually infinite	Virtually infinite	4 bytes	Painfully slow

**char** for alphanumeric or small range of integer numbers (8 bits = 256 maximum number of values)

**int** for medium range integer numbers (16 bits = 65536)

**long** for large range integer numbers (32 bits = 4294967296)

**float** works with fractional numbers, should be avoided for real-time control work, as it is WAY too slow)

## Operators

( ) grouping	[ ] array			1 <sup>st</sup> precedence
++ increment	-- decrement	! logical NOT		2 <sup>nd</sup> precedence
* multiplication	/ division	% modulo remainder		3 <sup>rd</sup> precedence
+ addition	- subtraction			4 <sup>th</sup> precedence
> greater than	< less than	>= greater or equal	<= less or equal	5 <sup>th</sup> precedence
== equal to	!= not equal to			
&& logical AND				
 logical OR				

Every command and datum has an address associated with it, where it can be retrieved for the computer to process. This is similar to a GPS device. The longer the range of the addresses, the more memory can be addressed.

# 3<sup>rd</sup> C Class with Mr. Czik

## Operators

Integer division truncates any fraction or remainder.

$$8 / 3 = 2 \text{ if all 8 and 3 are integers.}$$

The modulo (or remainder) operator will divide the first number by the second number, and return only the remainder left over after integer division.  $7 \% 3 = 1$        $10 \% 5 = 0$        $12 \% 7 = 5$

The assignment operator ( = ) is used to assign a value to a variable

$$a = 5 + 4 \qquad a = 9$$

The order in which operators are performed follows rules. In general, execution is from left to right. The previous table shows that some operators will be run before others, with top priority going to grouping that the programmer defines with parentheses ( ). This tends to be the best way to avoid confusion.

$$a = ( 5 + 3 ) * ( 4 + 7 ) = 15 * 28 = 88$$
$$a = 5 + 3 * 4 + 7 = 5 + 12 + 7 = 24$$

Makes a big difference, doesn't it?

## Statements

A statement is a line of code, ending in a semi-colon.

$$a = b + c; \qquad i = 0;$$

<pre>if (condition)     {         statement1;         statement2;     } else if</pre>	<pre>if (a == b)     {         c = d + 1;         f = g + h;         if ( f == 6)             {                 g = d + 6             }     } }</pre>
---	---

We can nest if statements within each other.

Remember, the == operator is for testing equality, as opposed to the = operator that is for assignment.

## 3<sup>rd</sup> C Class with Mr. Czik

As the C compiler will ignore all white space, the following line is the equivalent of the previous nested if statement.

```
if (a == b) { c = d + 1; f = g + h; if (f == 6) { g = d + 6; } }
```

But . . . it will make your (and everyone else's) head hurt.

Separate lines, tabs and comments help to read and understand code. Structured code (no goto statements!) is critical if more than one person will be working with the code, and very important even if there is only one programmer, as looking at code after days or weeks can be confusing. Memory is a frail thing.

### Boolean (Logic) Operators

Using Boolean operators in testing return only a 1 for true or a 0 for false.

>	greater than	
<	less than	
>=	greater than or equal to	
<=	less than or equal to	
!=	not equal to	
==	equal to	
&&	logical AND	if ( <u>this true</u> && <u>that true</u> ) both must be true
	logical OR	( <u>this true</u>    <u>that true</u> ) either or both true
!	logical NOT	( <u>this true</u> ! <u>that not true</u> ) 1 <sup>st</sup> true, 2 <sup>nd</sup> not true

if (a > b && c < d)                      both a > b and c < d must be true,  
to return logical 1 (otherwise, a logical 0)

if (a > b || c < d)                      either a > b and/or c < d must be true,  
to return logical 1 (otherwise, a logical 0)

if (a > b ! c < d)                      a > b must be true and c < d must not be  
true, to return logical 1 (otherwise, a logical 0)

### Breaking Out of a Loop Early

```
for (i = 0; i < 10; i ++)  
{  
    statement1  
    statement2  
}
```

### 3<sup>rd</sup> C Class with Mr. Czik

Insert the statement: `i = 10;` (or greater than 10), and execution will switch beyond the for loop (for example, in a if/else test).

```
if (c == d)
{
    i = 10
}
```

Also, inserting the **break** statement will automatically switch execution beyond the for loop.

```
if (c == d)
{
    break           ↓
}
```

Also, inserting the **continue** statement will automatically switch execution back to the top of the for loop, and continue to iterate, but not execute the code after the **continue** statement.

```
if (c == d)
{
    continue       ↑
}
```

Note: curly brackets are used to make all the code between the brackets into a virtual single statement, as technically, C only allows a single statement.

### Arrays

```
density = sensor();
for (i = 0; i < 8; i ++ )
{
    if (density == color[i])
    {
        navigate();
    }
}
```

Arrays are a storage location – a “bucket” to hold information – that is subdivided into smaller “buckets”, like cubbyholes to subdivide a storage cabinet.

### 3<sup>rd</sup> C Class with Mr. Czik

White	= 255	color [0] = 255
Violet	= 240	color [1] = 240
Blue	= 200	color [2] = 200
Green	= 160	color [3] = 160
Yellow	= 120	color [4] = 120
Orange	= 80	color [5] = 80
Red	= 40	color [6] = 40
Black	= 0	color [7] = 0

Like most C structures, the array starts counting from zero. All colors are set to arbitrary values. Arrays must be declared, as all variables are:

```
int color [8]
```

0	1	2	3	4	5	6	7
255	240	200	160	120	80	40	0