1 Intro

Class 1, 8/24/2017

- Handout 1 — My names
  - Mr. / Dr. / Professor / —
  - Raphael / Rafi / Refoyl
  - Finkel / Goldstein

- Extra 5 minutes on every class? What is a good ending time?
- Plagiarism — read aloud from handout 1
- Assignments on web and in handout 1.
- E-mail list: cs655401@cs.uky.edu; instructor uses to reach students.
- All students will have MultiLab accounts, although you may use any computer you like to do assignments. But your programs must run on MultiLab computers, because that’s how they will be graded.
- textbook — all homework comes from here
- Oral assignments are end-of-chapter assignments. I’ll announce the first one on Tuesday.

2 Software tools

A programming language is an example of a software tool.
3 McLennan’s Principles (elicit first)

4 Algol-like languages: review

- First generation: Fortran
  - constructs based on hardware
  - lexical: linear list of statements
  - control: sequence, goto, do, subroutines using reference parameter mode
  - data: arithmetic, including complex; arrays with a 3-d limit
  - name: separate scopes; common area

- Second generation: Algol 60
  - lexical: free format; keywords
  - control: nested; if, while, for, but baroque; subroutines with value and name parameter modes.
  - data: generalized arrays, but no complex
  - name: nested scopes with inheritance and local override

- Third generation: Pascal (return to simplicity)
  - data: user-defined types; records, enumerations, pointers
  - control: subroutines with value and reference parameter modes; case statement

- Fourth generation: Ada (abstract data types)
  - lexical: bracketed syntax
  - name: modules with controlled export; generic modules
  - control: concurrency with rendezvous

- Fifth generation: Other directions
• dataflow
• functional. We will study ML and Lisp.
• object-oriented. We will study Smalltalk.
• declarative (logic). We will study Prolog.

5 Block structure

• Introduced in Algol.
• A block is a nestable name scope.
• Identifiers can be local, nonlocal, or global with respect to a block.
• Nonlocal identifiers: the language must define whether to
  • inherit (typically allowed if there is no conflict)
  • override (typically true if there is a conflict)
  • require explicit import and export
• At elaboration time, constants get values, dynamic-sized types are bound to their size, space is allocated for variables.
• Definition: the non-local referencing environment (NLRE) of a procedure or block of code is the binding of non-local identifiers (typically variables, but also constants, types, procedures, and labels) to values.
• Deep binding: The NLRE of P is determined (bound) at the time that P is elaborated (and is the RE of the elaborating scope).
• Shallow binding: The NLRE of P is determined at the time that P is invoked (and is the RE of the calling scope).

6 Theme: binding time

• There is a range from early to late.
  • language-definition time (example: the fact that constants exist)
• compile time (example: values of constants in Pascal) We call compile-time bindings \textit{static}.
• link time (example: version of \texttt{printf} in C)
• elaboration time (example: value of \texttt{final int} in Java)
• statement-execution time (example: value of \texttt{int} variable) We call execution-time bindings \textit{dynamic}.

• Early binding is most efficient.
• Late binding is most capable.

7 \textbf{Imperative languages}

• Imperative languages involve statements that modify the current state by changing the values of variables.

• \textbf{A variable} is an identifier bound (usually statically) to a type, having a value that can change over time. The \textbf{L-value} of a variable is the use of a variable on the left side of an assignment (think of “address”); the \textbf{R-value} of a variable is its use on the right side (think of “current value”).

• \textbf{A type} is a set of values, associated (mostly statically) with operations defined on those values. Type \textbf{conversion} means expressing a value of one type as a value of another type.

  • \textit{coercion}: implicit conversion
  • \textit{cast}: explicit conversion
  • \textit{non-converting cast}: rarely needed. \texttt{qua} operator of Wisconsin Modula, \texttt{reinterpret_cast<>} of C++.

• An \textbf{operation} is a function or an operator symbol as shorthand. It can be heterogeneous.

