Using AND/OR

In C, the logic condition "AND" is represented by &&, and the logic condition "OR" is represented by ||. "|" is the symbol usually on the key directly above the enter key on the keyboard. ANDs and ORs are best learned by example. Let's start a new code named *code05.c*. This code is a sloppy way to compare the ages of three rock units. In reality, you would never write such a silly code for research, but it's a nice tutorial in how to use AND (i.e., "&&") in an IF statement.

Code05.c

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

int main(void)
{

//declare the variable we will use

double Age_of_Chinle_Formation;
double Age_of_Dakota_Formation;
double Age_of_Sundance_Formation;

//we give the age of each rock formation in millions of years

  Age_of_Chinle_Formation = 220.0;
  Age_of_Dakota_Formation = 100.0;
  Age_of_Sundance_Formation = 150.0;

// Lets find out which formation is the oldest.
// There are probably fancier ways to do this, but we'll do it
// by brute force.


// First, lets check Sundance
  // If Sundance is older than Dakota AND
  // Sundance is older than Chinle, then it is the oldest.

  if ( (Age_of_Sundance_Formation > Age_of_Dakota_Formation) &&
     (Age_of_Sundance_Formation > Age_of_Chinle_Formation) )
  {
      fprintf(stdout,"The Sundance Formation is the Oldest\n");
  }
  else
  {
      fprintf(stdout,"Well, Sundance is not the oldest\n");
  }

// Now, lets check Dakota
  // If Dakota is older than Chinle AND
  // Dakota is older than Sundance, then it is the oldest.
```

```
    if ( (Age_of_Dakota_Formation > Age_of_Chinle_Formation) &&
         (Age_of_Dakota_Formation > Age_of_Sundance_Formation) )
    {
          fprintf(stdout,"The Dakota Formation is the Oldest\n");
    }
    else
    {
          fprintf(stdout,"Well, Dakota is not the oldest\n");
    }

  // Now, lets check Chinle
    // If Chinle is older than Dakota AND
    // Chinle is older than Sundance, then it is the oldest.

    if ( (Age_of_Chinle_Formation > Age_of_Dakota_Formation) &&
         (Age_of_Chinle_Formation > Age_of_Sundance_Formation) )
    {
          fprintf(stdout,"The Chinle Formation is the Oldest\n");
    }
    else
    {
          fprintf(stdout,"Well, Chinle is not the oldest\n");
    }

  return 0;
  }
```

Your output should be:

```
Well, Sundance is not the oldest
Well, Dakota is not the oldest
The Chinle Formation is the Oldest
```

Try changing the dates of the formations, and you’ll find that the oldest is always selected.

The following code works with both AND and OR

Code06.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(void)
{

// Declare 4 integers
int ii;
int jj;
int kk;
int pp;

// Assign values to the integers.
// If we want jj and kk to have the same initial value,
// we can keep them both on the same line.

  ii = - 1;
  jj = kk = 2;
  pp = 3;

// Lets check whether any of these are negative integers.
// We'll use the OR conditional, which is represented as ||.

  if ( (ii < 0) || (jj<0) || (kk<0) || (pp<0) )
  {
     fprintf(stdout,"At least one of these integers is negative.\n");
  }

// Lets see if ii is equal to any other integer values.
// The conditional for "equal to" is ==

  if ( (ii == jj) || (ii==kk) || (ii==pp) )
  {
      fprintf(stdout,"Integer ii is equal to another integer\n");
  }
  else
  {
      fprintf(stdout,"Integer ii is unique\n");
  }

// Lets do the same thing for jj.
// Lets see if jj is equal to any other integer values.
// The conditional for "equal to" is ==

  if ( (jj == ii) || (jj==kk) || (jj==pp) )
  {
      fprintf(stdout,"Integer jj is equal to another integer\n");
  }
  else
  {
      fprintf(stdout,"Integer jj is unique\n");
  }
return 0;
}
```

The code output should be:

```
Integer ii is unique
Integer jj is equal to another integer
```

Lets do one more.

