Name: Chun-Kai Wang

USC netID (e.g., ttrojan): _____

CS 455 Final Exam
Spring 2015 [Bono]

May 13, 2015

There are 10 problems on the exam, with 73 points total available. There are 10 pages to the exam, including this one; make sure you have all of them. For on-campus students: if you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there).

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and **circling** your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC Net ID at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1	6 pts.	
Problem 2-5	12 pts.	
Problem 6	9 pts.	
Problem 7	5 pts.	
Problem 8	6 pts.	
Problem 9	20 pts.	
Problem 10	15 pts.	
TOTAL	73 pts.	

Problem 1 [6 points]

The following **Java** code attempts to use recursion to compute the maximum value in an array of numbers, but it has a bug:

```
// max: Computes the maximum value in array nums
// PRE: nums.length >= 1
public static int max(int[] nums) {
   return maxR(nums, 0);
}

// maxR (recursive helper function for max)
// Computes the maximum value in the sub-array of nums in positions startLoc
//   through nums.length-1
// PRE: nums.length >= 1
private static int maxR(int[] nums, int startLoc) {
   int maxSoFar = 0;
   if (startLoc == nums.length) {
      return maxSoFar;
   }
   if (nums[startLoc] > maxSoFar) {
      maxSoFar = nums[startLoc];
   }
   return maxR(nums, startLoc + 1);
}
```

Part A. Give a sample input array for max and the corresponding return value for max, such that the code above computes the *wrong* value for that array:

nums: (2, 1)

max(nums): 0

Part B. Give a sample input array for max and the corresponding return value for max, such that the code above computes the *correct* value for that array:

nums: -1, 0

max(nums): 0

Problem 2 [2 points]

Not counting any eliminated in the first iteration, roughly how many elements are eliminated from further consideration in the **second** loop iteration of a binary search in an array of *n* elements (i.e., assuming we didn't find the value in that iteration)?

n/4

Problem 3 [2 points]

What is true about the first *k* elements in the array before pass *k* of insertion sort? (circle the answer that best describes what is known)

a. they are *k* values in sorted order

b. they are the *k* smallest values

c. they are the *k* smallest values in sorted order

d. none of the above

Problem 4 [4 points]

If you wanted to sort an array of `Strings` in decreasing order by length using the **Java** `Arrays.sort` method you would also need what additional code (besides the call to the sort method)? (describe briefly– do not write code)

need to pass a comparator object into Arrays.sort as the second parameter
the comparator should implement Comparator interface,
and implement the method compare(), let it return, let it return negative numbers when its 1st parameter (String) is longer its 2nd parameter.

Problem 5 [4 points]

The **Java** `ListIterator` interface (used for iterating over a `LinkedList`) includes methods `hasNext()` and `next()`. The `Iterator` interface (used for iterating over a `Set`) also has methods `hasNext()` and `next()`.

In addition, the `ListIterator` interface also has the following method (where `ElmtType` is the type of an element in the `LinkedList`):

```
// Replaces the last element returned by next with the specified element
public void set(ElmtType newVal) . . .
```

Why is this method not provided in the `Iterator` interface?

Because it's not allow to directly change a element in a Set.

Problem 6 [9 points]

Consider the following **Java** code to compute the number of occurrences of each score from an array of scores:

```
// Returns an array of the number of occurrences of each score from
//   scores array.
//   E.g., after we call it as follows:
//           int[] counts = getCounts(scores, 100);
//   counts[0] is the number of 0's in scores,
//   counts[1] is  the number of 1's in scores, . . . ,
//   counts[100] is the number of 100's in scores
// PRE: all the elements in scores are in the range [0,maxScore]
public static int[] getCounts(ArrayList<Integer> scores, int maxScore) {

   int[] counts = new int[maxScore+1];  // all values init'd to 0

   for (int i = 0; i < scores.size(); i++) {

      for (int score = 0; score <= maxScore; score++) {

         if (scores.get(i) == score) {

            counts[score]++;    count[scores.get(i)]++;

         }

      }

   }
   return counts;
}
```

Part A [2]. What is the worst case big-O time for the **current** `getCounts` method in terms of `scores.size()` and `maxScore`?

O(scores.size() x maxScore)

Part B [5]. Modify the `getCounts` function to improve the big-O running time. You can make your modifications directly in the code above, using arrows to show where your new code should be inserted, crossing out code that you would get rid of, etc.

Part C [2]. What is the worst case big-O time for your **new version** of `getCounts`?

O(score.size())

Problem 7 [5 points]

(**Java**) Consider a spell-checker that finds misspellings in a document, along with the number of times each misspelling occurred, and prints them out in alphabetical order.

Example input document:

```
hte bigest burger wase the best burger and hte wort bruger was teh
smallest bruger hte gril prefered the big burger
```

Example output:

```
bigest 1
bruger 2
gril 1
hte 3
prefered 1
teh 1
wase 1
wort 1
```

Write variable declaration(s) for storing the data to solve this problem, such that we can use this data structure to solve this problem efficiently; also explain your approach via the prompts below. You may use Java library classes. Note: The spell-checker will also have at its disposal a dictionary of English words stored in a `HashSet`.

Variable declaration(s):

```
Map<String, Integer> map = new HashMap<>();
```

Description of what problem data is stored in this data structure and how it's organized:

the map contains entries with key = String which couldn't be found in the dictionary, and with value = the occurance of the key string

Description of how we are using the data structure to solve the problem (1-2 sentences maximum; do not write code):

for each word in the document {
  check if the dictionary contains this word
  ↳ if not → check if "map" contains this word
    ↳ if not → put an entry into the map, with key = this word and value = 1
    ↳ if yes → increase the value of the entry (with key = word) by 1

}

Problem 8 [6 points]

Suppose we are creating a `Queue` class in **C++**. This `Queue` class will be for a queue of `int`s (i.e., it won't be a template class). Furthermore suppose we have a `main` program that uses the `Queue` class, and the whole program is in a single file, whose contents is shown here (some details left off so it all fits here). The different program elements are numbered at right for your convenience. (Note: unlike the later Queue problem, this one does not use any additional outside class or struct in its implementation.)

```
#include <iostream>                                   (1)

using namespace std;                                  (2)

class Queue {
 public:
  Queue();
  bool isEmpty() const;
  int front() const;
  void enQ(int val);
  int deQ();

 private:
  . . . [instance variables here]
};                                                    (3) → Queue.h

Queue::Queue() {
  . . .
}

bool Queue::isEmpty() const {
  . . .
}

int Queue::front() const {
  . . .
}

. . . [other Queue method definitions here]           (4) → Queue.cpp

int main() {
   Queue q;

   while (cin >> val) {
      q.enQ(val);
   }

   while (!q.isEmpty()) {
      cout << q.deQ() << " ";
   }
}                                                     (5) → qProg.cpp
```

[problem continued next page]

Problem 8 (cont.)

