Announcements

- You are assigned a **seating zone**
  - Make note of the DCC 308 map and
  - Make note of your zone assignment
  - Seating in the wrong zone may incur penalty
- If you have DSS accommodations, you should have received an email from Meredith
  - If you haven’t, contact us as soon as you can
- Crib sheet: one standard sheet filled on one or both sides. We’ll collect the sheets

Exam 1 Topics

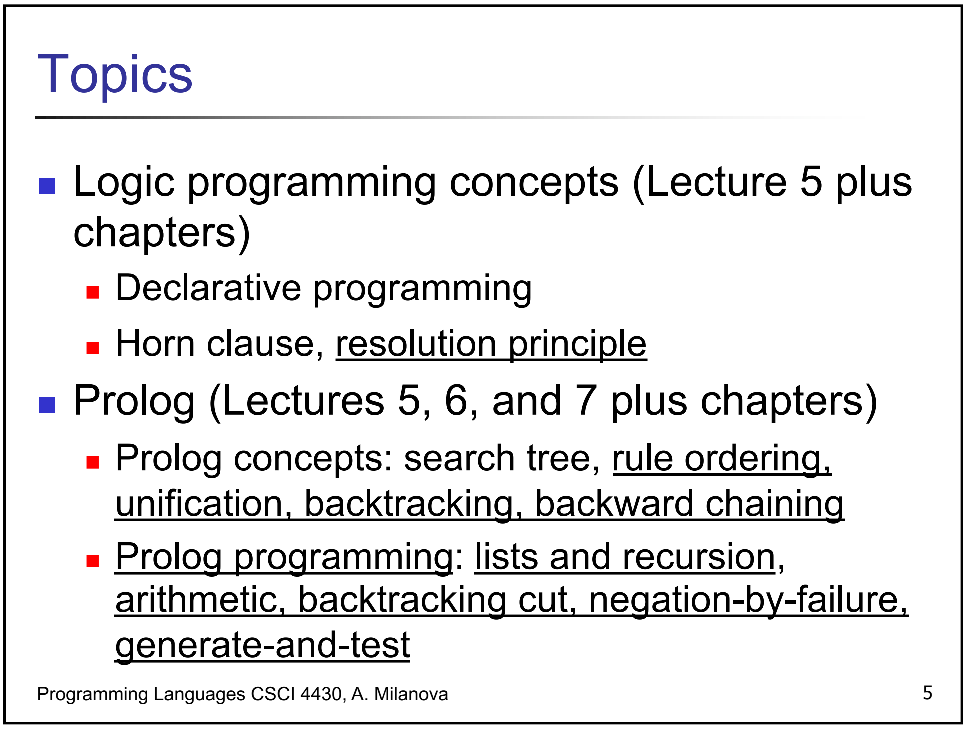
Topics

- Formal languages (Lecture 2 plus chapters)
  - Regular languages
    - Regular expressions
    - DFAs
    - Use of regular languages in programming languages
  - Context-free languages
    - Context-free grammars
    - Derivation, parse, ambiguity
    - Use of CFGs in programming languages
    - Expression grammars, precedence, and associativity

Topics

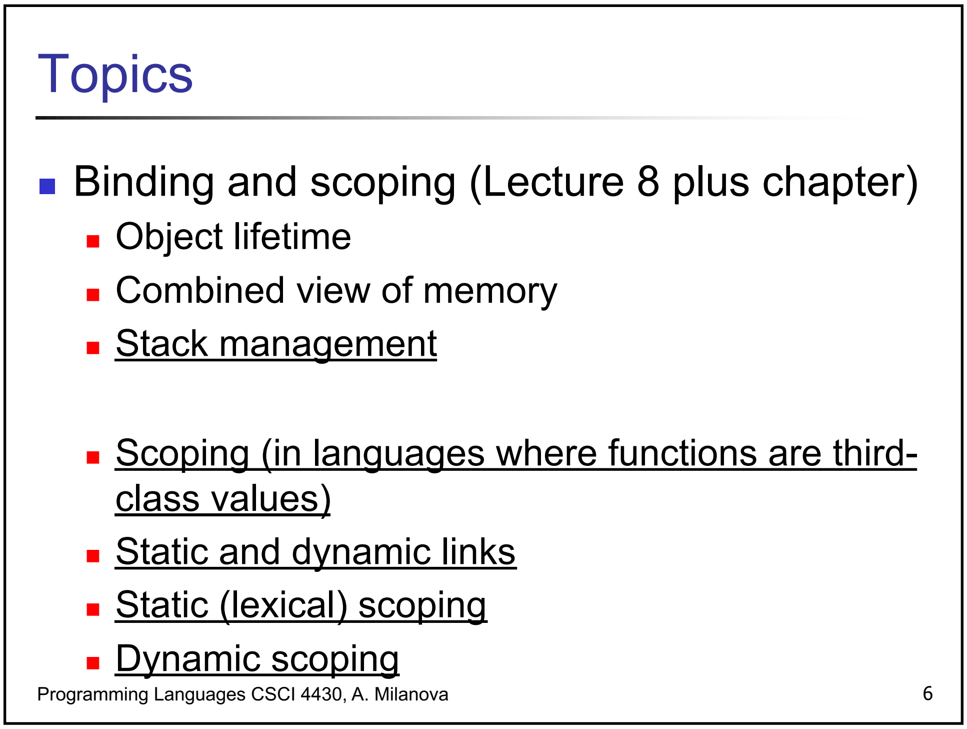
- Parsing (Lecture 3 plus chapters)

