```c
#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node* next;

};

struct Node* addToEmpty(struct Node* last, int data) {

if (last != NULL) return last;

// allocate memory to the new node

struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

// assign data to the new node

newNode->data = data;

// assign last to newNode

last = newNode;

// create link to iteself

last->next = last;

return last;

}

// add node to the front

struct Node* addFront(struct Node* last, int data) {

// check if the list is empty

if (last == NULL) return addToEmpty(last, data);
```

```
// allocate memory to the new node
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
// add data to the node
newNode->data = data;
// store the address of the current first node in the newNode
newNode->next = last->next;
// make newNode as head
last->next = newNode;
return last;
}
// add node to the end
struct Node* addEnd(struct Node* last, int data) {
// check if the node is empty
if (last == NULL) return addToEmpty(last, data);
// allocate memory to the new node
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
// add data to the node
newNode->data = data;
// store the address of the head node to next of newNode
newNode->next = last->next;
// point the current last node to the newNode
last->next = newNode;
// make newNode as the last node
last = newNode;
```

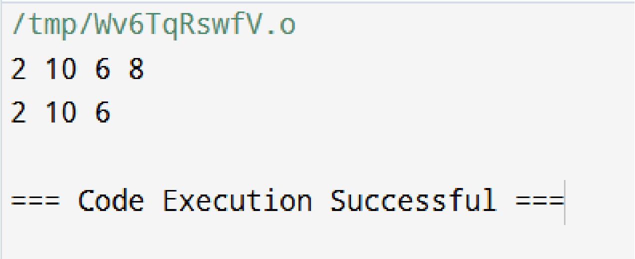
```
return last;
}
// insert node after a specific node
struct Node* addAfter(struct Node* last, int data, int item) {
// check if the list is empty
if (last == NULL) return NULL;
struct Node *newNode, *p;
p = last->next;
do {
// if the item is found, place newNode after it
if (p->data == item) {
// allocate memory to the new node
newNode = (struct Node*)malloc(sizeof(struct Node));
// add data to the node
newNode->data = data;
// make the next of the current node as the next of newNode
newNode->next = p->next;
// put newNode to the next of p
p->next = newNode;
// if p is the last node, make newNode as the last node
if (p == last) last = newNode;
return last;
}
p = p->next;
```

```
} while (p != last->next);
printf("¥nThe given node is not present in the list");
return last;
}
// delete a node
void deleteNode(struct Node** last, int key) {
// if linked list is empty
if (*last == NULL) return;
// if the list contains only a single node
if ((*last)->data == key && (*last)->next == *last) {
free(*last);
*last = NULL;
return;
}
struct Node *temp = *last, *d;
// if last is to be deleted
if ((*last)->data == key) {

// find the node before the last node
while (temp->next != *last) temp = temp->next;
// point temp node to the next of last i.e. first node
temp->next = (*last)->next;
free(*last);
*last = temp;
```

```
}
// travel to the node to be deleted
while (temp->next != *last && temp->next->data != key) {
temp = temp->next;
}
// if node to be deleted was found
if (temp->next->data == key) {
d = temp->next;
temp->next = d->next;
free(d);
}
}
void traverse(struct Node* last) {
struct Node* p;
if (last == NULL) {
printf("The list is empty");
return;
}
p = last->next;
do {
printf("%d ", p->data);
p = p->next;
} while (p != last->next);
}
```

```
int main() {
struct Node* last = NULL;
last = addToEmpty(last, 6);
last = addEnd(last, 8);
last = addFront(last, 2);
last = addAfter(last, 10, 2);
traverse(last);
deleteNode(&last, 8);
printf("¥n");
traverse(last);
return 0;
}
```



```
/tmp/Wv6TqRswfV.o
2 10 6 8
2 10 6

=== Code Execution Successful ===
```