Data Structures (COM 2000)
Assignment 2

Available Date: Tuesday, March 01, 2016
Due Date: 11.50 PM, Thursday, March 17, 2016
Total Mark: 80 marks
Assessment
Coursework: 40%
  Assignments (20%): A1 (7%), A2 (7%), A3 (6%)
  Coursework exams (20%): CWE1 (10%), CWE2 (10%)
Final Examination: 60% (one two-hour writing exam)

Assignment Requirements

I. Part 1 [30 marks] Binary search tree

A node of a binary search tree (BST) of integers is defined as

```
typedef struct {
   int    data;
   struct Node* left;
   struct Node* right;
} Node;
```

A binary search tree **`root`** is declared by

```
treePtr root = NULL;
```

where `treePtr` is defined as below.

```
typedef Node* treePtr;
```

Write a complete program **`BST.c`** to demonstrate basic operations on a binary search tree of integers. Your program should show the following options.

```
1. Insert a new node (iteration)
2. Insert a new node (recursion)
3. Tree traversal
4. Search a node (iteration)
5. Search a node (recursion)
6. Count number of nodes in tree
7. Count number of leaves in tree
8. Height of tree (root level = 0)
9. Height of tree (root level = 1)
10. Find the node with minimum key (iteration)
11. Find the node with minimum key (recursion)
12. Find the node with maximum key (iteration)
```

```
13. Find the node with maximum key (recursion)
14. Delete a node from BST (iteration)
15. Delete a node from BST (recursion)
16. Find the inorder successor (without using parent link)
17. Breadth-first traversal (BFT)
18. Exit
Select your choice (1-18):
```

The program allows a user to choose one of the above options by entering an integer from 1 to 18 to perform the corresponding action.
The marks are given as follows.

1. [1 mark] Insert a new node (iteration). The **`insert()`** function allows the user to iteratively insert a new element into the BST.
2. [1 mark] Insert a new node (recursion). The **`insertR()`** function allows the user to recursively add a new element into the BST.
3. [1 mark] Tree traversal. This option allows the user to select the inorder, preorder, or postorder traversal by entering an integer 1, 2, or 3, respectively, to perform the corresponding action.
4. [2 marks] Search a node (iteration). The **`search()`** function iteratively searches on the BST for the specified search key **`x`**, where **`x`** is input by the user. The **`search()`** function returns the node containing **`x`** if it is found; otherwise **`NULL`** is returned.
5. [2 marks] Search a node (recursion). The **`searchR()`** function recursively searches on the BST for the specified search key **`x`**, where **`x`** is entered by the user. The **`searchR()`** function returns the node containing **`x`** if it is found; otherwise **`NULL`** is returned.
6. [1 mark] Count number of nodes in tree. The **`countNodes()`** function returns the number of nodes in the BST.
7. [1 mark] Count number of leaves in tree. The **`countLeaves()`** function returns the number of leaf nodes in the BST.
8. [1 mark] Height of tree (root level = 0). The **`compHeight()`** function returns the height of the BST, where the height of a tree is the length of the longest simple path from the root to a leaf and the root is counted as level 0.
9. [1 mark] Height of tree (root level = 1). The **`numLevels()`** function returns the height of the BST, where the height of a tree is the number of levels in the tree and the root is counted as level 1.
10. [1 mark] Find the node with minimum key (iteration). The **`findMin()`** function iteratively searches on the BST for the node with the smallest value **`m`** of the data member. The **`findMin()`** function returns the node containing **`m`**.
11. [1 mark] Find the node with minimum key (recursion). The **`findMinR()`** function recursively searches on the BST for the node with the smallest value **`m`** of the data member. The **`findMinR()`** function returns the node containing **`m`**.
12. [1 mark] Find the node with maximum key (iteration). The **`findMax()`** function iteratively searches on the BST for the node with the largest value **`M`** of the data member. The **`findMax()`** function returns the node containing **`M`**.

13. [1 mark] Find the node with maximum key (recursion). The **`findMaxR()`** function recursively searches on the BST for the node with the largest value **`M`** of the data member. The **`findMaxR()`** function returns the node containing **`M`**.
14. [5 marks] Delete a node from BST (iteration). The **`Delete()`** function iteratively deletes an arbitrary node of the BST. Specifically, the program asks the user to enter the data member **`x`** of the node to be deleted. The **`Delete()`** function performs the deletion operation if **`x`** exists in the tree and returns 1; otherwise 0 is returned. The to-be-deleted node **`x`** is replaced with the leftmost node of the right subtree of **`x`**.
15. [5 marks] Delete a node from BST (recursion). The **`DeleteR()`** function recursively deletes an arbitrary node of the BST. Specifically, the program asks the user to enter the data member **`x`** of the node to be deleted. The **`DeleteRR()`** function performs the deletion operation if **`x`** exists in the tree and returns the new tree (i.e., the tree with **`x`** has been deleted); otherwise displays the message "`Not found`". The to-be-deleted node **`x`** is replaced with the leftmost node of the right subtree of **`x`**.
16. [2 marks] Find the inorder successor (without using parent link). The **`inOrderSuccessor()`** function allows the user to find the inorder successor of a given node in the BST. The inorder successor of a given node is the node comes after the node in the inorder traversal of the BST. The rightmost node of the BST has no inorder successor.
17. [3 marks] Breadth-first traversal (BFT). The **`BFT()`** function traverses the tree level by level, staring at the root. At each level, the nodes are traversed from left to right.
18. Quit your program.

