

```

1: //Insertion and Deletion in Single Linked List
2: #include <stdio.h>
3: #include <stdlib.h>
4:
5: struct Node
6: {
7:     int data;
8:     struct Node* next;
9: };
10:
11: struct Node* head = NULL;
12:
13: void insertAtBeginning(int num)
14: {
15:     struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
16:     new_node->data = num;
17:     new_node->next = head;
18:     head = new_node;
19:     printf("%d inserted at beginning of list\n", num);
20: }
21:
22: void insertAtEnd(int num)
23: {
24:     struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
25:     new_node->data = num;
26:     new_node->next = NULL;
27:
28:     if (head == NULL)
29:     {
30:         head = new_node;
31:     }
32:     else
33:     {
34:         struct Node* current = head;
35:         while (current->next != NULL)
36:         {
37:             current = current->next;
38:         }
39:         current->next = new_node;
40:     }
41:     printf("%d inserted at end of list\n", num);
42: }
43:
44: void insertAtPosition(int num, int position)
45: {
46:     if (position < 1)
47:     {
48:         printf("Invalid position\n");
49:         return;

```

```

50:     }
51:     if (position == 1)
52:     {
53:         insertAtBeginning(num);
54:         return;
55:     }
56:
57:     struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
58:     new_node->data = num;
59:
60:     struct Node* current = head;
61:     struct Node* previous = NULL;
62:     int count = 1;
63:
64:     while (current != NULL && count < position) {
65:         previous = current;
66:         current = current->next;
67:         count++;
68:     }
69:
70:     if (count < position)
71:     {
72:         printf("Position exceeds length of list\n");
73:         free(new_node);
74:         return;
75:     }
76:
77:     previous->next = new_node;
78:     new_node->next = current;
79:     printf("%d inserted at position %d\n", num, position);
80: }
81:
82: void deleteAtBeginning()
83: {
84:     if (head == NULL)
85:     {
86:         printf("List is empty\n");
87:         return;
88:     }
89:     struct Node* temp = head;
90:     head = head->next;
91:     printf("%d deleted from beginning of list\n", temp->data);
92:     free(temp);
93: }
94:
95: void deleteAtEnd()
96: {
97:     if (head == NULL)
98:     {

```

```

99:         printf("List is empty\n");
100:         return;
101:     }
102:     if (head->next == NULL)
103:     {
104:         printf("%d deleted from end of list\n", head->data);
105:         free(head);
106:         head = NULL;
107:         return;
108:     }
109:
110:     struct Node* current = head;
111:     struct Node* previous = NULL;
112:     while (current->next != NULL)
113:     {
114:         previous = current;
115:         current = current->next;
116:     }
117:
118:     printf("%d deleted from end of list\n", current->data);
119:     free(current);
120:     previous->next = NULL;
121: }
122:
123: void deleteAtPosition(int position)
124: {
125:     if (position < 1)
126:     {
127:         printf("Invalid position\n");
128:         return;
129:     }
130:     if (position == 1)
131:     {
132:         deleteAtBeginning();
133:         return;
134:     }
135:
136:     struct Node* current = head;
137:     struct Node* previous = NULL;
138:     int count = 1;
139:
140:     while (current != NULL && count < position) {
141:         previous = current;
142:         current = current->next;
143:         count++;
144:     }
145:
146:     if (count < position || current == NULL)
147:     {

```

```

148:         printf("Position %d exceeds length of list\n", position);
149:         return;
150:     }
151:
152:     printf("%d deleted from position %d\n", current->data, position);
153:     previous->next = current->next;
154:     free(current);
155: }
156:
157: void display()
158: {
159:     if (head == NULL)
160:     {
161:         printf("List is empty\n");
162:         return;
163:     }
164:
165:     struct Node* current = head;
166:     printf("List: ");
167:     while (current != NULL)
168:     {
169:         printf("%d ", current->data);
170:         current = current->next;
171:     }
172:     printf("\n");
173: }
174: int main()
175: {
176:     insertAtBeginning(3);
177:     insertAtEnd(5);
178:     insertAtBeginning(7);
179:     insertAtPosition(9,2);
180:
181:     display();
182:
183:     deleteAtPosition(2);
184:
185:     display();
186:
187:     deleteAtPosition(4);
188:
189:     display();
190:
191:     return 0;
192: }

```