York University
CSE 2011 Summer 2015 – Midterm
Instructor: Jeff Edmonds

Family Name: _____ Given Name: _____

Student #: _____ CSE Email: _____

0) Art	2 marks	
1) Linked List	$27 = 6 + 6 + 3 + 3 + 3 + 6$	
2) Invariants	$19 = 3 + 9 + 3 + 4$	
3) Binary Search Tree	$15 = 3 + 3 + 9$	
4) 4 3 2 + 5 *	4	
5) Recursion	6	
6) First Order Logic	$9 = 3 + 3 + 3$	
7) BigO	$6 = 3 + 3$	
8) Running Time	$14 = 4 + 4 + 6$	
Total	102 marks	

This exam is designed to be completed in 2 hours. Students are expected to complete the exam in that period. If unable to so so, they may continue based on the discretion of the instructor. Keep your answers short and clear.

0 Art therapy question: When half done the exam, draw a picture of how you are feeling.

1. (71% in 2015)
   Consider a stack implemented using a singling linked list with a variable *top* pointing at the first node.

   (a) Imagine constructing a new stack and then pushing 1, 2, and then 3 onto it. Draw a picture of what the stack then looks like. Show where pointers are pointing both with arrows AND by giving actual values to the variables and showing in the drawing what these values represent.

   (b) Give code for $stack.push(int\ x)$ which pushes the value x onto this stack.

   (c) What special code is needed to handle the case when the element is being pushed onto an empty stack?

   (d) Give the bigO (or Theta) of the running time of this push operation as a function of the number n of nodes in the stack. Why?

   (e) Draw (in (a)) a picture of the stack after you have popped all three of these items. Again show both arrows and actual values.

   (f) Seeing the code, $x = 5;\ y = x;\ z = y;$, I can trace the history of the value 5. It was first assigned to the variable x. Then it was copied from x to y and then from y to z.

   After you have popped the last item off the stack, what value is stored in the variable *top*?

   Trace the history of this value backwards in time. In your above code, indicate where this value first came from and which lines of code copied it from one variable to the next until finally got to *top* as it is now.

2. (68% in 2015) Invariants:

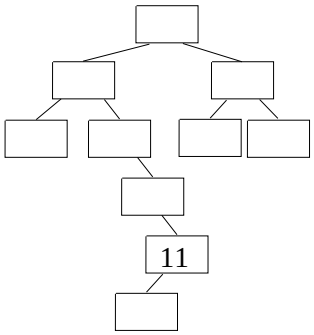
   (a) What is loop invariant?

   (b) Loop Invariant Steps: Your goal is to prove that no matter what the input is, as long as it meets the precondition, and no matter how many times your iterative algorithm iterates, as long as eventually the exit condition is met, then the post condition is guarantee to be achieved. According to Jeff's steps, what are three most important things to prove in order to accomplish this. Explain each in your own words. Also state each as an $A \Rightarrow B$ statement.

   (c) How is a system invariant same and different?

   (d) Give three or four sentences to explain how the system invariant is defined for the previous *Stack* question and how invariant steps are accomplished.

3. (60% in 2015) Consider the following **Binary Search Tree**.



(a) Starting from 11 label the nodes $10, 9, 8, \ldots$ in the order that they would be visited if you repeatedly called *previous* as coded in assignment 2 and $12, 13, 14, \ldots$ if you repeatedly called *next*.

(b) How do you know that value at node 11 is at most the value at node 12?

(c) Part of the instructions for the *next* routine was the following.

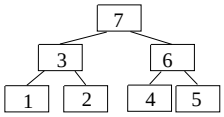
> If your current node does not have a right child, travel *up and left* as far as you can and then *up and right*. (Going "up and left" means testing of you are the right child of your parent and if so going to your parent.) Said another way follow the path up parent parent parent. Stop the first time that the node you came from is the left node of where you currently are.

For example, this should get you from node 11 to node 12.

Give the code for just this part of the assignment. No error correcting needed.

4. (74% in 2015)
   Read the string 4 3 2 + 5 *. When you see a number push it on the stack. When you see an operator, pop two numbers and perform the operation on them, and push the result. What is on the stack at the end?

5. (83% in 2015)
   Write a recursive program that takes as input a binary tree and prints out its nodes in the order shown in the following figure. What is the name of this order?



6. (46% in 2015)
Jeff feels that a major cause of students through to grad school not understanding computer science material is because they don't understand first order logic. Example:

$\forall x,\ \exists y,\ x + y = 0.$

This states that every number x has an additive inverse $y = -x$.

