CS 164 Programming Language and Compilers **Fall 2015**

Discussion 1: Cool

TAs: Melanie Cebula, Ben Mehne, and Matt Miller

- **Announcement**
  - PA1 due next[, next] Tuesday (9/8) at 11:59pm.
  - WA1/PA2 out next, next Wednesday (9/9).
  - Please contact GSI Ben Mehne to get a class account ASAP if you don't have one.
- **Outline**
  1. Understanding **Cool Syntax, Semantics, and Types**
  2. PA1 QA

Cool

Cool is a Classroom Object-Oriented Language - designed to both be powerful and easy to compile. This discussion will focus on some of the difficult parts of the language - all material discussed here is discussed in more depth in the Cool Manual.

1 Cool Syntax

Which of the following are syntactically valid?

1. `a = b = c`
   Answer: No - the equals sign is not associative and, hence, the above is ambiguous (are we comparing to see if a is equal to the result of (b=c) or c is equal to the result of (a=b)?)

2. `(* (* nested *) comments are (*in?*)valid*)`
   Answer: The above is syntactically valid - comments can be nested.

3. ```
   class A {};
   class B { b: Int; };
   class C { bar():Int { 0 }; };
   ```
   Answer: Yes - empty classes are permitable, and classes B and C are correct.

4. ```
   class Main inherits IO {
       Foobar(a:Int):Int {
           a+1
       };

       main() : SELF_TYPE {
           out_int(0)
       };
   };
   ```
   Answer: No - methods cannot begin with a capital letter (only types/classes can and must).

2 Cool Semantics

Cool is designed to be similar to other object-oriented languages and supports inheritance, garbage collection, and static typing.

Cool is also an expression language - most Cool constructs are expressions. Every expression has a value and type. Cool is made up of classes, which are made up of features (attributes/class variables and methods). Each method is made up of expressions (as opposed to statements as in some other languages, like Java).

2.1 Let

1. What is the purpose of let expressions?
   Answer: Let expressions are used to define the value of a variable (and declare it) for the evaluation of an expression

2. Can a let expression "redefine" a variable?
   Answer: Yes - the identifier that a let expression introduces can shadow an existing identifier.

3. Is this a valid expression in Cool? If not, why not? If it is valid, what does it evaluate to?
   `let z : Int <- 0, z : Int <- z+1 in z`
   Answer: It is a valid expression and it evaluates to 1.

2.2 Loops

1. Are loops expressions?
   Answer: Yes

2. What is the type of an evaluated loop?
   Answer: Object

3. What is the value of a loop after it is completed?
   Answer: void

2.3 Blocks

1. What is the value and type of a block expression?
   Answer: A block's value and type is that of the last expression in the block.

2. Why do blocks exist/why would you need to use them?
   Answer: Blocks are used when multiple expressions need to be evaluated. Multiple expressions only need to be evaluated when those expressions modify the value of externally-visible variables.

2.4 Case

1. How would you implement Cool's `case` in other languages?
   Answer: In C++, dynamic_cast with a try-catch; in Java, instanceof with if-statements and casting; and in Python, isinstance with if-statements (most languages have some means of switching on type) - there are more solutions for C++, Java, and Python than listed here.

2. How else can you control the flow of a program by the dynamic (as opposed to the static) type of an expression?
   Answer: Using inheritance and (dynamic) method dispatch, the runtime type can determine which method is called.

3. Why might you pick `case` over another method of control evaluation based on dynamic type?
   Answer: Case does not require the classes involved to be modified.

4. What are the runtime errors that case introduces?
   Answer: The case statement may not have a sufficient case to handle the dynamic type. The case statement may be evaluated on an expression that evaluates to void - no type information can be safely gained there.

3 Cool Types

Cool is a stronlgy typed, object-oriented language with inheritance - it supports the definition of new types in the form of objects and every variable has a type determined at compile time.

1. Is the following program valid? If so, what does it output? If not, why not?

```
class A {
    foobar(value:A):Int {
        1
   };
};

class B inherits A {
    foobar(value:B):Int {
        2
    };
};

class Main inherits IO {
    main() : SELF_TYPE
    {
        let z:A <- new B in out_int(z.foobar(z))
    };
};
```

   Answer: The program is not valid - the method foobar has the same name in B as in its parent class A, but it does not have the same argument type.

2. What is the type of the following expression (assuming no runtime error is observed)?

```
case <expr> of
    v : Int => v;
    v : String => v;
esac
```

   Answer: Object - the least type that is the parent of both Int and String is Object.

Further Readings

- Cool Manual (available on the course page)