- LL Parsing (Lectures 3 and 4 plus chapters)
  - Recursive-descent parsing, recursive-descent routines
  - LL(1) grammars
  - LL(1) parsing tables
  - FIRST, FOLLOW, PREDICT
  - LL(1) conflicts



Topics

- Logic programming concepts (Lecture 5 plus chapters)
  - Declarative programming
  - Horn clause, resolution principle
- Prolog (Lectures 5, 6, and 7 plus chapters)
  - Prolog concepts: search tree, rule ordering, unification, backtracking, backward chaining
  - Prolog programming: lists and recursion, arithmetic, backtracking cut, negation-by-failure, generate-and-test



Topics

- Binding and scoping (Lecture 8 plus chapter)
  - Object lifetime
  - Combined view of memory
  - Stack management

  - Scoping (in languages where functions are third-class values)
  - Static and dynamic links
  - Static (lexical) scoping
  - Dynamic scoping

Topics

- Attribute grammars (Lectures 9 and 10 plus Chapters)
  - Attributes
  - Attribute rules
  - Decorated parse trees
  - Bottom-up (i.e., S-attributed) grammars
  - Top-down (i.e., L-attributed) grammars

Questions on cut (!)

! is a subgoal that always succeeds, but with a side effect: it commits the interpreter to the bindings made since unification of parent goal.

Think of it this way:

```
(1)  p(X) :- q(X), !, r(...).
(2)  p(X) :- ... sth else
```

If q(x) then r(...) else ... sth else.

When code calls p(C), Prolog first tries (1). If q(x) is true it "runs" r(...). But if r(...) fails, then p(c) fails! This is because the interpreter is committed to (1) and to the bindings that made q(X) true, and cannot try other q(x) or (2). If q(x) fails, then the interpeter tries (2) and does ... sth else.

Quiz 1

Assume implicit Start → expr $$

Question 1. (2pts) Consider the expression grammar below.

$$expr \to expr \texttt{ x } expr \mid expr \texttt{ \# } expr \mid \texttt{id}$$

How many parse trees are there for string `id x id # id x id`?

(a) 0
(b) 1
(c) 2
[✓ (d) 5]

(id) x (id # id x id)

Break expressions into subexpressions

1 * 2 = 2 trees

(id x id) # (id x id)

1 * 1 = 1 tree

(id x id #id) x (id)

2 * 1 = 2 trees

2 + 1 + 2 = 5

Questions on FOLLOW

$A \to \alpha B$ => $FOLLOW(A) \subseteq FOLLOW(B)$

$A \to \alpha B \beta$ => $FOLLOW(A) \subseteq FOLLOW(B)$ if β derives ε.

$A \to \alpha B \beta$ => $FIRST(\beta) \subseteq FOLLOW(B)$

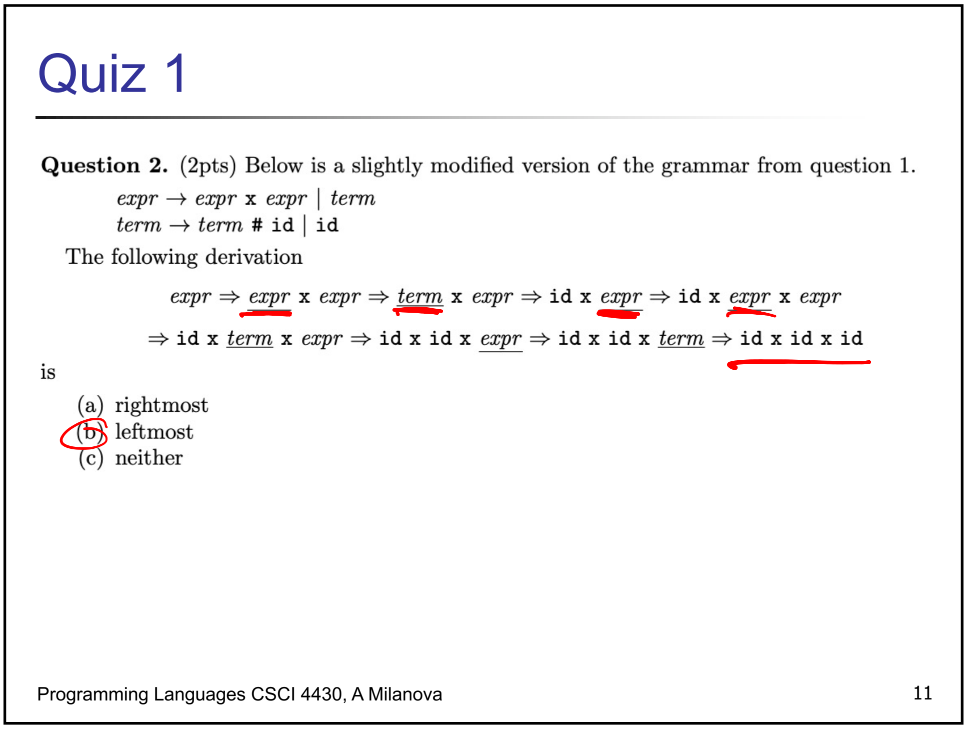
ε cannot be in $FOLLOW(A)$. ε can be in $FIRST(\alpha)$.