In the space provided below show what would be in each file for a version of this program that can use separate compilation. That is, we want to be able to compile the `Queue` module separately from the program that uses queues. We're calling that client program `qProg.cpp`. Show everything that would need to go in each file, such that, for things that were in the original file, you can **use the identifying numbers from the previous page** (i.e., you do not need to rewrite the code from the previous page) to indicate the contents that those numbers refer to. For your convenience, we provided the `#ifndef` – stuff that goes in header files.

```
#ifndef QUEUE_H
#define QUEUE_H
 ③



#endif
```

Queue.h

~~① ②~~

```
# include "Queue.h"
④
```

Queue.cpp

① ②

```
# include "Queue.h"
⑤
```

qProg.cpp

Problem 9 [20 points]

Implement the static Java method `isPermutation` which returns true iff its array parameter `b` is a permutation of its array parameter `a`, where we assume `a` has no duplicate values. Your solution may use no additional arrays.

Examples below:

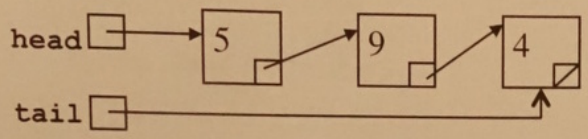
`a`	`b`	`isPermutation(a, b)`
`[1, 2]`	`[3, 2, 1]`	`false`
`[1, 2]`	`[2, 1]`	`true`
`[1, 3, 5]`	`[4, 2]`	`false`
`[1, 3, 5]`	`[3, 2, 1]`	`false`
`[]`	`[]`	`true`
`[3]`	`[3]`	`true`
`[2, 5, 4, 7]`	`[7, 5, 2, 4]`	`true`

```
// is array b a permutation of array a?
// (a and b are not changed by this method)
// PRE: a has no duplicate values
public static boolean isPermutation(int[] a, int[] b) {
    if ( a.length != b.length ) return false;
    for ( int i : a ){
        boolean found = false;
        for ( int j : b ){
            if ( i == j ){
                found = true;
                break;
            }
        }
        if ( !found ) return false;
    }
    return true;
```

Problem 10 [15 points]

Complete the implementation of the *C++* class `Queue`, below, which stores a queue of `int`s. You are required to use a linked list with head and tail pointer, described further here, as the representation.

A linked list with head and tail pointer means one where we store pointers to *both* the first element in the list (the "head") and the last element in the list (the "tail"). That means, instead of carrying around one variable for a linked list, we would need two variables. In this problem those two variables will be instance variables of the `Queue` class. Here's a diagram of a linked list with head and tail pointer that has three values in it:



With this representation, **your implementation must do all the operations in constant time.** In contrast, if we used a regular singly linked list instead, i.e., like the ones we've used in pa5 and in lab, not all the operations would be constant time. **Also, to get full credit, your code must have no memory leaks.**

Here is the class definition including instance variables. Complete the method definitions on the next page:

```
class Queue {

 public:
  Queue() // create an empty queue
  bool isEmpty() const;   // is the queue empty?
  int front() const;     // return front element in queue (queue is unchanged)
                         // pre: !isEmpty()
  void enQ(int val); // put a value on the queue
  int deQ();    // take a value off the queue and return it
                    // pre: !isEmpty()

 private:
   Node *head;
   Node *tail;
};
```

You may assume the following definitions have already been made, and you can use them:

```
struct Node {
  int data;
  Node * next;
  Node() { data = 0; next = NULL; }
  Node(int d) { data = d; next = NULL; }
  Node(int d, Node * n) { data = d; next = n; }
};
typedef Node * ListType;
```

[space for your answer on the next page]

Problem 9 (cont.)

```
// create an empty queue
Queue::Queue() {
    head = NULL;
    tail = NULL;
}

// is the queue empty?
bool Queue::isEmpty() const {
    return head == NULL;
}

// return front element in queue (queue is unchanged)
// pre: !isEmpty()
int Queue::front() const {
    return head->data;
}

// put a value on the queue
void Queue::enQ(int val) {
    Node* n = new Node(val);
    tail->next = n;
}

// take a value off the queue and return it
// pre: !isEmpty()
int Queue::deQ() {
    int temp = tail->data;
    if (head == tail) {
        delete head;
        head = NULL;
        tail = NULL;
        return temp;
    }
    Node* prev = head;
    while (prev->next != tail) { prev = prev->next }
    delete tail;
    tail = prev;
    tail->next = NULL;
    return temp;
}
```

Name: Chun-Kai Wang

USC NetID (e.g., ttrojan): _____

CS 455 Final Exam
Fall 2015 [Bono]

Dec. 15, 2015

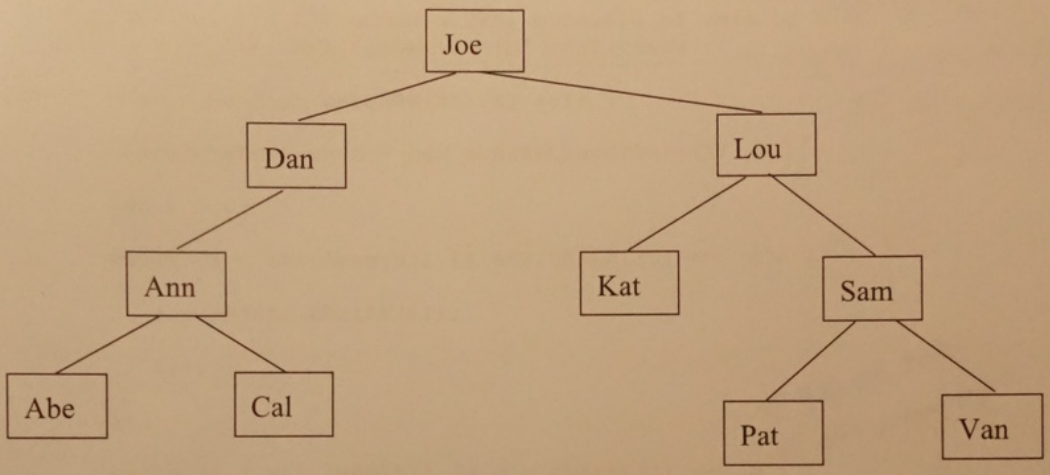
There are 6 problems on the exam, with 70 points total available. There are 10 pages to the exam (5 pages double-sided), including this one; make sure you have all of them. If you need additional space to write any answers, pages 8, 9, and 10 are left blank for that purpose. If you use these pages for answers you just need to direct us to look there.

