EE 312 – Final Exam – Summer 2018

I promise that all work on this test is my own, that I have received no outside assistance on it, and that I am adhering to the university honor code.

Name___________________________________ UTEID___________ Date________ Rec: Tu / Th

Problem Number	Question	Points Possible	Points Off	G	RG
1	Recursion	28			
2	Classes	22			
3	References	14			
4	Trees	14			
5	C++ programs	15			
6	Hash Table	12			
TOTAL POINTS OFF:					
SCORE OUT OF:					

Instructions:

1. There are 14 pages, not including the scratch sheet at the end. You may detach the last sheet. Turn in all these 16 pages, stapled together.
2. Please turn off your cell phones.
3. You have 180 minutes to complete the test.
4. You may not use a calculator.
5. There is scratch paper at the end. You may ask for additional scratch paper if required. If your actual answer is on the scratch paper, state that fact clearly and legibly in the space below the question, and include the scratch sheet with your answer sheet.
6. When code is required, write C/C++ code, as specified.
7. You may break problems up into smaller methods. (In other words you may add helper methods wherever you like.)
8. You may not use functions that are not in the standard C/C++ library, except for the ones your write, of course.
9. Assume 32-bit machines, and pointers and ints are 32-bits wide.
10. Style is not evaluated when grading, but neatness and legibility are required. Use indentation to help the graders read your code. Write darkly.
11. The proctors will not answer most questions. Write down any assumptions that you need to make to resolve your doubts.
12. When you finish, show us your UT ID, turn in the exam and all scratch paper. Use the stapler we have if required.

Here are some function signatures for the string class, map class, etc.

```
string class overloads most of the operators that one may expect, including (==, !=, <
etc.)
default constructor string();
copy constructor string(const string &str);
constructor from c-string string(const char* s);
size size t size();
char& operator[] (size t pos);
string& operator= (const string &str);
string& operator= (const char* s);
```

General function in std.

`int std::stoi (const string&  str, 0, 10);` //convert string to int, if string is a decimal integer such as “14”

map class functions

`map<K, V>();` // default constructor, where K is the key type and V is the value type.

`std::map& operator[] (const key_type& k);`

> If *k* matches the key of an element in the container, the function returns a reference to its mapped value.
>
> If *k* does not match the key of any element in the container, the function inserts a new element with that key and returns a reference to its mapped value

`map<K, V>::iterator <iterator_name> = <map_object>.find(<k>)`

Searches the map for an element with a *key* equivalent to *k* and returns an iterator to it if found, otherwise it returns an iterator to the iterator called map::end. You can use == to compare the iterator returned by find to map::end, if you need to.

vector class functions

`operator[]` Access element (public member function)
`size` Return size (public member function)
`push_back` Add element at the end (public member function)
`pop_back` Delete last element (public member function)

EID:________________

Question 1: Recursion

For this question, you have to write several recursive functions in C. Helper methods are not allowed, and no loops or static variables or global variables are permitted. Nor are you allowed calls to malloc.

(a)

We have a triangle made of blocks. The topmost row has 1 block, the next row down has 2 blocks, the next row has 3 blocks, and so on. Compute recursively the total number of blocks in such a triangle with the given number of rows.

Sample function calls and output:

```
     triangle(0) returns 0
     triangle(1) returns 1
     triangle(2) returns 3
```

```
int triangle (int rows) {
// rows >= 0
```

(b)

We'll say that a "pair" in a string is two instances of a `char` separated by a `char`.

So in "`AxA`", the `A`'s make a pair. Pairs can overlap, so "`AxAxA`" contains 3 pairs -- 2 for `A` and 1 for `x`.

Recursively compute the number of pairs in the given null-terminated string. n > 0.

```
     countPairs("axa", 3) returns 1
     countPairs("axax", 4) returns 2
     countPairs("axbx", 4) returns 1
```

```
int countPairs(char* str, size_t n) {
// n is length of string, not including null termination.
// n > 0
```

For the remaining parts, use C++. No loops, static, or global variables, or calls to malloc or new are allowed. Helper functions are not allowed.

A binary tree node has a value, a pointer to a left child and a pointer to a right child.

```
struct Node {
        int val;
        Node* left;
        Node* right;
};
```

c)

Write a recursive function `isFull` that returns a `true` iff a binary tree (not a BST) is a full tree. Every node in a full binary tree has 0 or 2 children. No data structures other than `Node` are allowed.

```
bool isFull(Node* n) {
```

d)

Write a recursive function that does an in-order traversal of a BST and returns the number of elements. Also, the `vector a` is initially empty, but at the end of the function call, it should contain the elements in the order of traversal. No data structures other than `Node` or `vector` are allowed.

```
     int inOrder(Node* n, vector<int>& a) {
```

e)
Write a recursive function that returns `true` iff a given `Node*` represents the root of a Binary Search Tree. You may use a function `bool isSorted(vector<int> a)` that returns `true` iff the `vector a` is sorted in increasing order. You may also use any functions from the above parts, even if you did to write them. Helper functions are allowed.

```
bool isBST(Node* n) {
```

Question 2: Classes and Inheritance

a) Write 4 C++ classes -- ROM, RAM, Register, and Disk -- that represent storage on your computer.
A RAM object has the capacity to store 10 `ints`, and a `read()` function that returns successive stored `ints` each time it is called. (For example, if the stored ints are `1, 2, 5` …, three calls to `read()` will return `1, 2, and 5.` On the 11th read, the returned value goes back to the first `int`, and the cycle repeats. You need not concern yourself with how the RAM is written into.
A ROM object can store a single `int`, and this is initialized during construction. A `read()` function returns this `int`.
A Register object stores 4 bytes. `void swap(int source, int dest)` swaps the data in the `source` byte and the `dest` byte. `source` and `dest` are integers from 0-3. Again, you need not concern yourself with how Register is initialized.
A Disk object consists of a variable number of RAM objects, the number being specified during construction through an `int` parameter. Disk has a `clear()` function that sets the contents (all bits) of all its RAMs to 0.

Hints: Avoid using the heap unless necessary. What data type can store a single byte?
Make sure that all classes have constructors, destructors and assignment operators as and when required. Avoid useless functions.

b)
Draw a reasonable inheritance tree that shows these classes, and any other classes required to demonstrate inheritance -- the class names should be in rectangles, and arrows should point from derived classes to base classes.
For full credit, your tree should have at least 3 levels of classes, and should be close to the way a software architect might structure the classes.

c)
Draw a diagram that shows these classes indicating any *has-a* relationships between these 4 classes, together with another hypothetical class that contains objects of all these 4 classes. An arrow should go from a contained class to a container class. Name the hypothetical class with a reasonable name.

Question 3: C++ references

a) What is the output of main()? Write only the output in the box.

```
#include <iostream>
using namespace std;
int func1(int x) {
    x--;
    return x - 1;
}
int func2(int& x, int& i, int& n) {
      for (int i = 0; i < n; i++) {
            x++;
      }
      n++;
      return x;
}
int main(void) {
   int x = 4;
   int n = 5;
   int i = 0;
   func1(x);
   func2(x, i, n);
   cout << x << " ";
   int func1Out = func1(x);
   cout << x << " " << func1Out << " ";
   int func2Out = func2(x, i, n);
   cout << n << " " << func2Out << " ";
   cout << func1(x) + func2(x, i, n) << endl;
   return 0;
}
```

```
class Foo {
      int n;
      string s;
      char* d; // array of chars
      uint32_t size; // size of array not including any null termination
};
```

b)
Does the class `Foo` need a copy constructor? If so, write down its copy constructor declaration. Remember to write `const` where needed.

c)
Does the class above need an assignment operator? If so, write down its function declaration. Remember to write `const` where needed. The function should be capable of being used with chaining.

d)
Does the class above need a destructor? If so, write down its function declaration. Remember to write `const` where needed.

e)
The function below returns a new `Foo` whose array is the concatenation of the two. Write down its optimal function signature. Remember to write `const` where needed. If nothing is to be written in a blank, write (only) `NA`.

```
_______________ Foo::operator+(______ Foo& rhs) __________;
```

f)
The function below concatenates `that's` array to `Foo's`. The function should be capable of chaining. Write down its optimal function signature. Remember to write `const` where needed.