$$ can be in $FOLLOW(A)$. $$ cannot be in $FIRST(\alpha)$.

$$PREDICT(A \to \alpha) = \begin{cases} FIRST(\alpha) & \text{if } \alpha \text{ does not derive } \varepsilon \\ (FIRST(\alpha) - \varepsilon) \cup FOLLOW(A) & \text{otherwise} \end{cases}$$

$FIRST(expr) = \{id\}$

$FOLLOW(expr) = \{x, \#, \$\$\}$



Quiz 1

Question 2. (2pts) Below is a slightly modified version of the grammar from question 1.

$expr \rightarrow expr\ \texttt{x}\ expr \mid term$
$term \rightarrow term\ \texttt{\#}\ \texttt{id} \mid \texttt{id}$

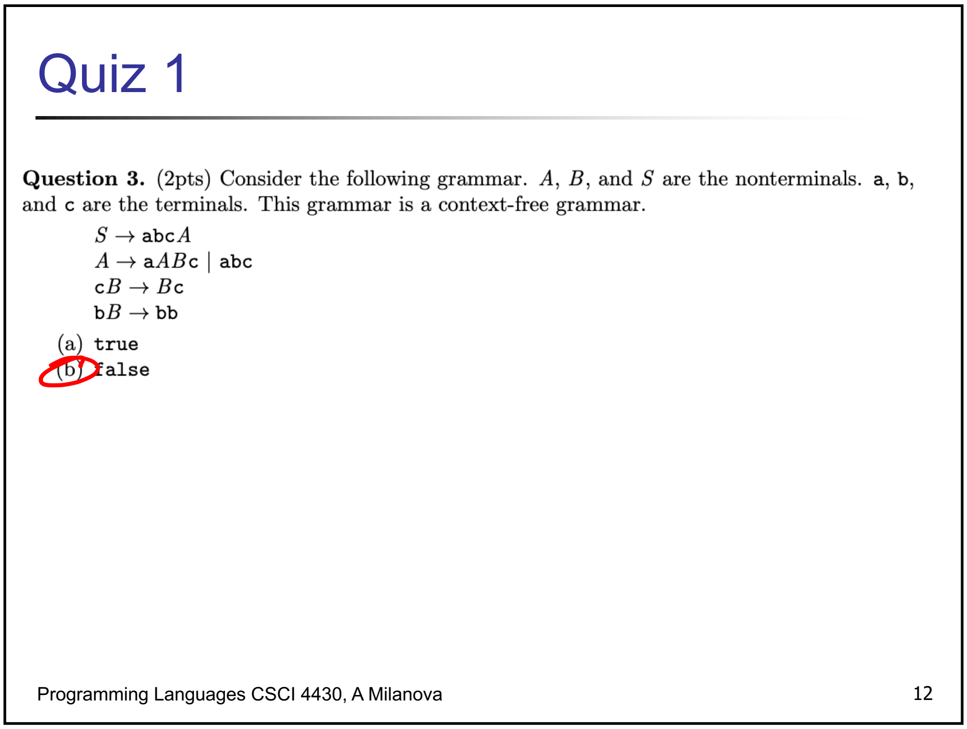
The following derivation

$$expr \Rightarrow \underline{expr}\ \texttt{x}\ expr \Rightarrow \underline{term}\ \texttt{x}\ expr \Rightarrow \texttt{id x}\ \underline{expr} \Rightarrow \texttt{id x}\ \underline{expr}\ \texttt{x}\ expr$$