Jeff uses the following game to prove a first order logic statement. Read the expression left to right. If a variable is quantified with an exists ($\exists$), then the prover produces an object for that variable. If it quantified with a forall ($\forall$), then the adversary does. The prover wins the game, if the inner statement is true. The statement is true, if the prover can always win. The proof of our example is as follows.

Proof of $\forall x,\ \exists y,\ x + y = 0$:
Let x be an arbitrary real number.
Let y be $-x$.
Note that $x + y = x + (-x) = 0$.

A computational problem is a function P from inputs I to required output $P(I)$.
An algorithm is a function A from inputs I to actual output $A(I)$.

$A(I) = P(I)$ *and* $Time(A, I) \leq 3|I|^2$

states that algorithm A solves problem P on instance I in at most $3n^2$ time.

For each of the following statements, check the first order logic statement that expresses the statement. Hint: Consider (but do not write) the game played to prove that the statement is true.

(a) Problem P is computable.

i. $\forall I\ \exists A\ A(I) = P(I)$.
ii. $\forall A\ \exists I\ A(I) = P(I)$.
iii. $\exists I\ \forall A\ A(I) = P(I)$.
iv. $\exists A\ \forall I\ A(I) = P(I)$.

(b) Problem P is uncomputable.

i. $\forall I\ \exists A\ A(I) \neq P(I)$.
ii. $\forall A\ \exists I\ A(I) \neq P(I)$.
iii. $\exists I\ \forall A\ A(I) \neq P(I)$.
iv. $\exists A\ \forall I\ A(I) \neq P(I)$.

(c) Algorithm A's running time is $\mathcal{O}(1)$. (Ignore the n_0 part.)

i. $\forall I\ \exists c\ Time(A, I) \leq c$.
ii. $\forall c\ \exists I\ Time(A, I) \leq c$.
iii. $\exists I\ \forall c\ Time(A, I) \leq c$.
iv. $\exists c\ \forall I\ Time(A, I) \leq c$.

7. (47% in 2015)
Consider the function $f(n) = 9n^3 \log^{100}(n) + 5 \cdot n^{5.5} \log^2(n)(\sin(n^2) + 5)$.
What is $\Theta(f(n))$? Why?
Is it one of the following: $\Theta(1)$, $2^{\Theta(n)}$, $\log^{\Theta(1)}(n)$, or $n^{\Theta(1)}$. Why? What is this class called?

8. (44% in 2015)
Running Time: Give the Theta (bigO) of time complexity (running time) of each of these programs. Explain your work.

algorithm $Ex1(\langle x_1, x_2, \ldots, x_n \rangle)$

$\langle pre-cond \rangle$**:** x is an array of n integers.

$\langle post-cond \rangle$**:** Some number of "Hi"s are printed

```
begin
    m = n
    while( m > 1 ) {
        loop i = 1...m {
            Print("Hi")
            Print("Hi")
        }
            Print("Hi")
        m = m/2
    }
end algorithm
```

algorithm $Ex2(N)$

$\langle pre-cond \rangle$**:** N is an integers.

$\langle post-cond \rangle$**:** Some number of "Hi"s are printed

```
begin
    loop i = 1...N {
        loop i = 1...N {
            Print("Hi")
            Print("Hi")
        }
    }
end algorithm
```

algorithm $Ex3(\langle x_1, x_2, \ldots, x_n \rangle)$

$\langle pre-cond \rangle$**:** x is an array of n integers.

$\langle post-cond \rangle$**:** Some number of "Hi"s are printed

```
begin
    if( n ≤ 1 )
        Print("Hi")
    else
        loop i = 1...n^3
            Print("Hi")
        end loop
        Ex3(⟨1 × x_1, x_2, ..., x_{n/2}⟩
        Ex3(⟨2 × x_1, x_2, ..., x_{n/2}⟩
        Ex3(⟨3 × x_1, x_2, ..., x_{n/2}⟩
        Ex3(⟨4 × x_1, x_2, ..., x_{n/2}⟩
        Ex3(⟨5 × x_1, x_2, ..., x_{n/2}⟩
        Ex3(⟨6 × x_1, x_2, ..., x_{n/2}⟩
        Ex3(⟨7 × x_1, x_2, ..., x_{n/2}⟩
        Ex3(⟨8 × x_1, x_2, ..., x_{n/2}⟩
    end if
end algorithm
```