

CS 242

## Final Review

TA Review Session

Final Exam

Monday Dec 10  
12:15 – 3:15 PM  
B01

## Announcements

- Today – last homework due 5PM
  - Homework graded tomorrow, available Friday
- No section this week
- Next week
  - Final exam – Monday afternoon
  - Local SCPD students come to campus for exam
    - **IMPORTANT** : Remote students email us by this Friday for faxing instructions.

## Outline

- Course overview
- 50 questions you should know how to answer
- Course summary

## Course Goals

- Understand how programming languages work
- Appreciate trade-offs in language design
- Be familiar with basic concepts so you can understand discussions about
  - Language features you haven't used
  - Analysis and environment tools
  - Implementation costs and program efficiency
  - Language support for program development

## General Themes in this Course

- Language provides an abstract view of machine
  - We don't see registers, length of instruction, etc.
  - We see functions, objects, pointers (in some lang), ...
  - If programs don't depend on implementation method, compiler writers can choose best implementation
- Language design is full of difficult trade-offs
  - Expressiveness vs efficiency

## Good languages designed for specific purpose

- C: systems programming
- JavaScript: make web pages interactive
- Lisp: symbolic computation, automated reasoning
- FP: functional programming, algebraic laws
- ML: theorem proving
- Clu, ML modules: modular programming
- Simula: simulation
- Smalltalk: Dynabook,
- C++: add objects to C

## Design Issues

- Language design involves many trade-offs
  - space vs. time
  - efficiency vs. safety
  - efficiency vs. flexibility
  - efficiency vs. portability
  - static detection of type errors vs. flexibility
  - simplicity vs. "expressiveness" etc
- These must be resolved in a manner that is
  - consistent with the language design goals

Many program properties are undecidable (can't determine statically )

- Halting problem
- null pointer detection
- alias detection
- perfect garbage detection
- etc.

Static type systems

- detect (some) program errors statically
- can support more efficient implementations
- are less flexible than either no type system or a dynamic one

## 50 things you should know to get an A in CS242

- Necessary, not sufficient condition

## 50 Questions to answer...

1. Write tail-recursive factorial in Lisp.
2. Explain the Lisp memory model. What is car and cdr?
3. Which is faster, an interpreter or a compiler?
4. Be able to evaluate  $(\lambda f. \lambda x. f(f x)) (\lambda y. y+1)$
5. Be able to explain the difference between alpha and beta reduction.

## 50 Questions to answer...

6. What makes a functional language different from a non-functional language? Parenthesis?
7. Cogently explain why everyone doesn't use functional programming.
8. Write a proof undecidability of the halting problem.
9. Appropriately reduce something to the halting problem.

## 50 Questions to answer...

10. Know the difference between pass-by-value, pass-by-reference. Be able to solve a problem related to this.
11. Explain a situation where compile time type checking is a bad idea.
12. Explain what type safety is and why Java has it and C does not.
13. Would adding type inference to C be a good idea?

### 50 Questions to answer...

- 14. Know how to do type inference tree questions.
- 15. Explain Polymorphism vs Overloading. How are they different. Give a language that uses each of them.
- 16. Describe an activation record. What is in it and why?
- 17. The control/access link points to the ...?
- 18. Explain dynamic scoping and give 2

### 50 Questions to answer...

- 19. Define and compare the upward and downward funarg problems.
- 20. Why do we need closures (other than to give you some diagrams to fill out)?
- 21. What are the 4 aspects of OOP? <you better know this>
- 22. What is the difference between subtyping and inheritance. Give a language where these are the same and one where they are different.

### 50 Questions to answer...

- 23. How are objects represented in Simula?
- 24. Draw the runtime Smalltalk representation of a colored point.
- 25. Explain why the Ingalls test may not be the best way to judge a PL. What aspect of languages does he focus on?
- 26. What is the main C++ design goal? Give 3 examples of C++ design decisions that directly support this goal.

### 50 Questions to answer...

- 27. C++ vs Smalltalk vs Java dynamic lookup, how are they different. Is C++'s really dynamic?
- 28. How are C++ vtables laid out in subclasses?
- 29. What is the difference between virtual and overloaded C++ functions?
- 30. Explain the diagram on C++ slide 28 without getting confused.