$$\Rightarrow \texttt{id x}\ \underline{term}\ \texttt{x}\ expr \Rightarrow \texttt{id x id x}\ \underline{expr} \Rightarrow \texttt{id x id x}\ \underline{term} \Rightarrow \underline{\texttt{id x id x id}}$$

is

(a) rightmost
[✓ (b) leftmost]
(c) neither

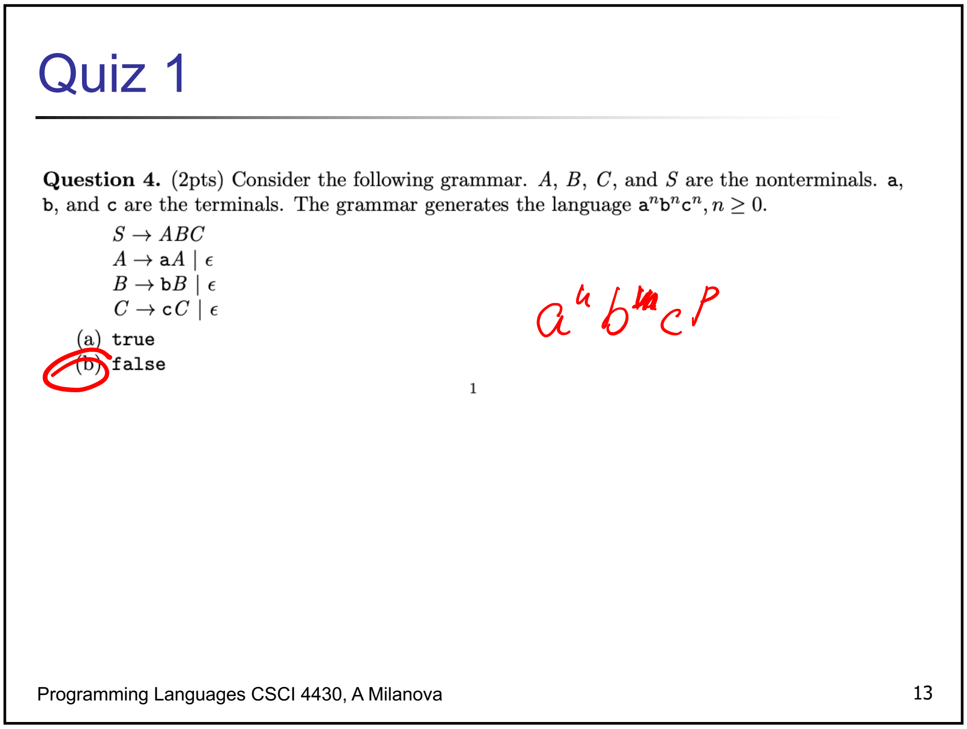


Quiz 1

Question 3. (2pts) Consider the following grammar. A, B, and S are the nonterminals. `a`, `b`, and `c` are the terminals. This grammar is a context-free grammar.

$S \rightarrow \texttt{abc}A$
$A \rightarrow \texttt{a}AB\texttt{c} \mid \texttt{abc}$
$\texttt{c}B \rightarrow B\texttt{c}$
$\texttt{b}B \rightarrow \texttt{bb}$

(a) `true`
[✓ (b) `false`]



Quiz 1

Question 4. (2pts) Consider the following grammar. A, B, C, and S are the nonterminals. a, b, and c are the terminals. The grammar generates the language $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n, n \geq 0$.

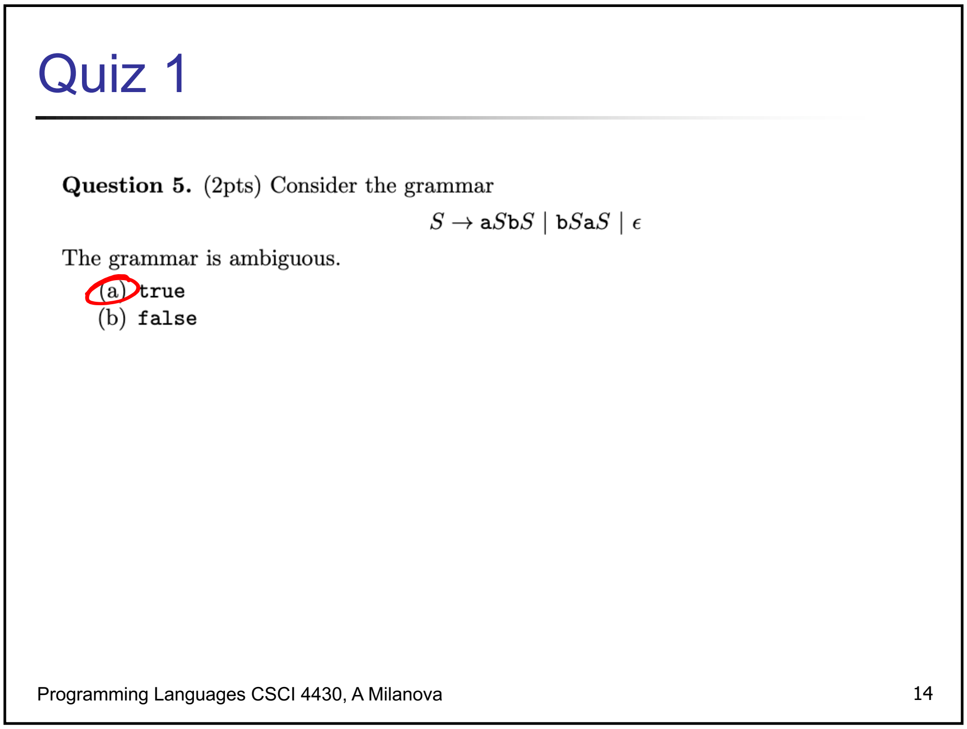
$$S \to ABC$$
$$A \to \mathtt{a}A \mid \epsilon$$
$$B \to \mathtt{b}B \mid \epsilon$$
$$C \to \mathtt{c}C \mid \epsilon$$