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and **circling** your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC username (a.k.a., NetID) at the top of the exam. Also put your NetID at the top right of the front side of each page of the exam. Please read over the whole test before beginning. Good luck!

Problem 1 [6 pts. total]

Part A. Consider the following binary search tree, whose root is shown nearest the top of the page (i.e., it's not a sideways tree):



A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

For each of the following lookups from the tree shown above, give the sequence of keys that the target key would have to be compared with to do the lookup.

Part A. lookup *Cal*

Joe → Dan → Ann → Cal

Part B. lookup *Molly*

Joe → Lou → Sam → Pat

Part C. lookup *Fred*

Joe → Dan

Problem 2 [2 pts.]

Give the big-O worst case time for checking if an array of size *n* is already sorted.

O(n)

USC NetID: _____

Problem 3 [10 points]

The following **Java** method is supposed to do what its method comment says below, but it doesn't work correctly in all cases (see the code handout for more about the `Stack` class):

```
/**
   Returns true iff the given string consists of zero or more 'a' characters
   followed by the same number of 'b' characters.
*/
public static boolean sameABs(String str) {

        Stack<Character> s = new Stack<Character>();

        int i = 0;

        while (i < str.length() && str.charAt(i) == 'a') {

           s.push(str.charAt(i));

           i++;

        }

        while (i < str.length() && str.charAt(i) == 'b') {

           s.pop();

           i++;

        }

        return (i == str.length() && s.empty());

    }

}
```

Part A [6]. Come up with some test cases for the function and verify whether they work on the code above. While there are more interesting test cases than just three, **write down three specific test cases as described below:**

A string for which `sameABs` correctly returns `true`: "a b"

A string for which `sameABs` correctly returns `false`: "a a b"

A string for which `sameABs` does not work correctly: "a b b"

Part B [4]. Fix the code so it works for any string. You should make your modifications directly in the code above, using arrows to show where your new code should be inserted, crossing out code that you would get rid of, etc.

Problem 4 [20 points]

In lecture and a lab we discussed **Java** code to create versions of a concordance that computed the number of occurrences of each distinct word in a file. **Here we're going to make a version of our Concord class that creates a concordance that has all the *locations* of each distinct word in a file. We will use the line number as a word's location.**

Here is an example of some input, and the corresponding output when we build a concordance for it and print it (just using `toString`-type format):

```
the big dog went to the big dog
and the other big dog went to the
the big elephant
```

```
{and=[2], big=[1, 2, 3], dog=[1, 2], elephant=[3], other=[2], the=[1, 2, 3], to=[1,
2], went=[1, 2]}
```

TreeMap < String , ArrayList< Integer >>

Note that even if a word appears more than once on a line, we only list that line number once. Some more details of the problem:

- You don't have to worry about putting words into some canonical form for the purposes of this problem (e.g., you may assume there is no punctuation or capital letters in the input).
- **For a file with *n* words your code must build the concordance in at worst *nlogn* time, and be able to print it out in alphabetical order in linear time** (you are not required to write the `toString` code).
- It is expected that you will use the Java library to make your code short and fast. See the code handout for a reminder of some classes and methods. You will be evaluated in part on appropriate use of these tools.
- We have written the code to read the file for you (see next page). **You need to fill in the private data, complete the constructor, and complete `build` by implementing the private method `processLine`.**

Write your answer in the spaces provided on the next page.

USC NetID: _____

Problem 4 (cont.)

```
public class Concord {
   // put private data here:
   private Map < String , ArrayList<Integer>> map ;

   public Concord() {          // creates an empty concordance
       map = new TreeMap<>();
   }
   public String toString() {  /* you don't have to write this method */ }

   public void build(Scanner in) {  // builds the concordance from data read from "in"
     int currLineNum =  1;
      while (in.hasNextLine()) {
         String currLine = in.nextLine();
         processLine(currLine, currLineNum);  // calls helper function below
         currLineNum++;
      }
   }
   // (helper func) adds all the words in this line to the concordance
   // as occurring on lineNum
   private void processLine(String line, int lineNum) {
       Scanner in  = new Scanner(line);
       while (in.hasNext()){
           ArrayList arrlist;
           String str = in.next();
           arrlist = (map.containsKey(str)) ? map.get(str) : new ArrayList<>();
           arrlist.add(lineNum);
           map.put(str, arrlist);
       }
   }
}
```

Problem 5 [10 pts. total]

Consider the following C++ code to dynamically create a `Student` object: (Note: the `Student` class is not shown; you may assume it has a constructor that takes a `String` argument.)

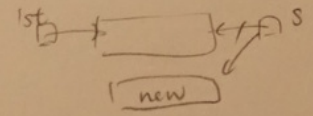
```
Student * joe;              // create a pointer to an object in C++
joe = new Student("Joe");   // dynamically allocate an object in C++
```

Consider each of the four possible headers for a C++ function foo that takes a student as a parameter.

Part A [4]. For each of the headers below, to the right of that header show the corresponding call to pass the student, `joe`, created in the code above, to the function. We started (but didn't finish) the first one for you.

main
String st.
若傳 string，則 java 再用 foo2 之方法.
foo
s

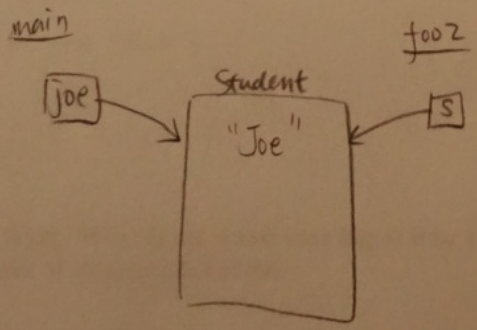
Header	Corresponding call
`void foo1(Student s);`	`foo1( *joe )`
`void foo2(Student * s);`	`foo2 (joe)` → 可改變 joe 所指向的 obj. 不能改變 joe 的內容 (不能使 joe 指向其它 obj)
`void foo3(Student * & s);`	`foo3 (joe)`
`void foo4(const Student * & s);`	`foo4 (joe)`



Part B [2]. In Java all objects are created dynamically (i.e., with new), as was the C++ object created at the top of the page. **Which of the above four function headers and calls most closely corresponds to the semantics of how objects are passed in Java? Write the name of that function below:** [By semantics we mean how it works, not what its syntax is.]

foo2

Part C [4]. Draw a box and pointer diagram for the call to the function you indicated in part B. Show all variables used and their values.



