CHAPTER 2

mikroC Pro for PIC Programming Language

Chapter Outline

Some of the popular C compilers used in the development of commercial, industrial, and educational programmable interface controller (PIC) 18 microcontroller applications are

- mikroC Pro for PIC C compiler
- PICC18 C compiler
- MPLAB C18 C compiler
- MPLAB XC8 C Compiler
- CCS C compiler

mikroC Pro for PIC C compiler has been developed by MikroElektronika (web site: www.microe.com) and is one of the easy-to-learn compilers with rich resources, such as a large number of library functions and an integrated development environment with built-in simulator, and an in-circuit debugger (e.g. mikroICD). A demo version of the compiler with a 2 K program limit is available from MikroElektronika.

PICC18 C compiler is another popular C compiler, developed by Hi-Tech Software (web site: www.htsoft.com). This compiler has two versions: the standard compiler and the professional version. A powerful simulator and an integrated development environment

PIC Microcontroller Projects in C. http://dx.doi.org/10.1016/B978-0-08-099924-1.00002-2

(Hi-Tide) is provided by the company. PICC18 is supported by the PROTEUS simulator (www.labcenter.co.uk) that can be used to simulate PIC microcontroller-based systems.

MPLAB C18 C compiler is a product of Microchip Inc. (web site: www.microchip.com). A limited-period demo version and a limited functionality version with no time limit of C18 are available from the Microchip web site. C18 includes a simulator and supports hardware and software development tools.

MPLAB XC8 C compiler is the latest C compiler from Microchip Inc. that supports all their 8-bit family of microcontrollers. The compiler is available for download free of charge.

CCS C compiler has been developed by Custom Computer Systems Inc. (web site: www.ccsinfo.com). The company provides a limited-period demo version of their compiler. CCS compiler provides a large number of built-in functions and supports an in-circuit debugger.

In this book, we shall be using the two popular C languages: mikroC Pro for PIC and MPLAB XC8. The details of mikroC Pro for PIC are given in this chapter. MPLAB XC8 is covered in detail in the next chapter.

2.1 Structure of a mikroC Pro for PIC Program

Figure 2.1 shows the simplest structure of a mikroC Pro for PIC program. This program flashes a light-emitting diode (LED) connected to port RB0 (bit 0 of PORTB) of a PIC

```
/*****************************************************************

                        LED FLASHING PROGRAM
                        ************************

 This program flashes an LED connected to port pin RB0 of PORTB with one
 second intervals.

  Programmer  : D. Ibrahim
  File        : LED.C
  Date        : July, 2013
  Micro       : PIC18F45K22
*****************************************************************/

void main()
{
    for(;;)                         // Endless loop
    {
        ANSELB = 0;                 // Configure PORTB digital
        TRISB = 0;                  // Configure PORTB as output
        PORTB.0 = 0;                // RB0 = 0
        Delay_Ms(1000);             // Wait 1 s
        PORTB.0 = 1;                // RB0 = 1
        Delay_Ms(1000);             // Wait 1 s
    }                               // End of loop
}
```

Figure 2.1: Structure of a Simple mikroC Pro for PIC Program.

microcontroller with 1 s intervals. Do not worry if you do not understand the operation of the program at this stage as all will be clear as we progress through this chapter. Some of the programming statements used in Figure 2.1 are described below in detail.

Comments are used by programmers to clarify the operation of the program or a programming statement. Two types of comments can be used in mikroC Pro for PIC programs: long comments extending several lines and short comments occupying only a single line. As shown in Figure 2.1, long comments start with characters "/*" and terminate with characters "*/". Similarly, short comments start with characters "//" and there is no need to terminate short comments.

In general, C language is case sensitive and variables with lower case names are different from those with upper case names. Currently, mikroC Pro for PIC variables are not case sensitive. The only exception is that identifiers **main** and **interrupt** must be written in lower case in mikroC Pro for PIC. In this book, we shall be assuming that the variables are case sensitive for compatibility with other C compilers and variables with same names but different cases shall not be used.

