Chapter 15

C Extra

In Chapter 2 we saw how to turn outputs on and off using C18 statements. Use the delay routines and execute a while loop.

In Chapter 3 we used the for loop.

The if statement was introduced in Chapters 4 and 5 and a modification of this the case statement was discussed in Chapter 14.

Chapter 14 showed how powerful the C language is for the microcontroller by calculating the value of C in the expression:

$$C = -\frac{t}{R(\log(1 - \frac{V}{V_{max}}))}$$

We have of course seen how to add, subtract, multiply, and divide variables and have used the "log" function to evaluate C in the above expression. But there are numerous other **mathematical functions** that we can use in our programs, taken from MPLAB C18 C Compiler Libraries, such as:

Function	**Description**
acos	Compute the inverse cosine (arccosine).
asin	Compute the inverse sine (arcsine).
atan	Compute the inverse tangent (arctangent).
atan2	Compute the inverse tangent (arctangent) of a ratio.
ceil	Compute the ceiling (least integer).
cos	Compute the cosine.
cosh	Compute the hyperbolic cosine.
exp	Compute the exponential e^x.
fabs	Compute the absolute value.
floor	Compute the floor (greatest integer).
fmod	Compute the remainder.
frexp	Split into fraction and exponent.
ieeetomchp	Convert an IEEE-754 format 32-bit floating point value Microchip 32-bit floating point format.
ldexp	Load exponent—compute x *2^n

log	Compute the natural logarithm.
log10	Compute the common (base 10) logarithm.
mchptoieee	Convert a Microchip format 32-bit floating point value IEEE-754 32-bit floating point format.
modf	Compute the modulus.
pow	Compute the exponential x^y.
sin	Compute the sine.
sinh	Compute the hyperbolic sine.
sqrt	Compute the square root.
tan	Compute the tangent.
tanh	Compute the hyperbolic tangent.

acos

Function: Compute the inverse cosine (arccosine)
Include: math.h
Prototype: float acos(float x);
Remarks: This function computes the inverse cosine (arccosine) of the argument x, which must be between –1 and +1. Arguments outside the permitted range produce domain errors and the result is NaN.
Return Value: The returned value is the arccosine in radians and is between 0 and π.
File Name: acos.c

asin

Function: Compute the inverse sine (arcsine).
Include: math.h
Prototype: float asin(float x);
Remarks: This function computes the inverse sine (arcsine) of the argument x, which must be between −1 and +1. Arguments outside the permitted range produce domain errors and the result is NaN.
Return Value: The returned value is the arcsine in radians and is between −π/2 and π/2.
File Name: asin.c

atan

Function: Compute the inverse tangent (arctangent).
Include: math.h
Prototype: float atan(float x);
Remarks: This function computes the inverse tangent (arctangent) of the argument x. If x is a NaN, a domain error occurs and the value returned is NaN.
Return Value: The returned value is in radians and between −π/2 and π/2.
File Name: atan.c

atan2

Function: Compute the inverse tangent (arctangent) of a ratio.
Include: math.h
Prototype: float atan2(float y, float x);
Remarks: This function computes the inverse tangent (arctangent) of y/x. If x or y is NaN, a domain occurs and the value returned is NaN. If x is a NaN, or if $x = y = 0$, or if $x = y = \infty$, a domain error occurs and the value returned is NaN.
Return Value: The returned value is in radians and between $-\pi$ and π.
File Name: atan2.c

ceil

Function: Compute the ceiling (least integer).
Include: math.h
Prototype: float ceil (float x);
Remarks: None.
Return Value: The smallest integer greater than or equal to x.
File Name: ceil.c

cos

Function: Compute the cosine.
Include: math.h
Prototype: float cos (float x);
Remarks: Computes the cosine of x (in radians). A domain error results from an argument that is infinite or NaN. Both cases return NaN.
Return Value: The cosine of argument x.
File Name: cos.c

cosh

Function: Compute the hyperbolic cosine.
Include: math.h
Prototype: float cosh (float x);
Remarks: None.
Return Value: The hyperbolic cosine of argument x.
File Name: cosh.c

exp

Function: Compute the exponential ex.
Include: math.h
Prototype: float exp (float x);
Remarks: A range error occurs if the magnitude of x is too large. The range of this function is limited to values for the exponent of between approximately

−103.2789 and 88.722283. The minimum value of the result is 2^{-149} and the maximum is 2^{127}.
Return Value: The value of the exponential e^x.
File Name: exp.c

fabs

Function: Compute the absolute value.
Include: math.h
Prototype: float fabs(float x);
Remarks: For floating point arguments that are zeroes and infinities, the return value is the argument with the sign bit cleared.
Return Value: The absolute value of x.
File Name: fabs.c

floor

Function: Compute the floor (greatest integer).
Include: math.h
Prototype: float floor(float x);
Remarks: None.
Return Value: The largest integer less than or equal to x.
File Name: floor.c

fmod

Function: Compute the remainder.
Include: math.h
Prototype: float fmod(float x, float y);
Remarks: None.
Return Value: The remainder for x modulo y.
File Name: fmod.c

frexp

Function: Split into fraction and exponent.
Include: math.h
Prototype: float frexp(float x, int *pexp);
Remarks: Separates the argument x into two parts that fit this formula: x = frexp(x, *pexp) x 2*pexp
The integer value, which is stored at location pexp, is chosen so that the fractional portion of the result is between ½ and 1.
Return Value: Fractional result that satisfies the conditions listed above.
File Name: frexp.c

ieeetomchp

Function: Convert an IEEE-754 format 32-bit floating point value into the Microchip 32-bit floating point format.
Include: math.h
Prototype: unsigned long ieeetomchp(float v);
Remarks: This function adjusts the sign bit of the floating point representation to be located as required by the Microchip format: eb f0 f1 f2
IEEE-754 32-bit seee eeee exxx xxxx xxxx xxxx xxxx xxxx
Microchip 32-bit eeee eeee sxxx xxxx xxxx xxxx xxxx xxxx
s = sign bit e = exponent x = significand
Return Value: The converted 32-bit value.
File Name: ieeetomchp.c

ldexp

Function: Load exponent—compute x * 2^n
Include: math.h
Prototype: float ldexp(float x, int n);
Remarks: None.
Return Value: Returns the value of x * 2^n.
File Name: ldexp.c

log

Function: Compute the natural logarithm.
Include: math.h
Prototype: float log(float x);
Remarks: A domain error occurs if the argument is not in the interval [0, +∞].
Return Value: Natural logarithm of x.
File Name: log.c

log10

Function: Compute the common (base 10) logarithm.
Include: math.h
Prototype: float log10(float x);
Remarks: A domain error occurs if the argument is not in the interval [0, +∞].
Return Value: log10x.
File Name: log10.c
mchptoieee
Function: Convert a Microchip format 32-bit floating point value into the IEEE-754 32-bit floating point format.

Include: math.h
Prototype: float ieeetomchp(unsigned long v);
Remarks: This function adjusts the sign bit of the floating point representation to be located as required by the IEEE format:
eb f0 f1 f2
IEEE-754 32-bit seee eeee exxx xxxx xxxx xxxx xxxx xxxx
Microchip 32-bit eeee eeee sxxx xxxx xxxx xxxx xxxx xxxx
s = sign bit e = exponent x = significand
Return Value: The converted floating point value.
File Name: mchptoieee.c

modf

Function: Compute the modulus.
Include: math.h
Prototype: float modf(float x, float *ipart);
Remarks: This function separates the argument x into integer and fractional parts. The fractional part is returned, and the integer part is stored at location ipart. If the argument is NaN, the results for both the fractional and integer part will be NaN as well.
Return Value: Fractional portion of x.
File Name: modf.c

pow

Function: Compute the exponential x^y
Include: math.h
Prototype: float pow(float x, float y);
Remarks: Domain errors occur if x is finite and negative, and y is finite and not an integer; also if x is zero and y is less than or equal to zero. A range error occurs if xy is too large or too small to be represented. In such a case, a correctly signed infinity or zero is returned and a range error is signaled.
Return Value: x^y.
File Name: pow.c

sin

Function: Compute the sine.
Include: math.h
Prototype: float sin(float x);
Remarks: Computes the sine of x (in radians). A domain error results from an argument that is infinite or NaN. Both cases return NaN.
Return Value: The sine of x.
File Name: sin.c

sinh

Function: Compute the hyperbolic sine.
Include: math.h
Prototype: float sinh(float x);
Remarks: None.
Return Value: The hyperbolic sine of argument x.
File Name: sinh.c

sqrt

Function: Compute the square root.
Include: math.h
Prototype: float sqrt(float x);
Remarks: A domain error occurs if the argument x is strictly negative. The principal square root exists and is computable for every non-negative floating point number x.
Return Value: The square root of x.
File Name: sqrt.c

tan

Function: Compute the tangent.
Include: math.h
Prototype: float tan(float x);
Remarks: Computes the tangent of x (in radians). A domain error occurs if the argument is infinite or NaN. Both cases return NaN.
Return Value: The tangent of x.
File Name: tan.c

tanh

Function: Compute the hyperbolic tangent.
Include: math.h
Prototype: float tanh(float x);
Remarks: If the argument is NaN, the return value is NaN.
Return Value: The hyperbolic tangent of x.
File Name: tanh.c

As an example of the use of these functions let us consider the square root, **sqrt**. We have already seen an application of the **log** function.

Suppose we wish to calculate the velocity, v ms^{-1}at which an object hits the ground when falling from a height, s. Newton's law to solve this is:

$v^2 = u^2+2gs$ where u is the initial velocity, 0 in our case and g is the acceleration due to gravity. $g = 9.80665\,ms^{-2}$

So $v^2=2gs$

$v = \sqrt{2gs}$ in C $v = \text{sqrt}(2 * g * s);$

The easiest way to observe the program running is in MPLAB SIM because we are not going to build a circuit to measure height.

I have taken the header program and added the relevant lines shown in Figure 15.1.

MPLAB is used in the simulator mode and the watch window has been added to view the value of *v*. The watch window was not needed because putting the mouse over *v* in line 20 would have shown the value of *v*.

The program is run and will halt at the break point line 22.

The program has included the variables in line 8 and the calculation on line 20.

With a height of 10 m the program shows an impact velocity of 14.00475 ms^{-1}.

As an exercise you could try adding a keypad and a display and writing the code so that you can enter the height and display the impact velocity on the screen in ms^{-1} and/or mph.

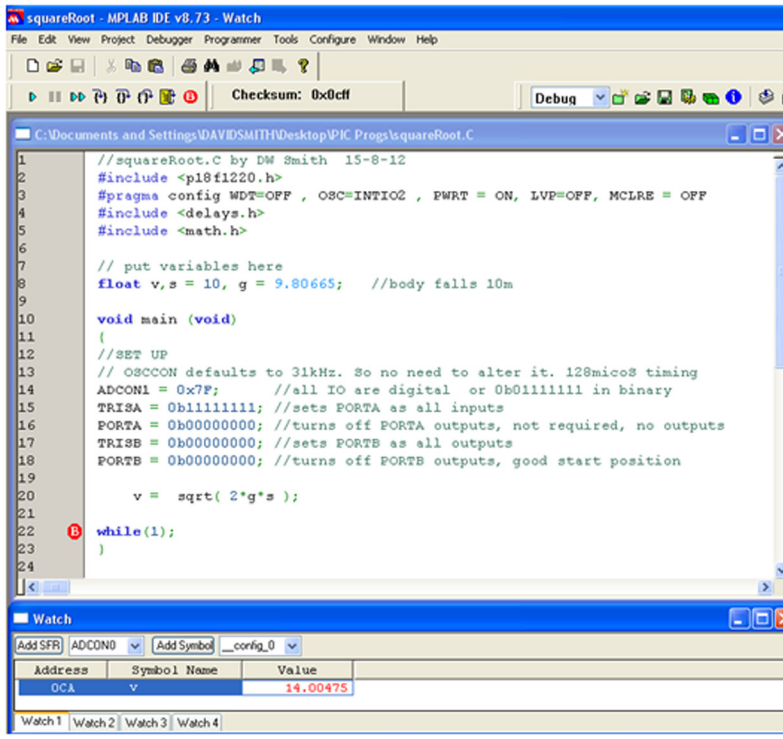

FIGURE 15.1 Calculating the impact velocity.

DATA TYPES

We have used different data types in our programs such as, **char**, **int,** and **float**. There are several data types shown below with their range of values that can also be used.

Data Type	Memory Size (bits)	Minimum Value	Maximum Value
Char	8	−128	127
signed char	8	−128	127
unsigned char	8	0	255
Int	16	−32,768	32,767
unsigned int	16	0	65,535
Short	16	−32,768	32,767
unsigned short	16	0	65,535
short long	24	−8,388,608	8,388,607
unsigned short long	24	0	16,777,215
Long	32	−2,147,483,648	2,147,483,648
unsigned long	32	0	4,294,967,295
float	32	$1.17549435e^{-38}$	$6.80564693e^{-38}$
double	32	$1.17549435e^{-38}$	$6.80564693e^{-38}$

In this book we have used **char**, **int,** and **float**. When possible use **char** because that will save on memory in the micro. If you need decimal places in your variable then you will need to use the **float** variable. When using a single digit as in temperatureTens to show on an LCD then use **char** and for integers <127. For larger digits use **int**.

CONDITIONAL OPERATORS

We have already seen the conditional operator, the double equals sign,==. This was used in an if statement to check if a value was equal to something.

Other conditional operators available are:

greater than
< less than
≥ greater than or equal to
≤ less than or equal to
!= not equal to
! not
&& and
|| or