Test 1: CPS 100

Owen Astrachan

February 23, 2000

Name: ________________________________
Login: __________
Honor code acknowledgment (signature) ________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>19 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>24 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>9 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>66 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 9 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 50 minutes.

In writing code you do not need to worry about specifying the proper \texttt{#include} header files. Assume that all the header files we’ve discussed are included in any code you write.

Class/struct declarations for nodes for trees and lists are on the last page of the test. You can remove this page to make it easier to have a reference to these classes when needed.
PROBLEM 1:  *(Check it twice)*

**Part A, 4 points**
Write the function `nodeCount` that returns the number of nodes in its parameter `list`.

```c
int nodeCount(Node * list)
// pre: list is NULL/0 terminated
// post: returns number of nodes in list
{
```

**Part B, 6 points**
Write the function `freeList` that returns all nodes in a list to the heap (by calling delete).

```c
void freeList(Node * & list)
// pre: list is NULL/0 terminated
// post: returns all nodes of list to heap (using delete)
//       list is NULL/0
{
```
Part C, 9 points
The function vec2list below constructs a linked-list storing the same values that are in a vector. Currently, its postcondition isn’t satisfied because the elements of the linked list are in the wrong order, they are backwards.

Node * vec2list(const tvector<string>& a)
// post: returns linked list with same values as a in same order
{
    Node * front = 0;
    int k;
    for(k=0; k < a.size(); k++)
    {
        front = new Node(a[k],front);
    }
    return front;  // return pointer to front of list
}

As written, the function has complexity \( O(n) \) for a list of \( n \) nodes.

Three methods are proposed for fixing the function so that its postcondition is satisfied. Next to each method below indicate whether it works correctly. You must draw a picture of the list returned if the vector contains the values \((1, 2, 3, 4)\) in that order. Draw a picture for each method.

I. Before returning front, add this code

```cpp
tstack<Node *> s;
Node * temp = 0;
while (front != 0)
{
    s.push(front);
    front = front->next;
}
while (! s.isEmpty())
{
    temp = s.top();
    s.pop();
    temp->next = front;
    front = temp;
}
```

II. Before returning front, add this code

```cpp
tqueue<Node *> q;
Node * temp = 0;
while (front != 0)
{
    q.enqueue(front);
    front = front->next;
}
while (! q.isEmpty())
{
    temp = q.front();
    q.dequeue();
    temp->next = front;
    front = temp;
}
```

III. Run the for loop from \( a.size()-1 \) down to zero rather than as written:

```cpp
for(k=a.size()-1; k >= 0; k--)
{
    front = new Node(a[k],front);
}
```
PROBLEM 2: (Bean Trees)

The tree shown below is a binary search tree.

```
              "lima"
             /   \
           "chickpea" "navy"
          /    /  \
       "black" "garbanzo" "pinto"
```

**Part A, 10 points**

1. What nodes of the tree above are leaves?

2. What is the postorder traversal of the tree above

3. Add the following strings, in order, to the tree. Draw new nodes attached to the tree above. Add “butter”, “kidney”, “soy”, and “red” in that order.

4. What are the minimum and maximum heights of a search tree storing 2048 = 2^{11} different strings?

**Part B, 8 points**

Write the function `swapCopy` that creates a new tree that’s a *swapped copy* of its parameter and returns a pointer to the root of the new tree. In a swapped copy, nodes with two-children are copied by interchanging the copies of the left and right subtrees. Nodes with no children or one child are simply copied (no swapping).

For the tree above, the swapped copy is shown below.

```
              "lima"
             /   \
           "navy" "chickpea"
          /    /  \
       "pinto" "garbanzo" "black"
```

(continued →)
Part C, 6 points

For this problem, the root is at level zero, the root’s children are at level one, and for any node, the node’s level is one more than its parent’s level. Write `levelCount` that returns the number of nodes on the specified level. For example, for the bean-tree diagrammed on the previous page, the call `levelCount(t,1)` should evaluate to 2 (chickpea and navy are on level 1); the call `levelCount(t,2)` should evaluate to 3; and the call `levelCount(t,4)` should evaluate to 0.

```c
int levelCount(Tree * t, int level)
// pre: 0 <= level
// post: returns # nodes at specified level in t
```
PROBLEM 3:  *A Recurring Nightmare*

