

Question1.cpp

```
1 // <----Lab 03 - Singly Linked List---->
2
3 // 1. Implement a singly linked list class with the following functions:
4 // a) Insert a node at head
5 // b) Insert a node at tail/end/back
6 // c) Insert a node at any position
7 // d) Delete a node by value
8 // e) Delete head
9 // f) Delete tail
10 // g) Delete a node at any position.
11
12 #include<iostream>
13 using namespace std;
14
15 class node {
16     public:
17         int data;
18         node* next;
19         node(int value) {
20             data=value;
21             next=NULL;
22         }
23         node(int value,node* nxt) {
24             data=value;
25             next=nxt;
26         }
27 };
28
29
30 class SLL { //SLL = Singly Linked List
31     node* head=NULL;
32     public:
33         void insertAttail(int value) {
34             node* n=new node(value);
35             if(head==NULL) { //Check if Linked List Empty.
36                 head=n;
37                 return;
38             }
39             node* temp=head;
40             while(temp->next!=NULL) {
41                 temp=temp->next;
42             }
43             temp->next=n;
44         }
45         void insertAtPos(int posvalue,int value) {
46             int count=0;
47             node* temp=head;
48             while(temp->next!=NULL&&count<posvalue-1) {
49                 temp=temp->next;
50                 count++;
51             }
52             node* n=new node(value,temp->next);
53             temp->next=n;
```

```
54     }
55     void display() {
56         node* temp=head;
57         cout<<"[HEAD] ";
58         while(temp!=NULL) {
59             cout<<temp->data<<"| " <<temp->next<<" -> ";
60             temp=temp->next;
61         }
62         cout<<"NULL [TAIL]"<<endl;
63     }
64     void insertAthead(int value) {
65         node* n=new node(value);
66         n->next=head;
67         head=n;
68     }
69     void deletion(int value) {
70         if(head==NULL) {
71             return;
72         }
73         node* temp=head;
74         if(head==NULL){
75             cout<<"Empty Linked List, returning\n";
76             return;
77         }
78         if(head->data==value){
79             head=head->next;
80             return;
81         }
82         while(temp->next->data!=value ) {
83             if(temp->next->next==NULL){
84                 cout<<"Value not found... Returning\n";
85                 return;
86             }
87             temp=temp->next;
88         }
89         node* todelete=temp->next;
90         temp->next=temp->next->next;
91
92         delete todelete;
93     }
94     void deleteAthead() {
95         if(head==NULL) {
96             return;
97         }
98         node* todelete=head;
99         head=head->next;
100         delete todelete;
101     }
102     void deleteAtPos(int posvalue) {
103         if(head==NULL) {
104             return;
105         }
106         int count=0;
107         node* temp=head;
108         while(temp->next!=NULL && count<posvalue-1) {
109             temp=temp->next;
```

```
110         count++;
111     }
112     node* todelete=temp->next;
113     temp->next=temp->next->next;
114     delete todelete;
115 }
116 void deleteAttail() {
117     if(head==NULL) { //If linked list empty.
118         return;
119     }
120     node* temp=head;
121     if(head->next==NULL) { //If linked list has 1 item only.
