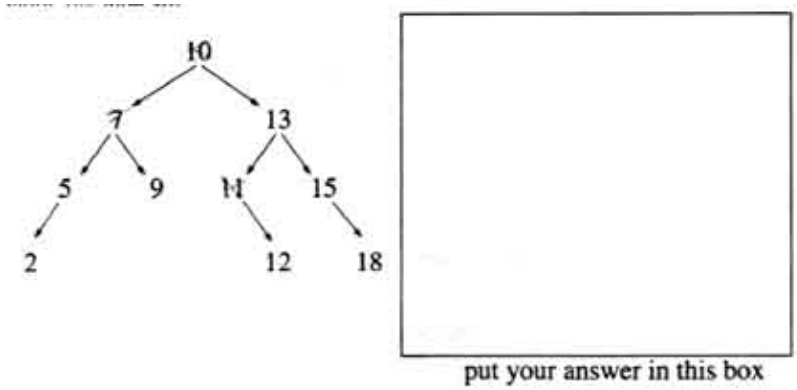
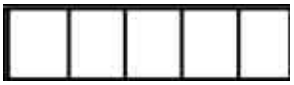
CS61B, Fall 1995
Midterm #2
K. Yelick

Problem #1

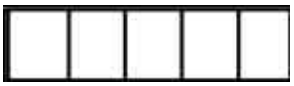
a. (2 points) Show the binary search tree (BST) that would result from applying the following operations to the given BST: insert(6), insert(8), delete(10), delete(7). (Just show the final BST.)



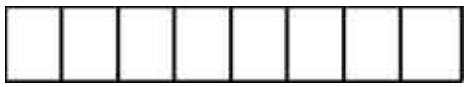
b. (2 points) Given below is a hash table with space for 5 entries. Assume we are using the (naive) hash function: **`key % 5`**. What will the table look like after the following 4 operations, assuming we are using chaining: insert(2), insert(8), insert(17), insert(12).



c. (2 points) Redo the previous problem assuing we are using linear probing



d. (2 points) Show the value of the heap that would result after inserting the following elements in this order: 19, 4, 20, 12, 18, 3, 32, 5.



e. (3 points) The **`list`** class is built using **`listNode`** objects. Allysa P. Hacker has decided to write a destructor for the **`listNode`** objects, using the code given below. Will this function work? If not, describe what is wrong.

```
//Precondition: this != NULL. //Postcondition: Delete this node and all nodes a
```

f. (2 points) Draw an inheritance hierarchy for the following classes:

```
class B : public C {
 ...
};
class C {
 ...
};
class A : public D {
```

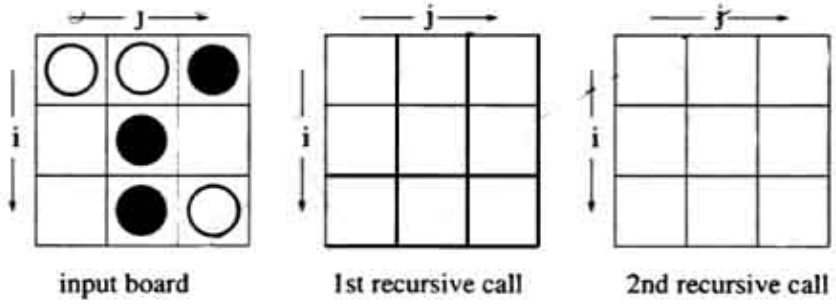
```
 ...
};
class D : public C {
 ...
};
```

Problem 2. (6 points)

Henry Pointerdexter is modifying his 61b Ataxx project to play the game of tic-tac-toe. The game of tic-tac-toe is played on a 3x3 board using white and black pieces. At each step, a player may place a piece on any open square. The goal of the game is to get three of your pieces in a row: left-right, up-down, or diagonally. The key function in Henry's MachinePlayer class is a recursive function that looks for the best move by doing backtracking search of the "game tree." Below is a sketch of his code. (There is nothing tricky here, this should look similar to your project 1 code and our solution.)

```
Search (Board *board, int &bestScore, Move &bestMove, int level)   if this is t
```

a. (2 points) Draw the two boards that would be passed to the **`Search`** function the next two times it is called, assuming that it is initially called by the white player with **`level = 0`** and **`board`** set to *input board* below.



b. (2 points) Tic-tac-toe involves no flipping or jumping: once a piece is placed, it will remain in that position throughout the game. How can this fact be used to optimize the search process?

c. (2 points) Henry has noticed that some boards come up over and over again from a different order of moves. He would like to save the search results in a hash table so that he can reuse it later. What should he use as keys in the hash table?

What item (or items) should he use as the value associated with each key in the hash table?

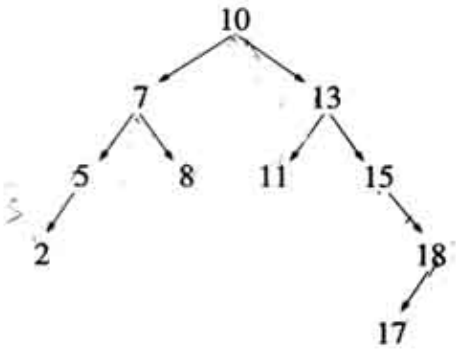
Problem 3. (13 points)

A *threaded tree* is a variation on a binary tree that makes use of the null pointers that exist at the bottom of the tree. If a node has no left child, the left child pointer will be used to point to the node that comes before it in an inorder traversal (the *inorder predecessor*). If a node has no right child, the right child pointer will be used to point to the node that comes after it in the order traversal (the *inorder successor*). (The predecessor pointer is null for the first node in the inorder traversal and the successor pointer is null for the last node in the inorder traversal.) Assume that we have modified the **`bstClass`** from Carrano to implement a threaded tree. Although several changes have to be made to the code, the only change to the data structure itself is the addition of two new boolean values in each **`treeNode:`**

```
struct treeNode
{  treeItemType Item;
   treeNodePtr LChildPtr, RChildPtr;
   bool leftIsPred;
   bool rightIsSucc;
};
```

- If **`leftIsPred`** is true, then the node has no left child. The LChildPtr therefore points to the inorder predecessor or is null if none exists.
- If **`rightIsSucc`** is true, then the node has no right child. The RChildPtr therefore points to the inorder successor or is null if none exists.

a. (3 points) Add the predecessor and successor pointers to the following picture. Label each predecessor with "P" and successor with "S," so we can tell which ones are which.



For parts b and c, you are to write part of an inorder traversal iterator like the one you wrote in Homework 7 (see appendix). Because of the additional pointers in the tree, we can eliminate the stack and instead use only a single "current" pointer within the iterator. For both functions, you may use helping functions, but show all of your code.

```
class bstInorder {     public;        // as before     private;        bestClass & theTree;  //
```

b. (5 points) Write the constructor for the bstInorder class. (The constructor should initialize the iterator so that other operations work correctly without first calling **`init()`**.)

c. (5 points) Write operator++ for the bstInorder class.

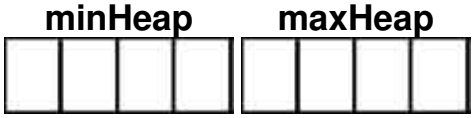
Problem 4. (8 points)

Ben Bitdiddle needs a data structure with the following operations.

```
class SomeDateStructure {    public:      // Constructors and destructor omitte
```

He wants both **`FindMin`** and **`FindMax`** to work in $O(1)$ time. He also want **`Insert, RemoveMin`** and **`RemoveMax`** to work in $O(\log n)$ time. His idea is to use two heaps, one (called MaxHeap) ordered with the larger elements at the top and one (called MinHeap) ordered with the smaller elements at the top. Both heaps are represented as vectors/arrays of HeapNodes.

a. (2 points) Fill in possible values for **`minHeap`** and **`maxHeap`** that contain the items 21, 22, 23, and 24. (There may be more than one correct answer.)



b. (6 points for part b-d combined) Consider the problem of deleting the minimum element from your answer to part a. To do this in $O(\log n)$ time, you need to asdd some information to each **`HeapNode`** beside the item. What other field(s) do you need? Give a name, type, and a few words to describe each field.

c. During insertion and removal, it is often necessary to move a node in a heap. How will you update the other heap in this case?

d. List the steps involved in removing the minimum element (21) from the heap in part a.
