Conditional Expressions

Boolean Expressions: An expression that evaluates to either TRUE or FALSE.

The most common types of boolean expressions are those that use relational operators. The general syntax of a conditional statement (which is a type of boolean expression) is this:

<expression> <relational operator> <expression>
(Note: The two expressions MUST match in type!!!)

Here are the 6 relational operators we will use:

1) Equal to (==)
2) Not equal to (!=)
3) Greater than (>)
4) Greater than or equal to (>=)
5) Less than (<)
6) Less than or equal to (<=)

We will typically only compare numerical expressions. You may compare characters with these operators as well, since they are stored internally as integers. However, you can NOT accurately compare strings with these operators.

We compare characters in alphabetical order, so for example, ‘a’ < ‘b’.

Here are some examples of boolean expressions. See if you can figure out if they are true or not:

Note: Assume that we have these beginning declarations:

```
char ch;
int x,y;

ch = ‘j’;
x = 6*2;
y = 3*x - 10% x;
```

Expression   Value

--------------   --------

1) `x + y > 40`

2) `ch == ‘k’`

3) `(x - y) != (7*(3 - x % 7))`

Boolean Operators are operators that take boolean expressions as operands. The following are boolean operators available to us in C: AND(&&), OR(||) and NOT(!).

The first two are take two boolean expressions as operands, and the last takes only a single boolean expression as an operand.

Let b1 and b2 be boolean expressions. Here is the syntax for complex boolean expressions put together with the boolean operators mentioned above:

(b1) &&(b2)
(b1) || (b2)
!(b1)

In particular these complex boolean expressions evaluate to boolean values as well. Here are truth tables to show how to evaluate these:

&&	T	F
T	T	F
F	F	F

<table>
<tr><th>||</th><th>T</th><th>F</th></tr>
<tr><td>T</td><td>T</td><td>T</td></tr>
<tr><td>F</td><td>T</td><td>F</td></tr>
</table>

In English, for b1 && b2 to be true, both expressions must be true.

For b1 || b2 to be true, at least one of b1 and b2 has to be true.

! simply negates the value of the original boolean expression.

If-else Construct

```
if (<boolean expression>)
    stmt1;
else
    stmt2;
stmtA;
stmtB;
```

The way this is executed is as follows:

1) **Check if the boolean expression is true.**
2) **If so, go ahead and execute stmt1, then skip to the end of the if.**
3) **Otherwise, go ahead and execute stmt2, and go to the end of the if.**
4) **Continue on, executing stmtA, stmtB, etc.**

A natural question to ask at this point is, "What if I want to execute MORE than one statement inside of either the if clause or the else clause?"

In order to deal with that case, C allows us to use a block of statements. A block of statements are statements surrounded by curly braces({}) on both sides. The compiler treats this entire group of statements as a single statement, syntactically.

So, using the information above, here is an adjusted & generalized template for an if-else statement:

```
if (<boolean expression>) {
     stmt11;
     stmt12;
     .
     .
     stmt1n;
}
else {
     stmt21;
     stmt22;
     .
     stmt2m;
}
stmtA;
stmtB;
```

The way this is executed is as follows:

1) Check if the boolean expression is true.
2) If so, go ahead and execute stmt11 through stmt1n in order.
3) Otherwise, go ahead and execute stmt21 through stmt2m in order.
4) Continue on, executing stmtA, stmtB, etc.

Use of else if

```
if (<boolean expression1>)
    <stmts1>
else if (<boolean expression2>)
    <stmts2>
else if (<boolean expression3>)
    <stmts3>
else
    <stmtsn>
stmtA
```

1) Check if boolean expression1 is true.
2) If so, go ahead and execute the first block of statements, <stmts1>
3) If not, go check if boolean expression2 is true.
4) If so, go ahead and execute the second block of statements, <stmts2>
5) Continue in this fashion if no until one of the boolean expressions is true.
6) Now, skip over the rest of the blocks, and continue executing statements with stmtA.

Note: In none of these clauses is an else clause necessary. There may be cases you do not want to execute any statements based on a particular decision.