$a^n b^m c^p$

(a) true

[✓ (b) false]

1



Quiz 1

Question 5. (2pts) Consider the grammar

$$S \to \mathtt{a}S\mathtt{b}S \mid \mathtt{b}S\mathtt{a}S \mid \epsilon$$

The grammar is ambiguous.

[✓ (a) true]

(b) false

Quiz 2

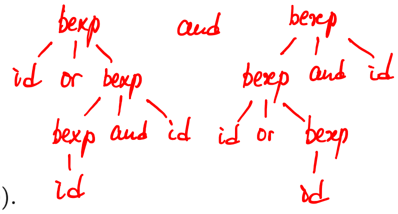
Questions 1-4 refer to the following boolean expression grammar:

start → *bexp* `$$`
bexp → `id` `or` *bexp* | *bexp* `and` `id` | `not` *bexp* | `id`

Question 1. (2pts) The grammar is ambiguous.

[✓ (a)] `true`
(b) `false`

Consider id or id and id :



Question 2. (2pts) The grammar is LL(1).

(a) `true`
[✓ (b)] `false`

Question 3. (1pts) `id` is in the FIRST(*bexp*).

[✓ (a)] `true`
(b) `false`

Question 4. (1pts) `id` is in the FOLLOW(*bexp*).

(a) `true`
[✓ (b)] `false`

FOLLOW(bexp) = { $$, and }

Quiz 2

start → *stmt* **$$**
stmt → **if b then** *stmt else_part* | **a**
else_part → **else** *stmt* | **ε**

Question 4. (2pts) Recall that there is a conflict in LL(1) table entry [*else_part*, `else`] as both *else_part* → `else` *stmt* and *else_part* → ϵ apply on token `else`. (Or in other words, `else` is in the PREDICT set of both productions.) How can you resolve the conflict, so that an `else` would associate with the nearest unmatched `then`?

[✓ (a)] Always expand by *else_part* → `else` *stmt* on `else`. → Forces else to associate with nearest then.
(b) Always expand by *else_part* → ϵ on `else`.

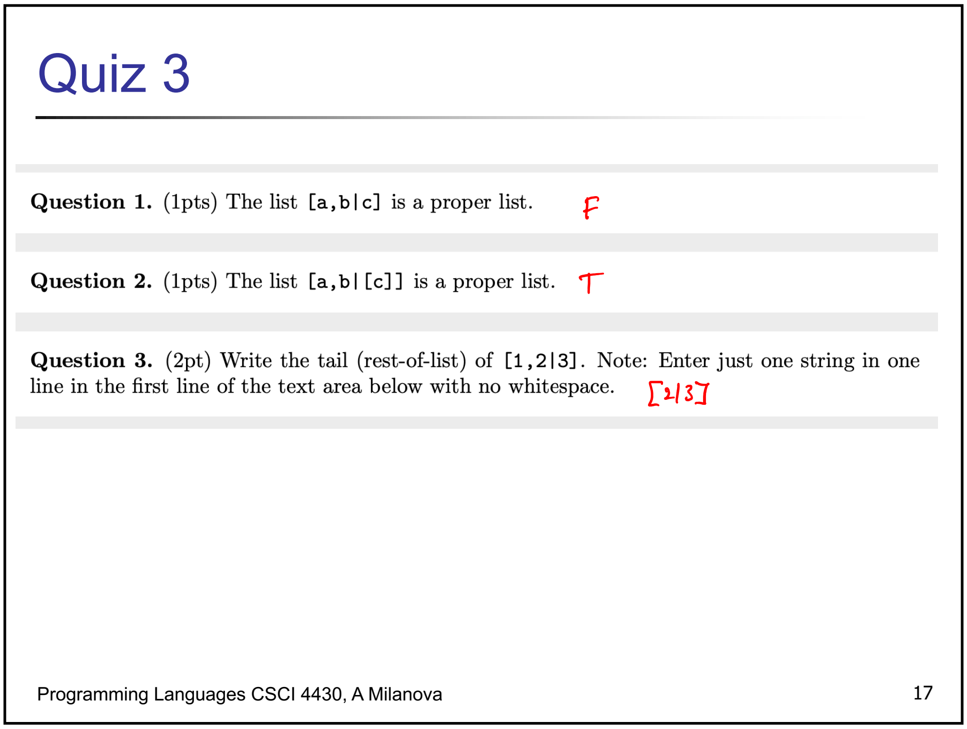
Conflict:

