Relational and Logical Expressions

COP 3223C – Introduction to Programming with C

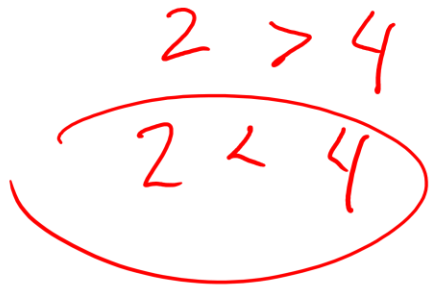
Fall 2025

Yancy Vance Paredes, PhD

Recall

- We have previously discussed various arithmetic operators; this time, we will learn a new set of operators
- The motivation is to be able to compare values and combine conditions
- This is key for conditionals and loops, enabling programs to change behavior

Relational and Equality Operators



Symbol	Operation	Type
<	Less than	Relational
<=	Less than or equal to	Relational
>=	Greater than or equal to	Relational
>	Greater than	Relational
==	Equal to	Equality
!=	Not equal to	Equality

Logical Operators

Symbol	Operation
&&	AND
||	OR
!	NOT (unary operator)

& address of

Truth Tables /1

Operand 1	Operand 2	&& (AND)
0	0	0
0	1	0
1	0	0
1	1	1

Operand 1	Operand 2	|| (OR)
0	0	0
0	1	1
1	0	1
1	1	1

Truth Tables /2

Operand	! (NOT)
0	1
1	0

Formulating Questions

- Questions can be formulated using a **combination of operators**
- Mostly, it involves **relational** and **equality** operators
- Complex questions involve using **logical** operators

Practice

We want to check if `var` is equal to 10

var == 10

Common Error

- Using = instead of == when checking for equality
- Remember = is the **assignment operator**
- If this is our solution to the prior problem, we have a **logic error**
- Remember C only checks if non-zero or zero!

Practice /1

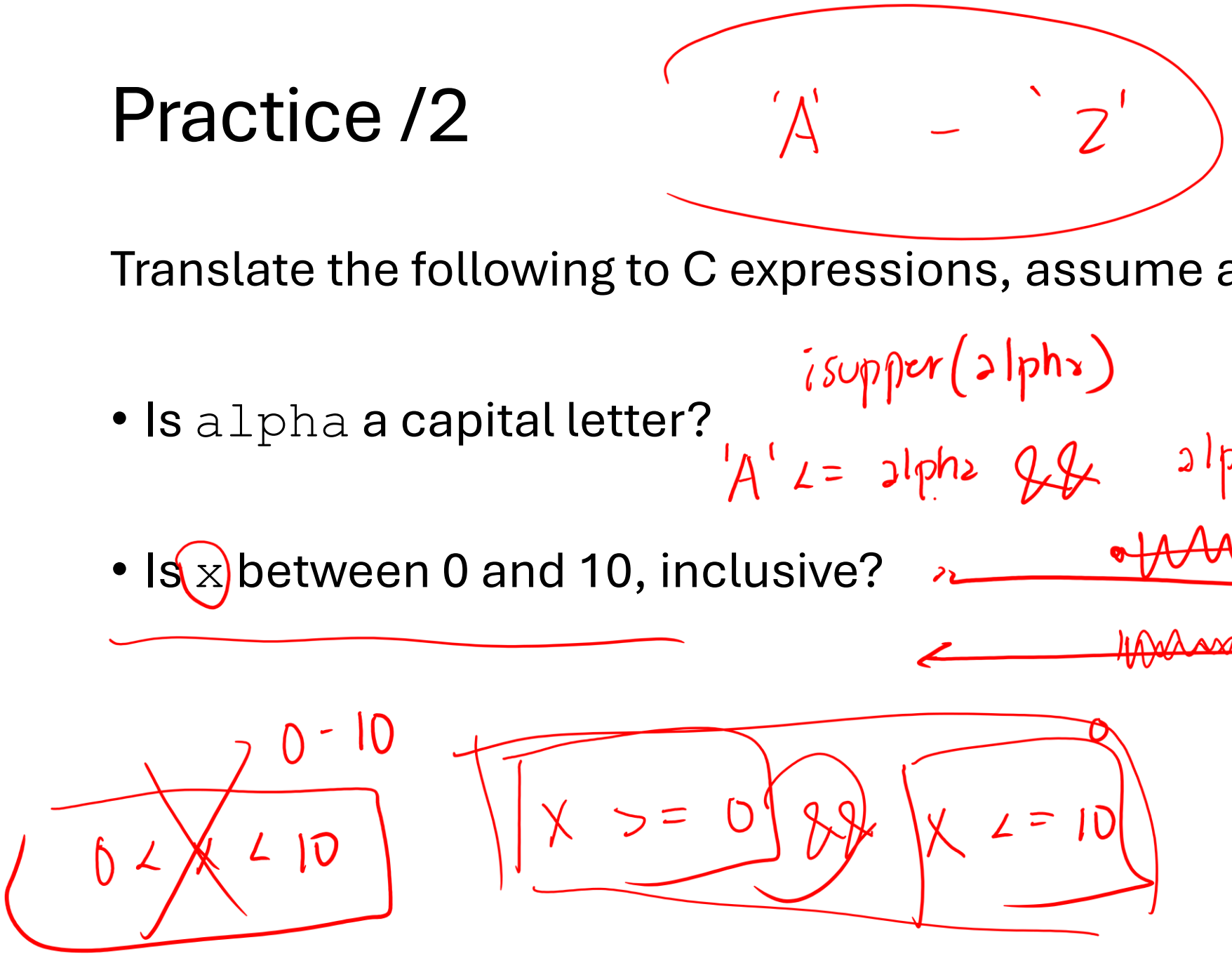
Translate the following to C expressions, assume all are **`int`**s:

- Is `sum` greater than 10? sum > 10
- Is `num` a positive integer? num > 0
- Is `val` an even number? val % 2 == 0

Practice /2

Translate the following to C expressions, assume all are **`int`**s:

- Is `alpha` a capital letter?
- Is `x` between 0 and 10, inclusive?



Function	What it does...
`islower()`	Checks to see if a character is lowercase.
`isupper()`	Checks to see if a character is uppercase.
`tolower()`	Converts a character to its lowercase version.
`toupper()`	Converts a character to its uppercase version.
`isalpha()`	Checks to see if a character an alphabet.
`isdigit()`	Checks to see if character is a digit.
`isspace()`	Checks to see if character is a whitespace.
`isalnum()`	Checks to see if character is either alphabet or number.
`ispunct()`	Checks to see if character is punctuation.

Common Error

Note that the following is syntatically correct in C:

```
0 <= x <= 10
```

So is:

```
0 < x < 10
```

If this is our solution to the previous problem, it has a **logic error**

Your Turn! /1

De Morgan's Law

Translate the following to a C expression, assume it is an **`int`**:

Is `x` not in between 0 and 10, inclusive?

```
!(0 <= x && x <= 10)
```

Your Turn! /2

In the English alphabet, what are vowels and what are consonants?

Order of Precedence /1

Operator	Precedence
! + − & (unary operators)	highest
* / %	
+ −	
< <= >= >	
== !=	
&&	
||	
=	lowest

