Question 1 (30 marks)

The specification of a **pointer-based List ADT** is given in Figure 1:

|  |
| --- |
| #include<iostream>  using namespace std;  typedef char ItemType;  class List  {  private:  struct Node  {  ItemType item; // item  Node\* next; // pointer pointing to next node  };  Node\* front; // pointer pointing to front node  public:  //constructor  **List();**    // add a new item at a specified position in the list  // 0 <= index < size  **bool add(int index, ItemType item);**    // remove an item at a specified position in the list  // 0 <= index < size  **void remove(int index);**    // get an item at a specified position in the list  // 0 <= index < size  **ItemType get(int index);**  // returns the number of occurrences of item in the list  **int count(ItemType item);**  // recursive function that  // returns the number of occurrences of item in the list  **int countR(ItemType item);**  // reverses the list  **void reverse();**  // recursive function that reverses the list  **void reverseR();**  // prints all the items in the list  **void display();**  }; |

Figure 1: Specification of a pointer-based List ADT

(a) Write the **count(ItemType item)** function that returns the number of occurrences of the item in the list.

|  |
| --- |
| int count(ItemType item)  {  Node \*temp = front;  int n = 0;  while (temp != NULL)  {  if (temp->item == item)  n++;  temp = temp->next;  }  return n;  } |

(b) Write the **reverse()** function that reverses the items in the list.

|  |
| --- |
| void reverse()  {  Node\* newList;  Node\* temp = front; // temp is used to traverse thru’ the list  while (temp != NULL)  {  newList.add(temp->item, 0); // add each item to position 0  temp = temp->next;  }  } |

(c) Write the recursive **countR(ItemType item)** function that returns the number of occurrences of the item in the list.

int countR(ItemType item)

{

return countR(front, item)

}

int countR(Node\* temp, ItemType item)

{

if (temp == NULL) // base case

return 0;

else // recursive step

if (temp->item == item)

return 1 + countR(temp->next, item);

else

return counrR(temp->next, item);

}

(d) Write the recursive **reverseR()** function that reverses the items in the list.

void reverseR()

{

front = reverseR(front);

}

Node\* reverseR(Node\* temp)

{

if (temp->next == NULL) // base case – it is the last item

return temp; // just return it

else

{ // recursive step

Node\* newList = reverseR(temp->next); // reverse list from next item

newList.add(temp->item); // add current item to the end of newList

return newList; // this is now the reversed list

}

}