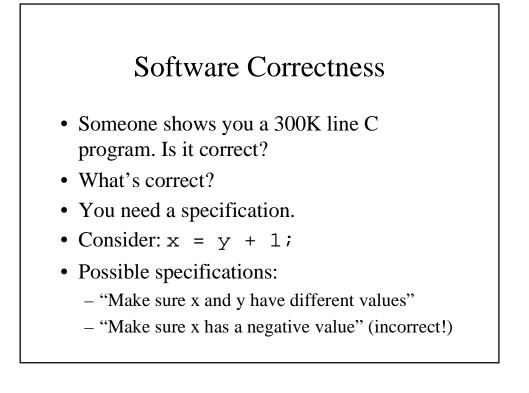
Design by Contract

The Goal

- Ensure the correctness of our software (correctness)
- Recover when it is not correct anyway (robustness)
- Correctness: Assertions
- Robustness: Exception handling
- DBC: Relationship between class and client is a formal agreement

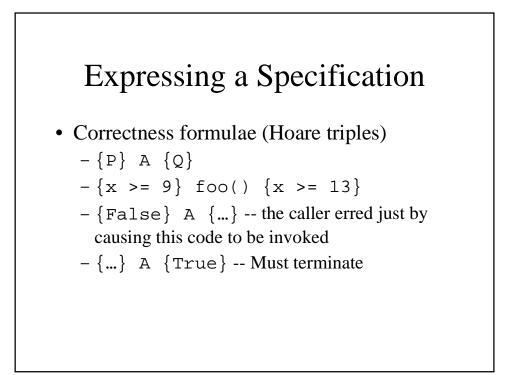
What Good Is It?

- Aid in documentation
- Aid in debugging
- Reliability (construct correct programs)
- Example: Ariane 5 crash, \$500 million loss
 - Conversion from a 64 bit # to 16 bit
 - The number didn't fit in 16 bits
 - Analysis had previously shown it would, so monitoring that assertion was turned off



Expressing a Specification: Assertions in C

- assert(x<0);</pre>
- Boolean expression
- Ignored unless in DEBUG mode
- If true, proceed, if false, abort
- Can get varying behavior in DEBUG and non-debug modes
- Eiffel gives you fine grained control on which assertions get checked

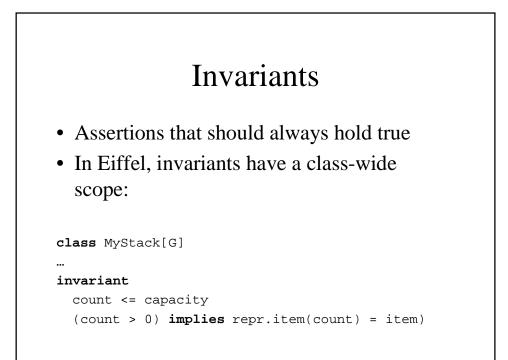


Preconditions and Postconditions

- The same idea, on a per-method basis
- Input requirements: preconditions
- Output requirements: *postconditions*
- preconditions: Caller's promise to the method
- postconditions: Method's promise to the caller

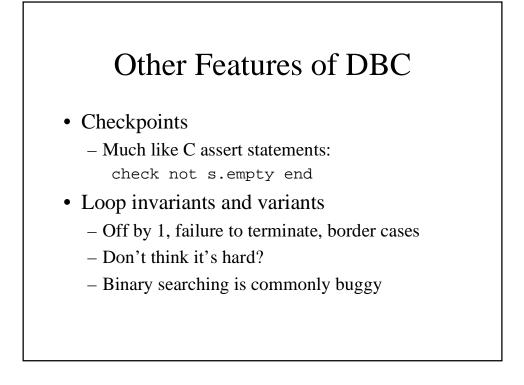
Example
class MyStack[G] feature
count: INTEGER
push(x: G) is
require
not full
do
code to perform the push
ensure
not empty
top = x
count = old count + 1
end

Contract Benefits and Obligati			
	Obligations	Benefits	
Client	Satisfy precondition: Only call push(x) if the stack is not full.	<i>From postcondition:</i> Stack gets updated to be non empty, w/ x on top, and count increased.	
Supplier	Satisfy postcondition: Update repr to have x on top, count increased by 1, not empty.	From precondition: Simpler implementation thanks to the assumption that the stack is not full.	



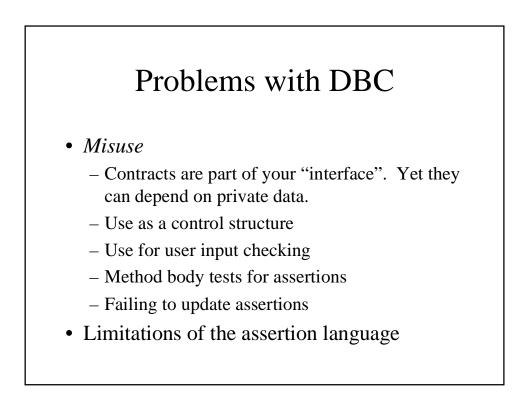
Invariants

- (Sometimes) It's unreasonable for invariants to *always* be true:
- Invariant: x != y
- swapping x and y would require 2 temporary variables, and some extra code
- When to suspend invariants?
 - obj.method(...) must satisfy on call and exit
 - method(...) need not (auxiliary tools)



Example Loop (gcd)

```
from
    x:= a; y:= b
invariant -- optional
    x>0;y>0
variant -- optional
    x.max(y)
until
    x = y
loop
    if x > y then x := x - y else y := y - x end
end
```



Eiffel's Assertion Language

- boolean expressions, + old, etc.
- No complex formal concepts (\forall, \exists)
- An engineering tradeoff:
 - Enough formal elements for reliability gains
 - Yet, keep it simple, learnable and efficient