Pre-class quiz #6 is due Monday October 7th at 17:00.

**Assigned reading for the quiz:**
- Epp, 4th edition: 3.2, 3.4
- Epp, 3rd edition: 2.2, 2.4
- Rosen, 6th edition: 1.3, 1.4
- Rosen, 7th edition: 1.4, 1.5

Assignment #2 is due Friday October 4th at 16:00.

---

By the start of class, you should be able to
- Evaluate the truth of predicates applied to particular values.
- Show a predicate logic statement is true by enumerating examples, i.e. one (all) in the domain for an existential (universal) quantifier.
- Show a predicate logic statement is false by enumerating counterexamples, i.e. all (one) in the domain for an existential (universal) quantifier.
- Translate between statements in formal predicate logic notation and equivalent statements in closely matching informal language, i.e., informal statements with clear and explicitly stated quantifiers.
Quiz 5 feedback:
- Very well done, once again.
- Question (avg: 5.87/10):
  Let D be the domain of 8-bit signed binary numbers, not mathematical integers.
  Is the following statement true?
  \( \forall x \in D, \forall y \in D, ((x > 0) \land (y > 0)) \Rightarrow (x + y) > 0 \)
- We will discuss the open-ended question on what it means for an algorithm to be faster/slower than another one next week.

CPSC 121: the BIG questions:
- How can we convince ourselves that an algorithm does what it's supposed to do?
  - We need to prove that it works.
  - We have done a few proofs in the last week or so.
  - Now we will learn how to decide if a proof is valid in a formal setting.
- How do we determine whether or not one algorithm is better than another one?
  - We can finally answer that question!

By the end of this module, you should be able to:
- Build statements about the relationships between properties of various objects using predicate logic. These may be:
  - real-world like “every candidate got votes from at least two people in every province” or
  - computing related like “on the \( i \)th repetition of this algorithm, the variable \( \text{min} \) contains the smallest element in the list between element 0 and element \( i \)”. 

Module Summary
- Predicates vs Propositions
- Examples
- More examples: sorted lists
- Algorithm efficiency revisited.
- Additional examples to consider.
Module 5: Predicate Logic

- Is Propositional Logic a complete model? Can it be used to model every real-world situation?
- Which of the following can it model effectively?
  a) Relationships among factory production lines (e.g., “wheel assembly” and “frame welding” both feed the undercarriage line).
  b) Defining what it means for a number to be prime.
  c) Generalizing from examples to abstract patterns like “everyone takes off their shoes at airport security”.
  d) It can model all of these effectively.
  e) It can not model any of these effectively.

Module 5: Predicate Logic

- What is predicate logic good for modeling?
  - Relationships among real-world objects.
  - Generalizations about patterns
  - Infinite domains
  - Generally, problems where the properties of the different concepts, or parts, depend on each other.

Module 5: Predicate Logic

- Examples of predicate logic use:
  - **Data structures**: Every key stored in the left subtree of a node N is smaller than the key stored at N.
  - **Language definition**: No path via references exists from any variable in scope to any memory location available for garbage collection...
  - **Databases**: the relational model is based on predicate logic.
  - **Algorithms**: in the worst case, every comparison sort requires at least \( cn \log_2 n \) comparisons to sort \( n \) values, for some constant \( c > 0 \).

Module 5: Predicate Logic

- Quantifiers scope:
  - A quantifier applies to everything to its right, up to the closing parenthesis of the () pair that “contains” it.
  - Example:
    \[
    \forall x \in D, (\exists y \in E, Q(x,y) \Rightarrow \forall z \in F, R(y,z)) \land P(x)
    \]
Module 5: Predicate Logic

- Quantifiers scope (continued)
  - Which of the following placements of parentheses yields the same meaning as:
    \[ \forall x \in \mathbb{Z}, \exists y \in \mathbb{Z}, x < y \land \text{Even}(y) \] ?
    a) \((\forall)x \in \mathbb{Z}, \exists y \in \mathbb{Z}, x < y \land \text{Even}(y))\)
    b) \((\forall)x \in \mathbb{Z}, \exists y \in \mathbb{Z}, x < y \land \text{Even}(y))\)
    c) \((\forall x \in \mathbb{Z}), \exists y \in \mathbb{Z}, x < y \land \text{Even}(y))\)
    d) \((\forall x \in \mathbb{Z}, \exists y \in \mathbb{Z}, x < y) \land \text{Even}(y))\)
    e) \((\forall x \in \mathbb{Z}, \exists y \in \mathbb{Z}, x < y \land \text{Even}(y))\)

