Question2.cpp

```cpp
// <----Lab 03 - Singly Linked List---->

// Q2.Solve the following problem using a Singly Linked List. Given a singly linked list of
//    characters, write a function to make word out of given letters in the list. Test Case:
//    Input:C->S->A->R->B->B->E->L->NULL,
//    Output:S->C->R->A->B->B->L->E->NULL

#include<iostream>
using namespace std;

string dictionary[]={"hello","rehan","maha","ali","class","university","section","data","structure","algorithm","sir",
                     "madam","word","sentence","biryani","cold","drink","village","book","library","bookshelf","campus",
                     "charger","phone","cable","computer","laptop","television",
                     "money","alot","large","amount","small","vocabulary","pneumonoultramicroscopicsilicovolcaniosis","goodbye","CGPA","GPA"};


string unsorted[38];

void sortdictionary(){   //Sorting due to anagram detection logic

    cout<<"DICTIONARY:  Word => SortedWord"<<endl;
    cout<<"_____"<<endl;
    for(int i=0;i<38;i++){
        unsorted[i]=dictionary[i];
    }

    for(int i=0;i<38;i++){ // for array
        //for bubble sort
        int count=0;
        while(dictionary[i][count]!='\0'){
            count++;
        }

        for(int j=0;j<count-1;j++){
            for(int k=0;k<count-1-j;k++){
                if(dictionary[i][k]>dictionary[i][k+1]){
                    char temp = dictionary[i][k];
                    dictionary[i][k] = dictionary[i][k+1];
                    dictionary[i][k+1] = temp;
                }
            }
        }
    }

    for(int i=0;i<38;i++){
        cout<<unsorted[i]<<" => "<<dictionary[i]<<endl;   //printing to check array.
    }
    cout<<endl<<endl<<endl;

}

```

```cpp

class node {
    public:
        char data;
        node* next;
        node(char value) {
            data=value;
            next=NULL;
        }
        node(char value,node* nxt) {
            data=value;
            next=nxt;
        }
};


class SLL {  //SLL = Singly Linked List
    node* head=NULL;
    public:
        void insertAttail(char value) {
            node* n=new node(value);
            if(head==NULL) {  //Check if Linked List Empty.
                head=n;
                return;
            }
            node* temp=head;
            while(temp->next!=NULL) {
                temp=temp->next;
            }
            temp->next=n;
        }
        void insertAtPos(int posvalue,char value) {
            int count=0;
            node* temp=head;
            while(temp->next!=NULL&&count<posvalue-1) {
                temp=temp->next;
                count++;
            }
            node* n=new node(value,temp->next);
            temp->next=n;
        }
        void display() {
            node* temp=head;
            cout<<"[HEAD] ";
            while(temp!=NULL) {
                cout<<temp->data<<"| "<<temp->next<<" -> ";
                temp=temp->next;
            }
            cout<<"NULL [TAIL]"<<endl;
        }
        void insertAthead(char value) {
            node* n=new node(value);
            n->next=head;
            head=n;
        }
        void deletion(char value) {
```

```
108             if(head==NULL) {
109                 return;
110             }
111             node* temp=head;
112             while(temp->next->data!=value ) {
113                 temp=temp->next;
114             }
115             node* todelete=temp->next;
116             temp->next=temp->next->next;
117 
118             delete todelete;
119         }
120         void deleteAthead() {
121             if(head==NULL) {
122                 return;
123             }
124             node* todelete=head;
125             head=head->next;
126             delete todelete;
127         }
128         void deleteAtPos(int posvalue) {
129             if(head==NULL) {
130                 return;
131             }
132             int count=0;
133             node* temp=head;
134             while(temp->next!=NULL && count<posvalue-1) {
135                 temp=temp->next;
136             }
137             node* todelete=temp->next;
138             temp->next=temp->next->next;
139 
140             delete todelete;
141 
142         }
143         void deleteAttail() {
144             if(head==NULL) { //If linked list empty.
145                 return;
146             }
147             node* temp=head;
148             if(head->next==NULL) { //If linked list has 1 item only.
149                 head=NULL;
150                 delete temp;
151             }
152             while(temp->next->next!=NULL) {
153                 temp=temp->next;
154             }
155             delete temp->next;
156             temp->next=NULL;
157         }
158 
159         string sortlist(){ //Sorting by data not value
160             node* temp=head;
161             while(temp->next!=NULL){
162                 node* temp2=head;
163                 while(temp2->next!=NULL){
```

```
                    if(temp2->data>temp2->next->data){
                        char tempchar = temp2->data;
                        temp2->data = temp2->next->data;
                        temp2->next->data = tempchar;
                    }
                    temp2=temp2->next;
                }
                temp=temp->next;
            }

            string check="";
            temp=head;
            while(temp->next!=NULL){
                check+=temp->data;
                temp=temp->next;
            }
            check+=temp->data;
            return check;
        }

        void reverse() {
            node* prev=NULL;
            node* after=NULL;
            while(head!=NULL) {
                after=head->next;
                head->next=prev;
                prev=head;
                head=after;
            }
            head=prev;
        }

        void FormWord() {

        string sortedWord = sortlist();
            for (int i = 0; i < 38; i++) {
                if (sortedWord == dictionary[i]) {
                    cout << "Found matching word in dictionary: " << unsorted[i] << endl;
                    formWord(unsorted[i]);
                    return;
                }
            }
            cout << "No matching word found in dictionary." << endl;
        }

    void formWord(string targetWord) {
        node* newHead = NULL;

        for (char ch : targetWord) {   //For each character in targetword  "Computer"
            node* temp = head;
            node* prev = NULL;
            while (temp != NULL && temp->data != ch) { //finding a character that goes into
the position
                prev = temp;
                temp = temp->next;
            }
```

```
219
220             if (temp == NULL) {   //if char not found then invalid
221                 cout << "Error: Cannot form the word." << endl;
222                 return;
223             }
224
225             if (prev == NULL) { //checks if character is in 1st position or not
226                 head = temp->next;
227             } else {
228                 prev->next = temp->next;  //link skips temp
229             }
230             temp->next = newHead;  //placed at head
231             newHead = temp;
232         }
233         head = newHead;  //Generates word but in reverse due to head logic
234         reverse();   //fixes the reversed word generated by formword.
235     }
236 };
237
238
239 int main(){
240     sortdictionary();
241     cout<<"-----------------------------------------"<<endl;
242     cout<<"DICTIONARY above this point\n";
243     cout<<"-----------------------------------------\n"<<endl;
244     SLL word;
245     char input=' ';
246     while(input!='@'){
247         cout<<"Enter character to put in linked list [Enter '@' to end input phase] \n(CASE
    SENSITIVE! use lower case): ";
248         cin>>input;
249         if(input!='@'){
250             word.insertAttail(input);
251         }
252     }
253     word.display();
254     cout<<endl<<"Checking if word can be formed.\n\n"<<endl;
255     cout<<endl;
256     word.FormWord();
257     cout<<endl;
258     word.display();
259
260 }
261
262
263
```