II. Part 2 [50 marks] Hashing methods

Write the following complete C (or C++) programs.

1. **`LP.c`** to implement the linear probing method. [10 marks]
2. **`QP.c`** to implement the quadratic probing method. [10 marks]
3. **`DH.c`** to implement the double hashing method. [10 marks]
4. **`CC.c`** to implement the coalesced chaining method. [10 marks]
5. **`SC.c`** to implement the separate/direct chaining method. [10 marks]

• Each of the above programs should displays the following menu when it is executed.

```
1. Insert a new key
2. Search a given key
3. Delete a given key
4. Display hash table
5. Quit
Select your option (1-5):
```

• Each of the above programs has the **`Insert()`**, **`Search()`**, **`Delete()`**, and **`Display()`** functions to perform the insertion, searching, deletion, and displaying operations, respectively. The **`Insert()`** function allows a user to insert a new key **`k`** into the hash table. It is supposed that the keys are distinct nonnegative integers. The **`Search()`** function allows a user to search the hash table for a given key **k**. The **Delete()** function allows a user to delete a given key **k** from the hash table. The **Display()** function shows the hash table contents on the screen.

• For the **LP.c** program, the size of the hash table is $M = 10$. The hash function is defined as $f(k) = k \% M$, where the symbol % is the modulo operator and the key k is a nonnegative integer. The rehash function is $f_t(k) = (f(k) + t) \% M$, where the collision count $t = 1, 2, ...$ The **Search()** function returns the index i of **k**, $0 \leq i \leq M - 1$, if **k** is found; otherwise, returns M. Each node of the hash table is defined by **typedef struct { int k; } Node;**.

• For the **QP.c** program, the size of the hash table is $M = 10$. The hash function is defined as $f(k) = k \% M$, where the symbol % is the modulo operator and the key k is a nonnegative integer. The rehash function is $f_t(k) = (f(k) + t^2) \% M$, where the collision count $t = 1, 2, ...$ The **Search()** function returns the index i of **k**, $0 \leq i \leq M - 1$, if **k** is found; otherwise, returns M. Each node of the hash table is defined by **typedef struct { int k; } Node;**.

• For the **DH.c** program, the size of the hash table is $M = 11$. The hash function is defined as $f(k) = k \% M$, where the symbol % is the modulo operator and the key k is a nonnegative integer. The rehash function is $f_t(k) = (f_{t-1}(k) + g(k)) \% M$, where the collision count $t = 1, 2, ...$, the second hash function is $g(k) = c - (k \% c)$, the constant $c = 5$, $f_0(k) = f(k)$. The **Search()** function returns the index i of **k**, $0 \leq i \leq M - 1$, if **k** is found; otherwise, returns M. Each node of the hash table is defined by **typedef struct { int k; } Node;**.

• For the **CC.c** program, the size of the hash table is $M = 10$. The **Search()** function returns the index i of **k**, $0 \leq i \leq M - 1$, if **k** is found; otherwise, returns M. The **Delete()** function works as the following illustration. Each node of the hash table is defined by **typedef struct { int k; int next; } Node;**.

Suppose that the current state of the hash table is as follows.

Index	*k*	*next*
0	10	9
1	-1	-1
2	-1	-1
3	-1	-1
4	-1	-1
5	15	8
6	26	-1
7	35	-1
8	25	7
9	30	-1

Initial state

Index	*k*	*next*
0	10	**-1**
1	-1	-1
2	-1	-1
3	-1	-1
4	-1	-1
5	15	8
6	26	-1
7	35	-1
8	25	7
9	**-1**	-1

If 30 is deleted.

Index	*k*	*next*
0	**30**	**-1**
1	-1	-1
2	-1	-1
3	-1	-1
4	-1	-1
5	15	8
6	26	-1
7	35	-1
8	25	7
9	**-1**	-1

If 10 is deleted.

Index	*k*	*next*
0	10	9
1	-1	-1
2	-1	-1
3	-1	-1
4	-1	-1
5	15	8
6	26	-1
7	**-1**	-1
8	25	**-1**
9	30	-1

If 35 is deleted.

Index	*k*	*next*
0	10	9
1	-1	-1
2	-1	-1
3	-1	-1
4	-1	-1
5	15	8
6	26	-1
7	**-1**	-1
8	**35**	**-1**
9	30	-1

If 25 is deleted.

Index	*k*	*next*
0	10	9
1	-1	-1
2	-1	-1
3	-1	-1
4	-1	-1
5	**25**	8
6	26	-1
7	**-1**	-1
8	**35**	**-1**
9	30	-1

If 15 is deleted.

• For the **`SC.c`** program, the size of the hash table is M = 10. The **`Search()`** function returns the pointer to the node containing **`k`** if **`k`** is found; otherwise, returns **`NULL`**. Each node of the hash table is defined by **`typedef struct { int k; struct Node *next; } Node;`**.

Submission: **carefully** submit your source program files (i.e., **`*.c`**) to Mr. Sterling Ramroach via the email: sramroach@gmail.com.

• At the top of your program, you should include the following information.

```
/* Student Full Name:
   Student ID:
   E-mail:
   Course Code:
*/
```

End of Assignment 2