DATA ABSTRACTION & MODULARITY

LECTURE 07

Review

- So far:
  - lots of language features
  - syntax, static semantics (type checking), and
  - dynamic semantics (evaluation)
  - how to build small programs

- Today:
  - new language feature: modules
  - how to build big programs: abstraction and specification

Problems With Writing Large Programs

- Wulf and Shaw: Global Variables Considered Harmful (1973)

1. Side effects — accesses hidden in functions
2. Indiscriminant access — can’t control
access 3. Screening — may lose access via

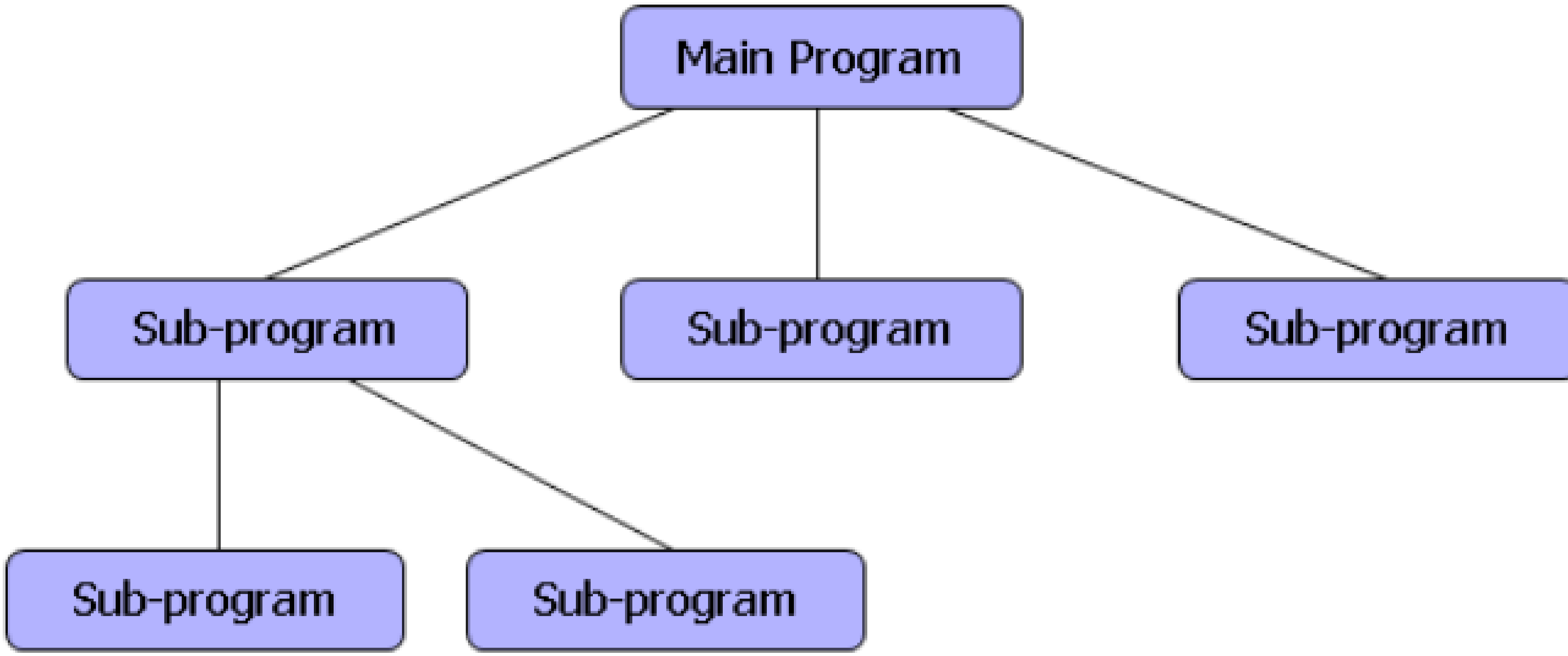
new declaration of variable

4. Aliasing

Stepwise Refinement

- "… program ... gradually developed in a sequence of refinement steps … In each step, instructions … are decomposed into more detailed instructions."
  - Niklaus Wirth, 1971

