CSE100 Midterm 1
Winter 2017

This exam is closed book, closed notes. **Write all your answers on the answer sheet. However, all exam sheets, including scratch paper must be turned in at the end of the exam.**

By signing your name below, you are asserting that all work on this exam is yours alone, and that you will not provide any information to anyone else taking the exam. In addition, you are agreeing that you will not discuss any part of this exam with anyone who is not currently taking the exam in this room until after the exam has been returned to you. This includes posting any information about this exam on Piazza or any other social media. Discussing any aspect of this exam with anyone outside of this room constitutes a violation of the academic integrity agreement for CSE 100.

Please write the following statement in the box below and then sign your name on the line below:

"I excel with integrity"

Signature: _____

Name (please print clearly):_____

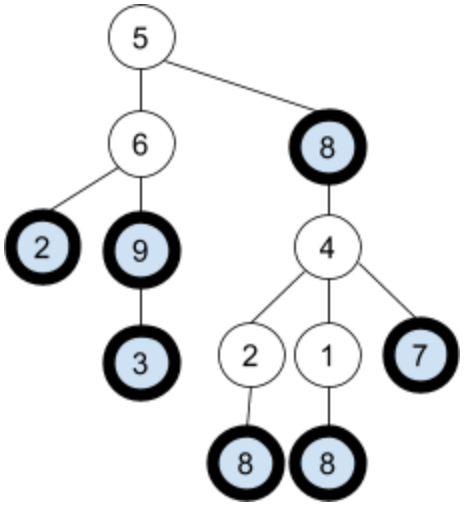
PID:_____

You have 50 minutes to complete this exam. Work to maximize points. If you don't know the answer to a problem, move on and come back later. Most importantly, stay calm and don't panic. You can do this.

DO NOT OPEN THIS EXAM UNTIL YOU ARE INSTRUCTED TO DO SO.
GOOD LUCK!

Problem 1: Working with TSTs and Hash tables [6 points]

1.1 Consider the following TST where nodes that end words are shaded and have a thick border.



1.1.i. (2 pts) List all of the keys in this TST. 52, 569, 5693, 8, 87, 828, 8418
1.1.ii. (2 pts) Which key or keys could have been inserted first in this TST? 5693, 569

1.2 (2 pts) Consider a hash table of size 100 that uses separate chaining for collision resolution. The hash table currently stores 15 elements in 13 different slots (there have been 2 collisions already). Assuming a uniform hash function, what is the probability that there will be at least 1 collision in the next 3 insertions? You should leave your answer as an arithmetic expression.

1- (87/100)*(86/100)*(85/100)

Problem 2: C++ Concepts and BSTs [8 points]

2.1. Recall the BST class from PA1. The class declaration is below:

```
template<typename Data>
class BST {
public:
  typedef BSTIterator<Data> iterator;

  BST();  // Constructor
  ~BST();  // Destructor
  std::pair<iterator, bool> insert(const Data& item);
  iterator find(const Data& item)  const;
  unsigned int size() const;
  int height() const;
  bool empty() const;
  iterator begin() const;
  iterator end() const;

private:
  BSTNode<Data>* root;
  unsigned int isize;
  // Private methods not shown
};
```

const function means it can't modify objects on which they were called (member variables).

2.1.i. (1 pt) Consider the signature of the find function. Explain what the `const` keyword at the end of the function header means (shown in bold), and write a line of code that would cause an error if it appeared in the body of the find function *because of* this `const` declaration (i.e. would not cause an error if this `const` declaration were not there). any code modifying isize or root, eg. root = root -> left

2.1.ii. (1 pt) In the code below, assume that the `BST` and `BSTIterator` classes are correctly implemented as in PA1, and that `myBST` holds an object of type `BST<int>` that has already been constructed.

```
BSTIterator<int> it = myBST.begin();
if (*it > 0) {
    std::cout << "BST contains only positive integers";
    std::cout << std::endl;
}
```

Will the code above ever cause a segmentation fault? If not, explain why not. If so, explain when and why it would cause a segmentation fault.

Yes. When BST is empty curr will be null/dereferencing will cause a segfault.

BST<double> tree; or BST<double> tree = BST<double>();
BST<double>* newTree = &tree;

2.1.iii. (1 pt) Write a line of code that creates a variable of type `BST<double>` using *automatic* (not dynamic) memory allocation. Then, write a line of code that creates a variable of type `pointer to BST<double>` and sets it to point to the `BST<double>` object created in the first line you wrote.

