

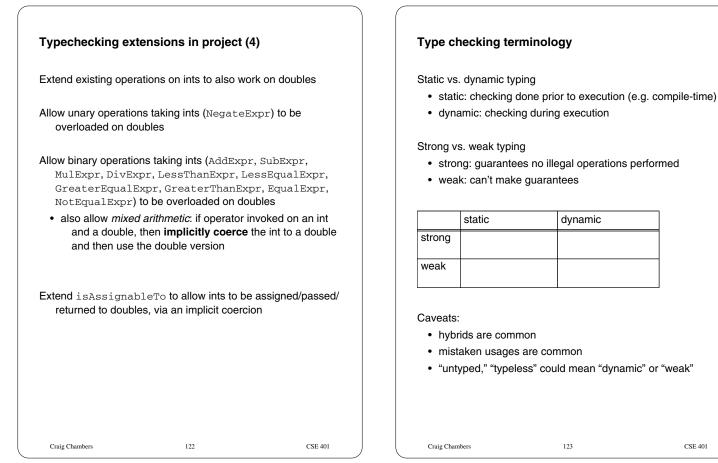
# Typechecking extensions in project (2)

Implement typechecking for new statements and expressions:

- IfStmt
  - · else stmt is optional
- ForStmt
  - · loop index variable must be declared to be an int
  - · initializer & increment expressions must be ints
  - · test expression must be a boolean
- BreakStmt
  - · must be nested in a loop
- DoubleLiteralExpr
- result is double
- OrExpr
  - like AndExpr

# Typechecking extensions in project (3)

- ArrayAssignStmt
  - array expr must be an array
  - index expr must be an int
  - rhs expr must be assignable to array's element type
- ArrayLookupExpr
  - array expr must be an array
  - index expr must be an int
  - · result is array's element type
- ArrayLengthExpr
  - array expr must be an array
- result is int
- ArrayNewExpr
  - length expr must be an int
  - element type must be a legal type
  - result is array of given element type



## Type equivalence

#### When is one type equal to another?

- implemented in MiniJava with
  - ResolvedType.equals(ResolvedType) method

"Obvious" for atomic types like int, boolean, class types

#### What about type "constructors" like arrays?

int[] a1; int[] a2; int[][] a3; boolean[] a4; Rectangle[] a5; Rectangle[] a6;

### Parameterized types in Java 1.5:

List<int> 11; List<int> 12; List<List<int>> 13;

#### In C:

int\* p1; int\* p2; struct {int x; } s1; struct {int x; } s2; typedef struct {int x;} S; S s3; S s4;

CSE 401

"untyped," "typeless" could mean "dynamic" or "weak"

# Name vs. structural equivalence

### Name equivalence:

two types are equal iff they came from the same textual occurrence of a type constructor

- implement with pointer equality of ResolvedType instances
- special case: type synonyms (e.g. typedef) don't define new types
- e.g. class types, struct types in C, datatypes in ML

#### Structural equivalence:

two types are equal iff they have same structure

- · if atomic types, then obvious
- · if type constructors:
  - same constructor
  - · recursively, equivalent arguments to constructor
- implement with recursive implementation of equals, or by canonicalization of types when types created then use pointer equality
- · e.g. atomic types, array types, record types in ML

125

CSE 401

Ту	pe conversions and coercions		Type casts	
In	<ul> <li>In Java, can explicitly convert an object of type double to one of type int</li> <li>can represent as unary operator</li> <li>typecheck, codegen normally</li> <li>In Java, can implicitly coerce an object of type int to one of type double</li> <li>compiler must insert unary conversion operators, based on result of type checking</li> </ul>		<ul> <li>In C and Java, can explicitly cast an object of one</li> <li>sometimes cast means a convers (casts between numeric types)</li> <li>sometimes cast means just a char without doing any computation (casts between pointer types or pointer and numeric types)</li> </ul>	ion
			In C: safety/correctness of casts not ch allows writing low-level code that's more often used to work around li C's static type system	s type-unsafe
			<ul> <li>In Java: downcasts from superclass to run-time type check to preserve typ</li> <li>static typechecker allows the cast</li> <li>codegen introduces run-time chec</li> <li>Java's main form of dynamic type c</li> </ul>	be safety ck
	Craig Chambers 126	CSE 401	Craig Chambers 127	CSE 40