Jawahar Education Societys Annasaheb Chudaman Patil College of Engineering, Kharghar, Navi Mumbai

NAME: PRIYUSH BHIMRAO KHOBRAGADE

PRN NO: 211112018

SUBJECT: DATA STRUCTURE LAB

AIM: Implement Doubly Linked List ADT.

PAGE NO.:
DATE.: / / 20

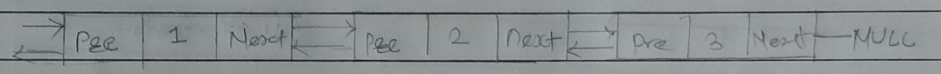
Practical -06

Aim :- Implement Doubly Linked List

Theory

Doubly Linked List

Doubly linked list is complex type of linked list a node contains a pointer to the previous as well as then next node in the sequence. Therefore, in doubly linked list, a node consist of three part: node data, pointer to next node in sequence (next pointer), pointer to the previous node (previous pointer).



- Structure of a node

```
struct node
{
    struct node *prev;
    int data;
    struct node *next;
};
```

- Operation or insertion on double link list:-

1) Inserting At Beginning of the list:-

"Adding the node into the linked list at beginning"

Ans

1) The prev pointer of first node will always be null & next will point to front.
2) if node insert is the first node of the list then we make front and end point
3) else we only front point to this node.

Teachers Signature _____ 6.1

AIM: Implement Doubly Linked List ADT.

PAGE NO.:
DATE.: / / 20

```
while(tp->next != NULL)
    tp = tp->next;
```

b] Inserting At specific in the list. [After a Node]

"Adding the node the linked list after specific node.

```
struct node* temp = head;
while (TRUE)
{
  temp = temp->next;
  Positions--;
}
```

c] Inserting At End of the List:

"Adding the node into the linked list at end"

Th

- The next pointer of last node will always be NULL and prev will point end.
- If the node is inserted is the first node of the list then we make front and end point to this node.
- Else we only make end point to this node.

```
while(tp->next != NULL)
    tp = tp->next;
```

```
tp->next = temp;
temp->prev = tp;
```

Teachers Signature

6.2

AIM: Implement Doubly Linked List ADT.

PAGE NO.:
DATE: / / 20

Removal / Deleting

* Deleting from Beginning of the list:

Remove at the node from beginning of the list.

Algorithm:

```
head = head -> next;
free (temp);
```

```
head = head -> next,
free (temp),
temp = NULL;
head -> prev = NULL;
```

* Deleting from End of the list.

Remove the node from end of the list.

Algorithm:-

```
while (temp -> next != NULL)
{
  temp = temp -> next;
}

while (temp -> next -> = NULL)
{
  temp = temp -> next;
}
! temp = temp -> prev;

temp -> next = NULL;
free (temp);
temp = NULL;
```

6.3

Teachers Signature _____

AIM: Implement Doubly Linked List ADT.

PAGE NO.:
DATE.: / / 20

Deleting a specific Node from the list.

Algorithm:-

```
while(position->1)
{
    temp = temp->next;
    position--;
}
temp2 = temp->prev;
temp2->next = temp->next;
temp->next->prev = temp2;
free(temp);
temp=(NULL);
```

- Conclusion:-

Hence we can understanding the Doubly linked list and inserting and deleting operation on it.

Teachers Signature _____ 6.4

AIM: Implement Doubly Linked List ADT.

Input:

```
/*
 * C Program to Implement a Doubly Linked List & provide Insertion, Deletion & Display Operations
 */
#include <stdio.h>
#include <stdlib.h>
struct node
{
   struct node *prev;
   int n;
    struct node *next;
}*h,*temp,*temp1,*temp2,*temp4;

void insert1();
void insert2();
void insert3();
void traversebeg();
void traverseend(int);
void sort();
void search();
void update();
void delete();

int count = 0;

void main()
{
   int ch;

   h = NULL;
   temp = temp1 = NULL;

   printf("\n 1 - Insert at beginning");
   printf("\n 2 - Insert at end");
   printf("\n 3 - Insert at specific position i");
   printf("\n 4 - Delete specific at i");
   printf("\n 5 - Display from beginning");
   printf("\n 6 - Display from end");
   printf("\n 7 - Search for element");
   printf("\n 8 - Sort the list");
   printf("\n 9 - Update an element");
   printf("\n 10 - Exit");

   while (1)
   {
      printf("\n Enter choice : ");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         insert1();
         break;
      case 2:
         insert2();
         break;
      case 3:
         insert3();
         break;
      case 4:
         delete();
         break;
      case 5:
         traversebeg();
         break;
      case 6:
         temp2 = h;
         if (temp2 == NULL)
            printf("\n Error : List empty to display ");
         else
         {
```

