CSCI 2100C Data Structures

Quiz 2
22 March 2022

Note:

- You must work on the paper sent to your own email address.
- The quiz lasts for 45 minutes, until 13:15.
- You should not communicate with other people (including other candidates) after the quiz starts, until 14:00 today.

IMPORTANT—How to submit your answers:

- You shall submit your answer to the Blackboard system by **13:35** today (that is, you will have 20 minutes after the quiz to prepare the submission).
- You should write down your student ID on the first page of the answers.
- You should include all your answers in one single file. The file should be in PDF or JPG format.

Answer ALL Questions.

1. Keith defines the Binary Search tree ADT **BSTreeADT** and the tree node ADT **TreeNodeADT**. In Keith's implementation, the key of a node is an integer, while the data field contains a void pointer that points to the actual data. The header file BSTreeADT.h Keith provides is shown below:

```
/* File: BSTreeADT.h */
#include <stdio.h>
#include <stdbool.h>

typedef struct BSTreeCDT *BSTreeADT;
typedef struct TreeNodeCDT *TreeNodeADT;

BSTreeADT EmptyBSTree(void);
BSTreeADT NonemptyBSTree(TreeNodeADT, BSTreeADT, BSTreeADT);
BSTreeADT LeftBSSubtree(BSTreeADT);
BSTreeADT RightBSSubtree(BSTreeADT);
bool BSTreeIsEmpty(BSTreeADT);

TreeNodeADT BSTRoot(BSTreeADT);
const TreeNodeADT SpecialErrNode = (TreeNodeADT) NULL;
TreeNodeADT NewTreeNode(int, void*);
    /* First argument is the node key. */
    /* Second argument points to the node data. */
int GetNodeKey(TreeNodeADT);
void* GetNodeData(TreeNodeADT);
```

Keith also correctly implements the Binary Search tree ADT **BSTreeADT** and the tree node ADT **TreeNodeADT** in file BSTreeADT.c, which is not shown here.

a) **(8 marks)** Write a function

int TreeHeight(BSTreeADT);

which returns the height of the argument Binary Search tree. Note that you can write *either* a recursive function *or* a non-recursive function.

Consider the following a function MyFunc that Isabella writes:

```
int MyFunc(BSTreeADT T) {
  if (BSTreeIsEmpty(T)) return 0;
  int L = MyFunc(LeftBSSubtree(T));
  int R = MyFunc(RightBSSubtree(T));
  return L > R ? L : (R - L);
}
```

b) **(4 marks)** Let n be the number of nodes and h be the height of the tree T. How many times will MyFunc be called when T is passed to MyFunc as the argument? Express your answer in terms of n and/ or h.

c) **(4 marks)** Assume that the tree T is balanced. Express n in big-O notation in terms of h.

d) **(4 marks)** Assume that the tree T is balanced. Express the computational complexity of MyFunc in Big-O notation in terms of h.

2. An ADT of list of integers is defined in the file **list.h** shown below.

```
#include <stdbool.h>

typedef struct listCDT *listADT;

typedef int listElementT;

listADT EmptyList(void);
listADT Cons(listElementT, listADT);
listElementT Head(listADT);
listADT Tail(listADT);
bool ListIsEmpty(listADT);
```

The function Fn1 accepts a listADT argument and returns a bool value false if any element at an even position, i.e., element 2, element 4, …, and so on, is an even integer or 0. It returns a bool value true otherwise. For examples,

- Fn1([]) = true
- Fn1([1]) = true
- Fn1([9,5]) = true
- Fn1([3,4,1,8,2]) = false
- Fn1([8,8,8,7,7]) = false

The correct implementation of Fn1 is shown below, with some parts missing:

```
#include "list.h"

bool Fn1(listADT X) {
   if ( Missing Part 1 ) return true;
   if (ListIsEmpty(Tail(X))) return  Missing Part 2 ;
   if ( Missing Part 3  % 2  Missing Part 4  0) return false;
   return Fn1( Missing Part 5 );
}
```

a) **(2 marks)** What should be written at $\boxed{\textbf{Missing Part 1}}$?

b) **(2 marks)** What should be written at $\boxed{\textbf{Missing Part 2}}$?

c) **(2 marks)** What should be written at $\boxed{\textbf{Missing Part 3}}$?

d) **(2 marks)** What should be written at $\boxed{\textbf{Missing Part 4}}$?

e) **(2 marks)** What should be written at $\boxed{\textbf{Missing Part 5}}$?