  • operators have \textbf{arity} (example: unary, binary), precedence, associativity
  • operators may be \textbf{infix} (+), \textbf{prefix} (unary –), \textbf{postfix} (\texttt{->})
  • operators may have short-circuit (lazy) semantics
• An operation is overloaded if its identifier or operator symbol has multiple visible definitions. Overloading is resolved (usually statically) by arity, operand types, and return type. Overloading resolution can be exponentially expensive. For instance, say we have four versions of +, depending on whether they take integers/floats and whether they return integers/floats. Then how do you resolve \((a + b) + (c + d)\)?

• A primitive type (or basic type) has no separately accessible components. Examples: integer, character, real, Boolean.

• A structured type has separately accessible components. Examples: pointer (dereference), record (field select), array (subscript), disjoint union (variant select). An associative array is an array whose index type is string.

• A constant is like a variable, but it has no L-value and an unchanging R-value. In Java, it’s denoted by the modifier final.

8 Iterators

• Exercises from Chapter 1.

• Iterators allow us to generalize for loops.

  • The control variable of the for loop ranges over a set of values generated piecemeal by an iterator. [book 39:9-10].
  • The iterator is like a procedure, taking parameters and returning values of a specified type.
  • The iterator uses a yield statement to return a value, but it maintains its RE (and its program counter) in order to continue on demand from the for loop.
  • A useful language-supplied iterator is \(\text{int\ upto}(\text{low, high})\), which yields all the values in the specified range.

• Iterators are especially useful for generating combinatorial structures.

  • Algorithm for generating all binary trees of \(n\) nodes: [book 41:11]
  • Same thing in Python, using “generators”:

\begin{verbatim}
1 def binGen(size):
2     if size > 0:
\end{verbatim}
for root in range(size):
    for left in binGen(root):
        for right in binGen(size - root - 1):
            yield("cons(" + left + "," + right + ")")
    else:
        yield "-"

for aTree in binGen(3):
    print aTree

• Trace of binGen(3).

Another example: yield all nodes in a tree (in pseudo-Python)
def treeNodes(tree):
    if tree != null:
        for element in treeNodes(tree.left):
            yield element
        yield tree.value
        for element in treeNodes(tree.right):
            yield element

Wouldn’t it be nice to have a yieldall construct:
def treeNodes(tree):
    if tree != null:
        yieldall treeNodes(tree.left):
        yield tree.value
        yieldall treeNodes(tree.right):

This construct might be able to use shortcuts to improve efficiency. JavaScript actually has it: yield*.

Another example: all combinations $C(n,k)$:
def comb(n,k,start):
    if k == 0:
        yield 
    elif k+start <= n:
        for rest in comb(n,k-1,start+1):
            yield str(start+1) + "," + rest
        for rest in comb(n,k,start+1):
yield rest

for result in comb(6, 3, 0):
    print result

9 Macro package to embed iterators in C

- Macros are `IterSTART`, `IterFOR`, `IterDONE`, `IterSUB`, `IterYIELD`.
- Usage: [book 48:14]
- Implementation
  - `setjump` and `longjmp` for linkage between `for` and the controlling iterator, between `yield` and its controlled loop.
  - Padding between stack frames to let `longjmp()` be called without harming frames higher on the stack. Three integers is enough in Linux on an i686.
  - A Helper routine to actually call the iterator and act as padding.
  - The top frame must be willing to assist in creating new frames.

10 Power loops

- How can you get a dynamic amount of `for`-loop nesting?
- Application: $n$ queens [book 57:29]
- Usual solution: single `for` loop with a recursive call.
- Cannot use that solution in Fortran, which does not allow recursion.
- Solution: Power loops. [book 57:28]
- Implementation: Only needs branches, no recursion. [book 59:31]
- How general is this facility?
- Do power loops violate principle 20?
11 IO

