National University of Computer and Emerging Sciences

School of Computing **Fall 2018** **Islamabad Campus**

CS-201 Data Structures

Serial No:

Midterm Exam

Total Time: 2 Hours

Total Marks: 120

Wednesday, October 24, 2018

Course Instructors

Dr. M. Adnan Tariq

Mr. Syed Muhammad Hassan Mustafa

_____

Signature of Invigilator

_____ _____ _____ _____

Student Name Roll No Section Signature

DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

Instructions:

1. Attempt on question paper. Attempt all of them. Read the question carefully, understand the question, and then attempt it.
2. Rough work space is provided separately on the question paper. No additional sheet will be provided for rough work. Marks are only awarded if correct working of the algorithm/dry run is shown in the provided rough space.
3. After asked to commence the exam, please verify that you have twelve (12) different printed pages including this title page. There are a total of two(2) questions.
4. Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.

	Q-1	Q-2	Total
Marks Obtained			
Total Marks	**60**	**60**	**120**

Vetted By: _____ **Vetter Signature:** _____

Question 1 [60 Marks]

1) Select the correct infix representation of the following prefix expression: `+-*^1234//56+78` [4 marks]

a. `1^2*3-4+5/6/7+8`
b. `1^23-4+5/6/(7+8)`
c. `1^2*3-4+5/6/(7+8)`
d. `((((1^2)*3)-4)+((5/6)/7+8))`

Space for rough work:

2) Select the correct postfix representation of the following infix expression: `a + b * (c – d) – e / (f + g / h )` [2 marks]

a. `a b c d - * + e - f g h / + /`
b. `a b c d - * e f g h / + / – +`
c. `a b c d - * + e f g h / + / –`
d. None of above

Space for rough work:

3) Recall algorithm to convert infix expression (with parenthesis) into postfix. Given the following infix expression: `A + B * ( C / D / E )`. What is the state of the `opstk` (i.e., operators on the stack) before reading the token/symbol E from the infix expression? [6 marks]

a. Opstk: + * (/ /
b. Opstk: + * (/
c. Opstk: + * /
d. Opstk: + * / /

Space for rough work:

4) Recall algorithm to convert infix expression into prefix (Hint: Using reversal of infix expression). Which of the following infix expressions will result in the violation of associativity property. [2 marks]

a. a - (b + c) * d
b. a - b + (c * d)
c. (a - b) + c * d
d. None of the above
e. All of the above

Space for rough work:

5) The array-based Queue throws an exception when the array's capacity has been reached. Consider the following alternative: Create a larger array, using the resize method. The cost of a resize that makes the array larger is proportional to the new size. Suppose we expand the array's capacity by one element. What will be the running time (Big-Oh notation) in the worst-case for a sequence of N insertions? [1 mark]

a. O(1)
b. O(N)
c. $O(N^2)$
d. O ($\log_2$ N)
e. None of above

6) The array-based Queue throws an exception when the array's capacity has been reached. Consider the following alternative: Create a larger array, using the resize method. The cost of a resize that makes the array larger is proportional to the new size. Suppose we double the array's capacity (assume the capacity is not zero). What will be the running time (Big-Oh notation) in the worst-case for a sequence of N insertions? [1 mark]

a. O(1)
b. O(N)

c. $O(N^2)$
d. $O(\log_2 N)$
e. None of above

7) A stack `bStack` contains the following items: `7, 8, -3, 14, 5` (item 7 is at the top of the stack). What is the output of the following program? [3 marks]

```
int x;
while (!bStack.isEmpty()){
    bStack.pop(x);
    if (x>0 && !bStack.isEmpty())
        bStack.pop();
    cout << x << endl;
}
```

a. 7, 8, -3, 14
b. 7, -3, 14, 5
c. 7, 14, 5
d. 7, -3, 14

Space for rough work:

8) Given a linked list with the following items: 1 → 5 → 6 → 7 → NULL. What will be the output of the following function? [8 marks]

```
void Function1(struct node** headRef) {
    struct node* first;
    struct node* rest;
    if (*headRef == NULL) return; // empty list base case

    first = *headRef;
    rest = first->next;

    if (rest == NULL) return; // empty rest base case
    Function1(&rest); // Recursively call

    first->next->next = first;
    first->next = NULL; // (tricky step -- make a drawing)
    *headRef = rest;
}
void main (){
    . . .
    Function1 (&head);
    . . .
}
```

a. 7→6→5→1→NULL (with head pointing to element 7)
b. 7→6→1→5→NULL (with head pointing to element 5)
c. 1→5→6→7→NULL (with head pointing to element 7)
d. 7→6→7→6 … (circular linked list containing 7 and 6)
e. 7→6→5→1→NULL (with head pointing to element 1)
f. Segmentation fault
g. None of above

Space for rough work:

9) Complete the `Function2` below to provide the same functionality/output as provided by `Function1` in the above question (i.e., Question 8). [8 marks]

```
01: void Function2(struct node** headRef) {
02:     struct node* result = NULL;
03:     struct node* current = *headRef;
04:     struct node* next;
05:     while (_____________________________) {
06:
07:         current->next = result;
08:
09:         current = next;
10:     }
11:

12: }
13: void main (){
14:     . . .
15:     Function2 (&head);
16:     . . .
17: }
```

a. Line 05: `current != NULL`
   Line 06: `next = current→next;`
   Line 08: `result = current;`
   Line 11: `headRef = result;`

b. Line 05: `current->next != NULL`
   Line 06: `next = current;`
   Line 08: `result = current;`
   Line 11: Not required
c. Line 05: `current != NULL`
   Line 06: `next = current→next;`
   Line 08: `result = current;`
   Line 11: `*headRef = result;`
d. Line 05: `current != NULL`
   Line 06: `next = current;`
   Line 08: Not required
   Line 11: `*headRef = result;`
e. None of the above

Space for rough work:

10) Complete the following function that remove duplicates from a sorted list (e.g., in linked list 1→2→2→3→5 duplicate 2 is removed and the resultant list will be 1→2→3→5). [8 marks]

```
01: void RemoveDuplicates(struct node* head) {
02:     struct node* current = head;
03:     if (current == NULL) return;
04:     while(current->next!=NULL) {
05:         if (_____________________________________________) {
06:
07:
08:
09:         }
10:         else {
11:             current = current->next;
12:         }
13:     }
14: }
15:
16: void main (){
17:     . . .
18:     RemoveDuplicates(head);
19:     . . .
20: }
```

a. Line 05: current→data == ptrNext→ data
   Line 06: ptrNext = current→next→next;
   Line 07: No need
   Line 08: current→next = ptrNext;
b. Line 05: current→data == current→next→data
   Line 06: struct node* ptrNext = current→next→next;
   Line 07: free(current→next);
   Line 08: current→next = ptrNext;
c. Line 05: current→data == current→next→data
   Line 06: ptrNext = current→next→next;
   Line 07: free(ptrNext);
   Line 08: No need
d. Line 05: current→data == current→next→data
   Line 06: struct node* ptrNext = current→next;
   Line 07: free(current);
   Line 08: current→next = ptrNext;
e. None of the above

Space for rough work:

11) Which of the following operations is performed more efficiently by doubly linked list than by singly linked list? [1 mark]

a. Deleting a node whose location is given
b. Searching of an unsorted list for a given item
c. Inserting a node after the node with given location
d. Traversing a list to process each node
e. None of the above

12) Suppose in a linked list address of first node is 1020 and node size is 4, what will be the address of node at 5th position? [1 mark]

a. 1022
b. 1025
c. 1040
d. 1024
e. None of the above

13) Suppose a circular queue of capacity `(n-1)` elements is implemented with an array of `n` elements. Assume that the insertion and deletion operations are carried out using `REAR` and `FRONT` as array index variables, respectively. Initially `REAR=FRONT=0`. The conditions to detect queue full and queue is empty are? [1 mark]

a. Full: `(REAR+1)mod n == FRONT`
   Empty: `REAR==FRONT`
b. Full: `(REAR+1)mod n == FRONT`
   Empty: `(FRONT+1) mod n == REAR`
c. Full: `REAR==FRONT`
   Empty: `(REAR+1) mod n==FRONT`
d. Full: `(FRONT+1)mod n==REAR`
   Empty: `REAR==FRONT`

14) How many minimum queues are needed to implement a stack? Consider the situation where no other data structure like arrays, linked list is available to you. [1 mark]

a. 1
b. 2
c. 3
d. 4
e. Not Possible

15) How many minimum stacks are needed to implement a queue. Consider the situation where no other data structure like arrays, linked list is available to you. [1 mark]

a. 1
b. 2
c. 3
d. 4
e. Not Possible

16) Complete the following code snippet to implement `enqueue` and `dequeue` operations of Queue using a user stack and function call stack. [12 marks]

```
01: struct Queue {
02:     stack<int> s;

03:     void enQueue(int x) {
04:

05:

07:     }

08:     int deQueue()  {
09:         if (s.empty()) {
10:             cout << "Q is empty";
11:             exit(0);
12:         }
13:

14:         s.pop(x);
15:         if (_____________________________)
16:             return x;
17:         int item = deQueue();
18:

19:         return _____________________;
20:     }
21: };
```

a. Line 04: `s.pop();`
   Line 05: `s.push(x);`
   Line 13: Not needed
   Line 15: `s.full()`
   Line 18: `s.push(x);`
   Line 19: `return item;`
b. Line 04: Not needed
   Line 05: `s.push(x);`
   Line 13: `int x;`
   Line 15: `s.empty()`
   Line 18: `s.push(item);`
   Line 19: `return x;`

c. Line 04: `s.push(x);`
   Line 05: Not needed
   Line 13: `int x;`
   Line 15: `s.empty()`
   Line 18: `s.push(x);`
   Line 19: `return item;`

d. Line 04: Not needed
   Line 05: `s.push(x);`
   Line 13: `int x;`
   Line 15: `s.full()`
   Line 18: `s.push(x);`
   Line 19: `return item;`

e. None of above

Space for rough work: