Assignment - 4

Tejaswini
AP19110010165
SEC - CSE - G1

1. Insertion And deleting:

```
Public class Linkedlist
{
    // Head of list
    public class Node head;

    // Head list Node
    public class Node
    {
      public int data;
      public Node next;
      public Node (int d)
      {
          data = d;
          next = null;
      }
    }

    // Function to delete the nth node from
    // the end of the given linked list
    void deleteNode (int key)
    {
        // First pointer will point to
        // the end of the given linked list
        void deleteNode (int key)
```

```
{
    // first pointer will point to
    // the head of the linked list
    Node first = head;
    // second pointer will point to the
    // nth node from the begining.
    Node second = head;
    for (int i=0; i< key; i++)
    {
        // if count of nodes in the given
        // linked list is <=N
        if (second.next == null)
        {
            // if count = N i.e delete the head node
            if (i == key-1)
                head = head.next;
            return;
        }
        second = second.next;
    }
    // increment both the pointers by one until
    // second pointer reaches the end
    while (second.next != null)
    {
        first = first.next;
        second = second.next;
    }
```

```
// first must be pointing to the
// Nth node from the end by now
// so, delete the node first is pointing to
first.next = first.next.next;
}
// function to insert a new Node at front of the list
public void push (int new_data)
{
    Node New_node = New Node (New_data);
    New_node.next = head;
    head = new_node;
}
// function to print the linked list
public void printList()
{
    Node tnode = head;
    while (tnode != null)
    {
        console.write (tnode.data + " ");
        tnode = tnode.next;
    }
}
// Driver code
public static void Main (string [] args)
{
    Linked List llist = new LinkedList ();
    llist.push (7);
    llist.push (1);
    llist.push (3);
```

```
llist.push(2);
console.writeline("\n Created Linked list is :");
llist.printList();

int N = 1;
llist.deleteNode(N);

console.WriteLine("\n Linked list after Deletion is:");
llist.printList();
}
}
```

2. Merging alternate nodes

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

// Data Structure to store a linked list node.
struct Node
{
    int data;
    struct Node * next;
};

// Helper function to print given linked list
void printList (struct Node * head)
{
    struct Node * Ptr = head;
    while (Ptr)
    {
        printf("%d->", Ptr->data);
```

```
        ptr = ptr->next;
    }
    printf("NULL\n");
}
// Helper function to insert new Node in the beginning of the
linked list void push(struct Node** head, int data)
void push(struct Node** head, int data)
{
    struct Node* NewNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}
// Function to construct a linked list by merging alternate nodes of
// two given linked lists using dummy node.
struct Node* shuffleMerge(struct Node* a, struct Node* b)
{
    struct Node dummy;
    struct Node* tail = &dummy;
    dummy.next = NULL;
    while(1)
    {
        // empty list cases.
        if(a == NULL)
        {
            tail->next = b;
            break;
```

```
    }
    else if (b == NULL)
    {
        tail->next = a;
        break;
    }
    // common case: move two nodes to tail
    else
    {
        tail->next = a;
        tail = a;
        a = a->next;

        tail->next = b;
        tail = b;
        b = b->next;
    }
  }
  return dummy.next;
}

// main method
int main(void)
{
```

```
// input keys
int keys[] = {1, 2, 3, 4, 5, 6, 7};
int n = sizeof(keys)/sizeof(keys[0]);

struct Node *a = NULL, *b = NULL;

for (int i = n-1; i >= 0; i = i-2)
    push(&a, keys[i]);

for (int i = n-2; i >= 0; i = i-2)
    push(&b, keys[i]);

// print both linked list
print("first list :");
print list(a);
print("second list :");
print list(b);
struct Node* head = shuffleMerge(a, b);
printf("After Merge :");
print list(head);
return 0;
}
```

3.

3. /* An efficient program to print subarray with sum as given sum */

```
#include <stdio.h>

/* Returns true if the there is a subarray of arr[] with
a sum equal to 'sum'
otherwise returns false. Also, prints the result */
int subArraySum(int arr[], int n, int sum)
{
    /* Initialize curr_sum as value of first element
    and starting point as 0*/
    int curr_sum = arr[0], start = 0, i;

    /* Add elements one by one to curr_sum and if
    the curr_sum exceeds the
    sum, then remove starting elements */
    for (i = 1; i <= n; i++)
    {
        // if curr_sum exceeds the sum, then remove the
        starting elements
        while (curr_sum > sum && start < i-1)
        {
            curr_sum = curr_sum - arr[start];
            start++;
        }

        // if curr_sum becomes equal to sum, then return
        if (curr_sum == sum)
```

```
{
    printf ("sum found between indexes %d and %d", start, i-1);
        return 1;
    }