- Attempt to strip a programming language of all non-essential elements.
- What’s left: \texttt{goto} with parameters, hence formal-actual bindings.
- Parameters can be integer, anonymous function, or continuation.
- A function is passed by \texttt{closure}: pointer to code, pointer to NLRE.
- A \texttt{continuation} is a function whose parameters have already been bound. It is passed by a closure along with the parameters.
- Examples \[\text{book 50:15 and following}\]
- \[\text{Class 6, 9/12/2017}: \text{more examples}\]
- \[\text{Class 7, 9/14/2017}\]
- re-work the Io examples \[\text{io.txt}\]

12 End-of chapter exercises: Chapter 2

13 ML-Introduction

- \[\text{Class 8, 9/19/2017}\]
- This material is out of order, presented now so students can begin to work on the next programming assignment.
- ML is \texttt{functional}, but we are particularly interested in its type system, with these important aspects.
  - The language is \texttt{strongly typed}.
  - The compiler \texttt{infers} the types of identifiers if possible.
  - \texttt{Higher-order types} are easily represented.
  - Types can be \texttt{polymorphic}, that is, expressed with respect to type identifiers.
- Examples, from online file \[\text{examples.ml}\]
14 General coroutines

- Class 9, 9/26/2017
- Problem: binary-tree node-equality test in symmetric order
- Solution
  - Independently advance in each tree
  - Each coroutine has its own stack; main has its own stack.
  - All the stacks are joined via static-chain pointers into a cactus stack.
  - A scope must not exit until all its children exit, or we must use a reference count. This is an example of the dangling NLRE problem.

- Syntax (Simula 67)
  - Records have initialization code that may execute `detach`.
  - Another coroutine may resume it via explicit `call`.

- Solution in Python using ”generators”. See binEqual.txt from class web page.

15 Types

- A type is a property of an R-value or of an identifier that can hold R-values.
- The property consists of a set of values.
- Strong typing means the compiler
  - knows the type of every R-value and identifier
  - enforces type compatibility on assignment and formal-actual binding
    - compatible means type equivalent, a subtype, or convertible
- A subtype consists of a subset of the values
Assignment and formal-actual binding require a dynamic check.

Subtype examples: range of integers, subclass of class

subTypeVar := baseTypeVar

Type equivalence

structural equivalence: expand type to a canonical string representation; equivalence is string equality.

- lax: ignore field names, ignore array bounds, ignore index type, flatten records.
- pointers require that we handle recursive types (and still build finite strings)

name equivalence: reduce a type to an instance of a type constructor

- type constructors: array, pointerTo, enum, struct, derived.
- lax (declaration equivalence): multiple uses of one type constructor are equivalent

Implementing structural type equivalence: exercise 3.8 at end of chapter.

16 Dimensions

- book Figure 3.7
- Applies to reals.
- Can be embedded into a strong type system.

17 ML again

- Remaining examples, from online file examples.ml
- Class 9, 9/26/2017
- See web page on currying
- Class 10, 10/3/2017
- Curry plus example
E1. global environment
E2. plus = fn (x,y) => x + y | E1
E3. curry = fn f => fn a => fn b => f(a,b) | E2
E4. curryplus = curry plus =
  fn a => fn b => f(a,b) | f=plus | E3
E5. successor = curryplus 1 =
  fn b => f(a,b) | a=1 | E4
E5. successor 12 = f(a,b) | b=12 | E5
  = plus(a,b) = plus(1,12) = x + y | x=1, y=12 | E1
  = 1+12 = 13

18 Unusual first-class values

- A **first-class value** can be returned from a function. In languages with variables, a first-class value can be stored in a variable.
- A **second-class value** can be an actual parameter.
- A **third-class value** can be used “in its ordinary way”.
- Labels and procedures
  - Usually third class values; the “ordinary way” is in a goto statement or a procedure-call statement.
  - They could be second class. They must be passed as a closure. For a label, the closure includes the RE of its target, so the stack can be unwound to the right place. For a procedure, the closure includes its NLRE to resolve non-local references.
  - To make them first class, we still need to build a closure, which we can then store in a variable. But the lifetime of that variable might exceed the life of the RE stored in the closure. This is the dangling-NLRE problem. [book 76:11]
- To resolve the dangling-NLRE problem
  - Let it happen and call the program “erroneous”.
  - Prevent it from happening by restricting the language.
    - Don’t let labels or procedures be first-class: Pascal
    - Don’t let scopes nest, so there is no need for closures (for procedures; labels are still problematic): C
• Only allow top-level procedures (or labels) be first-class:
  Modula-2
• Let it happen and make it work: allocate all frames from the
  heap and use explicit release (reference counts suffice).

