CS 106X Sample Final Exam #2 ANSWER KEY

1. Array List Implementation (write)

```
// solution 1
void ArrayIntList::stretch(int k) {
    if (k <= 0) {
        // remove all elements from the list
        mySize = 0;
    } else {
        // resize array as needed to fit
        if (myCapacity < mySize * k) {
            int* bigger = new int[k * mySize]();
            for (int i = 0; i < mySize; i++) {
                bigger[i] = myElements[i];
            }
            delete[] myElements;
            myElements = bigger;
            myCapacity = k * mySize;
        }

        // stretch the elements
        for (int i = mySize - 1; i >= 0; i--) {
            for (int j = 0; j < k; j++) {
                myElements[i * k + j] = myElements[i];
            }
        }
        mySize *= k;
    }
}
```

2. Linked Lists (read)

```
front -> [0] -> [4] -> [2] -> [4] -> [6] -> [10] -> [6] -> [0] -> [1] /
```

3. Linked Lists (write)

```
// solution 1
void combineDuplicates(ListNode*& front) {
    if (front == nullptr) {
        return;    // empty list case
    }

    // combine at front of list, if needed
    // (common bug: front->data changes as you are looping!)
    int mergeValue = front->data;
    while (front->next != nullptr && front->next->data == mergeValue) {
        // merge with next node (and delete next node)
        front->data += front->next->data;
        ListNode* trash = front->next;
        front->next = front->next->next;
        delete trash;
    }

    // combine throughout rest of list
    ListNode* curr = front;
    while (curr->next != NULL) {
        mergeValue = curr->next->data;
        while (curr->next->next != nullptr && curr->next->next->data == mergeValue) {
            // merge with next node (and delete next node)
            curr->next->data += curr->next->next->data;
            ListNode* trash = curr->next->next;
            curr->next->next = curr->next->next->next;
            delete trash;
        }
        curr = curr->next;
    }
}

////////////////////////////////////////////////////////////////////////////////////
```

CS 106X Sample Final Exam #2   ANSWER KEY

4. Binary Trees (read)

(a) after adding all values:

```
         Cersei
     /             \
Arya                Jamie
    \            /          \
   Bran     Danaerys        Littlefinger
                   \                    \
                  Hodor                 Tyrion
                                        /      \
                                    Ned        Varys
                                       \
                                       Stannis
                                        /
                                      Ramsay
```

(b) **No**, overall tree is not balanced. Unbalanced nodes: **Tyrion, Littlefinger, Jamie, Cersei, Ned**

(c) after removing Littlefinger, Cersei, and Jamie: Allow either of the two trees below.

```
          Bran
     /             \
Arya                Hodor
                 /          \
            Danaerys          Tyrion
                              /      \
                          Ned        Varys
                             \
                             Stannis
                              /
                            Ramsay
```

```
        Danaerys
     /            \
Arya                 Ned
    \            /          \
   Bran      Hodor           Tyrion
                             /      \
                      Stannis       Varys
                      /
                    Ramsay
```

CS 106X Sample Final Exam #2   ANSWER KEY

5. Binary Trees (write)

```
// solution 1
bool hasPath(TreeNode* node, int start, int end) {
    return hasPathHelper(node, start, false, end);
}

bool hasPathHelper(TreeNode* node, int start, bool seenStart, int end) {
    if (node == nullptr) {
        return false;
    } else {
        seenStart = seenStart || node->data == start;
        bool seenEnd = seenStart && node->data == end;
        return (seenStart && seenEnd) ||
                hasPathHelper(node->left, start, seenStart, end) ||
                hasPathHelper(node->right, start, seenStart, end);
    }
}

/////////////////////////////////////////////////////////////////////////////////////

// solution 2
bool hasPath(TreeNode* node, int start, int end) {
    return hasPathHelper(node, start, end);
}

bool hasPathHelper(TreeNode* node, int start, int end) {
    if (node == nullptr) {
        return false;
    } else if (node->data == start) {
        return end == start || contains(node, end);
    } else {
        return hasPathHelper(node->left, start, end) ||
               hasPathHelper(node->right, start, end);
    }
}

bool contains(TreeNode* node, int end) {
    if (node == nullptr) {
        return false;
    } else if (node->data == end) {
        return true;
    } else {
        return contains(node->left, end) || contains(node->right, end);
    }
}

/////////////////////////////////////////////////////////////////////////////////////
```