AIM: Implement Doubly Linked List ADT.

```
70              printf("\n Reverse order of linked list is : ");
71              traverseend(temp2->n);
72           }
73           break;
74        case 7:
75           search();
76           break;
77        case 8:
78           sort();
79           break;
80        case 9:
81           update();
82           break;
83        case 10:
84           exit(0);
85        default:
86           printf("\n Wrong choice menu");
87        }
88     }
89  }
90
91  /* TO create an empty node*/
92  void create()
93  {
94     int data;
95
96     temp =(struct node*)malloc(1*sizeof(struct node));
97     temp->prev = NULL;
98     temp->next = NULL;
99     printf("\n Enter value to node : ");
100     scanf("%d", &data);
101     temp->n = data;
102     count++;
103  }
104
105  /*  TO insert at beginning*/
106  void insert1()
107  {
108     if (h == NULL)
109     {
110        create();
111        h = temp;
112        temp1 = h;
113     }
114     else
115     {
116        create();
117        temp->next = h;
118        h->prev = temp;
119        h = temp;
120     }
121  }
122
123  /* To insert at end*/
124  void insert2()
125  {
126     if (h == NULL)
127     {
128        create();
129        h = temp;
130        temp1 = h;
131     }
132     else
133     {
134        create();
135        temp1->next = temp;
136        temp->prev = temp1;
137        temp1 = temp;
138     }
139  }
140
141  /* To insert at any position */
142  void insert3()
143  {
144     int pos, i = 2;
```

```
145
146     printf("\n Enter position to be inserted : ");
147     scanf("%d", &pos);
148     temp2 = h;
149
150     if ((pos < 1) || (pos >= count + 1))
151     {
152        printf("\n Position out of range to insert");
153        return;
154     }
155     if ((h == NULL) && (pos != 1))
156     {
157        printf("\n Empty list cannot insert other than 1st position");
158        return;
159     }
160     if ((h == NULL) && (pos == 1))
161     {
162        create();
163        h = temp;
164        temp1 = h;
165        return;
166     }
167     else
168     {
169        while (i < pos)
170        {
171           temp2 = temp2->next;
172           i++;
173        }
174        create();
175        temp->prev = temp2;
176        temp->next = temp2->next;
177        temp2->next->prev = temp;
178        temp2->next = temp;
179     }
180  }
181
182  /* To delete an element */
183  void delete()
184  {
185     int i = 1, pos;
186
187     printf("\n Enter position to be deleted : ");
188     scanf("%d", &pos);
189     temp2 = h;
190
191     if ((pos < 1) || (pos >= count + 1))
192     {
193        printf("\n Error : Position out of range to delete");
194        return;
195     }
196     if (h == NULL)
197     {
198        printf("\n Error : Empty list no elements to delete");
199        return;
200     }
201     else
202     {
203        while (i < pos)
204        {
205           temp2 = temp2->next;
206           i++;
207        }
208        if (i == 1)
209        {
210           if (temp2->next == NULL)
211           {
212              printf("Node deleted from list");
213              free(temp2);
214              temp2 = h = NULL;
215              return;
216           }
217        }
218        if (temp2->next == NULL)
219        {
```

