National University of Computer & Emerging Sciences, Karachi
Fall 2022 CS-Department - Solution
Midterm 1
26th September 2022, 8:30 AM – 9:30 AM

Course Code: CS2001/AI	**Course Name:** Data Structures
Instructor Name: Dr. Jawwad A Shamsi , Dr. Fahad Sherwani, Ms. Anam Qureshi, Mr. Zain ul Hassan, Mr. Shoaib Rauf, Mr. Shahroz, Ms. Sobia Iftikhar, Ms. Abeer Gauhar, Mr. Ali Fatmi	
Student Roll No:	**Section No:**

Instructions:

- Please return the question paper.
- Please read each question completely before answering it. There are 4 **questions and 2 pages**
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.
- Show all steps clearly.

Time: 60 minutes. **Max Marks**: 20 points

Question 1:. **Arrays** (CLO: 1) **5 points**

Suppose A, B, C are arrays of integers of size M, N, and M + N respectively. The numbers in array A appear in ascending order while the numbers in array B appear in descending order. Write a user defined function to produce third array C by merging arrays A and B in ascending order. Use A, B and C as arguments in the function.

Solution in C

Solution:

```
#include<iostream>

using namespace std;

void Merge(int A[], int B[], int C[], int N, int M, int &K);

int main()

{

        int A[100], B[100], C[200],i,n,m,k;

        cout<<"\nEnter number of elements you want to insert in first array ";

        cin>>n;

        cout<<"Enter element in ascending order\n";

        for(i=0;i<n;i++)

        {

                cout<<"Enter element "<<i+1<<":";

                cin>>A[i];
```

```
	}
	cout<<"\nEnter number of elements you want to insert in second array ";
	cin>>m;
	cout<<"Enter element in descending order\n";
	for(i=0;i<m;i++)
	{
		cout<<"Enter element "<<i+1<<":";
		cin>>B[i];
	}
	Merge(A,B,C,n,m,k);
	cout<<"\nThe Merged Array in Ascending Order"<<endl;
	for(i=0;i<k;i++)
	{
		cout<<C[i]<<" ";
	}
	return 0;
}
void Merge(int A[], int B[], int C[], int N, int M, int &K)
{
	int I=0, J=M-1;
	K=0;
	while (I<N && J>=0)
	{
		if (A[I]<B[J])
			C[K++]=A[I++];
		else if (A[I]>B[J])
			C[K++]=B[J--];
		else
		{
			C[K++]=A[I++];
```

```
                J--;
            }
        }
        for (int T=I;T<N;T++)
            C[K++]=A[T];
        for (int T=J;T>=0;T--)
            C[K++]=B[T];
}
```

Question 2: Recursion with Backtracking (CLO: 2) **5 points**

Given a square maze containing positive numbers, find a path from the corner cell (marked as 2 in bold) to the middle cell (marked as 0 in bold). You can move exactly 'n' steps from any cell in two directions i.e. right and down. where **n is value of the cell**. For instance, if a cell has a value 2, the number 2 indicates that movement along 2 cells are allowed. These 2 cells can be taken in any combination and in any of the allowable direction. For instance, 1 step right and 1 step down will be allowed; however, 2 cells right and 2 cells down will not be allowed as this will count to 4 steps in total. The movement should not exceed the boundary.

Your task is to write a function using recursion with backtracking to find a path from corner cell to middle cell in maze.

Sample Input: 5 x 5 maze

	j=0	j=1	j=2	j=3	j=4
i=0	**2**	**2**	**4**	**4**	**3**
i=1	**3**	**4**	**4**	**2**	**2**
i=2	**1**	**1**	**0**	**3**	**2**
i=3	**3**	**2**	**2**	**1**	**1**
i=4	**3**	**3**	**4**	**3**	**1**

Where cell (0,0) with value **2** is the source and the destination is (2,2) with value 0.

