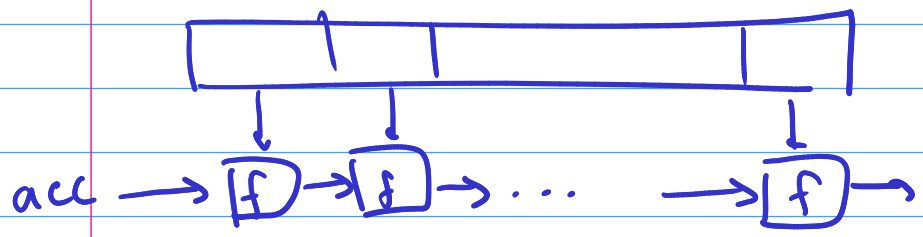
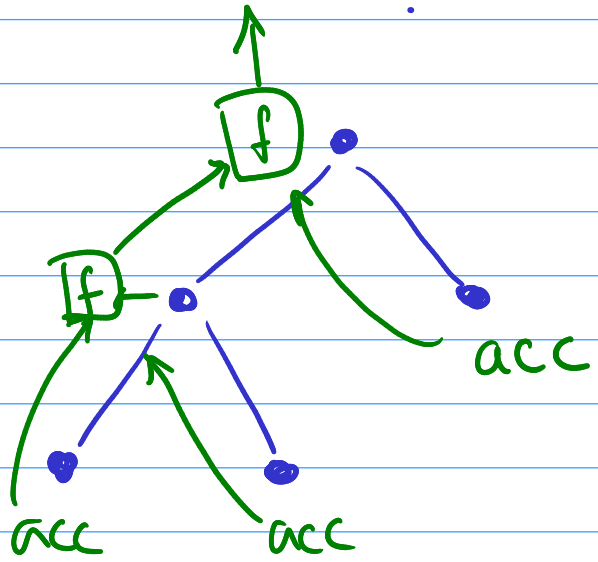

acc
f
f
f


f
f
acc
acc
acc

- Syntactic organization : Compilation units
- Namespace : C++ Namespaces, Java packages
- Abstraction : Classes, Functions
- Code reuse : Inheritance, parametric polymorphism

Vernacular	Values	Types	Functions
Module	Structure	Signature	Functors
	Implementation	Interface	

Ex 1 : Stacks

type stack = ———

push : int → stack → stack

pop : stack → int option * stack

Contract "Signature"

```
type stack = int list

let push (a : int) (s : stack) : stack = a :: s

let pop (s : stack) : int option * stack =
  match s with
  | hd :: tl -> (Some hd, tl)
  | [] -> (None, s)
```

Impl 1

```
type stack2 = EmptyStack | NonEmpty of int * stack2

let push2 a s = NonEmpty(a, s)

let pop2 s =
  match s with
  | NonEmpty(a, s2) -> (Some(a), s2)
  | EmptyStack -> (None, s)
```

Impl 2

"Structure"

```
module type Stack =
 sig
   type stack
   val emptyStack : stack
   val push : int -> stack -> stack
   val pop : stack -> int option * stack
 end

module ListStack : Stack =
  struct
    type stack = int list
    let emptyStack = []
    let push a s = a :: s
    let pop s =
      match s with
      | [] -> (None, s)
      | hd :: tl -> (Some hd, tl)
  end

module ADTStack : Stack =
  struct
    type stack = EmptyStack | NonEmpty of int * stack
    let emptyStack = EmptyStack
    let push a s = NonEmpty(a, s)
    let pop s =
      match s with
      | EmptyStack -> (None, s)
      | NonEmpty(a, s2) -> (Some a, s2)
  end
```

Extended Stacks

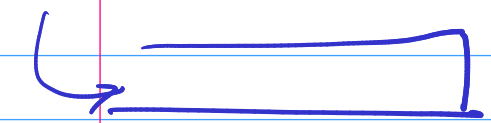
```
module type ExtendedStack =
  sig
    type stack
    val emptyStack : stack
    val push : int -> stack -> stack
    val pop : stack -> int option * stack
    val reverse : stack -> stack
  end
```

How to reverse a stack?

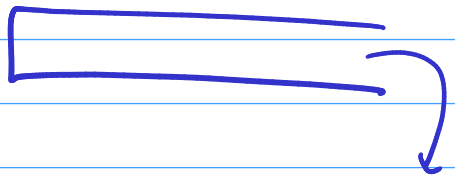
```
let reverseLS s =
  let rec helper acc s =
    match ListStack.pop s with
    | (Some hd, s2) -> helper (ListStack.push hd acc) s2
    | (None, _) -> acc in
  helper ListStack.emptyStack s

let reverseAS s =
  let rec helper acc s =
    match ADTStack.pop s with
    | (Some hd, s2) -> helper (ADTStack.push hd acc) s2
    | (None, _) -> acc in
  helper ADTStack.emptyStack s
```

```
module Extend(S : Stack) : ExtendedStack =
 struct
   include S
   let reverse s =
     let rec helper acc s =
       match S.pop s with
       | (Some hd, s2) -> helper (S.push hd acc) s2
       | (None, _) -> acc in
     helper S.emptyStack s
 end

module ExtendedLS = Extend(ListStack)

module ExtendedAS = Extend(ADTStack)
```

Ex 2 : Queue



O(1)



O(n) worst case

O(1) amortized