PIC 10A 1A

TA: Bumsu Kim

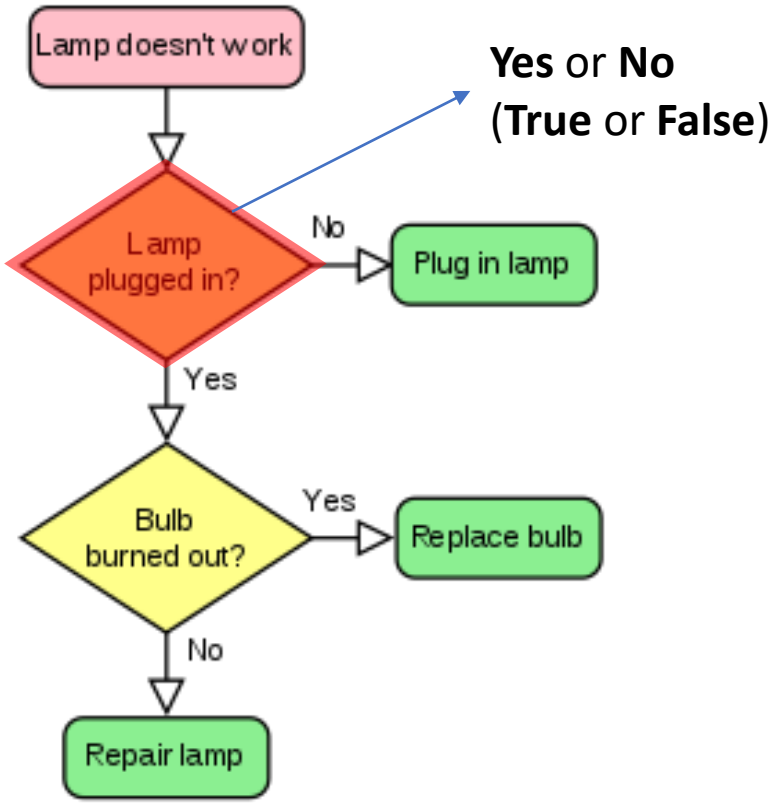
UCLA

Today…

- Control Flow – `if` and `if-else`
  - Type `bool`
- Good Coding Styles
- Exercise Problem – Grade Calculator v.1

Control Flow – `if` / `else` / `else if`

- Flowchart?



- Sometimes your program should run differently according to the current state
- In the world of programming, every question is basically a yes-no question
  - It's quite natural if you recall how the computer works; it's always binary. A light bulb (*bit*) can only be *on(1)/off(0)*
- More formally, the program evaluates an expression and determines if it is `true` or `false`
- Recall the "bool" (Boolean) type:

```
bool true_or_false; // bool literals: true and false
```

UCLA

The Type `bool`

- The type `bool` is one of the *fundamental types* in C++

- `bool` uses 1 byte (not 1 bit!) to store a logical value, "`true`" or "`false`".
  - `true` and `false` are `bool` literals

```
Microsoft Visual Studio Debug Console
1
true
```

- Comparison operators return `bool` values

```
5 < 3;
(bool)false
```

```
int a = 5, b = 3;
bool comparison = (a >= b);
cout << comparison << endl;
```

```
Microsoft Visual Studio Debug Console
1
```

```
int a = 5, b = 3;
bool comparison = (a >= b);
cout << comparison << endl;

cout << boolalpha;
cout << comparison << endl;
```

UCLA

The Type `bool`

- Every numeric value other than 0 is implicitly casted to `true`
  - 0 is false
- Other expressions may or may not be convertible to `bool`
  - More on this later…

- Just like operators +, –, /, * that are defined for the numeric types, there are operators designed for Boolean variables, called **logical operators**
  - Some of the common logical operators are NOT(unary), AND(binary), and OR (binary)
  - e.g.
    - **(NOT(** `true` **))** evaluates `false`
    - **(** `true` **AND** `false` **)** evaluates `false`, and **(** `true` **AND** `true` **)** evaluates `true`
    - **(** `false` **OR** `true` **)** evaluates `true`, etc.
  - Operator NOT : ! → ! (`true`) stands for **NOT(** `true` **)**
  - Operator AND : && → (`true && false`) stands for **(** `true` **AND** `false` **)**
  - Operator OR : || → (`false || true`) stands for **(** `false` **OR** `true` **)**

UCLA

Boolean Algebra

- Recall that
  - A && B is true only when A and B are both true
  - A || B is false only when A and B are both false
- De Morgan's law:
  - !(A && B) == (!A) || (!B)
  - !(A || B) == (!A) && (!B)

Truth Tables

&&		P	
		T	F
Q	T	T	F
	F	F	F

||		P	
		T	F
Q	T	T	T
	F	T	F

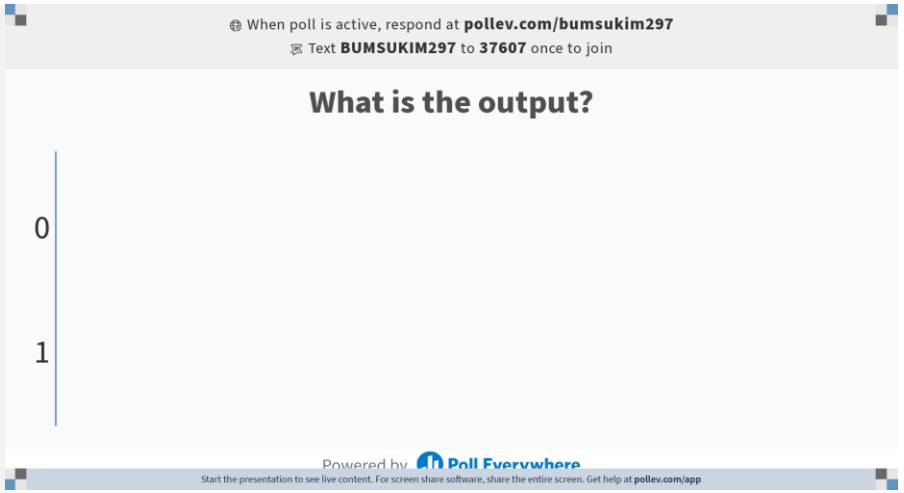
UCLA

Exercise Problem 1 (Boolean Algebra)

2. Consider the following program.

```
#include <iostream>
using namespace std;

int main() {
  bool b1 = true, b2 = false;
  cout << ( !(b1 && !b2) || b2 ) << endl;
}
```



What is the output?

A. 0

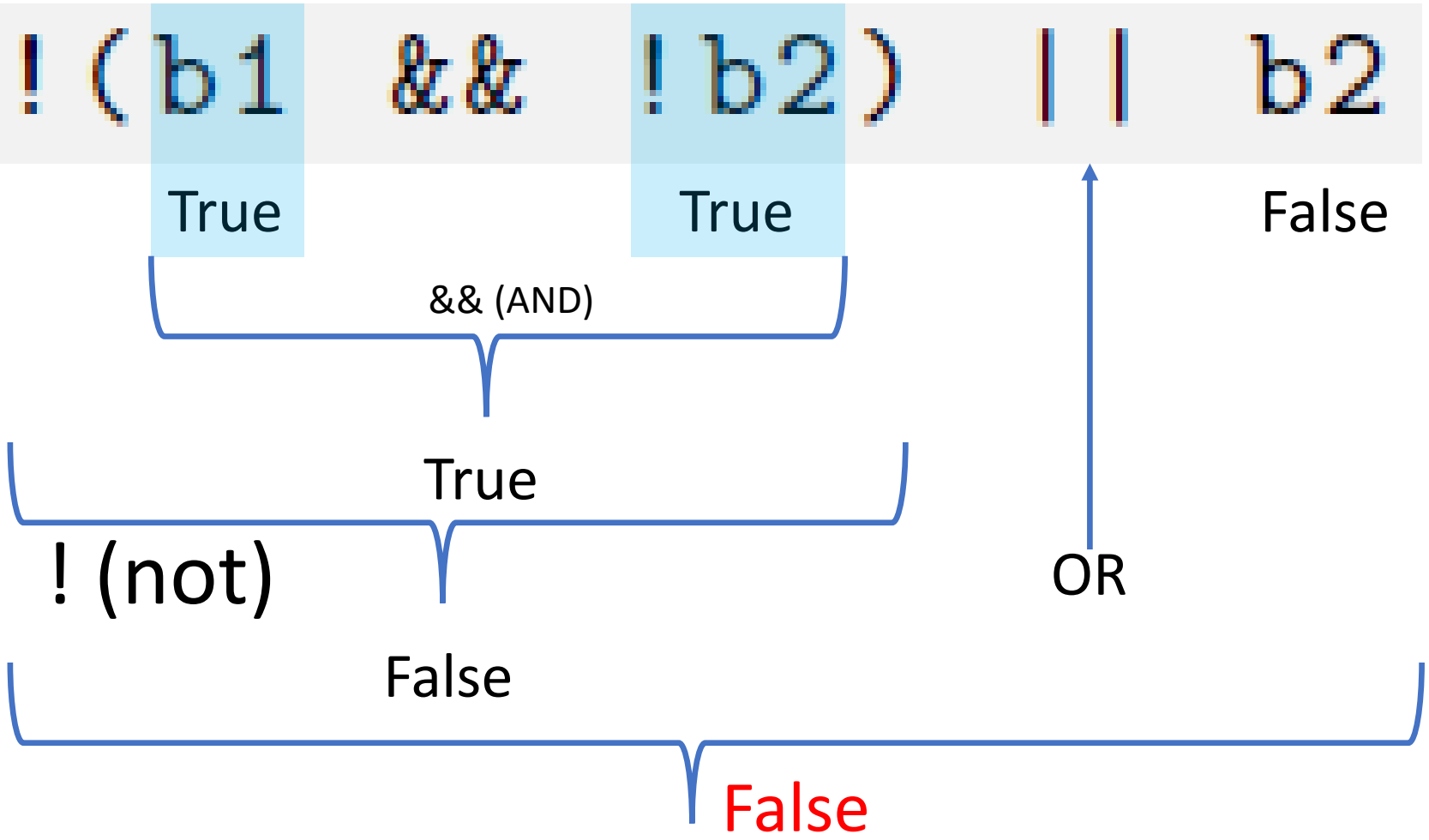
B. 1

- Recall that `true` outputs 1, and `false` outputs 0 on the console
  - (Can be strings "true" and "false," respectively, if you use the option `std::boolalpha`)

Exercise Problem 1 (Boolean Algebra)

```
bool b1 = true, b2 = false;
```

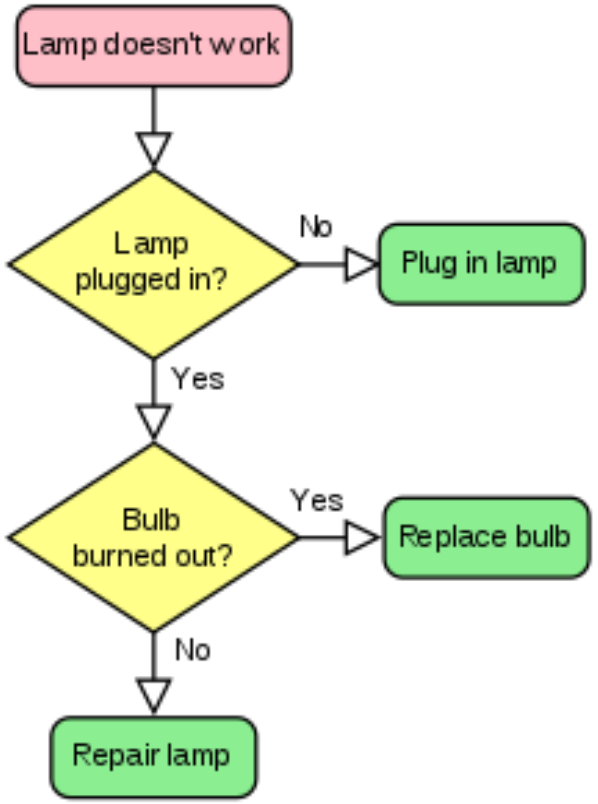
```
!(b1 && !b2) || b2
```



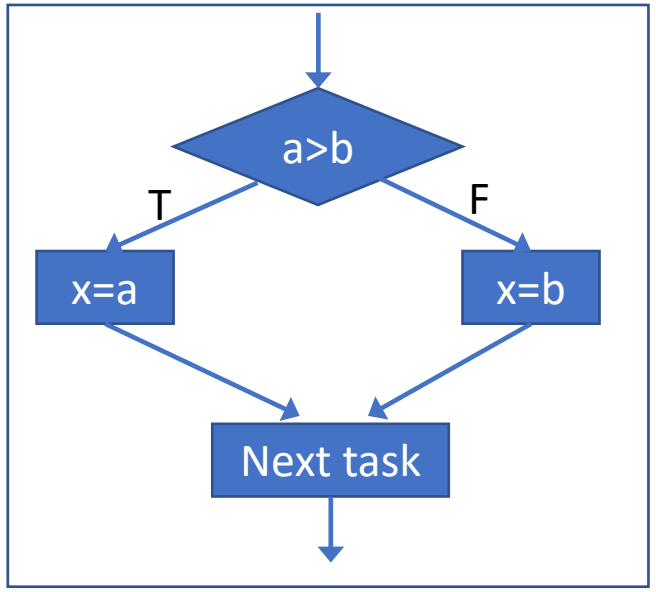
UCLA

Control Flow – `if` / `else` / `else if`

- Now you're ready to implement a flow chart like
- Only need to use "`if`" and "`else`"



- For instance, if you want `x = max(a, b)`, the flow char would look like the following:



- The implementation is simple:

```
if (a > b) { x = a; }
else { x = b; }
```

UCLA

A Useful Digression – Coding Style

- Recall that C++ is quite generous about the white spaces
- It is important, however, to follow a ***standard*** coding style for readability
- So,

```
if (a > b) { x = a; }
else { x = b; }
```

or even

```
if (a > b) x = a; else x = b;
```

Without curly braces!

is (grammatically) allowed, it is better to write it as either

```
if (a > b) {
      x = a;
} else {
      x = b;
}
```

or

```
if (a > b)
{
      x = a;
}
else
{
      x = b;
}
```

UCLA

A Useful Digression – Coding Style

- For instance, Google has its own C++ style guide:

Each of `if`, `else`, and `else if` belong on separate lines. There should be a space between the `if` and the open parenthesis, and between the close parenthesis and the curly brace (if any), but no space between the parentheses and the condition.

```
if (condition) {  // no spaces inside parentheses
  ...  // 2 space indent.
} else if (...) {  // The else goes on the same line as the closing brace.
  ...
} else {
  ...
}
```

Google C++ Style Guide: https://google.github.io/styleguide/cppguide.html#Conditionals

```
if(condition) {   // Bad - space missing after IF.
if ( condition ) { // Bad - space between the parentheses and the condition
if (condition){   // Bad - space missing before {.
if(condition){    // Doubly bad.
```

- Conclusion: Good coding style is important!

UCLA

Control Flow – `if` / `else` / `else if`

- Good coding practice: always use curly braces {} with control statements
- Even when you only need to execute a single expression

Bad

```
if (a > b)
    x = a;
else
    x = b;
```

vs

```
if (a > b) {
    x = a;
} else {
    x = b;
}
```

Good

UCLA

Use Curly Braces even with a Single Statement

A Quiz problem from a previous 10A course

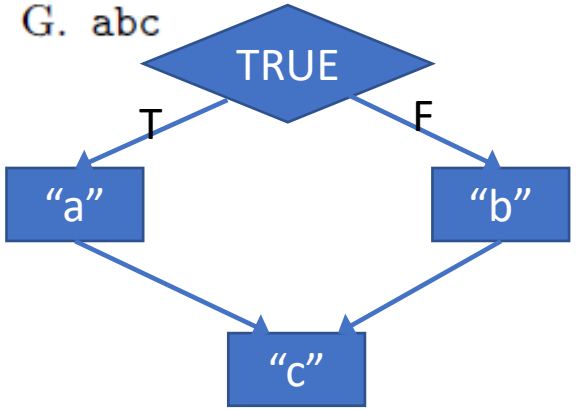
7. Consider the following program, which is poorly indented on purpose.

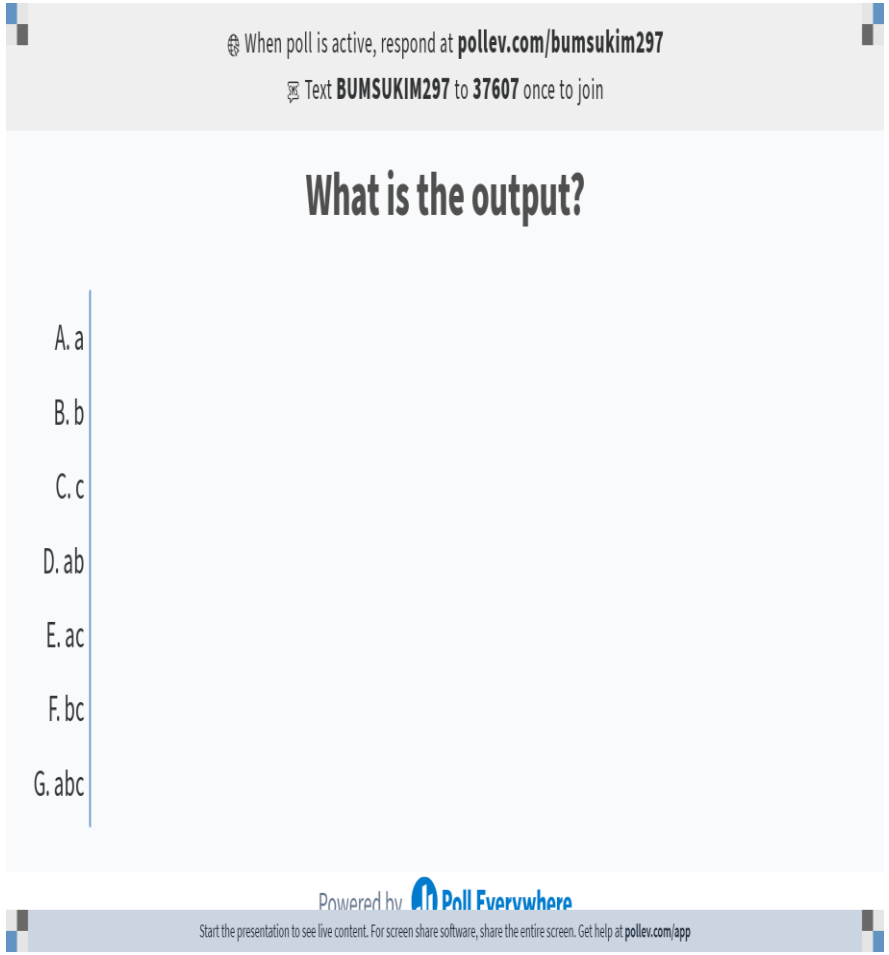
```
#include <iostream>
using namespace std;

int main() {
if (true)
cout << "a";
else
cout << "b";
cout << "c";
}
```

What is the output?

