**1.write a c program code for single linked list.**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

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

if (!newNode) {

printf("Memory allocation error\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

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

struct Node\* newNode = createNode(data);

newNode->next = \*head;

\*head = newNode;

}

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

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

void insertAfter(struct Node\* prevNode, int data) {

if (prevNode == NULL) {

printf("The given previous node cannot be NULL\n");

return;

}

struct Node\* newNode = createNode(data);

newNode->next = prevNode->next;

prevNode->next = newNode;

}

void deleteNode(struct Node\*\* head, int key) {

struct Node\* temp = \*head;

struct Node\* prev = NULL;

if (temp != NULL && temp->data == key) {

\*head = temp->next;

free(temp);

return;

}

while (temp != NULL && temp->data != key) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) {

printf("Key not found\n");

return;

}

prev->next = temp->next;

free(temp);

}

struct Node\* searchNode(struct Node\* head, int key) {

struct Node\* current = head;

while (current != NULL) {

if (current->data == key) {

return current;

}

current = current->next;

}

return NULL;

}

void updateNode(struct Node\* head, int key, int newData) {

struct Node\* node = searchNode(head, key);

if (node != NULL) {

node->data = newData;

} else {

printf("Node with key %d not found\n", key);

}

}

void printList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

insertAtEnd(&head, 1);

insertAtEnd(&head, 2);

insertAtBeginning(&head, 0);

insertAfter(head->next, 3);

printf("Linked list:\n");

printList(head);

struct Node\* node = searchNode(head, 2);

if (node != NULL) {

printf("Node with data %d found\n", node->data);

} else {

printf("Node not found\n");

}

updateNode(head, 2, 4);

printf("List after updating node with data 2 to 4:\n");

printList(head);

deleteNode(&head, 3);

printf("List after deleting node with data 3:\n");

printList(head);

return 0;

}

**Sample output:**

Linked list of insert node:

0 -> 1 -> 3 -> 2 -> NULL

Node with data 2 found

List after updating node with data 2 to 4:

0 -> 1 -> 3 -> 4 -> NULL

List after deleting node with data 3:

0 -> 1 -> 4 -> NULL

**2.write a c programm code for double linked list:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

struct Node\* prev;

};

struct Node\* createNode(int data) {

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

if (!newNode) {

printf("Memory allocation error\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;

}

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

struct Node\* newNode = createNode(data);

newNode->next = \*head;

if (\*head != NULL) {

(\*head)->prev = newNode;

}

\*head = newNode;

}

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

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

void insertAfter(struct Node\* prevNode, int data) {

if (prevNode == NULL) {

printf("The given previous node cannot be NULL\n");

return;

}

struct Node\* newNode = createNode(data);

newNode->next = prevNode->next;

newNode->prev = prevNode;

if (prevNode->next != NULL) {

prevNode->next->prev = newNode;

}

prevNode->next = newNode;

}

void deleteNode(struct Node\*\* head, int key) {

struct Node\* temp = \*head;

while (temp != NULL && temp->data != key) {

temp = temp->next;

}

if (temp == NULL) {

printf("Key not found\n");

return;

}

if (\*head == temp) {

\*head = temp->next;

}

if (temp->next != NULL) {

temp->next->prev = temp->prev;

}

if (temp->prev != NULL) {

temp->prev->next = temp->next;

}

free(temp);

}

struct Node\* searchNode(struct Node\* head, int key) {

struct Node\* current = head;

while (current != NULL) {

if (current->data == key) {

return current;

}

current = current->next;

}

return NULL;

}

void updateNode(struct Node\* head, int key, int newData) {

struct Node\* node = searchNode(head, key);

if (node != NULL) {

node->data = newData;

} else {

printf("Node with key %d not found\n", key);

}

}

void printList(struct Node\* head) {

struct Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

void printListReverse(struct Node\* head) {

struct Node\* temp = head;

if (temp == NULL) {

return;

}

while (temp->next != NULL) {

temp = temp->next;

}

while (temp != NULL) {

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

temp = temp->prev;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

insertAtEnd(&head, 1);

insertAtEnd(&head, 2);

insertAtBeginning(&head, 0);

insertAfter(head->next, 3);

printf("Linked list (forward):\n");

printList(head);

printf("Linked list (reverse):\n");

printListReverse(head);

struct Node\* node = searchNode(head, 2);

if (node != NULL) {

printf("Node with data %d found\n", node->data);