Program Structure


Main Program
Sub-program
Sub-program
Sub-program
Sub-program
Sub-program

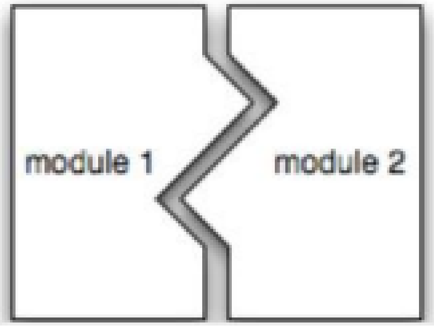
Data Refinement

- “As tasks are refined, so the data may have to be refined, decomposed, or structured, and it is natural to refine program and data specifications in parallel”

SOLUTION

Modular programming: code comprises independent *modules*

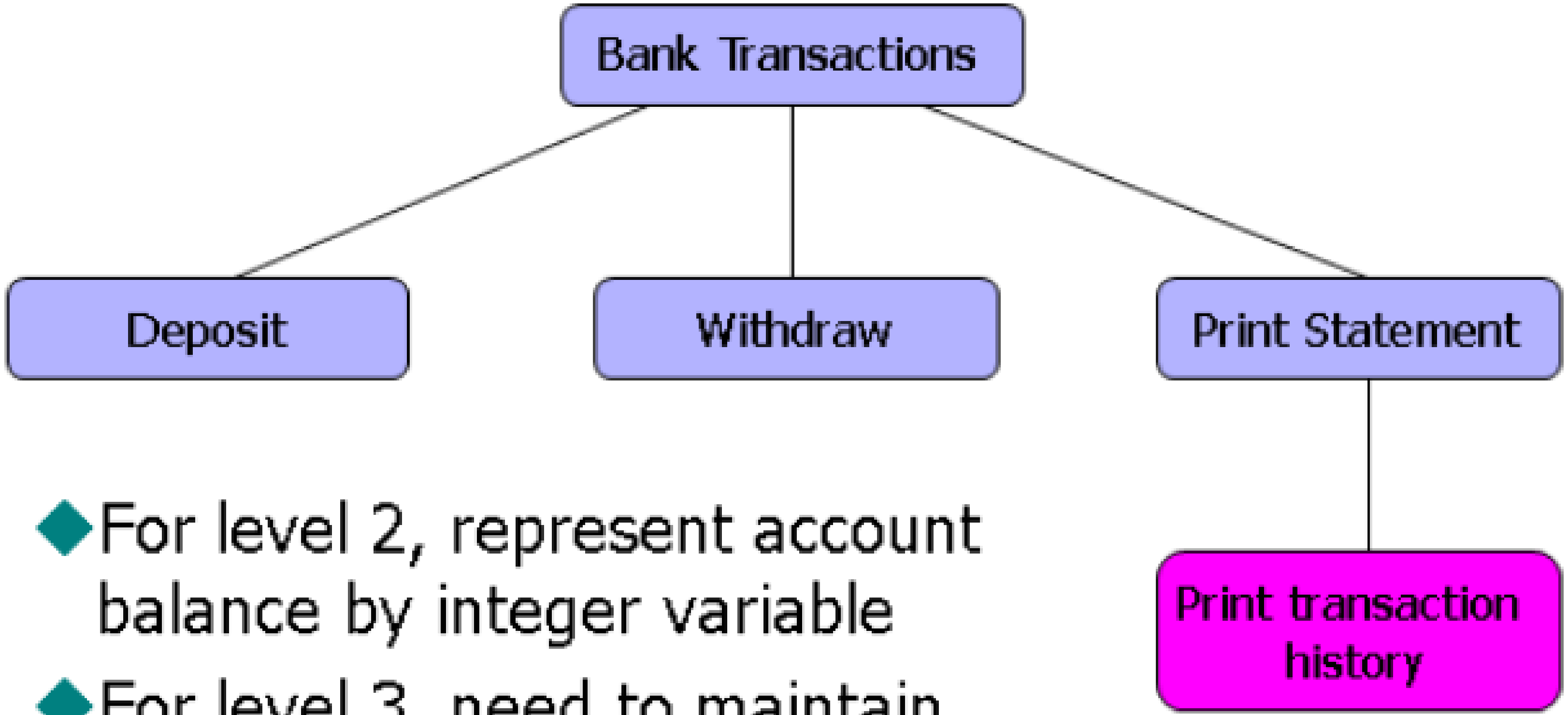
- developed separately
- understand behavior of module in isolation
- reason locally, not globally



What is Modularity?

