

```
// Linked list operations in C
```

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

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

```
// Create a node
```

```
struct Node {
```

```
    int data;
```

```
    struct Node* next;
```

```
};
```

```
// Insert at the beginning
```

```
void insertAtBeginning(struct Node** head_ref, int new_data) {
```

```
    // Allocate memory to a node
```

```
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
```

```
    // insert the data
```

```
    new_node->data = new_data;
```

```
    new_node->next = (*head_ref);
```

```
    // Move head to new node
```

```
    (*head_ref) = new_node;
```

```
}
```

```
// Insert a node after a node
```

```
void insertAfter(struct Node* prev_node, int new_data) {
```

```
    if (prev_node == NULL) {
```

```
        printf("the given previous node cannot be NULL");
```

```
        return;
```

```
    }
```

```
struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));  
new_node->data = new_data;  
new_node->next = prev_node->next;  
prev_node->next = new_node;  
}
```

// Insert the the end

```
void insertAtEnd(struct Node** head_ref, int new_data) {  
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));  
    struct Node* last = *head_ref;  
  
    new_node->data = new_data;  
    new_node->next = NULL;  
  
    if (*head_ref == NULL) {  
        *head_ref = new_node;  
        return;  
    }
```

```
    while (last->next != NULL) last = last->next;
```

```
    last->next = new_node;  
    return;  
}
```

```
void deleteNode(struct Node** head_ref, int key) {  
    struct Node *temp = *head_ref, *prev;  
  
    if (temp != NULL && temp->data == key) {  
        *head_ref = temp->next;
```

```
free(temp);
```

```
return;
```

```
}
```

```
while (temp != NULL && temp->data != key) {
```

```
    prev = temp;
```

```
    temp = temp->next;
```

```
}
```

```
if (temp == NULL) return
```

```
prev->next = temp->next;
```

```
free(temp);
```

```
}
```

```
int searchNode(struct Node** head_ref, int key) {
```

```
    struct Node* current = *head_ref;
```

```
    while (current != NULL) {
```

```
        if (current->data == key) return 1;
```

```
        current = current->next;
```

```
    }
```

```
    return 0;
```

```
}
```

```
void sortLinkedList(struct Node** head_ref) {
```

```
    struct Node *current = *head_ref, *index = NULL;
```

```
    int temp;
```

```
    if (head_ref == NULL) {
```

```

return;

} else {
while (current != NULL) {

    index = current->next;

    while (index != NULL) {
        if (current->data > index->data) {
            temp = current->data;
            current->data = index->data;
            index->data = temp;
        }
        index = index->next;
    }
    current = current->next;
}
}
}

```

```

void printList(struct Node* node) {
    while (node != NULL) {
        printf(" %d ", node->data);
        node = node->next;
    }
}

```

```

int main() {
    struct Node* head = NULL;

    insertAtEnd(&head, 1);

```

```

insertAtBeginning(&head, 2);

insertAtBeginning(&head, 3);

insertAtEnd(&head, 4);

insertAfter(head->next, 5);


printf("Linked list: ");

printList(head);


printf("\nAfter deleting an element: ");

deleteNode(&head, 3);

printList(head);


int item_to_find = 3;

if (searchNode(&head, item_to_find)) {

printf("\n%d is found", item_to_find);

} else {

printf("\n%d is not found", item_to_find);

}

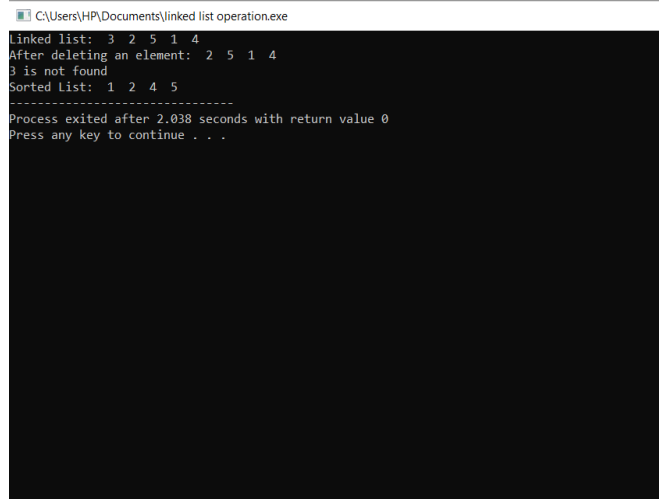

sortLinkedList(&head);

printf("\nSorted List: ");

printList(head);

}

```



```

C:\Users\HP\Documents\linked list operation.exe
Linked list: 3 2 5 1 4
After deleting an element: 2 5 1 4
3 is not found
Sorted List: 1 2 4 5
-----
Process exited after 2.038 seconds with return value 0
Press any key to continue . . .

```