C3 - Control Structures in C

Decision Making

Relational Operators

< less than

<= less than or equal to

> greater than

>= greater than or equal to

== compare equal

!= not equal

In C:

a TRUE condition is assigned a value of NON-ZERO.

A FALSE condition is assigned a value of ZERO.

Examples of conditions:

(a > b)

(x == y + z)

(x) where if x = 0, false
if x = 2, true/non-zero

Logical Operators:

&& **and** keyboard location, on the “7” key
|| **or** keyboard location, on key with “\” usually in upper right corner
! **not** keyboard location, on the “1” key

Results of logical operators (Truth Table):

A	B	A && B	A || B	!A	!B
F	F	F	F	T	T
F	T	F	T	T	F
T	F	F	T	F	T
T	T	T	T	F	F

Practice with Conditions:

```
float  a = 2.2, b = -1.2;
int i = 5, done = 1;
```

(! (a == 2 * b)) Will it be true or false?

Practice with Conditions:

```
float  a = 2.2, b = -1.2;
int i = 5, done = 1;
```

(! (a == 2 * b)) Will it be true or false?

!(2.2 == 2 * -1.2)

!(2.2 == -2.4)

!(false)

true

Practice with Conditions:

```
float  a = 2.2, b = -1.2;
int i = 5, done = 1;
```

((a < 10.0) && (b > 5.0)) Will it be true or false?

Practice with Conditions:

```
float  a = 2.2, b = -1.2;
int i = 5, done = 1;
```

((a < 10.0) && (b > 5.0)) Will it be true or false?

```
((2.2 < 10.0) && (-1.2 > 5.0))
     true      &&   false
               false
```

Practice with Conditions:

```
float  a = 2.2, b = -1.2;
int i = 5, done = 1;
```

((abs(i) > 2) || done) Will it be true or false?

Practice with Conditions:

```
float  a = 2.2, b = -1.2;
int i = 5, done = 1;
```

((abs(i) > 2) || done) Will it be true or false?

```
((abs(5) > 2) || done)
        5 > 2    ||  1
         true    ||  true
                true
```

Precedence:

1) ( ) innermost first

2) ++ -- post-increment. left to right
   ++ -- pre-increment. right to left

3) **!** + - left to right. (+ positive, - negative)
   += -= *= /= %= right to left.

4) * / % left to right
5) + - left to right (add and subtract)

6) < <= > >= left to right
   == != left to right

7) **&&** left to right
8) **| |** left to right
9) = right to left (assignment)

The simple *if* structure

The simple IF statement:

```
if(condition)
{
    statement 1;
}
```

if condition is True, do statement 1.
if condition is False, skip statement 1.

Examples:

```
if (a < b)
{
    sum = sum + a;        /* simple if */
}


if (time > 1.5) {         /* another simple if */
    scanf("%d", &distance);
}
```

Use a Compound Statement or Block

a set of statements enclosed in braces { }

```
if(condition)
{
  statement 1;
  statement 2;
  statement 3;
}
```

```
if(condition) {

   statement 1;
   statement 2;
   statement 3;
}
```

Two different styles of indentation.
My version of VIM defaults to the style on the <u>left</u>.
The K&R book uses the style on the **<u>right</u>**.
Use indentation with a consistent style.

In both styles, the contents inside the braces are indented!!!

Example:

```
if(x != 0.0)                    /* compound if */
{
    sum = sum +x;
    count = count + 1;
    printf("/nEnter another number:  ");
    scanf("%f", &x);
}
```

Nested IF:

```
if (gpa >= 3.0)
{
    printf("Honor Roll \n");
    if (gpa > 3.5)
    {
        printf("President's List \n");
    }
}
```

Nested IF:

```
if (gpa >= 3.0)
{
    printf("Honor Roll \n");
    if (gpa > 3.5)
    {
        printf("President's List \n");
    }
    if(gpa < 2.0)
    {
        printf("Flirting with trouble \n");
    }
}
```

"Flirting with trouble" can never print from this code.

if-else statement:

```
if (course_code != 2)
{
   printf("No course listed \n");          /* true section */
   printf("No room listed \n");
}
else
{
   printf("Computer Science \n");          /* false section */
   printf("Ruthann Biel \n");
}
```

Two examples – both result in same output.

Example 1:

```
if (marital_status == 's')
{
    if (gender == 'M')
    {
        if (age >= 18)
        {
            if (age <= 26)
            {
                printf("All criteria are met.\n");
            }
        }
    }
}
```

Two examples – both result in same output.