- When we program, we try to solve a problem by
  – Step1: decompose the problem into smaller sub problems
  – Step2: try to solve each sub-problem separately – Each solution is a separate component that includes • Interface: types and operations visible to the outside • Specification: intended behavior and property of interface • Implementation: data structures and functions hidden from outside
- Example: a banking program

Example



- For level 2, represent account balance by integer variable
- For level 3, need to maintain list of past transactions

Modularity: Basic Concepts

- Component
  - Meaningful program unit
    - Function, data structure, module, …
- Interface
  - Types and operations defined within a component that are visible outside the component
- Specification
  - Intended behavior of component, expressed as property observable through interface
- Implementation
  - Data structures and functions inside component

Example: Function Component

- Component
  - Function to compute square root
- Interface
  - float sqroot (float x)
- Specification
  - If x>1, then sqrt(x)*sqrt(x) ≈ x.
- Implementation

```
float sqroot (float x){
 float y = x/2; float step=x/4; int i;
 for (i=0; i<20; i++){if ((y*y)<x) y=y+step; else y=y-step; step = step/2;}
 return y;
}
```

Example : Data Type

- Component
  - Priority queue: data structure that returns elements in order of decreasing priority
- Interface
  - Type pq
  - Operations empty : pq

insert : elt * pq → pq deletemax : pq → elt *

pq
- Specification
  - Insert adds to set of stored elements
  - Deletemax returns max elt and pq of remaining elts

Advantages of Modularity

- Many modern programming languages offer modules that have the following important features: –
  - They provide a way of grouping together related data and operations.
  - They provide clean, well-defined interfaces to users of their services.
  - They hide internal details of operation to prevent interference. – They can be separately compiled.

Advantages of Modularity

- Modules are an important tool for “dividing and conquering” a large software task by combining separate components that interact cleanly.
- They ease software maintenance by allowing changes to be made locally

Example: Java

- classes, packages
  - organize identifiers (classes, methods,

fields, etc.) into namespaces

- interfaces
  - describe related classes
- public, protected, private
  - control what is visible outside a namespace

What About C?

- C does not have an explicit concept of module. – But by careful use of header files, we can arrange for separately compiled C program files to have the above four properties of modules:

1. They provide a way of grouping together related data and operations.
2. They provide clean, well-defined interfaces to users of their services.
3. They hide internal details of operation to prevent interference.
4. They can be separately compiled.

Modularity in OOP

- In object-oriented languages like C++, Java and Python, we also have other constructs that help us to implement abstract data types like classes, interfaces, packages and modules.

- As an example, in Python, once we write a module, we can export classes and functions so that they can be used by other programs that import the module. What we don’t export remains hidden from the programs using the module.

Basic Concepts: Abstraction

- An abstraction separates interface from implementation

– Hide implementation details from outside (the client) • Function/procedure abstraction

– Client: caller of the function
– Implementation: function body
– Interface and specification: function declaration
– Enforced by scoping rules

- Data abstraction
  – Client: Algorithms that use the data structure
  – Implementation: representation of data
  – Priority queue can be binary search tree or partially-sorted array
  – Interface and specification: operations on the data structure – Enforced by type system

