1.INSERTION OF ELEMENTS IN AN ARRAY

#include <stdio.h>

void main() {

int arr[100];

int n, i, element, position;

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

printf("Enter %d elements: ", n);

for (i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

printf("Enter the element to be inserted: ");

scanf("%d", &element);

printf("Enter the position at which the element is to be inserted: ");

scanf("%d", &position);

for (i = n; i >= position; i--) {

arr[i] = arr[i - 1];

}

arr[position - 1] = element;

n++;

printf("Array after insertion: ");

for (i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

2.DELETION OF ELEMENTS FROM AN ARRAY

#include <stdio.h>

void main() {

int arr[100];

int n, i, position;

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

printf("Enter %d elements: ", n);

for (i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

printf("Enter the position of the element to be deleted: ");

scanf("%d", &position);

if (position < 1 || position > n) {

printf("Invalid position!\n");

} else {

for (i = position - 1; i < n - 1; i++) {

arr[i] = arr[i + 1];

}

n--;

printf("Array after deletion: ");

for (i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

}

3.REVERSAL OF AN ARRAY

#include <stdio.h>

void main() {

int arr[100]; // Array of size 100

int n, i;

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

printf("Enter %d elements: ", n);

for (i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

for (i = 0; i < n / 2; i++) {

int temp = arr[i];

arr[i] = arr[n - i - 1];

arr[n - i - 1] = temp;

}

printf("Array after reversing: ");

for (i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

}

4.SUM OF TWO MATRICES

#include <stdio.h>

int main() {

int r, c, i, j;

int m1[100][100], m2[100][100], sum[100][100];

printf("Enter the number of rows: ");

scanf("%d", &r);

printf("Enter the number of columns: ");

scanf("%d", &c);

printf("Enter elements of the first matrix:\n");

for (i = 0; i < r; i++) {

for (j = 0; j < c; j++) {

scanf("%d", &m1[i][j]);

} }

printf("Enter elements of the second matrix:\n");

for (i = 0; i < r; i++) {

for (j = 0; j < c; j++) {

scanf("%d", &m2[i][j]);

} }

for (i = 0; i < r; i++) {

for (j = 0; j < c; j++) {

sum[i][j] = m1[i][j] + m2[i][j];

} }

printf("Sum of the two matrices:\n");

for (i = 0; i < r; i++) {

for (j = 0; j < c; j++) {

printf("%d ", sum[i][j]); }

printf("\n"); }

5.MULTIPLICATION OF TWO MATRICES

#include <stdio.h>

void main() {

int m1[100][100], m2[100][100], result[100][100];

int r1, c1, r2, c2;

int i, j, k;

printf("Enter the number of rows and columns for the first matrix: ");

scanf("%d %d", &r1, &c1);

printf("Enter the number of rows and columns for the second matrix: ");

scanf("%d %d", &r2, &c2);

if (cols1 != rows2) {

printf("Matrix multiplication is not possible.");

} else {

printf("Enter elements of the first matrix:\n");

for (i = 0; i < r1; i++) {

for (j = 0; j < c1; j++) {

scanf("%d", &m1[i][j]);

}}

printf("Enter elements of the second matrix:\n");

for (i = 0; i < r2; i++) {

for (j = 0; j < c2; j++) {

scanf("%d", &m2[i][j]);

}}

for (i = 0; i < r1; i++) {

for (j = 0; j < c2; j++) {

result[i][j] = 0;}}

for (i = 0; i < r1; i++) {

for (j = 0; j < c2; j++) {

for (k = 0; k < c1; k++) {

result[i][j] += m1[i][k] \* m2[k][j];} } }

printf("Resulting matrix after multiplication:\n");

for (i = 0; i < r1; i++) {

for (j = 0; j < c2; j++) {

printf("%d ", result[i][j]);

}

printf("\n");

}

}}

6.TRANSPOSE OF A MATRIX

#include <stdio.h>

int main() {

int matrix[100][100], transpose[100][100];

int r, c;

int i, j;

printf("Enter the number of rows and columns of the matrix: ");

scanf("%d %d", &r, &c);

printf("Enter elements of the matrix:\n");

for (i = 0; i < r; i++) {

for (j = 0; j < c; j++) {

printf("Element [%d][%d]: ", i + 1, j + 1);

scanf("%d", &matrix[i][j]);

}

}

for (i = 0; i < r; i++) {

for (j = 0; j < c; j++) {

transpose[j][i] = matrix[i][j];

}

} printf("Transpose of the matrix:\n");

for (i = 0; i < c; i++) {

for (j = 0; j < r; j++) {

printf("%d ", transpose[i][j]);

}

printf("\n");

}

}

8 LinkedList Traversing

#include<stdlib.h>

struct Node

{

int data;

struct Node\* next;

};

struct Node\* createNode(int new\_data)

{ struct Node\* new\_node =

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

new\_node->data = new\_data;

new\_node->next = NULL;

return new\_node; }

void traverseList(struct Node\* head)

{

while (head != NULL)

{

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

head = head->next;

}

printf("\n");

}

int main()

{

struct Node\* head = createNode(10);

head->next = createNode(20);

head->next->next = createNode(30);

head->next->next->next = createNode(40);

traverseList(head);

    return 0;

 }

9LinkedList Insertion

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node\*next;

};

struct Node\*createnode(int new\_data)

{

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

new\_node->data=new\_data;

new\_node->next=NULL;

return new\_node;

