MPLAB MCC18 C-PROGRAMMING TUTORIAL

M.F. van Lieshout
TU/e, fac. ID
Or. 14-09-2005
Transl. 17-05-2006

Basic rules for programming in C

There are some basic rules when programming in C:

- Comments for only 1 line of code start with 2 slashes: //
  ```
  //This is a comment
  ```
  Add many comments to your code, otherwise it is very hard to remember how your program works after a few weeks!
- Comments for more than 1 line start with /* en end with */
  ```
  /*
  This is a comment.
  This is another comment.
  */
  ```
- At the end of each line with some instruction, a semi-colon (;) has to be placed.
  ```
  a=a+3;
  ```
- Parts of the program that belong together (functions, statements, etc.), are between { and }.
  ```
  void main(void)    //Function
  {
       //Add code
  }
  ```

The structure of the program

The structure of a program in C is as follows:

- Add libraries with functions.
  This is done with the following line of code:
  ```
  #include <filename.h>
  ```
- Declare (global) variables.
  The different types of variables will be discussed later in this document.
- Making prototypes of the functions.
  A prototype of a function ensures that the function can be called anywhere in the program. Without a prototype, only functions can be called if they are already declared above of the current function.
- Functions.
  Functions are very useful to make it easy to repeat tasks. They can have input and output variables. The output variable type is in front of the function name (int, char, …), the input variable is behind the function, between brackets:
  ```
  int calc(int x)
  ```
  The function is called in the following way:
  ```
  a=calc(3);
  ```
  The number 3 is used as input, in the variable a the result will be placed that is returned by the function calc.
- Main function.
  This is the function that will be called first when starting your microcontroller. From there, other functions are called.

```
//A comment begins with 2 slashes
//example
#include <18f4550.h>

//Declaring variables
int a, b, c, par1, d0;
char zz, s3, temp;
```

```
//Making prototypes
int calc (int p);

//Function
int calc (int p)
{
p=p+1;
//Add code
return p;
}

//Main function
void main(void)
{
//Add code
a=calc(3);
}
```

Variable types

Type	Memory usage	Possible values
bit	1 bit	0, 1
char	8 bits	-128…127
unsigned char	8 bits	0…255
signed char	8 bits	-128…127
int	16 bits	-32k7…32k7
unsigned int	16 bits	0…65k5
signed int	16 bits	-32k7…32k7
long int	32 bits	-2G1…2G1
unsigned long int	32 bits	0…4G3
signed long int	32 bits	-2G1…2G1
float	32 bits	± 10^(±38)
double	32 bits	± 10^(±38)

Possible ways to give variable a the decimal value 15:

```
a = 15;          //Decimal
a = 0b00001111;  //Binair
a = 0x0F;        //Hexadecimal
```

Operators

Relational operators:

>	Greater than
>=	Greater than or similar to
<	Less than
<=	Less than or similar to
==	Similar to
!=	Not similar to

Bit operators:

&	Bitwise AND
|	Bitwise OR

^	Bitwise XOR
<<	Shift to left
>>	Shift to right

Increasing and decreasing:
```
x--;    //This is the same as x = x – 1;
x++;    //This is the same as x = x + 1;
```

Example:
```
int a, b, c;
a = 19;      //a     00010011
b = 6;       //b     00001110

c = a & b;   //c     00000010 → 2
```

Mathematical operators:

+	Adding
-	Subtracting
*	Multiplication
/	Division
%	Modulus (= remainder after division)

Statements

IF...ELSE

```
if (a==0)   //If a is similar to 0...
{
    b++;    //...than increase b by 1
}
else        //otherwise
{
    b--;    //decrease b by 1
}
```

WHILE

```
while (a>3)     //As long as a is higher than 3
{
    b = b + 3;  //Add 3 to b
    a--;        //Decrease a by 1
}
```

FOR

```
for (i = 0 ; i < 100 ; i++)   //From i=0 to i=100 with step size 1
{
    b++;                      //Increase b by 1
    a--;                      //Decrease a by 1
}
```

