# Language support for ADTs

- ADTs provide a way of providing an interface to a logical data type while hiding the underlying implementation
- Will lead naturally to OO considerations as later topic
- Benefits of ADTS well known to CS students:
  - Abstraction  reduces conceptual load for developers while working on any one part of the system
  - Separation of interface from implementation simplifies maintanance
  - Generalized encapsulation facilitates code reuse

# Support vs enforcement

- Many languages provide some level of support for user-defined abstract data types: creating public interfaces and underlying implementations
- Real question is whether the implementation hiding is simply supported or is it enforced – can your ADT completely hide its underlying implementation from the programmer using it, e.g. some languages that don't enforce:

  C++ supports, but doesn't enforce (developer can still see many implementation details of class in header file)

  C doesn't really even provide much support, structs can encapsulate data, but (short of function pointers) associated operations need to be handled through independent functions

# ADT feature considerations

- Need syntax for defining both the public interface and hidden implementation, and linking the two together
- What operations should be supported across all kinds of ADTs? (create, destroy, copy, access fields/methods, test for equality – same item vs same content)
- Can some variables/constants be shared across all instances of an ADT?
- Are nested definitions supported (ADTs whose member fields are ADTS, possibly defined only within the context of the parent ADT?)
- Can we create generic/templated ADTs?

# ADT implementation considerations

- How is parameter passing and assignment of ADTs implemented? (deep copy vs shallow/reference)
- As with earlier discussion of structs, arrays, etc: do instances get created on stack or in heap?
- For the functions/methods associated with ADT, do we actually replicate the function/method object code with each instance of the ADT, or do all instances somehow reference one common object code block?
- Will get into implementation details with C++ in OO section