ECE551
Final

Name: NetID:

There are 8 questions, with the point values as shown below. You have 175 minutes with a total of 115 points. Pace yourself accordingly.

This exam must be individual work. You may not collaborate with your fellow students. However, this exam is open book and open notes. You may use any printed materials, but no electronic nor interactice resources.

I certify that the work shown on this exam is my own work, and that I have neither given nor received improper assistance of any form in the completion of this work.

Signature:

#	Question	Points Earned	Points Possible
1	Multiple Choice		10
2	Concurrency		10
3	Data Structure Concepts		12
4	OO Implementation		15
5	Algorithmic Basics		12
6	Coding 1		12
7	Coding 2		22
8	Coding 3		22
	Total		115
	Percent		100

Question 1 Multiple Choice [10 pts]

1. Which data structure can implement a Map or Set ADT with O(1) amortized access time?
   - (a) Linked List
   - (b) Min-Heap
   - (c) Max-Heap
   - (d) Graph
   - (e) Hash Table
2. Which of the following is detrimental to code maintainability?
   - (a) Code Duplication
   - (b) Object Oriented Design
   - (c) Abstraction
   - (d) Dynamic Dispatch
   - (e) None of the Above
3. Which of the following accurately describes pointers?
   - (a) Variables which change how method calls are dispatched
   - (b) Variables which cannot be modified once an initial value is set
   - (c) Variables whose value is actually a function
   - (d) Variables whose value represents the memory location of some other piece of data
   - (e) None of the Above

4. What is the main dis-advantage of merge sort?
   (a) Its worst case running time is $O(N^2)$
   (b) Its average case running time is $O(N^2)$
   (c) It requires allocating extra space for temporary arrays
   (d) It requires very complex operations at each step which take a long time.
   (e) None of the Above

5. Which algorithmic category best describes Prims's MST algorithm?
   (a) Dynamic Programming
   (b) Brute Force
   (c) Genetic
   (d) Greedy
   (e) None of the Above

Question 2 Concurrency [10 pts]

1. Briefly explain the concept of a "thread."

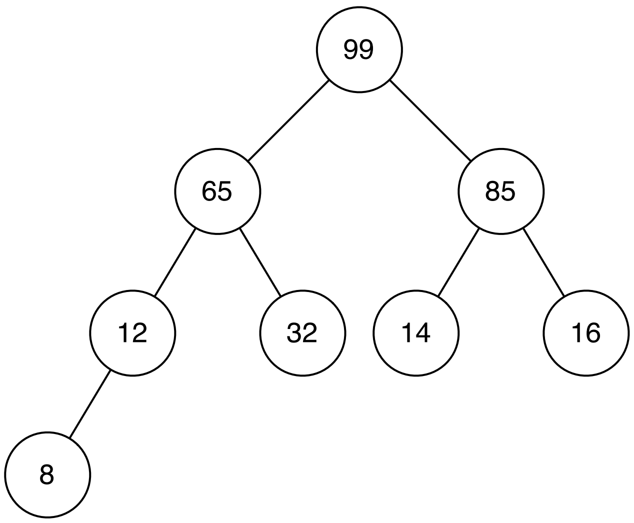
2. Briefly explain what a race condition is.

3. Explain how a concurrent program protects against race conditions.

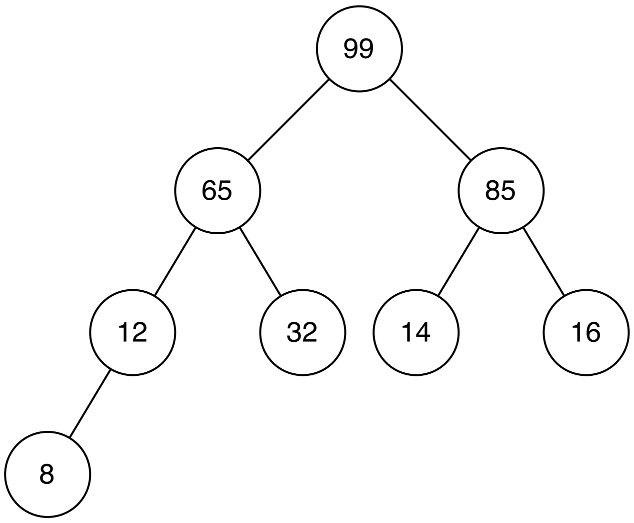
Question 3 Data Structure Concepts [12 pts]