122         head=NULL;
123         delete temp;
124         return;
125     }
126     while(temp->next->next!=NULL) {
127         temp=temp->next;
128     }
129     delete temp->next;
130     temp->next=NULL;
131 }
132
133 };
134
135
136 int main() {
137     SLL list;
138     float input=0;
139     int value;
140     while(input!=0.5) {
141         cout<<"-----\n";
142         cout<<"CURRENT LINKED LIST:\n";
143         list.display();
144         cout<<"-----\n";
145         cout<<"What would you like to do with the linked list?\n";
146         cout<<"1. Insert\t2. Delete\nEnter 0.5 to Exit\n[Anything else will default to Delete]\n";
147         cin>>input;
148         if(input==1) {
149             cout<<"Enter Value to insert: ";
150             cin>>value;
151             cout<<"Where to Insert in Linked List?\n";
152             cout<<"1. At head\t2. At tail\t3. At specified Position\n[Any other value will default  
to Insertion at Head]\n";
153             cin>>input;
154             if(input == 2){
155                 list.insertAttail(value);
156             }
157             else if(input == 3){
158                 int pos;
159                 cout<<"Enter the Position to insert into: ";
160                 cin>>pos;
161                 list.insertAtPos(pos,value);
162             }
163             else{
164                 list.insertAthead(value);
```

```
165         }
166
167     }
168     else if(input==0.5){
169         break;
170     }
171     else{
172         cout<<"Where to Delete from Linked List?\n";
173         cout<<"1. At head\t2. At tail\t3. At specified Position\t 4. Delete a specific
Value\n[Any other value will default to Deletion from Head]\n";
174         cin>>input;
175         if(input == 2){
176             list.deleteAttail();
177         }
178         else if(input == 3){
179             int pos;
180             cout<<"Enter the Position to Delete from: ";
181             cin>>pos;
182             list.deleteAtPos(pos);
183         }
184         else if(input == 4){
185             int pos;
186             cout<<"Enter the Value to Delete: ";
187             cin>>value;
188             list.deletion(value);
189         }
190         else{
191             list.deleteAthead();
192         }
193     }
194 }
195 }
196 }
```

Question2.cpp

```

1 // <---Lab 03 - Singly Linked List--->
2
3 // 2. Solve the following problem using a Singly Linked List. Given a singly linked list of
4 // characters, write a function to make word out of given letters in the list. Test Case:
5 // Input:C->S->A->R->B->B->E->L->NULL,
6 // Output:S->C->R->A->B->B->L->E->NULL
7
8
9 #include<iostream>
10 using namespace std;
11
12 string dictionary[]={"hello","ahmed","aziz","ali","class","university","section","data","
13 structure","algorithm","sir",
14 "madam","word","sentence","biryani","cold","drink","village","book","library",
15 "bookshelf","campus",
16 "charger","phone","cable","computer","laptop","television",
17 "money","alot","large","amount","small","vocabulary","
18 pneumonoultramicroscopicsilicovolcaniosis","goodbye","CGPA","GPA"};
19
20 string unsorted[38];
21
22 void sortdictionary(){ //Sorting due to anagram detection logic
23
24     cout<<"DICTIONARY: Word => SortedWord"<<endl;
25     cout<<"_____ "<<endl;
26     for(int i=0;i<38;i++){
27         unsorted[i]=dictionary[i];
28     }
29
30     for(int i=0;i<38;i++){ // for array
31         //for bubble sort
32         int count=0;
33         while(dictionary[i][count]!='\0'){
34             count++;
35         }
36
37         for(int j=0;j<count-1;j++){
38             for(int k=0;k<count-1-j;k++){
39                 if(dictionary[i][k]>dictionary[i][k+1]){
40                     char temp = dictionary[i][k];
41                     dictionary[i][k] = dictionary[i][k+1];
42                     dictionary[i][k+1] = temp;
43                 }
44             }
45         }
46
47         for(int i=0;i<38;i++){
48             cout<<unsorted[i]<<" => "<<dictionary[i]<<endl; //printing to check array.
49         }
50     }
51 }

```

```
52
53
54 class node {
55     public:
56         char data;
57         node* next;
58         node(char value) {
59             data=value;
60             next=NULL;
61         }
62         node(char value,node* nxt) {
63             data=value;
64             next=nxt;
65         }
66 };
67
68
69 class SLL { //SLL = Singly Linked List
70     node* head=NULL;
71     public:
72         void insertAttail(char value) {
73             node* n=new node(value);
74             if(head==NULL) { //Check if Linked List Empty.