Most Simple Example: Sales Tax

The most simple use of an if statement is where we execute some code if a condition is true and don't execute it if it's not true. Consider how sales tax is calculated. Some items are taxes while others are not. Thus, we start with the initial price of the item. Then, if it's taxed, we add to the price the tax amount.

Here is the sales tax program:

```
#include <stdio.h>

#define TAXRATE .065

int main() {

    double price;
    int isTaxed;

    // Get the item price.
    printf("What is the price of the item you are buying?\n");
    scanf("%lf", &price);

    // And whether it's taxed or not.
    printf("Is it a taxed item? (1=yes, 0=no)\n");
    scanf("%d", &isTaxed);

    // We could also do isTaxed == 1, but 1 itself counts as true.
    if (isTaxed)
        price = price + price*TAXRATE;

    // Print out the final total.
    printf("Your final total is $%.2lf.\n", price);

    return 0;
}
```

Extending the Quadratic Solver Program

Our previous program solving the quadratic equation only worked if the equation had real roots. Now, we would like for our program to behave differently if the values for a, b and c produce complex roots. Perhaps we can just print out an error message stating that the roots of the equation are complex. The if statement allows us to have the program act in different ways, depending on the user input. In this situation, we just want to check the value of the discriminant to see if it's non-negative or not. Note the use of the block of code when it was necessary, for the else branch. Here is the program:

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

int main(void) {

    int a, b, c;
    double disc, root1, root2;

    // Read in the coefficients for the quadratic.
    printf("Enter in a, b and c from your quadratic equation.\n");
    scanf("%d%d%d", &a, &b, &c);

    // Calculate the discriminant.
    disc = pow(b,2) - 4*a*c;

    // Take care of the complex number case.
    if (disc < 0)
        printf("Sorry your quadratic has complex roots.\n");

    // Go ahead and calculate both roots.
    else {
        root1 = (-b + sqrt(b*b-4*a*c))/(2*a);
        root2 = (-b - sqrt(disc))/(2*a);
        printf("The roots are %.2lf and %.2lf.\n", root1, root2);
    }

    return 0;
}
```

Determining the slope of a line

// Arup Guha
// 9/8/03
// Program Description: This program asks the user for the
// constants A, B, and C in the equation Ax+By = C for a line.
// Based on this information, the algorithm will output if the
// slope is positive, negative, zero, or undefined (as is the case
// with a vertical line.)

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

     // Obtain values for a, b and c from user.
     double a, b, c;
     printf(“Enter a, b, and c.\n”)
     scanf("%lf%lf%lf",&a,&b,&c)

     // Take care of the invalid line case.
     if (( a == 0 ) && ( b == 0 ))
        printf(“Invalid equation.\n”);
     else {

          // Handle each case (vertical, horizontal, positive
          // slope and negative slope) separately.
          if ( b == 0 )
             printf(“Vertical line.\n”);
          else if ( a == 0 )
             printf(“Slope = 0\n”);
          else if ( -a/b > 0 )
             printf(“Slope is positive.”);
          else
             printf(“Slope is negative.\n”);
     }
}
```

Couple other notes about boolean operators and the if statement

Nested if statements

It is possible to have an if statement inside of an if statement. One of the statements inside of an if clause or an else clause can be another if statement.

Matching else problem

If you do not specify using curly braces{}, an else clause always matches the nearest possible if clause.

Here is a small example:

```
if (month == 4)
  if (day > 15)
    printf("Your taxes are late.\n");
  else
    printf("File your taxes by 4/15.\n");
```

If the variable month was 7 at the start of this code, then nothing would get printed out.

If you want the else to match the first if, here is how you would have to do it:

```
if (wage >= 5.25) {
  if (hours >= 30)
    printf("You make over $150 a week.\n");
}
else
  printf("You get below minimum wage.\n");
```

Now, the computer treats everything inside of {}s as a single statement inside of the outside if clause.

Order of Operations

The order of precendence of && and || is lower than any arithmetic operator. If you ever have any doubt how the computer will interpret your expressions, use parentheses to explicitly dictate how your expression gets evaluated. (Incidentally, the precedence of && is higher than ||.)

```
(x > 7) || (y<6)  && (z = 3)
```

is interpreted as:

```
(x > 7) || ( (y<6)  && (z = 3) )
```

I suggest you write it with ()s.