6. Graphs (write)

```
// solution 1
Vector<Vertex*> findLongestPath(BasicGraph& graph) {
    graph.resetData();         // optional
    Vector<Vertex*> chosen;
    Vector<Vertex*> longest;

    // try to find longest path from every possible starting vertex
    for (Vertex* v : graph.getVertexSet()) {
        findLongestPathHelper(graph, chosen, longest, v);
    }
    return longest;
}

void findLongestPathHelper(BasicGraph& graph, Vector<Vertex*>& chosen,
                                          Vector<Vertex*>& longest, Vertex* start) {
    // "choose" this vertex  (mark as visited and add to path so far)
    start->visited = true;
    chosen.add(start);

    // remember what is the longest path we have seen so far
    if (chosen.size() > longest.size()) {
        longest = chosen;
    }

    // for each neighbor, explore
    for (Vertex* neighbor : graph.getNeighbors(start)) {
        if (!neighbor->visited) {
            findLongestPathHelper(graph, chosen, longest, neighbor);
        }
    }

    // "un-choose" this vertex  (un-mark and remove from path so far)
    start->visited = false;
    chosen.remove(chosen.size() - 1);
}

/////////////////////////////////////////////////////////////////////////////////////
```

7. Hashing (read)

```
HashMap map;
map.put(19, 9);
map.put(4, 4);
map.put(44, 19);
map.remove(9);
map.put(23, 54);
map.put(73, 54);
map.put(83, 9);
map.put(99, 4);
map.remove(4);
map.put(0, 0);
map.put(-2, -88);
if (!map.containsKey(73)) {
    map.put(66, 77);
}
map.put(333, 0);
```

```
      +---+
0     |   |--> 0:0
      +---+
1     | / |
      +---+
2     |   |--> -2:-88
      +---+
3     |   |--> 83:9 --> 23:54
      +---+
4     |   |--> 44:19
      +---+
5     | / |
      +---+
6     | / |
      +---+
7     | / |
      +---+
8     | / |
      +---+
9     | / |
      +---+
10    | / |
      +---+
11    | / |
      +---+
12    | / |
      +---+
13    |   |--> 333:0 --> 73:54
      +---+
14    | / |
      +---+
15    | / |
      +---+
16    | / |
      +---+
17    | / |
      +---+
18    | / |
      +---+
19    |   |--> 99:4 --> 19:9
      +---+

size        =  9
capacity    =  20
load factor =  0.45
```

8. Inheritance and Polymorphism (read)

```
var1->m1();                     // Cecil m1 / Kain m1
var1->m2();                     // Cecil m1 / Kain m1 / Kain m2
var1->m3();                     // COMPILER ERROR
var2->m3();                     // Rosa m3 / Cecil m1 / Kain m1 / Kain m2
var3->m2();                     // Golbez m2 / Cecil m1 / Kain m1
var3->m4();                     // COMPILER ERROR
var4->m3();                     // Cecil m3 / Cecil m2 / Cecil m1 / Kain m1

((Cecil*) var1)->m3();          // Cecil m3 / Cecil m1 / Kain m1 / Kain m2
((Rosa*) var1)->m4();           // CRASH / RUNTIME ERROR
((Rosa*) var2)->m4();           // Rosa m4
((Golbez*) var3)->m4();         // COMPILER ERROR
```

9. Object-oriented Programming and Inheritance (write)

```
// MemoryCalculator.h
class MemoryCalculator : public Calculator {
public:
    MemoryCalculator(int seed);
    virtual int kthPrime(int k);
    virtual int getComputeCount() const;
    virtual int getMemoCount() const;

    bool operator ==(const MemoryCalculator& mc2) const;   // okay to declare outside class
    bool operator !=(const MemoryCalculator& mc2) const;

private:
    Map<int, int> m_primes;
    int m_memoCount;
};

// MemoryCalculator.cpp
MemoryCalculator::MemoryCalculator(int seed)
        : Calculator(seed) {   // call superclass constructor
    m_memoCount = 0;
}

int MemoryCalculator::kthPrime(int k) {
    if (m_primes.containsKey(k)) {
        m_memoCount++;                      // computed this prime before; retrieve from cache
        return m_primes[k];
    } else {
        int kth = Calculator::kthPrime(k);
        m_primes[k] = kth;
        return kth;                         // never computed before; compute and memoize
    }
}

int MemoryCalculator::getComputeCount() const {
    return m_primes.size();
}

int MemoryCalculator::getMemoCount() const {
    return m_memoCount;
}

bool MemoryCalculator::operator ==(const MemoryCalculator& mc2) const {
    return getSeed() == mc2.getSeed()
            && getComputeCount() == mc2.getComputeCount()
            && getMemoCount() == mc2.getMemoCount()
            && m_primes == mc2.m_primes;
}

bool MemoryCalculator::operator !=(const MemoryCalculator& mc2) const {
    return !(*this == mc2);    // okay to redundantly repeat == code but negated
}
```