CS 240 Fall 2014 **Name [5 pts]:** Keni Mou

Exam 2 November 10, 2014

Templates (20 pts)

1. Suppose I have the following classes, intended to be used with two different linked lists:

```
class IntElement {
      int value;
      IntElement *next;
      // etc.
};
```

```
class CharElement {
      char value;
      CharElement *next;
      // etc.
};
```

(a) [5 pts] I would like to collapse my code into one linked list implementation, instead of two. *Circle all of the correct templated class definitions* for a list element, below:

```
template <Type>
class Element {
      Switch (Type) {
            IntElement int;
            CharElement char;
      }
      // etc.
};
```

```
template class<Element> {
      Element value;
      Element *next;
      // etc.
};
```

```
template <typename Type>
class Element {
      Type value;
      Type *next;
      // etc.
};
```

```
class Element<template> {
      template value;
      Element *next;
      // etc.
};
```

```
template <Type> {
      Type value;
      Type *next;
      // etc.
};
```

```
template <Type> {
      Type value;
      Element<Type> *next;
      // etc.
};
```

```
template <typename Type>
class Element {
      Type value;
      Element<Type> *next;
      // etc.
};
```

```
template <Type>
class Type {
      Element value;
      Element *next;
      // etc.
};
```

(b) [5 pts] Which of the following lines of code would I be most likely to use to instantiate a new linked list of integers (circle one)?

(a) `List<int> mylist;`

(b) `Type List<ListElement>;`

(c) `List<Type> myintlist;`

(d) `int mylist;`

(e) `List<Element> mylist;`

(f) `template<int> mylist;`

(g) `int List;`

(h) `Type List(int);`

(2) Consider the following code:

```
#include <iostream>
using namespace std;

template <typename Type>
void display(Type t) {
   cout "VALUE: " << t << endl;
}

main() {
   char c(A);
   int i=1;
   display<char>(c); // also works as just display(c)
   display<int>(i);  // also works as just display(i)
}
```

(a) [5 pts] Explain what the *C++ compiler will do*, with the templated function, to build object code for this program.

The compiler will generate separate code for char and int.

Just like generating two functions :
void display (char t)
void display

(b) [5 pts] Suppose I want to use the templated `display()` function above to display instances of my own class, called `Collection`. Which of the following best describes one thing I should do to make that work? (Circle one):

(a) Make `display()` be a *friend* function of Collection.

(b) Implement `operator<<()` as a *friend* function of Collection

(c) Make a copy of `display()`, with all the "`Type`"s changed to "`Collections`", and implement it as a member function of `Collection`.

(d) Implement a value constructor of the `display` class that takes a `Collection` object as a parameter.

Recursion (25 pts)

(3) [12 pts] As described in class, the Fibonacci numbers are as follows:

1, 1, 2, 3, 5, 8, 13, 21, 34, etc.

That is, the first Fibonacci number is 1, the 2nd is also 1, and subsequent Fibonacci numbers are the sum of the previous two Fibonacci numbers.

Below are some attempts at recursive functions intended to return the i[th] Fibonacci number. For each one, Circle YES or NO to indicate whether it will work properly. If you circle NO, briefly indicate why it will not work. Please assume that the function is called initially with a positive integer parameter.

Works? YES / (NO) If not, why not?
No base case, will cause infinite loop.

```
int fib1(int i) {
   return(fib1(i-1) + fib1(i));
}
```

Works? YES / (NO) If not, why not?
No base case, will cause infinit loop

```
int fib2(int i) {
   return(fib2(i-2) + fib2(i-1));
}
```

Works? YES / (NO) If not, why not?
doesn't have base case (i==2)

```
int fib3(int i) {
   return(fib3(i-2) + fib3(i-1));
   if (i==1) return 1;
}
```

Works? YES / (NO) If not, why not?
doesn't have base case (i==2)

```
int fib4(int i) {
   if (i==1) return 1;
   return(fib4(i-2) + fib4(i-1));
}
```

Works? (YES) / NO If not, why not?

```
int fib5(int i) {
   if (i==1) return 1;
   if (i==2) return 1;
   return(fib5(i-2) + fib5(i-1));
}
```

Works? YES / (NO) If not, why not?
Wrong sequence.
if i==2, fib6(i-2) = fib6(0)
where when i==0 is not handled.
this will cause infinite loop.

```
int fib6(int i) {
   if (i==1) return 1;
   return(fib6(i-2) + fib6(i-1));
   if (i==2) return 1;
}
```

---

(4) [8 pts] Write a recursive function that returns the sum of the first N even numbers.

```
int sumEven(int n){
    if(n<=0) return 0;
    if(n==1) return 2;
    return n*2 + sumEven(n-1)
}
```

