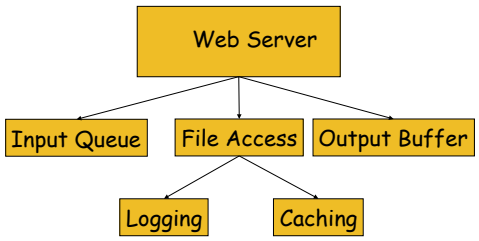
Modules and Abstraction

CSCI 334
Stephen Freund

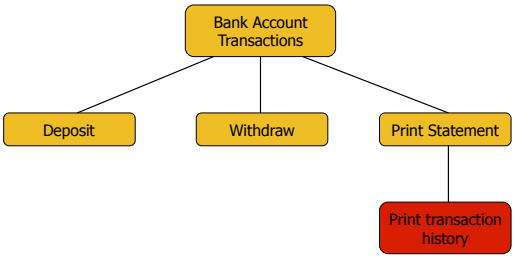
Program Design



Software Engineering Techniques

- Top Down
  - start with main tasks
  - iteratively refine
- Bottom Up
  - define basic concepts
  - combine
- Prototyping
  - define "approximation" of whole system
  - add functionality, extend data rep

Data Refinement Example



Example: power Function

- Interface
  - `fun power : int * int -> int`
- Specification
  - `power(m,n)` returns the value m^n
- Implementation

```
fun power(m,n) =
    if n > 0 then m * power(m,n-1)
             else 1;
```

Example: Stack

- Interface

```
type Stack
val empty : Stack;
fun push : int * Stack -> Stack
fun pop : Stack -> int * Stack
```

- Specification
  - Pre/Post conditions
  - "behaves like a stack..."
- Implementation

```
type Stack = int list;

val empty = nil;

fun push(n, ns) = n::ns;

fun pop(nil) = (0, empty)
  | pop(n::ns) = (n, ns);
```

abstype Stack

```
abstype Stack =
    StackRep of int list
with
   val empty = StackRep(nil);
   fun push(n, StackRep(ns)) = StackRep(n::ns);
   fun pop(StackRep(nil)) = (0,empty)
     | pop(StackRep(n::ns)) = (n,StackRep(ns));
end;
```

Using an Abstract Type

```
- abstype Stack =
      ...
type Stack;
empty = - : Stack
push = fn: int * Stack -> Stack
pop = fn: Stack -> int * Stack

- val st = push(3, empty);
val st = - : Stack
- val (top,_) = pop(st);
val it = 3 : int
```

Stack abstype (2nd version)

top index & array

```
abstype Stack =
    StackRep of int * int array
with
   val empty = (0, Array.array(100, 0));

   fun push(n, StackRep(top, a)) = (
      Array.update(a,top,n);
      StackRep(top + 1, a)
    )

   fun pop(StackRep(top, a)) =
      (Array.sub(a,(top - 1)), StackRep(top - 1, a));
end;
```

Expr abstype

```
- abstype Expr =
    VarX | Times of Expr * Expr
  with
    fun buildX()              = VarX;
    fun buildTimes(e1,e2)     = Times(e1,e2);
    fun exprToString ... = ...;
    fun eval ... = ...;
  end;

- val e = buildTimes(buildX(), buildX());
val e = - : Expr
- exprToString e;
val it = "x * x": string
```

Module

- Interface
  - set of name and type declarations
- Spec
  - same as before…
- Implementation
  - code/decls to define names
- Examples:
  - modules in Modula
  - Ada packages
  - ML structures
  - Java packages

File System Module

```
INTERFACE FileSystem;
  CONST max_files : INTEGER = 10;
  TYPE
    Byte = BITS 8 FOR [0 .. 255];
    File = RECORD (* name, permissions, etc. *) END;
  END;
  PROCEDURE open(name : String): File;
  PROCEDURE read(File f; VAR b: ARRAY OF Byte);
  PROCEDURE close(File f);
END FileSystem.

IMPLEMENTATION FileSystem;
  TYPE TempFile = ...;
  PROCEDURE open(name : String): File
     BEGIN ... END
  ...
```

Generic Abstractions

- Parameterize component by types
  - Ada generics, ML functors, C++ STL library
- Example

```
class Stack<T> {
  private Vector<T> elems;
  public boolean empty() { ... }
  public void push(T elem) { ... }
  public T pop() { ... }
}

Stack<String> stringStack;
Stack<int> intStack;
```

Interview with Stepanov

- Question: What is the origin of STL?
- Answer: In 1976, still back in the USSR, I got a very serious case of food poisoning from eating raw fish. While in the hospital, in the state of delirium, I suddenly realized that the ability to add numbers in parallel depends on the fact that addition is associative.

  In other words, I realized that ... *algorithms are defined on algebraic structures.*

Example (in Java Syntax...)

```
public <A> Iterator<A> iterator(Vector<A> v) { ... }

public <A> Iterator<A> sort(Iterator<A> data,
                            Comparator<A> comp) { ... }

public <A> Iterator<A> merge(Iterator<A> a1, Iterator<A> a2,
                             Comparator<A> comp) { ... }


Vector<String> strings;
OrderedVector<String> orderedStrings;
Comparator<String> stringComp;
Iterator<String> iter = merge(sort(strings, stringComp),
                              iterator(orderedStrings),
                              stringComp);
```

C++ STL Efficiency

	N = 50,000	N = 500,000
C (qsort)	1.42	18.16
C++ (arrays)	0.28	3.84
C++ (STL vector)	0.27	3.80