USC NetID: _____

Problem 6 [22 points total]

Part A [20]. Write the C++ Boolean function `allUnique`, which takes a linked list of `int`s and returns true iff all the values in the list are unique (i.e., there are no duplicate values in the linked list). Restrictions: the list must be unmodified by the function and the function is only allowed to use a constant amount of extra memory. (In contrast, the `sameABs` problem earlier on the exam used O(n) extra memory for a Stack).

The `Node` and `ListType` definitions are on the code handout.

Examples:

list	***return value of call to `allUnique(list)`***
`()`	`true`
`(1)`	`true`
`(3 2 1)`	`true`
`(1 1 1)`	`false`
`(3 2 3)`	`false`
`(4 3 7 3 4 5)`	`false`

```
// PRE: list is a well-formed list.
bool allUnique(ListType list) {
  if (list == NULL || list->next == NULL) return true;
  ListType ref = list;
  ListType curr;
  while (ref->next != NULL){
    curr = ref->next;
    while (curr != NULL){
      if (curr->data == ref->data) return false;
      curr = curr->next;
    }
    ref = ref->next
  }
  return true;
}
```

Part B [2]. What is the worst case big-O time for your solution to part A, as a function of, n, the number of elements in ths list:

$O(n^2)$

Name: Chun-Kai Wang

USC NetID (e.g., ttrojan): _____

CS 455 Final Exam
Spring 2016 [Bono]

May 11, 2016

There are 7 problems on the exam, with 76 points total available. There are 10 pages to the exam (5 pages double-sided), including this one; make sure you have all of them. There is also a one-page double-sided code handout that accompanies the exam. If you need additional space to write any answers, pages 9 and 10 of the exam are left blank for that purpose. If you use these pages for answers you just need to direct us to look there. *Do not detach any pages from this exam.*

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and **circling** your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC username (a.k.a., NetID) at the top of the exam. Also put your NetID at the top right of the front side of each page of the exam. Please read over the whole test before beginning. Good luck!

Problem 1 [8 points]

[C++] Suppose we are developing some linked list functions in C++ (called f1, f2, f3, and f4). The functions are currently defined and used in the following single-file program (some details left off so it all fits here.) The different program elements are numbered at right for your convenience. *(problem continued on the next page)*

```
#include <iostream>                    (1)

using namespace std;                   (2)

struct Node {                          (3) listFunc.h
 int data;
 Node *next;
 Node(int item);
 Node(int item, Node *n);
};

typedef Node * ListType;               (4) listFunc.h (是否要在.cpp中?)

Node::Node(int item) { . . .}          (5) listFunc.cpp
Node::Node(int item, Node *n) { . . .}

// Note: the following 4 functions do no I/O   (6) listFunc.cpp
void f1(ListType list) { . . . }
void f2(ListType list) { . . . }
int f3(ListType list) { . . . }
void f4(ListType & list, int n) { . . }

int main() {                           (7) testListFunc.cpp
   ListType myList;
   . . .
   f1(list);
   f4(list, 3);
   cout << f3(list);
   f2(list);

   return 0;
}
```

declar

defin

在.h中会先declaration ↑↳④

USC NetID: _____

Problem 1 (cont.)

In the space provided below show what would be in each file for a version of this program that can use separate compilation. That is, we want to be able to compile the module with the list functions (to be in `listFuncs.h` and `listFuncs.cpp`) separately from the program that uses them. We're calling that client program `testListFuncs.cpp`. Show everything that would need to go in each file, such that, for things that were in the original file, you can **use the identifying numbers from the previous page** (i.e., you do not need to rewrite the code from the previous page) to indicate the contents that those numbers refer to. For your convenience, we provided the `#ifndef` – stuff that goes in header files. Also, we already showed where (5) would go.

`listFuncs.h`

```
#ifndef LIST_FUNCS_H
#define LIST_FUNCS_H

(3) (4)
void f1(ListType);
void f2(ListType);
   :

#endif
```

declaration

`listFuncs.cpp`

```
(2) ? #include "listFuncs.h"
(1)
(5)  // we did this part for you
(6)
```

`testListFuncs.cpp`

```
(1) (2) (4) #include "listFuncs.h"

(7)
```

Problem 2 [7 points]

Give the big-O worst case time to solve each of the following problems (put your answers in the space provided to the left):

O(n) 1. compute the minimum and maximum values in an array of size *n*

O(log n) 2. search for a value in a sorted array of size *n*

O(n) 3. reverse the values in an array of size *n*

O(n) 4. do `list.get(i)` where `list` is a Java `LinkedList` object of size *n*

O(1) 5. do `list.get(i)` where `list` is a Java `ArrayList` object of size *n*

$O(n^2)$ 6. use insertion sort to sort *n* values in an array

O(n) 7. determine whether a linked list is already sorted

Problem 3 [4 points]

Suppose you are implementing your own Stack class. For each of the following, answer one of: front, end, or doesn't matter.

Part A. For a singly-linked list representation of a Stack, it would ~~me~~ be most efficient to keep the top of the stack at which end of the linked list? [Note: a singly-linked list is the kind we've been implementing in C++]

front

Part B. For a partially-filled array representation of a Stack, it would me most efficient to keep the top of the stack at which end of the partially-filled array?

end

Problem 4 [2 points]

Explain one benefit of implementing and testing a subset of a class (or program) before implementing the rest of the class (or program). (2 sentences maximum)

then, if error happens, it should come from the code of the subset.
and the debugging would be easier ^ fewer codes to be checked.
because a subset would have

USC NetID: _____

Problem 5 [15 points]

[Java] Write a *recursive* Java method to find the number of occurrences of a particular value in an array. A solution that doesn't use recursion will receive little to no credit. Hint: you will need a helper method to do the actual recursion.

