**LINKED LIST:**

**INSERTION AT BEGINNING**

SOURCE CODE:

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

void insertStart (struct Node \*\*head, int data)

{

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

newNode->data = data;

newNode->next = \*head;

//changing the new head to this freshly entered node

\*head = newNode;

}

void display (struct Node \*node)

{

//as linked list will end when Node is Null

while (node != NULL)

{

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

node = node->next;

}

printf ("\n");

}

int main ()

{

//creating 4 pointers of type struct Node

//So these can point to address of struct type variable

struct Node \*head = NULL;

struct Node \*node2 = NULL;

struct Node \*node3 = NULL;

struct Node \*node4 = NULL;

// allocate 3 nodes in the heap

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

node2 = (struct Node \*) malloc (sizeof (struct Node));

node3 = (struct Node \*) malloc (sizeof (struct Node));

node4 = (struct Node \*) malloc (sizeof (struct Node));

head->data = 15; // data set for head node

head->next = node2; // next pointer assigned to address of node2

node2->data = 10;

node2->next = node3;

node3->data = 12;

node3->next = node4;

node4->data = 3;

node4->next = NULL;

printf ("Linklist : ");

display (head);

// Need '&' i.e. address as we need to change head

insertStart (&head, 25);

printf ("\nAfter Inserting Element\n");

printf ("\nLinklist : ");

// no need for '&' as head need not be changed

// only doing traversal

display (head);

return 0;

}

OUTPUT:

Linklist : 15 10 12 3

After Inserting Element

Linklist : 25 15 10 12 3

INSERTION AT ENDING

SOURCE CODE:

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

void insertStart (struct Node \*\*head, int data)

{

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

newNode->data = data;

newNode->next = \*head;

//changing the new head to this freshly entered node

\*head = newNode;

}

void insertLast (struct Node \*\*head, int data)

{

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

newNode->data = data;

newNode->next = NULL;

//need this if there is no node present in linked list at all

if (\*head == NULL)

{

\*head = newNode;

return;

}

struct Node \*temp = \*head;

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

void display (struct Node \*node)

{

//as linked list will end when Node is Null

while (node != NULL)

{

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

node = node->next;

}

printf ("\n");

}

int main ()

{

struct Node \*head = NULL;

// Need '&' i.e. address as we need to change head

insertStart (&head, 12);

insertStart (&head, 16);

insertStart (&head, 20);

insertLast (&head, 10);

insertLast (&head, 14);

insertLast (&head, 18);

insertLast (&head, 11);

// no need for '&' as head need not be changed

// only doing traversal

display (head);

return 0;

}

OUTPUT:

20 16 12 10 14 18 11

INSERTION AT MIDDLE:

SOURCE CODE:

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

int calcSize (struct Node \*node)

{

int size = 0;

while (node != NULL)

{

node = node->next;

size++;

}

return size;

}

void insertPosition (int pos, int data, struct Node \*\*head)

{

int size = calcSize (\*head);

//If pos is 0 then should use insertStart method

//If pos is less than 0 then can't enter at all

//If pos is greater than size then bufferbound issue

if (pos < 1 || size < pos)

{

printf ("Can't insert, %d is not a valid position\n", pos);

}

else

{

struct Node \*temp = \*head;

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

newNode->data = data;

newNode->next = NULL;

while (--pos)

{

temp = temp->next;

}

//(25)->next = 10 as 12->next is 10

newNode->next = temp->next;

// (12)->next = 25

temp->next = newNode;

//new node added in b/w 12 and 10

}

}

void insertStart (struct Node \*\*head, int data)

{

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

newNode->data = data;

newNode->next = \*head;

//changing the new head to this freshly entered node

\*head = newNode;

}

void insertLast (struct Node \*\*head, int data)

{

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

newNode->data = data;

newNode->next = NULL;

//need this if there is no node present in linked list at all

if (\*head == NULL)

{

\*head = newNode;

return;

}

struct Node \*temp = \*head;

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

void display (struct Node \*node)

{

//as linked list will end when Node is Null

while (node != NULL)

{

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

node = node->next;

}

printf ("\n");

}

int main ()

{

struct Node \*head = NULL;

// Need '&' i.e. address as we need to change head

insertStart (&head, 12);

insertStart (&head, 16);

insertStart (&head, 20);

insertLast (&head, 10);

insertLast (&head, 14);

insertLast (&head, 18);

insertLast (&head, 11);

//Inserts after 3rd position

insertPosition (3, 25, &head);

// no need for '&' as head need not be changed

// only doing traversal

display (head);

return 0;

}

OUTPUT:

20 16 12 25 10 14 18 11

DELETION AT BEGINNING, MIDDLE, END

SOURCE CODE:

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

void delete (struct Node \*\*head, int delVal)

{

struct Node \*temp = \*head;

struct Node \*previous;

//Case when there is only 1 node in the list

if (temp->next == NULL)

{

\*head = NULL;

free (temp);

printf ("Value %d, deleted \n", delVal);

return;

}

//if the head node itself needs to be deleted

if (temp != NULL && temp->data == delVal)

