CSCI2100C Data Structures

Assignment 3
Due Date: 12 April 2022

Written Exercises

1. Suppose t1 is initially an empty AVL tree. Nodes with the following keys are then inserted, in the following sequence, into t1:
   4, 10, 11, 8, 5, 6, 3, 9, 12, 13, 7, 2, 1
   Show clearly what t1 should look like after each node is inserted. Remember to perform rotations whenever necessary.

2. Repeat the above question, assuming that the tree is only an ordinary binary search tree (not an AVL tree). You do not need to show what the tree should look like after each node is inserted. Only the final tree needs to be shown.

3. What is the computational complexity of the following function?

```
int Mystery1(int n) {
    int i, j, sum;
    sum = 0;
    for (i=0; i<n; i++) for (j=0; j<i; j++) sum += i*j;
    return (sum);
}
```

4. Consider the situation when Quicksort is used to sort an array of n elements into ascending order, but the elements are originally arranged in descending order. Express the complexity of the Quicksort algorithm in this case in big-O notation.

Programming Exercises

5. In C, you write functions that have variable number of arguments if you #include <stdarg.h>. For example, one can write a function test1 that prints an arbitrary number of integers, as follows:

```
#include <stdio.h>
#include <stdarg.h>

void test1(int number, ...) {
    va_list args;
    int x;

    va_start(args, number);
    for (int k=0; k<number; k++) {
            x = va_arg(args, int);
            printf("%d ", x);
    }

    va_end(args); // va_end is used, *instead of* return.
}
```

   then you can write the following function calls:

```
    test1(1, 3); /* this prints 3 */
    test1(3, 7, 8, 9); /* this prints 7 8 9 */
    test1(2, 4, 5); /* this prints 4 5 */
```

and the integer actual arguments will all be printed.

To use this feature, you must include <stdarg.h>, and declare a variable of type va_list in the function. Note that you can have a number of named arguments in the function definition, followed by a number of unnamed arguments (collectively represented by three dots ('...')). Before you access the arguments, you must first call va_start(a, x), where the first argument a is a variable of type va_list, and x is the identifier of the last named argument (the one just before ', ...'). After that, you can access the arguments one after another by calling va_arg(a, t), where a is the va_list variable and t is the name of the type of the arguments. Finally, you must call va_end(a) to implement the clean-up process and the return operation, where a is the va_list variable. If you call return before va_end, no one knows what will happen.

In the above example we only have one named argument (i.e., number) before ', ...'. In practice one can have more named arguments ', ...', as long as ', ...' comes last.

We can make use of this feature to define a general tree, in which each tree node can have arbitrary number of children. The Tree ADT header file is shown below:

```
/* File: Tree.h */
#include <stdarg.h>
#include <stdlib.h>
#include <stdbool.h>
#include "TreeNode.h"

typedef struct TreeCDT *TreeADT;

TreeADT EmptyTree(void);
TreeADT NonemptyTree(TreeNodeADT, int, ...);
   // The first argument is the root;
   // The second argument is the number of children of the root;
   // The other arguments are subtrees of type TreeADT.
TreeADT Subtree(int, TreeADT);
   // Returns the nth subtree where n is given in the first argument.  n = 1, 2, ...
int NumberOfSubtrees(TreeADT); // number of children of the root
bool TreeIsEmpty(TreeADT);
TreeNodeADT Root(TreeADT);
```

The file TreeNode.h is shown below:

```
/* File: TreeNode.h */
#include <stdlib.h>

typedef struct TreeNodeCDT *TreeNodeADT;
typedef char NodeContentT; // For simplicity, we only store one character in a node.

TreeNodeADT NewNode(NodeContentT);
NodeContentT GetNodeContent(TreeNodeADT);
```

Therefore, the value of t1 will be the same in the following two program segments.

```
TreeADT t1, t2, t3, t4, t5;
TreeNodeADT R;

...
...

t1 = NonemptyTree (R, 3, t2, t3, t4); // 3 subtrees
```

After that, the call

t5 = Subtree(2, t1); // Subtree 2 of t1

assigns t3 to t5.

Part of the file Tree.c is shown below.

```
/* File: Tree.c */
#include <stdarg.h>
#include <stdlib.h>
#include <stdbool.h>
#include "Tree.h"
#include "TreeNode.h"

struct TreeCDT {
  TreeNodeADT rt;
  int nrSubtrees;
  TreeADT subtree[100];
  // The maximum number of subtrees is 99.
  // If nrSubtrees = k, then only subtree[1] to subtree[k] are used.
  // subtree[0] is never used.
};
```

The complete TreeNode.c file is shown below.

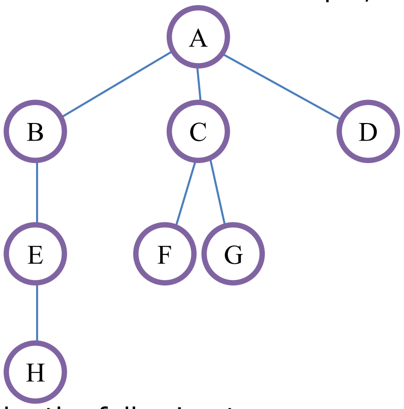
```
/* File: TreeNode.c */
#include <stdlib.h>
#include "TreeNode.h"

struct TreeNodeCDT { NodeContentT NodeContent; };

TreeNodeADT NewNode(NodeContentT C) {
  TreeNodeADT N = (TreeNodeADT) malloc(sizeof(*N));
  N->NodeContent = C;
  return N;
}

NodeContentT GetNodeContent(TreeNodeADT N) {
  return N->NodeContent;
}
```

__**First task:**__

Complete the Tree.c file.

Second task:

We first note that a tree can be two arrays: the first represents the structure of the tree, the second stores the node contents. For example, consider the following tree:



This tree can be represented by the following two arrays.

Array 1 (integers to indicate the parent; the first element is unused)

🟧	0	0	0	1	2	2	4

Array 2: (node contents, i.e., characters in this question)

A	B	C	D	E	F	G	H

Write the following two functions in the application file. That is, you cannot write these functions in Tree.c or TreeNode.c.

(1) A function BFS, which takes a TreeADT argument, and returns a character array of the node contents of the tree, which corresponds the Array 2 in the previous task.

```
char *BFS(TreeADT);
```

Hint: for each tree node, enqueue all its subtrees in a queueADT of tree. Then dequeue a tree from the queue, and enqueue all its subtrees to the same queue. Repeat these steps.

(2) A function Tree2Array, which takes a TreeADT argument, and returns an integer array that represents the tree structures, which corresponds to the Array 1 in the previous task.

```
int *Tree2Array(TreeADT);
```

— End of Assignment —