Welcome to cs655 - 801 (Spring-2020)

Seyedamirhossein Hesamian (Amir)

- Please send email questions to:
  - boyland@uwm.edu
  - rodenbe4@uwm.edu
  - hesamian@uwm.edu

- Office hour: TBD
- Room: EMS 9th floor computer lab
- Slides: [github.com/amir734jj/cs655-lecture-notes] (https://github.com/amir734jj/cs655-lecture-notes)

Compiler Steps (according to textbook)

- Front-end:
  - Lexical analysis: Tokenizing
  - Syntax analysis: Parsing
  - Semantic analysis
    - Name resolution, binding, type-checking
    - Optimize AST
    - Intermediate code generation
- Back-end
  - Code generation
    - Machine independent code generation
    - Target code generation
    - Machine specific code generation

Essential Concepts

- AST
- BNF / E-BNF (Option, Repetition, Grouping, Concatination)
- Visitor pattern

Reading Recommendations

- Cool manual
- Textbook
- optional textbook "Dragon Book"

Cool

Subset of Scala

Cool or "Classroom Object Oriented Language" is a:

- static (not dynamic): types are determined at the compile-time as oppose to run-time
- strong (not weak): there *are* restrictions for type conversions
- manifest (not inferred): variable types are explicitly defined as oppose to implicit

Cool (Cont.)

More about "static" aspect.

Types are defined (or deduced) in AST (or Abstract Syntax Tree) before code is generated. In dynamic languages like Python, JavaScript types are derived at the run-time hence, REPL (or Read–Eval–Print-Loop).

Cool (Cont.)

More about "strong" aspect.

There is no pointer in Cool but we have reference type variables (Any, ArrayAny and etc.). All types extend `Any` (actually `Any` extends `Nothing` but that is a special type). Moreover, we can only extend one type in Cool unlike C++ multiple inheritance is not allowed.

Cool (Cont.)

More about pattern matching

Cool is strongly typed because we can only type convert between types that are possible. For example, in the following we should **not** be able to do pattern matching from `C` to `String` (in one step):

```
Nothing -> Any -> | A -> B -> C
                  | String
                  | ArrayAny
```

Exercise: How to break this type system restriction?

Cool (Cont.)

More about manifest.

Unlike Scala which is a super-set of Cool, we have be explict about types. Scala comes with "duck" typing.

```
var i : Int = 234;
```

Cool syntax vs Java (part #1)

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

```
// We need to extend IO to be able to use `out`
class Main() extends IO() {
   // This is a expression "block"
   // All block expressions that are "feature" (or root level) run after primary constructor is invoked
   {
        out("Hello, World");
   }
}
```

Cool syntax vs Java (part #2)

```java
public class HelloWorld {
    public static int factorial(int n) {
        return n == 0 ? 1 : n * factorial(n - 1);
    }

    public static void main(String[] args) {
        System.out.println("result: " + factorial(10));
    }
}
```

```
class Main() extends IO() {
    // self keyword is similar to `this`, it's a way to access class scope
    // == is a syntax sugar for .equals() of Int which is an object
    def factorial(n : Int) = if (n == 0) 1 else self.factorial(n - 1);
    {
      // Notice how there is no implict conversion from Int to String unlike java
      out("result: ".concat(factorial(10).toString()));
    }
}
```

Vector

Very similar to `ArrayList` in Java but not thread-safe and resizes by doubling the size as oppose to increase the size by half.

- Vector
  - `size(): Int`
  - `add(Any): Unit`
  - `clear(): Unit`
  - `elements(): Enumeration` (i.e. return `VectorEnumeration`)
- Enumeration:
  - `next(): Any`
  - `hasNext(): Boolean`

Concerning `IO`

We will be using:

- `abort(): Nothing` // halts the program

Concerning `ArrayAny`

We will be using:

- `.get(Int)` : get array at index
- `.set(Int, Any)` : set array at index
- `.resize(Int)` : resizes the array`

TODO

- `class Enumeration() extends IO()`
- `class Vector()`
- `class VectorEnumeration(var elements: ArrayAny, var n: Int) extends Enumeration()`