Example 2:

```
if (marital_status == 'S'
   && gender == 'M'
   && age >= 18
   && age <= 26)
{
        printf("All criteria are met.\n");
}
```

```
if (road_status == 'S')  /* for slick road */
{
   if (temp > 32)
   {
      printf("Wet Roads Ahead \n");
      printf("Stopping Time Doubled \n");
   }
   else
   {
      printf("Icy Roads Ahead \n");
      printf("Stopping Time Quadrupled \n");
   }
}
else
{
   printf("Drive Carefully! \n");
}
```

The *else if* structure

if-else-if structure:

```
if (weight <= 50.0)
{
   category = 1;
}
else if (weight <= 125.0)
{
   category = 2;
}
else if (weight <= 200.0)
{
   category = 3;
}
else
{
   category = 4;
}
```

if-else-if structure generic form:

```
if (condition 1)
{
   statements 1;
}
else if (condition 2)
{
   statements 2;
}
          /* repeat else-if as many times as needed within reason */
else      /* the else is optional but often used for catching errors */
{
   last set of statements;
}
```

The Conditional Operator

Alternative for the if-else

Conditional Operator = **?:**

A ternary operator can be used instead of ***if-else***

(Ternary means it has 3 parts with 2 operators.)

Conditional Operator - ?:

```
if(count <= 100)                    /* if-else way /
{
    count +=5;
}
else
{
    sum = count + hiho;
}
                                    /* Conditional Operator way */
```

count <= 100 ? count +=5 : sum = count + hiho;

condition to be evaluated

section done on **True** condition

section done on **False** condition

```
#include <stdio.h>            /* voltage.c */
#include <stdlib.h>
int main (void)
{
   float led_voltage;        /* Voltage across LED in volts. */
   float resistor_voltage;   /* Voltage across resistor in volts. */
   float source_voltage;     /* Voltage of the source in volts.  */
   float circuit_current;    /* Current in the LED in amperes  */
   float resistor_value;     /* Value of resistor in ohms. */

   printf("\n\nEnter the source voltage in volts =>  ");
   scanf("%f", &source_voltage);
   printf("\n\nEnter value of resistor in ohms =>  ");
   scanf("%f", &resistor_value);

   led_voltage  = (source_voltage < 2.3) ? source_voltage: 2.3;
   resistor_voltage = source_voltage – led_voltage;
   circuit_current = resistor_voltage / resistor_value;
   printf ("Total circuit current is %f amperes. \n", circuit_current);
   system("pause");
   return EXIT_SUCCESS;
}
```

Example of the two ways: ?: *and* if-else

(1)

```
led_voltage  = (source_voltage < 2.3) ? source_voltage: 2.3;
```

(2)

```
if (source_voltage < 2.3)
    led_voltage = source_voltage;
else
    led_voltage = 2.3;
```

Examples of RUNs of the previous program:

```
Enter the source voltage in volts =>  2
Enter value of resistor in ohms =>  5
Total circuit current is 0.000000 amperes.
```

```
Enter the source voltage in volts =>  5
Enter value of resistor in ohms =>  5
Total circuit current is 0.540000 amperes.
```

The *switch* structure

The ***switch*** and the ***if-else-if*** produce similar results but they operate in a different way internally.

The ***if-else-if*** tests each condition until a True is encountered. It does that section, then jumps out of the structure.

The ***switch*** creates an internal table, determines where it should jump to, and then does it. It does not repeatedly test for true.

```
int code;
switch (code)
{
   case 10:
      printf (“Too hot – turn equipment off \n”);
      break;
   case 11:
      printf(“Caution – recheck in 5 minutes. \n”);
      break;
   case 13:
       printf(“Turn on the circulating fan. \n”);
       break;
   default:
       printf(“Normal mode of operation. \n”);
       break;
}
```

```
int code;
switch (code)
{
   case 10:
      printf (“Too hot – turn equipment off \n”);
      break;
   case 11:
   case 12:
      printf(“Caution – recheck in 5 minutes. \n”);
      break;
   case 13: case 14:
       printf(“Turn on the circulating fan. \n”);
       break;
   default:
       printf(“Normal mode of operation. \n”);
       break;
}
/* Note two cases for one print statement, two styles */
```

/* General form of the Switch */

```
switch (controlling expression or variable)
{
   case label_1:
      statements;
      break;
   case label_2:
       statements;
       break;
   …
   default:
       statements;
       break;
}
/* default – optional, recommended*/
/* break – forces flow-of-control out of the switch statement */
```

C-3 Control Structures in C

The End