	else
else_part	else_part → else stmt else_part → ε

Question 5. (2pts) There exist unambiguous grammars that are not LL(1) grammars.

[✓ (a)] `true`
(b) `false`

E.g. expr → expr + num | num

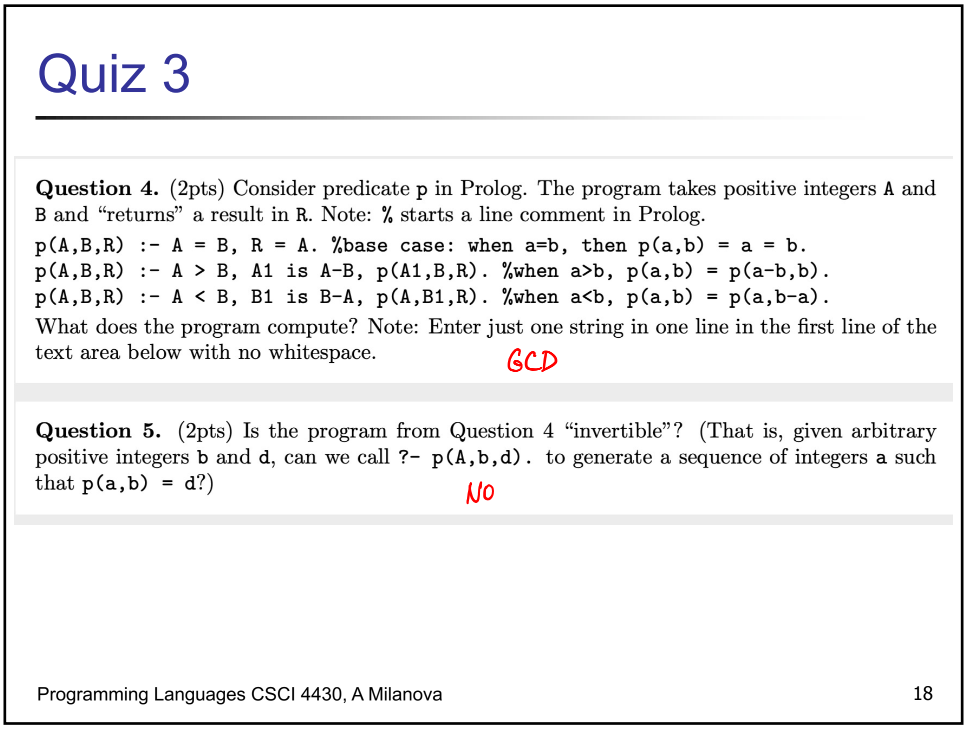


Quiz 3

Question 1. (1pts) The list `[a,b|c]` is a proper list. F

Question 2. (1pts) The list `[a,b|[c]]` is a proper list. T

Question 3. (2pt) Write the tail (rest-of-list) of `[1,2|3]`. Note: Enter just one string in one line in the first line of the text area below with no whitespace. [2|3]



Quiz 3

Question 4. (2pts) Consider predicate `p` in Prolog. The program takes positive integers `A` and `B` and "returns" a result in `R`. Note: `%` starts a line comment in Prolog.

```
p(A,B,R) :- A = B, R = A. %base case: when a=b, then p(a,b) = a = b.
p(A,B,R) :- A > B, A1 is A-B, p(A1,B,R). %when a>b, p(a,b) = p(a-b,b).
p(A,B,R) :- A < B, B1 is B-A, p(A,B1,R). %when a<b, p(a,b) = p(a,b-a).
```

What does the program compute? Note: Enter just one string in one line in the first line of the text area below with no whitespace. GCD

Question 5. (2pts) Is the program from Question 4 "invertible"? (That is, given arbitrary positive integers `b` and `d`, can we call `?- p(A,b,d).` to generate a sequence of integers `a` such that `p(a,b) = d`?) NO

Quiz 3

Question 6. (2pts) Recall our favorite `classmates` Prolog program:

```
takes(jane, his).
takes(jane, cs).
takes(ajit, art).
takes(ajit, cs).
classmates(X,Y) :- takes(X,Z),takes(Y,Z).
```

Query `?- classmates(A,B).` has this many answers (an answer is a pair of bindings `A = ...,` `B = ...`):
Enter just one number on a single line in the first line of the text area below with no whitespace.

