CS224 – Properties of Identifiers

Purpose: Our programs are full of identifiers. These identifiers can represent many different things: methods, variables, etc. Therefore we need to attach *attributes* to these identifiers. Attributes include any information we want to *bind*, or associate with an identifier. For variable identifiers one important attribute is the *type* of the variable. There are many different ways programming languages can handle types and variables and we will examine these mechanisms

Knowledge: This activity will help you become familiar with the following content knowledge:

- The difference between static and dynamic binding
- The difference between static and dynamic type checking
- How programs can infer the type of variables

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 30 minutes.

1. An attribute is a property of an identifier. Can you determine at least four different attributes of the identifier `f` in the following code?

   ```
   public bool f(int n){
               return n==0;
   }
   ```

   Static binding is when an attribute is given to an identifier during compilation and is maintained in a **symbol table**.

   Dynamic binding is when an attribute is given to an identifier during execution of the program and is maintained in an **environment**.

   In the code above, give an attribute of `n` which is bound statically.
   Give an attribute of `n` which is bound dynamicaly.

2. An important attribute is the type of a variable. Languages can require that you specify or annotate the types of all variables. Alternatively languages can allow but not require type annotations or languages can even prohibit type annotations. Can you name a language that you have used that requires type annotations? Can you name a language that you have used that allows but does not require type annotations? Have you used a language where no type annotations are allowed?

3. Type variables or generic types can be used to achieve **polymorphism**. Polymorphism means that the variables can take on more than one type. For example, in Haskell the function `hd` can take a parameter which is a list of any type:

   ```
   hd :: [a] -> a
   ```

   How could we use type variables to describe the type of the function `first`?

   ```
   first (x,y) = x
   ```

4. Haskell is able to perform **type inference** to determine what types should be. For example given,

   ```
   fact n = if n==0 then 1 else n*(fact (n-1))
   ```

   Haskell would start by inferring that the function takes some type and returns another type, so `fact :: a -> b`

   Then from examining the code, Haskell can infer that the type variable `a` must be `int`. Why?
   How can Haskell infer that `b` is also `int`?

   Consider the code for the function `m`:

   ```
   m [] = []
   m (x:y) = (x,x) : (m y)
   ```

   Starting with `m :: [a]->[b]`, what can Haskell infer for the type `b`?

5. A **strongly typed** programming language checks all types statically (during compilation) wheras a weakly typed language does not. List some of the trade-offs between having a strongly typed language versus a weakly typed language for you, the programmer.