} else {

printf("Node not found\n");

}

updateNode(head, 2, 4);

printf("List after updating node with data 2 to 4:\n");

printList(head);

deleteNode(&head, 3);

printf("List after deleting node with data 3:\n");

printList(head);

return 0;

}

**Sample output:**

Linked list (forward):

0 -> 1 -> 3 -> 2 -> NULL

Linked list (reverse):

2 -> 3 -> 1 -> 0 -> NULL

Node with data 2 found

List after updating node with data 2 to 4:

Node with data 2 found

List after updating node with data 2 to 4:

0 -> 1 -> 4 -> NULL

**3.write a cprogram code for circular single linked list.**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the circular singly 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));

if (!newNode) {

printf("Memory allocation error\n");

exit(1);

}

newNode->data = data;

newNode->next = newNode; // Pointing to itself to form a circular list

return newNode;

}

// Insert a node at the beginning

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

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

newNode->next = \*head;

temp->next = newNode;

\*head = newNode;

}

// Insert a node at the end

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

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = \*head;

}

// Insert a node after a given node

void insertAfter(struct Node\* prevNode, int data) {

if (prevNode == NULL) {

printf("The given previous node cannot be NULL\n");

return;

}

struct Node\* newNode = createNode(data);

newNode->next = prevNode->next;

prevNode->next = newNode;

}

// Delete a node by key

void deleteNode(struct Node\*\* head, int key) {

if (\*head == NULL) {

return;

}

struct Node \*temp = \*head, \*prev = NULL;

while (temp->data != key) {

if (temp->next == \*head) {

printf("Node with key %d not found\n", key);

return;

}

prev = temp;

temp = temp->next;

}

if (temp->next == \*head && prev == NULL) {

free(temp);

\*head = NULL;

return;

}

if (temp == \*head) {

prev = \*head;

while (prev->next != \*head) {

prev = prev->next;

}

\*head = temp->next;

prev->next = \*head;

free(temp);

} else if (temp->next == \*head) {

prev->next = \*head;

free(temp);

} else {

prev->next = temp->next;

free(temp);

}

}

// Search for a node by key

struct Node\* searchNode(struct Node\* head, int key) {

struct Node\* current = head;

if (head == NULL) {

return NULL;

}

do {

if (current->data == key) {

return current;

}

current = current->next;

} while (current != head);

return NULL;

}

// Update a node's data by key

void updateNode(struct Node\* head, int key, int newData) {

struct Node\* node = searchNode(head, key);

if (node != NULL) {

node->data = newData;

} else {

printf("Node with key %d not found\n", key);

}

}

// Function to print the circular singly linked list

void printList(struct Node\* head) {

struct Node\* temp = head;

if (head == NULL) {

printf("List is empty\n");

return;

}

do {

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

temp = temp->next;

} while (temp != head);

printf("%d (head)\n", head->data); // to indicate the circular nature

}

int main() {

struct Node\* head = NULL;

// Inserting nodes

insertAtEnd(&head, 1);

insertAtEnd(&head, 2);

insertAtBeginning(&head, 0);

insertAfter(head->next, 3);

// Printing the linked list

printf("Circular singly linked list:\n");

printList(head);

// Searching for a node

struct Node\* node = searchNode(head, 2);

if (node != NULL) {

printf("Node with data %d found\n", node->data);

} else {

printf("Node not found\n");

}

// Updating a node

updateNode(head, 2, 4);

printf("List after updating node with data 2 to 4:\n");

printList(head);

// Deleting a node

deleteNode(&head, 3);

printf("List after deleting node with data 3:\n");

printList(head);

return 0;

}

**Sample output:**

linked list insert:

0 -> 1 -> 3 -> 2 -> 0 (head) Node with data 2 found

List after updating node with data 2 to 4:

0 -> 1 -> 3 -> 4 -> 0 (head)

List after deleting node with data 3:

1. -> 1 -> 4 -> 0 (head)

**4.Write a C program to find Odd or Even number from a given set of numbers**

#include <stdio.h>

int main() {

int n, i, number;

// Read the number of elements

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

scanf("%d", &n);

// Read the elements and check if they are odd or even

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

printf("Enter number %d: ", i + 1);

scanf("%d", &number);

if (number % 2 == 0) {

printf("%d is Even\n", number);

} else {

printf("%d is Odd\n", number);

}

}

return 0;