    // Add this element to curr_sum
    if (i<n)
    curr_sum = curr_sum + arr[i];
}

//if we reach here, then no subarray
printf("No Subarray found");
return 0;
}

// Driver program to test above function
int main()
{
int arr[] = {15, 2, 5, 9, 6, 13, 18, 20};
int n = size of (arr) / size of (arr[0]);
int sum = 23;
subarray Sum (arr, n, sum);
return 0;
}
```

4.(i) reverse Order;

```
#include <stdio.h>
#include "stack.h"
#include "qq.h"

int main()
{
    int n, arr[20], i, j=0;
    struct stack s;
    int it stack (&s);
    printf ("Enter no");
    scanf ("%d", &n);
    for (i=0; i<n; i++)
    {
        printf("Enter values:");
        scanf ("%d", & arr[i]);
    }
    for (i=0; i<n; i++)
    {
      insert (arr[i]);
    }
     while (j!=n)
     {
         push (&s, del ());
         j++;
     }
    printf ("Reverse is");
    while (s.top! =-1)
    {
      printf ("%d", pop(&s));
    }
```

```
    printf("\n");
    return 0;
}
```

4.(i) Alternative Alternate Nodes:

```
// C code to print Alternate nodes
#include <stdio.h>
#include <stdlib.h>
#include <stdlib.h>
/* Link list node */
struct Node
{
    int data;
    struct Node * next;
};
/* function to get the alternate
   nodes of the linked list */
void printAlternateNode(struct Node *head)
{
    int count = 0;
    while (head != NULL)
    {
        // when count is even print the nodes
        if (count % 2 == 0)
            print ("%d", head->data);
```

```
// count the nodes
    count++;

// move on the next node.
    head = head->next;
  }
}

// function to push node at head.
void push(struct Node ** head_ref, int new_data)
{
  struct Node * new_node = (struct Node *) malloc(sizeof(struct
                                                  Node));

  new_node->data = new_data;
  new_node->next = (*head_ref);
  (*head_ref) = new_node;
}

// Driver code.
int main()
{
  /* start with the empty list */
  struct Node * head = NULL;

  /* Use push() function to construct the the below list
          8->23->11->29->12 */

  push(&head, 12);
  push(&head, 29);
  push(&head, 11);
```

```
push (&head, 23);
push (&head, 8);
printAlternateNode(head);
return 0;
}
```

5.(i)

ARRAY	LINKED LIST
1. A data structure consisting of a collection of elements each identified by the array index.	1. A linear collection of data elements whose order is not given by their location in memory.
2. Supports random access, so the programer can directly access an element in the array using the index.	2. Supports sequential access, so the programmer has to sequentially go through each element or node until reaching the required element.
3. Elements are stored in contiguous memory locations.	3. Elements can be stored anywhere in the memory.
4. Programmer has to specify the size of the array at the time of declaring the array.	4. There is no need for specifying the size of a linked list.
5. Memory allocation happens at compile time; it is a static memory allocation.	5. Memory allocation happens at runtime; it is a dynamic memory allocation.

5. Elements are independent of each other.

6. An element or node points to the next node or both next node and previous node.

5. (ii)

```
#include <bits/stdc++.h>
using namespace std;

// Returns maximum possible equal sum of three stacks
// with removal of top elements allowed
int maxSum(int stack1[], int stack2[], int stack3[],
           int n1, int n2, int n3)
{
  int sum1=0, sum2=0, sum3=0;
// Finding the initial sum of stack 1.
  for (int i=0; i<n1; i++)
    sum1 += stack1[i];
// Finding the initial sum of stack 2.
  for (int i=0; i<n2; i++)
    sum2 += stack2[i];
// Finding the initial sum of stack 3.
  for (int i=0; i<n3; i++)
    sum3 += stack3[i];
// As given in question, first element is top
// of stack.
  int top1=0, top2=0, top3=0;
  int ans=0;
  while (1)
```

```
{
  // if sum of all three stack are equal.
  if (sum1 == sum2 && sum2 == sum3)
    return sum1;

  // Finding the stack with maximum sum and
  // removing its top element.
  if (sum1 >= sum2 && sum1 >= sum3)
      sum1 -= stack1[top1++];
  else if (sum2 >= sum3 && sum2 >= sum3)
      sum2 -= stack2[top2++];
  else if (sum3 >= sum2 && sum3 >= sum1)
      sum3 -= stack3[top3++];
  }
}

// Driven program.
int main()
{
  int stack1[] = {2,1,3,5,1};
  int stack2[] = {4,3,2};
  int stack3[] = {2,1,1,1};

  int n1 = sizeof(stack1)/sizeof(stack1[0]);
  int n2 = sizeof(stack2)/sizeof(stack2[0]);
  int n3 = sizeof(stack3)/sizeof(stack3[0]);

  cout << maxSum(stack1, stack2, stack3, n1, n2, n3) << endl;
  return 0;
}
```