Solution

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

```
#define N 5

bool findpath(int matrix[N][N], int i, int j,int solution[N][N]);
void printPath(int solution[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf(" %d ", solution[i][j]);
        printf("\n");
    }
}

bool Safe(int matrix[N][N], int i, int j)
{
    if (i >= 0 && i < N && j >= 0 &&  j < N && matrix[i][j] != 0)
    {
            return true;
            }
    return false;
}

bool soultionmatrix(int maze[N][N])
{
    int sol[N][N] = { { 0, 0, 0, 0 },
                      { 0, 0, 0, 0 },
                      { 0, 0, 0, 0 },
                      { 0, 0, 0, 0 } };
```

```
    if (findpath(maze, 0, 0, sol) == false) {
        printf("Solution doesn't exist");
        return false;
    }

    printPath(sol);
    return true;
}


bool findpath(int maze[N][N], int x, int y,int solution[N][N])
{

    if (x == 2 && y == 2)
            {
        solution[x][y] = 1;
        return true;
    }


    if (Safe(maze, x, y) == true) {

        solution[x][y] = 1;

        for (int i = 1; i <= maze[x][y] && i < N; i++) {


            if (findpath(maze, x + i, y, solution) == true)
```

```
            return true;



        if (findpath(maze, x, y + i, solution) == true)
            return true;
      }



      solution[x][y] = 0;
      return false;
   }

   return false;
}



int main()
{
        //First Matrix solution exsists
   int matrix[N][N] ={{ 2, 1, 1, 0, 2},
                  { 3, 0, 1, 1, 3 },
                    { 0, 1, 1, 1, 1 },
                    { 1, 0, 0, 1, 0 },
                                        { 1, 0, 0, 1, 0 }};

        //Second Matrix solution exsists
//    int matrix[N][N] ={{ 2, 2, 4, 4, 3},
//                    { 3, 4, 4, 2, 2 },
```

```
//                  { 1, 1, 0, 3, 2 },
//                  { 3, 2, 2, 1, 1 },
//                                          { 3, 3, 4, 3, 1 }};


        //Third Matrix solution does not exsist
//         int matrix[N][N] ={ { 1, 0, 1, 0, 0 },
//                             { 1, 0, 1, 2, 2 },
//                               { 1, 0, 0, 3, 2 },
//                               { 1, 0, 1, 1, 1 },
//                                                   { 1, 0, 1, 3, 1 }};



   soultionmatrix(matrix);
   return 0;
}
```

Question 3: Linked List (CLO:3) **5 points**

Write a function to reverse the specified portion of the given linked list.

For instance:

Input:

Linked List: 1→ 2 → 3 → 4 → 5 → 6 → 7

Start position = 2

End position = 5

Output:

1→ 5 → 4 → 3 → 2 → 6 → 7

Solution

Algorithmic Steps:

1. Skip the first ‘m’ nodes.
2. Traverse and reverse the sublist from position m to n.
3. Fix the pointers and return the head node.

Code (with comments):

```
void reverse(Node* &head, int m, int n)
{
    // base case
    if (m > n) {
        return;
    }
    Node* prev = NULL;        // the previous pointer
    Node* curr = head;        // the main pointer
    // 1. Skip the first `m` nodes
    for (int i = 1; curr != NULL && i < m; i++)
    {
        prev = curr;
        curr = curr->next;
    }
    // `prev` now points to (m-1)'th node
    // `curr` now points to m'th node
    Node* start = curr;
    Node* end = NULL;
    // 2. Traverse and reverse the sublist from position `m` to `n`
    for (int i = 1; curr != NULL && i <= n - m + 1; i++)
    {
        // Take note of the next node
        Node* next = curr->next;
        // move the current node onto the `end`
        curr->next = end;
        end = curr;
```

```
        // move to the next node
        curr = next;
    }
    /*
        `start` points to the m'th node
        `end` now points to the n'th node
        `curr` now points to the (n+1)'th node
    */
    // 3. Fix the pointers and return the head node
    if (start)
    {
        start->next = curr;
        if (prev != NULL) {
            prev->next = end;
        }
        // when m = 1, `prev` is nullptr
        else {
            // fix the head pointer to point to the new front
            head = end;
        }
    }
}
```

.

Question 4 Elementary Sorting (CLO:3) **5 points**

Write a function that takes a NxN 2D array and its dimension N as parameters and sort the given array such that after sorting the values in the array are in column-wise ascending order.

Example:

Before sorting:

2	**3**	**2**	**8**
9	**4**	**54**	**5**
1	**7**	**4**	**11**
6	**1**	**9**	**2**

After sorting:

1	**2**	**5**	**9**
1	**3**	**6**	**9**
2	**4**	**7**	**11**
2	**4**	**8**	**54**

Solution:

The 2-D array can be converted into a 1-D array and the preferred sorting algorithm can be applied

**********Good Luck **********