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

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

int c = 0;

struct Node *createNode(int data)
{
  struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
  if (newNode == NULL)
  {
    printf("Memory allocation failed.\n");
    exit(1);
  }
  newNode->data = data;
  newNode->next = newNode;
  return newNode;
}

struct Node *createList(struct Node *head)
{
  struct Node *ptr, *newNode;
  int value, cont = 0;
  printf("Enter the value you want to enter: ");
  scanf("%d", &value);
  while (1)
  {
    newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    if (head == NULL)
    {
      head = ptr = newNode;
      head->next = head;
      c++;
    }
    else
    {
      ptr->next = newNode;
      ptr = ptr->next;
      c++;
    }
    printf("Enter 0 to continue and 1 to discontinue: ");
    scanf("%d", &cont);
    if (cont == 1)
    {
      newNode->next = head;
      break;
    }
    else
    {
      printf("Enter the value: ");
      scanf("%d", &value);
    }
  }
  return head;
}

void display(struct Node *head)
{
  struct Node *current = head;
  if (current == NULL)
  {
    printf("NULL\n");
    return;
  }
  if (head->next == current)
  {
    printf("%d -> %d\n", current->data, current->data);
    return;
  }
  while (current->next != head)
  {
    printf("%d -> ", current->data);
    current = current->next;
  }
  printf("%d -> ", current->data);
  printf("%d\n", head->data);
}

struct Node *insertFront(struct Node *head)
{
  int data;
  printf("Enter the data: ");
  scanf("%d", &data);
  struct Node *newNode = createNode(data);
  if (head == NULL)
  {
    head = newNode;
    newNode->next = head;
    c++;
    return head;
  }
  struct Node *current = head;
  while (current->next != head)
  {
    current = current->next;
  }
  newNode->next = head;
  current->next = newNode;
  c++;
  return newNode;
}
```

```c
struct Node *insertEnd(struct Node *head)
{
  int data;
  printf("Enter the data: ");
  scanf("%d", &data);
  struct Node *newNode = createNode(data);
  if (head == NULL)
  {
    head = newNode;
    c++;
    return head;
  }
  struct Node *current = head;
  while (current->next != head)
  {
    current = current->next;
  }
  current->next = newNode;
  newNode->next = head;
  c++;
  return head;
}

struct Node *insertAtPosition(struct Node *head)
{
  int data, position, i;

  printf("Enter the position you want: ");
  scanf("%d", &position);

  if (position == 1)
  {
    c++;
    return (insertFront(head));
  }
  else if (position == c)
  {
    c++;
    return (insertEnd(head));
  }
  else if ((position < 1) || (c <= 0) || (position > c))
  {
    printf("Position out of Bounds\n");
    return head;
  }

  printf("Enter data: ");
  scanf("%d", &data);
  struct Node *newNode = createNode(data);
  struct Node *current = head;
  for (i = 1; i <= position - 2; i++)
  {
    current = current->next;
  }
  newNode->next = current->next;
  current->next = newNode;
  c++;
  return head;
}

struct Node *deleteFront(struct Node *head)
{
  struct Node *ptr, *preptr;
  if (head == NULL)
  {
    printf("The list is empty.\n");
    return head;
  }
  if (head->next == head)
  {
    c--;
    free(head);
    head = NULL;
    return head;
  }
  ptr = preptr = head;
  while (ptr->next != head)
  {
    ptr = ptr->next;
  }

  ptr->next = preptr->next;
  head = ptr->next;
  c--;
  free(preptr);
  return head;
}

struct Node *deleteEnd(struct Node *head)
{
  struct Node *ptr, *preptr;
  if (head == NULL)
  {
    printf("The list is empty.\n");
    return head;
  }
  if (head->next == head)
  {
    c--;
    free(head);
    head = NULL;
    return head;
  }
  ptr = preptr = head;
  while (ptr->next != head)
  {
    preptr = ptr;
    ptr = ptr->next;
  }
  preptr->next = head;
  c--;
  free(ptr);

  return head;
}

struct Node *deleteAtPosition(struct Node *head)
{
  struct Node *ptr, *preptr;
  ptr = preptr = head;
  int position, i;
  printf("Enter position: ");
  scanf("%d", &position);
  if (position < 1 || head == NULL)
  {
    printf("Invalid position.\n");
    return head;
  }
  if (position == 1 && head->next == head)
  {
    c--;
    free(head);
    head = NULL;
    return head;
  }
  if (position == 1)
  {
    c--;
    return (deleteFront(head));
  }
  if (position == c)
  {
    c--;
    return (deleteEnd(head));
  }
  if ((position < 1) || (position > c) || (c <= 0))
  {
    printf("\nPosition of out of bounds\n");
    return head;
  }
  for (i = 1; i < position; i++)
  {
    preptr = ptr;
    ptr = ptr->next;
  }

  preptr->next = ptr->next;
  c--;
  free(ptr);

  return head;
}
```

```c
int main()
{
  struct Node *head = NULL;
  int choice, data, position;

  do
  {
    printf("1. Create the list\n");
    printf("2. Insert at the beginning\n");
    printf("3. Insert at the end\n");
    printf("4. Insert at a specific position\n");
    printf("5. Delete from the beginning\n");
    printf("6. Delete from the end\n");
    printf("7. Delete from a specific position\n");
    printf("8. Display the list\n");
    printf("9. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice)
    {
    case 1:
      head = createList(head);
      break;
    case 2:
      head = insertFront(head);
      break;
    case 3:
      head = insertEnd(head);
      break;
    case 4:
      head = insertAtPosition(head);
      break;
    case 5:
      head = deleteFront(head);
      break;
    case 6:
      head = deleteEnd(head);
      break;
    case 7:

      head = deleteAtPosition(head);
      break;
    case 8:
      display(head);
      break;
    case 9:
      printf("Exiting...\n");
      break;
    default:
      printf("Invalid choice.\n");
    }
  } while (choice != 9);
  struct Node *current = head->next;
  while (current->next != head)
  {
    struct Node *temp = current;
    current = current->next;
    free(temp);
  }
  free(head);

  return 0;
}
```