Module 5: Predicate Logic

- Negation scope:
  - Which of the following placements of parentheses yields the same meaning as:
    \[ \sim \exists x \in \mathbb{Z}^+, \forall y \in \mathbb{Z}^+, x < y \land \text{Even}(y) \] ?
    a) \((\sim)\exists x \in \mathbb{Z}^+, \forall y \in \mathbb{Z}^+, x < y \land \text{Even}(y))\)
    b) \((\sim(\exists)) \in \mathbb{Z}^+, \forall y \in \mathbb{Z}^+, x < y \land \text{Even}(y))\)
    c) \((\sim(\exists x \in \mathbb{Z}^+)), \forall y \in \mathbb{Z}^+, x < y \land \text{Even}(y))\)
    d) \((\sim(\exists x \in \mathbb{Z}^+, \forall y \in \mathbb{Z}^+, x < y)) \land \text{Even}(y))\)
    e) \((\sim(\exists x \in \mathbb{Z}^+, \forall y \in \mathbb{Z}^+, x < y \land \text{Even}(y)))\)

Module 5: Predicate Logic

- What is the difference between a **proposition** and a **predicate**?
  a) A predicate may contain one or more quantifiers, but a proposition never does.
  b) A proposition's name is a lowercase letter, whereas a predicate's name is an uppercase letter.
  c) A predicate may contain unbound variables, but a proposition does not.
  d) They are the same thing, using different names.
  e) None of the above.

Module 5: Predicate Logic

- Which variables does this formula's truth depend on?
  \[ \forall i \in \mathbb{Z}^+, (i \geq n) \leftrightarrow \sim \exists v \in \mathbb{Z}^+, \text{HasValue}(l, i, v) \] ?
  a) \(i\) and \(v\)
  b) \(l\) and \(n\)
  c) \(n\) and \(v\)
  d) \(i\) and \(n\)
  e) None of these are correct.
Module 5: Predicate Logic

- Module Summary
  - Predicates vs Propositions
  - Examples
  - More examples: sorted lists
  - Algorithm efficiency revisited.
  - Additional examples to consider.

Given the definitions:
- \( F(x) \): the set of foods.
- \( E(x) \): Alice eats food \( x \).
- \( g \): Alice grows.
- \( s \): Alice shrinks.

Express these statements using predicate logic:
- Eating food causes Alice to grow or shrink.
- Alice shrunk when she ate some food.

Given the definitions:
- \( F(x) \): \( x \) is a fierce creature.
- \( L(x) \): \( x \) is a lion
- \( C(x) \): \( x \) drinks coffee
- \( D \): the set of all creatures.
- \( T(x,y) \): creature \( x \) has “tasted” creature \( y \).
Module 5: Predicate Logic

- Express these statements using predicate logic:
  - All lions are fierce.
  - Some lions do not drink coffee.

- Consider the statement
  All fierce creatures are not lions

- Give two different (not logically equivalent) translations into predicate logic.
- Why did we end up with two translations?

Module Summary

- Predicates vs Propositions
- Examples
- More examples: sorted lists
- Algorithm efficiency revisited.
- Additional examples to consider.
Module 5: Predicate Logic

- Definitions:
  - Assume that \( L \) represents a list of values.
  - The length of \( L \) is denoted by \((\text{length } L)\).
  - The \( i \)th element of \( L \) is denoted by \((\text{list-ref } L \ i)\).
    - The first element of \( L \) is \((\text{list-ref } L \ 0)\).
- Are \text{length} and \text{list-ref} predicates?
  - No: a predicate is a function that returns \text{true} or \text{false}.
  - What do these functions return?
    - \text{length}: an integer.
    - \text{list-ref}: a value whose type depends on the contents of \( L \).

Problem:
- Define a predicate \(\text{Sorted}(L)\) whose value is \text{true} if and only if \( L \) is sorted in non-decreasing order.
- We can use the functions \text{length} and \text{list-ref}.

Assumption:
- The call \((\text{list-ref } L \ i)\) returns an undefined value if \( i \) is negative, or greater than or equal to \((\text{length } L)\).
  - Recall the first element of \( L \) is \((\text{list-ref } L \ 0)\), so \( L \)'s last element is \((\text{list-ref } L \ (- \ (\text{length } L) \ 1))\).

