Rochester Institute of Technology
Department of Computer Science

Dynamic Encapsulation of C++ Objects for Distributed Object-Oriented Systems

A Masters Thesis Defense

June 19, 1997

Frank Barrus
shaggy@csh.rit.edu
Computer Science House

Thesis Defense Overview

- Why: history
- Theory
- Dynamic Encapsulation Model
- DC++ Syntax
- DECO
- Conclusions

History

- Shag/OS v0.34 - 1992-3
  - Dynamic objects
  - C Macros
- Shag/OS v1.00 - 1995-?
  - C++
  - Dynamic -> Static
  - DC++

Theory overview

Will cover quickly -- ask questions if necessary

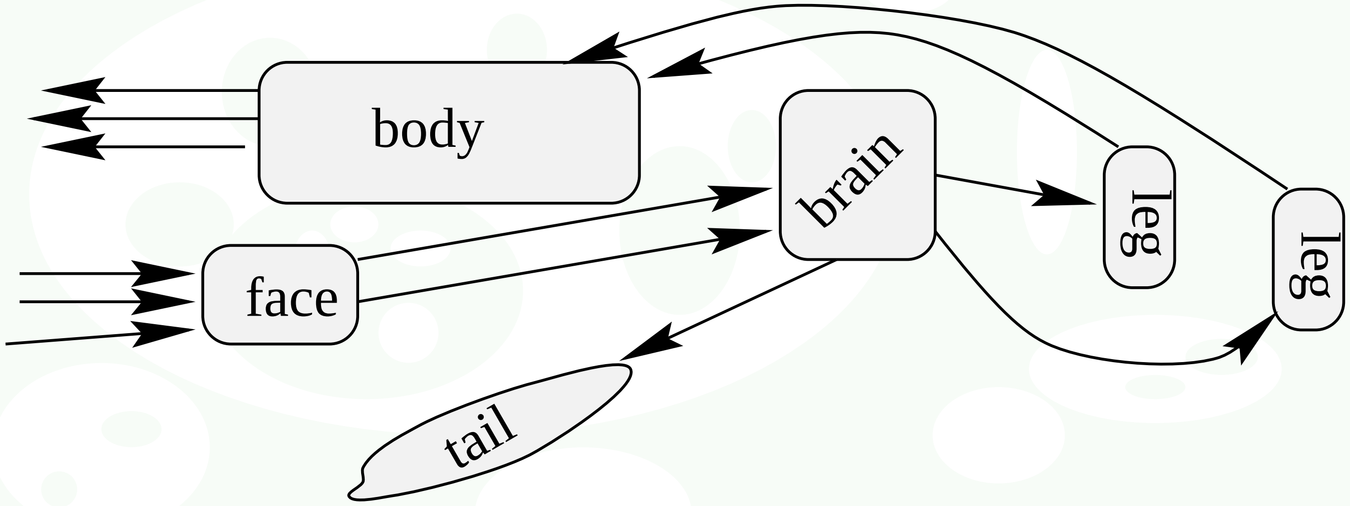
- Objects
- Types/Classes, Inheritance
- Signatures, Conformance
- Encapsulation (static/dynamic)
- Extensional/Intensional Objects
- Forwarders

Objects

- Interface (features)
  - attributes
  - operations
  - methods
- Implementation
  - state
  - routines (functions, procedures)
- Parent objects
  - delegation