{

//Case 1 head becomes 30

\*head = temp->next; //changing head to next in the list

printf ("Value %d, deleted \n", delVal);

//case 1: 22 deleted and freed

free (temp);

return;

}

//run until we find the value to be deleted in the list

while (temp != NULL && temp->data != delVal)

{

//store previous link node as we need to change its next val

previous = temp;

temp = temp->next;

}

//if value is not present then

//temp will have traversed to last node NULL

if (temp == NULL)

{

printf ("Value not found\n");

return;

}

// Case 2: (24)->next = 16 (as 20->next = 16)

// Case 3: (16)->next = NULL (as 12->next = NULL)

previous->next = temp->next;

free (temp);

//case 2: 20 deleted and freed

//case 3: 12 deleted and freed

printf ("Value %d, deleted \n", delVal);

}

void display (struct Node \*node)

{

//as linked list will end when Node is Null

while (node != NULL)

{

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

node = node->next;

}

printf ("\n");

}

int main ()

{

//creating 4 pointers of type struct Node

//So these can point to address of struct type variable

struct Node \*head = NULL;

struct Node \*node2 = NULL;

struct Node \*node3 = NULL;

struct Node \*node4 = NULL;

struct Node \*node5 = NULL;

struct Node \*node6 = NULL;

// allocate 3 nodes in the heap

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

node2 = (struct Node \*) malloc (sizeof (struct Node));

node3 = (struct Node \*) malloc (sizeof (struct Node));

node4 = (struct Node \*) malloc (sizeof (struct Node));

node5 = (struct Node \*) malloc (sizeof (struct Node));

node6 = (struct Node \*) malloc (sizeof (struct Node));

head->data = 22; // data set for head node

head->next = node2; // next pointer assigned to address of node2

node2->data = 30;

node2->next = node3;

node3->data = 24;

node3->next = node4;

node4->data = 20;

node4->next = node5;

node5->data = 16;

node5->next = node6;

node6->data = 12;

node6->next = NULL;

/\*No need for & i.e. address as we do not

need to change head address

\*/

printf ("Linked List Before Operations : ");

display (head);

//deleting first occurance of a value in linked list

delete (&head, 22);

delete (&head, 20);

delete (&head, 12);

printf ("Linked List After Operations : ");

display (head);

return 0;

}

OUTPUT:

Linked List Before Operations : 22 30 24 20 16 12

Value 22, deleted

Value 20, deleted

Value 12, deleted

Linked List After Operations : 30 24 16

**REVERSING A LINKED LIST**

SOURCE CODE:

#include <stdio.h>

struct Node {

   int data;

   struct Node\* next;

};

Node\* insertNode(int key) {

   Node\* temp = new Node;

   temp->data = key;

   temp->next = NULL;

   return temp;

}

void tailRecRevese(Node\* current, Node\* previous, Node\*\* head){

   if (!current->next) {

      \*head = current;

      current->next = previous;

      return;

   }

   Node\* next = current->next;

   current->next = previous;

   tailRecRevese(next, current, head);

}

void tailRecReveseLL(Node\*\* head){

   if (!head)

      return;

   tailRecRevese(\*head, NULL, head);

}

void printLinkedList(Node\* head){

   while (head != NULL) {

      printf("%d ", head->data);

      head = head->next;

   }

   printf("  
");

}

int main(){

   Node\* head1 = insertNode(9);

   head1->next = insertNode(32);

   head1->next->next = insertNode(65);

   head1->next->next->next = insertNode(10);

   head1->next->next->next->next = insertNode(85);

   printf("Linked list : \t");

   printLinkedList(head1);

   tailRecReveseLL(&head1);

   printf("Reversed linked list : \t");

   printLinkedList(head1);

   return 0;

}

OUTPUT:

Linked list : 9 32 65 10 85

Reversed linked list : 85 10 65 32 9

**DISPLAY THE ELEMENTS OF LINKED LIST**

SOURCE CODE:

#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*next;

} \*first = NULL;

void create(int A[], int n)

{

int i;

struct Node \*t, \*last;

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

first->data = A[0];

first->next = NULL;

last = first;

for (i = 1; i < n; i++)

{

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

t->data = A[i];

t->next = NULL;

last->next = t;

last = t;

}

}

void Display(struct Node \*p)

{

while (p != NULL)

{

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

p = p->next;

}

}

int main()

{

struct Node \*temp;

int A[] = { 3, 5, 7, 10, 25, 8, 32, 2 };

create(A, 8);

Display (first);

return 0;

}

OUTPUT: 3 5 7 10 25 8 32 2

**Write a program to check if a given singly linked list is a palindrome. A linked list is a palindrome**

**if the elements read the same backward as forward.**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \*next;

};

void insertFirst (struct Node \*\*head, int data)

{

// dynamically create memory for this newNode

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

// assign data value

newNode->data = data;

// change the next node of this newNode

// to current head of Linked List

newNode->next = \*head;

//re-assign head to this newNode

\*head = newNode;

}

void display (struct Node \*node)

{

printf ("Linked List : \n");

// as linked list will end when Node is Null

while (node != NULL)

{

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

node = node->next;

}

printf ("\n");

}

int size (struct Node \*node)

