Question2.cpp

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

// 2. 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","ahmed","aziz","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;
        }
```

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

```
164                 while(temp2->next!=NULL){
165                     if(temp2->data>temp2->next->data){
166                         char tempchar = temp2->data;
167                         temp2->data = temp2->next->data;
168                         temp2->next->data = tempchar;
169                     }
170                     temp2=temp2->next;
171                 }
172                 temp=temp->next;
173             }
174 
175             string check="";
176             temp=head;
177             while(temp->next!=NULL){
178                 check+=temp->data;
179                 temp=temp->next;
180             }
181             check+=temp->data;
182             return check;
183         }
184 
185         void reverse() {
186             node* prev=NULL;
187             node* after=NULL;
188             while(head!=NULL) {
189                 after=head->next;
190                 head->next=prev;
191                 prev=head;
192                 head=after;
193             }
194             head=prev;
195         }
196 
197         void FormWord() {
198 
199         string sortedWord = sortlist();
200             for (int i = 0; i < 38; i++) {
201                 if (sortedWord == dictionary[i]) {
202                     cout << "Found matching word in dictionary: " << unsorted[i] << endl;
203                     formWord(unsorted[i]);
204                     return;
205                 }
206             }
207             cout << "No matching word found in dictionary." << endl;
208         }
209 
210     void formWord(string targetWord) {
211         node* newHead = NULL;
212 
213         for (char ch : targetWord) {   //For each character in targetword  "Computer"
214             node* temp = head;
215             node* prev = NULL;
216             while (temp != NULL && temp->data != ch) { //finding a character that goes into the
    position
217                 prev = temp;
218                 temp = temp->next;
```

```
            }

            if (temp == NULL) {   //if char not found then invalid
                cout << "Error: Cannot form the word." << endl;
                return;
            }

            if (prev == NULL) { //checks if character is in 1st position or not
                head = temp->next;
            } else {
                prev->next = temp->next;  //link skips temp
            }
            temp->next = newHead;  //placed at head
            newHead = temp;
        }
        head = newHead;  //Generates word but in reverse due to head logic
        reverse();   //fixes the reversed word generated by formword.
    }
};


int main(){
    sortdictionary();
    cout<<"----------------------------------------"<<endl;
    cout<<"DICTIONARY above this point\n";
    cout<<"----------------------------------------\n"<<endl;
    SLL word;
    char input=' ';
    while(input!='@'){
        cout<<"Enter character to put in linked list [Enter '@' to end input phase] \n(CASE
SENSITIVE! use lower case): ";
        cin>>input;
        if(input!='@'){
            word.insertAttail(input);
        }
    }
    word.display();
    cout<<endl<<"Checking if word can be formed.\n\n"<<endl;
    cout<<endl;
    word.FormWord();
    cout<<endl;
    word.display();

}
```