CS 242

Review

John Mitchell

Final Exam
Wednesday Dec 8
8:30 – 11:30 AM
Gates B01, B03

Thanks!

- Teaching Assistants
  - Mike Cammarano
  - TJ Giuli
  - Hendra Tjahayadi
- Graders
  - Andrew Adams
  - Kenny Lau
  - Vishal Patel
  - Tait Larson
  - Aman Naimat
  - Justin Pettit
  - and more …

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

There are many programming languages

- Early languages
  - Fortran, Cobol, APL, ...
- Algol family
  - Algol 60, Algol 68, Pascal, …, PL/1, … Clu, Ada, Modula, Cedar/Mesa, ...
- Functional languages
  - Lisp, FP, SASL, ML, Miranda, Haskell, Scheme, Setl, ...
- Object-oriented languages
  - Smalltalk, Self, Cecil, …
  - Modula-3, Eiffel, Sather, …
  - C++, Objective C, …. Java

- Concurrent languages
  - Actors, Occam, ...
  - Pai-Lisp, …
- Proprietary and special purpose languages
  - TCL, Applescript, Telescript, ...
  - Postscript, Latex, RTF, …
  - Domain-specific language
- Specification languages
  - CORBA IDL, ...
  - Z, VDM, LOTOS, VHDL, …

General Themes in this Course

- Language provides an abstract view of machine
  - We don't see registers, length of instruction, etc.
- The right language can make a problem easy; wrong language can make a problem hard
  - Could have said a lot more about this
- Language design is full of difficult trade-offs
  - Expressiveness vs efficiency, ...
  - Important to decide what the language is for
  - Every feature requires implementation data structures and algorithms

Good languages designed with specific goals (often an intended application)

- C: systems programming
- 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
- Java: set-top box, internet programming

A good language design presents abstract machine, an idealized view of computer

- Lisp: cons cells, read-eval-print loop
- FP: ??
- ML: functions are basic control structure, memory model includes closures and reference cells
- C: the underlying machine + abstractions
- Simula: activation records and stack; object references
- Smalltalk: objects and methods
- C++: ??
- Java: Java virtual machine

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
  - preserves the integrity of abstract machine

- In general, high-level languages/features are:
  - slower than lower-level languages
    - C slower than assembly
    - C++ slower than C
    - Java slower than C++
  - provide for programs that would be difficult/impossible otherwise
    - Microsoft Word in assembly language?
    - Extensible virtual environment without objects?
    - Concurrency control without semaphores or monitors?

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

- Halting problem
- nil 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

Languages are still evolving

- Object systems
- Adoption of garbage collection
- Concurrency primitives; abstract view of concurrent systems
- Domain-specific languages
- Network programming
- Aspect-oriented programming and many other “fads”
  - Every good idea is a fad until is sticks

Summary of the course

- Lisp, 1960
- Fundamentals
  - lambda calculus
  - denotational semantics
  - functional prog
- ML and type systems
- Block structure and activation records
- Exceptions and continuations
- Modularity and objects
  - encapsulation
  - dynamic lookup
  - subtyping
  - inheritance
- Simula and Smalltalk
- C++
- Java
- Concurrency

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.

Fundamentals

- Grammars, parsing
- Lambda calculus
- Denotational semantics
- Functional vs. Imperative Programming
  - Is implicit parallelism a good idea?
  - Is implicit *anything* a good idea?

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
  - Exceptions

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
- Overloading
  - Symbol with multiple meanings, resolved at compile time

Type inference algorithm

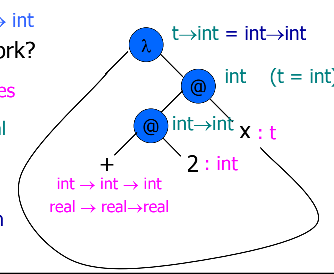
- Example
  - fun f(x) = 2+x;
  - > val it = fn : int → int
- How does this work?

