**NAME:SAYALI JIVAN CHAUDHARI**

**ROLL NO.:14**

**PRN NO.2023015400005055**

**1)implementation of singly linked list**

**//insert at head in singly linked list**

#include <iostream>

using namespace std;

class node {

public:

int data;

node\* next;

node(int d){

data = d;

next = NULL;

}

};

void insertAthead(node\*& head, int data)

{

node\* n = new node(data);

n->next = head;

head = n;

}

void print(node\* head)

{

while (head != NULL) {

cout << head->data << "->";

head = head->next;

}

}

int main()

{

node\* head = NULL;

insertAthead(head, 5);

insertAthead(head, 2);

insertAthead(head, 8);

insertAthead(head, 3);

print(head);

}

**//insert at middle**

**// C++ implementation to insert node at the middle**

**// of the linked list**

#include <bits/stdc++.h>

using namespace std;

// structure of a node

struct Node {

int data;

Node\* next;

};

// function to create and return a node

Node\* getNode(int data)

{

// allocating space

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

// inserting the required data

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// function to insert node at the middle

// of the linked list

void insertAtMid(Node\*\* head\_ref, int x)

{

// if list is empty

if (\*head\_ref == NULL)

\*head\_ref = getNode(x);

else {

// get a new node

Node\* newNode = getNode(x);

// assign values to the slow and fast

// pointers

Node\* slow = \*head\_ref;

Node\* fast = (\*head\_ref)->next;

while (fast && fast->next) {

// move slow pointer to next node

slow = slow->next;

// move fast pointer two nodes at a time

fast = fast->next->next;

}

// insert the 'newNode' and adjust the

// required links

newNode->next = slow->next;

slow->next = newNode;

}

}

// function to display the linked list

void display(Node\* head)

{

while (head != NULL) {

cout << head->data << " ";

head = head->next;

}

}

// Driver program to test above

int main()

{

// Creating the list 1->2->4->5

Node\* head = NULL;

head = getNode(1);

head->next = getNode(2);

head->next->next = getNode(4);

head->next->next->next = getNode(5);

cout << "Linked list before insertion: ";

display(head);

int x = 3;

insertAtMid(&head, x);

cout << "

Linked list after insertion: ";

display(head);

return 0;

}

**//insert at tail**

**// C++ program to demonstrate inserting a node**

**// at the end of a Linked List**

#include <bits/stdc++>

using namespace std;

// A linked list node

class Node {

public:

int data;

Node\* next;

};

// Given a reference (pointer to pointer)

// to the head of a list and an int, inserts

// a new node at the front of the list.

void push(Node\*\* head\_ref, int new\_data)

{

// Create a new node

Node\* new\_node = new Node();

new\_node->data = new\_data;

// Make the new node point to the current head

new\_node->next = (\*head\_ref);

// Update the head to point to the new node

(\*head\_ref) = new\_node;

}

// Given a reference (pointer to pointer)

// to the head of a list and an int,

// appends a new node at the end

void append(Node\*\* head\_ref, int new\_data)

{

// Create a new node

Node\* new\_node = new Node();

new\_node->data = new\_data;

// Store the head reference in a temporary variable

Node\* last = \*head\_ref;

// Set the next pointer of the new node as NULL since it

// will be the last node

new\_node->next = NULL;

// If the Linked List is empty, make the new node as the

// head and return

if (\*head\_ref == NULL) {

\*head\_ref = new\_node;

return;

}

// Else traverse till the last node

while (last->next != NULL) {

last = last->next;

}

// Change the next pointer of the last node to point to

// the new node

last->next = new\_node;

}

// This function prints the contents of

// the linked list starting from the head

void printList(Node\* node)

{

while (node != NULL) {

cout << " " << node->data;

node = node->next;

}

}

// Driver code

int main()

{

// Start with an empty list

Node\* head = NULL;

// Insert nodes at the beginning of the linked list

push(&head, 6);

push(&head, 5);

push(&head, 4);

push(&head, 3);

push(&head, 2);

cout << "Created Linked list is: ";

printList(head);

// Insert 1 at the end

append(&head, 1);

cout << "\nAfter inserting 1 at the end: ";

printList(head);

return 0;

}

**// C++ program to delete a node at any position**

**// singly linked list recursively**

#include <bits/stdc++>

using namespace std;

struct node {

int info;

node\* link = NULL;

node() {}

node(int a)

: info(a)

{

}

};

// Deletes the node containing 'info'

// part as val and alter the head of

// the linked list (recursive method)

void deleteNode(node\*& head, int val)

{

// Check if list is empty or we

// reach at the end of the

// list.

if (head == NULL) {

cout << "Element not present in the list\n";

return;

}

// If current node is the

// node to be deleted

if (head->info == val) {

node\* t = head;

// If it's start of the node head

// node points to second node

head = head->link;

// Else changes previous node's

// link to current node's link

delete (t);

return;

}

deleteNode(head->link, val);

}

// Utility function to add a

// node in the linked list

// Here we are passing head by

// reference thus no need to

// return it to the main function

void push(node\*& head, int data)

{

node\* newNode = new node(data);

newNode->link = head;

head = newNode;

}

// Utility function to print

// the linked list (recursive

// method)

void print(node\* head)

{

// cout<<endl gets implicitly

// typecasted to bool value

// 'true'

if (head == NULL and cout << endl)

return;

cout << head->info << ' ';

print(head->link);

}

int main()

{

// Starting with an empty linked list

node\* head = NULL;

// Adds new element at the

// beginning of the list

push(head, 10);

push(head, 12);

push(head, 14);

push(head, 15);

// original list

print(head);

// Call to delete function

deleteNode(head, 20);

// 20 is not present thus no change

// in the list

print(head);

deleteNode(head, 10);

print(head);

deleteNode(head, 14);

print(head);

return 0;

}