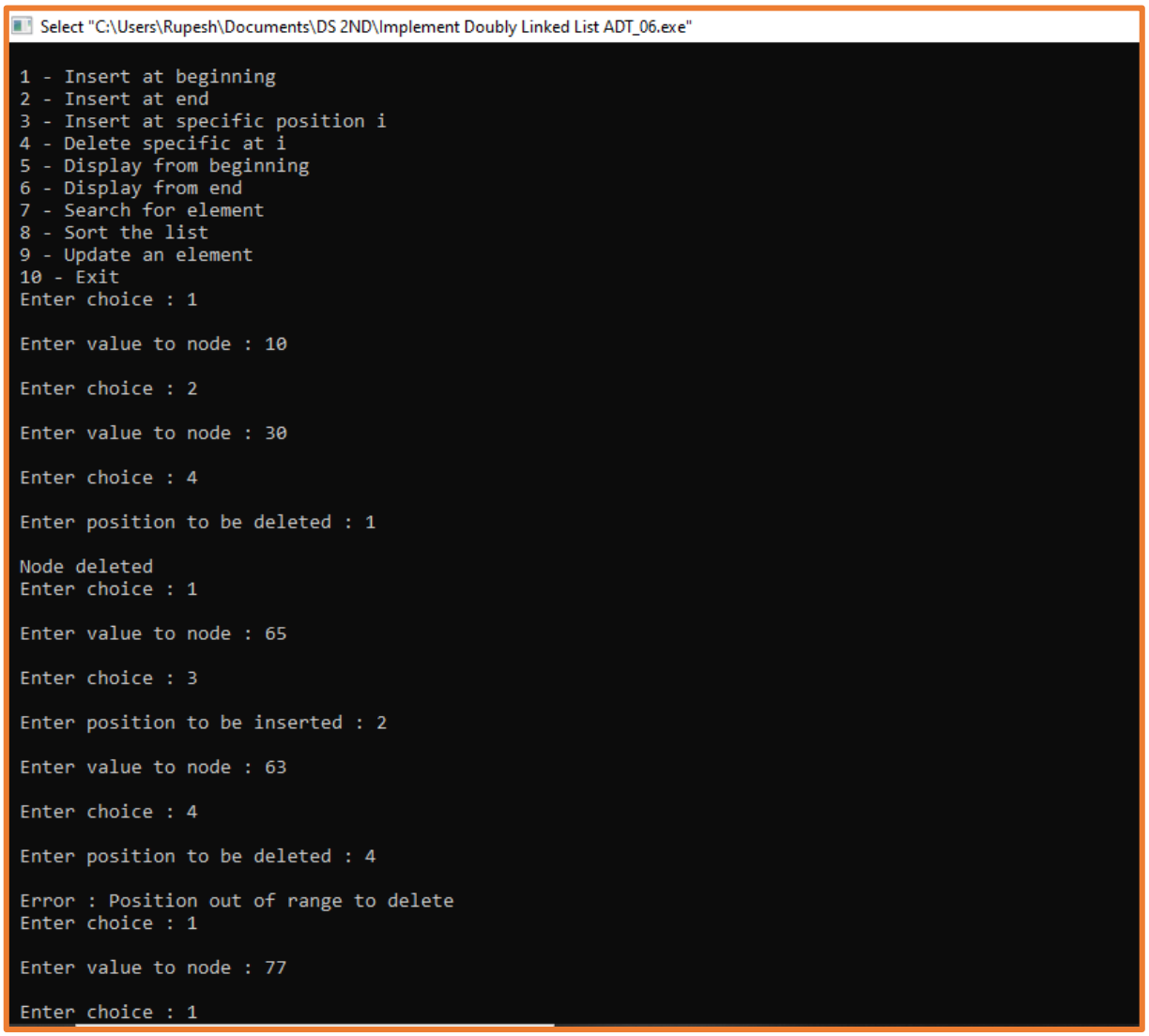
```
220             temp2->prev->next = NULL;
221             free(temp2);
222             printf("Node deleted from list");
223             return;
224         }
225         temp2->next->prev = temp2->prev;
226         if (i != 1)
227             temp2->prev->next = temp2->next;    /* Might not need this statement if i == 1 check */
228         if (i == 1)
229             h = temp2->next;
230         printf("\n Node deleted");
231         free(temp2);
232     }
233     count--;
234 }
235
236 /* Traverse from beginning */
237 void traversebeg()
238 {
239     temp2 = h;
240
241     if (temp2 == NULL)
242     {
243         printf("List empty to display \n");
244         return;
245     }
246     printf("\n Linked list elements from beginning : ");
247
248     while (temp2->next != NULL)
249     {
250         printf(" %d ", temp2->n);
251         temp2 = temp2->next;
252     }
253     printf(" %d ", temp2->n);
254 }
255
256 /* To traverse from end recursively */
257 void traverseend(int i)
258 {
259     if (temp2 != NULL)
260     {
261         i = temp2->n;
262         temp2 = temp2->next;
263         traverseend(i);
264         printf(" %d ", i);
265     }
266 }
267
268 /* To search for an element in the list */
269 void search()
270 {
271     int data, count = 0;
272     temp2 = h;
273
274     if (temp2 == NULL)
275     {
276         printf("\n Error : List empty to search for data");
277         return;
278     }
279     printf("\n Enter value to search : ");
280     scanf("%d", &data);
281     while (temp2 != NULL)
282     {
283         if (temp2->n == data)
284         {
285             printf("\n Data found in %d position",count + 1);
286             return;
287         }
288         else
289             temp2 = temp2->next;
290             count++;
291     }
292     printf("\n Error : %d not found in list", data);
293 }
294
```

AIM: Implement Doubly Linked List ADT.

```
295 /* To update a node value in the list */
296 void update()
297 {
298    int data, data1;
299
300    printf("\n Enter node data to be updated : ");
301    scanf("%d", &data);
302    printf("\n Enter new data : ");
303    scanf("%d", &data1);
304    temp2 = h;
305    if (temp2 == NULL)
306    {
307       printf("\n Error : List empty no node to update");
308       return;
309    }
310    while (temp2 != NULL)
311    {
312       if (temp2->n == data)
313       {
314
315          temp2->n = data1;
316          traversebeg();
317          return;
318       }
319       else
320          temp2 = temp2->next;
321    }
322
323    printf("\n Error : %d not found in list to update", data);
324 }
325
326 /* To sort the linked list */
327 void sort()
328 {
329    int i, j, x;
330
331    temp2 = h;
332    temp4 = h;
333
334    if (temp2 == NULL)
335    {
336       printf("\n List empty to sort");
337       return;
338    }
339
340    for (temp2 = h; temp2 != NULL; temp2 = temp2->next)
341    {
342       for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)
343       {
344          if (temp2->n > temp4->n)
345          {
346             x = temp2->n;
347             temp2->n = temp4->n;
348             temp4->n = x;
349          }
350       }
351    }
352    traversebeg();
353 }
```

AIM: Implement Doubly Linked List ADT.

