

PRN: 2020BTEIT00041

NAME: Om Vivek Gcharge

Q.) Doubly Linked List:

CODE:

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  class Node{
5  public:
6
7      int data;
8      Node* next;
9      Node* prev;
10
11     Node(int data){
12         this->data = data;
13         this->next = NULL;
14         this->prev = NULL;
15     }
16 };
17
18
19 class Head{
20 public:
21
22     int count;
23     Node* first;
24     Node* last;
25
26     Head(int count, Node* f, Node* l){
27         this->count = count;
28         this->first = f;
29         this->last = l;
30     }
31 };
32
33 /*
34  insertAtHead function
35 */
36 void insertAtHead(Head* h, int data){
37
38     Node* new_node = new Node(data);
39
40     if(h->count == 0){
41         h->first = new_node;
42         h->last = new_node;
43         h->count++;
44     }
45     else{
46         new_node->next = h->first;
47         h->first->prev = new_node;
48
49         h->first = new_node;
50         h->count++;
51     }
52 }
53
54 /*
55  InsertAtTail function
56 */
57 void insertAtTail(Head* h, int data){
58
59     if(h->count == 0){
60         insertAtHead(h, data);
61         return;
62     }
63     else{
64         Node* new_node = new Node(data);
65
66         h->last->next = new_node;
67         new_node->prev = h->last;
68         h->count++;
69         return;
70     }
71 }
72
73 void insertAfter(Head* h, int data, int location){
74
75     if(location == 0){
76         insertAtHead(h, data);
```

```

77         return;
78     }
79     else if(location == h->count){
80         insertAtTail(h, data);
81         return;
82     }
83     else{
84         Node* p = h->first;
85         Node* new_node = new Node(data);
86
87         for(int i=1; i<h->count; i++){
88             if(i == location){
89                 new_node->next = p->next;
90                 new_node->prev = p;
91
92                 p->next = new_node;
93                 new_node->next->prev = new_node;
94                 h->count++;
95
96                 return;
97             }
98
99             p = p->next;
100         }
101     }
102 }
103
104 void delNode_byVal(Head* h, int val){
105
106     if(h->count==0){
107         cout<<"List is empty, can't delete."<<"\n";
108         return;
109     }
110
111     Node* p = h->first;
112
113     // To Del head
114     if(p->data==val){
115         h->first = p->next;
116         p->next = NULL;
117         free(p);
118         h->count--;
119
120         return;
121     }
122
123     Node* q;
124     for(int i=0; i<h->count; i++){
125
126         if(p->data==val){
127             q->next = p->next;
128             p->next = NULL;
129             free(p);
130             h->count--;
131
132             return;
133         }
134
135         q = p;
136         p = p->next;
137     }
138 }
139
140 void delNodeAt(Head* h, int location){
141
142     if(h->count==0){
143         cout<<"List is empty, can't delete."<<"\n";
144         return;
145     }
146
147     Node* p = h->first;
148
149     // To Del head
150     if(location==0){
151         h->first = p->next;
152         p->next = NULL;

```

```

153     free(p);
154     h->count--;
155
156     return;
157 }
158
159 Node* q;
160 for(int i=1; i<h->count; i++){
161
162     if(i==location){
163         q->next = p->next;
164         p->next = NULL;
165         free(p);
166         h->count--;
167
168         return;
169     }
170
171     q = p;
172     p = p->next;
173 }
174 }
175
176 bool searchList(Head* h, int key){
177     Node* p = h->first;
178
179     for(int i=0; i<h->count; i++){
180
181         if(p->data == key){
182             return true;
183         }
184
185         p = p->next;
186     }
187
188     return false;
189 }
190
191 void printListHead(Head* h){
192     Node* p = h->first;
193
194     for(int i=0; i<h->count; i++){
195         cout<<p->data<<" ";
196         p = p->next;
197     }
198 }
199
200 void printListTail(Head* h){
201     Node* p = h->last;
202
203     for(int i=0; i<h->count; i++){
204         cout<<p->data<<" ";
205         p = p->prev;
206     }
207 }
208
209 int main(){
210     Head* h = new Head(0, NULL, NULL);
211
212     int opt, data, index;
213     char choice;
214     while(1){
215         cout<<"\nMENU\n a. Add directly to LinkedList.\n b. Use functions.\n";
216         cout<<"Choose: ";
217         cin>>choice;
218
219         switch (choice){
220
221             case 'a':
222                 int num;
223                 cout<<"Enter the data to add in LinkedList (enter '-1' if you want stop): ";
224                 while(1){
225                     cin>>num;
226                     if(num == -1) break;
227                     Node* addNode = new Node(num);
228                     if(h->count == 0){

```

