Daniel Frey
CS 3050-001
Assignment 7
12-7-18

1. **Define Lisp function** find **that takes 2 parameters: x and y. Returns x if x appears in y, returns an empty list otherwise.**

```
(defun find(x y)
  (cond
    ((null y) nil)
    ((equal x (car y)) x)
    ((consp (car y)) (or (find x (car y))
                         (find x (cdr y))))
    (t (find x (cdr y))))
)
```

```
[7]> (find 'x '(1 3 (x y)))
X
[8]> (find 3 '(1 3 (x y)))
3
[9]> (find 2 '(1 3 (x y)))
NIL
```

2. **Try to evaluate lambda terms to normal form.**

   a. **λx. λy. (λz. z + 1) y** → λx. λy. y + 1

   b. **(λf. λx. f(f x)) (λy. y + 1)** → λx. (λy. y + 1)((λy. y + 1) x) → λx. (λy. y + 1) x + 1 → λx. x + 1 + 1

   c. **(λx. λy. x y) y 3**
   Rename bound y:
   (λx. λw. x w) y 3 → (λw. y w) 3 → y 3

3. **Write ML program to merge and sort two lists in descending order without built-in functions.**

This works by merging two given lists together, then sorting the list. To sort the list, the list must be split up and merged back together in the correct order. It uses the functions merge, split, sort, and the wrapper for all of them merge_sort.

```
fun merge(x, []) = x
  | merge([], y) = y
  | merge(x::xl, y::yl) =
      if x > y
        then x::merge(xl, y::yl)
      else
        y::merge(x::xl, yl)

fun split [] = ([],[])
  | split [a] = ([a], [])
```

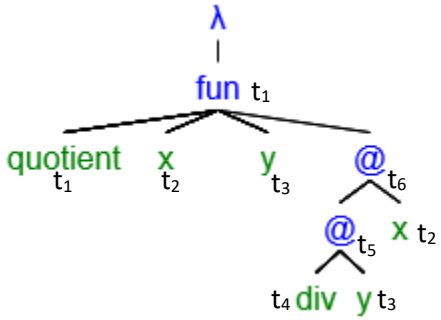
```
  | split (a::b::c) =
      let val (x, y) = split c
      in
         (a::x, b::y)
      end;

fun sort([]) = []
  | sort([x]) = [x]
  | sort(x) =
      let
         val (y, z) = split x
      in
         merge(sort(y), sort(z))
      end;

fun merge_sort([], []) = []
  | merge_sort(x, []) = sort(x)
  | merge_sort([], y) = sort(y)
  | merge_sort(x::xl, y::yl) =
      let
         val lst = merge(x::xl, y::yl)
      in
         sort(lst)
      end;
```

4. **Show the three steps followed in ML type-inference algorithm for the following example.**
   fun quotient(x, y) = x div y;

   1. Create parse tree and assign types to expressions.



Expression	Type
λ<x, y>. div x y	t_1
div x y	t_6
div y	t_5
x	t_2
div	t_4
y	t_3

   2. Generate constraints using parse tree.

   $t_1 = t_2 * t_3 \rightarrow t_6$

   $t_5 = t_2 \rightarrow t_6$

$t_4 = t_3 \rightarrow t_5$
$t_4 = Int \rightarrow (Int \rightarrow Int)$

3. Solve constraints.
$t_4 = t_3 \rightarrow t_5$
$t_4 = Int \rightarrow (Int \rightarrow Int)$
$t_3 \rightarrow t_5 = Int \rightarrow (Int \rightarrow Int)$
$\therefore \mathbf{t_3} = Int, \mathbf{t_5} = Int \rightarrow Int$

$t_5 = t_2 \rightarrow t_6$
$t_5 = Int \rightarrow Int$
$t_2 \rightarrow t_6 = Int \rightarrow Int$
$\therefore \mathbf{t_2} = Int, \mathbf{t_6} = Int$

$t_1 = t_2 * t_3 \rightarrow t_6$
$\therefore Int * Int \rightarrow Int$

5. **What is polymorphism, different types of polymorphism, explain each with simple program.**

Polymorphism is when something can have multiple forms. Objects may get processed differently depending on their data type or class.
Types of polymorphism: parametric (generic implementations), ad hoc (overloading), subtype (OOP, parent/child relation).

