```c
//Insertion and Deletion in Circular Single linked list

#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int data;
    struct Node* next;
};

struct Node* head = NULL;

void insertAtBeginning(int data)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (head == NULL)
    {
        head = newNode;
        head->next = head; // point to itself to create circular list
        return;
    }

    struct Node* current = head;
    while (current->next != head)
    {
        current = current->next;
    }

    newNode->next = head;
    current->next = newNode;
    head = newNode;
}

void insertAtEnd(int data)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (head == NULL)
    {
        head = newNode;
        head->next = head; // point to itself to create circular list
        return;
    }
```

```
50:
51:     struct Node* current = head;
52:     while (current->next != head)
53:     {
54:         current = current->next;
55:     }
56:
57:     current->next = newNode;
58:     newNode->next = head;
59: }
60:
61: void insertAtPosition(int data, int position)
62: {
63:     if (position < 1)
64:     {
65:         printf("Invalid position\n");
66:         return;
67:     }
68:
69:     if (position == 1)
70:     {
71:         insertAtBeginning(data);
72:         return;
73:     }
74:
75:     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
76:     newNode->data = data;
77:     newNode->next = NULL;
78:
79:     struct Node* current = head;
80:     int count = 1;
81:     while (count < position - 1 && current->next != head)
82:     {
83:         current = current->next;
84:         count++;
85:     }
86:
87:     if (count != position - 1)
88:     {
89:         printf("Invalid position\n");
90:         return;
91:     }
92:
93:     newNode->next = current->next;
94:     current->next = newNode;
95: }
96:
97: void deleteAtBeginning()
98: {
```

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

```c

    // Traverse to the given position
    while (current->next != head && count < position)
    {
        previous = current;
        current = current->next;
        count++;
    }

    // If given position is the head node
    if (current == head)
    {
        // Move previous to the last node
        previous = head;
        while (previous->next != head)
        {
            previous = previous->next;
        }

        head = head->next;
        previous->next = head;
        free(current);
    }
    // If given position is in the middle
    else if (current->next != head)
    {
        previous->next = current->next;
        free(current);
    }
    // If given position is the last node
    else
    {
        previous->next = head;
        free(current);
    }

    printf("Node at position %d deleted successfully\n", position);
}

void display()
{
    struct Node *temp = head;

    // If list is empty
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }
```

```
197:
198:     // Traverse the list and print data of each node
199:     printf("Nodes of circular linked list: ");
200:     do {
201:         printf("%d ", temp->data);
202:         temp = temp->next;
203:     } while (temp != head);
204:
205:     printf("\n");
206: }
207: int main()
208: {
209:     insertAtBeginning(3);
210:     insertAtEnd(5);
211:     insertAtBeginning(7);
212:     insertAtPosition(9,2);
213:
214:     display();
215:
216:     deleteAtPosition(2);
217:
218:     display();
219:
220:     deleteAtPosition(4);
221:
222:     display();
223:
224:     return 0;
225: }
```