Boolean Data Lesson #2 Outline

1. Relational Operations #1
2. Relational Operations #2
3. Relational Expressions Example #1
4. Relational Expressions Example #2
5. Structure of Boolean Expressions
6. Boolean Expressions with Parentheses
7. Precedence Order of Boolean Operations
8. Boolean Precedence Order Example #1
9. Boolean Precedence Order Example #2
10. Boolean Precedence Order Example
11. Relational Expressions Example #1
12. Relational Expressions Example #2
13. Relational Expressions Example #3
14. Relational Expressions Example #4
15. Relational Expressions Example #5
16. Relational Expressions Example #6
17. Relational Expressions Example #7
18. Why Not Use `a < b < c`? #1
19. Why Not Use `a < b < c`? #2
20. Short Circuiting
21. Short Circuit Example #1
22. Short Circuit Example #2
23. Short Circuit Example #3

Relational Operations #1

A ***relational*** operation is a binary operation that compares two numeric operands and produces a Boolean result.

For example:

```
CS1313_lab_section == 14
cm_per_km != 100
age < 21
number_of_students <= number_of_chairs
credit_hours > 30
electoral_votes >= 270
```

Relational Operations #2

Operation	Operator	Usage	Result
Equal to	`==`	`x == y`	`1` if the value of `x` is exactly the same as the value of `y`; otherwise `0`
Not equal to	`!=`	`x != y`	`1` if the value of `x` is different from the value of `y`; otherwise `0`
Less than	`<`	`x < y`	`1` if the value of `x` is less than the value of `y`; otherwise `0`
Less than or equal to	`<=`	`x <= y`	`1` if the value of `x` is less than or equal to the value of `y`; otherwise `0`
Greater than	`>`	`x > y`	`1` if the value of `x` is greater than the value of `y`; otherwise `0`
Greater than or equal to	`>=`	`x >= y`	`1` if the value of `x` is greater than or equal to the value of `y`; otherwise `0`

Relational Expressions Example #1

```
#include <stdio.h>
int main ()
{ /* main */
    int CS1313_size, METR2011_size;
    printf("How many students are in CS1313?\n");
    scanf("%d", &CS1313_size);
    printf("How many students are in METR2011?\n");
    scanf("%d", &METR2011_size);
    printf("%d == %d: %d\n", CS1313_size, METR2011_size,
        CS1313_size == METR2011_size);
    printf("%d != %d: %d\n", CS1313_size, METR2011_size,
        CS1313_size != METR2011_size);
    printf("%d <  %d: %d\n", CS1313_size, METR2011_size,
        CS1313_size <  METR2011_size);
    printf("%d <= %d: %d\n", CS1313_size, METR2011_size,
        CS1313_size <= METR2011_size);
    printf("%d >  %d: %d\n", CS1313_size, METR2011_size,
        CS1313_size >  METR2011_size);
    printf("%d >= %d: %d\n", CS1313_size, METR2011_size,
        CS1313_size >= METR2011_size);
} /* main */
```

Relational Expressions Example #2

```
% gcc -o relational relational.c
% relational
How many students are in CS1313?
198
How many students are in METR2011?
96
198 == 96: 0
198 != 96: 1
198 <  96: 0
198 <= 96: 0
198 >  96: 1
198 >= 96: 1
```

Structure of Boolean Expressions

A Boolean expression can be long and complicated.

For example:

```
a || (b || c && !d) && e && (f || g) && h
```

Terms and operators can be mixed together in almost limitless variety, but they must follow these rules:

a unary operator has a term immediately to its right, and

a binary operator has terms on both its left and its right.

Boolean Expressions with Parentheses

Parentheses can be placed around any unary or binary subexpression:

```
(a || b) || (c && (d && (!e)))
```

Putting a term in parentheses may change the value of the expression, because a term inside parentheses will be calculated first. For example:

```
a || b  && c
```

is evaluated as "b AND c, OR a," but

```
(a || b) && c
```

is evaluated as "a OR b, AND c."