}

struct Node\*insertatfront(struct Node\*head, int new\_data)

{

struct Node\*new\_node=createnode(new\_data);

new\_node->next=head;

return new\_node;

}

void printlist(struct Node\*head)

{

struct Node\*curr=head;

while(curr!=NULL)

{

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

curr=curr->next;

}

printf("\n");

}

int main()

{

struct Node\*head=createnode(21);

head->next=createnode(27);

head->next->next=createnode(42);

head->next->next->next=createnode(26);

printf("original link list");

printlist(head);

printf("after inserting node in front");

int data=90;

head=insertatfront(head,data);

printlist(head);

    return 0;

}

10 LinkedList Delete\_

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node\* next;

};

struct Node\* newNode(int data)

{

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

node->data=data;

node->next=NULL;

return node;

}

struct Node\* deleteNode(struct Node\* head, int position)

{

struct Node\* temp = head;

struct Node\* prev = NULL;

if (temp == NULL)

return head;

if (position == 1)

{

head = temp->next;

free(temp);

return head;

}

for (int i = 1; temp != NULL && i < position; i++)

{

prev = temp;

temp = temp->next;

}

if (temp != NULL)

{

prev->next = temp->next;

free(temp);

}

else

{

printf("Data not present\n");

}

return head;

}

void printList(struct Node\* head)

{

while (head != NULL)

{

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

head = head->next;

}

printf("NULL\n");

}

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);

printf("Original List: ");

printList(head);

int position = 4;

head = deleteNode(head, position);

printf("List after deletion: ");

printList(head);

while (head != NULL)

{

struct Node\* temp = head;

head = head->next;

free(temp);

}

return 0;

}

11 LinkedList Search

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct Node

{

int data;

struct Node\* next;

};

struct Node\* createNode(int new\_data)

{

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

new\_node->data = new\_data;

new\_node->next = NULL;

return new\_node;

}

bool searchKey(struct Node\* head, int key)

{

struct Node\* curr = head;

while (curr != NULL)

{

if (curr->data == key)

return true;

curr = curr->next;

}

return false;

}

int main()

{

struct Node\* head = createNode(3);

head->next = createNode(3);

head->next->next = createNode(10);

head->next->next->next = createNode(42);

head->next->next->next->next = createNode(54);

int key;

printf("Enter the element to search: ");

scanf("%d", &key);

if(searchKey(head, key))

printf("Yes \n");

else

printf("No \n");

return 0;

}

12LinkedList Node

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node\*next;

};

struct Node\*createnode(int new\_data)

{

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

new\_node->data=new\_data;

new\_node->next=NULL;

return new\_node;

}

struct Node\*insertatfront(struct Node\*head,int new\_data)

{

struct Node\*new\_node=createnode(new\_data);

new\_node->next=head;

return new\_node;

}

void printlist(struct Node\*head)

{

struct Node\*curr=head;

while(curr!=NULL)

{

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

curr=curr->next;

}

printf("\n");

}

int main()

{

struct Node\*head = createnode(2);

head->next = createnode(3);

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

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

printf("original link list");

printlist(head);

printf("After inserting node in front");

int data;

printf("Enter the Data to Insert: ");

scanf("%d", &data);

head=insertatfront(head,data);

printlist(head);

return 0;

}

13 LinkedListTwoWay Insertion

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

new\_node->data = data;

new\_node->prev = NULL;

new\_node->next = NULL;

return new\_node;

}

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

struct Node\* new\_node = createNode(data);

new\_node->next = head;

if (head != NULL) {

head->prev = new\_node;

}

return new\_node;

}

void printList(struct Node\* head) {

struct Node\* curr= head;

while (curr != NULL) {

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

curr = curr->next;

}

printf("\n");

}

int main() {

int data;

struct Node \*head = createNode(2);

head->next = createNode(3); head->next->prev = head;

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

head->next->next->prev = head->next;

printf("Original Linked List: ");

printList(head);

printf("\nEnter the node to enter at the start: ");

scanf("%d", &data);

head = insertBegin(head, data);

printf("After inserting Node &d at the front: ", data);

printList(head);

return 0;

}

14 LinkedListTwoWay Deletion

#include <stdio.h>

#include <stdlib.h>

struct n {

int data;

struct n\* next;

struct n\* prev;

};

void deleten(struct n\*\* head, struct n\* del)

{

if (\*head == del) {

\*head = del->next;

}

if (del->next != NULL) {

del->next->prev = del->prev;

}

if (del->prev != NULL) {

del->prev->next = del->next;

}

free(del);

}

int main() {

struct n\* head = NULL;

struct n\* n1 = (struct n\*)malloc(sizeof(struct n));

n1->data = 1; n1->next = NULL;

n1->prev = NULL;

head = n1;

struct n\* n2 = (struct n\*)malloc(sizeof(struct n));

n2->data = 2;

n2->next = NULL;

n2->prev = n1;

n1->next = n2;

struct n\* n3 = (struct n\*)malloc(sizeof(struct n));

n3->data = 3;

n3->next = NULL;

n3->prev = n2;

n2->next = n3;

struct n\* n4 = (struct n\*)malloc(sizeof(struct n));

n4->data = 4;

n4->next = NULL;

n4->prev = n3;

n3->next = n4;

struct n\* n5 = (struct n\*)malloc(sizeof(struct n));

n5->data = 5;

n5->next = NULL;

n5->prev = n4;

n4->next = n5;

deleten (&head, n2);

struct n\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

return 0;

}