## Prelim

Current plan is to have the prelim on March 9 at 7:30 (the regularly scheduled time), not March 10. This means it conflicts with:

- CHEM 106
- ENGRD 202
- ILRST 210
- OR&IE 321
- OR&IE 521
- PHYS 213
- PHYS 214
- T&AM 310

If you're taking one of those courses and it is actually having a prelim, let me and/or Prof. Keich know.



## Logic Concepts

The most common mathematical argument is an *implication*.

- *If* $x = 2$ *then* $x^2 = 4$

The implication is sometimes not as obvious:

- $x^2 = 4$ if $x = 2$
- $x^2 = 4$ when $x = 2$
- $x = 2$ implies $x^2 = 4$
- Suppose $x = 2$. Then $x^2 = 4$.
- whenever $x = 2$, $x^2 = 4$
- $x = 2$ only if $x^2 = 4$
- The condition $x = 2$ is sufficient for $x^2 = 4$
- The condition $x^2 = 4$ is necessary for $x = 2$

Note that the order of $x = 2$ and $x^2 = 4$ change.

We denote the implication "If $A$ then $B$" by

$$A \Rightarrow B$$

YOU NEED TO LEARN TO RECOGNIZE IMPLICATIONS.



Implications chain:

- If $A \Rightarrow B$ and $B \Rightarrow C$ then $A \Rightarrow C$
- $((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)$

The *converse* of $A \Rightarrow B$ is $B \Rightarrow A$.

- *They are not equivalent.*
- $x = 2 \Rightarrow x^2 = 4$ is true; $x^2 = 4 \Rightarrow x = 2$ is not ($x$ could be $-2$)

The contrapositive of $A \Rightarrow B$ is $\neg B \Rightarrow \neg A$.

- $\neg$ stands for negation
- A statement is *equivalent* to its contrapositive.
- If $x^2 \neq 4$ then $x \neq 2$.
- If you're asked to prove $A \Rightarrow B$, one way to do it (which is sometimes easier) is to show $\neg B \Rightarrow \neg A$



## Equivalence

If both $A \Rightarrow B$ and $B \Rightarrow A$ are true, we write:

$$A \Leftrightarrow B$$

$A$ is *equivalent* to $B$ ($A$ if and only if $B$; $A$ iff $B$)

$$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$$

$S$ is a square if and only if $S$ is both a rectangle and a rhombus.

- $S$ being a rectangle and a rhombus is sufficient for $S$ to be a square
- $S$ being a rectangle and a rhombus is necessary for $S$ to be a square

## Quantifiers

*Quantifiers* are words like *every*, *all*, *some*:

- *Every* prime other than two is odd
- *Some* real numbers are not integers

*Any* is ambiguous: sometimes it means *every*, and sometimes it means *some*

- Anybody knows that $1 + 1 = 2$
- He'd be happy to get an $A$ in any course

Avoid *any*: use every (= all) or some.



## Negation

The negation of $A$, written $\neg A$, is true exactly if $A$ is false:

- The negation of $x = 2$ is $x \neq 2$

Be careful when negating quantifiers!

- What is the negation of $A =$ "Some of John's answers are correct"
- Is it $B=$ "Some of John's answers are not correct"
  - No! $A$ and $B$ can be simultaneously true
- It's "All of John's answers are incorrect".



## Algorithms

An *algorithm* is a recipe for solving a problem.

In the book, a particular language is used for describing algorithms.

- You need to learn the language well enough to read the examples
- You need to learn to express your solution to a problem algorithmically and *unambiguously*
- YOU DO NOT NEED TO LEARN IN DETAIL ALL THE IDIOSYNCRACIES OF THE PARTICULAR LANGUAGE USED IN THE BOOK.
  - You will not be tested on it, nor will most of the questions in homework use it



## Main Features of the Language

- Assignment statements
  - $x \leftarrow 3$
- **if** ... **then** ... **else** statements
  - **if** $x = 3$ **then** $y \leftarrow y + 1$ **else** $y \leftarrow z$ **endif**
  - $x = 3$ is a *test* or *predicate*; it evaluates to either **true** or **false**
- Selection statement

**if** $B_1$ **then** $S_1$
  $B_2$ **then** $S_2$
  $\vdots$
  $B_k$ **then** $S_k$
  [**else** $S_{k+1}$]
**endif**

# Iteration

Lots of variants:

**repeat until** $B$
  $S$
**endrepeat**

or

**repeat**
  $S$
**endrepeat when** $B$

or

**repeat while** $B$
  $S$
**endrepeat**

(Same as **while** $B$ **do** $S$)

or

**for** $C = 1$ **to** $n$
  $S$
**endfor**



# Input and Output

Programs start with input statements of the form:

**Input** $x$, $a_0$, ..., $a_k$

- the values of the variables $x, a_0, \ldots, a_k$ are assumed to be available at the beginning of the program

Programs end with output statements of the form:

**Output** $P$

**Example**

**Input** $a_0, a_1, \ldots, a_n, x$

  $P \leftarrow a_n$
  **for** $k = 1$ **to** $n$
    $P \leftarrow Px + a_{n-k}$
**Output** $P$

What does this compute?



# Procedure Calls

It is useful to extend our algorithmic language to have procedures that we can call repeatedly. For example, we may want to have a procedure for computing gcd or factorial, that we can call with different arguments. Here's the notation used in the book:

**procedure** Name(variable list)
  procedure body (includes a **return** statement)
**endpro**

- The **return** statement returns control to the portion of the algorithm from where the procedure was called

**Example:**

**procedure** Factorial($n$)
  $fact \leftarrow 1$
  $m \leftarrow n$
  **repeat until** $m = 1$
    $fact \leftarrow fact \times m$
    $m \leftarrow m - 1$
  **endrepeat**
  **return** *fact*
**endpro**



# Recursion

*Recursion* occurs when a procedure calls itself.

Classic example: Towers of Hanoi

**Problem:** Move all the rings from pole 1 and pole 2, moving one ring at a time, and never having a larger ring on top of a smaller one.

How do we solve this?

- Think recursively!
- Suppose you could solve it for $n-1$ rings? How could you do it for $n$?

**Solution**

- Move top $n-1$ rings from pole 1 to pole 3 (we can do this by assumption)
  - Pretend largest ring isn't there at all
- Move largest ring from pole 1 to pole 2
- Move top $n-1$ rings from pole 3 to pole 2 (we can do this by assumption)
  - Again, pretend largest ring isn't there

This solution translates to a recursive algorithm:

- Suppose robot($r \rightarrow s$) is a command to a robot to move the top ring on pole $r$ to pole $s$
- Note that if $r, s \in \{1, 2, 3\}$, then $6-r-s$ is the other number in the set

**procedure** H($n, r, s$) [Move $n$ disks from $r$ to $s$]
  **if** $n = 1$ **then** robot($r \rightarrow s$)
    **else** $H(n-1, r, 6-r-s)$
      robot($r \rightarrow s$)
      $H(n-1, 6-r-s, s)$
  **endif**
  **return**
**endpro**



# Tree of Calls

Suppose there are initially three rings on pole 1, which we want to move to pole 2:



# Analysis of Algorithms

For a particular algorithm, we want to know:

- How much time it takes
- How much space it takes

What does that mean?

- In general, the time/space will depend on the input size
  - The more items you have to sort, the longer it will take
- Therefore want the answer as a function of the input size
  - What is the best/worst/average case as a function of the input size.

Given an algorithm to solve a problem, may want to know if you can do better.

- What is the *intrinsic complexity* of a problem?

This is what *computational complexity* is about.



# Towers of Hanoi: Analysis

**procedure** H($n, r, s$) [Move $n$ disks from $r$ to $s$]
  **if** $n = 1$ **then** robot($r \rightarrow s$)
    **else** $H(n-1, r, 6-r-s)$
      robot($r \rightarrow s$)
      $H(n-1, 6-r-s, s)$
  **endif**
  **return**
**endpro**

Let $h_n$ = # moves to move $n$ rings from pole $r$ to pole $s$.

- Clearly $h_1 = 1$
- Algorithm shows that $h_n = 2h_{n-1} + 1$
  - $h_2 = 3$; $h_3 = 7$; $h_4 = 15$; ...
  - $h_n = 2^n - 1$

We'll prove this formally later, when we also show that this is optimal.

## Binary Search: Analysis

Sequential search is terrible for finding a word in a dictionary. Can do much better with random access.

- it's like playing 20 questions — cut the search space in half with each question!

