18CSC201J
DS LAB EXERCISES
UNIT-2

DATE: 11.12.2021

NAME: ANINDYA SHANKAR DASGUPTA
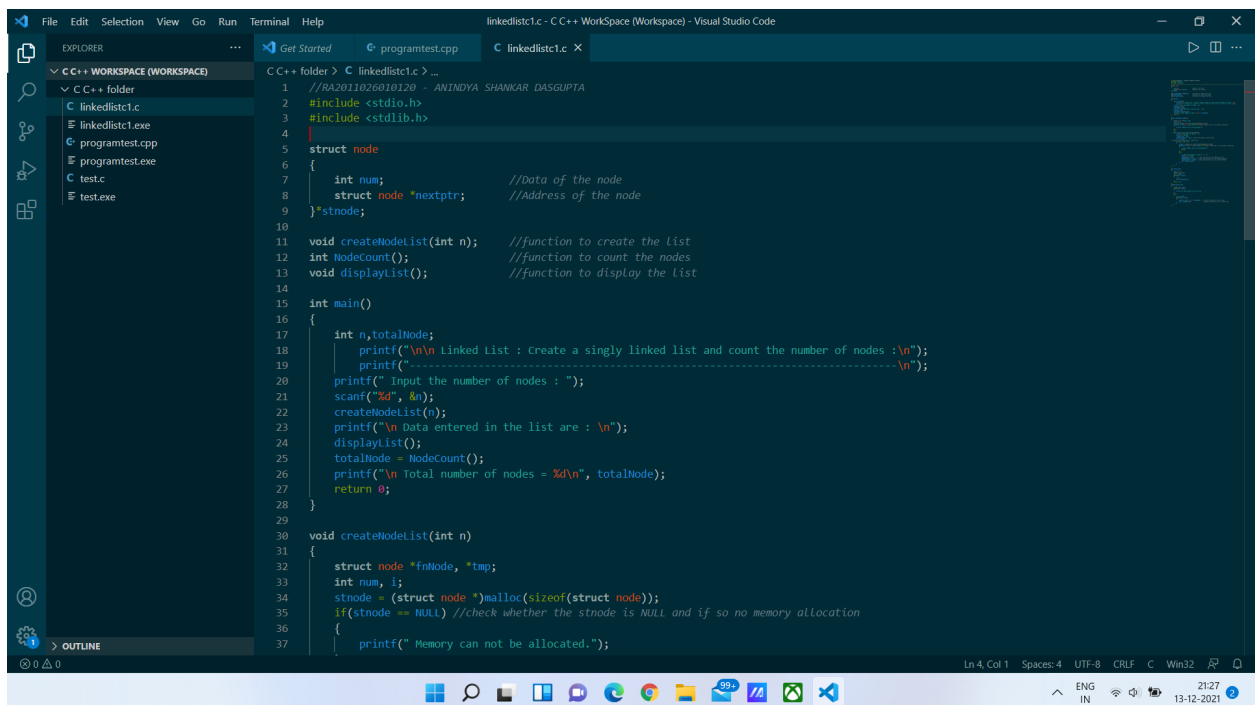REG NO: RA2011026010120
CSE Q1 (AI-ML)

Simple Problems:

<u>Q:</u>

1. (2.) Write a program in C to create a singly linked list of n nodes and count the number of nodes

<u>A:</u>

- <u>Approach:</u>
    1. There are certain ambiguities in the questions, like if the list to be created is empty or has some nodes already. We must run some conditions before inserting to check if the list is empty or not.
    2. If the list is empty, and we create a linked list of 'n' nodes then the number of nodes automatically will be 'n' itself, and there is no need to write a code separately for that purpose.

- <u>Code (with documentation):</u>

```c
37              printf(" Memory can not be allocated.");
38          }
39          else
40          {
41  // reads data for the node through keyboard
42              printf(" Input data for node 1 : ");
43              scanf("%d", &num);
44              stnode-> num = num;
45              stnode-> nextptr = NULL; //Links the address field to NULL
46              tmp = stnode;
47  //Creates n nodes and adds to linked list
48              for(i=2; i<=n; i++)
49              {
50                  fnNode = (struct node *)malloc(sizeof(struct node));
51                  if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation
52                  {
53                      printf(" Memory can not be allocated.");
54                      break;
55                  }
56                  else
57                  {
58                      printf(" Input data for node %d : ", i);
59                      scanf(" %d", &num);
60                      fnNode->num = num;        // links the num field of fnNode with num
61                      fnNode->nextptr = NULL; // links the address field of fnNode with NULL
62                      tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode
63                      tmp = tmp->nextptr;
64                  }
65              }
66          }
67  }
68
69  int NodeCount()
70  {
71      int ctr = 0;
72      struct node *tmp;
73      tmp = stnode;
74      while(tmp != NULL)
```

```c
74      while(tmp != NULL)
75      {
76          ctr++;
77          tmp = tmp->nextptr;
78      }
79      return ctr;
80  }
81  void displayList()
82  {
83      struct node *tmp;
84      if(stnode == NULL)
85      {
86          printf(" No data found in the list.");
87      }
88      else
89      {
90          tmp = stnode;
91          while(tmp != NULL)
92          {
93              printf(" Data = %d\n", tmp->num);   // prints the data of current node
94              tmp = tmp->nextptr;                 // advances the position of current node
95          }
96      }
97  }
```