Precedence Order of Boolean Operations

In the absence of parentheses to explicitly state the order of operations, the order of precedence is:

1. relational operations, left to right
2. !, left to right
3. &&, left to right
4. ||, left to right

After taking into account the above rules, the expression as a whole is evaluated left to right.

Rule of Thumb: If you can't remember the priority order of the operators, use lots of parentheses.

Boolean Precedence Order Example #1

```
!       0       ||      1

    1           ||      1
```

1

but

```
!       (0      ||      1)

!               1
```

0

Boolean Precedence Order Example #2

```
0     &&    1     ||    1     &&    1
      0           ||    1     &&    1
      0           ||          1
                  1
```

but

```
0     &&    (1    ||    1)    &&    1
0     &&          1           &&    1
            0                 &&    1
                  0
```

Boolean Precedence Order Example

```
% cat logic_expressions.c
#include <stdio.h>

int main ()
{ /* main */
    printf("! 0 || 1  = %d\n", ! 0 || 1);
    printf("!(0 || 1) = %d\n", !(0 || 1));
    printf("0 &&  1 || 1  && 1 = %d\n",
        0 &&  1 || 1  && 1);
    printf("0 && (1 || 1) && 1 = %d\n",
        0 && (1 || 1) && 1);
} /* main */
% gcc -o logic_expressions logic_expressions.c
% lgcexpr
! 0 || 1  = 1
!(0 || 1) = 0
0 &&  1 || 1  && 1 = 1
0 && (1 || 1) && 1 = 0
```

Relational Expressions Example #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int program_success_code = 0;
    int a, b, c;
    char b_equals_a, b_equals_c;
    char b_between_a_and_c, b_between_c_and_a;
    char b_outside_a_and_c;
    char a_lt_b_lt_c, c_lt_b_lt_a;
```

Relational Expressions Example #2

```
    printf("Enter three different integers:\n");
    scanf("%d %d %d", &a, &b, &c);
    printf("The integers you entered are:\n");
    printf("a = %d, b = %d, c = %d\n", a, b, c);
    b_equals_a = (b == a);
    b_equals_c = (b == c);
    b_between_a_and_c = ((a < b) && (b < c));
    b_between_c_and_a = ((c < b) && (b < a));
    b_outside_a_and_c =
        !(b_between_a_and_c || b_between_c_and_a);
    a_lt_b_lt_c = a < b < c;
    c_lt_b_lt_a = c < b < a;
    printf("b == a: %d\n", b_equals_a);
    printf("b == c: %d\n", b_equals_c);
    printf("a < b && b < c: %d\n", b_between_a_and_c);
    printf("c < b && b < a: %d\n", b_between_c_and_a);
    printf("a < b < c: %d\n", a_lt_b_lt_c);
    printf("c < b < a: %d\n", c_lt_b_lt_a);
    printf("b outside a and c: %d\n",
        b_outside_a_and_c);
    return program_success_code;
} /* main */
```

Relational Expressions Example #3

```
% gcc -o comparisons comparisons.c
% comparisons
Enter three different integers:
4 4 5
The integers you entered are:
a = 4, b = 4, c = 5
b == a: 1
b == c: 0
a < b && b < c: 0
c < b && b < a: 0
a < b < c: 1
c < b < a: 1
b outside a and c: 1
```

Relational Expressions Example #4

```
% comparisons
Enter three different integers:
4 5 5
The integers you entered are:
a = 4, b = 5, c = 5
b == a: 0
b == c: 1
a < b && b < c: 0
c < b && b < a: 0
a < b < c: 1
c < b < a: 1
b outside a and c: 1
```

Relational Expressions Example #5

