COOL: The Language

Adapted from material by:
Prof. Alex Aiken  (UCB)

Cool Overview
- Classroom Object Oriented Language
- Designed to
  - Be implementable as a course project in one semester / two quarters.
  - Give a taste of implementation of a modern programming language with
    - Abstraction and Encapsulation
    - Strong typing, Static typing
    - Reuse (single inheritance)
    - Dynamic Dispatch
    - Automatic Memory management
- But leaves out many features of a production language, for tractability.

A Simple Example
- Cool programs are sets of class definitions.
  - A special class Main with a special method main().
  - No separate notion of a subroutine.
- A class is a collection of attributes and methods.
- Instances of a class are objects.

Cool Objects
- The expression "new Point" creates a new object of class Point.
- An object can be thought of as a record with a slot for each attribute.
  - The expression "new Point" creates a new object of class Point.
  - An object can be thought of as a record with a slot for each attribute.

```
class Point {
    x : Int ← 0;
    y : Int ← 0;
};
```

```
class Point {
    x : Int ← 0;
    y : Int ← 0;
    (* use default value *)
};
```

```
0 0
```
Methods

- A class can also define methods for manipulating attributes.

```plaintext
class Point {
  x : Int ← 0;
y : Int ← 0;
movePoint(newx : Int, newy : Int): Point {
  x ← newx;
y ← newy;
  self;
} -- close block expression
}; -- close class
```

- Methods can refer to the current object using `self`.

Information Hiding in Cool

- Methods are `global`. (Cf. `public`)
- Attributes are `local` to a class. (Cf. `private`, `protected`)
- They can only be accessed by the class’s methods.

```plaintext
class Point {
  . .
x () : Int { x }
  setx (newx : Int) : Int { x ← newx }
};
```

Methods

- Each object knows how to access the code of a method. It is as if the object contains a slot pointing to the code.

```plaintext
x  y  movePoint
0  0  *
```

- In reality, implementations save space by sharing these pointers among instances of the same class.

```plaintext
x  y  methods
0  0  
movePoint  *
```

Inheritance

- We can extend points to colored points using subclassing => class hierarchy.

```plaintext
class ColorPoint inherits Point {
  color : Int ← 0;
movePoint (newx : Int, newy : Int): Point {
  { color ← 0;
    x ← newx; y ← newy;
    self;
  }
};
```
Cool Types

- Every class is a type.
- Base classes:
  - Int for integers
  - Bool for boolean values: true, false
  - String for strings
  - Object root of the class hierarchy
- All variables must be declared.
  - Compiler infers types for expressions.

Cool Type Checking

- Is well typed if A is an ancestor of B in the class hierarchy.
  - Anywhere an A instance is expected a B instance can be used. (Cf. polymorphism)
- Type safety:
  - A well-typed program cannot result in runtime type errors. (Cf. Static type checking)

Method Invocation and Inheritance

- Methods are invoked by dispatch.
- Understanding dispatch in the presence of inheritance is a subtle aspect of OO languages.

```
p : Point;
p ← new ColorPoint;
p.movePoint(1,2);
```

- p has static type Point.
- p has dynamic type ColorPoint.
- p.movePoint must invoke the ColorPoint version.

Method Invocation

- Example: invoke one-argument method m(x)

```
x : A;
x ← new B;
```

- 1. Eval. e
- 2. Find class of e
- 3. Find code of m
- 4. Eval. arg. e'
- 5. Bind self and x
- 6. Run method
Other Expressions

- Expression language (every expression has a type and a value)
  - Loops: \(\text{while } E \text{ loop } E \text{ pool}\)
  - Conditionals: \(\text{if } E \text{ then } E \text{ else } E \text{ fi}\)
  - Case statement: \(\text{case } E \text{ of } x : \text{Type } \Rightarrow E; \ldots \text{ esac}\)
  - Arithmetic, logical operations:
    - Assignment: \(x \leftarrow E\)
    - Primitive I/O: \(\text{out\_string}(s), \text{in\_string}(), \ldots\)
- Missing features:
  - Arrays, Floating point operations, Interfaces, Exceptions, …

Cool Memory Management

- Memory is allocated every time \textit{new} is invoked.
- Memory is deallocated automatically when an object is not reachable anymore.
  - Done by the garbage collector (GC).
  - There is a Cool GC.
  - (Cf. Java, C++)
- Portion of the Run-time System

A complete COOL program

```cooll
class Main inherits IO {
  pal(s : String) : Bool {
    (*...expression...*)
  }
}
main() : SELF_TYPE {
  (*...statements...*)
};
```
pal(s : String) : Bool {  
  if s.length() = 0  
    then true  
  else if s.length() = 1  
    then true  
  else if s.substr(0, 1) = s.substr(s.length() - 1, 1)  
    then pal(s.substr(1, s.length() - 2))  
  else false  
  fi  
  fi  
  fi 
};

main() : SELF_TYPE {  
  {  
    out_string("enter a string → ");  
    if pal(in_string())  
      then out_string("that was a palindrome
")  
    else out_string("that was not a palindrome
")  
    fi;  
  }  
};

About COOL

• The language is defined in the document COOLAID: The Cool Reference Manual.
  ➢ Lexical Structure
  ➢ Cool Syntax
  ➢ Cool Semantics
    ➢ Static: Type System
    ➢ Dynamic: Operational Approach
  ➢ Runtime system

• Some of the support code for Project 3 (and beyond if you take 781) is described in A Tour of Cool Support Code.

Cool Installation at WSU

• Cool is installed on gandalf.cs.wright.edu in the directory: /usr/local/lib/cool
  ➢ Add the following to your PATH to access Cool executables: /usr/local/lib/cool/bin
  ➢ There are several example Cool programs in the directory: /usr/local/lib/cool/examples

• To compile a Cool program type:
  
  coolc <filename.cl>

  The compiler produces MIPS assembly code.

• To “execute” the program use the SPIM simulator:
  
  spim -file <filename.m>