Code07.c

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

int main(void)
{

// Declare an integer named number.
int number;

  // Make a loop which varies this integer
  // from 0 through 20, in multiples of 2.

  for (number = 0; number<=20; number=number+2)
  {
     // print out the number
     fprintf(stdout,"\nNumber is: %d\n",number);

     // Let check whether the number fits into one of the
     // following categories:

     if (number<5)
     {
       fprintf(stdout,"Number is less than 5\n");
     }
     else if ( (number >= 5) && (number <= 10) )
     {
       fprintf(stdout,"Number is between 5 and 10\n");
     }
     else
     {
       fprintf(stdout,"Number must be greater than 10\n");
     }

     // Lets see if the number is either less than 5 or greater
     // than 10.

     if ( (number < 5) || (number > 10) )
     {
        fprintf(stdout,"This number is either less than 5 or greater than 10\n");
     }
  }

return 0;
}
```

You should get the following output:

```
Number is: 0
Number is less than 5
This number is either less than 5 or greater than 10

Number is: 2
Number is less than 5
This number is either less than 5 or greater than 10

Number is: 4
Number is less than 5
This number is either less than 5 or greater than 10

Number is: 6
Number is between 5 and 10

Number is: 8
Number is between 5 and 10

Number is: 10
Number is between 5 and 10

Number is: 12
Number must be greater than 10
This number is either less than 5 or greater than 10

Number is: 14
Number must be greater than 10
This number is either less than 5 or greater than 10

Number is: 16
Number must be greater than 10
This number is either less than 5 or greater than 10

Number is: 18
Number must be greater than 10
This number is either less than 5 or greater than 10

Number is: 20
Number must be greater than 10
This number is either less than 5 or greater than 10
```

You can get quite creative from here. Remember, the best way to learn from these tutorials is to modify them to better test how these things work. If you're curious about something, try it out. Furthermore, you will also want to start constructing your own codes (aside from these tutorials) to gain more experience.

I should add one more comment about comparing real numbers in conditionals. For example, look at the following segment of code:

```
double A = 20.0

if (A  == 20.0 )
{
     fprintf(stdout,"A is 20.0 \n");
}
```

Because A=20.0, you would expect that the conditional would be satisfied and the print statement would be executed. In reality, this would only randomly work! Why? Remember that all real numbers have some uncertainty attached to them in the last significant digit. Therefore, even though we declare A as being equal to 20.0, it is quite possible that actually A=20.0000000000003. There is usually a random number associated with the last significant digit of a real number. In this case, A does not equal 20, so the condition would not be satisfied. **It is important to remember this!!! Real numbers almost always have some random digit in the least significant digit.** This is often called numerical noise or numerical error. An integer on the other hand is always equal to the proper integer because there is no random digit involved.

Here's how I would rewrite the code above. I would remove the == condition and replace it with checking whether A is within a tiny range around the value 20.0. Here's what I would do:

Choose a small number (I call it eps, meaning epsilon) that is smaller than the significant digits that we care about.

```
double eps=1e-8;    //eps is a very small number.
double A = 20.0;

// instead of asking whether A ==20, we provide a small range
if ( (A > (20.0-eps)) && (A<(20.0+eps))
{
     fprintf(stdout,"A is 20.0 \n");
}
```

We replaced "if(A == 20.0)" with "if ((A > (20.0-eps)) && (A<(20.0+eps))"

Another example that could cause a similar problem:

```
double A = 20.0;

if (A  <= 20.0 )
{
      fprintf(stdout,"A less than or equal to 20.0 \n");
}
```

If the computer actually registers A as A=20.0000000000003, then this condition would not be satisfied.

How would I would fix this. The concept is similar to before.

First one:

```
double eps=1e-8;    //eps is a very small number.
double A = 20.0;

if (A  <= (20.0+eps) )
{
      fprintf(stdout,"A less than or equal to 20.0 \n");
}
```

Where I replaced "if (A<=20.0)" with "if (A <= (20.0+eps))". This way, if A (which is supposed to be 20.0) has a spurious number in its last significant digit, the condition should be satisfied. Of course, one has to use careful judgment in choosing a value of eps (or whatever variable name you give it) appropriate for the problem. eps should be bigger than the random noise but small enough not to affect your science that you want to program!