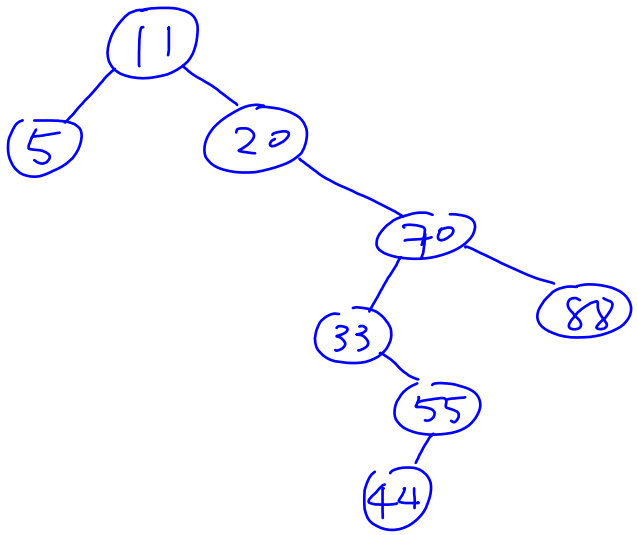
(5) [5 pts] As described in class and in the textbook, the Towers of Hanoi game requires its player to move a stack of disks---all having different sizes---from Peg A to Peg B, using a third peg (Peg C) as necessary. You may not place a larger disk on top of a smaller one. Suppose I have figured out a way to solve the Towers of Hanoi problem for 6345 disks and I have it all written down in my notebook. But I can't quite work out how to solve it for 6346 disks. Can you help?

First treat the 6346^{th} disk as the floor (since it is bigger than everyone else), move the rest 6345 disks to Peg C. Then move the 6346^{th} disk to Peg B. Last, treat the 6346^{th} disk as the floor again and move the rest 6345 disks from Peg C to Peg B.

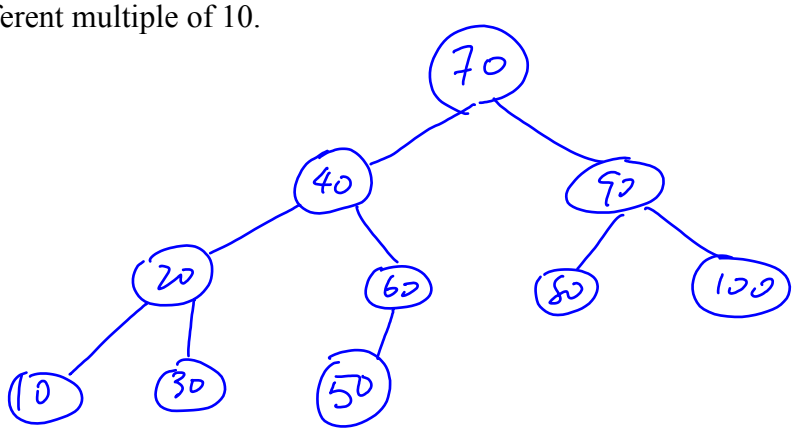
Trees (30 pts)

(6) [5 pts] Draw the Binary Search Tree (BST) that would result from inserting the following integers into an initially empty BST, in the order shown. (Insert 11 first, followed by 20, etc.)

11, 20, 70, 88, 5, 33, 55, 44, 1



(7) [5 pts] Draw a *COMPLETE* Binary Search Tree (BST) that contains exactly 10 nodes, each of which contains a different multiple of 10.

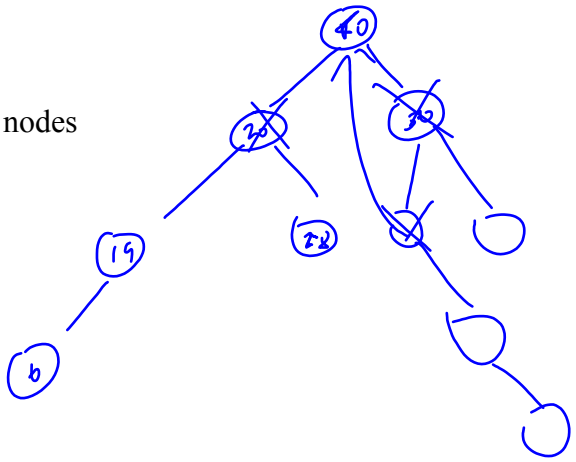


(8) [5 pts] Write the C++ class definition---data members only---for a class named `StringTreeNode`. Each instance of `StringTreeNode` is intended to operate within a binary tree that uses a *linked representation* (as opposed to an array representation).

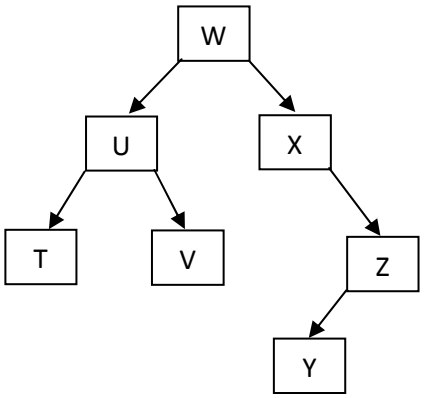
```
#include <string>
class StringTreeNode {
    public:
        string data;
        StringTreeNode *left;
        StringTreeNode *right;
};
```

(9) [5 pts] Consider the standard binary search tree `delete()` (or `remove()`) operation that we discussed in class. Not counting the value/node being removed, what is the *largest* number elements in the tree that might have a *different value stored in its parent*, after the delete finishes than before the delete operation was called?

(a) 0
(b) 1
(c) 2
(d) 3
(e) 4
(f) O ($\log_2$N) nodes
(g) O (N) nodes
(h) O (N $\log_2$N) nodes



(10) [5 pts] Consider the following Binary Search Tree of characters:



Suppose I have the following functions for a recursive traversal:

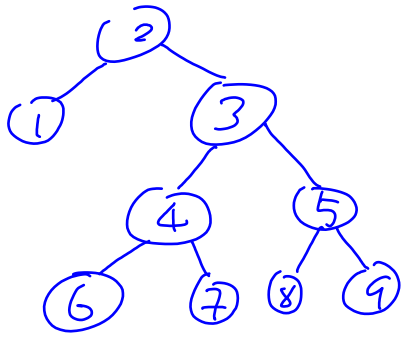
```
void BST::shownodes() {
   shownodesAux(root);
}

void BST::shownodesAux(Node *node) {
   if (node==NULL) return;
   shownodesAux(node->left);
   cout << node->data;
   shownodesAux(node->left);
}
```

Notice that I seem to be *trying* to do an ***inorder*** traversal, but I have two "`left`"s and no "`right`"s in my buggy code. What will `shownodes()` print when called on an object containing the tree above?

TUTWTUT

(11) [5 pts] Can a binary tree with four levels and nine nodes be *unbalanced*? (The root counts as a level.)
If so, draw such a tree and explain why it is unbalanced. If not, explain why not.

the left subtree of root only has 1 level, while the right subtree of root has 3 levels. The difference is greater than 1. Thus it is unbalanced.



Sorting (20 pts)

(12) [10 pts] Quicksort operates by selecting a pivot value, "splitting" the array into two parts, and then recursively calling Quicksort on the left part of the array, and also on the right part of the array. Suppose we are sorting a number of elements that is a power of 2. (2, 4, 8, 16, etc.) Write TRUE or FALSE next to each of the following:

False ____ After splitting an array into two parts, each part has the same number of elements.

True ____ After splitting an array into two parts, all of the elements in the left array will be smaller than (or equal to) all of the elements in the right array.

False ____ For Quicksort to operate properly, it must search through the entire array to find the right pivot, each time the Quicksort function is called.

True ____ After the left part of some sub-array is (quick)sorted and the right part of that sub-array is (quick)sorted, the entire sub-array is sorted.

False ____ Quicksort sorts all size N arrays equally quickly. In other words, the order of the values in the input array does not influence the runtime complexity of quicksort.

(13) [5 pts] We decided that Counting Sort operates in linear (O(N)) time, whereas other sorting algorithms require $O(N^2)$ or O(NlogN) time. Furthermore, we said that it can be *proved* that sorting algorithms can *not* operate any faster than O(NlogN). Which of the following best explains this apparent contradiction?

(a) Counting sort is an external sorting algorithm, because it uses storage outside of the input array. External sorts can run faster than NlogN.

(b) Counting sort restricts the range of inputs. Sorts that restrict the range of inputs can run faster than NlogN.

(c) Counting sort was discovered *after* the proof that sorting algorithms cannot run faster than NlogN.

(d) Only one phase (the "counting" part) of Counting sort runs in O(N) time… when you add the second phase (the part to write the values back out into the input array), the runtime complexity becomes O(NlogN), not O(N).

(e) "Professor" Lewis often contradicts himself; it's best to tune him out.

(14) [5 pts] What does it mean for a sorting algorithm to be stable?

(a) It uses only the input array and a contstant amount of additional storage.

(b) It will not fail when called for very large values of N.

(c) It will run for a long time without causing a segmentation fault, core dump, or bus error.

(d) It can be implemented in many different programming languages.

(e) When two elements have equal values, then their relative order in the input array matches their relative order in the output array.