```
% comparisons
Enter three different integers:
4 5 6
The integers you entered are:
a = 4, b = 5, c = 6
b == a: 0
b == c: 0
a < b && b < c: 1
c < b && b < a: 0
a < b < c: 1
c < b < a: 1
b outside a and c: 0
```

Relational Expressions Example #6

```
% comparisons
Enter three different integers:
6 5 4
The integers you entered are:
a = 6, b = 5, c = 4
b == a: 0
b == c: 0
a < b && b < c: 0
c < b && b < a: 1
a < b < c: 1
c < b < a: 1
b outside a and c: 0
```

Relational Expressions Example #7

```
% comparisons
Enter three different integers:
4 3 5
The integers you entered are:
a = 4, b = 3, c = 5
b == a: 0
b == c: 0
a < b && b < c: 0
c < b && b < a: 0
a < b < c: 1
c < b < a: 1
b outside a and c: 1
```

Why Not Use `a < b < c`? #1

```
b_between_a_and_c =
    ((a < b) && (b < c));
b_between_c_and_a =
    ((c < b) && (b < a));
b_outside_a_and_c =
    !(b_between_a_and_c ||
      b_between_c_and_a);
a_lt_b_lt_c = a < b < c;
c_lt_b_lt_a = c < b < a;
```

Expressions like

`a < b < c` and `c < b < a`

WON'T accomplish what they look like they should.

Why not?

Why Not Use `a < b < c`? #2

Consider the expression `a < b < c`, and suppose that `a` is `6`, `b` is `5` and `c` is `4`; that is, `6 < 5 < 4`, which we know in real life is **false**.

But let's evaluate the expression as written.

1. Using the precedence rules, we evaluate left to right, so first we evaluate the subexpression `a < b`, which is a relational expression, so its result must be true (`1`) or false (`0`) – in this case false (`0`).
2. We then plug that result into the rest of the expression, getting `0 < c`; that is, `0 < 4`, which is **true** – so the value for `a < b < c` is wrong!

Instead, we need to use this: `(a < b) && (b < c)`

Short Circuiting

When a C program evaluates a Boolean expression,
it may happen that, after evaluating some of the terms,
the result can no longer change, regardless of
what the remaining terms evaluate to.

In that case, the program will stop bothering to evaluate
the rest of the expression, because evaluating
the rest of the expression wouldn't make any difference,
but would **waste time**.

In such a case, we say that the Boolean expression will
short circuit: the rest of the expression won't be evaluated,
because evaluating it would waste time, given that
it won't change the result.

Short Circuit Example #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int maximum_short_height_in_cm = 170;
    const int program_success_code       =   0;
    int  my_height_in_cm = 160;
    char I_am_Henry = 1;
    char I_am_tall;
    char my_middle_initial = 'J';

    I_am_tall =
        (!I_am_Henry) &&
        (my_height_in_cm >
         maximum_short_height_in_cm);
    printf("I_am_Henry = %d\n", I_am_Henry);
    printf("my_height_in_cm = %d\n",
        my_height_in_cm);
    printf("I_am_tall = %d\n", I_am_tall);
    printf("my_middle_initial = %c\n",
        my_middle_initial);
    return program_success_code;
} /* main */
```

Short Circuit Example #2

```
% gcc -o short_circuit short_circuit.c
% short_circuit
I_am_Henry = 1
my_height_in_cm = 160
I_am_tall = 0
my_middle_initial = J
```

In the example above, the relational expression never gets evaluated, because the first operand in the AND operation && evaluates to `0`, and therefore the entire AND operation must evaluate to `0`.

Short Circuit Example #3

```
int  my_height_in_cm = 160;
char I_am_Henry = 1;
char I_am_tall;
…
I_am_tall =
     (!I_am_Henry) &&
     (my_height_in_cm <
      maximum_short_height_in_cm);
…
```

In the example above, the relational expression never gets evaluated, because the first operand in the AND operation && evaluates to `0`, and therefore the entire AND operation must evaluate to `0`.