Lesson #4

Logical Operators and Selection Statements

Control Structures

- Control structures combine individual instructions into a single logical unit with one entry point at the top and one exit point at the bottom.
- 3 kinds of control structures:
  - Sequence (default control structure)
  - Selection (branches)
  - Repetition (loops)

Conditions

- A condition is an expression that is either true or false.
- Ex: temperature=28;
  - temperature < 0 will be false
  - temperature >20 will be true
- Comparison (relational) operators are:
  - **< > <= >= == !=**

Logical Operators

- A logical expression contains one or more comparison and/or logical operators.
- The logical operators are:
  - **&&**: the **and** operator, true only when all operands are true.
  - **||**: the **or** operator, false only when all operands are false.
  - **!**: the **not** operator, false when the operand is true, true when the operand is false.

Updated Operator Precedence Rule

2.1 Function calls
2.2 Unary operators (-, +, !, (int), (double))
2.3 *, /, %
2.4 +, -
2.5 <, <=, >=, >
2.6 ==, !=
2.7 &&
2.8 ||
2.9 = (assignment operator)

Fast Evaluation of Logical Expressions

```
int x=3, y=4, z=10, w;

w = x == 3 || y < 10;
```

is w true or false?

```
w = x%3-z<x&&z>=x/y || x<y;
```

- The trick is to divide the expression in sub-expressions between the || and evaluate the easiest first. As soon as you find a *true* sub-expression, the whole expression is true.

Fast Evaluation of Logical Expressions

- When there is no || operator. Divide the expression in parts between the && operators. If one part is false, the expression is false.
- int a=1, b=2, c=3;
- Is this expression true or false?

```
a < 5 - 3 && b == 5 - c && b > 4
```

Building Logical Expressions

- Are x and y both greater than 10?
  - x > 10 && y > 10
- Is either x equal to 1 or 3?
  - x == 1 || x == 3
- Is x between y and z?
  - x >= y && x <= z (never y <= x <= z)

    for example if a=7; 3 <= a <= 5 is true!!
- Is x an even number?
  - x % 2 == 0

Comparing Characters

- '9' >= '0' ?
- 'a' < 'e' ?
- 'B' <= 'A' ?
- 'Z' == 'z' ?

- Is a letter lowercase?
  - letter >= 'a' && letter <= 'z'

- Does the variable *ch* contain a letter?
  - (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')

Logical Assignment

- Logical values can be assigned to variables. Logical variables are **int** in C.

```
int age, senior;
scanf ("%d", &age);
senior = age >= 65;
```

- *senior* will contain 1 if true and 0 if false.

Integer and Logical Values

- In C, logical values (true or false) are represented by integer constant and variables.
- False is always 0.
- True is all the non-zero values.
- 1 always means *true* of course, but 2, 4.5, and –10000 are also values interpreted as *true*.

Condition Complements

- The opposite (or complement) of (x == 0) is !(x==0) or (x != 0).
- The opposite of (x > 3) is !(x > 3) or (x <=3).
- The opposite of (a == 10 && b > 5) is !(a == 10 && b > 5) or (a != 10 || b <=5)

De Morgan's Theorem:

```
The complement of exp1&&exp2 is
  comp_exp1||comp_exp2.

The complement of exp1||exp2 is
  comp_exp1&&comp_exp2.
```

The *if* Statement

Syntax of a simple if statement:

```
if (condition)
     statement if condition is true;
else
     statement if condition is false;
```

Note: Never put a ; after the condition.

A Simple *if* Statement

```
int temp;

printf ("What is the temperature?");
scanf ("%d", &temp);

if (temp >= 20)
     printf ("The temperature is warm.\n");
else
     printf ("The temperature is cool.\n");
```

if Statement with Only One Alternative

```
int temp;

printf ("What is the temperature?");
scanf ("%d", &temp);

if (temp >= 100)
    printf ("WARNING! Boiling water\n");
```

if with Compound Statements

- ***if*** statements expect only one statement per branch (true/false). To have more, we use compound statements (enclosed in { }).

```
int temp;

printf ("What is the temperature?");
scanf ("%d", &temp);
if (temp >= 100)
{
   printf ("WARNING! Boiling water.\n");
   printf ("Turn off the heat!\n");
}
```

Nested *if* Statements

- Nested ifs are ifs inside ifs. It is good practice to only expand the false branch. The true branch is always terminal. If you need to expand the true branch, reverse the condition and expand the false branch.

```
if (temp >= 100)
    printf ("Boiling water!\n");
else
   if (temp <=0)
       printf ("Freezing water!\n");
```

Nested *if* Statements

- Order is very important with nested ifs:

```
if (noise <= 50)
   printf ("Quiet\n");
else
   if (noise <= 70)
        printf ("Intrusive\n");
   else
        if (noise <=90)
             printf ("Deafening\n");
        else
             printf ("Dangerous\n");
```

The switch **Statement**

- The switch statement is a useful alternative for multiple branches (not just true and false).
- It works not with conditions but with control values.
- The control values must be int or char only, never double.
- The control values must be discrete, never ranges.

The switch Statement

- Syntax:

```
switch (control value)
{
   case value1:
   case value2:
                  statement(s) if value1 or value2
                  matches the control value;
                 break;
   case value3:
                  statement(s) if value3
                 matches the control value;
                 break;
 /* other possible values here */
   default:
                statement(s) if no case value
                matches the control value;
}
```

A switch Statement Example

```
switch (color)
{
    case 'R':
    case 'r':
                printf ("STOP!");
                break;
    case 'Y':
    case 'y':
                printf ("CAUTION!");
                break;
    case 'g':
    case 'G':
                printf ("GO!");
                break;
    default:
                printf ("Invalid color");
}
```

End of lesson