LINKED LIST IN DATA STRUCTURE

**1.Insertion in linked list using function:**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node \*next;

};

void traversal(struct Node \*ptr){

while(ptr!=NULL){

printf("The data of the linked list : %d\n", ptr->data);

ptr=ptr->next;

}

}

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

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

ptr->data=data;

ptr->next=head;

return ptr;

}

int insertAtSpecefication(struct Node \*prev, int data){

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

temp->data=data;

temp->next=prev->next;

prev->next=temp;

}

struct Node insertEnd(struct Node \*lastnode, int data){

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

temp->data=data;

temp->next=NULL;

lastnode->next=temp;

}

int main(){

struct Node \*head;

struct Node \*Node1=( struct Node \*)malloc(sizeof( struct Node ));

struct Node \*Node2=( struct Node \*)malloc(sizeof( struct Node ));

struct Node \*Node3=( struct Node \*)malloc(sizeof( struct Node ));

Node1->data=10;

Node2->data=15;

Node3->data=25;

Node1->next=Node2;

Node2->next=Node3;

Node3->next=NULL;

traversal(Node1);

head=insertAtFirst(Node1,5);

insertAtSpecefication(Node2,20);

insertEnd(Node3,30);

traversal(head);

}

**2.Insertion in linked list without function:**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node \*next;

};

void display(struct Node \*ptr){

while(ptr!=NULL){

printf("The linked list: %d\n",ptr->data);

ptr= ptr->next;

}

}

int main(){

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

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

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

head->data=10;

middle->data=15;

tail->data=25;

head->next=middle;

middle->next=tail;

tail->next=NULL;

struct Node \*new\_node=(struct Node\*)malloc(sizeof(struct Node));

new\_node->data=5;

new\_node->next=head;

head=new\_node;

struct Node \*new\_middle\_node=(struct Node\*)malloc(sizeof(struct Node));

new\_middle\_node->data=20;

new\_middle\_node->next=tail;

middle->next=new\_middle\_node;

struct Node \*last\_node=(struct Node\*)malloc(sizeof(struct Node));

last\_node->data=30;

last\_node->next=NULL;

tail->next=last\_node;

display(head);

}

**3.Summation and count of linked list**

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node \*next;

};

void traversal(struct Node \*ptr){

while(ptr!=NULL){

printf("The data of the linked list : %d\n", ptr->data);

ptr=ptr->next;

}

}

void summation(struct Node \*ptr){

int sum=0;

while(ptr!=NULL){

sum=sum+ptr->data;

ptr=ptr->next;

}

printf("The sum of this linked list is:%d\n",sum);

}

void count(struct Node \*ptr){

int count=0;

while(ptr!=NULL){

count++;

ptr=ptr->next;

}

printf("The count of the linked list : %d\n",count);

}

int main(){

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

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

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

head->data=10;

middle->data=15;

tail->data=20;

head->next=middle;

middle->next=tail;

tail->next=NULL;

summation(head);

count(head);

}

**4. Deletion of a linked list**

#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;

}

}

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

struct Node \* ptr = head;

head = head->next;

free(ptr);

return head;

}

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;

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));

head->data = 4;

head->next = second;

second->data = 3;

second->next = third;

third->data = 8;

third->next = fourth;

fourth->data = 1;

fourth->next = NULL;

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

linkedListTraversal(head);

head = deleteAtLast(head);

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

linkedListTraversal(head);

return 0;

**}**

**5. Update a linked list**

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a node in the linked list

struct Node {

int data;

struct Node\* next;

};

// Function to update the value of a node at a given position

void update(struct Node\* head, int position, int newValue) {

int currentPosition = 0;

while (head != NULL) {

if (currentPosition == position) {

head->data = newValue;

return;

}

head = head->next;

currentPosition++;

}

printf("Position %d not found in the list.\n", position);

}

// Function to print the linked list

void display(struct Node\* head) {

while (head != NULL) {

printf("%d -> ", head->data);

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

// Creating a linked list with 5 nodes

for (int i = 1; i <= 5; i++) {

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

newNode->data = i \* 10;

newNode->next = head;

head = newNode;

}

printf("Original Linked List: ");

display(head);

// Update the value at position 2 to 35

update(head, 2, 35);

printf("Updated Linked List: ");

display(head);

return 0;

}

**6. Index finding in Linked List**

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a node in the linked list

struct Node {

int data;

struct Node\* next;

};

// Function to find the index of a value in the linked list

int findIndex(struct Node\* head, int target) {

int index = 0;

while (head != NULL) {

if (head->data == target) {

return index;

}

head = head->next;

index++;

}

// Value not found in the list

return -1;

}

// Function to print the linked list

void display(struct Node\* head) {

while (head != NULL) {

printf("%d -> ", head->data);

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

// Creating a linked list with values 10, 20, 30, 40, and 50

for (int i = 1; i <= 5; i++) {

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

newNode->data = i \* 10;

newNode->next = head;

head = newNode;

}

printf("Linked List: ");

display(head);

int target = 30;

int index = findIndex(head, target);

if (index != -1) {

printf("The value %d is found at index %d.\n", target, index);

} else {

printf("The value %d is not found in the list.\n", target);

}

return 0;

}

**7. Revers in Linked List**

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a node in the linked list

struct Node {

int data;

struct Node\* next;

};

// Function to create a new node

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to insert a node at the beginning of the linked list

void insertAtBeginning(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;

}

// Function to reverse the linked list

void reverseList(struct Node\*\* head) {

struct Node\* prev = NULL;

struct Node\* current = \*head;

struct Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*head = prev;

}

// Function to print the linked list

void display(struct Node\* head) {

while (head != NULL) {

printf("%d -> ", head->data);

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

// Insert some nodes at the beginning

insertAtBeginning(&head, 10);

insertAtBeginning(&head, 20);

insertAtBeginning(&head, 30);

insertAtBeginning(&head, 40);

insertAtBeginning(&head, 50);

printf("Original Linked List: ");

display(head);

// Reverse the linked list

reverseList(&head);

printf("Reversed Linked List: ");

display(head);

return 0;

}

**8. Palindrome in linked List**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

// Define a structure for a node in the linked list

struct Node {

char data;

struct Node\* next;

};

// Function to create a new node

struct Node\* createNode(char data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to insert a node at the beginning of the linked list

void insertAtBeginning(struct Node\*\* head, char data) {

struct Node\* newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;

}

// Function to reverse a linked list

void reverseList(struct Node\*\* head) {

struct Node\* prev = NULL;

struct Node\* current = \*head;

struct Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

\*head = prev;

}

// Function to check if a linked list is a palindrome

bool isPalindrome(struct Node\* head) {

struct Node\* original = head;

// Reverse the linked list

reverseList(&head);

// Compare the original and reversed linked lists

while (original != NULL && head != NULL) {

if (original->data != head->data) {

return false; // Not a palindrome

}

original = original->next;

head = head->next;

}

return true; // Palindrome

}

// Function to print the linked list

void display(struct Node\* head) {

while (head != NULL) {

printf("%c -> ", head->data);

head = head->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

// Insert characters into the linked list

insertAtBeginning(&head, 'r');

insertAtBeginning(&head, 'a');

insertAtBeginning(&head, 'c');

insertAtBeginning(&head, 'e');

insertAtBeginning(&head, 'c');

insertAtBeginning(&head, 'a');

insertAtBeginning(&head, 'r');

printf("Linked List: ");

display(head);

if (isPalindrome(head)) {

printf("The linked list is a palindrome.\n");

} else {

printf("The linked list is not a palindrome.\n");

}

return 0;

}