75                 head=n;
76                 return;
77             }
78             node* temp=head;
79             while(temp->next!=NULL) {
80                 temp=temp->next;
81             }
82             temp->next=n;
83         }
84         void insertAtPos(int posvalue,char value) {
85             int count=0;
86             node* temp=head;
87             while(temp->next!=NULL&&count<posvalue-1) {
88                 temp=temp->next;
89                 count++;
90             }
91             node* n=new node(value,temp->next);
92             temp->next=n;
93         }
94         void display() {
95             node* temp=head;
96             cout<<"[HEAD] ";
97             while(temp!=NULL) {
98                 cout<<temp->data<<" | "<<temp->next<<" -> ";
99                 temp=temp->next;
100             }
101             cout<<"NULL [TAIL]"<<endl;
102         }
103         void insertAthead(char value) {
104             node* n=new node(value);
105             n->next=head;
106             head=n;
107         }
108     }
```

```
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;
```



```
219     }
220
221     if (temp == NULL) { //if char not found then invalid
222         cout << "Error: Cannot form the word." << endl;
223         return;
224     }
225
226     if (prev == NULL) { //checks if character is in 1st position or not
227         head = temp->next;
228     } else {
229         prev->next = temp->next; //link skips temp
230     }
231     temp->next = newHead; //placed at head
232     newHead = temp;
233 }
234 head = newHead; //Generates word but in reverse due to head logic
235 reverse(); //fixes the reversed word generated by formword.
236 }
237 };
238
239
240 int main(){
241     sortdictionary();
242     cout<<"-----"<<endl;
243     cout<<"DICTIONARY above this point\n";
244     cout<<"-----\n"<<endl;
245     SLL word;
246     char input=' ';
247     while(input!='@'){
248         cout<<"Enter character to put in linked list [Enter '@' to end input phase] \n(CASE
SENSITIVE! use lower case): ";
249         cin>>input;
250         if(input!='@'){
251             word.insertAttail(input);
252         }
253     }
254     word.display();
255     cout<<endl<<"Checking if word can be formed.\n\n"<<endl;
256     cout<<endl;
257     word.FormWord();
258     cout<<endl;
259     word.display();
260
261 }
262
263
264
```

Question3.cpp

```
1 // <---Lab 03 - Singly Linked List--->
2
3 // 3. Use the class of SLL created by you during the lab task 1. Do the following:
4 // a) Reverse the linked list
5 // b) Sort the contents of linked list
6 // c) Find the duplicates in the linked list
7
8 #include<iostream>
9 using namespace std;
10
11 class node {
12     public:
13         int data;
14         node* next;
15         node(int value) {
16             data=value;
17             next=NULL;
18         }
19         node(int value,node* nxt) {
20             data=value;
21             next=nxt;
22         }
23 };
24
25
26 class SLL { //SLL = Singly Linked List made in task 1
27     node* head=NULL;
28     public:
29         void insertAttail(int value) {
30             node* n=new node(value);
31             if(head==NULL) { //Check if Linked List Empty.
32                 head=n;
33                 return;
34             }
35             node* temp=head;
36             while(temp->next!=NULL) {
37                 temp=temp->next;
38             }
39             temp->next=n;
40         }
41         void insertAtPos(int posvalue,int value) {
42             int count=0;
43             node* temp=head;
44             while(temp->next!=NULL&&count<posvalue-1) {
45                 temp=temp->next;
46                 count++;
47             }
48             node* n=new node(value,temp->next);
49             temp->next=n;
50         }
51         void display() {
52             node* temp=head;
53             cout<<"[HEAD] ";
```

```
54     while(temp!=NULL) {
55         cout<<temp->data<<" | "<<temp->next<<" -> ";
56         temp=temp->next;
57     }
58     cout<<"NULL [TAIL]"<<endl;
59 }
60 void insertAthead(int value) {
61     node* n=new node(value);
62     n->next=head;
63     head=n;
64 }
65 void deletion(int value) {
66     if(head==NULL) {
67         return;
68     }
69     node* temp=head;
70     while(temp->next->data!=value ) {
71         temp=temp->next;
72     }
73     node* todelete=temp->next;
74     temp->next=temp->next->next;
75
76     delete todelete;
77 }
78 void deleteAthead() {
79     if(head==NULL) {
80         return;
81     }
82     node* todelete=head;
83     head=head->next;
84     delete todelete;
85 }
86 void deleteAtPos(int posvalue) {
87     if(head==NULL) {
88         return;
89     }
90     int count=0;
91     node* temp=head;
92     while(temp->next!=NULL && count<posvalue-1) {
93         temp=temp->next;
94     }
95     node* todelete=temp->next;
96     temp->next=temp->next->next;
97
98     delete todelete;
99
100 }
101 void deleteAttail() {
102     if(head==NULL) { //If linked list empty.