In C language, variable names can begin with an alphabetical character or with the underscore character. In essence, variable names can be any of the characters a−z and A−Z, the digits 0−9, and the underscore character "_". Each variable name should be unique within the first 31 characters of its name. Variable names can contain upper case and lower case characters and numeric characters can be used inside a variable name. Some names are reserved for the compiler itself and they cannot be used as variable names in our programs.

mikroC Pro for PIC language supports the variable types shown in Table 2.1.

Table 2.1: mikroC Pro for PIC Variable Types

Type	Size (Bits)	Range
unsigned char	8	0 to 255
unsigned short int	8	0 to 255
unsigned int	16	0 to 65535
unsigned long int	32	0 to 4294967295
signed char	8	−128 to 127
signed short int	8	−128 to 127
signed int	16	−32768 to 32767
signed long int	32	−2147483648 to 2147483647
float	32	±1.17549435082E-38 to ±6.80564774407E38
double	32	±1.17549435082E-38 to ±6.80564774407E38
long double	32	±1.17549435082E-38 to ±6.80564774407E38

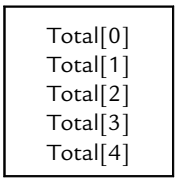
Constants represent fixed values (numeric or character) in programs that cannot be changed. Constants are stored in the flash program memory of the PIC microcontroller; thus, the valuable and limited random-access memory (RAM) is not wasted.

2.2 Arrays

An array is declared by specifying its type, name, and the number of elements it will store. For example,

```
unsigned int Total[5];
```

Creates an array of type unsigned int, with name Total, and having five elements. The first element of an array is indexed with 0. Thus, in the above example, Total[0] refers to the first element of this. The array Total is stored in memory in five consecutive locations as follows:



Data can be stored in the array by specifying the array name and index. For example, to store 25 in the second element of the array, we have to write:

```
Total[1] = 25;
```

Similarly, the contents of an array can be read by specifying the array name and its index. For example, to copy the third array element to a variable called temp we have to write:

```
Temp = Total[2];
```

The contents of an array can be initialized during its declaration. An example is given below where array months has 12 elements and months[0] = 31, months[1] = 28, and so on.

```
unsigned char months[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

The above array can also be declared without specifying the size of the array:

```
unsigned char months[ ] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

Character arrays can be declared similarly. In the following example, a character array named Hex_Letters is declared with six elements:

```
unsigned char Hex_Letters[ ] = {'A', 'B', 'C', 'D', 'E', 'F'};
```

Strings are character arrays with a null terminator. Strings can either be declared by enclosing the string in double quotes, or each character of the array can be specified within single quotes, and then terminated with a null character:

```
unsigned char Mystring[ ] = "COMP";
```

And

```
unsigned char Mystring[ ] = {'C', 'O', 'M', 'P', '\0'};
```

In C programming language, we can also declare arrays with multiple dimensions. In the following example, a two-dimensional array named P is created having three rows and four columns. Altogether, the array has 12 elements. The first element of the array is P[0][0], and the last element is P[2][3]. The structure of this array is shown below:

P[0][0]	P[0][1]	P[0][2]	P[0][3]
P[1][0]	P[1][1]	P[1][2]	P[1][3]
P[2][0]	P[2][1]	P[2][2]	P[2][3]

2.3 Pointers

Pointers are an important part of the C language and they hold the memory addresses of variables. Pointers are declared same as any other variables, but with the character ("*") in front of the variable name. In general, pointers can be created to point to (or hold the addresses of) character variables, integer variables, long variables, floating point variables, or functions.

