ROBOTICS

Programming in C
An introduction to programming
In the beginning...
things were written in **machine code**
and directly executed by the CPU

```c
mem[0]=0x23; // load register a with following
mem[1]=0x00;
mem[2]=0xa8; // output a to r0
mem[3]=0x17; // increment a
mem[4]=0xa9; // output a to r1
mem[5]=0x17; // increment a
mem[6]=0xaa; // output a to r2
mem[7]=0x17; // increment a
mem[8]=0xab; // output a to r3
mem[9]=0x17; // increment a
mem[10]=0xac; // output a to r4
mem[11]=0x17; // increment a
mem[12]=0xad; // output a to r5
mem[13]=0x17; // increment a
mem[14]=0xae; // output a to r6
mem[15]=0x17; // increment a
mem[16]=0xaf; // output a to r7
mem[17]=0x17; // increment a
mem[18]=0x04; // jump to first page with following
mem[19]=0x02;
```
Then along came abstraction, with an **assembly language** to provide symbolic references for the numeric machine codes

**Main:**

```assembly
clr PORTB ; initialize PORTB
bsf STATUS , RP0 ; Move to bank 1
movlw PORTB_DIR ; value for TRISB
movwf TRISB ; set by defined variable
bcf STATUS , RP0 ; Move to bank 1
movlw MAX_BITS
movwf BIT_COUNT ; sets the bit count to seven
clr INPUT_BYTE
```

**SSTestFall:**

```assembly
btfs PORTB , SS_BIT ; check slave bit, if clear, skip next
goto SSTestFall ; loop to check again
goto ClockTestFall ; move on
```
and now we have...
`#include "saast.h"`  // custom macros

`#define MAX 7`

```c
void main(void){
    int i;

    m_init();  // initialize the system

    while(1){
        for(i=0; i<MAX; i++){
            toggle(PORTE,i);  // toggle Port E pins
        }
    }
}
```
C is case-sensitive!
white space does not matter
don’t forget the semicolon;
don’t forget the {} 
#define constants
declare variables before use
no magic numbers!
compile and test as you go
comment your code!
please, comment your code...
translating common compiler errors

main.c: In function 'main':
main.c:17: error: 'i' undeclared (first use in this function)
main.c:17: error: (Each undeclared identifier is reported only once
main.c:17: error: for each function it appears in.)
make: *** [main.o] Error 1

(undeclared variable)

main.c: In function 'main':
main.c:43: error: expected declaration or statement at end of input
make: *** [main.o] Error 1

(missing “}”)

main.c: In function 'main':
main.c:19: error: expected ';' before '}') token
make: *** [main.o] Error 1

(missing “;”)
preprocessor directives

directives processed before compilation

include other files (generally “header” files with other # defines, function prototypes, etc.)

```c
#include <filename.h>        // file in the include path
#include "filename.h"       // file in the current directory
```

define constants (essentially a find & replace - no semicolon!)

```c
#define CONSTANT value
#define ENC_LINES 1024
#define TRUE 1
```
variables

variables must be declared before they are used!

```plaintext
type variable=initial, variable=initial;
int x;
short y, z;
long foo = 456;
unsigned int a=5, b=6;
char c = 'b';
```

<table>
<thead>
<tr>
<th>type</th>
<th>bits</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>8</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>unsigned char</td>
<td>8</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>int</td>
<td>16</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>unsigned int</td>
<td>16</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>long</td>
<td>32</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>32</td>
<td>0</td>
<td>4294967295</td>
</tr>
<tr>
<td>float / double</td>
<td>IEEE32</td>
<td>1.175494E-38</td>
<td>3.402823E+38</td>
</tr>
</tbody>
</table>

ultimately, everything is binary to the CPU
### Basic Operators

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Conditional</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>==</td>
</tr>
<tr>
<td>-</td>
<td>!=</td>
</tr>
<tr>
<td>*</td>
<td>&lt;</td>
</tr>
<tr>
<td>/</td>
<td>&lt;=</td>
</tr>
<tr>
<td>%</td>
<td>&gt;</td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unary</th>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>--</td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>!</td>
</tr>
</tbody>
</table>

All arithmetic and bitwise operators can be used in assignments.
operator precedence

higher operators will be applied first

parenthesis       ()  []
structure access  .  ->
uminary           !  ~  ++  --  -  *  &
multiply, divide, modulus  *  /  %
add, subtract     +  -
bit shifts        >>  <<
inequality        <  <=  >=  >
equal, not equal  ==  !=
bitwise AND       &
bitwise exclusive OR  ^
bitwise OR        |
logical AND       &&
logical OR        ||
ternary conditional  ?  :
assignment       =  *=  /=  %=  +=  -=  <<=  >>=  &=  |=  ^=
comma             ,

(when in doubt, add parentheses!)
iteration

**WHILE**: as long as the expression equals any non-zero value, the directives will be executed repeatedly

```c
while(expression){
    directives;
}
```

**FOR**: as long as the initialized variable is less than the termination value, the directives will be executed repeatedly

```c
for(initialization; continuation; increment){
    directives;
}
```
IF: if the expression equals any non-zero value, directives will be executed

```cpp
if(expression) {
    directives;
} else {
    other directives;
}
```

expressions can be formed using:

```
if(a==b)
if(a!=b)
if(a<b)
if(a&&b)
if(!c)
```

**examples**

```
if(a==b)
if(a!=b)
if(a<b)
if(a&&b)
if((a==5) && (b!=4))
if(!c)
```