Assign types to leaves

Propagate to internal nodes and generate constraints

Solve by substitution

Graph for λx. ((plus 2) x)



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
  - language expressiveness => implementation complexity

Summary of scope issues

- Block-structured lang uses stack of activ 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 call
  - Stack deallocation may fail if function returned from call
  - Closures *not* needed if functions not in nested blocks

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

- Step-wise refinement and modularity
  - History of software design
- 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
- Compare oo to conventional (non-oo) lang
  - Can represent encapsulation and dynamic lookup
  - Need inheritance and subtyping as basic constructs

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++
  - ...

Objects in Simula

- Class
  - A procedure that returns a pointer to its activation record
- Object
  - Activation record produced by call to a class
- Object access
  - Access any local variable or procedures using dot notation: object.
- Memory management
  - Objects are garbage collected
  - Simula Begin pg 48-49: user destructors undesirable

Smalltalk

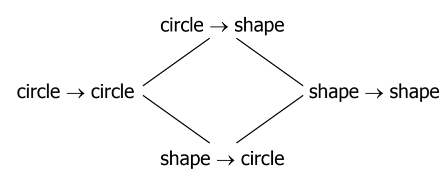
- Major language that popularized objects
- Developed at Xerox PARC 1970's (Smalltalk-80)
- Object metaphor extended and refined
  - Used some ideas from Simula, but very different lang
  - Everything is an object, even a class
  - All operations are "messages to objects"
  - Very flexible and powerful language
    - Similar to "everything is a list" in Lisp, but more so
- Method dictionary and lookup procedure
  - Run-time search; no static type system
- Independent subtyping and inheritance

C++

- Design Principles: Goals, Constraints
- Object-oriented features
  - Some good decisions, some problem areas
- Classes, Inheritance and Implementation
  - Base class and Derived class (inheritance)
  - Run-time structures: offset known at compile time
- Subtyping
  - Subtyping principles
  - Abstract base classes
  - Specializing types of public members
- Multiple Inheritance

Examples

- If circle <: shape, then



C++ compilers recognize limited forms of function subtyping

Subtyping with functions

```
class Point {
  public:
    int getX();
    virtual Point *move(int);
  protected:    ...
  private:      ...
};
```

```
class ColorPoint: public Point {
  public:
    int getX();
    int getColor();
    ColorPoint * move(int);
    void darken(int);
  protected:   ...
  private:     ...
};
```

Inherited, but repeated here for clarity

- In principle: can have ColorPoint <: Point
- In practice: some compilers allow, others have not

This is covariant case; contravariance is another story

Java function subtyping

- Signature Conformance
  - Subclass method signatures must conform to those of superclass
- Argument types, Return type, Exceptions:

  How much conformance is really needed?
- Java rule
  - Arguments and returns must have identical types, may remove exceptions

vtable for Multiple Inheritance

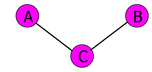
```
class A {
   public:
      int x;
      virtual void f();
};
class B {
   public:
      int y;
      virtual void g();
      virtual void f();
};
```

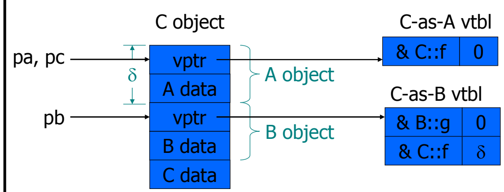
```
class C: public A, public B {
   public:
      int z;
      virtual void f();
};

C *pc = new C;
B *pb = pc;
A *pa = pc;
```

Three pointers to same object, but different static types.

Object and classes





- Offset δ in vtbl is used in call to pb->f, since C::f may refer to A data that is above the pointer pb
- Call to pc->g can proceed through C-as-B vtbl

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
  - Single inheritance
  - Final classes and methods

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
    - …
  - Portability and security are main considerations

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
  - Monitor: synchronized access to private data

Concurrency (II)

- Actors
  - Simple object-based metaphor
- Concurrent ML
  - Threads, synchronous communication, events
- 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
  - Separate cache for each thread; coherence issues

Good Luck!

- Think about main points of course
  - Homework made you think about certain details
  - What's the big picture?
  - What would you like to remember 5 years from now?
  - 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
- I hope course will be useful to you in the future
  - Send me email in 1 year, 2 years, 5 years