  – Modules A collection of related data and function abstractions

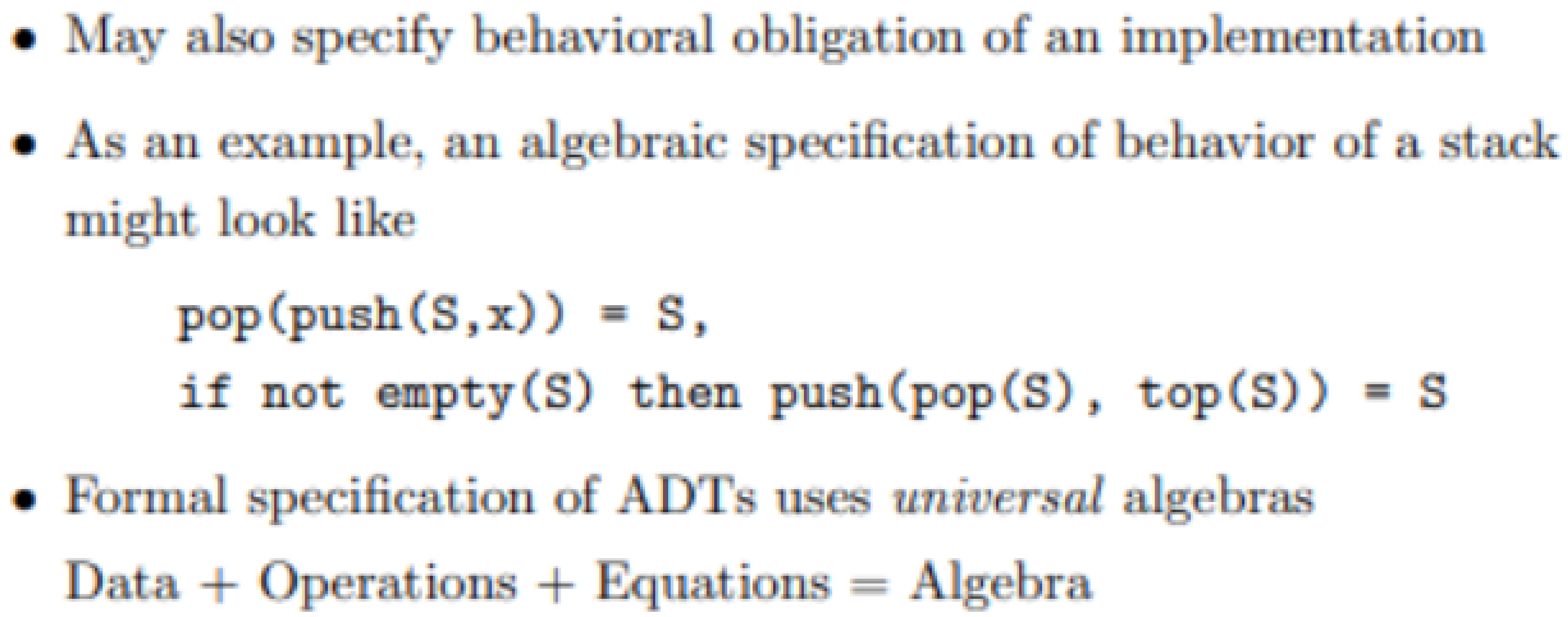
Abstract Data Types

- A major thrust of programming language design in 1970's
- Package data structure and its operations in same module
- Data type consists of set of objects plus set of operations on the objects of the type (constructors, accessors, destructors)
- Want mechanism to build new data types (extensible types)
- Should be treated same way as built-in types
- Representation should be hidden from users (abstraction)
- Users only have access via operations provided by the ADT (encapsulation)
- Distinguish between specification and implementation

ADT SPECIFICATION

- ADT specification declares data type and operations without implementation details, but possibly with semantics of operations
- Provides information needed to use ADT in programs
- Typically includes
  1. Data structures: constants, types, and variables accessible to user (although details may be hidden)
  2. Declarations of functions and procedures accessible to user (bodies not provided)

ADT (cont)



- May also specify behavioral obligation of an implementation
- As an example, an algebraic specification of behavior of a stack might look like

  ```
  pop(push(S,x)) = S,
  if not empty(S) then push(pop(S), top(S)) = S
  ```

- Formal specification of ADTs uses *universal* algebras

  Data + Operations + Equations = Algebra

Recap

- A module is a unit of organization of a software system that packages together a collection of entities (such as data and operations) and that carefully controls what external users of the module can see and use.
- Modules have ways of hiding things inside their boundaries to prevent external users from accessing them. This is called information hiding

Recap

- Abstract data types are collections of objects and operations that present well defined interfaces to their users, meanwhile hiding the way they are represented in terms of lower level representations.
- • Abstract data types are theoretical concepts. Modules can be used to implement abstract data types.

EXERCISE

- Define ADT for stack
- Define ADT for Queue