```

229         h->first = addNode;
230         h->last = addNode;
231         h->count++;
232     }
233     else{
234         h->last->next = addNode;
235         addNode->prev = h->last;
236         h->last = addNode;
237         h->count++;
238     }
239 }
240 break;
241 case 'b':
242     cout<<"\nMENU 1.0\n 1.Add at head\n 2.Add at tail\n 3.Add after\n 4.Delete (by Value)\n 5.Delete (by Index)\n 6.Search\n 7.Display from Head\n 8.Display from Tail\n";
243     cout<<"Enter your option: ";
244
245     cin>>opt;
246     cout<<"\n";
247     if(opt>8) break;
248
249     switch(opt){
250
251     case 1:
252         cout<<"Enter data to add: ";
253         cin>>data;
254         cout<<"Adding data...\n";
255         insertAtHead(h, data);
256         cout<<"\n";
257         break;
258
259     case 2:
260         cout<<"Enter data to add: ";
261         cin>>data;
262         cout<<"Adding data...\n";
263         insertAtTail(h, data);
264         cout<<"\n";
265         break;
266
267     case 3:
268         cout<<"Enter the index: ";
269         cin>>index;
270         cout<<"Enter data to add: ";
271         cin>>data;
272         cout<<"Adding data...\n";
273         insertAfter(h, data, index);
274         cout<<"\n";
275         break;
276
277     case 4:
278         cout<<"Enter data to delete: ";
279         cin>>data;
280         cout<<"Deleting data...\n";
281         delNode_byVal(h, data);
282         cout<<"\n";
283         break;
284
285     case 5:
286         cout<<"Enter the index: ";
287         cin>>index;
288         cout<<"Deleting data...\n";
289         delNodeAt(h, index);
290         cout<<"\n";
291         break;
292
293     case 6:
294         cout<<"Enter data to search: ";
295         cin>>data;
296         cout<<"Searching data...\n";
297         cout<<searchList(h, data)<<endl;
298         break;
299
300     case 7:
301         cout<<"Displaying the LinkedList from Head: ";
302         printListHead(h);
303         cout<<"\n";

```

```
304         break;
305
306     case 8:
307         cout<<"Displaying the LinkedList from Tail: ";
308         printListTail(h);
309         cout<<"\n";
310         break;
311
312     case 9:
313         break;
314
315     default:
316         break;
317
318     }
319
320     default:
321         break;
322     }
323 }
324
325 return 0;
326 }
```

OUTPUT:

MENU

- a. Add directly to LinkedList.
- b. Use functions.

Choose: a

Enter the data to add in LinkedList (enter '-1' if you want stop): 1 2 3 4 5
-1

MENU

- a. Add directly to LinkedList.
- b. Use functions.

Choose: b

MENU 1.0

- 1.Add at head
- 2.Add at tail
- 3.Add after
- 4.Delete (by Value)
- 5.Delete (by Index))
- 6.Search
- 7.Display from Head
- 8.Display from Tail
- 9.Exit

Enter your option: 7

Displaying the LinkedList from Head: 1 2 3 4 5

MENU

- a. Add directly to LinkedList.
- b. Use functions.

Choose: b

MENU 1.0

- 1.Add at head
- 2.Add at tail
- 3.Add after
- 4.Delete (by Value)
- 5.Delete (by Index))
- 6.Search
- 7.Display from Head
- 8.Display from Tail
- 9.Exit

Enter your option: 8

Displaying the LinkedList from Tail: 5 4 3 2 1

MENU

- a. Add directly to LinkedList.
- b. Use functions.

Choose: b

MENU 1.0

- 1.Add at head
- 2.Add at tail
- 3.Add after
- 4.Delete (by Value)
- 5.Delete (by Index))
- 6.Search
- 7.Display from Head
- 8.Display from Tail
- 9.Exit

Enter your option: 3

Enter the index: 2

Enter data to add: 3

Adding data...