Order of Precedence /2

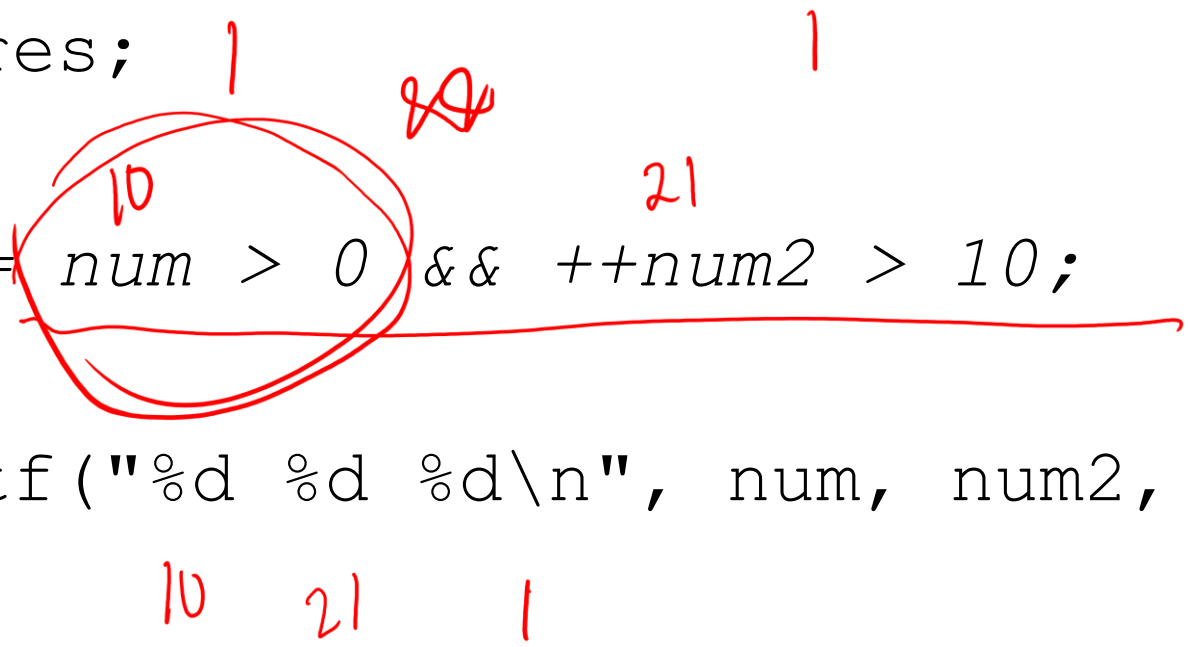
- It is **best practice** to use parentheses to **prevent ambiguity**
- This is mainly for the programmer
- Remember expectation vs. reality

Code Tracing /1

```
int num = 10;
int num2 = 20;
int res;

res = num > 0 && ++num2 > 10;

printf("%d %d %d\n", num, num2, res);
```

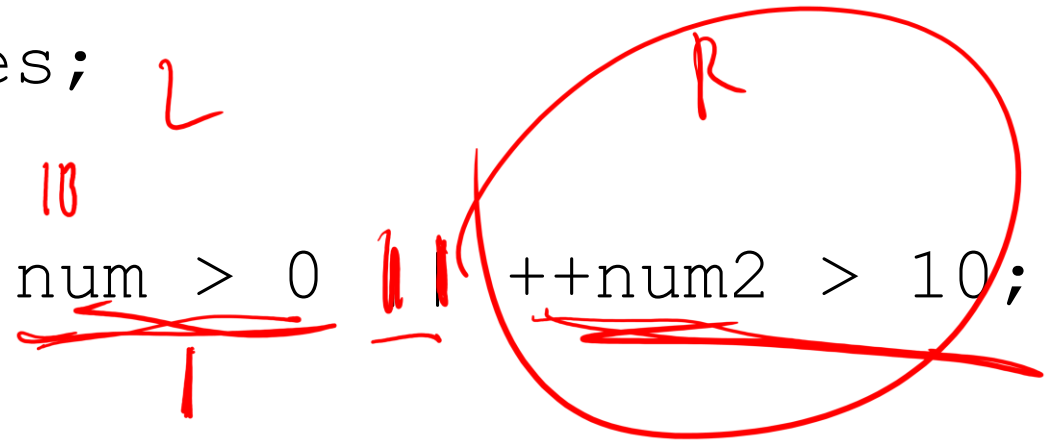


Code Tracing /2

```
int num = 10;
int num2 = 20;
int res;

res = num > 0 || ++num2 > 10;

printf("%d %d %d\n", num, num2, res);
```



Discussion

- The previous examples demonstrated the concept of **short-circuit evaluation**

- If the result of a logical expression can be determined with **certainty**, the remaining conditions are **not evaluated**

Challenge

What is the output?

```
int x = 4, y = 9, z = 2;
int result = x + y * z < 20 || y - z < 10 && x * z == 9;



printf("%d", result);
```

Questions?