19 Haskell

• Haskell is a functional programming language much like ML, with
  additional features.
• Haskell uses indentation for grouping, much like Python. This de-
  sign makes programs compact, but harder to read (if routines are
  long) and very difficult for blind programmers to construct.
• All functions are fully curried.
• Haskell evaluates all expressions lazily.
• Examples online
• [Class 11, 10/10/2017]
• Finish examples

20 Lisp introduction

• Lisp is homoiconic: programs and data structures have the same
  form, so one can execute data and one can manipulate program.
• Lisp is functional (functions have no side effects, and there are no
  variables)
• Lisp has been very influential and was widely used in AI.
• Lisp by examples
• [Class 12, 10/17/2017]
• Assign Chapter 3 exercises.
• Lisp deep binding (in examples)
• The Lisp metacircular interpreter [book 135:31 ff], also on web page.
• Pure Lisp (a restriction for Assignment 3): no use of set, rplaca,
  rplacd, and a few other non-functional features. Although defun
does change the context (introducing new functions always has a
side effect), we allow it.
21 What does polymorphic mean?

People use the term **polymorphic** to mean various features.

- Static procedure overloading with compile-time resolution (Ada, Java). Here, the type of the procedure (its **signature**) determines whether it is a viable candidate for resolving the overloading.

- Dynamic method binding (Java, Smalltalk, C++ deferred binding). Here, the dynamic type (class) of the value (object) determines which procedure (method) to invoke.

- Types described by type identifiers (perhaps with the compiler inferring the types of values) (ML, Haskell). Here, the dynamic type constraints on the parameter and return value determine the effective type of the function.

- Passing an ADT as a parameter (Russell).

- Generic packages (Ada), templates (C++), generics (Java).

22 Can types be second or first class?

- Letting a type be second-class is a step toward polymorphism.
  - Second-class types allow generic packages (Ada), templates (C++).
  - However, we only allow types to be passed at compile time during generic-package instantiation.
  - One can restrict the range of the actual type (the “type of the type”) by a Java-like interface (or Haskell-like class).

- An attempt at first-class types: **dynamic**. Actually, this idea attempts to extend strong typing to dynamic types; it is not the same as making types first-class.

- Another attempt at first-class types: Russell.
  - But what Russell calls a “type” we would call an ADT (abstract data type), which is something like a Java class.
  - So a Russell actually introduces dynamic class definitions.
  - We will see dynamic class definitions in Smalltalk later.

- Java reflection: class **Class**. Example from Cycle Shop at Home.
23 The Lambda Calculus

- Mathematical foundation of ML and of Lisp.
- Three kinds of **term**
  - **identifier**, such as \( x \)
  - **abstraction**, such as \( \lambda x . (* x 2) \)
  - **application**, such as \( (f x) \)

- **Syntax of terms**
  - parentheses are optional; application and abstraction are left-associative, and application has a higher precedence.
  - Curried functions may be written without currying:
    \[
    (\lambda x . (\lambda y . (\lambda z . T))) = (\lambda x y z . T)
    \]

- **Class 14, 10/24/2017**
- Chapter 3 exercises
- Identifiers can be **free** or **bound** in terms.
- \( x \) is bound in \( \lambda x . T \).
- \( x \) is free in \( T \) if:
  - \( T \) is just \( x \)
  - \( T \) is \( (f p) \), and \( x \) is free in either \( f \) or \( p \).
  - \( T \) is \( (\lambda y . T) \), \( x \) is not \( y \), and \( x \) is free in \( T \).
  - **Examples:** book 153:49