Objects

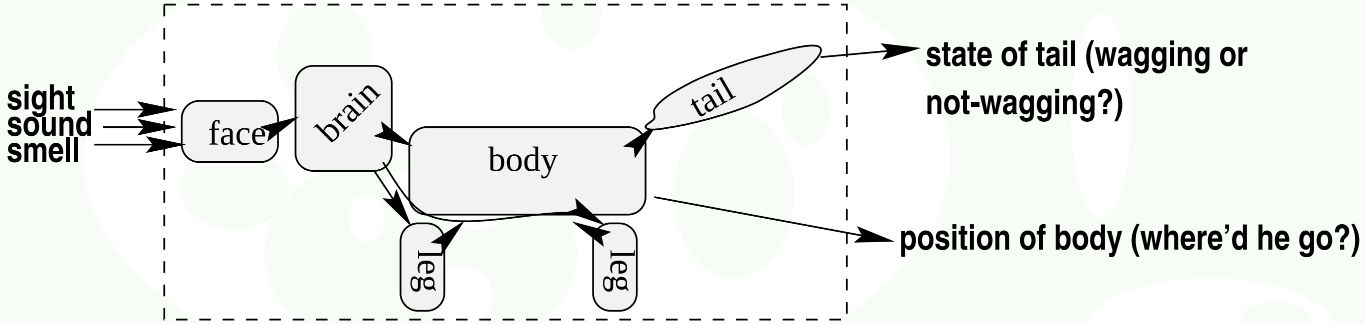
example1



Objects

example1

our_dog:



(our_dog is a member of the class TwoLeggedCanines--
a very rare species)

Current Object

- **self**
  original invoker of an operation
- **here**
  owner of current implementation routine
- **this**
  hybrid: `self or here`
- **super**
  parent object

Classes

- Static Parent Objects
- Implementation (concrete)
- Classification/Organization
- Inheritance hierarchy
  - code reuse
  - interface reuse

Types

- Interface
- Abstraction
  - “selective-ignorance”
    (A.Koenig)
- Organization hierarchy for feature names
- Ensure correct semantics

Signatures

- Function
  - Arity, Argument Types, Return Type
- Method
  - Function, Self
- Class
  - Data Structure
  - Methods
- Type
  - Methods

Conformance

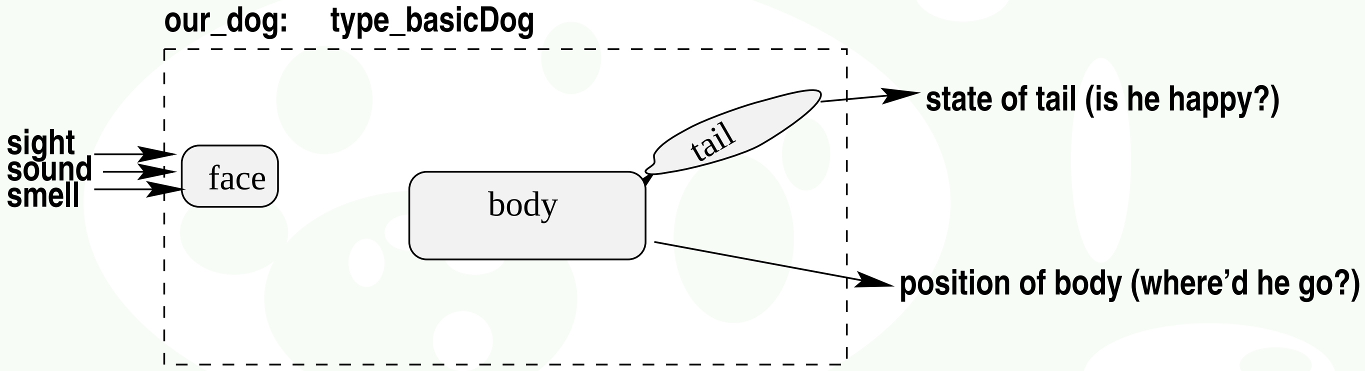
- Function
  - Covariance
  - Contravariance
- Method
  - Function, Self
- Class
  - Data extension
- Type
  - Method addition
  - Method overriding

Encapsulation

- Static
  - Compile-time
  - Code knows internals
  - C++, Eiffel
- Dynamic
  - Run-time
  - Code knows only interface
  - Smalltalk, CLOS