}

**Sample output:**

Enter the number of elements: 2

Enter number 1: 3

3 is Odd

Enter number 2: 4

4 is Even

**5. Write a C program to implement Array operations such as Insert, Delete and Display.**

#include <stdio.h>

#define MAX\_SIZE 100 // Define maximum size of the array

void insert(int arr[], int \*n, int pos, int value) {

if (\*n >= MAX\_SIZE) {

printf("Array is full, cannot insert element.\n");

return;

}

if (pos < 0 || pos > \*n) {

printf("Invalid position.\n");

return;

}

for (int i = \*n; i > pos; i--) {

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

}

arr[pos] = value;

(\*n)++;

}

void delete(int arr[], int \*n, int pos) {

if (\*n <= 0) {

printf("Array is empty, cannot delete element.\n");

return;

}

if (pos < 0 || pos >= \*n) {

printf("Invalid position.\n");

return;

}

for (int i = pos; i < \*n - 1; i++) {

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

}

(\*n)--;

}

void display(int arr[], int n) {

if (n == 0) {

printf("Array is empty.\n");

return;

}

printf("Array elements: ");

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

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

}

printf("\n");

}

int main() {

int arr[MAX\_SIZE];

int n = 0; // Current size of the array

int choice, pos, value;

while (1) {

printf("\nArray Operations:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Display\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter the position to insert (0 to %d): ", n);

scanf("%d", &pos);

printf("Enter the value to insert: ");

scanf("%d", &value);

insert(arr, &n, pos, value);

break;

case 2:

printf("Enter the position to delete (0 to %d): ", n - 1);

scanf("%d", &pos);

delete(arr, &n, pos);

break;

case 3:

display(arr, n);

break;

case 4:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

**Sample output:**

Array Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter the position to insert (0 to 0): 0

Enter the value to insert: 10

Array Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 1

Enter the position to insert (0 to 1): 1

Enter the value to insert: 20

Array Operations:

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice: 3

Array elements: 10 20

5.Write a C program to perform Matrix Multiplication.

#include <stdio.h>

// Function to read a matrix

void readMatrix(int matrix[][10], int rows, int columns) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

printf("Enter element [%d][%d]: ", i, j);

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

}

}

}

// Function to print a matrix

void printMatrix(int matrix[][10], int rows, int columns) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < columns; j++) {

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

}

printf("\n");

}

}

// Function to multiply two matrices

void multiplyMatrices(int firstMatrix[][10], int secondMatrix[][10], int resultMatrix[][10], int row1, int col1, int row2, int col2) {

for (int i = 0; i < row1; i++) {

for (int j = 0; j < col2; j++) {

resultMatrix[i][j] = 0;

for (int k = 0; k < col1; k++) {

resultMatrix[i][j] += firstMatrix[i][k] \* secondMatrix[k][j];

}

}

}

}

int main() {

int row1, col1, row2, col2;

int firstMatrix[10][10], secondMatrix[10][10], resultMatrix[10][10];

// Input dimensions of the first matrix

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

scanf("%d %d", &row1, &col1);

// Input dimensions of the second matrix

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

scanf("%d %d", &row2, &col2);

// Check if matrix multiplication is possible

if (col1 != row2) {

printf("Matrix multiplication not possible. Number of columns in the first matrix must be equal to the number of rows in the second matrix.\n");

return -1;

}

// Read the first matrix

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

readMatrix(firstMatrix, row1, col1);

// Read the second matrix

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

readMatrix(secondMatrix, row2, col2);

// Multiply the matrices

multiplyMatrices(firstMatrix, secondMatrix, resultMatrix, row1, col1, row2, col2);

// Print the result

printf("Resultant Matrix:\n");

printMatrix(resultMatrix, row1, col2);

return 0;

}

**Sample output:**

Enter rows and columns for the first matrix: 2 3

Enter rows and columns for the second matrix: 3 2

Enter elements of the first matrix:

Enter element [0][0]: 1

Enter element [0][1]: 2

Enter element [0][2]: 3

Enter element [1][0]: 4

Enter element [1][1]: 5

Enter element [1][2]: 6

Enter elements of the second matrix:

Enter element [0][0]: 7

Enter element [0][1]: 8

Enter element [1][0]: 9

Enter element [1][1]: 10

Enter element [2][0]: 11

Enter element [2][1]: 12

Resultant Matrix:

58 64

139 154