Computer Science Program, The University of Texas, Dallas

Abstract Data Types (ADTs)

Why ADTs?

Main Concepts of ADTs

ADTs as Objects

Why Abstract Data Types (ADTs)?

Why Module Interconnection Languages (MILs)?

"Programming-in-the-small vs. programming-in-the-large"

	Programming-in-the-small	*Programming-in-the-small*
Goal	building "programs"	building "software"
Problem	Usually clear, small	Usually unclear, large
Emphasis	Detailed design & impl.	sw. architecture
Technique	Structured programming	"divide & conquer" "separation of concerns"
Notation	PLs	(Formal) (OO) Specification Lang.
Manpower	single person/small number	multi-person
Version	usually single	multi-version

interface-oriented

- **hiding the representations**

  localized change: change in detailed design & impl. does not affect other (client) objects

- **specifying the representations**

  conceptual consolution

- **conquering complexity**

  decomposing problems into collections of interacting components (-> encapsulation)

Main Concepts of ADTs

- ***model system as a collection of ADTs***
  - algebraic specification
  - model-theoretic (axiomatic) specification -> Z
- ***Larch is for Algebraic Interface Specification***

  ["Larch: Languages and Tools for Formal Specification",
  J. V. Guttag, J. J. Horning, S. J. Garland, K. D. Jones, A. Modet & J. M. Wing,
  Springer-Verlag, 1993]
- ***Each ADT includes:***
  - data objects
  - operations on data objects

    (specified as functions with domain and range)
  - essential properties of operations algebraic equations in FOL with equality

    (specified in algebraic equations in FOL with equality)
- ***Two-tiered approach to software development***
  - LCL - Larch Common Language

    PL-independent notation for writing interface spec.
  - LSL - Larch Shared Language

    PL-dependent notation for writing interface spec.
    LSLs exist for SmallTalk, Modula-3, C, C++, CLU, ...
- ***large library of predefined specs for common data types***
- ***Larch theorem prover for correctness of properties***

Main Concepts of ADTs

- Example

```
                              /* A trait describes an ADT, i.e., data objects, ops, properties of ops
    Stack (E, C): trait          & links to other ADTs */
                              /* E is an element (e.g., int, str, bool) of C, a container */
      introduces
           new: -> C
           push: C, E -> C
           top: C -> E exempting top (new)
           pop: C -> C exempting pop (new)
           isEmpty: C -> Bool
      asserts
           C generated by new, push
           forall stk: C, e: E
                top ( push ( stk, e ) ) == e
                pop ( push ( stk, e )) == stk
                isEmpty (new)
                ~ isEmpty ( push ( stk, e ))
      implies
           LinearContainer (push for insert, top for first, pop for rest)
```

Main Concepts of ADTs

- Questions

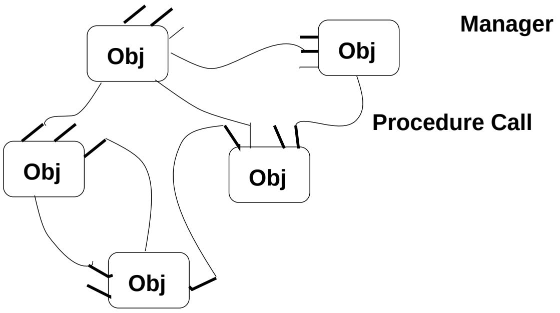
```
Stack (E, C): trait      /* A trait describes an ADT, i.e., data objects, ops, properties of ops
                            & links to other ADTs */
                         /* E is an element (e.g., int, str, bool) of C, a container */
    introduces
            new: -> C
            push: C, E -> C
            top: C -> E exempting top (new)
            pop: C -> C exempting pop (new)
            isEmpty: C -> Bool
    asserts
           C generated by new, push
           forall stk: C, e: E
                  top ( push ( stk, e ) ) == e
                  pop ( push ( stk, e )) == stk
                  isEmpty (new)
                  ~ isEmpty ( push ( stk, e ))
    implies
           LinearContainer (push for insert, top for first, pop for rest)
```

- pop (top (new)) ==
- isEmpty (pop (push (new, 5))) ==
- first (rest (insert (insert (new, 10), 11))) ==

/* Assume new for new */

ADTs as Objects

- Does Fortran/COBOL/C/Pascal offer subroutines for operations?
  Does Fortran/COBOL/C/Pascal offer information hiding?
- Objects (also called "managers") are responsible for
  - preserving the integrity of their resources &
  - hiding the representations from other objects
  - interactions take place mostly thru function and procedure invocations



- encourages reuse (<- hiding & encapsulation)
- interaction via procedure call
  - *a (client) object should know the identity of another (server) object (cf. pipe&filter)*
  - *change of an object identity necessitates modification of all other objects that explicitly invoke it*