2.1.iv. (2 pts) Finish the code shown below to calculate and return the height method for the BST class declared above. You may ***not*** assume you have a variable that already stores the height. As in PA1, an empty tree has height -1, and a tree with one node has height 0.

You are adding the necessary code to the `help` function, which takes one argument named `n` (you must determine its type). Note that you might find the `std::max` function helpful, which returns the maximum of two arguments. E.g. `std::max(3, 7)` returns 7.

```
template <typename Data>
int BST<Data>::height() const
{
    return help(root);
}

template <typename Data>

_____[BLANK A (help FUNCTION HEADER)]_____
{
    if (n == nullptr) {

        return ___[BLANK B]_____;
    }
    else {

        return _______[BLANK C]_____________;
    }

}
```

int BST<Data>::help(BSTNode<Data> *n) const

-1

1 + std::max(help(n->left), help(n->right))

Consider the code below:

```
int myfunc(int & param)
{
    param = param / 2;
    return param;
}

int main()
{
    int first = 10;
    int second = 20;
    int & third = second;
    int *ptr = &first;
    second = myfunc(first);
    *ptr = 30;
 }
```

2.2 (1 pts) In the box on your answer sheet, draw the **final** state of the memory diagram for the variables in main after the following lines of code are executed, as we have done in class. You do not need to show changes as the code executes, but you will not be penalized if you do, as long as the final state of memory is clear in your diagram. You do not need to include the variable(s) in `myfunc`, but you will not be penalized if you do.

first (30),
second and third same box (5),
ptr (points to first box)

2.3. (2 pts) Explain briefly how to ***fix*** **the memory leak** in the insert implementation below. Assume that the destructor is implemented correctly. Note that there is no bug in `insert` other than the memory leak. Numbers on the left give line numbers.

```
1 bool BSTInt::insert(int item)
2 {
3   BSTNodeInt* newNode = new BSTNodeInt(item);
4   if (!root) {
5     root = newNode;
6     ++isize;
7     return true;
8   }
9
10  BSTNodeInt* curr = root;
11  while (curr != nullptr) {
12    if (item < curr->data) {
13        if (curr->left == nullptr) {
14            curr->left = newNode;
15            newNode->parent = curr;
16            return true;
17        }
18        else {
19            curr = curr->left;
20        }
21    }
22    else if (curr->data < item) {
23        if (curr->right == nullptr) {
24            curr->right = newNode;
25            newNode->parent = curr;
26            return true;
27        }
28        else {
29            curr = curr->right;
30        }
31    else {
32        // found the value.  Nothing to insert.
33        return false;
34    }
35 }
```

Call delete on newNode before line 33

OR

remove line 3 and create the new node just before lines 5, 14 and 24

Problem 3: Data structure running times and properties [6 Points]

3.1 (2 pts) What is the *average (or expected) case* running time for a successful find for each of the following data structures? Assume that you are storing N keys that all have length D. Express each as the tightest Big-O bound possible. (Each incorrect answer is worth -1, so you must get at least 2 correct to get any points on this problem).

3.1.i. Unsorted Linked List O(N)

3.1.ii. Multi-way trie O(D)

3.1.iii. Binary search tree (not necessarily balanced) O(logN)

A **3.2** (1 pt) Which of the following data structures are considered *space inefficient* in that it often allocates a lot of space it does not use. Select all that apply.

A. Multi-way trie
B. Ternary Search Tree
C. Binary Search Tree
D. Linked List

3.3 In our proof of the average case running time to successfully find an element in a linked list, we ended up with the following formula:

$$\frac{1+2+3+...+(n-2)+(n-1)+n}{n} = O(n)$$

sum of # of comparisons: the 1st item needs 1, the 2nd needs 2, etc.

3.3.i. (2 pts) Explain in 1-2 sentences what the terms (1, 2, 3, …, (n-2), (n-1), n) in the numerator mean.

3.3.ii. (1 pt) Explain in 1-2 sentences how the assumption "All keys are equally likely to be searched for" shows up in this formula.

By dividing the sum of # of comparisons by n.
(without the assumption, you would have to weigh # of comparisons for different keys differently, instead of just dividing by n)

WRITE YOUR ANSWERS ON THE ANSWER SHEET. This exam is closed book, closed notes.

Scratch paper

WRITE YOUR ANSWERS ON THE ANSWER SHEET. This exam is closed book, closed notes.