C 5_1_clinked_list.c > main()

```c
/* Write a C program to implement a circular linked list with the following operations :--

a) Insert an element at a specific position specified by the user. (consider pos of head as 0)
b) Insert an element at the beginning of the list (consider head as beginning)
c) Insert an element at the end of the list. (consider last element as end)
d) Delete an element from a specific position specified by the user.
e) Delete the first element from the list.
f) Delete the last element from the list.

Note: The program should continuously prompt the user to select an
operation and execute it until the user enters 0 to terminate. */

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

typedef struct node{
    int data;
    struct node *next;
}node;

node *head=NULL;

void insertAtPosition(int data, int position){
    node *newnode=(node*)malloc(sizeof(node));
    newnode->data=data;
    if(head==NULL){
        if(position==0){
            newnode->next=newnode;
            head=newnode;
        }
        else{
            printf("Invalid position!\n");
            free(newnode);
        }
        return;
    }
    node *temp=head;
    if(position==0){
        while (temp->next!=head){
            temp=temp->next;
        }
        newnode->next=head;
        temp->next=newnode;
```

```
43          temp->next=newnode;
44          head=newnode;
45          return;
46      }
47      for(int i=0;i<position-1&&temp->next!=head;i++){
48          temp=temp->next;
49      }
50      newnode->next=temp->next;
51      temp->next=newnode;
52  }
53
54  void insertAtBeginning(int data){
55      insertAtPosition(data, 0);
56  }
57
58  void insertAtEnd(int data){
59      node *newnode=(node*)malloc(sizeof(node));
60      newnode->data=data;
61      if(head==NULL){
62          newnode->next=newnode;
63          head=newnode;
64          return;
65      }
66      node *temp=head;
67      while (temp->next!=head){
68          temp=temp->next;
69      }
70      temp->next=newnode;
71      newnode->next=head;
72  }
73  void deleteFirst();
74  void deleteLast();
75
76  void deleteAtPosition(int position){
77      if(head==NULL){
78          printf("List is empty!\n");
79          return;
80      }
81      if(position==0){
82          deleteFirst();
83          return;
84      }
85      node *temp=head, *prev=NULL;
86      for(int i=0;i<position&&temp->next!=head;i++){
87          prev=temp;
88          temp=temp->next;
```

```
88          temp=temp->next;
89      }
90      if(temp==head){
91          printf("Invalid position!\n");
92          return;
93      }
94      prev->next=temp->next;
95      free(temp);
96  }
97
98  void deleteFirst(){
99      if(head==NULL){
100         printf("List is empty!\n");
101         return;
102     }
103     node *temp=head;
104     if(head->next==head){
105         head=NULL;
106         free(temp);
107         return;
108     }
109     node *last=head;
110     while (last->next!=head) last=last->next;
111     head=head->next;
112     last->next=head;
113     free(temp);
114 }
115
116 void deleteLast(){
117     if(head==NULL){
118         printf("List is empty!\n");
119         return;
120     }
121     node *temp=head, *prev=NULL;
122     if(head->next==head){
123         free(head);
124         head=NULL;
125         return;
126     }
127     while (temp->next!=head){
128         prev=temp;
129         temp=temp->next;
130     }
131     prev->next=head;
132     free(temp);
133 }
134
```

```c
    prev->next=head;
    free(temp);
}

void display(){
    if(head==NULL){
        printf("List is empty!\n");
        return;
    }
    node *temp=head;
    printf("Circular Linked List: ");
    do{
        printf("%d ", temp->data);
        temp=temp->next;
    } while (temp!=head);
    printf("\n");
}

int main(){
    int choice, data, position;
    while (1){
        printf("\nSelect an operation :--\n");
        printf("1. Insert at a specific position\n");
        printf("2. Insert at the beginning\n");
        printf("3. Insert at the end\n");
        printf("4. Delete from a specific position\n");
        printf("5. Delete first element\n");
        printf("6. Delete last element\n");
        printf("7. Display list\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter data and position: ");
                scanf("%d %d", &data, &position);
                insertAtPosition(data, position);
                display();
                break;
            case 2:
                printf("Enter data: ");
                scanf("%d", &data);
                insertAtBeginning(data);
                display();
                break;
```

```
            scanf("%d", &data);
            insertAtBeginning(data);
            display();
            break;
        case 3:
            printf("Enter data: ");
            scanf("%d", &data);
            insertAtEnd(data);
            display();
            break;
        case 4:
            printf("Enter position to delete: ");
            scanf("%d", &position);
            deleteAtPosition(position);
            display();
            break;
        case 5:
            deleteFirst();
            display();
            break;
        case 6:
            deleteLast();
            display();
            break;
        case 7:
            display();
            break;
        case 0:
            printf("Exiting program...\n");
            return 0;
        default:
            printf("Invalid choice!\n");
    }
  }
}
```

```
Select an operation :--
1. Insert at a specific position
2. Insert at the beginning
3. Insert at the end
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 1
Enter data and position: 8 0
Circular Linked List: 8

Select an operation :--
1. Insert at a specific position
2. Insert at the beginning
3. Insert at the end
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 2
Enter data: 3
Circular Linked List: 3 8

Select an operation :--
1. Insert at a specific position
2. Insert at the beginning
3. Insert at the end
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 3
Enter data: 5
Circular Linked List: 3 8 5
```

```
Select an operation :--
1. Insert at a specific position
2. Insert at the beginning
3. Insert at the end
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 4
Enter position to delete: 1
Circular Linked List: 3 5

Select an operation :--
1. Insert at a specific position
2. Insert at the beginning
3. Insert at the end
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 5
Circular Linked List: 5

Select an operation :--
1. Insert at a specific position
2. Insert at the beginning
3. Insert at the end
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 3
Enter data: 1
Circular Linked List: 5 1
```

```
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 3
Enter data: 1
Circular Linked List: 5 1

Select an operation :--
1. Insert at a specific position
2. Insert at the beginning
3. Insert at the end
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 6
Circular Linked List: 5

Select an operation :--
1. Insert at a specific position
2. Insert at the beginning
3. Insert at the end
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 7
Circular Linked List: 5

Select an operation :--
1. Insert at a specific position
2. Insert at the beginning
3. Insert at the end
4. Delete from a specific position
5. Delete first element
6. Delete last element
7. Display list
0. Exit
Enter your choice: 0
Exiting program...
```