```

MENU
  a. Add directly to LinkedList.
  b. Use functions.
Choose: b

MENU 1.0
  1.Add at head
  2.Add at tail
  3.Add after
  4.Delete (by Value)
  5.Delete (by Index))
  6.Search
  7.Display from Head
  8.Display from Tail
  9.Exit
Enter your option: 7

Displaying the LinkedList from Head: 1 2 3 3 4 5

```

ALGORITHM:

Algorithm to create Doubly Linked list

Begin:

alloc (head)

If (head == NULL) then

write ('Unable to allocate memory')

End if

Else then

read (data)

head.data \leftarrow data;

head.prev \leftarrow NULL;

head.next \leftarrow NULL;

last \leftarrow head;

write ('List created successfully')

End else

End

Algorithm to traverse Doubly Linked list

Input : head {Pointer to the first node of the list}

Begin:

```
If (head == NULL) then
write ('List is empty')
End if
Else then
temp ← head;
While (temp != NULL) do
write ('Data = ', temp.data)
temp ← temp.next;
End while
End else
End
```

Algorithm to insert a node at the beginning:

Input : head {A pointer pointing to the first node of the list}

Begin:

alloc (newNode)

If (newNode == NULL) then

write ('Unable to allocate memory')

End if

Else then

read (data)

newNode.data ← data;

newNode.prev ← NULL;

newNode.next ← head;

head.prev ← newNode;

head ← newNode;

write('Node added successfully at the beginning of List')

End else

End

Algorithm to insert a node at the end of Doubly linked list

Input : last {Pointer to the last node of doubly linked list}

Begin:

```

alloc (newNode)
If (newNode == NULL) then
write ('Unable to allocate memory')
End if
Else then
read (data)
newNode.data  $\leftarrow$  data;
newNode.next  $\leftarrow$  NULL;
newNode.prev  $\leftarrow$  last;
last.next  $\leftarrow$  newNode;
last  $\leftarrow$  newNode;
write ('Node added successfully at the end of List')
End else
End

```

Algorithm to insert node at any position of doubly linked list

Input : head {Pointer to the first node of doubly linked list}

: last {Pointer to the last node of doubly linked list}

: N {Position where node is to be inserted}

Begin:

temp \leftarrow head

For $i \leftarrow 1$ to $N-1$ do

If (temp == NULL) then

break

End if

temp \leftarrow temp.next;

End for

If ($N == 1$) then

insertAtBeginning()

End if

Else If (temp == last) then

insertAtEnd()

```

End if
Else If (temp != NULL) then
alloc (newNode)
read (data)
newNode.data ← data;
newNode.next ← temp.next
newNode.prev ← temp
If (temp.next != NULL) then
temp.next.prev ← newNode;
End if
temp.next ← newNode;
write('Node added successfully')
End if
End

```

Algorithm to delete node from beginning

Input: head {Pointer to first node of the linked list}

Begin:

```

If (head == NULL) then
write ('Can't delete from an empty list')
End if
Else then
toDelete ← head;
head ← head.next;
head.prev ← NULL;
unalloc (toDelete)
write ('Successfully deleted first node from the list')
End if
End

```

Algorithm to delete node from end

Input: last {Pointer to last node of the linked list}

Begin:

```

If (last == NULL) then
write ('Can't delete from an empty list')
End if
Else then
toDelete ← last;
last ← last.prev;
last.next ← NULL;
unalloc (toDelete)
write ('Successfully deleted last node from the list')
End if
End

```

Algorithm to delete node from any position

Input : head {Pointer to the first node of the list}

last {Pointer to the last node of the list}

N {Position to be deleted from list}

Begin:

```
current ← head;
```

```
For i ← 1 to N and current != NULL do
```

```
current ← current.next;
```

```
End for
```

```
If (N == 1) then
```

```
deleteFromBeginning()
```

```
End if
```

```
Else if (current == last) then
```

```
deleteFromEnd()
```

```
End if
```

```
Else if (current != NULL) then
```

```
current.prev.next ← current.next
```

```
If (current.next != NULL) then
```

```
current.next.prev ← current.prev;
```

```
End if
```

```
unalloc (current)
write ('Node deleted successfully from ', N, ' position')
End if
Else then
write ('Invalid position')
End if
End
```

Algorithm to insert a node at the beginning:

Input : head {A pointer pointing to the first node of the list}

Data {Data to search}

Begin:

```
alloc (newNode)
while(newNode != NULL)
{
if(newNode->data == Data) then
return 1;
newNode = newNode->next;
}
return 0;
End
```