Let L be a list of length m.

f) **(2 marks)** In the best case, how many elements have to be considered to execute Fn1(L)? Express your answer in terms of m.

g) **(2 marks)** In the worst case, how many elements have to be considered to execute Fn1(L)? Express your answer in terms of m.

h) **(2 marks)** In the best case, what is the computational complexity of Fn1(L)?

i) **(2 marks)** In the worst case, what is the computational complexity of Fn1(L)?

j) **(2 marks)** On average, what is the computational complexity of Fn1(L)?

3. Consider he ADT of list of integers defined in the file **list.h** shown in Question 2. Kenneth has got an innovative idea of using queues to implement the listADT. The queueADT header file queue.h Kenneth uses is shown below for your reference. He has also correctly implemented the queue ADT.

```
/* File: queue.h */
#include <stdbool.h>

typedef struct queueCDT *queueADT;
typedef int queueElementT;

queueADT EmptyQueue(void);
void Enqueue(queueADT, queueElementT);
queueElementT Dequeue(queueADT);
bool QueueIsEmpty(queueADT);
```

Based on the file **list.h** shown in Question 2, Kenneth completes the implementation list.c shown below.

```
/* File: list.c */
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "list.h"
#include "queue.h"

struct listCDT { queueADT Q; };

listADT EmptyList() {
   listADT list = (listADT) malloc(sizeof(*list));
   list->Q = EmptyQueue();
   return list;
}

listADT Cons(listElementT x, listADT L) {
   listADT list = (listADT) malloc(sizeof(*list));
   list->Q = EmptyQueue();
   Enqueue(list->Q , x);
   { queueADT t = EmptyQueue();
      while (!QueueIsEmpty(L->Q)) Enqueue(t, Dequeue(L->Q));
      while (!QueueIsEmpty(t)) {
         int tmp = Dequeue(t);
         Enqueue(L->Q, tmp);
         Enqueue(list->Q, tmp);
      }
   }
   return list;
}

listElementT Head(listADT L) {
   if (ListIsEmpty(L)) exit(EXIT_FAILURE);
   queueADT tmp = EmptyQueue();
   int e = Dequeue(L->Q);
   while (!QueueIsEmpty(L->Q)) Enqueue(tmp, Dequeue(L->Q));
   Enqueue(L->Q, e);
   while (!QueueIsEmpty(tmp)) Enqueue(L->Q, Dequeue(tmp));
   return (listElementT) e;
}

listADT Tail(listADT L) { … } // not shown

bool ListIsEmpty(listADT L) { … } // not shown
```

a) **(5 marks)** Write the ListIsEmpty function.

Consider the Tail function shown below, with some parts missing.

```
listADT Tail(listADT L) {
   if (ListIsEmpty(L)) exit(EXIT_FAILURE);
   listADT v = EmptyList();
   int h = Missing Part P ;
   { queueADT t = EmptyQueue();
      while (!QueueIsEmpty( Missing Part Q ))
         Enqueue(t, Missing Part R );
      Enqueue(L->Q, h);
      while (!QueueIsEmpty(t)) {
         int tmp = Dequeue(t);
         Enqueue(L->Q, Missing Part S );
         Enqueue( Missing Part T , tmp);
      }
   }
   return v;
}
```

b) **(3 marks)** What should be written at **Missing Part P**?

c) **(3 marks)** What should be written at **Missing Part Q**?

d) **(3 marks)** What should be written at **Missing Part R**?

e) **(3 marks)** What should be written at **Missing Part S**?

f) **(3 marks)** What should be written at **Missing Part T**?

