Math 55 - Spring 04 - Lecture notes # 1 - Jan 20 (Tuesday)

```
  Name, class, URL (www.cs.berkeley.edu/~demmel/ma55) on board
  Head TA Mike West speaks on bureaucracy
  Advertise CS 70 (T Th 2-3:30) as an "honors" alternative to Ma55
  All material on web page and at Copy Central Northside
  Read Course Outline on web page for course rules and grading policies
    You are responsible for reading this and knowing the rules!

  Class outline:
    Basics (Chap 1) - common language for rest of course
       logic (basis of proofs,
              logical operations in programs (CS61)
              hardware design (CS150))
       sets and functions (same proof of 3 results:
            (1) can show that can't write a program to
                implement every possible function,
                because there are more functions than programs;
            (2) can show it is impossible to to
                write a debugger that finds "infinite loops"
                in other programs (CS 172))
            (3) there are more real numbers than rational numbers
                (even though there are infinitely many of both
       Used in most of math curriculum

    Integer algorithms (Chap 2)
       Big-O notation (measure approximately how fast a function f(n)
                       increases as n increases)
       Using Big-O to analyze speed of algorithms (used in CS61, CS170)
       Prime numbers, GCD (Greatest common divisors),
       "modular arithmetic" (a mod b), Chinese remainder theorem
       Applications: generating random numbers
                     integer with "bignums"
                     cryptography (eg RSA used in Netscape)
                     (Ma 113, CS 170)

    More proof techniques (Chap 3)
       induction (knowing when a theorem, or program, is correct)
       (CS 17x, most of math curriculum)

    Probability theory and counting (Chaps 4, 5, Lenstra's notes)
       analyzing, designing algorithms (CS 170, CS 162, CS 188...)
```

```
        (sometimes an algorithm that uses "random numbers" is
         faster than one that does not)
     how many ways to pair up n students into teams of 2 or 3 or ...
     how to gamble in Reno
     algorithms that use random numbers
       EX suppose you build a web search engine
          (or web page to sell books or ...)
          you buy 100 PCs and put them in a room
          when a request comes in, you pick a PC at random, and send it
            to that PC
          what is the average waiting time to service a request?

Read Chapter 1 of book, homework due Wednesday Jan 28
       at the beginning of section
Do problems
      sec 1.1: 6, 16(a,c,e,g), 40, 42, 48, 60
          1.2: 4a, 8(a,b), 10 (for 8a and 8b), 20, 38 (assume results of 36, 37)
          1.3: 8, 14, 26(a,c,e), 56(a,b)
          1.4: 4, 8(a,c,e), 28(a,c,e,g,i)
          1.5: 6, 8(a,c,e), 22, 52

  Begin course:

          DEF: Proposition is a statement that must be either
                     true (T) or false (F)
           EG: 2+2 = 4  (Y)
               2+2 = 3  (Y)
               Is 2+2=4? (N)
               Let x = 3. (N)
               x+1 = 2   (N, unless we know the value of x)
               Every even number > 2 is the sum of two primes. (Y)
          Notation: propositions denoted p,q,r ...
          DEF: not p , p or q (disjunction), p and q (conjunction),
               p xor q (exclusive or)
          Truth tables
          DEF: a compound proposition is formed by simple propositions
               connected by logical operators and parentheses;
           EG: (p and q) or (r and s)
               programming example below
          DEF: p -> q, "if p, then q", "p implies q", "p sufficient for q",
                "q necessary for p";  p -> q same as not(p) or q;
            EG: if it is sunny, then we go to the beach
```

```
              p = it is sunny, q = we go to the beach
              (T unless it is sunny (p)  and we don't go (not q))
            Truth table
ASK&WAIT  EG: if (today is Friday), then (2+3 = 6);
              on which days is this true?
              p = (today is Friday), q = (2+3=6)
              T every day except Friday
ASK&WAIT  EG: if (1+1=3) then (2+2=4)
          Note that p and q need not have anything to do with one another
          in order to form the expression p -> q
        DEF: The converse of p -> q is q -> p; they need not be true
             at the same time
ASK&WAIT  EG: If I am exactly 18 years old then I am allowed to vote;
              converse is
              if I am allowed to vote, then I am exactly 18 years old
              note that converse not necessarily true if original is
        DEF: The contrapositive of p -> q is not q -> not p;
             these are the same
ASK&WAIT  EG: If I am exactly 18 years old then I am allowed to vote;
              converse is
              if I am not allowed to vote, then
                     I am not exactly 18 years old
              contrapositive is true
        Truth table proof of prop <=> contrapositive
        DEF: p <-> q "p if and only q",  "p->q and q->p", biconditional

Application to programming (C or C++ syntax)

        if (compound proposition, or logical expression) then do something...
        if ((n>0) && (m<k)) then do something ...
           (n>0) and (m<k) are propositions whose value (T or F)
           can be evaluated when you get to this line in the program,
           and && means "and", || means "or", etc.

        in computer 1 represents true and 0 false (representation details
           in CS61C, CS150)

Application to Web search
        In www.google.com you can type into the Advanced Search form:
             With all the words: guano
             and With at least one of the words: geometry tickle
        Same as "find all web pages W for which the proposition is true"
```

```
               (W contains guano) and
                  ((W contains geometry) or (W contains tickle))
ASK&WAIT: what web pages do you get?
            EX: Geometry Learning center page on disease histoplasmosis,
                   caused by fungus growing in bat guano,
                   contains guano and geometry, not tickle
            EX: Magazine article about "tickle down economics"
                   of Christmas toys and a town in Florida
                   that smells like alligator guano
                   contains guano and tickle, not geometry
            EX: poem in on-line poetry magazine Lynx
                   contains all 3
      Now let's try google with
               With all the words: guano
               and With at least one of the words: geometry tickle
               and Without the words the
ASK&WAIT: Same as "find all web pages W for which the proposition is true"
               (W contains guano) and
                  ((W contains geometry) or (W contains tickle)) and
                  (not(W contains the))
ASK&WAIT: what web pages do you get?
            EX: "G" page of an Esperanto dictionary
                  contains guano, geometry, not tickle or the
```