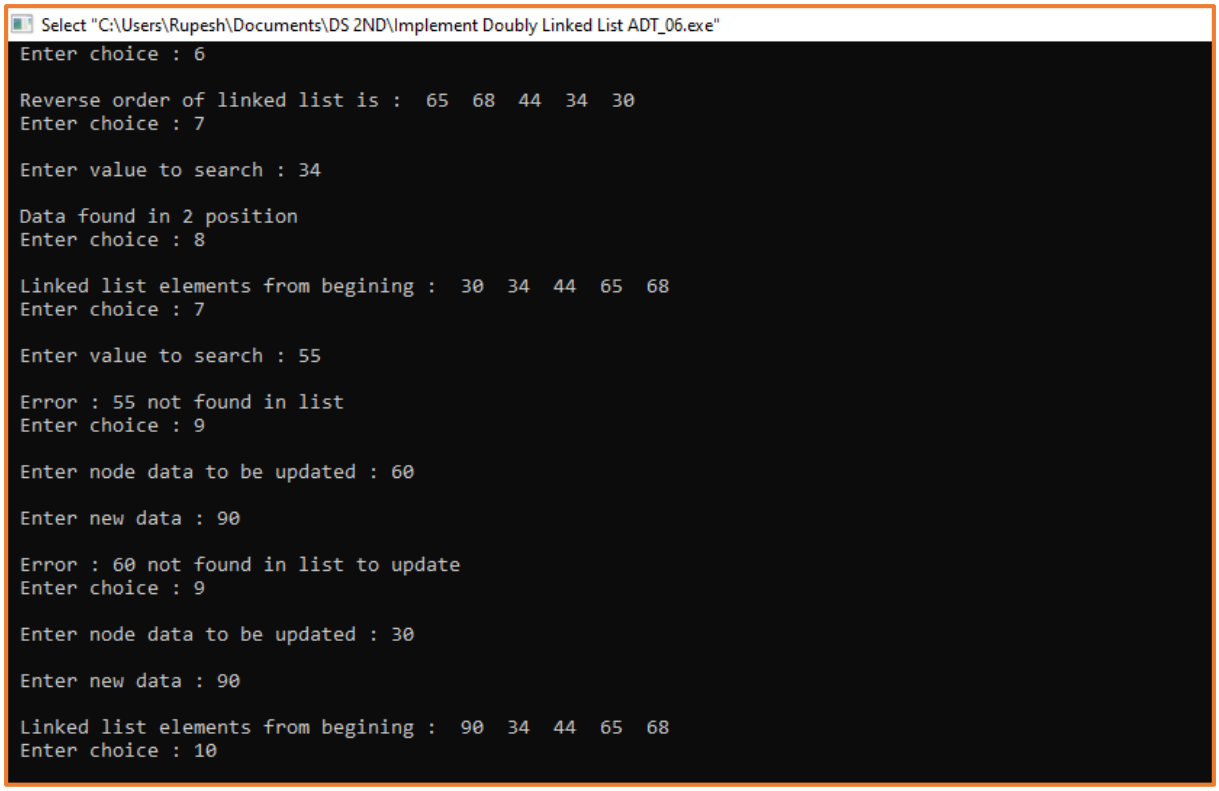
Output:-



AIM: Implement Doubly Linked List ADT.

```
Enter value to node : 77

Enter choice : 1

Enter value to node : 44

Enter choice : 3

Enter position to be inserted : 2

Enter value to node : 34

Enter choice : 4

Enter position to be deleted : 3

Node deleted
Enter choice : 7

Enter value to search : 15

Error : 15 not found in list
Enter choice : 8

Linked list elements from begining :  30  34  44  63  65
Enter choice : 9

Enter node data to be updated : 33

Enter new data : 89

Error : 33 not found in list to update
Enter choice : 9

Enter node data to be updated : 63

Enter new data : 68

Linked list elements from begining :  30  34  44  68  65
Enter choice : 6

Reverse order of linked list is :  65  68  44  34  30
Enter choice : 7
```

AIM: Implement Doubly Linked List ADT.



Conclusion: -Hence we can understanding the doubly linked list and inserting and deleting operation on it.