{

int counter=0;

// as linked list will end when Node is Null

while (node != NULL)

{

node = node->next;

counter++;

}

return counter;

}

int checkPalindrome (struct Node \*head, int counter)

{

int i = 0, j;

struct Node \*front, \*rear;

while (i != counter / 2)

{

front = rear = head;

for (j = 0; j < i; j++)

{

front = front->next;

}

for (j = 0; j < counter - (i + 1); j++)

{

rear = rear->next;

}

if (front->data != rear->data)

{

return 0;

}

else

{

i++;

}

}

return 1;

}

int main ()

{

struct Node \*head = NULL;

int counter,result;

// Need '&' i.e. address as we need to change head

insertFirst (&head, 20);

insertFirst (&head, 30);

insertFirst (&head, 40);

insertFirst (&head, 30);

insertFirst (&head, 20);

// no need of '&' as we are not changing head just displaying Linked List

display (head);

counter = size(head);

result = checkPalindrome (head, counter);

if (result == 1)

{

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

}

else

{

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

}

return 0;

}

OUTPUT:

Linked List :

20 30 40 30 20

The linked list is a palindrome.

**DETECT CYCLE IN LINKED LIST**

#include <stdio.h>

#include <stdbool.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\* newNode(int data) {

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

node->data = data;

node->next = NULL;

return node;

}

// Function to detect cycle and find the starting node of the cycle

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

if (head == NULL || head->next == NULL) {

return NULL; // No cycle if empty list or only one node

}

struct Node \*slow = head, \*fast = head;

// Move slow and fast pointers until they meet

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

// Cycle detected

break;

}

}

if (fast == NULL || fast->next == NULL) {

return NULL; // No cycle if fast pointer reaches end (no meeting point)

}

// Move slow to the head and move both slow and fast one step at a time until they meet again

slow = head;

while (slow != fast) {

slow = slow->next;

fast = fast->next;

}

return slow; // Return the starting node of the cycle

}

// Function to create a cycle in the linked list for testing

void createCycle(struct Node\* head, int pos) {

struct Node\* tail = head;

struct Node\* cycleNode = NULL;

int count = 1;

// Traverse to the end node and keep track of the node to form a cycle

while (tail->next != NULL) {

if (count == pos) {

cycleNode = tail;

}

tail = tail->next;

count++;

}

// Create the cycle

tail->next = cycleNode;

}

// Function to print the linked list

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("NULL\n");

}

// Function to free memory allocated to the linked list

void freeList(struct Node\* head) {

struct Node\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp);

}

}

// Main function

int main() {

struct Node\* head = newNode(1);

head->next = newNode(2);

head->next->next = newNode(3);

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

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

// Create a cycle for testing (connecting the last node to the second node)

createCycle(head, 2);

struct Node\* cycleStart = detectCycle(head);

if (cycleStart != NULL) {

printf("Cycle detected in the linked list.\n");

printf("Starting node of the cycle: %d\n", cycleStart->data);

} else {

printf("No cycle detected in the linked list.\n");

}

freeList(head); // Free allocated memory

return 0;

}

OUTPUT:

Cycle detected in the linked list.

Starting node of the cycle: 2

free(): double free detected in tcache 2

Aborted

**MERGE TWO LINKED LISTS**

#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\* newNode(int data) {

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

node->data = data;

node->next = NULL;

return node;

}

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

void insert(struct Node\*\* headRef, int data) {

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

newNode->data = data;

newNode->next = NULL;

if (\*headRef == NULL) {

\*headRef = newNode;

return;

}

struct Node\* temp = \*headRef;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

// Function to merge two sorted linked lists

struct Node\* mergeSortedLists(struct Node\* list1, struct Node\* list2) {

if (list1 == NULL) {

return list2;

}

if (list2 == NULL) {

return list1;

}

struct Node\* merged = NULL;

if (list1->data <= list2->data) {

merged = list1;

merged->next = mergeSortedLists(list1->next, list2);

} else {

merged = list2;

merged->next = mergeSortedLists(list1, list2->next);

}

return merged;

}

// Function to print the linked list

void printList(struct Node\* head) {

while (head != NULL) {

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

head = head->next;

}

printf("NULL\n");

}

// Function to free memory allocated to the linked list

void freeList(struct Node\* head) {

struct Node\* temp;

while (head != NULL) {

temp = head;

head = head->next;

free(temp);

}

}

// Main function

int main() {

// Creating and merging two sorted lists

struct Node\* list1 = NULL;

struct Node\* list2 = NULL;

insert(&list1, 1);

insert(&list1, 3);

insert(&list1, 5);

insert(&list2, 2);

insert(&list2, 4);

insert(&list2, 6);

printf("List 1: ");

printList(list1);

printf("List 2: ");

printList(list2);

struct Node\* mergedList = mergeSortedLists(list1, list2);

printf("Merged Sorted List: ");

printList(mergedList);

freeList(mergedList); // Free allocated memory

return 0;

}

OUTPUT:

List 1: 1 -> 3 -> 5 -> NULL

List 2: 2 -> 4 -> 6 -> NULL

Merged Sorted List: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL