Test 1: CPS 100

Owen Astrachan

February 14, 2002

Name: ___________________________ (1 pt)
Login: __________________________
Honor code acknowledgment (signature) __________________________

<table>
<thead>
<tr>
<th></th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>15 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>16 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>70 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 8 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.
In writing code you do not need to worry about specifying the proper #include header files. Assume that all the header files we’ve discussed are included in any code you write.
The declaration for linked list nodes on this test is:

```cpp
struct ListNode
{
    string info;
    ListNode * next;

    ListNode(const string& s, ListNode * ptr)
    : info(s), next(ptr)
    { }
};
```

The declaration for binary search tree nodes on this test is:

```cpp
struct TreeNode
{
    string info;
    TreeNode * left;
    TreeNode * right;

    TreeNode(const string& s, TreeNode * lt, TreeNode * rt)
    : info(s), left(lt), right(rt)
    { }
};
```

Some common recurrences and their solutions:

- \( T(n) = T(n/2) + O(1) \) \( O(\log n) \)
- \( T(n) = T(n/2) + O(n) \) \( O(n) \)
- \( T(n) = 2T(n/2) + O(1) \) \( O(n) \)
- \( T(n) = 2T(n/2) + O(n) \) \( O(n \log n) \)
- \( T(n) = T(n-1) + O(1) \) \( O(n) \)
- \( T(n) = T(n-1) + O(n) \) \( O(n^2) \)
PROBLEM 1: \((\text{Cheeri-Oh})\)

Part A (5 pts)
Consider the code fragment below. What is the \textit{exact} value printed when \(n = 1,001\) (show work for partial credit) \textbf{and} what is the big-Oh complexity of this loop in terms of \(n\) (with a brief justification).

```cpp
int sum = 0;
for(int k=0; k < n; k++) {
    sum += k;
}
cout << sum << endl;
```

Part B (5 pts)
Consider the code fragment below. What is the \textit{exact} value printed when \(n = 742\) (show work for partial credit) \textbf{and} with is the big-Oh complexity of this loop in terms of \(n\) (with a brief justification).

```cpp
int sum = 0;
for(int k=1; k <= n; k = k * 2) {
    sum += k;
}
cout << sum << endl;
```

Part C (5 pts)
Consider the code fragment below. What is its big-Oh complexity in terms of \(n\), briefly justify your answer.

```cpp
int sum = 0;
for(int k=0; k < n; k += 2) {
    for(int j=1; j <= n; j++) {
        sum += j;
    }
    for(int j=0; j < k; j++) {
        sum += 2*j;
        cout << "be my valentine" << endl;
    }
}
cout << sum << endl;
```
PROBLEM 2:  (Mad Max)
Part A (4 pts)
Consider the function `maxList` below. It returns the alphabetically last string in a list. For example, for the list ("lemon", "apple", "orange", "guava") it returns "orange".
Write a recurrence relation for `maxList` in terms of the number of nodes in the list passed to `maxList`. Justify your answer briefly.

```cpp
string max(const string& a, const string& b)
// post: return larger of a, b
{
    if (a > b) return a;
    return b;
}

string maxList(ListNode * list)
// post: return alphabetically last/largest string in list
{
    if (list == 0) return "bumblebee";
    if (list->next == 0) return list->info;

    string after = maxList(list->next);
    return max(list->info, after);
}
```

Part B (6 pts)
The function `maxList` returns the correct value for any non-NULL/empty list. Write a function `maxTree` that works for trees, i.e., it returns the alphabetically last/greatest value in a binary tree. Your function should work correctly for any non-NULL/empty tree. The tree passed to `maxTree` is not necessarily a search tree, e.g., for the tree below the function `maxTree` should return the string "Zucchini".

```cpp
string maxTree(TreeNode * tree)
// pre: tree is not, necessarily, a search tree
// post: return alphabetically last/largest value in tree
```

```
    Carrot
      /     \
   Potato  Onion
     /     /     \     /     \     \   
   Tomato Zucchini Beet
```
PROBLEM 3:  (Countdown (10 points))

The code below is taken from readwords3.cpp and was suggested as useful in implementing the Anagram program. You may find it useful in writing the function countUnique specified after it.

```cpp
void uniquify(const tvector<string>& list)  
// pre: list is sorted and contains list.size() entries  
// post: prints each string in list once with count of occurrences
{
    string last = list[0];  
    int count = 1;
    for(int k=1; k < list.size(); k++) {
        if (list[k] != last) {
            count++;
        } else {
            cout << count << "\t" << last << endl;
            last = list[k];
            count = 1;
        }
    }
    cout << count << "\t" << last << endl;
}
```

Write a function that returns the number of unique/different strings in a sorted linked-list. For the list below countUnique should return 3 since there are three different strings in the list.

("dog", "dog", "elf", "frog", "frog", "frog", "frog")

For full credit your function must execute in \(O(n)\) time for an n-node list.

```cpp
int countUnique(ListNode * list)  
// pre: list is sorted, list is NULL/0 terminated, no header node  
// post: returns # different/unique strings in list
{
```
PROBLEM 4:  (*I've been duped (8 points))

Write the function removeDups below that removes duplicate copies of strings stored more than once from a sorted linked list. For example, given the list on the previous page, removeDups(list) should change the list to represent:

("dog", "elf", "frog")

You do not need to return removed nodes to the heap. Note that list is not passed by reference.

    void removeDups(ListNode * list)
    // pre: list is sorted
    // post: list is sorted, but contains no duplicates
PROBLEM 5: *(Shiftless (10 points))*

Part A (8 pts)
Write the function `lastToFirst` that moves the last node of a linked list so that it’s the first node. No new nodes should be created. For example, if `list` is `(a,b,c,d)` then the call `lastToFirst(list)` should change `list` to be `(d,a,b,c).

```c
void lastToFirst(ListNode * & list) {
    // pre: list represents (a,..., y, z)
    // post: list represents z, a, ..., y [move last node to first]
}
```

Part B (2 points)
Describe the list resulting from the code below if list contains `n` nodes. Assume `lastToFirst` works as specified, regardless of what you wrote in Part A.

```c
for(int k=0; k < n; k++) {
    lastToFirst(list);
}
```
PROBLEM 6: *(A Bird in the Hand is worth two in the Tree)*

The tree below is a binary search tree.

```
Lark
  Grouse
  Cardinal
  Seagull
  Penguin
  Vulture
```

**Part A (4 pts)**
Write the pre-order traversal of the tree.

**Part B (4 pts)**
Write what is printed by the function `print` below when passed a pointer to the root, “Lark” node above.

```c
void print(TreeNode * t)
{
    if (t != 0) {
        print(t->right);
        print(t->left);
        cout << t->info << endl;
    }
}
```

**Part C (4 pts)**
Add the strings “Flamingo, “Pelican”, and “Quail” to the tree (draw them) in that order, “Flamingo” first and “Quail” last.

**Part D (4 pts)**
In class we experimented with the following two implementations of a set of strings: a sorted vector and a binary search tree. Both data structures support $O(\log n)$ search for an element, but the binary search tree was much faster than the sorted vector. Why (be terse, but precise)?