CMSC 202 **Questions for the Final Exam** Spring 1996
Sections 101, 102 & 103 Handout #27

The final exam will be a two–hour exam on Thursday, 16 May at 10:30 a.m. in LH4. The exam will be taken entirely from the following questions (although some of the questions may be changed in minor ways).

1. All questions for Test 1 (from Handout 10).
2. All questions for Test 2 (from Handout 18).
3. All exercises from the topic handouts (recursion, pointers, functional parameters, stacks and queues, binary trees).
4. Answer the following True/False questions:

   ________ A binary tree is a tree in which every node has exactly two children.

   ________ It is possible for a tree to have more than one root.

   ________ If a tree has just one node, that node is both a leaf and an internal node.

   ________ If a depth–first search encounters node A before node B, then a breadth–first search will encounter node B before node A.

   ________ In the worst case, quicksort requires O(n) time.

   ________ In the average case, quicksort requires O(log n) time.

   ________ In C, if you dereference a `NULL` pointer, you will get the value 0.

   ________ A pointer value is a pointer variable to which a value has not yet been assigned.

   ________ A recursive function can have at most one base case.

   ________ In C, `malloc` returns a pointer value.

   ________ In C, if two pointer values of the same type point to the same item, then they are the same pointer value.

   ________ Binary search on any ordered binary tree requires $O(\log n)$ time.

   ________ A recursive function does not need a base case if it makes only one recursive call.

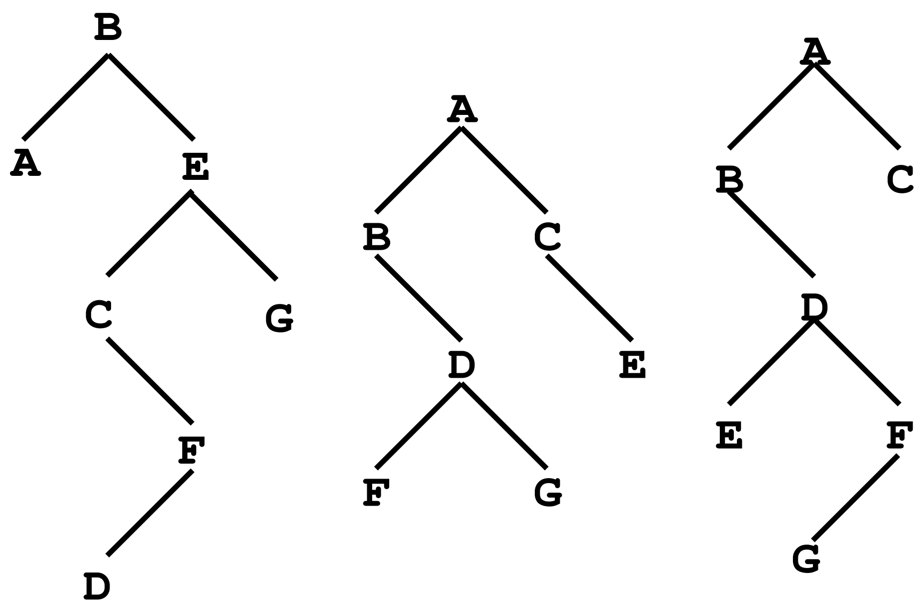
   ________ Depth–first search will always find a desired node in a tree faster than will breadth-first search.

   ________ Hashing is a method to sort a set of data values by using 'buckets.'

   ________ Quicksort is always faster than selection sort.

   ________ On average, quicksort is faster than selection sort.

5. Briefly (*i.e.*, in 1–3 sentences) define each of the following terms:
   - Trie
   - Selection sort
   - Merge sort
   - Quick sort
   - Big–Oh
   - $O(n^2)$ algorithm
   - $O(n \log n)$ algorithm
   - $O(n)$ algorithm
   - $O(\log n)$ algorithm
   - $O(1)$ algorithm
   - Linear search
   - Binary search
   - Hash function
   - Functional parameter
   - Data abstraction
   - Procedural abstraction
   - Binary tree
   - Leaf of a tree
   - Descendents of a node in a tree
   - Ancestors of a node in a tree
   - Depth of a node in a tree
   - Height of a tree
6. For each of the following trees:



fill in each of the following blanks:

A. The root(s) of this tree is/are ________________

B. The leave(s) of this tree is/are ________________

C. The internal node(s) of this tree is/are ________________

D. A pre-order traversal of this tree visits the nodes in the order ________________

E. An in-order traversal of this tree visits the nodes in the order ________________

F. A post-order traversal of this tree visits the nodes in the order ________________

G. The parent of node E is/are ________________

H. The ancestors of node E is/are ________________

I. The descendants of node E is/are ________________

J. The height of this tree is ________________

7. Write a C function called `copy_tree` that takes a `tree_type` as its argument and returns a copy of that tree.

8. Write a C function called `same_tree` that takes two `tree_type`s as its arguments and returns `TRUE` if those trees are identical (*i.e.*, they have the same structure, and the same values in the same places), and returns `FALSE` otherwise.

9. Write the `mirror` function. `Mirror` takes a `tree_type` as its argument, and returns a new `tree_type` that is the mirror image of the original tree. `Mirror` should not modify its argument in any way. You may use only the routines `tree_cons`, `tree_data`, `left_child`, `right_child`, and `empty_tree`, and the constant `the_empty_tree`, to manipulate trees.

10. Write a C function called `pre_order_traversal` that takes a `tree_type` and an operation as its arguments and performs the operation on each node of the tree in a pre–order fashion. The operation should be a `void` function that takes a `tree_type` as its argument.

11. Write a C function called `count_occurrences` that takes a `tree_type` and a predicate as its arguments and returns the number of nodes in the tree that fulfill the predicate (*i.e.*, cause the predicate to return `TRUE`). The predicate should take a `tree_type` as its argument and return a `BOOL`.

12. Explain why selection sort is an $O(n^2)$ algorithm on average.

13. Explain why quicksort is an $O(n \log n)$ algorithm on average, but is an $O(n^2)$ algorithm worst case.

14. Here is a C implementation (not necessarily exactly the one we covered in class) of the `quicksort` routine and its auxiliary function, `split`. The code is syntactically correct, but it contains exactly three logical errors. Identify each error (1 point each), state (in English) the reason it is an error (1 point each), and show (in C) how the error can be corrected (1 point each). Assume that the types `array_type`, `array_item_type`, and `BOOL`, as well as the constants `TRUE` and `FALSE`, have been defined.

15. Here is a C implementation (not necessarily exactly the one we covered in class) of the `merge_sort` routine and its auxiliary function, `merge`. The code is syntactically correct, but it contains exactly three logical errors. Identify each error (1 point each), state (in English) the reason it is an error (1 point each), and show (in C) how the error can be corrected (1 point each). Assume that the types `array_type`, `array_item_type`, and `BOOL`, as well as the constants `TRUE` and `FALSE`, have been defined.

As an example of this last sort of problem, here is a find–the–bugs question for depth–first search:

Here is a portion of the C code to implement breadth-first search using an explicit stack. The code is syntactically correct, but it contains exactly four logical errors. Identify each error, state (in English) the reason it is an error, and show (in C) how the error can be corrected:

```
tree_type
dfs_queue(tree_type tree, BOOL (*predicate)(tree_type))
{
  queue_type queue;
  tree_type guess;
  enqueue(tree, queue);
  while (!empty_queue(queue)) {
    guess = dequeue(queue);
    if (!empty_tree(guess))
      continue;
    if (predicate(tree))
      return(guess);
    enqueue(left_child(guess), queue);
    enqueue(right_child(guess), queue);
  }
}
```