### 50 Questions to answer...

- 31. How does C++ deal with diamond inheritance? And what is diamond inheritance?
- 32-34. Study, study, study.
- 35. Is Java like Smalltalk in that everything is an Object?

### 50 Questions to answer...

- 36. What is the difference between a C++ destructor and a Java finalize method?
- 37. Explain why Java arrays aren't really covariant. Since they aren't covariant, they must be contravariant, right?
- 38. What are the steps from getting a class of the network to running it in the JVM? Why is each step run?

## 50 Questions to answer...

- 39. What is the purpose of stack inspection and how does it work?
- 40. What is the purpose of Java sandboxing and what kinds of actions are not allowed for programs in a sandbox?
- 41. What concurrency issues are not present in a multiprogramming environment that you have in a multiprocessor environment?

## 50 Questions to answer...

- 42. Write some Java code that has a critical section. Explain how you might get a race condition from your code.
- 43. Why does the Actor model of concurrency have forwarders?
- 44. How does the synchronize keyword work in Java? How do you synchronize blocks of code versus functions?
- 45. Why do many interoperability schemes use C/C++ as a "Lingua Franca"? Why not use Java?

## 50 Questions to answer...

- 46. What would a perfect world interoperability scheme look like?
- 47. What purpose does an interface definition language serve? What are two examples of IDL's given in class?
- 48. What's a buffer overflow and how do language features in Java help prevent it?
- 49. How do you draw the layout of memory in relation to scoping?
- 50. Have you studied?

## Summary of the course

- JavaScript
  - ◆ Modularity and objects
    - encapsulation
    - dynamic lookup
- Lisp
  - subtyping
  - inheritance
- Fundamentals
  - lambda calculus
  - functional prog
- ML and type systems
  - ◆ Simula and Smalltalk
  - ◆ C++ , Java
  - ◆ Concurrency
- Block structure and activation records
  - ◆ Logic programming
- Exceptions and

## Fundamentals

- Grammars, parsing
- Lambda calculus
- Functional vs. Imperative Programming

## JavaScript Language Features

- Stack memory management
  - Parameters, local variables in activation records
- Garbage collection
  - Automatic reclamation of inaccessible memory
- Closures
  - Function together with environment (global variables)
- Exceptions
  - Jump to previously declared location, passing values
- Object features
  - Dynamic lookup, Encapsulation, Subtyping, Inheritance

## Lisp Summary

- Successful language
  - Symbolic computation, experimental programming
- Specific language ideas
  - Expression-oriented: functions and recursion
  - Lists as basic data structures
  - Programs as data, with universal function `eval`
  - Stack implementation of recursion via "public pushdown list"
  - Idea of garbage collection

## Algol Family and ML

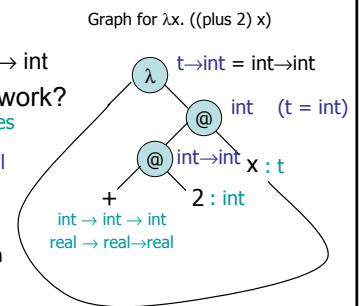
- Evolution of Algol family
  - Recursive functions and parameter passing
  - Evolution of types and data structuring
- ML: Combination of Lisp and Algol-like features
  - Expression-oriented
  - Higher-order functions
  - Garbage collection
  - Abstract data types
  - Module system

## Types and Type Checking

- Types are important in modern languages
  - Program organization and documentation
  - Prevent program errors
  - Provide important information to compiler
- Type inference
  - Determine best type for an expression, based on known information about symbols in the expression
- Polymorphism
  - Single algorithm (function) can have many types

## Type inference algorithm

- Example
  - `fun f(x) = 2+x;`
  - `> val it = fn : int → int`
- How does this work?
  - Assign types to leaves
  - Propagate to internal nodes and generate constraints
  - Solve by substitution



## Block structure and storage mgmt

- Block-structured languages and stack storage
- In-line Blocks
  - activation records
  - storage for local, global variables
- First-order functions
  - parameter passing
  - tail recursion and iteration
- Higher-order functions
  - deviations from stack discipline