**Input** $n$ [number of words in list]
$w_1, \ldots, w_n$ [alphabetized list]
$w$ [search word]

**Algorithm BinSearch**

$F \leftarrow 1$; $L \leftarrow n$ [Initialize range]
$i \leftarrow \lfloor (F+L)/2 \rfloor$
**repeat until** $w = w_i$ or $F > L$
  **if** $w < w_i$ **then** $L \leftarrow i-1$ **else** $F \leftarrow i+1$ **endif**
  $i \leftarrow \lfloor (F+L)/2 \rfloor$
**end repeat**
**if** $w = w_i$ **then** print $i$ **else** print 'failure' **endif**

How many times do we go through the loop?

- Best case: 0
- Average case: too hard for us
- Worst case: $\lfloor \log_2(n) \rfloor + 1$
  - After each loop iteration, $F - L$ is halved.



## Methods of Proof

One way of proving things is by induction.

- That's coming next.

What if you can't use induction?

Typically you're trying to prove a statement like "Given $X$, prove (or show that) $Y$". This means you have to prove

$$X \Rightarrow Y$$

In the proof, you're allowed to assume $X$, and then show that $Y$ is true, using $X$.

- A special case: if there is no $X$, you just have to prove $Y$ or *true* $\Rightarrow Y$.

Alternatively, you can do a *proof by contradiction*: Assume that $Y$ is false, and show that $X$ is false.

- This amounts to proving

$$\neg Y \Rightarrow \neg X$$



## Example

**Theorem** $n$ is odd iff $n^2$ is odd, for $n \in N^+$.

**Proof:** We have to show

1. $n$ odd $\Rightarrow n^2$ odd
2. $n^2$ odd $\Rightarrow n$ odd

For (1), if $n$ is odd, it is of the form $2k+1$. Hence,

$$n^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$$

Thus, $n^2$ is odd.

For (2), we proceed by contradiction. Suppose $n^2$ is odd and $n$ is even. Then $n = 2k$ for some $k$, and $n^2 = 4k^2$. Thus, $n^2$ is even. This is a contradiction. Thus, $n$ must be odd.



## A Proof By Contradiction

**Theorem:** $\sqrt{2}$ is irrational.

**Proof:** By contradiction. Suppose $\sqrt{2}$ is rational. Then $\sqrt{2} = a/b$ for some $a, b \in N^+$. We can assume that $a/b$ is in lowest terms.

- Therefore, $a$ and $b$ can't both be even.

Squaring both sides, we get

$$2 = a^2/b^2$$

Thus, $a^2 = 2b^2$, so $a^2$ is even. This means that $a$ must be even.

Suppose $a = 2c$. Then $a^2 = 4c^2$.

Thus, $4c^2 = 2b^2$, so $b^2 = 2c^2$. This means that $b^2$ is even, and hence so is $b$.

Contradiction!

Thus, $\sqrt{2}$ must be irrational.

## Induction

This is perhaps the most important technique we'll learn for proving things.

**Idea:** To prove that a statement is true for all natural numbers, show that it is true for 1 (*base case* or *basis step*) and show that if it is true for $n$, it is also true for $n+1$ (*inductive step*).

- The base case does not have to be 1; it could be 0, 2, 3, ...
- If the base case is $k$, then you are proving the statement for all $n \geq k$.

It is sometimes quite difficult to formulate the statement to prove.

IN THIS COURSE, I WILL BE VERY FUSSY ABOUT THE FORMULATION OF THE STATEMENT TO PROVE. YOU MUST STATE IT VERY CLEARLY. I WILL ALSO BE PICKY ABOUT THE FORM OF THE INDUCTIVE PROOF.



## Writing Up a Proof by Induction

1. State the hypothesis very clearly:
   - Let $P(n)$ be the statement ... [some statement involving $n$]
2. The basis step
   - $P(k)$ holds because ... [where $k$ is the base case, usually 0 or 1]
3. Inductive step
   - Assume $P(n)$. We prove $P(n+1)$ holds as follows ... Thus, $P(n) \Rightarrow P(n+1)$.
4. Conclusion
   - Thus, we have shown by induction that $P(n)$ holds for all $n \geq k$ (where $k$ was what you used for your basis step). [It's not necessary to always write the conclusion explicitly.]



## A Simple Example

**Theorem:** For all positive integers $n$,

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}.$$

**Proof:** By induction. Let $P(n)$ be the statement

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}.$$

*Basis:* $P(1)$ asserts that $\Sigma_{k=1}^{1} k = \frac{1(1+1)}{2}$. Since the LHS and RHS are both 1, this is true.

