

6.170 Lecture 4 Abstraction and Specification



John Guttag
MIT EECS

Two Kinds of Abstraction

Abstraction by parameterization
Beloved by 6.001

Abstraction by specification
Beloved by 6.170

Lambda Abstraction

Replace variables by formal parameters
Actual bound to formal at time of call

Increase the applicability of procedures
Good reason not to hardwire things
E.g., file names

Good reason not to use global or static variables



Potential Drawbacks to Using Parameters

Loss of efficiency

Only if large things need to be copied from actual to formal
Usually only copying a pointer

Long argument lists

Not if things packaged well

Lose state across calls

Issue if state associated with procedure rather than objects
Why we have static variables

Abstraction by Specification

Roughly speaking, separate what from how
Why also important, but something different

Describes required behavior

Not means of achieving it
Often not even exact behavior
Compute square root of x within ϵ

Specification denotes a (usually infinite) set of programs

An abstraction of each of its members

E.g., set of all procedures that sort lists

Abstracts from what kinds of lists
Abstracts from properties of sort, e.g., stable or not
Abstracts from algorithm, e.g., merge sort

An Example

```
int squareRoot(int in)
  requires in is a perfect square
  effects returns a square root of in
```

Admits a variety of algorithms for computing square root
Probably a good thing

Admits a number of different answers

Always positive square root
Always negative square root
Either negative or positive square root

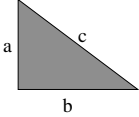
Is this a good thing ?

Maybe
Maybe not

6.170 **Relevance Depends Upon Intended Uses**

More general specifications
Lead to better implementations
More flexibility for later specialization
Inheritance later in term

More specific specifications
Lead to wider usability
 $c = \sqrt{a^2 + b^2}$



© John Chapin/John Guttag Spring 2000 Slide 7

6.170 **Specifications and Implementations**

Try to write specification first
Often amend while implementing
Easier to implement (e.g., constraint added)
More useful (e.g., constraint not needed)

Usually one implementation/specification
Important that one not confuse the two
Think of spec. as denoting set of potential implementations

Users should depend only on properties
Guaranteed by specification

Provides
Locality of using programs
Modifiability for implementer of abstraction

© John Chapin/John Guttag Spring 2000 Slide 8