Which of the following is/are a problem with this definition?
- \(\text{Sorted}(L) \equiv \forall i \in \mathbb{N} \ \forall j \in \mathbb{N} \ (\text{list-ref } L \ i) \land \ (\text{list-ref } L \ j) \land \ v_i < v_j\)
  - a) There is no quantifier for \( L \).
  - b) There are no quantifiers for \( v_i \) and \( v_j \).
  - c) We can not use \( \land \) with \((\text{list-ref } L \ i)\) and \((\text{list-ref } L \ j)\)
  - d) Both (a) and (b)
  - e) Both (b) and (c)

Which of the following is a problem with this definition?
- \(\text{Sorted}(L) \equiv \forall i \in \mathbb{N} \ \forall j \in \mathbb{N} \ i < j \rightarrow (\text{list-ref } L \ i) < (\text{list-ref } L \ j)\)
  - a) It is too restrictive (it does not allow for equal values).
  - b) It does not restrict the ranges of \( i \) and \( j \).
  - c) It is missing quantifiers.
  - d) Both (a) and (b)
  - e) Both (b) and (c)
Module 5: Predicate Logic

- How do we modify the attempt on the previous slide to get a working predicate?
  
  \[ \text{Sorted}(L) \equiv \forall i \in \mathbb{N} \forall j \in \mathbb{N} \quad i < j \rightarrow (\text{list-ref} L i) < (\text{list-ref} L j) \]

Module 5: Predicate Logic

- There exists means there is at least one.
- How do we write there is exactly one?
  
  - lists have exactly one element at each valid index.

Definitions:
  
  - There is exactly one \( \equiv \) There is at least one \( \land \) There is at most one.
  - There is at most one with property \( P \) \( \equiv \forall x \in D, \forall y \in D, P(x) \land P(y) \rightarrow x = y. \)
  - There are at least two \( \equiv \exists x \in D, \exists y \in D, x \neq y \land P(x) \land P(y). \)

Module 5: Predicate Logic

- Soon we will use English more often than we will write every predicate explicitly using logic.
- However the ability to use predicate logic will help us think things through and not overlook minor (but important) details.

  "when we become comfortable with formal manipulations, we can use them to check our intuition, and then we can use our intuition to check our formal manipulations." -- Epp, (4th ed), p. 127
What does it mean for one algorithm to be generally faster than another algorithm?

Here are some answers we saw on the quiz:

- One must consider the use of the two algorithms, because the speed for each algorithm may differ depending on what they are being used to determine. Also, if the two algorithms are running on different machines, one cannot compare the speeds with enough accuracy to determine a definitely which algorithm may be faster.
- For an algorithm to be generally faster I would expect it to be faster for more complicated and complex cases. This is because in simpler cases like looking for a person with the last name starting with A or Z you can just do that yourself by hand. To me, computers are meant to be used in more complicated scenarios that would take a lot of time for humans to do by themselves.

More responses:

- If Alg A is "generally faster" than Alg B (as suggested by the informal graph given in the problem statement), then as the problem size grows larger, the amount of time required to execute the algorithm does not increase without bound.
- Then, it's the longer term behaviour of the algorithm that should count in a general judgement. This could be determined using the mathematics of functions and set theory given an appropriate model of the computational system.
- For one algorithm to be generally faster than the other, it would have to be more efficient and stable. In the example above, we see that Algorithm B looks like it is faster, but past a certain problem size, the time it takes will increase exponentially, soon surpassing the time it would have taken for Algorithm A to do the same thing.

Consider the following problem:

- Given: a sorted list of names with phone numbers.
- Wanted: the phone number for a given name \( N \).
- Which algorithm is generally faster?
  - Algorithm \( L \): check the first name. If it's not \( N \), then check the second name. Then the third name, etc.
  - Algorithm \( B \): check the name in the middle of the list. If \( N \) comes earlier alphabetically then search the first half of the list using \( B \). If it comes later search the second half of the list instead. Repeat until you have found \( N \) or you're looking at an empty sublist.

Assumptions:

- Reading the name after the current name takes 1s on average.
- Reading a name given its position takes 10s on average.
- For a list with 15 names:
  - Algorithm \( L \) takes \( 15 \times 1s = 15s \) in the worst case.
  - Algorithm \( B \) takes \( 5 \times 10s = 50s \) in the worst case.
Module 5: Predicate Logic

- For a list with 63 names:
  - Algorithm L takes $63 \times 1s = 1m 3s$ in the worst case.
  - Algorithm B takes $7 \times 10s = 1m 10s$ in the worst case.

- For a list with 1048575 names:
  - Algorithm L takes $1048575 \times 1s = 12d 3h 16m 15s$ in the worst case.
  - Algorithm B takes $21 \times 10s = 3m 30s$ in the worst case.

Module 5: Predicate Logic

- How do we determine whether or not an algorithm is generally faster than another?
  - We want to measure how good the algorithm is, in a way that does not depend on
    - the programming language used to implement it.
    - the quality of the compiler or interpreter.
    - the speed of the computer it is executed on.
  - One idea is to count the number of elementary steps of the algorithm as a function of the size of its input $n$.
    - A step is anything that can be computed in constant time, that is, independent from $n$.

Module 5: Predicate Logic

- Is an algorithm with $3n$ steps faster than one with $6n$ steps?
  a) Yes, always.
  b) No, never.
  c) Sometimes.
  d) None of the above.