Encapsulation

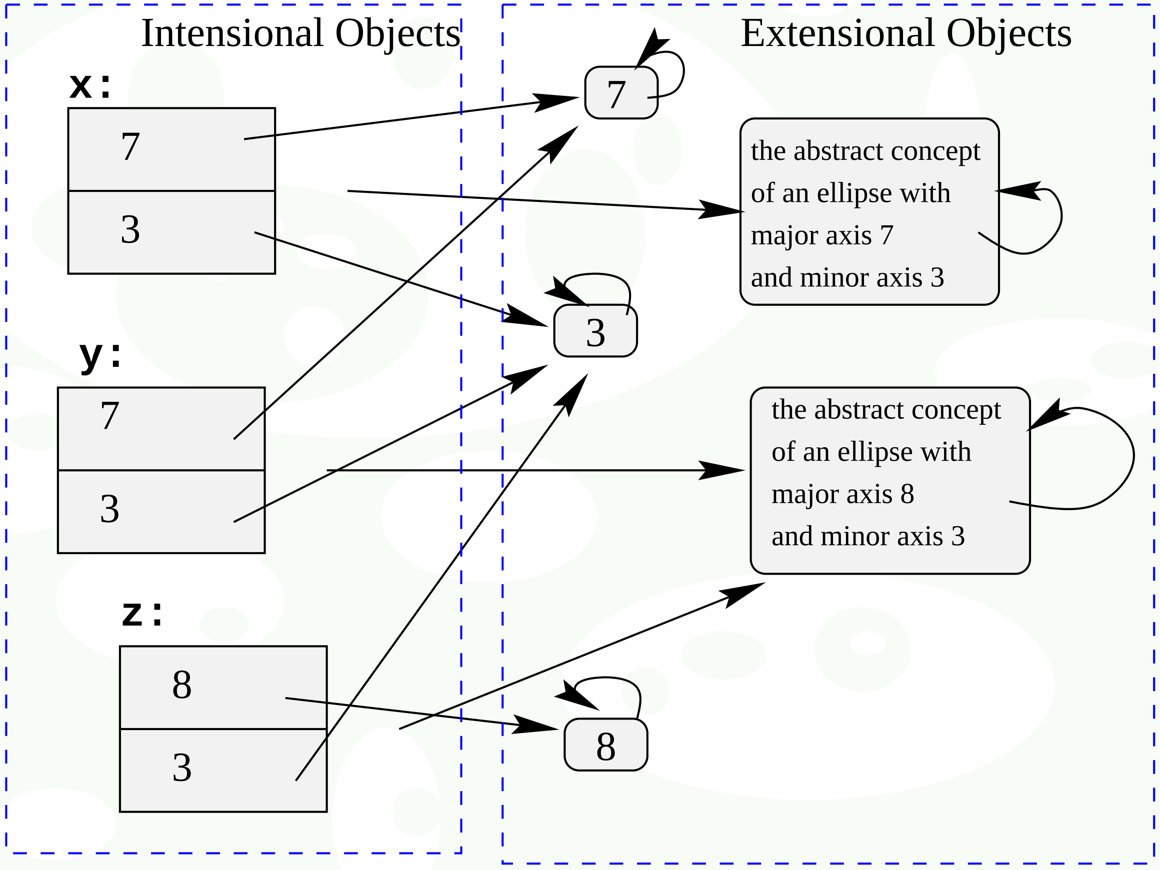
example



Extensional/Intensional Objects

- Extensional Objects
  - Simply exist
  - No state
  - Abstract concepts
  - Immutable
- Intensional Objects
  - Have a state
  - Can refer to other intensional/extensional objects
  - Mutable