Show the results of performing each of the following operations on the shown data structures:

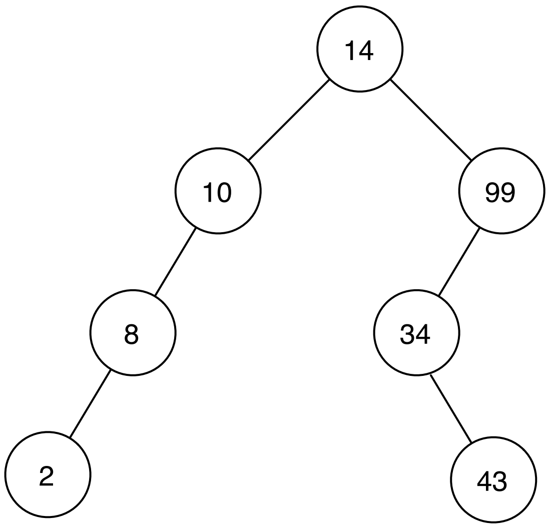
1. Add 77 to the following max-heap



2. Remove the maximum element from the following max-heap:



3. Add 7 to the following (regular, un-balanced) BST



4. Remove 14 from the BST in the previous part (before you added 7).

Question 4 OO Implementation [15 pts]

Suppose I have the following class:

```
class A {
public:
  int x;
  int y;
  virtual void foo() { /* code */ }
  virtual int bar(int x) { /* code */ }
};
```

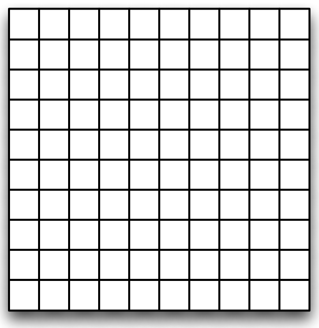
1. Write an equivalent C struct (you may write the type of the vtable as `void **`).

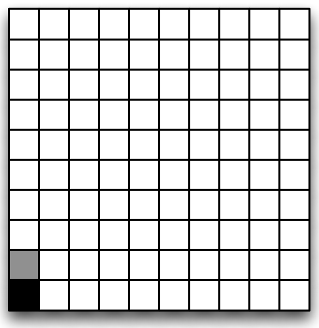
2. Suppose that `foo` is in element 0 of the vtable, and `bar` is in element 1, and that `x` is declared as an `A*`. Translate `x->bar(3);` to C. You may assume that `intFnPtr` has been `typedef`ed as a pointer to a function taking of the appropriate type for `bar` (you can call a function via a pointer just like you can call a function via its name—its name is just a pointer to the function):

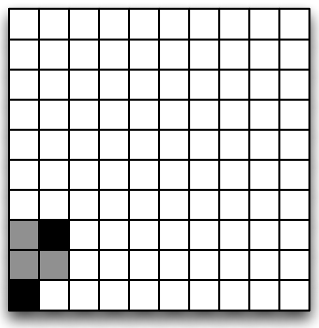
3. If we inherit from a class *virtually*, what additional information must be placed in the vtable?

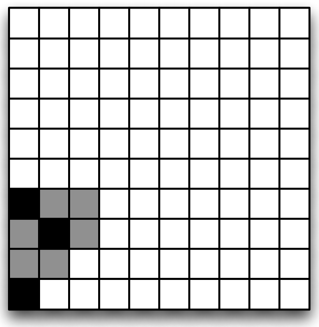
Question 5 Algorithmic Basics [12 pts]

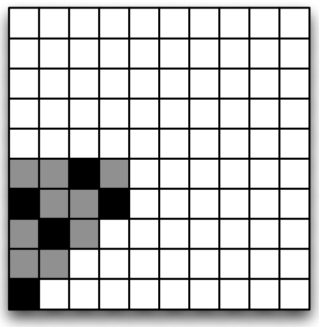
The diagrams shown beloware the result of executing an algorithm which has one parameter, N, which must be a non-negative integer, and colors boxes (grey or black) on a 10x10 grid of white squares. For values of N from 0 to 5, the algorithm produces the following patterns.

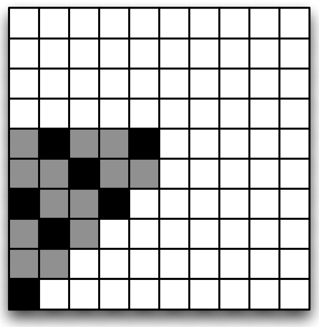

N=0


N=1


N=2


N=3


N=4


N=5

Write down the algorithm used to generate these patterns *in a clear, step-by-step fashion, which could be directly translated to code.*

Question 6 Coding 1 [12 pts]

Write the `sumEven` function in the following `LinkedList` class (which only holds `int`s). This function should return the sum of all of the *even* elements of the list. For example, if the list held `1,5,6,7,9,12` then this function should return `6+12=18`.

```
class LinkedList {
  private:
    class Node{
       public:
        Node * next;
        int data;
        Node(int _data): next(NULL), data(_data) {}
        Node(int _data, Node * _next): next(_next), data(_data) {}
    };
  Node * head;
public:
  int sumEven() {






  }
};
```

Question 7 Coding 2 [22 pts]

Suppose you have already written a templated `Map` class, with the following interface:

```
template<class K, class V>
class Map {
public:
  Map();
  void add(const K& key, const V& val);
  V find(const K& key) const;
  void remove(const K& key);
  class const_iterator {
    iterator & operator++();
    const pair<K&,V&> operator*();
    bool operator==(const const_iterator & rhs);
    bool operator!=(const const_iterator & rhs);
  };
  const_iterator begin() const;
  const_iterator end() const;
};
```

and you also have the following abstract `Function` class:

```
template<class R, class A>
class Function {
public:
  virtual R invoke(A arg) =0;
};
```

Write a function which re-maps all of the keys in a `Map`, creating a new `Map` which has exactly the same keys as the first, but whose values are computed from a re-mapping function, which is passed in as a `Function` object. Specifically, if the original Map has the (key,value) pair (k,v) then the output map should have(k,f(v)) where f is the function represented by the Function object passed in.

(answer on the next page)

```
template<class K, class V1, class V2>
Map<K,V2> * remap(Map<K,V1> * inMap, Function<V2,const V1 &> * f){
```

```
}
```

Question 8 Coding 3 [22 pts]

Suppose you have the following BinaryTree class (which holds ints):

```
class BinaryTree {
  private:
    class Node {
    public:
      int data;
      Node * left;
      Node * right;
    };
    Node * root;
 public:
  //constructors, destructors, other methods not shown
  bool hasPathSum (int target) {
    //you will write this
  }
};
```

You must write the hasPathSum method which determines if the BinaryTree has a continuous sequence of items along a path which sum to the specified target. For example, in the BST from Question 3, part 3, path sum of 20 may be formed by 10+8+2, however, a path sum of 16 may **not** be formed by 14+2 since they are not in a contiguous path. You may write any helper methods you wish. We do not care about efficiency, only correctness.

(answer on the next page)

```
class BinaryTree {
 //everything else omitted to give you space to write







   bool hasPathSum (int target) {









   }
};
```