103         return;
104     }
105     node* temp=head;
106     if(head->next==NULL) { //If linked list has 1 item only.
107         head=NULL;
108         delete temp;
109     }
```

```
110     while(temp->next->next!=NULL) {
111         temp=temp->next;
112     }
113     delete temp->next;
114     temp->next=NULL;
115 }
116
117 void sortlist() { //sorting by swapping values
118     node* temp=head;
119     while(temp->next!=NULL) {
120         node* temp2=head;
121         while(temp2->next!=NULL) {
122             if(temp2->data>temp2->next->data) {
123                 char tempchar = temp2->data;
124                 temp2->data = temp2->next->data;
125                 temp2->next->data = tempchar;
126             }
127             temp2=temp2->next;
128         }
129         temp=temp->next;
130     }
131 }
132
133 void reverse() {
134     node* prev=NULL;
135     node* after=NULL;
136     while(head!=NULL) {
137         after=head->next;
138         head->next=prev;
139         prev=head;
140         head=after;
141     }
142     head=prev;
143 }
144
145 void duplicates() {
146     int* items = new int[100]; // Assuming a maximum of 100 unique items
147     int* itemcount = new int[100];
148     int size = 0;
149
150     node* temp = head;
151
152     while(temp != NULL) {
153         bool duplicate = false;
154         for(int i = 0; i < size; i++) {
155             if(temp->data == items[i]) {
156                 duplicate = true;
157                 itemcount[i]++;
158                 break;
159             }
160         }
161         if(!duplicate) {
162             items[size] = temp->data;
163             itemcount[size] = 1;
164             size++;
165         }
166     }
167 }
```

```
166         temp = temp->next;
167     }
168     cout<<"Unique Items: "<<size<<endl;
169     for(int i = 0; i < size; i++) {
170         cout<<"Data: "<<items[i]<<" , Count : "<<itemcount[i]<<endl;
171     }
172
173     delete[] items;
174     delete[] itemcount;
175
176 }
177
178 };
179
180 int main() {
181     SLL list;
182     float input=0;
183     while(input!=0.5) {
184         cout<<"Enter integer to put in linked list [Enter 0.5 to end input phase]: ";
185         cin>>input;
186         if(input!=0.5) {
187             list.insertAthead((int)input);
188         }
189     }
190     cout<<"\n\n";
191     list.display();
192     char option;
193     cout<<"\n\nDo you want to... \n'R'\tReverse the Linked List.\n'S'\tSort the Linked List.\n'
D'\tFind Duplicates in Linked List?\n";
194     cin>>option;
195     if(option=='R' || option=='r'){
196         cout<<"Reversing the Linked List."<<endl;
197         list.sortlist();
198         list.display();
199     }
200     else if(option=='S' || option=='s'){
201         cout<<"Sorting the Linked List."<<endl;
202         list.sortlist();
203         list.display();
204     }
205     else if(option=='D' || option=='d'){
206         cout<<"Finding Duplicates: "<<endl;
207         list.duplicates();
208     }
209     else{
210         cout<<"Invalid Input. defaulting to Sort."<<endl;
211         list.sortlist();
212         list.display();
213     }
214 }
```