....

```
 Next Goal: simplifying compound propositions or replacing
              an expression depending on p, q, r, ... combined with
              "and", "or", etc.  with another "equivalent" and "simpler"
              one that has the same truth/false value for all values of
              p, q, r, ... (just like when you learned algebra in high school;
              algebra with variables that can only have values True and False
              is called "Boolean Algebra")

          What is "equivalence"?
          DEF: p <-> q means (p -> q) and (q -> p); true if p and q are
               both true or both false (show truth table)
          DEF: propositions p and q are called logically equivalent if
               p <-> q is always true (a tautology); write p <=> q;
          EG:  p <=> not(not(p))
          EG:  p or not(p)  <=> True  (p or  not(p) a "tautology")
          EG:  p and not(p) <=> False (p and not(p) a "contradiction")
```

When is q "simpler" than p? Roughly, if q is a "smaller" compound expression, with fewer operations like and, or etc

Motivation:

1) Proofs: if the compound prop. p equivalent to True (p a tautology), then you have proved a theorem
2) In programming, simple expressions (in "if" statements) are easier to understand and debug, faster to evaluate, since each "and" or "or" costs an operation

3) In computer design: A computer stores and computes with "bits" which are either 1 (true) or 0 (false), represented by electrical signals. The computations are done with little pieces of hardware that, say, take two input bits and compute their "and" to get an output bit. Each little piece of hardware corresponds to one of the operations "and", "or", etc. that we have talked about, and so computer hardware is just evaluating compound propositions. A "simpler" compound proposition needs fewer pieces of hardware to evaluate it, which is smaller, cheaper, faster... See also CS150, Chap 10, other "CAD" courses in EECS, and whole industries.

Rules we can use to simplify compound propositions:

```
          EG:  True or q <=> True        (domination)
ASK&WAIT  EG:  (I am a student at Stanford) or (2+2=4) <=> True
          EG:  False and q <=> False        (domination)
          EG:  p or q <=> q or p (commutativity)
          EG:  ( p or q ) or r <=> p or ( q or r ) (associativity)
          EG:  DeMorgan's laws
               not(p or q) <=> (not p) and (not q)
               not(p and q) <=> (not p) or (not q)
               proof: Truth table
ASK&WAIT  EG:  Prove that not(p -> q) -> p a tautology, i.e. <=> True
               not(p -> q) -> p
               not(not(p) or q) -> p            Def ->
               not(not(not(p) or q)) or p       Def ->
               (not(p) or q) or p               Double negative
                not(p) or (q or p)              associativity
                not(p) or (p or q)              commutativity
               (not(p) or p) or q               associativity
               (True       ) or q               not(p) or p <=> True
                True                            true or q <=> true
             Like musical instrument: practice
```

Also: Truth Table (Ref: EE290H - logic verification, 150 vars)
EG: p or (q and r) <=> (p or q) and (p or q) distributivity
EG: p and (q or r) <=> (p and q) or (p and q) distributivity
(notice that names of rules -- distributivity etc -- are
same as rules for regular algebra with numbers, since the rules
are analogous)

How does one interpret an expression like not p or q?
as (not p) or q, or as not(p or q), which are quite different
The precedence for operators in absence of parentheses is
Highest to Lowest: not, (and, or), (-> , <->)
If there is ambiguity, I will use parentheses