Examples:
Parametric:
fun swap(x, y) = (y, x)
val swap = fn : ('a * 'b) → ('b * 'a)

Ad Hoc:
1 + 2 = 3
1.0 + 2.0 = 3.0

Subtype:
class A {}
class B extends A {}
B b = new B();
A a = b;
(B is a subtype of class A, B <: A)

6. **Draw pictorial snapshot of runtime stack memory for the following ML code.**

```
let val x = ref 2;
   fun foo(y) = x := !x + y; y
in let val x = 5
   foo(x)
   end
end;
```

Run-time Stack

x	2
Control Link	
Access Link	
foo	x
x	!x + y
Control Link	
Access Link	
x	5
foo	foo(5)
Control Link	
Access Link	
y	5

7. **What is tail recursion? Convert function to tail recursion.**
   fun find(x, []) = 0 | find(x, y1::y2) if (x = y1) then 1 + find(x, y2) else find(x, y2);

   Tail recursion is when the recursive call is the last thing executed.

   Convert:

```
fun find(x, y) =
  let fun aux(x, y, acc) =
    if null y
      then acc
    else if x = hd y
      then aux(x, tl y, acc + 1)
    else
      aux(x, tl y, acc)
  in
    aux(x, y, 0)
  end;
```

8. **What is Object-Oriented Programming? Explain each of the OOP concepts.**

   Object-oriented programming is a programming paradigm centered around objects; attributes and their methods.

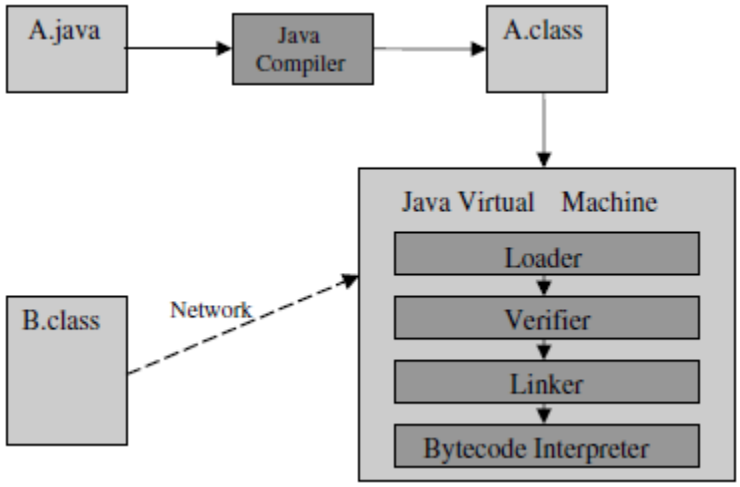
   Concepts: encapsulation, inheritance, polymorphism, and abstraction.
   Encapsulation is when the data and functions to manipulate the data are grouped together.
   Inheritance is the ability to reuse the definition of an object to define another object.
   Polymorphism is using one implementation that applies to multiple things.
   Abstraction is hiding the implementation details. Typically, hidden data that is manipulated by public functions.

9. **Write a Python program to find the number of times a given letter occurs in a string recursively.**

```
def countLetter(char, string):
  if not string:
```

```
    return 0
elif char == string[0]:
    return 1 + countLetter(char, string[1:])
else:
    return countLetter(char, string[1:])
```

10. **Draw the architecture of Java System. Briefly explain each part.**



Class loader will load the classes as they are needed if they are not already loaded.
Bytecode verifier checks to make sure that classes have the required properties.
Linker involves creating the static fields of classes or interfaces. This step also resolves names, as well as replacing symbolic references.
Bytecode interpreter executes bytecode and does runtime tests.