1. Source Code for the Implementation of Single Linked List:
2. include <stdio.h>
3. include <conio.h>
4. include <stdlib.h>
5. struct slinklist
6. {
7. int data;
8. struct slinklist *next;
9. };
10. typedef struct slinklist node;
11. node *start = NULL;
12. int menu()
13. {
14. int ch;
15. clrscr();
16. printf("\n 1.Create a list ");
17. printf("\n--------------------------");
18. printf("\n 2.Insert a node at beginning ");
19. printf("\n 3.Insert a node at end");
20. printf("\n 4.Insert a node at middle");
21. printf("\n--------------------------");
22. printf("\n 5.Delete a node from beginning");
23. printf("\n 6.Delete a node from Last");
24. printf("\n 7.Delete a node from Middle");
25. printf("\n--------------------------");
26. printf("\n 8.Traverse the list (Left to Right)");
27. printf("\n 9.Traverse the list (Right to Left)");
28.
29. printf("\n--------------------------");
30. printf("\n 10. Count nodes ");
31. printf("\n 11. Exit ");
32. printf("\n\n Enter your choice: ");
33. scanf("%d",&ch);
34. return ch;
35. }
36. node* getnode()
37. {
38. node * newnode;
39. newnode = (node *) malloc(sizeof(node));
40. printf("\n Enter data: ");
41. scanf("%d", &newnode -> data);
42. newnode -> next = NULL; return
43. newnode;
44. }
45. int countnode(node *ptr)
46. {
47. int count=0;
48. while(ptr != NULL)
49. {
50. count++;
51. ptr = ptr -> next;
52. }

```
53. return (count);
54. }
55. void createlist(int n)
56. {
57. int i;
58. node *newnode;
59. node *temp;
60. for(i = 0; i < n; i++)
61. {
62. newnode = getnode();
63. if(start == NULL)
64. {
65. start = newnode;
66. }
67. else
68. {
69. temp = start;
70. while(temp -> next != NULL)
71. temp = temp -> next;
72. temp -> next = newnode;
73. }
74. }
75. }
76. void traverse()
77. {
78. node *temp;
79. temp = start;
80. printf("\n The contents of List (Left to Right): \n");
81. if(start == NULL)
82. {
83. printf("\n Empty List");
84. return;
85. }
86. else
87. {
88.
89. while(temp != NULL)
90. {
91. printf("%d-->", temp ->
92. data); temp = temp -> next;
93. }
94. }
95. printf(" X ");
96. }
97. void rev_traverse(node *start)
98. {
99. if(start == NULL)
100.         {
101.         return;
102.         }
103.         else
104.         {
```

```
105.        rev_traverse(start -> next);
106.        printf("%d -->", start -> data);
107.        }
108.        }
109.        void insert_at_beg()
110.        {
111.        node *newnode;
112.        newnode = getnode();
113.        if(start == NULL)
114.        {
115.        start = newnode;
116.        }
117.        else
118.        {
119.        newnode -> next =
120.        start; start = newnode;
121.        }
122.        }
123.        void insert_at_end()
124.        {
125.        node *newnode, *temp;
126.        newnode = getnode();
127.        if(start == NULL)
128.        {
129.        start = newnode;
130.        }
131.        else
132.        {
133.        temp = start;
134.        while(temp -> next != NULL)
135.        temp = temp -> next;
136.        temp -> next = newnode;
137.        }
138.        }
139.        void insert_at_mid()
140.        {
141.        node *newnode, *temp, *prev;
142.        int pos, nodectr, ctr = 1;
143.        newnode = getnode();
144.        printf("\n Enter the position: ");
145.        scanf("%d", &pos);
146.        nodectr = countnode(start);
147.
148.
149.
150.        if(pos > 1 && pos < nodectr)
151.        {
152.        temp = prev = start;
153.        while(ctr < pos)
154.        {
155.        prev = temp;
156.        temp = temp ->
```

```
157.        next; ctr++;
158.        }
159.        prev -> next = newnode;
160.        newnode -> next = temp;
161.        }
162.        else
163.        printf("position %d is not a middle position", pos);
164.        }
165.        void delete_at_beg()
166.        {
167.        node *temp;
168.        if(start == NULL)
169.        {
170.        printf("\n No nodes are exist..");
171.        return ;
172.        }
173.        else
174.        {
175.        temp = start;
176.        start = temp -> next;
177.        free(temp);
178.        printf("\n Node deleted ");
179.        }
180.        }
181.        void delete_at_last()
182.        {
183.        node *temp, *prev;
184.        if(start == NULL)
185.        {
186.        printf("\n Empty
187.        List.."); return ;
188.        }
189.        else
190.        {
191.        temp = start;
192.        prev = start;
193.        }
194.        while(temp -> next != NULL)
195.        {
196.        prev = temp;
197.        temp = temp -> next;
198.        }
199.        prev -> next = NULL;
200.        free(temp);
201.        printf("\n Node deleted ");
202.        }
203.        void delete_at_mid()
204.        {
205.        int ctr = 1, pos,
206.        nodectr; node *temp,
207.        *prev; if(start == NULL)
208.        {
```

```
209.        printf("\n Empty List..");
210.
211.
212.
213.
214.        return ;
215.        }
216.        else
217.        {
218.        printf("\n Enter position of node to delete: ");
219.        scanf("%d", &pos);
220.        nodectr = countnode(start);
221.        if(pos > nodectr)
222.        {
223.        printf("\nThis node doesnot exist");
224.        }
225.        if(pos > 1 && pos < nodectr)
226.        {
227.        temp = prev = start;
228.        while(ctr < pos)
229.        {
230.        prev = temp;
231.        temp = temp ->
232.        next; ctr ++;
233.        }
234.        prev -> next = temp -> next;
235.        free(temp);
236.        printf("\n Node deleted..");
237.        }
238.        else
239.        {
240.        printf("\n Invalid position..");
241.        getch();
242.        }
243.        }
244.        }
245.        void main(void)
246.        {
247.        int ch, n;
248.        clrscr();
249.        while(1)
250.        {
251.        ch = menu();
252.        switch(ch)
253.        {
254.        case 1:
255.        if(start == NULL)
256.        {
257.        printf("\n Number of nodes you want to create: ");
258.        scanf("%d", &n);
259.        createlist(n);
260.        printf("\n List created..");
```

```
261.        }
262.        else
263.        printf("\n List is already created..");
264.        break;
265.        case 2:
266.        insert_at_beg();
267.        break;
268.        case 3:
269.        insert_at_end();
270.        break;
271.        case 4:
272.        insert_at_mid();
273.        break;
274.
275.
276.
277.        case 5:
278.        delete_at_beg();
279.        break;
280.        case 6:
281.        delete_at_last();
282.        break;
283.        case 7:
284.        delete_at_mid();
285.        break;
286.        case 8:
287.        traverse();
288.        break;
289.        case 9:
290.        printf("\n The contents of List (Right to Left): \n");
291.        rev_traverse(start);
292.        printf(" X ");
293.        break;
294.        case 10:
295.        printf("\n No of nodes : %d ", countnode(start));
296.        break;
297.        case 11 :
298.        exit(0);
299.        }
300.        getch();
301.        }
302.        }
303.
```