6

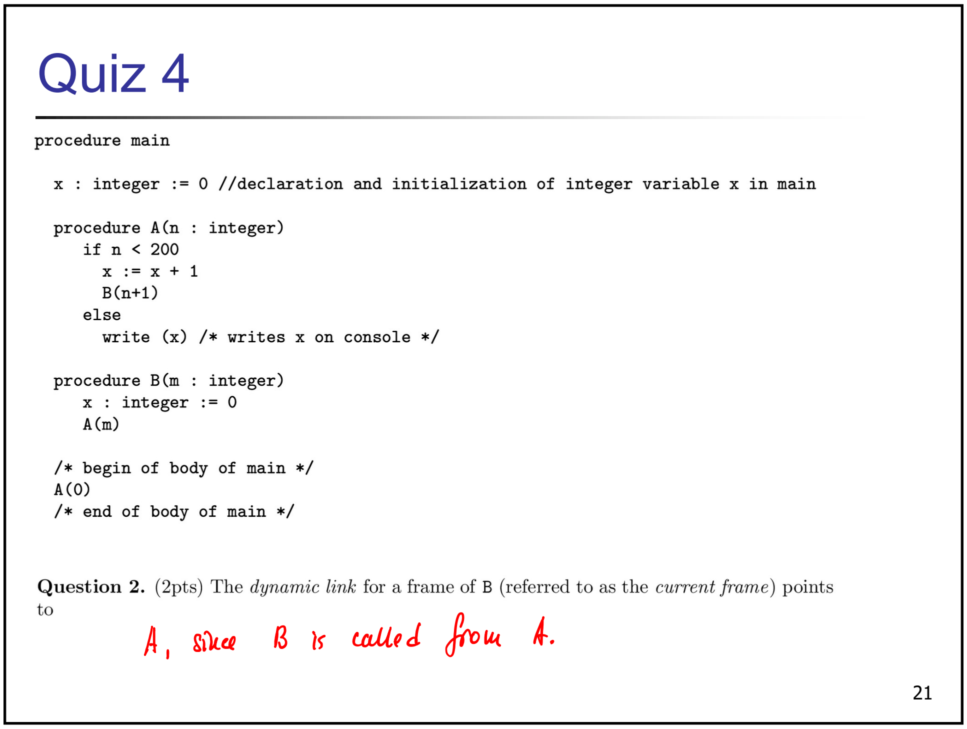
Because there are 6 different bindings of X, Y and Z.

Quiz 4

```
procedure main

  x : integer := 0 //declaration and initialization of integer variable x in main

  procedure A(n : integer)
     if n < 200
       x := x + 1
       B(n+1)
     else
       write (x) /* writes x on console */

  procedure B(m : integer)
     x : integer := 0
     A(m)

  /* begin of body of main */
  A(0)
  /* end of body of main */
```

Question 1. (2pts) The *static link* for a frame of `B` (referred to as the *current frame*) points to

main



Quiz 4

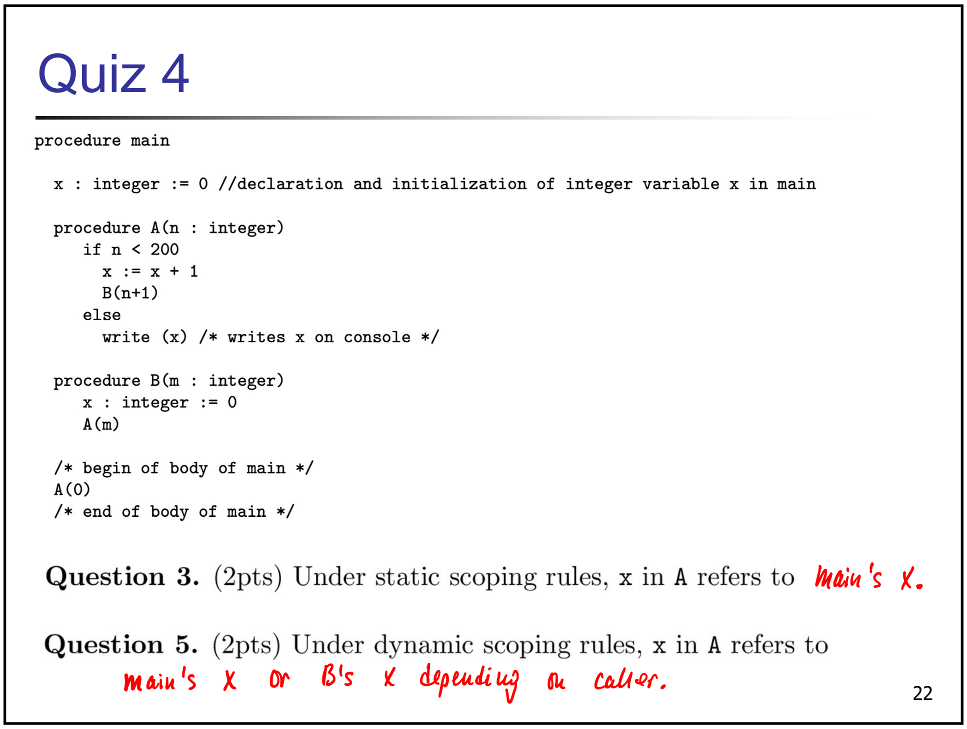
```
procedure main

  x : integer := 0 //declaration and initialization of integer variable x in main

  procedure A(n : integer)
     if n < 200
       x := x + 1
       B(n+1)
     else
       write (x) /* writes x on console */

  procedure B(m : integer)
     x : integer := 0
     A(m)

  /* begin of body of main */
  A(0)
  /* end of body of main */
```