C 5_2_stack.c > main()

```c
/* Write a C/C++ code to implement stack with following operations using
array.

a) create ()=Create a stack.
b) push()=Pushing (storing) an element on the stack
c) pop()=Removing (accessing) an element from the stack.
d) peek()=Get the top data element of the stack,without removing it
e) isFull()=Check ifstack is full.
f) isEmpty()=Check whether the stack is empty,and return true or false.

Note:--
(i) Create a separate function for each of the operations defined above.
(ii) Define the stack as follows.

#define MAXSIZE 100
struct stack{
int stArr[MAXSIZE];
int top;
};
typedef struct stack STACK;

*/

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

#define MAXSIZE 100

typedef struct stack{
    int stArr[MAXSIZE];
    int top;
}STACK;

void create(STACK *s){
    s->top=-1;
}

int isFull(STACK *s){
    return s->top==MAXSIZE-1;
}

int isEmpty(STACK *s){
    return s->top==-1;
}

void push(STACK *s,int value){
    if(isFull(s)){
        printf("Stack overflow! Cannot push %d\n",value);
        return;
```

```c
void push(STACK *s,int value){
    if(isFull(s)){
        printf("Stack overflow! Cannot push %d\n",value);
        return;
    }
    s->stArr[++s->top]=value;
    printf("Pushed %d onto stack\n",value);
}

int pop(STACK *s){
    if(isEmpty(s)){
        printf("Stack underflow! Cannot pop\n");
        return -1;
    }
    return s->stArr[s->top--];
}

int peek(STACK *s){
    if(isEmpty(s)){
        printf("Stack is empty! No top element\n");
        return -1;
    }
    return s->stArr[s->top];
}

int main(){
    STACK s;
    // testing
    create(&s);
    push(&s,10);
    push(&s,20);
    push(&s,30);
    printf("Top element is %d\n",peek(&s));
    printf("Popped %d from stack\n",pop(&s));
    printf("Popped %d from stack\n",pop(&s));
    printf("Stack is empty: %s\n",isEmpty(&s) ? "Yes" : "No");
    return 0;
}
```

```
Pushed 10 onto stack
Pushed 20 onto stack
Pushed 30 onto stack
Top element is 30
Popped 30 from stack
Popped 20 from stack
Stack is empty: No
```

C 5_3_stack_using_linkedlist.c > main()

```c
/* Write a C/C++ code to implement stack with all the operations defined in
Q2 using Linked list. */

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

typedef struct node{
    int data;
    struct node *next;
}node;

typedef struct stack{
    node *top;
}STACK;

void create(STACK *s){
    s->top=NULL;
}

int isEmpty(STACK *s){
    return s->top == NULL;
}

void push(STACK *s,int value){
    node *newnode=(node*)malloc(sizeof(node));
    if(!newnode){
        printf("Heap overflow! Cannot push %d\n",value);
        return;
    }
    newnode->data=value;
    newnode->next=s->top;
    s->top=newnode;
    printf("Pushed %d onto stack\n",value);
}

int pop(STACK *s){
    if(isEmpty(s)){
        printf("Stack underflow! Cannot pop\n");
        return -1;
    }
    node *temp=s->top;
    int poppedValue=temp->data;
    s->top=s->top->next;
    free(temp);
    return poppedValue;
}

```

```c
int peek(STACK *s){
    if(isEmpty(s)){
        printf("Stack is empty! No top element\n");
        return -1;
    }
    return s->top->data;
}

int main(){
    STACK s;
    // testing
    create(&s);
    push(&s,10);
    push(&s,20);
    push(&s,30);
    printf("Top element is %d\n",peek(&s));
    printf("Popped %d from stack\n",pop(&s));
    printf("Popped %d from stack\n",pop(&s));
    printf("Stack is empty: %s\n",isEmpty(&s) ? "Yes" : "No");
    return 0;
}
```

```
Pushed 10 onto stack
Pushed 20 onto stack
Pushed 30 onto stack
Top element is 30
Popped 30 from stack
Popped 20 from stack
Stack is empty: No
```