**Part A, 4 points**

Write a recurrence relation that describes the running time of the function `value` below in the average case that trees are approximately balanced. For extra credit (3 points) solve the recurrence.

```c
int numNodes(Tree * t) {
    if (t == 0) return 0;
    return 1 + numNodes(t->left) + numNodes(t->right);
}

int value(Tree * t) {
    if (t == 0) return 0;
    int lcount = numNodes(t->left);
    int rcount = numNodes(t->right);
    if (lcount > rcount) {
        return 1 + value(t->left);
    } else {
        return 1 + value(t->right);
    }
}
```

**Part B, 4 points**

What is the complexity (using big-Oh) of the code below in terms of n. Justify your answer.

```c
int sum=0;
for(k=0; k < n; k += 2) {
    for(j=1; j <= k; j *= 3) {
        sum += j*k;
    }
}
```
A tree \( S \) is a \textit{subset} of another tree \( T \) if every value of \( S \) is contained in \( T \). There may be values in \( T \) that are not in \( S \), but every value stored in \( S \) must also stored in \( T \) for \( S \) to be a subset of \( T \). The empty tree is a subset of every tree (there are no values in the empty tree not in another tree).

For search trees, the function below correctly determines if a single value is contained in a tree.

```cpp
bool contains(Tree * t, const string& s)
// pre: t is a search tree
// post: returns true if t contains s, else return false
{
    if (t == 0) return false;
    else if (t->info == s) return true;
    else if (t->info < s) return contains(t->right,s);
    else return contains(t->left,s);
}
```

**Part A, 6 points**
Write the function \texttt{subset} that returns true if search tree \( S \) is a subset of search tree \( T \), and returns false otherwise.

Hint: traverse \( S \) in any order calling \texttt{contains} for each value in \( S \).

```cpp
bool subset(Tree * s, Tree * t)
// pre: s and t are both search trees
// post: returns true if s is a subset of t, false otherwise
{
    //
}
```

**Part B, 3 points**
What is the complexity of the code you wrote in Part A. In answering the question assume \( S \) contains \( n \) nodes and \( T \) contains \( m \) nodes, and that trees are roughly balanced (average case). The answer should be in terms of both \( n \) and \( m \). Justify your answer briefly.
PROBLEM 5:  (*MultiSet (6 points)*)

Multiset A is a *subset* of multiset B if every element of A (occurring \(k\) times in A) occurs at least \(k\) times in B.

For example

("ant", "bear", "cat", "ant")

is a subset of

("ant", "bear", "cat", "ant", "ant", "bear", "dog")

but it’s not a subset of

("ant", "bear", "cat", "cat", "cat", "cat", "cat")

because “ant” occurs only once in the second set.

A class *MSSubset* (next page) can be used to determine if a MultiSet a is a subset of another MultiSet b as follows.

```
MultiSet a,b;  // fill a and b with values

MSSubset sub(a);
b.apply(sub);
if (sub.isSubset()) cout << "a is a subset of b" << endl;
```

You must write the body of *MSSubset::apply* so that the class works as intended. The header file and member functions are on the last page.

```
void MSSubset::apply(const string & word, int count)
// sets myStatusOK to false when appropriate
```
struct Node
{
    string info;
    Node * next;
    Node (const string& s, Node * link)
        : info(s), next(link)
    {
    }
};

struct Tree
{
    string info;
    Tree * left;
    Tree * right;
    Tree(const string& s, Tree * lptr, Tree * rptr)
        : info(s), left(lptr), right(rptr)
    {
    }
};

#include "msapplicant.h"
#include "multiset.h"

class MSSubset : public MSApplicant
{
public:
    MSSubset(const MultiSet& set);
    bool isSubset() const;
    virtual void apply(const string & word, int count);

private:
    const MultiSet& mySet;
    bool  myStatusOK;
};

MSSubset::MSSubset()
    : mySet(set),
     myStatusOK(true)
{
}

bool MSSubset::isSubset() const
// post: returns false if subset relationship doesn’t hold
//       else returns true
{
    return myStatusOK;
}