A. a B. b C. c D. ab E. ac F. bc G. abc

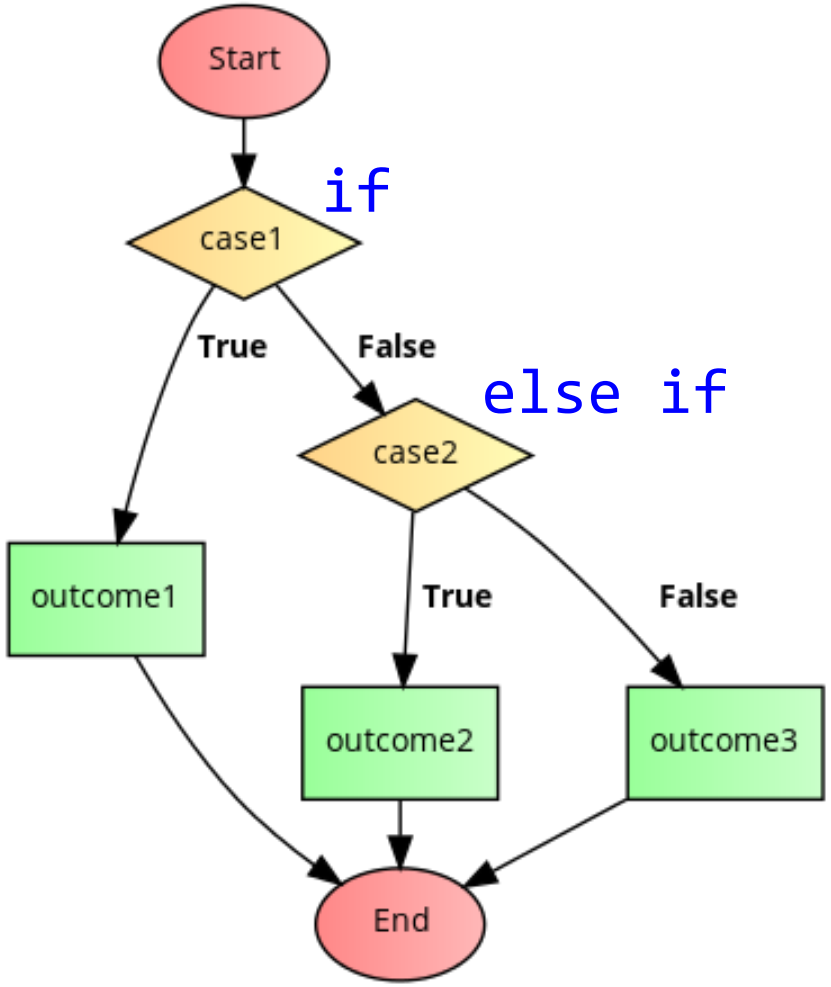




UCLA

Control Flow – `if` / `else` / `else if`

- `else if` is just a compound form of `else{ if ( … ) }`



Implementation

```
if (case1) {
      outcome1;
}
else if (case2) {
      outcome2;
}
else {
      outcome3;
}
```

UCLA

Exercise – Grade Calculator ver.1

- Write a program that will calculate a student's final score in some class on the following dual grading system:
- Assume that there will be 3 homework assignments total, and the lowest homework score will be dropped.

Scheme A	Scheme B
Midterm Exam 30%	Midterm exam score dropped
Final Exam 40%	Final Exam 70%
Homework 30%	Homework 30%

- The maximum of the two scores obtained from the two schemes will be the final score
- In addition to printing the final score, you should also determine the letter grade based on the following scale:

```
90 <= A <= 100, 80 <= B < 90, 70 <= C < 80,  60 <= D < 70,  0 <= F < 60.
```

- Input and output should be exactly of the following format:

```
Please enter the midterm score (0 - 100): 80.0
Please enter the final exam score (0 - 100): 85.0
Please enter the homework 1 score (0 - 100): 80.0
Please enter the homework 2 score (0 - 100): 90.0
Please enter the homework 3 score (0 - 100): 40.0
Your final score based on Scheme A is 83.5
Your final score based on Scheme B is 85
 Your final score is 85
Your course grade is B
```

UCLA

Your Feedback is welcome

- Don’t hesitate to give a feedback on the discussion
- You can use a link in BruinLearn (Google Form)