0217

National University of Computer and Emerging Sciences
Chiniot-Faisalabad Campus

Data Structures (CS2001)

Sessional-II Exam

Date: April 8th 2024

Course Instructor(s)

Mr. Rizwan ul Haq, Ms. Nabeela Ashraf, Ms. Ayesha Liaqat, Dr. Anwar Shah, Mr. Muhammad Usman Joyia

Total Time (Hrs):	1
Total Marks:	45
Total Questions:	3

[redacted]

Roll No   Section   Student Signature

[redacted]

Do not write below this line

Attempt all questions.

CLO # 2: Evaluate different data structures in terms of memory complexity and time

14

Q1: [15]

a. Apply stack to convert this infix expression into postfix expression.

(9 + ((8 + 7) * 6)) + (5 * (4 + ((3 * 2) + 1)))

Symbol	Postfix String	Opstk
(		(
9	9	(
+	9	(+
(	9	(+(
(	9	(+((
8	98	(+((
+	98	(+((+
7	987	(+((+
)~~*~~	987+	(+(
*	987+	(+(*
6	987+6	(+(*
)	987+6*	(+
)	987+6*+	
+	987+6*+	+
(	987+6*+	+(

02

5	987+6*+5	+(
*	987+5*+5	+(*
(	987+5*+5	+(*(
4	987+5*+54	+(*(
+	987+5*+54	+(*(+
(	987+5*+54	+(*(+(
(	987+5*+54	+(*(+((
3	987+5*+543	+(*(+((
*	987+5*+543	+(*(+((*
2	987+5*+5432	+(*(+((*
)	987+5*+5432*	+(*(+(
+	987+5*+5432*	+(*(+(+
1	987+5*+5432*1	+(*(+(+
)	987+5*+5432*1+	+(*(+
)	987+5*+5432*1++	+(*
)	987+5*+5432*1++*	+
	987+5*+5432*1++*+	

10

b. Evaluate postfix Expression obtained from Part a using stack.

Symbol	Opnd1	Opnd2	Result	Stack
9	9			9
8	9	8		9.8
7	8	7		9.8.7
+	8 9	15 7	15	9.15
5	15	5		9.15.5 6
*	15 9	75 5	75	9.75
+	84		84	84
5	84	5		84.5
4	5	4		84.5.4
3	4	3		84.5.4.3
2	3	2		84.5.4.3.2 ?
*	4	6	6	84.5.4.6
1	6	1		84.5.4.6.1
+	4	7	7	84.5.4.7
+	5	11	11	84.5.11
*	84	55	55	84.55
+	139		139	139

4

84
+55
139

CLO # 3: Design/create appropriate data structures to solve real-world problems related to the program

Q2: [10+5=15]

Queue is a data structure that allows one element to be enqueued or dequeued at a time. Consider the following implementation of Queue for the question.

```
class Queue
{
        struct QueueNode
        {
                int data;
                QueueNode *next;
        }*front,*rear;

        bool isEmpty();
public:
        Queue();
        ~Queue();
        void Enqueue(int val); //Add am element to the back of the queue
        bool Dequeue(int &val);//Removes the element from the front and
                                 //returns its value in val
};
```

a. The above provided implementation does not provide functionality of peek, that allows to check the element at the front of the queue but it remains at the front of the queue. You need to implement a new class for this named enhancedQueue. Write down the code for **Enqueue**, **Dequeue** and a new functionality **Peek** for the below given enhancedQueue class using C++.
Hint: You can create any temporary objects for the Queue class in your function if required.
Note: No other function can be created.

```
class enhancedQueue {
        Queue myQ;

public:
        enhancedQueue();
        ~enhancedQueue();
        void Enqueue(int val);
        bool Dequeue(int &val);
        bool Peek(int &val);

};
```

b. A **Stack** class is already implemented with **push** and **pop** operations. You are required to reverse any queue using the stack. Write down the code for the function with prototype provided below using C++ syntax.
Note: Stack is already implemented and has two functions available to access as below

```
class Stack {
   int* stAr;
   const int SIZE;
   int top;
   public:
      Stack(int s);
      ~Stack();
      bool push(int x);
      bool pop(int& val);
   };
void reverseQueue(Queue &Q); //Write down implementation for this function
```

CLO # 3: Design/create appropriate data structures to solve real-world problems related to the program

Q3: [5+10=15]

a. Design a binary search tree (BST) to store student records where each student has a unique ID and associated data (such as name, age, GPA, etc.). Assume that the student IDs are integers. You are provided with dry-run scenarios for inserting and traversing records in the BST. Design the BST and demonstrate the outcomes of each dry run.