*Inductive step:* Assume $P(n)$. We prove $P(n+1)$.

$$\begin{aligned}\Sigma_{k=1}^{n+1} k &= \Sigma_{k=1}^{n} k + (n+1) \\ &= \tfrac{n(n+1)}{2} + (n+1) \text{[Induction hypothesis]} \\ &= \tfrac{n(n+1)+2(n+1)}{2} \\ &= \tfrac{(n+1)(n+2)}{2}\end{aligned}$$

Thus, $P(n)$ implies $P(n+1)$, so the result is true by induction.



**Notes:**

- You can write $\overset{P(n)}{=}$ instead of writing "Induction hypothesis" at the end of the line, or you can write "$P(n)$" at the end of the line.
  - Whatever you write, make sure it's clear when you're applying the induction hypothesis
- Notice how we rewrite $\Sigma_{k=1}^{n+1} k$ so as to be able to appeal to the induction hypothesis. This is standard operating procedure.

## Another example

**Theorem:** $(1+x)^n \geq 1+nx$ for all nonnegative integers $n$ and all $x \geq 0$.

**Proof:** By induction on $n$. Let $P(n)$ be the statement $(1+x)^n \geq 1+nx$.

*Basis:* $P(0)$ says $(1+x)^0 \geq 1$. This is clearly true.

*Inductive Step:* Assume $P(n)$. We prove $P(n+1)$.

$$\begin{aligned}(1+x)^{n+1} &= (1+x)^n(1+x)\\ &\geq (1+nx)(1+x) \text{[Induction hypothesis]}\\ &= 1+nx+x+nx^2\\ &= 1+(n+1)x+nx^2\\ &\geq 1+(n+1)x\end{aligned}$$

This argument actuallly works for if $x \geq -1$.

- Why? Why does it fail if $x < -1$?



## Towers of Hanoi

**Theorem:** It takes $2^n - 1$ moves to perform $H(n, r, s)$, for all positive $n$, and all $r, s \in \{1, 2, 3\}$.

**Proof:** Let $P(n)$ be the statement "It takes $2^n - 1$ moves to perform $H(n, r, s)$ and all $r, s \in \{1, 2, 3\}$."

- Note that "for all positive $n$" is not part of $P(n)$!
- $P(n)$ is a statement about a particular $n$.
- If it were part of $P(n)$, what would $P(1)$ be?

*Basis:* $P(1)$ is immediate: robot($r \leftarrow s$) is the only move in $H(1, r, s)$, and $2^1 - 1 = 1$.

*Inductive step:* Assume $P(n)$. To perform $H(n+1, r, s)$, we first do $H(n, r, 6-r-s)$, then robot($r \leftarrow s$), then $H(n, 6-r-s, s)$. Altogether, this takes $2^n - 1 + 1 + 2^n - 1 = 2^{n+1} - 1$ steps.



## A Matching Lower Bound

**Theorem:** Any algorithm to move $n$ rings from pole $r$ to pole $s$ requires at least $2^n - 1$ steps.

**Proof:** By induction, taking the statement of the theorem to be $P(n)$.

*Basis:* Easy: Clearly it requires (at least) 1 step to move 1 ring from pole $r$ to pole $s$.

*Inductive step:* Assume $P(n)$. Suppose you have a sequence of steps to move $n+1$ rings from $r$ to $s$. There's a first time and a last time you move ring $n+1$:

- Let $k$ be the first time
- Let $k'$ be the last time.
- Possibly $k = k'$ (if you only move ring $n+1$ once)

Suppose at step $k$, you move ring $n+1$ from pole $r$ to pole $s'$.

- You can't assume that $s' = s$, although this is optimal.



Key point:

- The top $n$ rings have to be on the third pole, $6-r-s'$
- Otherwise, you couldn't move ring $n+1$ from $r$ to $s'$.

By $P(n)$, it took at least $2^n - 1$ moves to get the top $n$ rings to pole $6-r-s'$.

At step $k'$, the last time you moved ring $n+1$, suppose you moved it from pole $r'$ to $s$ (it has to end up at $s$).

- the other $n$ rings must be on pole $6-r'-s$.
- By $P(n)$, it takes at least $2^n - 1$ moves to get them to ring $s$ (where they have to end up).

So, altogether, there are at least $2(2^n - 1) + 1 = 2^{n+1} - 1$ moves in your sequence:

- at least $2^n - 1$ moves before step $k$
- at least $2^n - 1$ moves after step $k'$
- step $k$ itself.

If course, if $k \neq k'$ (that is, if you move ring $n+1$ more than once) there are even more moves in your sequence.