Extensional/Intensional *example*



Forwarders

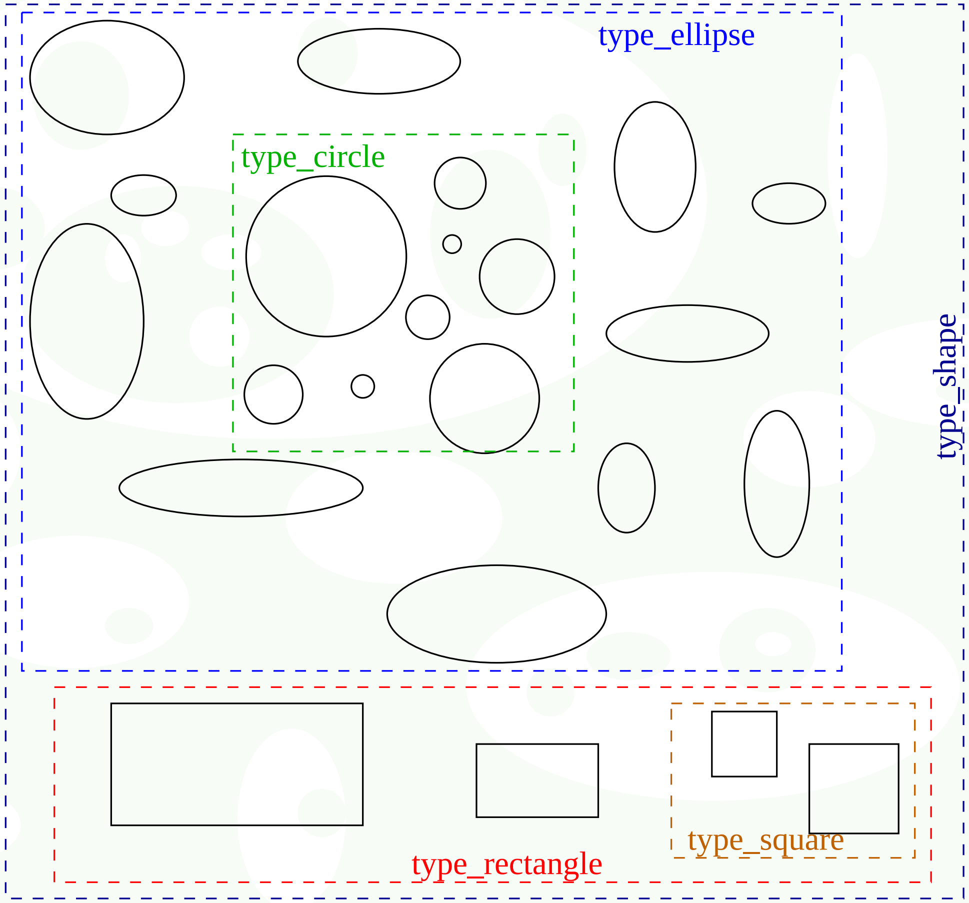
- Can be attached to or removed from objects
- “forward” messages to themselves
- Original object becomes superobject
- Applications:
  - Security
  - Debugging
  - Patching
  - “magic”
- ‘self’ must be the forwarder

Some C++ Problems

- Types == Classes
  - single hierarchy
- Static Encapsulation
  - code needs recompiling if classes change
- Casting
- Implicit numerical conversions

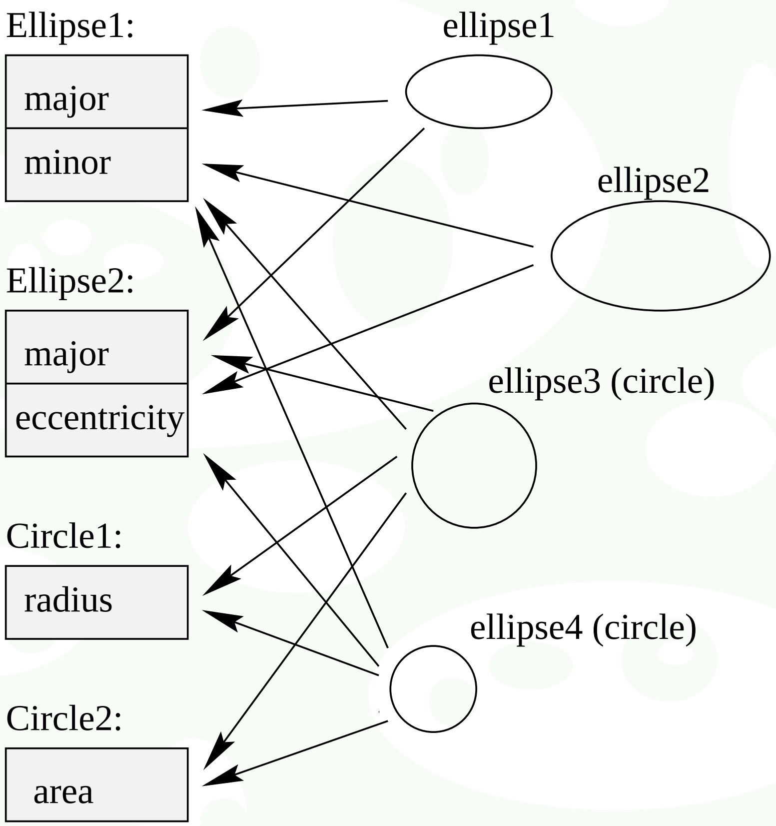
Circles and Ellipses

classification



Circles and Ellipses

assignment



Circles and Ellipses

hierarchies

Read-only:

shape

rectangle ellipse

square circle

Write-only:

square circle

rectangle ellipse

shape

Read-write:

square

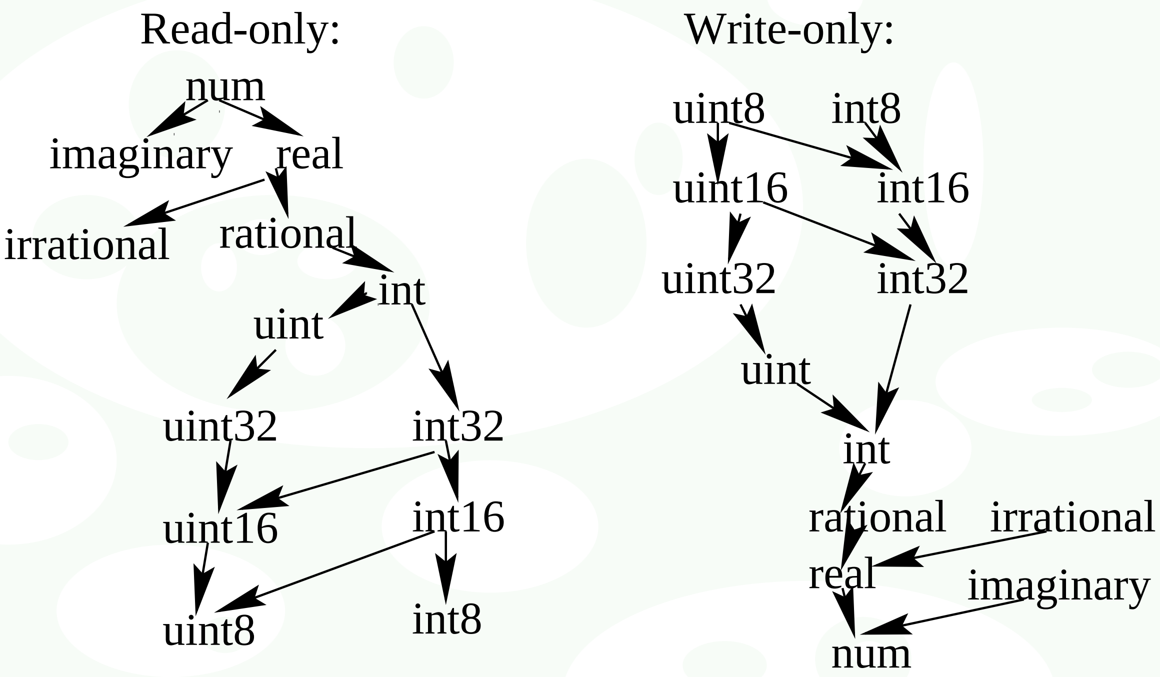
shape ellipse

circle rectangle

(no implicit subtype relationship)

Numbers

hierarchies



Dynamic Encapsulation Model (DEM)

principles

- as flexible as is reasonable
  - static type checking when possible
  - allow dynamic checking as well
- sharing (memory conservation)
- not tied to one language
- transparent networking
- persistence

Dynamic Encapsulation Model (DEM)

features

- Class/Type Objects
- Object References
- Bound Typed References
- Object Managers
- Data Managers
- Dynamic Inheritance

Class Objects

- Lists of all supported methods
- Signature information
- Conformance checking
- Create an instance
- Bind to a type