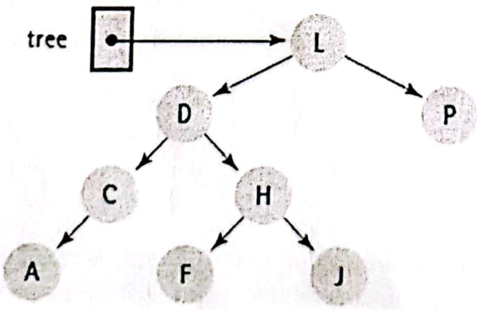
   i. Insert student records stepwise with IDs: 50, 30, 70, 20, 40, 60, 80, 45.

   ii. Perform the pre-order and post-order traversal for the above BST.

   Pre-order: 50, 30, 20, 40, 45, 70, 60, 80

   Post-order: 20, 45, 40, 30, 60, 80, 70, 50

b. Delete the nodes from the below BST in the following order:
J, C, L, D, A



When deleting a node with two children, use the **immediate predecessor** and handle the other two cases as well. Implement the below function **recursively**.

```
void deleteNode(Node*& root, int x);
```

National University of Computer and Emerging Sciences, Chiniot-Faisalabad Campus

Course Name:	Data Structures	Course Code:	CS2001
Degree Program:	BS(CS) | BS(SE) | BS(AI)	Semester:	Spring-2024
Exam Duration:	60 Minutes	Total Marks:	40
Paper Date:	Monday, February 26, 2024	No of Page(s):	04
Sections:	ALL		
Exam Term & Type:	1st Sessional I Closed Book	Required Answer Book:	No
Course Instructor	Mr. Muhammad Usman Joyia, Dr. Anwar Shah, Mr. Rizwan ul Haq Ms. Nabeela Ashraf, Ms. Ayesha Liaqat		

Student Name: [redacted] Roll No. [redacted] Invigilator's Signature

Q.Part#	Q.1	Q.2	Q.3	Q.4	Total Obtained	Total Marks	Examiner Signature and Date
Marks	10	10	10	10	26	40	6/2/24

Instruction/Notes: Attempt all questions. Programmable calculators are not allowed.

Question 1: Assuming that the ***List*** is a class and ***Reverse***() is the member function that is supposed to be reversing the whole singly-linked list. You are required to design an inplace solution to reverse the linkedlist in a way that all links in the list are reversed and the last node becomes the head. **(10 Marks)**

Hint: Inplace solution doesn't use the extra memory to copy the value or elements available in the original data structure. However, you are free to use pointers/references as many as you want.

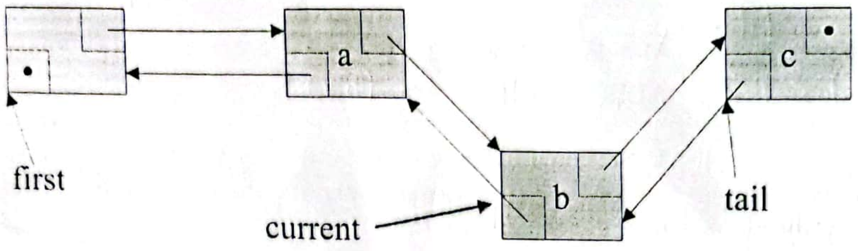
```
void List::Reverse(){
    ~~Node * temp = head;~~   if(head == nullptr){
    ~~Node * curr~~             }  return;
    Node * prevPtr = NULL;
    Node * currPtr = head;
    Node * nextPtr;
    while(currPtr ~~->next~~ != NULL){
        nextPtr = currPtr->next;
        currPtr->next = prevPtr;
        prevPtr = currPtr;
        currPtr = nextPtr;
    }
    ~~currPtr = currPtr->next;~~
    prevPtr = head;
}
    head = prevPtr
```

9

Question 2: Consider the scenarios given below for doubly linked list and answer them for the questions. **(5+5 = 10 Marks)**

7

a) Consider the following scenarios for **insertion** in the doubly linked list. Your task is to rearrange the following steps in a *sequence* for the process of **insertion** in the doubly linked list. (Just arrange **Characters** rather than writing complete sentence!) **(05 Marks)**



