#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node \* next;

};

void linkedListTraversal(struct Node \*ptr)

{

while (ptr != NULL)

{

printf("Element: %d\n", ptr->data);

ptr = ptr->next;

}

}

// Case 1

struct Node \* insertAtFirst(struct Node \*head, int data){

struct Node \* ptr = (struct Node \*) malloc(sizeof(struct Node));

ptr->data = data;

ptr->next = head;

return ptr;

}

// Case 2

struct Node \* insertAtIndex(struct Node \*head, int data, int index){

struct Node \* ptr = (struct Node \*) malloc(sizeof(struct Node));

struct Node \* p = head;

int i = 0;

while (i!=index-1)

{

p = p->next;

i++;

}

ptr->data = data;

ptr->next = p->next;

p->next = ptr;

return head;

}

// Case 3

struct Node \* insertAtEnd(struct Node \*head, int data){

struct Node \* ptr = (struct Node \*) malloc(sizeof(struct Node));

ptr->data = data;

struct Node \* p = head;

while(p->next!=NULL){

p = p->next;

}

p->next = ptr;

ptr->next = NULL;

return head;

}

// Case 4

struct Node \* insertAfterNode(struct Node \*head, struct Node \*prevNode, int data){

struct Node \* ptr = (struct Node \*) malloc(sizeof(struct Node));

ptr->data = data;

ptr->next = prevNode->next;

prevNode->next = ptr;

return head;

}

int main(){

struct Node \*head;

struct Node \*second;

struct Node \*third;

struct Node \*fourth;

// Allocate memory for nodes in the linked list in Heap

head = (struct Node \*)malloc(sizeof(struct Node));

second = (struct Node \*)malloc(sizeof(struct Node));

third = (struct Node \*)malloc(sizeof(struct Node));

fourth = (struct Node \*)malloc(sizeof(struct Node));

// Link first and second nodes

head->data = 7;

head->next = second;

// Link second and third nodes

second->data = 11;

second->next = third;

// Link third and fourth nodes

third->data = 41;

third->next = fourth;

// Terminate the list at the third node

fourth->data = 66;

fourth->next = NULL;

printf("Linked list before insertion\n");

linkedListTraversal(head);

// head = insertAtFirst(head, 56);

// head = insertAtIndex(head, 56, 1);

// head = insertAtEnd(head, 56);

head = insertAfterNode(head, third, 45);

printf("\nLinked list after insertion\n");

linkedListTraversal(head);

return 0;

}

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

void linkedListTraversal(struct Node \*ptr)

{

while (ptr != NULL)

{

printf("Element: %d\n", ptr->data);

ptr = ptr->next;

}

}

// Case 1: Deleting the first element from the linked list

struct Node \* deleteFirst(struct Node \* head){

struct Node \* ptr = head;

head = head->next;

free(ptr);

return head;

}

// Case 2: Deleting the element at a given index from the linked list

struct Node \* deleteAtIndex(struct Node \* head, int index){

struct Node \*p = head;

struct Node \*q = head->next;

for (int i = 0; i < index-1; i++)

{

p = p->next;

q = q->next;

}

p->next = q->next;

free(q);

return head;

}

// Case 3: Deleting the last element

struct Node \* deleteAtLast(struct Node \* head){

struct Node \*p = head;

struct Node \*q = head->next;

while(q->next !=NULL)

{

p = p->next;

q = q->next;

}

p->next = NULL;

free(q);

return head;

}

// Case 4: Deleting the element with a given value from the linked list

struct Node \* deleteAtIndex(struct Node \* head, int value){

struct Node \*p = head;

struct Node \*q = head->next;

while(q->data!=value && q->next!= NULL)

{

p = p->next;

q = q->next;

}

if(q->data == value){

p->next = q->next;

free(q);

}

return head;

}

int main()

{

struct Node \*head;

struct Node \*second;

struct Node \*third;

struct Node \*fourth;

// Allocate memory for nodes in the linked list in Heap

head = (struct Node \*)malloc(sizeof(struct Node));

second = (struct Node \*)malloc(sizeof(struct Node));

third = (struct Node \*)malloc(sizeof(struct Node));

fourth = (struct Node \*)malloc(sizeof(struct Node));

// Link first and second nodes

head->data = 4;

head->next = second;

// Link second and third nodes

second->data = 3;

second->next = third;

// Link third and fourth nodes

third->data = 8;

third->next = fourth;

// Terminate the list at the third node

fourth->data = 1;

fourth->next = NULL;

printf("Linked list before deletion\n");

linkedListTraversal(head);

// head = deleteFirst(head); // For deleting first element of the linked list

// head = deleteAtIndex(head, 2);

head = deleteAtLast(head);

printf("Linked list after deletion\n");

linkedListTraversal(head);

return 0;

}

Deletion

#include <stdio.h>

#include <stdlib.h>

Struct Node

{

Int data;

Struct Node \*next;

};

Void linkedListTraversal(struct Node \*ptr)

{

While (ptr != NULL)

{

Printf(“Element: %d\n”, ptr->data);

Ptr = ptr->next;

}

}

// Case 1: Deleting the first element from the linked list

Struct Node \* deleteFirst(struct Node \* head){

Struct Node \* ptr = head;

Head = head->next;

Free(ptr);

Return head;

}

// Case 2: Deleting the element at a given index from the linked list

Struct Node \* deleteAtIndex(struct Node \* head, int index){

Struct Node \*p = head;

Struct Node \*q = head->next;

For (int I = 0; I < index-1; i++)

{

P = p->next;

Q = q->next;

}

p->next = q->next;

free(q);

return head;

}

// Case 3: Deleting the last element

Struct Node \* deleteAtLast(struct Node \* head){

Struct Node \*p = head;

Struct Node \*q = head->next;

While(q->next !=NULL)

{

P = p->next;

Q = q->next;

}

p->next = NULL;

free(q);

return head;

}

// Case 4: Deleting the element with a given value from the linked list

Struct Node \* deleteAtIndex(struct Node \* head, int value){

Struct Node \*p = head;

Struct Node \*q = head->next;

While(q->data!=value && q->next!= NULL)

{

P = p->next;

Q = q->next;

}

If(q->data == value){

p->next = q->next;

free(q);

}

Return head;

}

Int main()

{

Struct Node \*head;

Struct Node \*second;

Struct Node \*third;

Struct Node \*fourth;

// Allocate memory for nodes in the linked list in Heap

Head = (struct Node \*)malloc(sizeof(struct Node));

Second = (struct Node \*)malloc(sizeof(struct Node));

Third = (struct Node \*)malloc(sizeof(struct Node));

Fourth = (struct Node \*)malloc(sizeof(struct Node));

// Link first and second nodes

Head->data = 4;

Head->next = second;

// Link second and third nodes

Second->data = 3;

Second->next = third;

// Link third and fourth nodes

Third->data = 8;

Third->next = fourth;

// Terminate the list at the third node

Fourth->data = 1;

Fourth->next = NULL;

Printf(“Linked list before deletion\n”);

linkedListTraversal(head);

// head = deleteFirst(head); // For deleting first element of the linked list

// head = deleteAtIndex(head, 2);

Head = deleteAtLast(head);

Printf(“Linked list after deletion\n”);

linkedListTraversal(head);

return 0;

}