In the following example, an unsigned character pointer named pnt is declared:

```
unsigned char *pnt;
```

When a new pointer is created, its content is initially unspecified and it does not hold the address of any variable. We can assign the address of a variable to a pointer using the ("&") character:

```
pnt = &Count;
```

Now pnt holds the address of variable Count. Variable Count can be set to a value by using the character ("*") in front of its pointer. For example, Count can be set to 10 using its pointer:

```
*pnt = 10;      //Count = 10
```

Which is same as

```
Count = 10;      //Count = 10
```

Or, the value of Count can be copied to variable Cnt using its pointer:

```
Cnt = *pnt;     //Cnt = Count
```

2.4 Structures

A structure is created by using the keyword **struct**, followed by a structure name, and a list of member declarations. Optionally, variables of the same type as the structure can be declared at the end of the structure.

The following example declares a structure named Person:

```
struct Person
{
  unsigned char name[20];
  unsigned char surname[20];
  unsigned char nationality[20];
  unsigned char age;
}
```

Declaring a structure does not occupy any space in memory, but the compiler creates a template describing the names and types of the data objects or member elements that will eventually be stored within such a structure variable. It is only when variables of the same type as the structure are created that these variables occupy space in memory. For example, two variables **Me** and **You** of type Person can be created by the statement:

```
struct Person Me, You;
```

Variables of type Person can also be created during the declaration of the structure as shown below:

```
struct Person
{
  unsigned char name[20];
  unsigned char surname[20];
  unsigned char nationality[20];
  unsigned char age;
} Me, You;
```

We can assign values to members of a structure by specifying the name of the structure, followed by a dot (“.”), and the name of the member. In the following example, the **age** of structure variable **Me** is set to 25, and variable **M** is assigned to the value of **age** in structure variable **You**:

```
Me.age = 25;
M = You.age;
```

Structure members can be initialized during the declaration of the structure. In the following example, the radius and height of structure Cylinder are initialized to 1.2 and 2.5, respectively:

```
struct Cylinder
{
  float radius;
  float height;
} MyCylinder = {1.2, 2.5};
```

2.5 Operators in C

Operators are applied to variables and other objects in expressions and they cause some conditions or some computations to occur.

mikroC Pro for PIC language supports the following operators:

- Arithmetic operators
- Logical operators
- Bitwise operators
- Conditional operators
- Assignment operators
- Relational operators
- Preprocessor operators

2.6 Modifying the Flow of Control

Statements are normally executed sequentially from the beginning to the end of a program. We can use control statements to modify the normal sequential flow of control in a C program. The following control statements are available in mikroC Pro for PIC programs:

- Selection statements
- Unconditional modification of flow
- Iteration statements

There are two selection statements: **If** and **switch**.

If Statement

The general format of the **if** statement is

```
if(expression)
  Statement1;
else
  Statement2;
```

or,

```
if(expression)Statement1; else Statement2;
```

In the following example, if the value of **x** is greater than **MAX** then variable **P** is incremented by 1, otherwise it is decremented by 1:

```
if(x > MAX)
  P++;
else
  P--;
```

We can have more than one statement by enclosing the statements within curly brackets. For example,

```
if(x > MAX)
{
  P++;
  Cnt = P;
  Sum = Sum + Cnt;
}
else
  P--;
```

In the above example, if **x** is greater than **MAX** then the three statements within the curly brackets are executed, otherwise the statement **P**−− is executed.

Another example using the **if** statement is given below:

```
if(x > 0 && x < 10)
{
  Total + = Sum;
  Sum++;
}
else
{
  Total = 0;
  Sum = 0;
}
```

The **switch** statement is used when there are a number of conditions and different operations are performed when a condition is true. The syntax of the **switch** statement is

```
switch (condition)
{
 case condition1:
  Statements;
  break;
 case condition2:
  Statements;
```

```
  break;
.....
.....
 case condition:
  Statements;
  break;
 default:
  Statements;
}
```

In mikroC Pro for PIC there are four ways that iteration can be performed and we will look at each one with examples:

- Using **for** statement
- Using **while** statement
- Using **do** statement
- Using **goto** statement

for Statement

The syntax of the **for** statement is

```
for(initial expression; condition expression; increment expression)
{
  Statements;
}
```

The following example shows how a loop can be set up to execute 10 times. In this example, variable **i** starts from 0 and increments by 1 at the end of each iteration. The loop terminates when i = 10, in which case the condition i < 10 becomes false. On exit from the loop, the value of **i** is 10:

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

The parameters of a **for** loop are all optional and can be omitted. If the **condition expression** is left out, it is assumed to be true. In the following example, an endless loop is formed where the **condition expression** is always true and the value of **i** starts with 0 and is incremented after each iteration:

```
/* Endless loop with incrementing i */
for(i = 0; ; i++)
{
  Statements;
}
```

Another example of an endless loop is given below where all the parameters are omitted:

```
/* Example of endless loop */
for(; ;)
{
  Statements;
}
```

while *Statement*

This is another statement that can be used to create iteration in programs. The syntax of the **while** statement is

```
while (condition)
{
  Statements;
}
```

The following code shows how to set up a loop to execute 10 times using the **while** statement:

```
/* A loop that executes 10 times */
k = 0;
while (k < 10)
{
  Statements;
  k++;
}
```

At the beginning of the code, variable **k** is 0. Since **k** is less than 10, the **while** loop starts. Inside the loop, the value of **k** is incremented by 1 after each iteration. The loop repeats as long as $k < 10$ and is terminated when $k = 10$. At the end of the loop, the value of **k** is 10.

Notice that an endless loop will be formed if **k** is not incremented inside the loop:

```
/* An endless loop */
k = 0;
while (k < 10)
{
  Statements;
}
```

An endless loop can also be formed by setting the **condition** to be always true:

```
/* An endless loop */
while (k = k)
{
  Statements;
}
```

do *Statement*

The **do** statement is similar to the **while** statement but here the loop executes until the **condition** becomes false, or the loop executes as long as the **condition** is true. The **condition** is tested at the end of the loop. The syntax of the **do** statement is

```
do
{
  Statements;
} while (condition);
```

The following code shows how to set up a loop to execute 10 times using the **do** statement:

```
/* Execute 10 times */
k = 0;
do
{
  Statements;
  k++;
} while (k < 10);
```

An endless loop will be formed if the condition is not modified inside the loop as shown in the following example. Here **k** is always less than 10:

```
/* An endless loop */
k = 0;
do
{
  Statements;
} while (k < 10);
```

An endless loop can also be created if the condition is set to be true all the time.

goto Statement

Although not recommended, the **goto** statement can be used together with the **if** statement to create iterations in a program. The following example shows how to set up a loop to execute 10 times using the **goto** and **if** statements:

```
/* Execute 10 times */
k = 0;
```

Loop:

```
Statements;
k++;
if(k < 10)goto Loop;
```

2.7 mikroC Pro for PIC Functions

An example function definition is shown below. This function, named **Mult**, receives two integer arguments **a** and **b** and returns their product. Notice that using brackets in a return statement are optional:

```
int Mult(int a, int b)
{
  return (a*b);
}
```

When a function is called, it generally expects to be given the number of arguments expressed in the function's argument list. For example, the above function can be called as

```
z = Mult(x,y);
```

Where variable **z** has the data type **int**. In the above example, when the function is called, variable **x** is copied to **a**, and variable **y** is copied to **b** on entry to function **Mult**.

Some functions do not return any data and the data type of such functions must be declared as **void**. An example is given below:

```
void LED(unsigned char D)
{
  PORTB = D;
}
```

void functions can be called without any assignment statements, but the brackets must be used to tell the compiler that a function call is made.

2.8 mikroC Pro for PIC Library Functions

mikroC Pro for PIC provides a large set of library functions that can be used in our programs. mikroC Pro for PIC user manual gives detailed descriptions of each library function with examples. Table 2.2 gives a list of the mikroC Pro for PIC library functions, organized in functional order.

2.9 Summary

This chapter presented an introduction to the mikroC Pro for PIC language. A C program may contain a number of functions and variables and a main program. The beginning of the main program is indicated by the statement **void main()**.

A variable stores a value used during the computation. All variables in C must be declared before they are used. A variable can be an 8-bit character, a 16-bit integer, a 32-bit long,

Table 2.2: mikroC Pro for PIC Library Functions

Library	Description
ADC	Analog-to-digital conversion functions
CAN	CAN bus functions
CANSPI	SPI-based CAN bus functions
Compact flash	Compact flash memory functions
EEPROM	EEPROM memory read/write functions
Ethernet	Ethernet functions
SPI ethernet	SPI-based ethernet functions
Flash memory	Flash memory functions
Graphics LCD	Standard graphics LCD functions
T6963C graphics LCD	T6963-based graphics LCD functions
I^2C	I^2C bus functions
Keypad	Keypad functions
LCD	Standard LCD functions
Manchester code	Manchester code functions
Multimedia	Multimedia functions
One Wire	One Wire functions
PS/2	PS/2 functions
PWM	PWM functions
RS-485	RS-485 communication functions
Sound	Sound functions
SPI	SPI bus functions
USART	USART serial communication functions
Util	Utilities functions
SPI graphics LCD	SPI-based graphics LCD functions
Port expander	Port expander functions
SPI LCD	SPI-based LCD functions
ANSI C Ctype	C Ctype functions
ANSI C Math	C Math functions
ANSI C Stdlib	C Stdlib functions
ANSI C String	C String functions
Conversion	Conversion functions
Trigonometry	Trigonometry functions
Time	Time functions

or a floating point number. Constants are stored in the flash program memory of PIC microcontrollers and thus using them saves valuable and limited RAM.

Various flow control and iteration statements such as **if**, **switch**, **while**, **do**, **break**, and so on have been described in the chapter with examples.

Pointers are used to store the addresses of variables. As we shall see in the next chapter, pointers can be used to pass information back and forth between a function and its calling point. For example, pointers can be used to pass variables between a main program and a function.

Library functions simplify programmers' tasks by providing ready and tested routines that can be called and used in our programs. Examples are also given on how to use various library functions in our main programs.

2.10 Exercises

1. Write a C program to set bits 0 and 7 of PORTC to logic 1.
2. Write a C program to count down continuously and send the count to PORTB.
3. Write a C program to multiply each element of a 10 element array with number 2.
4. It is required to write a C program to add two matrices **P** and **Q**. Assume that the dimension of each matrix is 3×3 and store the result in another matrix called **W**.
5. What is meant by program repetition? Describe the operation of the **while**, **do-while**, and **for** loops in C.
6. What is an array? Write example statements to define the following arrays:
   a. An array of 10 integers.
   b. An array of 30 floats.
   c. A two-dimensional array having 6 rows and 10 columns.
7. How many times do each of the following loops iterate and what is the final value of the variable **j** in each case?
   a. `for(j = 0; j < 5; j++)`
   b. `for(j = 1; j < 10; j++)`
   c. `for(j = 0; j <= 10; j++)`
   d. `for(j = 0; j <= 10; j += 2)`
   e. `for(j = 10; j > 0; j –= 2)`
8. Write a program to calculate the average value of the numbers stored in an array. Assume that the array is called **M** and it has 20 elements.
9. Derive equivalent **if-else** statements for the following tests:
   a. `(a > b) ? 0 : 1`
   b. `(x < y) ? (a > b) : (c > d)`
10. What can you say about the following **for** loop:

```
Cnt = 0;
for(;;)
{
 Cnt++;
}
```

11. Write a function to calculate the circumference of a rectangle. The function should receive the two sides of the rectangle as floating point numbers and then return the circumference as a floating point number.
12. Write a function to convert inches to centimeters. The function should receive inches as a floating point number and then calculate the equivalent centimeters.
13. An LED is connected to port pin RB7 of a PIC18F45K22 microcontroller. Write a program to flash the LED such that the ON time is 5 s and the OFF time is 3 s.
14. Write a function to perform the following operations on two-dimensional matrices:
    a. Add matrices
    b. Subtract matrices
    c. Multiply matrices