Boolean Data Lesson #1 Outline

1. Boolean Data Outline
2. Data Types
3. C Boolean Data Type: `char` or `int`
4. C Built-In Boolean Data Type: `bool`
5. `bool` Data Type: Not Used in CS1313
6. Boolean Declaration
7. Boolean or Character?
8. Boolean or Character Example #1
9. Boolean or Character Example #2
10. Boolean, Character or Integer? #1
11. Boolean, Character or Integer? #2
12. Boolean Literal Constants
13. Using Boolean Literal Constants #1
14. Using Boolean Literal Constants #2
15. What is a Boolean Expression? #1
16. What is a Boolean Expression? #2
17. What is a Boolean Expression? #3
18. What is a Boolean Expression? #4
19. What is a Boolean Expression? #5
20. Boolean Expressions
21. Boolean Operations
22. C Boolean Expression Evaluation Values
23. Boolean Expression Example #1
24. Boolean Expression Example #2
25. Boolean Variables Example #1
26. Boolean Variables Example #2

Data Types

A **data type** is (surprise!) a type of data:

- Numeric
  - `int`: ***integer***
  - `float`: ***floating point*** (also known as ***real***)
- Non-numeric
  - `char`: ***character***

Note that this list of data types **ISN'T** exhaustive –
there are many more data types (and you can define your own).

```
#include <stdio.h>
int main ()
{ /* main */
    float standard_deviation, relative_humidity;
    int   count, number_of_silly_people;
    char  middle_initial, hometown[30];
} /* main */
```

C Boolean Data Type: `char` or `int`

The C data type typically used for storing Boolean values is `char`, although `int` will also work.

Like numeric data types, Booleans have particular ways of being stored in memory and particular ways of being operated on.

Conceptually, a Boolean value represents a single bit in memory.

But, the `char` and `int` data types aren't implemented this way – if for no other reason than that computers can't address a single bit, because the smallest collection of bits that they can address is a byte (8 bits) – or, in a few cases, a word.

C Built-In Boolean Data Type: `bool`

C also has a built-in data type for Booleans:

`bool`

The `bool` data type has possible values

`false`

and

`true`

However, some C compilers don't have available by default the `bool` data type nor the Boolean values `true` and `false`; you have to make them available using this directive:

```
#include <stdbool.h>
```

(after `#include <stdio.h>`).

`bool` Data Type: Not Used in CS1313

In CS1313, we **WON'T** use the `bool` data type, nor its values `true` and `false`.

Instead, we'll use `char` or `int`.

Similarly, we'll use `0` for false and `1` (or any nonzero integer value) for true.

Boolean Declaration

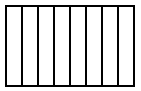
```
char CS1313_lectures_are_fascinating;
```

This declaration tells the compiler to grab a group of bytes, name them `CS1313_lectures_are_fascinating`, and think of them as storing a Boolean value (either **true** or **false**).

How many bytes?

Even though conceptually a Boolean represents a single bit, in practice `char` variables are usually implemented using 8 bits (1 byte):

`CS1313_lectures_are_fascinating :`



Boolean or Character?

Question: How does the C compiler know that a particular `char` declaration is a Boolean rather than a character?

Answer: It doesn't.

Whether a `char` (or an `int`) is treated by a program as a Boolean or as a character (respectively, an integer) **depends entirely on how you use it** in the program.

Boolean or Character 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 */
```

Boolean or Character Example #2

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

Whether a `char` (or an `int`) is treated by a program as a Boolean or a character (respectively, an integer) **depends entirely on how you use it** in the program.

Boolean, Character or Integer? #1

In the previous example program, we had `char` variables named `I_am_Henry` and `I_am_tall`.

We treated them as Boolean variables in the calculation subsection, but in the output subsection we had:

```
printf("I_am_Henry = %d\n", I_am_Henry);
printf("I_am_tall = %d\n", I_am_tall);
```

How can this be?

Boolean, Character or Integer? #2

```
char I_am_Henry = 1;
char I_am_tall;
…
I_am_tall = (!I_am_Henry) && … ;
…
printf("I_am_Henry = %d\n", I_am_Henry);
…
printf("I_am_tall = %d\n", I_am_tall);
```

How can it be that the **same variable** is **simultaneously a Boolean, a character and an integer**?

It turns out that **`char`** not only means character, it also means an integer of 1 byte (8 bits).

This is confusing, but you'll get used to it.

Boolean Literal Constants

In C, a ***Boolean literal constant*** can have either of two possible values (but not both at the same time, of course):

- to represent **false**: `0`
- to represent **true**: anything other than `0` (usually `1`)

Using Boolean Literal Constants #1

We can use Boolean literal constants in several ways:

- In declaring and initializing a **named constant**:

```
const char true = 1;
```

- In declaring and initializing a **variable**:

```
char I_am_getting_a_bad_grade = 0;
```

- In an **assignment**:

```
this_is_my_first_guess = 1;
```

- In an **expression**:

```
Henry_isnt_tall = Henry_is_tall && 0;
```

Using Boolean Literal Constants #2

The first two of these uses – in a named constant declaration and in a variable declaration – are considered good programming practice, **AND SO IS THE THIRD** (in an assignment), which is a way that **Booleans are different from numeric data**.

As for using Boolean literal constants in expressions, it's not so much that it's considered bad programming practice, it's just that it's kind of pointless.

What is a Boolean Expression? #1

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

In programming, a ***Boolean expression*** is a combination of:

- ***Boolean Operands***
- ***Boolean Operators***
- **Parentheses**: ( )

What is a Boolean Expression? #2

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

In programming, a ***Boolean expression*** is a combination of:

- ***Boolean Operands***, such as:
  - Boolean literal constants (`0` for **false**, nonzero for **true**)
  - Boolean named constants
  - Boolean variables
  - ***Boolean-valued function invocations***
- ***Boolean Operators***
- **Parentheses**: `(` `)`

What is a Boolean Expression? #3

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

In programming, a ***Boolean expression*** is a combination of:

- ***Boolean Operands***
- ***Boolean Operators***, such as:
  - Relational Operators (which have **numeric operands**)
  - Logical Operators
- **Parentheses**: ( )

What is a Boolean Expression? #4

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

In programming, a ***Boolean expression*** is a combination of:

- ***Boolean Operands***
- ***Boolean Operators***, such as:
  - Relational Operators (which have **numeric operands**)
    - Is Equal: ==
    - Not Equal: !=
    - Less Than: <
    - Less Than or Equal To: <=
    - Greater Than: >
    - Greater Than or Equal To: >=
  - Logical Operators
- **Parentheses**: ( )

What is a Boolean Expression? #5

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

In programming, a ***Boolean expression*** is a combination of:

- ***Boolean Operands***
- ***Boolean Operators***, such as:
  - Relational Operators (which have **numeric operands**)
  - Logical Operators
    - ***Negation*** (NOT): `!`
    - ***Conjunction*** (AND): `&&`
    - ***Disjunction*** (OR): `||`
- **Parentheses**: ( )

Boolean Expressions

Just like a numeric (arithmetic) expression,
a ***Boolean expression*** is a combination of Boolean terms (such as variables, named constants, literal constants and Boolean-valued function calls), Boolean operators (for example, `!`, `&&`, `||`, relational comparisons) and parentheses.

```
I_am_happy
!I_am_happy
it_is_raining && it_is_cold
it_is_raining || it_is_cold
(!it_is_raining) || (it_is_cold && I_am_happy)
```

Boolean Operations

Like arithmetic operations, Boolean operations come in two varieties: ***unary*** and ***binary***.

A unary operation is an operation that uses only one term; a binary operation uses two terms.

Boolean operations include:

Operation	**Kind**	**Operator**	**Usage**	**Effect**
Identity	Unary	None	`x`	No change to value of `x`
Negation	Unary	`!`	`!x`	Inverts value of `x`
Conjunction (**AND**)	Binary	`&&`	`x && y`	1 if both `x` is nonzero **AND** `y` is nonzero; otherwise 0
Disjunction (Inclusive **OR**)	Binary	`||`	`x || y`	1 if either `x` is nonzero **OR** `y` is nonzero, or both; otherwise 0

C Boolean Expression Evaluation Values

C Boolean expressions evaluate to either:

- `0` (representing **false**)
- `1` (representing **true**)

Note that **any nonzero value represents true**, but, when C evaluates a Boolean expression, then if that expression evaluates to true, then specifically its value is `1`.

Note that **only `0` represents false, ever**.

Boolean Expression Example #1

```
#include <stdio.h>

int main ()
{ /* main */
    const char true = 1, false = 0;

    printf(" true = %d,  false = %d\n",  true,  false);
    printf("!true = %d, !false = %d\n", !true, !false);
    printf("\n");
    printf("true  || true  = %d\n", true  || true);
    printf("true  || false = %d\n", true  || false);
    printf("false || true  = %d\n", false || true);
    printf("false || false = %d\n", false || false);
    printf("\n");
    printf("true  && true  = %d\n", true  && true);
    printf("true  && false = %d\n", true  && false);
    printf("false && true  = %d\n", false && true);
    printf("false && false = %d\n", false && false);
} /* main */
```

Boolean Expression Example #2

```
% gcc -o logic_expression_simple logic_expression_simple.c
% logic_expression_simple
 true = 1,  false = 0
!true = 0, !false = 1

true  || true  = 1
true  || false = 1
false || true  = 1
false || false = 0

true  && true  = 1
true  && false = 0
false && true  = 0
false && false = 0
```

Boolean Variables Example #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int true = 1;
    const int false = 0;
    int project_due_soon;
    int been_putting_project_off;
    int start_working_on_project_today;

    printf("Let's find out whether you should start working today!\n");
    printf("Is it true that you have a programming project due soon?\n");
    printf("  (Answer %d for true, %d for false.)\n", true, false);
    scanf("%d", &project_due_soon);
    printf("Is it true that you have been putting off working on it?\n");
    printf("  (Answer %d for true, %d for false.)\n", true, false);
    scanf("%d", &been_putting_project_off);
    start_working_on_project_today =
        project_due_soon && been_putting_project_off;
    printf("Is it true that you should start ");
    printf("working on it today?\n");
    printf("ANSWER: %d\n",
        start_working_on_project_today);
} /* main */
```

Boolean Variables Example #2

```
% gcc -o pp_logic pp_logic.c
% pp_logic
Let's find out whether you should start working today!
Is it true that you have a programming project due soon?
  (Answer 1 for true, 0 for false.)
1
Is it true that you have been putting off working on it?
  (Answer 1 for true, 0 for false.)
1
Is it true that you should start working on it today?
ANSWER: 1
```