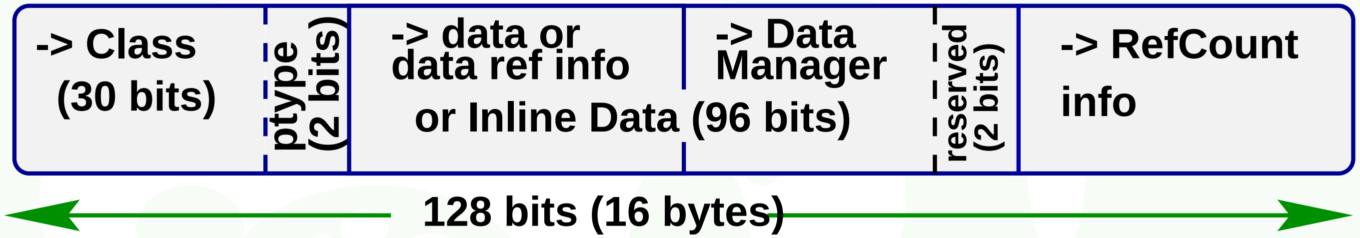
Type Objects

- List methods
- Signature information
- Conformance checking
- Bind to other types

Object References

- Stored in object descriptor
- Class
- Data
  - Inline
  - Data Manager
- Intensional/Mutable
- Can refer to object references or extensional objects

Object Descriptor



Bound Typed-References

- Use a ‘hidden’ object descriptor
- Type is statically known
- Are “ready to go”
- Use vtable-style mechanism
  - with thunks
  - several variations
- Can refer to object references or extensional objects

Data Managers

- Responsible for managing storage of raw data
  - Alloc/Free/Resize
  - Reference counting and garbage collection
  - Copy-on-write (COW) semantics
  - Might cooperate with Virtual Memory system
  - Could handle remote data transport or persistence
- Located via object descriptor
- Are also objects

Object Managers

- Accessed via ‘proxy’ object descriptors
  - class = a particular object manager
  - data = some sort of handle
- Used for remote object access
- Used for persistence
- Examples
  - File handles
  - Remote object handles
  - User-level handles for kernel-level services
  - Thread handles

Object Migration

- Permanent (call-by-move)
  - Copy of object is transported
  - Local proxy is created using Object Manager
  - Local object is replaced with proxy
- Temporary (call-by-visit)
  - Copy of object is transported
  - Local proxy is created using Object Manager
  - Local object is attached to original object with a forwarder
  - Later, local object is updated, and forwarder is removed

Object Migration *(continued)*

Dynamic Inheritance

- Superobjects
- Forwarders
  - must conform to type of object
    (will occur implicitly, due to delegation)
- Mixins
  - partially implemented types
  - potential for run-time errors

DC++ Goals

- Support DEM model
- Simple creation of types/classes
- Reuse existing C++ code
- Minimal number of new keywords

DC++ Syntax

additions

- dynamic
  - `dynamic class`
  - `dynamic virtual class`
  - shortcut:

```
#define dtype dynamic virtual class
#define dclass dynamic class
```

- ?=
  - conditional assignment

DC++ Syntax

conventions

- `t_typename`
  - extensional type
- `m_typename`
  - mutable type
- `r_typename`
  - read-only mutable type
- `w_typename`
  - write-only mutable type
- `r_typename != t_typename`

DC++ Syntax

types

```
dynamic virtual class t_ellipse {
    int major() const;
    int minor() const;
};
```

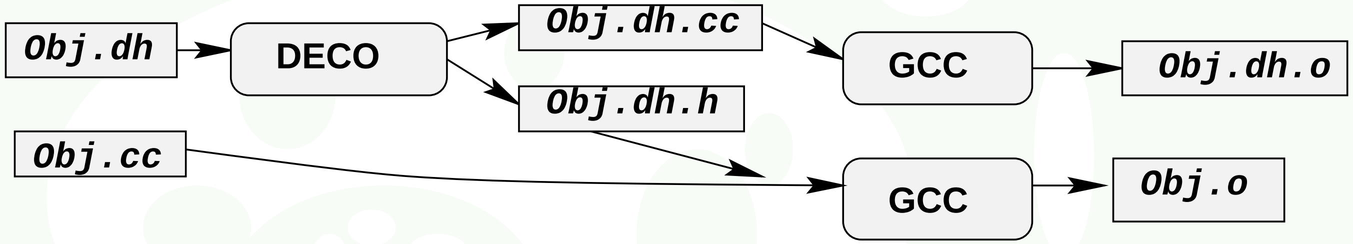
DC++ Syntax

classes

```
dynamic class D_ellipse : public Ellipse {
t_ellipse:
    int major() = Ellipse::major;
    int minor() = Ellipse::minor;
};
```