4. The ADT of list of integers is defined in the file **list.h** shown in Question 2.

   The function isExtension accepts two listADT arguments, and returns 1 if the second list is the same as, or is an extension of, the first list, or it returns 0 otherwise. For examples,

   - isExtension([], []) = 1
   - isExtension([], [8,5,7,8,9]) = 1
   - isExtension([5], []) = 0
   - isExtension([7,6], [2,3]) = 0
   - isExtension([2,5], [2,5,8,1,4]) = 1
   - isExtension([3,8,4], [3,8,4]) = 1
   - isExtension([7,9,3], [7,9]) = 0

The following shows a nonrecursive version of the function.

```
int isExtension(listADT L1, listADT L2) {
    if ( Missing Part A ) return 1;
    if (ListIsEmpty(L2)) return 0;
    listADT X1 = L1;
    listADT X2 = L2;
    while (!ListIsEmpty(X2)) {
        if ( Missing Part B ) return 1;
        if (Head(X1) != Head(X2))  Missing Part C ;
        X1 =  Missing Part D ;
        X2 = Tail(X2);
    }
    if (ListIsEmpty(X1)) return 1; else return 0;
}
```

a) **(1 mark)** What should be written at **Missing Part A**?

b) **(1 mark)** What should be written at **Missing Part B**?

c) **(1 mark)** What should be written at **Missing Part C**?

d) **(1 mark)** What should be written at **Missing Part D**?

e) **(10 marks)** Write a recursive version of function isExtension.

Consider the following function:

```
listADT MythFn(listADT L1, listADT L2) {
   if (ListIsEmpty(L1) || ListIsEmpty(L2)) return EmptyList();
   if (Head(L1) >= Head(L2)) return Cons(-1, MythFn(Tail(L1), Tail(L2)));
   if (Head(L1) < Head(L2)) return Cons(1, MythFn(Tail(L1), Tail(L2)));
   return Cons(0, MythFn(Tail(L1), Tail(L2)));
}
```

f) **(1 mark)** What does MythFn return if L1 is [] and L2 is []?

g) **(1 mark)** What does MythFn return if L1 is [3,4,8] and L2 is []?

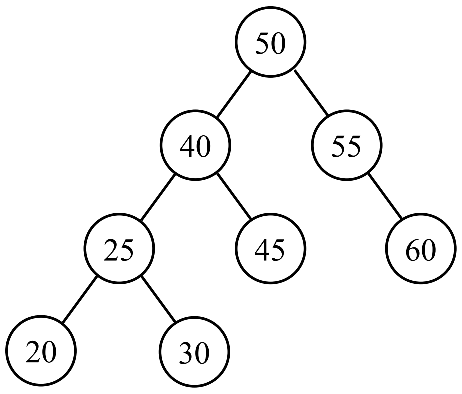
h) **(2 marks)** What does MythFn return if L1 is [5,9] and L2 is [6]?

i) **(2 marks)** What does MythFn return if L1 is [2,2,3,7] and L2 is [2,3,4,6]?

5. **(10 marks)** Using a sequence of diagrams, show clearly how **selection sort** is used to sort the following sequence into descending order:

5, 1, 2, 8, 4, 7, 3, 6, 3, 9

6. **(10 marks)** Consider the following AVL tree t1, in which the numbers shown are the keys of the respective nodes.



Nodes with the following keys are then inserted, in the following sequence, into t1:

1, 5, 70, 75, 10, 22, 15, 80, 31, 35

Using a sequence of diagrams, show clearly the process of node insertion. One diagram should be provided to show what t1 looks like after each of these 10 nodes is inserted and rotations are performed (if rotation is necessary).

— End of Quiz—