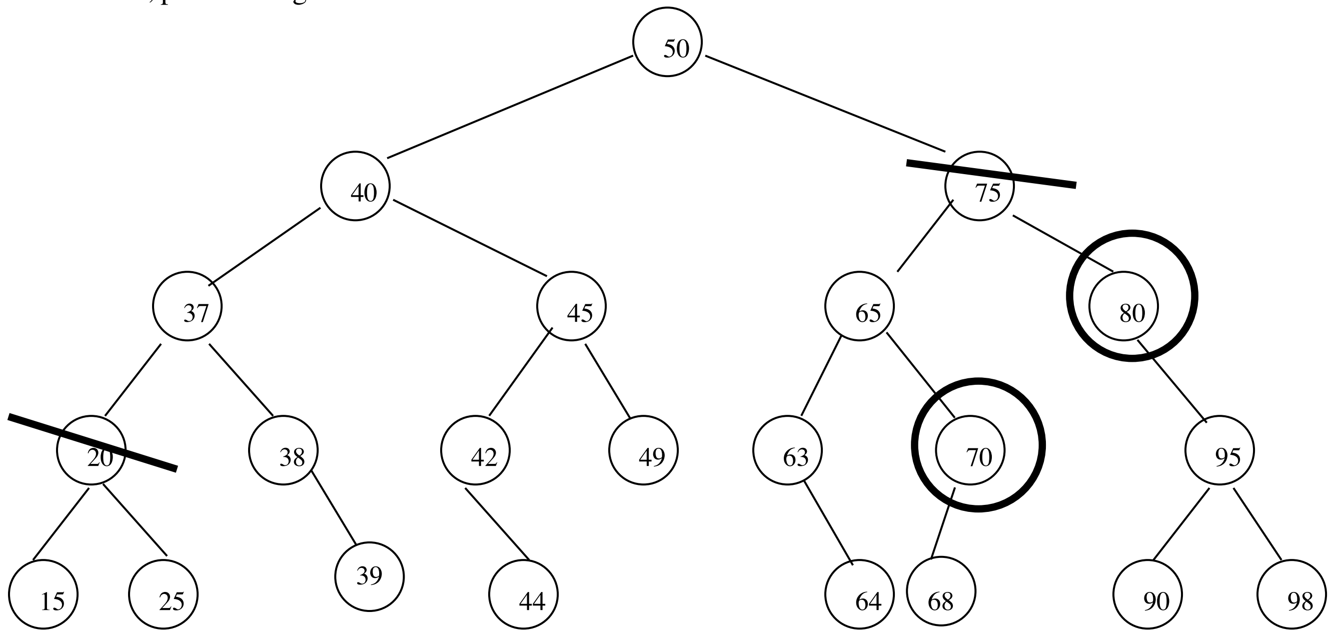
```
______________Foo::operator+=(_______Foo& that) ___________;
```

Question 4: Trees

a) Given a sequence of N integers, how might you make a binary search tree of minimum height containing these values? Describe your algorithm in pseudo-code or clear English. We are looking for the general algorithm, which would yield close to the minimum height for large trees.

b) Draw a binary search tree that is of height 4 from these integers: 15, 3, 6, 9, 8, 7, 10, 12, 0, 4. Recall that the height of a tree is the number of edges from the root to a leaf, maximized over all leaves. A tree with one node has height = 0.

Consider the following **binary search tree**. **Use the original tree** below when answering each subpart (c) through (g). In all cases, the answer should be such that the fewest number of changes need to be made to the tree. If you have two choices, pick the larger one.



(c) If the value 69 is inserted into this tree, which node becomes its parent?

____

(d) If the value 43 is inserted into this tree, which node becomes its parent?

____

(e) If we delete node 20, which node should be its replacement node?

____

(f) If we delete node 75, which node should be its replacement node?

____

(g) If we delete the root node 50, which node should be selected as its replacement node?

____

(h) Write a C++ `Node struct` for a tree where each `Node` has an `int` value, a parent, and any number of children. You may use other data structures from the C++ Std Lib that we used in class.

Question 5: C++ programs

a)

Convert the following class to a template class that can hold any type of class instead of `ints`. Exact syntax is not expected.

```
class Array {
    int* data;
    int size_of_data;
    int value_of_last_element_in_data; // value of last element of data array
};
```

b)
Write a program snippet to read tokens from a file using the function that returns a std::string:

```
      string read_next_token();
```

The file has commands:

`var <variable_name> <int val>` // creates a variable and sets its value to val
`set <variable_name> <int val>` // sets a variable's value to val, creating it if it is not present.
`output <variable_name>` // prints the value of the variable_name; exits the program with the exit(<int>) function if `variable_`name was not previously created. Use the `std::stoi` function to convert from string to int.
`END` // reading this token means the token stream has ended.

The commands themselves will be well-formed, with the only possible error being the output error listed above. Newlines, spaces etc. are handled by `read_next_token().` Use a `map` data structure for your symbol table.

Question 6: Hash tables in C++

A HashTable with an array of length size holds `Point` objects, where `Point` is defined below.

```
struct Point {
    int x_coord;
    int y_coord;
};
```

a)
Write a suitable `hashCode` function for the `Point` class that returns a `long`, that can be used in a hash table. The size of the hash table is not known by the `Point` objects.

b)
Write a suitable function for the `HashTable` class that returns an index from 0 to (size – 1), given a `long` hash value.

c)
For this part, assume that your hash table holds `ints`, and is of size 7. The index of the location of a value `v` in the hash table is simply `v%size`. Show the state of the hash table after the following elements have been added. Assume separate chaining using linked lists at each index, as we did in class.
Elements added: 4, 9, 0, 1, 18, 2, 19, 21

hash table index	values
0	
1	

d)
Explain briefly why we increase the size of the hash table and rehash all the values before putting them back into the new table, even though separate chaining hash table never actually fill up.

Blank page, use for scratch work.