DECO Usage

an an IDL

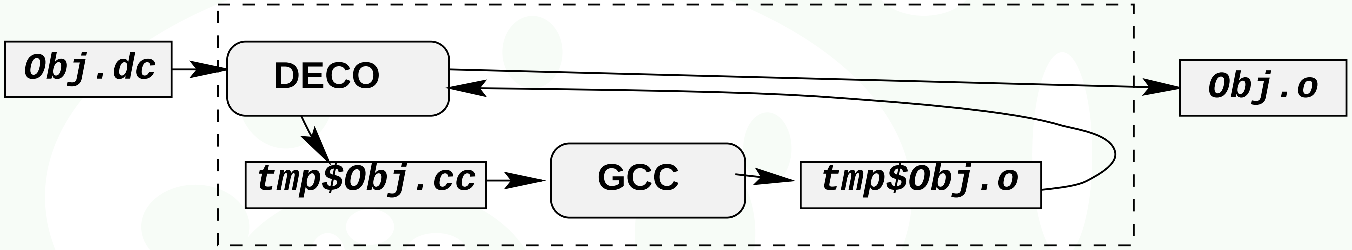


- **Commands:**

```
% deco Obj.dh
% g++ -c Obj.dh.cc
% g++ -c Obj.cc
% g++ -o objtest objtest.o Obj.o \
      Obj.dh.o -ldeco
```

DECO Usage

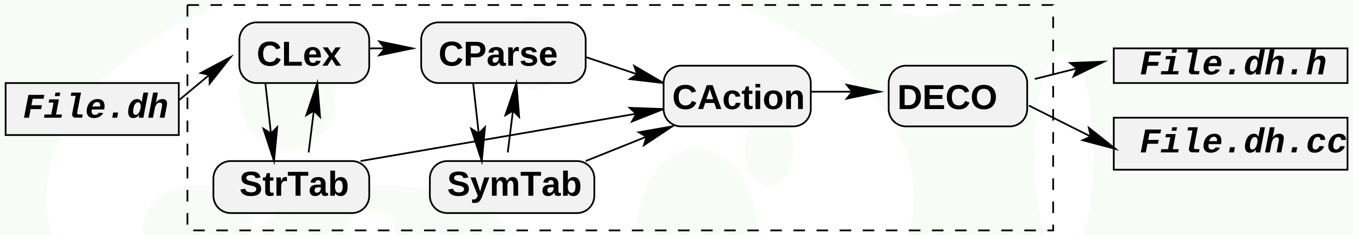
as a compiler



- **Commands:**

```
% deco -c Obj.dc
% g++ -o objtest objtest.o Obj.o -ldeco
```

DECO Internals



- **CLex**
  - lexical analysis of C/C++/DC++
- **CParse**
  - syntactical analysis and parsing
- **CAction**
  - interface used by CLex and CParse-- allows parser and main program to be modular and independent
- **DECO**
  - responds to CAction requests and generates C++ code

Complications

- Concurrency
  - Data sharing
  - Type binding
- "Self" usage
- Exceptions
- Fast dispatching
- Fast/efficient binding
- Garbage collection

Conclusions

- Separation of Type and Class
  - increased expressiveness
  - decreased redundant code
- Dynamic Encapsulation
  - increased expressiveness
  - still want static type checking
- C++
  - difficult to parse (ambiguities)
  - already too large and complex

For more information...

- Thesis webpage:
  http://www.csh.rit.edu/˜shaggy/thesis.html
- Email:
  shaggy@csh.rit.edu
- Bibliography
  attached to handouts