- **Simplification by \( \beta \) reduction:** book 153:50
  - applicative order: evaluate inner \( \lambda \) first.
  - normal order: evaluate outer \( \lambda \) first.
  - Church-Rosser theorem: if \( T \rightarrow S \) and \( T \rightarrow R \), then there is some \( U \) such that \( S \rightarrow U \) and \( R \rightarrow U \)

- **Renaming by \( \alpha \) conversion:** book 153:52

- **Normal form**
  - \( \beta \) reduction until no more reduction is possible.
  - It is not always possible to reduce to a normal form: book 155:55
• **Combinator**: term with no free identifiers. Important combinator \( Y \):
\[
Y = (\lambda f . (\lambda x . f (x x)) (\lambda x . f (x x)))
\]

• Simplification by \( \eta \) reduction: \( (\lambda x . F x) \xrightarrow{\eta} F \)

• **Class 15, 10/26/2017**

• Connection between Lambda Calculus and ML. **book 156:60**

## 24 Smalltalk

• Examples (online)

• **Class 16, 10/31/2017**

• More examples (online)

• **Class 17, 11/2/2017**

• The two hierarchies of classes in Smalltalk
  
  • `class` gives the is-a hierarchy.
  
  • `superclass` gives the subclass-of hierarchy.

## 25 Intellectual history of object oriented programming

• Records in Cobol, Pascal. Fields are variables, always visible.

• Records (called classes) in Simula67. Fields can also be procedures. Their NLRE is the record in which they sit. (Already some object-orientation: Subclasses inherit fields with possibility of override, and there is some control over visibility.)

• Abstract data types (ADTs) in languages like CLU (“clusters”), Modula (“modules”), Ada (“packages”). Separation of specification from implementation. Typically ADTs export a type and some operations. Clients then create instances of that type, either on the stack as local variables or from the heap.

• Monitors, which are ADTs with concurrency control. Unfortunately, concurrency control protects program, not data. We’ll say more about monitors later.
• Classes (Smalltalk, C++, Java). Do not export types, only variables and procedures. The entire class becomes a type.

26 Object-oriented programming: Overview

• Nomenclature: Objects (values) are instances of classes (types). They communicate by sending messages (procedure calls) to each other to invoke methods (procedures). The state of an object is defined by the values of its instance variables. The set of methods that instances of a class accepts constitute its protocol. Together, the methods and the instance variables are called the members of a class.

• Data encapsulation: One may only affect the state of an object by invoking its methods. (Doesn’t hold if instance variables are exposed.)

• Inheritance: Subclasses (a form of subtype) inherit the members of their superclass. Programmers introduce subclasses either to specialize or to reuse code.

• Overloading: Methods with the same name may appear in multiple classes; the meaning of the method invoked on an object depends on the class to which that object belongs.

• Deferred binding: Binding a method invocation to a specific method can happen at invocation time (deferred binding: Java, Smalltalk) or at compile time (C++). In typed languages, compile time works well, but for subclasses that redefine a method, it can give unwanted results.
## 27 Control over information hiding

<table>
<thead>
<tr>
<th>language</th>
<th>mode</th>
<th>other instance of same class</th>
<th>related (^1)</th>
<th>inherited by subclass</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smalltalk</td>
<td>variable</td>
<td>n</td>
<td>-</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>method</td>
<td>y</td>
<td>-</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>C++/Java</td>
<td>public</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>C++/Java</td>
<td>protected</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>C++/Java</td>
<td>private</td>
<td>y</td>
<td>y/n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>Java</td>
<td>package-private</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>Eiffel</td>
<td>(default)</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>Eiffel</td>
<td>specified</td>
<td>if related</td>
<td>y</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>Eiffel</td>
<td>none</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
</tr>
</tbody>
</table>

\(^1\)"related" means friend class (C++), class in the same package (Java), or class to which a member is explicitly exported (Eiffel).