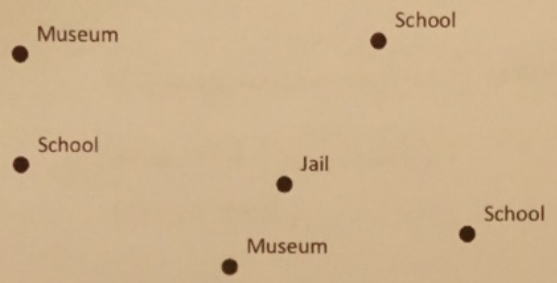
Examples (array shown as comma-separated values with brackets):

nums	***val***	***countVals(nums,val)***
`[4,5,3]`	5	1
`[-4,3,-10,3]`	3	2
`[2]`	3	0
`[]`	5	0
`[5,5,5,5]`	5	4

```
// returns the number of occurrences of the value val in the array nums
// (uses recursion)
public static int countVals(int[] nums, int val) {
    return helper(nums, val, 0);
}

private static int helper(int[] nums, int val, int pos){
    if(pos == nums.length) return 0;
    int match = (nums[pos]==val)? 1 : 0;
    return match + helper(nums, val, pos+1);
}
```

Problem 6 [20 points]

[Java] Suppose we want to annotate a map of a city with interesting sights. We might want to know the locations of museums or schools, for example. Here's an example of such data as it might appear on a map (other map features are not shown):



This problem concerns implementing a class to store the data for the sights. For the purposes of this problem we'll assume that any given map location can only have *one* label: e.g., location `(3207, 1245)` is a `School`. But the same label can occur at multiple locations: e.g., there may be three schools in town, thus 3 different points have the label `School`, as in the example above. Also, we'll assume that locations on a map will be represented using the Java `Point` class (that is not be a realistic way to represent a geographic location, but we'll use it here to simplify the problem), and that their coordinates can have any value, including negative values.

Our `Sights` class will have three methods in addition to the constructor: one for adding a new labeled location, one for getting the label for a location, and one for getting all the locations that have a particular label. This and the next page have the exact specification of these methods.

Implement the `Sights` class. For full credit, your representation/implementation should be as efficient as possible, taking advantage of Java library classes we have used this semester.

Note: The code handout has more information about the Java library, including the `Point` class.

```
public class Sights {
   // put private data here:

      Map<Point, String> map;


   public Sights() {        // creates an empty Sights object

         map = new HashMap<>();


   }
```

[class definition continued on the next page]

USC NetID: _____

Problem 6 (cont.)

```
// if this location isn't already part of these Sights, add the given
// (loc, label) pair to the Sights, and returns true.
// otherwise, returns false, and Sights object is unchanged
public boolean add(Point loc, String label) {

    if (map.containsKey(loc)) return false;
    map.add(loc, label);
    return true;

}

// returns the label for this location or null if this location isn't in the Sights
public String getLabel(Point loc) {

    if (map.containsKey(loc)) {
        return map.get(loc);
    }
    return null;

}

// returns the list of locations that have this label
// or null if there are no sights with this label
public ArrayList<Point> getPoints(String label) {

    Set<Map.Entry<Point, String>> set = map.entrySet();
    ArrayList<Point> arrlist = new ArrayList<>();
    for (Map.Entry<Point, String> e : set) {
        if (e.getValue().equals(label)) {
            arrlist.add(e.getKey());
        }
    }
    return arrlist;

}
```

Problem 7 [20 points]

[C++] Write the C++ function `compress`, which takes a linked list of `int`s passed by reference, and replaces each run in the list with a single value. A run is a sequence of two or more of the same value all next to each other. Your code should have no memory leaks (this includes reclaiming any memory no longer used).

The `Node` and `ListType` definitions are on the code handout.

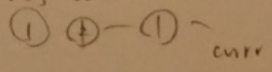
Examples:

`list`	***`list` after call to `compress(list)`***
`()`	`()`
`(1)`	`(1)`
`(3 2 1)`	`(3 2 1)`
`(1 1 1)`	`(1)`
`(3 2 2 3)`	`(3 2 3)`
`(2 2 3 3 3 3 1 2 2)`	`(2 3 1 2)`

```
// PRE: list is a well-formed linked list.
void compress(ListType & list)
```

```
if (list == NULL || list->next == NULL) return ;

ListType ref = list;
ListType curr = list->next;

while (curr != NULL){
    if (curr->data == ref->data){
        ListType temp = curr;
        curr = curr->next;
        delete temp;
        ref->next = curr;
    } else {
        ref = curr;
        curr = curr->next;
    }

return;
```

ref curr



Name: Chun-Kai Wang

USC NetID (e.g., ttrojan): _____

CS 455 Final Exam
Fall 2016 [Bono]

December 13, 2016

There are 6 problems on the exam, with 67 points total available. There are 10 pages to the exam (5 pages double-sided), including this one; make sure you have all of them. There is also a one-page code handout that accompanies the exam. If you need additional space to write any answers, pages 8, 9, and 10 of the exam are left blank for that purpose. If you use these pages for answers you just need to direct us to look there. ***Do not detach any pages from this exam.***

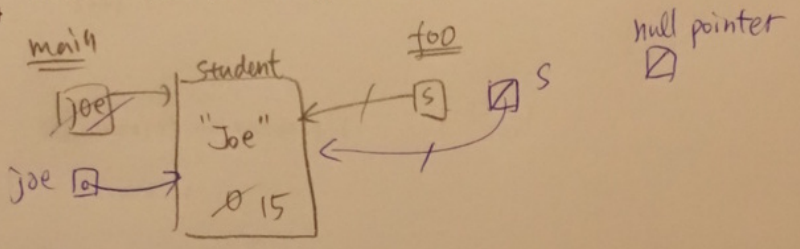
Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and **circling** your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC username (a.k.a., NetID) at the top of the exam. Also, put your NetID at the top right of the front side of each page of the exam. Please read over the whole test before beginning. Good luck!

Problem 1 [10 points]

[C++] Consider the following two C++ programs that use a `Student` class and answer the questions below each one (see code handout for documentation of `Student` class methods):

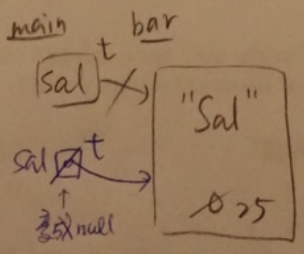
```
// Program 1

void foo(Student * s) {
   s->addScore(15);
   s = NULL;
}

int main() {
   Student * joe = new Student("Joe");
   foo(joe);
   cout << joe->getScore() << endl;
}
```



Part A. In the space above, draw a box-and-pointer diagram (a.k.a., memory diagram) showing all object variables, objects, and their state as they have changed during the code sequence. This includes showing `foo`'s parameters.

Part B. What is the output of Program 1? 15

```
// Program 2

void bar(Student * & t) {
   t->addScore(25);
   t = NULL;
}

int main() {
   Student * sal = new Student("Sal");
   bar(sal);
   cout << sal->getScore();
}
```



Part C. In the space above, draw a box-and-pointer diagram for Program 2 like you did for Part A.

Part D. What is the output of Program 2? crash!

the sal is now NULL, the sal->getScore() might throw exceptions or have runtime error

USC NetID: _____

Problem 2 [8 points]

[Java] Consider the following code to do linear search in an array (written in Java). It doesn't currently work in all cases that satisfy the precondition.

```
// returns the location of target in nums, or -1 if not found.
// PRE: nums is not null and nums has no duplicates
public static int lookup(int[] nums, int target) {

   inti  = 0;

   while (nums[i] != target) {

      i++;

   }

   if (nums[i] == target) {

      return i;

   }

   else {

      return -1;

   }

}
```

&& i < nums.length

Part A. Provide a test case where the code does **not** work correctly, and the corresponding result of that test case. (Show your answer under the headings below.)

nums	**target**	**result**
(0, 1, 2)	3	the while loop won't stop when the index is out of array's boundary

Part B. Provide a test case where the code **does** work correctly, and the corresponding result of that test case. (Show your answer under the headings below.)

nums	**target**	**result**
(0, 1, 2)	2	2

Part C. Fix the code above. Do not rewrite the whole method, but rather make minimal changes right into the code above, using arrows to show where your new code should be inserted, *crossing out* code that you would get rid of, etc.

Problem 3 [10 points]

[Java] The binary search algorithm can be implemented recursively. Below is a partially complete recursive binary search implementation (consists of the two functions below). **Complete the code below so it does what its comments say. The only missing parts are indicated by the boxes and underlined areas. (i.e., fill in the boxes and underlined areas provided.)**

```
// returns the location of target in words array, or -1 if not found.
// PRE: words is in alphabetical order and has no duplicates
public static int binSearch(String[] words, String target) {
   return binSearchR(words, target, 0, words.length-1);
}


// returns the location of target in the part of words with indices [low,high]
// or -1 if not found
// PRE: words is in alphabetical order and has no duplicates
public static int binSearchR(String[] words, String target, int low, int high) {
   if (low >high) {
      return -1;
   }
   int mid = (low + high) / 2;
   if (target.equals(words[mid])) {
      return mid;
   }
   else if (target.compareTo(words[mid]) < 0) {    // target < words[mid]

      return binSearchR(words, target, low, mid-1);
   }
   else {                                          // words[mid] < target

      return binSearchR(words, target, mid+1, high);
   }
}
```

lo hi
mid

USC NetID: _____

Problem 4 [4 pts]

[Java] Recall that the `Concord` class computes the number of occurrences of all words from a document. Consider the two variations of our `Concord` class below. Version A is the one we did in lecture and lab: it takes a `Scanner` as a *parameter* to the `addData` method. Version B's `addData`, instead creates a *local* variable for the `Scanner` (and initializes it inside the method). To highlight the differences between the two they are shown in **bold** in the code below. (The other methods (not shown) of both classes are a constructor that creates an empty version of the concordance (i.e., with no words in it), and another method to print out the results.)

Describe a scenario that illustrates an advantage of Version A over Version B. Limit your answer to one to two sentences.

version B can only use System.in as input source,
while version A can use other sources (ex. File, String) as input.
↳ also ver.A can read several times (just build diff. scanner obj)
but ver.B can only read once (from the screen)

Version A.

```
// Computes the number of occurrences of all words from a Scanner
public class Concord {
    // . . . [other details of Concord left out]

    /**
      Add data from Scanner to concordance.
      @param in data to scan.  "in" will be at the end of its data after this
      operation.
     */
    public void addData(Scanner in) {
       while (in.hasNext()) {
          String word = in.next();
          < . . . add this occurrence of word to the number of occurrences . . . >
       }
    }
}
```

Version B.

```
// Computes the number of occurrences of all words in the input.
public class Concord {
    // . . . [other details of Concord left out]

    /**
      Add data from System.in to concordance (reads until end-of-file).
     */
    public void addData() {
       Scanner in = new Scanner(System.in);
       while (in.hasNext()) {
          String word = in.next();
          < . . . add this occurrence of word to the number of occurrences . . . >
       }
    }
}
```

Problem 5 [20 pts]

Write the *Java* function `printReversedSentences` which reverses the order of all the words in each sentence from a Scanner (it does not reverse the order of the sentences themselves), printing the reversed versions. For full credit you must use a Java `Stack` (see code handout for interface). A sentence has at least one word and is terminated with a period. To make the problem easier, we are not going to worry about proper capitalization. Here is an example of this function at work (pay attention to the placement of the periods):

input text

```
ahem. mary had a little lamb. its fleece was white as snow. the end.
```

corresponding output text

```
ahem. lamb little a had mary. snow as white was fleece its. end the.
```

```
//  prints out a version of the words from the scanner such that the order of
//  words is reversed in each sentence.  Processes all the words from the
//  Scanner.  All output will be on the one line (i.e., no newlines printed).
//  PRE: the scanner contains a sequence of zero or more sentences, each one
//     ending with a period.
public static void printReversedSentences(Scanner in) {
```

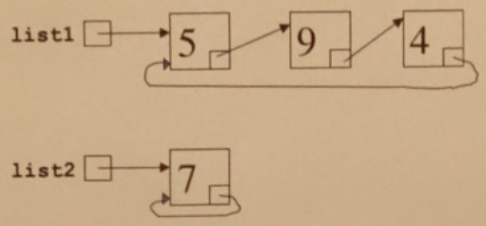
```
Stack<String> stack = new Stack<>();
while(in.hasNext()){
    String st = in.next();
    if(st.charAt(st.length()-1)!='.'){
        stack.push(st);
    }else{
        st = st.substring(0, st.length()-1);
        System.out.print(st);
        while(!stack.empty()){
            System.out.print(" "+stack.pop());
        }
        System.out.print(". ");
    }
}
```

V

USC NetID: _____

Problem 6 [15 points]

[C++] A circular list, used in some list applications, is a special linked list representation where the last node in the list points back to the **first** node. Here are some examples of a circular list:



Write the C++ function `isCircularList`, which tells whether its parameter is a well-formed linked list or a circular list. Hint: in C++ you can use the equality operator (==) on pointers (analogous to comparing object references in Java for equality).

The `Node` and `ListType` definitions are on the code handout.

```
// returns true iff list is a circular list
// PRE: list has at least one node and is either a well-formed linked list
//      or a circular list
bool isCircularList(Node * list) {

    Node* curr = list->next;

    while (curr != NULL){
        if (curr == list) return true;
    } curr = curr->next;
    return false;
}
```

Name: Chun-Kai Wang

USC NetID (e.g., `ttrojan`): _____

CS 455 Final Exam
Spring 2017 [Bono]

May 10, 2017

There are 7 problems on the exam, with 74 points total available. There are 12 pages to the exam (6 pages double-sided), including this one; make sure you have all of them. There is also a one-page code handout that accompanies the exam. If you need additional space to write any answers or for scratch work, pages 10, 11, and 12 of the exam are left blank for that purpose. If you use these pages for answers you just need to direct us to look there. ***Do not detach any pages from this exam.***

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and **circling** your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC username (a.k.a., NetID) at the top of the exam. Also, put your NetID at the top right of the front side of each page of the exam. Please read over the whole test before beginning. Good luck!

Problem 1 [10 pts]

TreeMap, HashMap

[Java] Briefly describe two possible internal representations one could use in an implementation of the Java `Map` interface, such that each of the representations is *different* than the two implementations that are provided in the Java library that we discussed this semester. For each one you describe, give the worst-case big-O time to do each of the `Map` operations listed below.

```
// get the value associated with this key
// if no entry with this key exists, returns null
public ValueType get(KeyType key)

// inserts an entry or updates the value that goes with an existing
// key.  Returns the old value associated with the key, or null
// if this key was not previously in the map
public ValueType put(KeyType key, ValueType value)

// remove the entry with this key
// returns the associated value or null if not found
public ValueType remove(KeyType key)
```

Description of representation 1:

using an ArrayList of objects which contains key and value. as instance variables

Big-O times for representation 1:

O(n) `get` O(n) `put` O(n) `remove`

Description of representation 2:

using a LinkedList of objects which contains key and value as instance variables

Big-O times for representation 2:

O(n) `get` O(n) `put` O(n) `remove`

USC NetID: _____

Problem 2 [8 pts.]

[Java] Implement the `boolean` Java method `sortByScore`, which sorts an `ArrayList` of `Student's` so they are in decreasing order by score. For students with the same score, they must be in increasing order by name. ***You must use the Java sort utility*** (more info about that on the code handout), and include any additional code necessary to make it work (outside of the `sortByScore` method).

Here is an example of contents of such an `ArrayList` shown in the order it would be in after a call to `sortByScore`:

Jan 98
Lin 98
Ann 84
Fred 84
Moe 84
Zhou 80
Aarti 72

Here is the `Student` class it uses (only shows interface, since that's all we are using here). This is an already completed class; ***you may not modify `Student` for this problem***.

```
public class Student {
   public Student(String name, int score) { . . . }
   public String getName() { . . . }
   public int getScore() { . . . }
   public void changeScore(int newScore) { . . . }

   . . .  // private stuff not shown
}
```

```
public static void sortByScore(ArrayList<Student> students) {
    Collections.sort(students, new CompName());
    Collections.sort(students, new CompScoreDec());
}

public class CompName implements Comparator<Student>{
    public int compare(Student a, Student b){
        return a.getName().compareTo(b.getName());
    }
}

public class CompScoreDec implements Comparator<Student>{
    public int compare(Student a, Student b){
        return b.getScore().compareTo(a.getScore());
    }
}
```

Problem 3 [19 pts. total]

Part A (15). [Java] Write the complete class definition for a *Java* class for a `Stack` of `int`s that uses an array representation (not `ArrayList`) such that all stack operations run in amortized constant time or better. Hint: Use the strategy such that if you run out of space in the array, you double its size. (The "amortized" part of the time is because of the need to increase the array size periodically.) You may not use any Java library classes or methods in your code, except the the `Arrays.copyOf` method, described on the code handout. You do not have to write method comments for your class.

The complete interface for the `Stack` class for this problem is shown by example below (Note: method names shown in bold):

```
Stack s = new Stack();            // creates an empty stack
s.push(3);           // pushes 3 onto the stack
s.push(0);           // pushes 0 onto the stack
int val1 = s.pop();    // pops top element from the stack and returns its value.
                       // only legal on non-empty stack
int val = s.top();     // returns the top element (stack is unchanged)
                       // only legal on non-empty stack
if (s.isEmpty()) {  // returns true if stack is empty, otherwise false
  System.out.println("stack is empty.");
}
```

Space for your answer is given on this and the next page.

```
public class Stack {
  private int[] arr; private int curr;
  public Stack(){ arr = new int[1]; curr = -1; }
  public void push(int a){
      if(curr == arr.length-1){ arr = Arrays.copyOf(arr, arr.length*2);}
      curr++;
      arr[curr] = a;
  }
  public int pop(){ curr--;
                  } return arr[curr+1];
  public int top(){ return arr[curr]; }
  public boolean isEmpty(){ return (curr > -1) ? true : false; }
}
```

USC NetID: _____

Problem 3 (cont.)

(additional space for your answer to Part A; Part B)

Part B (4). Write a representation invariant for your class implementation:

① (when the stack is not empty) arr[curr] is the top element of the stack

② $-1 \leq curr < arr.length$

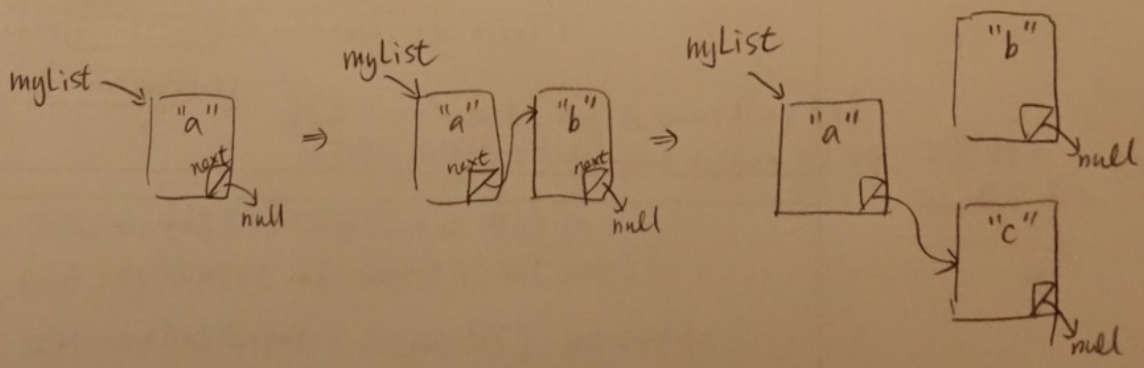
③ (when the stack is not empty) the stack elements are in positions [0, curr] of arr.

④ (when the stack is empty, curr == -1)

Problem 4 [9 points]

[C++] Consider the following buggy C++ function, `insertFront`, that attempts to insert a value in the front of a linked list, and the program that uses it below. (See code handout for documentation of `Node` and `ListType`):

```
// printList: prints list using the format: (cab sit dog dad)
// prints an empty list as: ()
// PRE: list is a well-formed list
void printList(ListType list) {  /* code not shown */  }

void insertFront(ListType & list, const string & val) {

   Node * newGuy = new Node(val);
   list->next = newGuy;

}

int main() {

   ListType myList = new Node("a");
   insertFront(myList, "b");
   insertFront(myList, "c");  // ***

   insertFront(myList, "d");

   printList(myList);

}
```



Part A. In the space above, draw a box-and-pointer diagram (a.k.a., memory diagram) showing all variables, objects, and their state as they have changed during the code sequence up *through* the statement labeled `***` (do not show the stuff after that). This includes showing `insertFront`'s parameters (you can name them $\texttt{list}_1$, $\texttt{val}_1$, in the first call and $\texttt{list}_2$, $\texttt{val}_2$, in the second call).

Part B. What is the output of the program?

(a d)

USC NetID: _____

Problem 5 [8 points]

[C++] The following is a simplified version of the `freq.cpp` program we did in lecture, but that is now organized as a multi-file program, so that we can compile the histogram functions (using `histogram.h` and `histogram.cpp`) separately from this program that uses them (`freq.cpp`). **In the space provided below show what needs to be in `histogram.h`** (in the blank box at the bottom of the page). **You may not change the contents of the other two files.** A full credit answer has what's needed and nothing more. Note: all places where functions from this program are called are shown in the code. Note: no "#fndef", etc., needed, because no classes are defined

```
#include "histogram.h"
const int MAX = 10;
const int NUM_COUNTS = MAX + 1;

int main() {
   int counts[NUM_COUNTS];
   initCounts(counts,  NUM_COUNTS);
   readAndCompute(counts,  NUM_COUNTS);
   printHist(counts,  NUM_COUNTS);
}
```

freq.cpp

```
#include "histogram.h"
#include <iostream>
using namespace std;
// Some functions definitions are elided for brevity
void initCounts(int counts[], int size) { . . . }
void readAndCompute(int counts[], int size) { . . . }
void printTicks(int numTicks) { . . . }
void printHist(int counts[], int size) {
   . . .
   printTicks(…);
   . . .
}
```

histogram.cpp

就算 printTicks 的 def 在 printHist 之後 也只要在此 file 上方 ~~forward~~ declare 就好、不需放入 header file.

在 header 中的只有那些要給 client 用的 fn.

```
Void initCounts ( int counts[], int size);
Void readAndCompute ( int counts[], int size);
void printHist ( int counts[], int size);
```

histogram.h

no need for
不需要 printTicks (int)

Problem 6 [5 pts]

[C++] Consider the following class and in particular the **add** method that attempts to insert a value into a hash table that uses chaining. Note: you may assume that the hash function used, `hashCode`, and the helper function, `insertFront` (described below), both work correctly:

```
// inserts elmt in the front of list
// pre: list is a well-formed list
// Note: assume this insertFront works correctly
void insertFront(ListType &list, const string & elmt);


class HashSetOfStrings {
  public:
    HashSetOfStrings();  // creates an empty set
    //  Add elmt to the set if it is not already present.
    //  If elmt is already present returns false and no change is made to the set.
    bool add(const string & elmt);
    // . . . [other methods of HashSetOfStrings not shown]
  private:
   const int HASH_SIZE = 9973;
   ListType hashArray[HASH_SIZE];          // fixed size array in this version
   // returns a value in the range 0 through HASH_SIZE-1
   int hashCode(const string & elmt) const;
   // . . . other details of class left out
};


bool HashSetOfStrings::add(const string & elmt) {
       int hashIndex = hashCode(elmt);
       if (hashArray[hashIndex] != NULL) { return false; }
       insertFront(hashArray[hashIndex], elmt);
       return true;
}
```

Give a concrete example that illustrates a scenario where the add code does not work. Your answer may consist of some combination a description of the scenario, code, and an explanation of what happens that is incorrect.

If two string str1 and str2, which str1 not equals to str2,
but hashCode(str1) == hashCode(str2), then, if we to add(str2) after
add(str1), the function would return false.
because hashArray[hashCode(str2)] != NULL (there's str1 already in this bucket)
however, because str2 doesn't equal to str1,
so function "add" shouldn't return false!

USC NetID: _____

Problem 7 [15 points]

[C++] Write the C++ function `gut`, which removes all of the middle elements from a linked list. That is, it keeps only the first and last elements of the list. For full credit, your function must reclaim any dynamic memory no longer used by the list.

The `Node` and `ListType` definitions are on the code handout. Some examples:

`list`	**`list` after call to `gut(list)`**
`(the big sloppy dog)`	`(the dog)`
`(the dog)`	`(the dog)`
`(the)`	`(the)`
`(the wet dog)`	`(the dog)`
`()`	`()`

```
// removes all but the first and last element from the list
// PRE: list a well-formed linked list
void gut(ListType & list) {
    if ( list == NULL || list.next == NULL || list.next.next = NULL) return ;

    ListType curr = list.next ;
    ListType prev = list ;
    while ( curr != NULL){
        delete prev ;
        prev = curr ;
        curr = curr.next ;
    }

    list.next = prev ;
    return ;
}
```