An Exercise in Comprehension

- Choose either blue or green.
- Raise your hand only if you have either a right hand or a left hand.
- Did you do either HW#1 or HW#2?

What's going on here?
Each of these statements uses the ***same*** *English form, yet we think of them differently.*
Computers can't work that way!

The Devil You Know

You probably have a sense of what AND, OR, and NOT mean… but is it correct *for C*?

In programs (and logic) these have precise meanings.

Use your intuitive understanding, but keep an eye out for where it differs from C's interpretation.

Don't believe me?

OK, what's your intuitive sense of 2/3?

CSE142
Computer Programming I

(Not So)
Complex Conditions

… or just what do you mean by "or"?

Overview

- Complex conditions
- Boolean operators
- Using Booleans
- Understanding Booleans
  - Truth tables
  - Negating a condition
  - DeMorgan's Laws

Complex Conditionals

Our natural notion of truth is imprecise…

Yet programs need to make complex decisions:

- if I have at least $15 or you have at least $15, then we can go to the movies.
- if the temperature is below 32 degrees and it's raining, then it's snowing.
- if it's not the case that it's Saturday or Sunday, then it's a work day.

Precise Manipulation of Truth: Boolean Operators in C

Complex conditionals often invoke words like AND, OR, NOT, TRUE, and FALSE.

The Boolean operators AND, OR, and NOT have symbols and precise definitions in C:

AND	OR	NOT
&&	||	!

As you know, TRUE and FALSE are 1 and 0 in C. We can #define words for them:

```
#define TRUE   1
#define FALSE 0
```

Complex Conditionals in C

if I have at least $15 or you have at least $15, then we can go to the movies.

```
if (myMoney >= 15.0 || yourMoney >= 15.0) {
        bCanGoToMovies = TRUE;
}
```

if the temperature is below 32 degrees and it's raining, then it's snowing.

```
if (temp < 32.0 && bIsRaining) {
        bIsSnowing = TRUE;
}
```

I-7
4/23/01

Nested if vs. && (AND)

```
if (age < 25) {
        if (gender == 'M') {
                insurance_rate *= 2;
        }
}

if ((age < 25) && (gender == 'M')) {
        insurance_rate *= 2;
}
```

I-8
4/23/01

Using ! (NOT)

if it's not the case that it's Saturday or Sunday, then it's a work day.

```
#define SATURDAY 6
#define SUNDAY     7
if (! (today ==  SATURDAY || today == SUNDAY) ) {
    bWeekday = TRUE;
}
if (bWeekday) {
    bMustWork = TRUE;
}
```

Hey! We didn't check what bWeekday equals!
Will that work?

I-9
4/23/01

Conditional Expressions… Are Just Expressions

Like arithmetic expressions, they have a value:

TRUE (non-zero) or FALSE (zero)

(values are actually int)

When using relational (<, ==, …) and Boolean (&&, ||, !) operators: TRUE is 1; FALSE is 0

We can store the value in an int variable!

```
m = (z >= 0.0);
```

I-10
4/23/01

Boolean Operators… Are Just Operators

High (Evaluate First) Low (Evaluate Last)

! Unary - | * / % | - + | < > <= >= | == != | && | ||

a = 2;
b = 4;
*z = (a + 3 >= 5 && !(b < 5)) || a * b + b != 7 ;*

I-11
4/23/01

```
z = (a + 3 >= 5  &&  !(b < 5)) || a * b + b != 7
z = (a + 3 >= 5  &&  ! 1) || a * b + b != 7
z = (a + 3 >= 5  &&  0) || a * b + b != 7
z = (5 >= 5  &&  0) || a * b + b != 7
z = (1  &&  0) || a * b + b != 7
z = 0 || a * b + b != 7
z = 0 || 8 + b != 7
z = 0 || 12 != 7
z = 0 || 1
z = 1
1
```

a = 2
b = 4

But should we count on this?
As before, use parentheses where it's unclear!

I-12
4/23/01

Understanding Boolean Ops: Truth Tables

A "truth table" lists all possible combinations of values for a Boolean expression with the result of each combination.

This is the basic way we understand Boolean operators!

Truth Tables for && and ||: Know Your P && Q

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

P and Q stand for any conditional expressions

Truth Table for ! (NOT)

P	!P
T	F
F	T

Negating a Condition

Suppose we want a while loop to terminate as soon as either x is 17 or x is 42.

Which is it?

- while (x!=17 || x!=42) …
- while (x!=17 && x!=42) …
- either way? something else?

Truth tables and DeMorgan's laws give us tools for answering such questions.

! Example

```
int high_risk ;
high_risk =  (age < 25  &&  sex  ==  'M' ) ;
if ( high_risk ) {      /* Do nothing */
} else {
    printf ( "Cheap rates. \n") ;
}
```

```
if ( ! high_risk ) {
        printf ( "Cheap rates. \n") ;
}
```

P	!P
T	F
F	T

Equivalence of Conditional Expressions

```
if ( ! (age < 25  &&  sex  ==  'M' ) )
        printf ( "Cheap rates. \n") ;
```

is equivalent to…

```
if ( age >= 25  ||  sex  !=  'M' ) )
        printf ( "Cheap rates. \n") ;
```

Or… is it?
How can we tell??

DeMorgan to the Rescue

DeMorgan's laws state how to handle NOT in complex conditions.

They state:

! (P && Q) is equivalent to **(!P || !Q)**

! (P || Q) is equivalent to **(!P && !Q)**

This applies for any Boolean expressions P and Q, which might themselves be complex expressions

How can we prove this?

Proof of DeMorgan's Law

Is it really true that !(P&&Q) == (!P || !Q) ?

P	Q	(P&&Q)	!(P&&Q)	!P	!Q	(! P || !Q)
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

Exercise: Prove the other law

Back to the Question…
age < 25 && sex == 'M'

We wanted a while loop to terminate as soon as either x is 17 or x is 42. I.e., loop should terminate if (x==17 || x==42)

So the loop condition is

```
while ( ! (x==17 || x==42) ) ...
```

Using DeMorgan's laws, we can rewrite as

```
while (x != 17 && x != 42) ...
```

A truth table would show that

```
while (x != 17 || x != 42) ...
```

is wrong!

Summary

- Boolean operators make our *sense* of logic precise and computable.
- Complex conditions are useful in loops, if statements, and even assignment statements!
- Operators &&, ||, and ! are part of C.
- TRUE and FALSE can be #defined.
- Truth tables and DeMorgan's laws help evaluate complex expressions.

QOTD: Just Exactly WHAT Do You Mean By That?

We started this all off with "either… or" meaning either one of these is true or the other is true **but not both**.

How can we express "either P or Q" in C?