A. newNode->prev->next = newNode; 5

B. newNode->next = current->next; 3

C. current = newNode 2

D. newNode->next->prev = newNode; 6

E. newNode->prev = current; 4

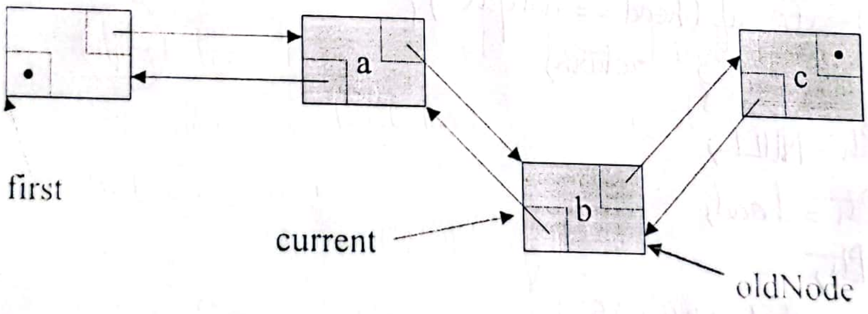
F. newNode = new DoublyLinkedListNode() 1

4

Provide ordering of statement here:

F. C. B. E. A. D.

b) Consider the following scenario for **deletion** in the doubly linked list. Your task is to rearrange the following steps in a *sequence* for the process of **deletion** in the doubly linked list. (Just arrange **Characters** rather than writing complete sentence!) **(05 Marks)**



3

A. current = oldNode->prev; 2

B. oldNode->next->prev = oldNode->prev; 4

C. delete oldNode; 5

D. oldNode=current; 1

E. oldNode->prev->next = oldNode->next; 3

Provide ordering of statement here:

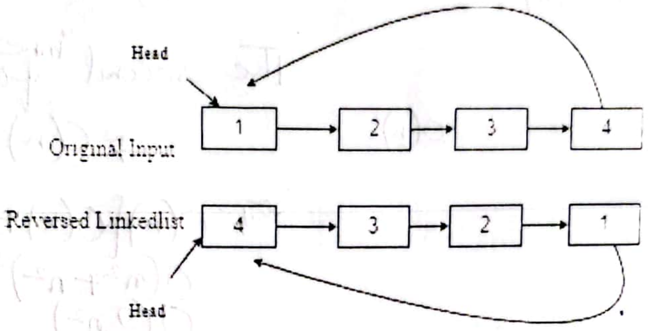
A. D. E. B. C.

What if head node

2

Question 3: Assuming again that the ***List*** is a class and ***Reverse***() is the member function that is supposed to be reversing the whole circular-linked list. You are required to design an outplace solution to reverse the linkedlist in a way that all links in the list are reversed and the last node becomes the head. To reverse the list, you're only allowed to insert the node at front in new list. **(10 Marks)**

Hint: Outplace solution always use the extra memory to copy the value or elements available in the original data structure



```
void List::Reverse(){
        if(head == nullptr){
            return;
        }

        Node * prevPtr = NULL;
        Node * currPtr = head;
        Node * nextPtr;

     while(currPtr != head){
          nextPtr = currPtr->next;
          currPtr->next = prevPtr;
          prevPtr = currPtr;
          currPtr = nextPtr;
       }
          prevPtr->next = head;
          prevPtr = head;

   }
```

2

8

Question 4: a) Explain the worst-case time complexity of this program in terms of the input variable **N**. Identify the loops responsible for the dominant terms in the time complexity and explain their behavior as **N** grows. **(05 Marks)**

```
#include <stdio.h>

int main() {
    int N = 10;  O(1)
    int sum = 0; O(1)

    for (int i = 1; i <= N; i++) { O(n)
        int j = 1;

        while (j <= N) {  O(n)
            sum += i + j;
            j++;
        }

        for (int k = 0; k < N; k++) { O(n)
            sum += k;
        }
    }

    Std::cout<<"sum is "<<sum<<std::endl;
    return 0;
}
```

The outer first for-loop:

$\Rightarrow O(n)$

The inner while-loop:

$\Rightarrow O(n)$

The second inner for-loop:

$\Rightarrow O(n)$

So, $O(n)[O(n)+O(n)]$

$O(n^2+n^2)$

$O(2n^2)$

5

b) Linear and binary search are two of the most fundamental and commonly used search algorithms. Binary search is a more efficient algorithm that searches a sorted list by repeatedly dividing the search interval in half, requiring the list to be sorted beforehand. The following Array A has sorted data.

Array A

Data	4	8	10	15	18	21	24	27	29	33	34	37	39	41
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Your task is to mathematically calculate the worst time complexity for binary search. There will be no marks for theoretical discussions. **(05 Marks)**

Binary Search

As, the Binary Search has Linearithmic time Complexity, so, the calculation of worst case in big-Oh notation is:

$O(n.\log n)$

As, outer-loop consists of Linear time complexity, $O(n)$, running upto n-times.

For outer loop:

$O(n)$

As, the inner loop consists of logarithmic time complexity $O(\log n)$, dividing the iteration at each

3