- Dry-run:
  For sample input/output shown below:

Block-1	Declared struct node (node of list of struct type) Accordingly, declared node variables to be used: stnode , fnnode , tmp They are dynamically allocated during runtime.
Block-2	n = user-input number of nodes = 12

Block-3
Function:
createNodelist
	stnode: head node ; fnnode: created new node ; tmp: previous node in list
	stnode->data = 8; stnode->next = NULL; tmp = stnode
	i=2: fnnode->data = 4; fnnode->next=NULL;
	    newly created node
	    tmp->next=fnnode;
	    so, tmp : previous node in list is linked to newly created node.
	    tmp = temp->next;
	    newly linked node becomes previous node for next node to be inserted.
	i=3: fnnode->data = 9; same procedure followed
	.
	
	.
	i=12: fnnode->data = 16; fnnode->NULL;
	    tmp->next = fnnode;
	    tmp = tmp->next;
	    so, tmp is now last node, has data = 16 and points to NULL
	    Linked list is now: 8->4->9->13->0->44->6->125->2->-5->22->16

Block-4 Function to print list	tmp: used as current node checked if tmp = NULL ,i.e if list is empty List non-empty tmp = stnode : first node/head Then, tmp is traversed through the list till it reaches end (12 times) Each time, current node value is printed: tmp = head: data = 8 is printed and next node is passed to tmp : tmp = tmp->next tmp = node2: data = 4 ….. tmp = node12: data = 16 is printed. tmp -> next then points to NULL (last node) So, loop is broken
Block-5	Using the same traversal method as display function, A counter variable is incremented till tmp (current node used for traversal) reaches end of linked list, to find out number of nodes. Then it is also displayed. Here counter-variable = ctr = 12 is printed.

- **Input/Output:**

Input-

Output-

- Result:

    1. Time-complexity:
       For non-looped statements, each is considered to take up 1 unit (as they are executed only once).
       Let sum of these be = cnst (constant value)
       There are 3 loops:
       for loop: 1 to n : n times
       while loop: n times
       while loop: n times
       Total time-complexity = cnst + 3*n
       Asymptotic upper-bound time-complexity = O(cnst+3*n) = O(n)

    2. Space-complexity:
       Each node = 8 bytes.
       n nodes = 8*n bytes.
       1 extra node variable = 8 bytes (fnnode)
       int n,ctr,num,i = 4*4 = 16 bytes
       Total space complexity = 16+8+8*n bytes = 24+8*n bytes
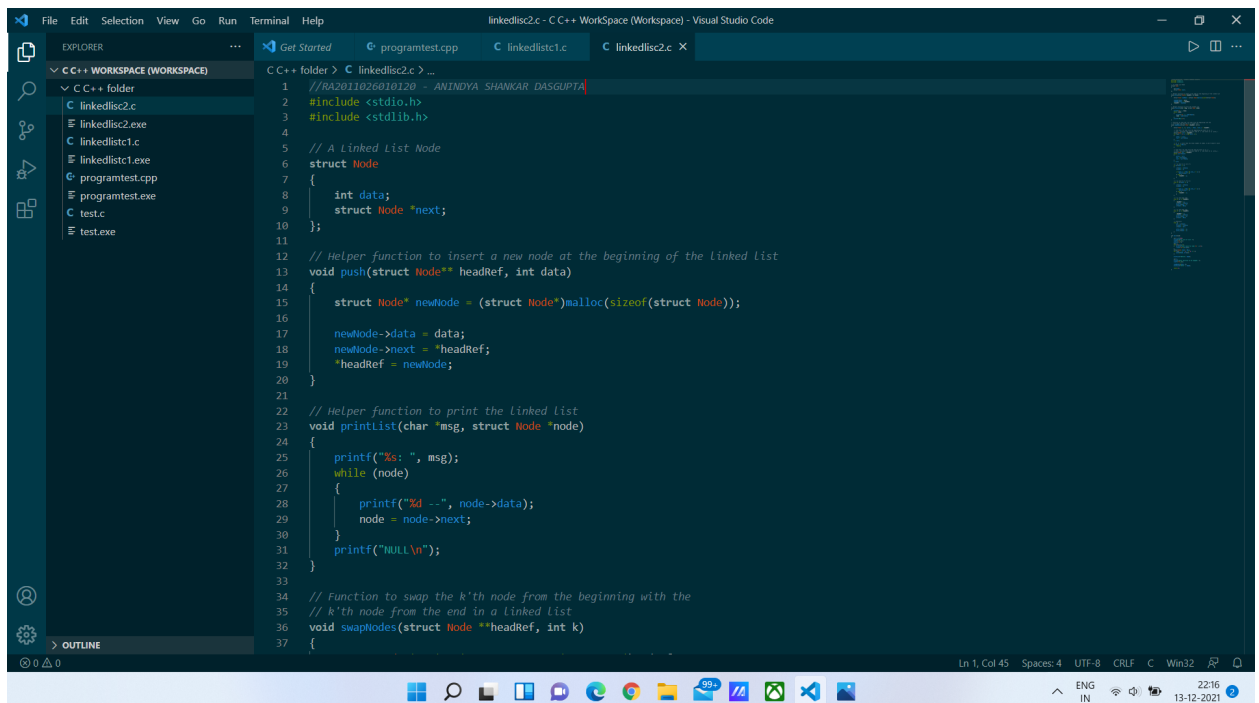   Asymptotic upper-bound space-complexity = O(24+8*n) = O(n)

