Sample questions for the final.

1. Insert the following values into an initially empty binary search tree: 50, 40, 30, 20, 80, 70, 60, 45, 42, 48, 90. Show the result binary search tree.

2. Place the following values into the empty nodes of the tree below so that it forms a valid binary search tree: 15, 10, 4, 2, 40, 35, 18, 1, 23, 16, 9, 3.

3. Write the output from the inorder traversal of the tree above (with the values filled in, of course)?

4. What is the output from the following program?

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

void threePar(int r, int * s, int & t){
    int temp = 20;
    *s = r;
    t = *s;
    r = temp;
}

int main(){
    int x, y, z;
    int * ptr;
    x = 5;
    y = 10;
    z = 15;
    ptr = & y;
    cout << "Before: " << x << " " << y << " " << z << endl;
    threePar(x, ptr, z); // call
    cout << "After: " << x << " " << y << " " << z << endl;
    return 0;
}
```
5. The *left-most path* in a binary search tree is the sequence of values encountered by starting at the root and repeatedly taking the left-child link. Write a recursive function to print the left-most path of binary search tree.

6. The **peek()** function in the Stack lets you find out which item is ready to be popped next. Write a **peek()** function for the Queue class which returns the next item to be dequeued.

7. Write a code fragment to exchange the values in two adjacent nodes of a linked list. Assume that the pointer variable **cursor** points to the first of the two nodes.

8. Suppose that you have a stack S and an empty second stack T. How can you delete the occurrence of a specified item from S, leaving the order of the remaining items unchanged?

9. Write an implementation of ADT stack that uses a dynamically allocated array to represent the stack items.

10. Find the errors in the following program fragment. (You might want to simply write out a correct version, making as few changes as needed.)

```c
// A recursive function to return the product of all values in an array A of n elements

void computeProd(int A[], int n)
{
    if ( n == 0 )
    {
        return 1;
    }
    computeProd(n-1);
    answer * A[n];
}
```