Registers

The microcontroller is completely controlled with registers. To define outputs, a register has to be used, but also when using AD converters. Normally, registers are 8 bit. All the registers used in MPLAB MCC18 have exact the same name as the name stated in the datasheet. Registers can be set in different ways, some examples for the register PORTB will follow. PORTB is used for writing to and reading from pins on port B (take a look in the datasheet and find out yourself!). Writing a 1 to one of the bits of PORTB results in a high voltage on the corresponding output pin. A 0 will result in a low voltage. The MSB (most significant bit, the most left one) is used for pin B7, the LSB for pin B0.

```
PORTB = 0b11111111; //All pins of port B are made high
PORTB = 255;        //All pins of port B are made high
PORTB = 0xFF;       //All pins of port B are made high
PORTB = 0b10101010; //Pin B7 on, B6 off, B5 on, B4 off, etc.
PORTB = 170;        //Pin B7 on, B6 off, B5 on, B4 off, etc.
PORTB = 0xAA;       //Pin B7 on, B6 off, B5 on, B4 off, etc.
```

To set or reset one single bit in a register (one of the 8 bits), the register name is used and ‘bits.’ is added, as well as the name of the bit. The names of the bits are also mentioned in the datasheet. Some examples:

```
PORTBbits.RB0 = 1   //Pin B0 on
PORTBbits.RB7 = 0   //Pin B7 off
```

As said before: **everything** can be controlled with registers. A short list with some often used registers (a complete list can be found in the datasheet, of course):

```
//Input, output
PORTA     //PORT A output
PORTB     //PORT B output
PORTC     //PORT C output
PORTD     //PORT D output
PORTE     //PORT E output
TRISA     //PORT A direction
TRISB     //PORT B direction
TRISC     //PORT C direction
TRISD     //PORT D direction
TRISE     //PORT E direction
//Analog digital converter
ADCON0    //ADC settings
ADCON1    //ADC settings
ADCON2    //ADC settings
ADRESH    //AD result
ADRESL    //AD result
//Timers
TMR0L     //Timer 0 value
TMR0H     //Timer 0 value
T0CON     //Timer 0 settings
TMR1L     //Timer 1 value
TMR1H     //Timer 1 value
T1CON     //Timer 1 settings
TMR2      //Timer 2 value
T2CON     //Timer 2 settings
PR2       //Timer 2 period
TMR3L     //Timer 3 value
TMR3H     //Timer 3 value
T3CON     //Timer 3 settings
//Interrupts
RCON, INTCON, INTCON2, INTCON3, PIR1, PIR2, PIE1, PIE2, IPR1, IPR2
```

All registers are controlled in the same way!!! Some examples:

```
TRISB = 0b11110000;    //B4-B7 input, B0-B3 output
```

```
ADCON0bits.ADON = 1;    //ADC on
ADCON2bits.ADFM = 1;    //ADC result is right justified in ADRESL
                        //and ADRESH
INTCONbits.GIE = 0;     //All interrupts off
T1CONbits.TMR1ON = 1;   //Timer 1 on
```

Libraries

Furthermore, there are several libraries with standard functions available. For a manual with descriptions of all these functions and libraries, go to www.microchip.com , and look for documentation on the compiler MCC18. These libraries can be used by adding the #include statement with the library name:

```
#include <delays.h>
#include <adc.h>
```

Bootloader

When a bootloader is used, a special part of code has to beincluded in the program, after the #include statements:

```
//Always include this code, it's necessary when using a bootloader
extern void _startup (void);
#pragma code _RESET_INTERRUPT_VECTOR = 0x000800
void _reset (void)
{
    _asm goto _startup _endasm
}
#pragma code

#pragma code _HIGH_INTERRUPT_VECTOR = 0x000808
void _high_ISR (void)
{
    ;
}
#pragma code

#pragma code _LOW_INTERRUPT_VECTOR = 0x000818
void _low_ISR (void)
{
    ;
}
#pragma code
//End bootloader code
```