## Summary of scope issues

- Block-structured lang uses stack of activation records
  - Activation records contain parameters, local vars, ...
  - Also pointers to enclosing scope
- Several different parameter passing mechanisms
- Tail calls may be optimized
- Function parameters/results require closures
  - Closure environment pointer used on function

## Control

- Structured Programming
  - Go to considered harmful
- Exceptions
  - “structured” jumps that may return a value
  - dynamic scoping of exception handler
- Continuations
  - Function representing the rest of the program
  - Generalized form of tail recursion

## Modularity and Data Abstraction

- Language support for information hiding
  - Abstract data types
  - Datatype induction
  - Packages and modules
- Generic abstractions
  - Datatypes and modules with type parameters
  - Design of STL

## Concepts in OO programming

- Four main language ideas
  - Encapsulation
  - Dynamic lookup
  - Subtyping
  - Inheritance
- Why OOP ?
  - Extensible abstractions; separate interface from impl
- Program organization
  - Work queue, geometry program, design

## Simula 67

- First object-oriented language
- Designed for simulation
  - Later recognized as general-purpose prog language
- Extension of Algol 60
- Standardized as Simula (no “67”) in 1977
- Inspiration to many later designers
  - Smalltalk
  - C++
  -

## Simula Summary

- Class: “procedure” that returns ptr to activation record
- Objects: closure created by a class
- Encapsulation
  - used as basis for C++
- Subtyping: determined by class hierarchy
- Inheritance: provided by class prefixing
- Object access
  - Access any local variable or procedures using dot notation
- Memory management
  - Objects garbage collected

## Smalltalk

- Major language that popularized objects
- Object metaphor extended and refined
  - Used some ideas from Simula, but very different lang
  - Everything is an object, even a class
  - All operations are “messages to objects”
  - Very flexible and powerful language
    - Similar to “everything is a list” in Lisp, but more so
- Method dictionary and lookup procedure
  - Run-time search; no static type system

## Smalltalk Summary

- Class
  - creates objects that share methods
  - pointers to template, dictionary, parent class
- Objects: created by a class, contains instance variables
- Encapsulation
  - methods public, instance variables hidden
- Subtyping: implicit, no static type system
- Inheritance: subclasses, self, super

## C++ Summary

- Objects
  - Created by classes
  - Contain member data and pointer to class
- Classes: virtual function table
- Inheritance
  - Public and private base classes, multiple inheritance
- Subtyping: Occurs with public base classes only
- Encapsulation

## Function subtyping

- Subtyping principle
  - $A <: B$  if an A expression can be safely used in any context where a B expression is required
- Subtyping for function results
  - If  $A <: B$ , then  $C \rightarrow A <: C \rightarrow B$
- Subtyping for function arguments
  - If  $A <: B$ , then  $B \rightarrow C <: A \rightarrow C$
- Terminology
  - Covariance:  $A <: B$  implies  $F(A) <: F(B)$
  - Contravariance:  $A <: B$  implies  $F(B) <:$

## Java Summary

- Objects
  - have fields and methods
  - alloc on heap, access by pointer, garbage collected
- Classes
  - Public, Private, Protected, Package (not exactly C++)
  - Can have static (class) members
  - Constructors and finalize methods
- Inheritance

## Java Summary (II)

- Subtyping
  - Determined from inheritance hierarchy
  - Class may implement multiple interfaces
- Virtual machine
  - Load bytecode for classes at run time
  - Verifier checks bytecode
  - Interpreter also makes run-time checks
    - type casts
    - array bounds
    - ...

## Concurrency

- Concurrent programming requires
  - Ability to create processes (threads)
  - Communication
  - Synchronization
  - Attention to atomicity
    - What if one process stops in a bad state, another continues?
- Language support
  - Synchronous communication
  - Semaphore: list of waiting processes

## Concurrency (II)

- Actors, Cobegin / Coend
- Java language
  - Threads: objects from subclass of Thread
  - Communication: shared variables, method calls
  - Synchronization: every object has a lock
  - Atomicity: no explicit support for rollback
- Java memory model
  - Compiler optimizations...

## Good Luck!

- Think about main points of course
  - Homework made you think about certain details
  - What's the big picture?
  - Look at homework and sample exams
    - Some final exam problems will resemble homework
    - Some may ask you to use what you learned in this course to understand language combinations or features we did not talk about

