Hacettepe University
Department of Computer Engineering

BBM 201 – Data Structures
Resit Exam for Fall 2019 Semester
September 11, 2020

Problem	Points	Grade
1	15	
2	15	
3	25	
4	20	
5	25	
Total	100	

HONOR STATEMENT

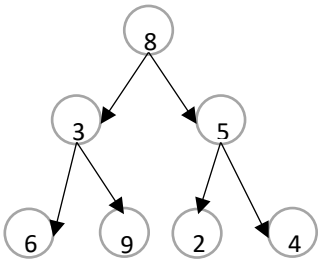
I pledge on my honor that I will not give nor receive any unauthorized help on this exam, and that all submitted work I will upload will be my own.

INSTRUCTIONS

- You have exactly **120 minutes** to finish and submit your answers to the exam.
- You must solve the questions on physical sheets of paper legibly (easy to read) with your own **handwriting**. You will upload your answers to the exam submission form by scanning or taking a picture of each answer sheet. Your answers should be complete, understandable and readable in the images you upload. Unreadable answers would be regarded as void.
- Only image files (such as JPG or PNG) or PDF files can be uploaded to the submission form. When you need to upload your answer in multiple files, you must clearly indicate their order on the answer sheets.
- For each question, you should **only** use the data structures, definitions and functions provided (or explicitly allowed) with that question. You **MUST NOT** use other data structures, definitions or functions, which are not explicitly allowed in the question.

QUESTION 1 (15)

We will use an array in order to store a full binary tree.



=>

8	3	5	6	9	2	4

Complete the given code that returns the n^{th} node in the given level. For example, find_data(3, 2) will return 9, find_data(2, 1) will return 3, and so on. You must realize the function using the definitions given below with less than or equal to 4 code lines.

```
#define ARRAY_SIZE 7
int tree[ARRAY_SIZE];
int pow (int x, int y); //assume pow function is predefined and returns x^y.

int find_data(int level, int n){

        _____________________________________________________________;
        _____________________________________________________________;
        _____________________________________________________________;
        _____________________________________________________________;

}
```

QUESTION 2 (15)

An important drawback with an array-based implementation of a stack is the resizing that must be done when the stack is full. A solution for overcoming this problem employs periodic resizing of the array by some constant amount c. That is, when the array of size k is full, a new array of size k + c is created and the elements from the original array are copied into the new one, maintaining the original order so that the stack remains intact.

Give **a complete analysis** of this solution over a sequence of **n** push operations on the stack. Assume that the stack is empty at the beginning. **Report the total cost and the average cost per push in Big-Oh notation.**

QUESTION 3 (25)

```
int magic(int low, int high) {
    return (int) low + (int) (random()%(high-low+1));
}

struct Node {
    int value = magic(0,10);
    struct Node* next;
};
```

Based on the definitions given above, fill in the blanks to realize the described function definitions exactly.
Each blank is a single line of code. Therefore, you MUST NOT write more than a single line of code in a blank.

(10) a) Create a C++ compatible function called **CreateCircularLL**(int **n**) which generates a circular linked list that consists of **n** number of nodes. The entry node to the list must be named **firstNode**. You must realize the function with less than or equal to 10 code lines.

```
Node* CreateCircularLL(int n) {
    _____________________________ = new Node;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
}
```

(15) b) Create a C++ compatible function called **SearchCircularLL**(Node* **firstNode**) that finds which is the foremost element of the circular list that has the value 0. **firstNode** is the entry node of a circular list created by CreateCircularLL above. As an example, considering the list (6,1,2,0,5,0), 4th element is the foremost element with the value 0, therefore SearchCircularLL on this list would return 4. Assume that the list contains at least one element with 0 value. Also assume that **no new memory allocation** is necessary in SearchCircularLL. You must realize the function with less than or equal to 10 code lines.

```
int SearchCircularLL(Node* firstNode) {
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
    ____________________________________________________________;
}
```

QUESTION 4 (20)

Complete the **recursive** function **SearchValue** that searches for a given value in a regular linked list and returns the position of the value in the linked list if found, otherwise returns -1. The first item in the linked list has position 1. You must realize the function with less than or equal to 6 code lines.

```
struct Node{
       int data;
       struct Node* next;
};


int SearchValue (struct Node* firstNode, int value, int index) {
      _____________________________________________________________;
      _____________________________________________________________;
      _____________________________________________________________;
      _____________________________________________________________;
      _____________________________________________________________;
      _____________________________________________________________;
}
```

QUESTION 5 (25)

(15) a) Consider a virtual ant moves in space from point (x=0, y=0, z=0) to (x=a, y=b, z=c), where a,b and c are positive integers. **The ant moves in integer units, always in positive directions and parallel to the coordinate axis**. In other words, when it moves from (0,0,0) it can go to (1,0,0) or (0,1,0) or (0,0,1) and nowhere else. Write a **recursive** function **countWays(int a, int b, int c)** in the space given below to compute in how many **different** ways the ant can reach point (a,b,c) when it starts from (0,0,0). In other words, your function must count the number of distinct paths from (0,0,0) to (a,b,c) according to the above rules.

```
int countWays(int a, int b, int c) {




}
```

(10) b) Complete the recursive function **mod(int n, int d)** below, which returns the remainder (i.e., ***modulus)*** of $n \div d$. Please note that this does not imply integer division of n by d. And you **must *not*** use division (/), multiplication (*), or modulus (%) operators, or any other helper functions. You should also assume that $n \geq 0$ and $d \geq 1$.

```
int mod(int n, int d) {




}
```