Module 5: Predicate Logic

- Example:
  - One algorithm performs $6n$ steps of the following type (only the first 6 are written):
    
    \[
    \begin{align*}
    3 + 8 & \quad 2 + 4 & \quad 6 + 9 \\
    2 + 11 & \quad 5 + 6 & \quad 7 + 1
    \end{align*}
    \]
  - The other algorithm performs $3n$ steps of the following type (only the first 3 are written):
    
    \[
    \begin{align*}
    \int_2^5 \frac{x^4-x^2}{2} \, dx & \quad \int_1^6 x \cos(x) + \sin(\pi) \, dx & \quad \int_1^\infty \frac{1}{x^3} \, dx
    \end{align*}
    \]
  - Which one is faster?
Module 5: Predicate Logic

- Facts about execution times:
  - we cannot rely on the values of the constants in front of the functions describing the number of steps.
  - it's almost impossible to compute the number of steps exactly.
- So we want to come up with
  - a way to count steps that ignores these constants.
  - an approximation of the correct number of steps.

Terminology: an algorithm runs in \( O(g) \) time, stated “big-Oh of g time”, if it performs at most \( g(n) \) steps (approximately).

Examples:
- Algorithm L runs in \( O(n) \) time.
- Algorithm B runs in \( O(\log_2 n) \) time.
- The algorithm we used to order students by date of birth runs in \( O(n^2) \) time.
- Let's see how we can define \( O \) using quantifiers.

Which of the following predicates says that the number of steps \( f(n) \) is (approximately) at most \( n^2 \)?

- a) \( \forall c \in \mathbb{R}^+ \forall n \in \mathbb{N} \ f(n) \leq cn^2 \)
- b) \( \exists c \in \mathbb{R}^+ \exists n \in \mathbb{N} \ f(n) \leq cn^2 \)
- c) \( \exists c \in \mathbb{R}^+ \forall n \in \mathbb{N} \ f(n) \leq cn^2 \)
- d) \( \forall c \in \mathbb{R}^+ \exists n \in \mathbb{N} \ f(n) \leq cn^2 \)
- e) None of the above.

For which of the following functions \( f(n) \) is the predicate from the previous slide true?

- a) \( f(n) = n \)
- b) \( f(n) = n^2/2 \)
- c) \( f(n) = 3n^2 \)
- d) \( f(n) = 2^n \)
- e) Both of (a) and (b) only
- f) All of (a), (b), (c)
- g) All of (a), (b), (c) and (d).
Module 5: Predicate Logic

- Which of the following two functions grows faster?
  a) \( f(n) = n \)
  b) \( f(n) = n \log_2 n \)
  c) Neither; they both grow equally fast.

- Is the following predicate true for \( f(n) = n \)?
  \[ \exists c \in \mathbb{R}^+ \ \forall n \in \mathbb{N} \ \ n \geq n_0 \rightarrow f(n) \leq cn \log_2 n \]
  a) Yes
  b) No

So we define \( O(g) \) by:
\[ f \text{ is in } O(g) \text{ if } \exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \forall n \in \mathbb{N} \ n \geq n_0 \rightarrow f(n) \leq cg(n) \]
Module 5: Predicate Logic

- Some common running times:

- Revisiting sorted lists:
  - Recall
    \[
    \text{Sorted}(L) \equiv \forall i \in N \forall j \in N \quad (0 \leq i)^\wedge (i < j)^\wedge (j < (\text{length } L)) \rightarrow (\text{list-ref } L i) \leq (\text{list-ref } L j)
    \]
  - If we verify that L is sorted using this definition, how many comparisons will we need?
  
  - Can we do better?

- Here is another definition:
  \[
  \text{Sorted}(L) \equiv \forall i \in N \quad (0 \leq i)^\wedge (i < (\text{length } L) - 1) \rightarrow (\text{list-ref } L i) \leq (\text{list-ref } L i+1)
  \]
  - These two definitions are logically equivalent.
  - If we verify that L is sorted using this definition, how many comparisons will we need?

- Module Summary
  - Predicates vs Propositions
  - Examples
  - More examples: sorted lists
  - Algorithm efficiency revisited
  - Additional examples to consider
Module 5: Predicate Logic

- Specifying the behaviour of a function/method that takes a list \( l \) and a value \( x \):
  - Translate “returns true if and only if either \( l \) and \( x \) are both equal to \text{null}, or \( l \) contains at least one element \( e \) that is equal to \( x \).

- Define a predicate \( \text{Prime}(x) \) that evaluates to true if and only if \( x \) is a prime. Assume that you have a predicate \( | \) such that \( x | y \) is true if and only if \( x \) divides \( y \) (that is, \( y/x \) is an integer).