Question 2. (2pts) The *dynamic link* for a frame of B (referred to as the *current frame*) points to

A, since B is called from A.

21

21



Quiz 4

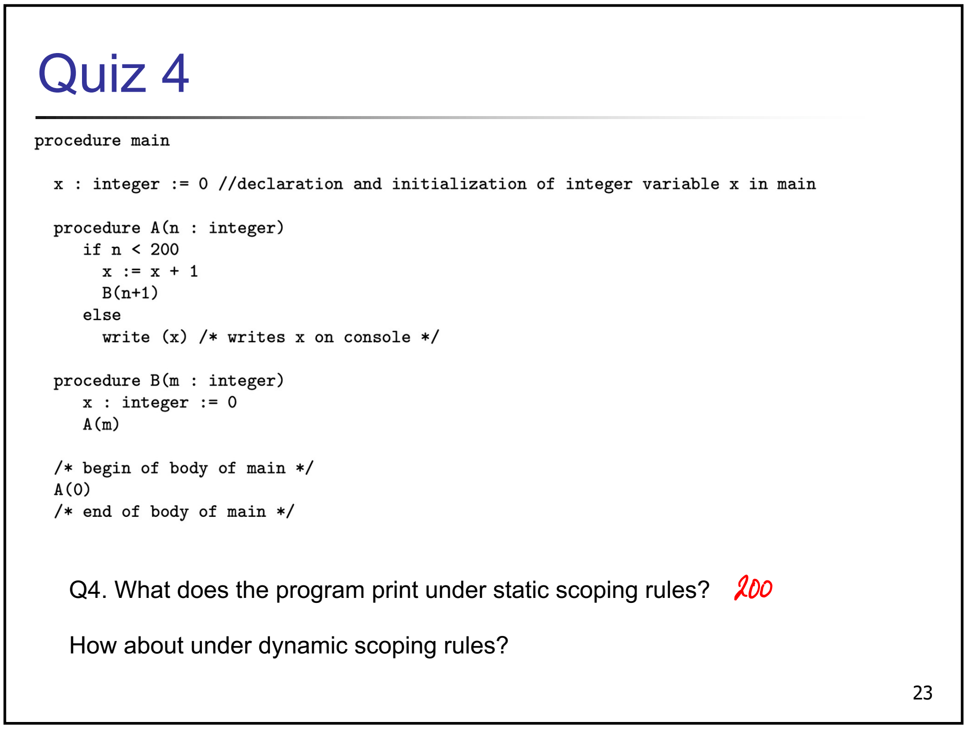
```
procedure main

  x : integer := 0 //declaration and initialization of integer variable x in main

  procedure A(n : integer)
     if n < 200
       x := x + 1
       B(n+1)
     else
       write (x) /* writes x on console */

  procedure B(m : integer)
     x : integer := 0
     A(m)

  /* begin of body of main */
  A(0)
  /* end of body of main */
```

Question 3. (2pts) Under static scoping rules, x in A refers to main's x.

Question 5. (2pts) Under dynamic scoping rules, x in A refers to
main's x or B's x depending on caller.

22

22



Quiz 4

```
procedure main

  x : integer := 0 //declaration and initialization of integer variable x in main

  procedure A(n : integer)
     if n < 200
       x := x + 1
       B(n+1)
     else
       write (x) /* writes x on console */

  procedure B(m : integer)
     x : integer := 0
     A(m)

  /* begin of body of main */
  A(0)
  /* end of body of main */
```

Q4. What does the program print under static scoping rules? 200

How about under dynamic scoping rules?