GOTO

Here, we'll discuss something that many scientific programmers think should be kicked out of C language all together. That is the **goto** command. While I want to introduce you to **goto**, I encourage you not to use it because it can cause a lot of problems if you make a mistake. Basically, **goto**, forces to the code to go to somewhere else. In practice, it is a sloppy way to exit a loop.

The goto is initiated by

```
goto some_word;
```

where *some_word* can be any text aside from the special words that we are not allowed to use (see earlier section for a list of those special words). The code will then jump to the following line, wherever it is in that particular function.

```
some_word:;
```

Note the colon after *some_word.* Also, some compilers don't like it if you don't follow the colon with a semicolon.

Another thing about **goto.** You can only *goto* places within the same function. In other words, you cannot jump across functions. The following example shows how **goto** can be used to kick you out of a loop if a condition is satisfied.

code14.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(void)
{

// declare an integer, pp, and set it equal to 0.
int pp = 0;

    // By putting a 1 in the condition, we ensure that the loop
    // will continue forever because 1 means TRUE.
    while (1)
    {

        // If pp is greater than 10, get out of the loop using goto.
        // the goto will take you to the place where the following
        // text is seen again.

        if (pp > 10) goto i_am_so_out_of_here;
```

```
        // Lets print some stuff out
        fprintf(stdout,"The value of pp is :%d\n", pp);

        // If we're still here, lets make pp one bigger.
        pp++;
    }

// here is where we will transport to from the goto command

i_am_so_out_of_here:;

// lets brag about that
fprintf(stdout,"I made it out of the loop\n");

return 0;
}
```

The output should be:

```
The value of pp is :0
The value of pp is :1
The value of pp is :2
The value of pp is :3
The value of pp is :4
The value of pp is :5
The value of pp is :6
The value of pp is :7
The value of pp is :8
The value of pp is :9
The value of pp is :10
I made it out of the loop
```

Some Math Functions

The next code will introduce you to some of the math functions available from the math library. Remember, to activate the math library, your code must have the line:

```
#include<math.>
```

near the beginning, and you must type the **–lm** flag when you compile.

gcc –lm ***your_code***

Before we get to the code, it is important that you realize that you CANNOT mix up integers and real numbers. They are both intrinsically different creatures in the code world. If you put an integer in place of where a double is expected, you may have some serious trouble, and vice versa. This is very important!!! C leaves it up to us to make sure we don't make this mistake.

code15.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(void)
{

// Lets declare some integers and doubles to play with

int integer1 = -5;
int integer2 = 2;

double number1=-26.234;
double number2= 0.7222;
double number3 = 3.0;
double number4 = 5.0;

// We'll use "result" for doubles and "iresult" for integers.
// Don't mix these up!!!
double result;
int iresult;

// Lets explore some math

// To find the absolute value of a real number, use "fabs()"
  result = fabs(number1);
  fprintf(stdout," The absolute value of %f is %f\n", number1,result);

// To find the absolute value of an integer number, use "abs()"
  iresult = abs(integer1);
  fprintf(stdout," The absolute value of %d is %d\n", integer1,iresult);

// To calculate a number to the power of another number, use "pow()".
// The pow() function takes two arguments.  The first is raised to the second"
// Lets calculate the number two cubed.

  result = pow(2.0,3.0);
  fprintf(stdout,"Two cubed is: %f\n", result);

// We could have done the same thing with variables.

  result = pow(number3, number4);
  fprintf(stdout,"%f to the %f power is %f\n", number3,number4,result);

// We can calculate the square root 2 different ways.
// We could use either the pow() function or the sqrt() function.

  // calculate the square root of 10
  result = pow(10.0,0.5);
  fprintf(stdout,"The square root of 10 is: %f\n",result);

  // OR we could do it this way;

  result = sqrt(10.0);
```

```
    fprintf(stdout,"The square root of 10 is: %f\n",result);

  // The math library also has many constants available.
  // Here, we'll mainly just use pi, which is M_PI.

    fprintf(stdout,"The value of pi is: %f\n", M_PI);

  // Lets print the cosine of pi.
  // Note how we can include functions within our print statments.

    fprintf(stdout,"The cosine of PI is %f\n", cos(M_PI));

  return 0;
  }
```

The output should be:

```
The absolute value of -26.234000 is 26.234000
 The absolute value of -5 is 5
Two cubed is: 8.000000
3.000000 to the 5.000000 power is 243.000000
The square root of 10 is: 3.162278
The square root of 10 is: 3.162278
The value of pi is: 3.141593
The cosine of PI is -1.000000
```

Here is a table listing the most common math functions. There are more! One very important thing to note is this: **Trigonometric functions always assume that the arguments are in radians!** Never forget this!

common double functions

In these examples, the argument (here it is *x*, but it could be any variable name) and the resulting value are doubles. You can actually use floats with these too, but we will only use doubles in this course.

sin(x)	sine of x. Make sure x is in radians.
cos(x)	cosine of x. Make sure x is in radians.
tan(x)	tangent of x. Make sure x is in radians.
asin(x)	arcsine of x. Answer is in radians.
acos(x)	arccosine of x. Answer is in radians.
atan(y,x)	arctangent where x and y are the sides of the right triangle. Answer is in radians.
log(x)	natural logarithm of x.
log10(x)	10-based logarithm of x.
sqrt(x)	square root of x.
exp(x)	the exponential of x.
pow(x,y)	x raised to the power of y
fabs(x)	the absolute value of x
floor(x)	rounds x down to an integer value. The result is still a double.
ceil(x)	rounds x up to an integer value. The result is still a double.
sinh(x), cosh(x), and tanh(x)	Hyperbolic trig functions.

modulus operator

The % operator is very useful, particularly for determining whether an integer is odd or even. Remember that % returns the remainder of one number divided by another. For example 8%3=2 because 8 divided by 3 equals 2 and 2/3. Therefore, the remainder is 2. The following code shows an example of how to determine whether an integer is even or odd, using the modulus operator. If you divide an even number by 2, the remainder will always be zero, and if you divide an odd number by 2, the remainder will always be one.

code16.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(void)
{

int pp;

   // loop through pp from 0 to 10.
   for (pp=0; pp<=10; pp++)
   {

     //if pp is even, then say that
     if (pp%2 == 0)
     {
       fprintf(stdout,"The number %d is an even number\n",pp);
     }
     //if pp is odd, then say that
     if (pp%2 == 1)
     {
       fprintf(stdout,"The number %d is an odd number\n",pp);
     }

   }
return 0;
}
```

The output is:

```
The number 0 is an even number
The number 1 is an odd number
The number 2 is an even number
The number 3 is an odd number
The number 4 is an even number
The number 5 is an odd number
The number 6 is an even number
The number 7 is an odd number
The number 8 is an even number
The number 9 is an odd number
The number 10 is an even number
```

Ok, let's do some math with the next tutorial. There are a couple of new formatting things to observe in this code. First of all, in the print statement, notice the place holder %3d This means to print an integer with a field size of 3, meaning that 3 spaces will be preserved in the output text. This can help keep the screen output tidy. Play around with this in your code; try changing this value. Also, notice that the fprintf statement is carried over several lines. I did this to allow everything to fit nicely on this page, but in actuality I would have probably made my terminal window a little longer. In either case, this example shows how you can write a print statement over several rows in your source code. Sometimes this is useful.

code17.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int main(void)
{

//This code determines the x,y,z coordinates
//along a circle of radius=1.0 along
//the 45 degrees latitude (i.e., theta =pi/4).
// The code will print coordinates over 20
// points along the circle.
// We'll use both x,y,z coordinates and
// spherical coordinates theta and phi.

double x,y,z,phi;
double delta_phi, phi_in_degrees;
int ipoint;

// the radius of the sphere
double radius = 1.0;
// the theta coordinate is pi over 2
double theta = M_PI/4.0;
// The number of points along the circle
// that we'll inspect.
int number_of_points = 20;

// We will inspect the circle at 20 points
// along the circle.  We must determine the
// angular increment along that circle.  We'll
// name this angular increment delta_phi.
// The circumference of a circle is 2pi, and
// we'll divide that by 20 points.

    delta_phi = 2.0 * M_PI/(1.0*number_of_points);
```

```
// We will loop through each increment, starting
// at phi=0.0, and increment by delta_phi each time.
// We must first set the value of phi to zero, which
// is the starting point.

    phi = 0.0;
    for (ipoint = 1; ipoint <= number_of_points; ipoint++)
    {
       // Using standard trig, calculate the
       // x,y,z coordinates from theta and phi.

       x = radius * sin(theta)*cos(phi);
       y = radius * sin(theta)*sin(phi);
       z = radius * cos(theta);

       // To make our output look nicer,
       // lets have a variable that stores
       // phi in degrees, as opposed to radians.

       phi_in_degrees = phi * 180.0/M_PI;

       // Lets print what we have.  If we want,
       // we can carry the print statement over
       // several rows.

       fprintf(stdout,"point: %3d degrees: %f"
                  " x: %f y: %f z: %f\n",ipoint,phi_in_degrees,
                  x,y,z);


       // advance to the next value of phi
       phi = phi+delta_phi;
    }

return 0;
}
```

The output is:

```
point:   1 degrees: 0.000000 x: 0.707107 y: 0.000000 z: 0.707107
point:   2 degrees: 18.000000 x: 0.672499 y: 0.218508 z: 0.707107
point:   3 degrees: 36.000000 x: 0.572061 y: 0.415627 z: 0.707107
point:   4 degrees: 54.000000 x: 0.415627 y: 0.572061 z: 0.707107
point:   5 degrees: 72.000000 x: 0.218508 y: 0.672499 z: 0.707107
point:   6 degrees: 90.000000 x: 0.000000 y: 0.707107 z: 0.707107
point:   7 degrees: 108.000000 x: -0.218508 y: 0.672499 z: 0.707107
point:   8 degrees: 126.000000 x: -0.415627 y: 0.572061 z: 0.707107
point:   9 degrees: 144.000000 x: -0.572061 y: 0.415627 z: 0.707107
point:  10 degrees: 162.000000 x: -0.672499 y: 0.218508 z: 0.707107
point:  11 degrees: 180.000000 x: -0.707107 y: 0.000000 z: 0.707107
point:  12 degrees: 198.000000 x: -0.672499 y: -0.218508 z: 0.707107
point:  13 degrees: 216.000000 x: -0.572061 y: -0.415627 z: 0.707107
point:  14 degrees: 234.000000 x: -0.415627 y: -0.572061 z: 0.707107
point:  15 degrees: 252.000000 x: -0.218508 y: -0.672499 z: 0.707107
point:  16 degrees: 270.000000 x: -0.000000 y: -0.707107 z: 0.707107
```

```
point:  17 degrees: 288.000000 x: 0.218508 y: -0.672499 z: 0.707107
point:  18 degrees: 306.000000 x: 0.415627 y: -0.572061 z: 0.707107
point:  19 degrees: 324.000000 x: 0.572061 y: -0.415627 z: 0.707107
point:  20 degrees: 342.000000 x: 0.672499 y: -0.218508 z: 0.707107
```