RA2011026010120

Scenario based or Difficult Problems:

Q:
1. (16.) Given a singly linked list, swap kth node from beginning with kth node from end. Swapping of data is not allowed, only pointers should be changed.

A:
- Approach:
    1. Not given if the list is empty or not. We must check explicitly.
    2. Value of k entered may exceed the size of the list. We should print an error message for that scenario.
    3. The list is supposed to be given but it is not given in question and we have to take user-input.
    4. Value of k entered may correspond to the exact middle node of the list, in which case the same list is printed as output without changes.

- Code (with documentation):

```c
struct Node *x, *y, *prev_x = NULL, *prev_y = *headRef;

// Find the k'th node from the beginning and store it in `x`.
// Also, calculate the previous node of `x` and store it in `prev_x`.
struct Node* curr = *headRef;
for (int i = 1; i < k && curr; i++)
{
    prev_x = curr;
    curr = curr->next;
}
x = curr;

// If `k` is more than the total number of nodes, X and Y doesn't exist
if (curr == NULL) {
    return;
}

// Find the k'th node from the end and store it in `y`.
// Also, calculate the previous node of `y` and store it in `prev_y`.
struct Node* ptr = *headRef;
while (curr->next)
{
    prev_y = ptr;
    ptr = ptr->next;
    curr = curr->next;
}
y = ptr;

// Y is next to X (X -> Y)
if (x->next == y)
{
    x->next = y->next;
    y->next = x;

    if (prev_x != NULL && prev_x != x) {
        prev_x->next = y;
    } else {
        *headRef = y;
    }
}

// X is next to Y (Y -> X)
else if (y->next == x)
{
    y->next = x->next;
    x->next = y;

    if (prev_y != NULL && prev_y != y) {
        prev_y->next = x;
    } else {
        *headRef = x;
    }
}

// X is the head node
else if (x == *headRef)
{
    *headRef = y;
    y->next = x->next;
    prev_y->next = x;
    x->next = NULL;
}

// Y is the head node
else if (y == *headRef)
{
    *headRef = x;
    x->next = y->next;
    prev_x->next = y;
    y->next = NULL;
}

// Otherwise
```

- Dry-run:

Block-1	Initializing nodes and other variables newNode; headref; node
Block-2 Inserting in list	First, size of list = n = 5 Insertion as follows: i=-; list: empty : (headref ->NULL) i=0: list: headref->1->NULL i=1: list: headref->2->NULL . . i=4: list: headref->1->2->3->4->5->NULL
Block-3 Swapping	k = 2 traversing using curr variable from first node to kth node curr->data = 2; curr->next = 3 (node-3) Store this value in x: x = 2 Similarly, traverse from last node to kth last node curr->data = 4 Store this value in y: y=4 Check if adjacent nodes Then swap x with y then pass these values to the previously kth nodes Now list: headref->1->4->3->2->5->NULL
Block-4	Printing new list

- Input/Output:

Input-

```
Enter size of list: 5

Enter value for node 1: 1
Enter value for node 2: 2
Enter value for node 3: 3
Enter value for node 4: 4
Enter value for node 5: 5
```

Output-

```
Before: 1 --2 --3 --4 --5 --NULL
Enter position to be swapped: 2
After : 1 --4 --3 --2 --5 --NULL
PS C:\Users\Anindya\Documents\VSCode\C C++ folder>
```

- Result:

1. Time complexity:
   For each statement complexity = 1
   Let there be cnst no. of such statements: complexity = cnst
   Loops :
   for loop : n times (enter array values)
   for loop : n times (push array values to linked list)
   while loop : n times (traversing through list to print) x2
   while loop: k times (finding kth element)   x2
   Total time-complexity = $cnst + 2*n + 2*n + 2*k$
   Asymptotic upper-bound time-complexity = $O(cnst + 4*n + 2*k) = O(n)$

2. Space complexity:
   No. of nodes = n : Each node = 8 bytes : 8*n bytes
   2 extra nodes for traversing : 16 bytes
   2 extra nodes for swapping : 16 bytes
   int n,k,arr[50] : $4*3 = 12$ bytes
   Total space-complexity = $8*n + 44$
   Asymptotic upper-bound space-complexity = $O(8*n + 44) = O(n)$