**// Program to implement a Stack as a Linked List**

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a node

typedef struct node {

int info;

struct node \*next;

} Node;

/\* Function Prototypes \*/

void push(Node \*\*);

void pop(Node \*\*);

void display(Node \*);

int main() {

int choice;

Node \*Top = NULL; // Initialize stack as empty (Top = NULL)

do {

printf("\n Main Menu");

printf("\n1. Push");

printf("\n2. Pop");

printf("\n3. Exit");

printf("\n Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

push(&Top);

break;

case 2:

pop(&Top);

break;

case 3:

exit(0);

default:

printf("\n Invalid choice");

}

} while (1);

}

/\* Push function to insert elements into the stack \*/

void push(Node \*\*Top) {

int item;

Node \*nptr = (Node \*)malloc(sizeof(Node)); // Allocate memory for a new node

if (nptr == NULL) { // Check for memory allocation failure

printf("\n Overflow");

return;

}

printf("\n Enter the item to be inserted in the stack: ");

scanf("%d", &item);

nptr->info = item; // Store the value in the new node

nptr->next = \*Top; // Link the new node to the current top node

\*Top = nptr; // Update the top pointer to the new node

display(\*Top); // Display the updated stack

}

/\* Pop function to remove elements from the stack \*/

void pop(Node \*\*Top) {

int popped;

Node \*temp;

if (\*Top == NULL) { // Check if the stack is empty

printf("\n Underflow");

return;

popped = (\*Top)->info; // Get the value of the top element

temp = \*Top; // Temporarily store the top node

\*Top = (\*Top)->next; // Update the top pointer to the next node

free(temp); // Free the memory of the removed node

printf("\n Popped element is: %d", popped);

display(\*Top); // Display the updated stack

}

/\* Display function to print the elements in the stack \*/

void display(Node \*Top) {

Node \*temp;

if (Top == NULL) { // Check if the stack is empty

printf("\n Stack is empty!!");

return;

}

printf("\n The Stack is: ");

for (temp = Top; temp != NULL; temp = temp->next) { // Traverse the stack

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

}

 printf("\n");

}

**// Program to implement Queue as a Linked List**

#include <stdio.h>

#include <stdlib.h>

#define True 1

#define False 0

typedef struct node {

int info;

struct node \*next;

} Node;

typedef struct queue {

Node \*Front;

Node \*Rear;

} que;

void createqueue(que \*);

int isempty(que \*);

void qinsert(que \*, int);

int qdelete(que \*);

int main() {

que q;

int choice, val, element;

createqueue(&q);

do {

printf("\n\n\t Main Menu");

printf("\n1. Insert");

printf("\n2. Delete");

printf("\n3. Exit");

printf("\n Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\n Enter the value to be inserted: ");

scanf("%d", &element);

qinsert(&q, element);

break;

case 2:

val = qdelete(&q);

if (val == -1)

printf("\n Underflow - Queue is empty");

else

printf("\n Deleted item is: %d", val);

break;

case 3:

exit(0);

default:

printf("\n Invalid choice");

}

} while (1);

return 0;

}

/\* Initializes the queue \*/

void createqueue(que \*q) {

q->Front = NULL;

q->Rear = NULL;

}

/\* Checks if the queue is empty \*/

int isempty(que \*q) {

if (q->Front == NULL) {

return True;

} else {

return False;

}

}

/\* Inserts a value into the queue \*/

void qinsert(que \*q, int val) {

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

if (nptr == NULL) {

printf("\n Overflow - Unable to allocate memory");

return;

}

nptr->info = val;

nptr->next = NULL;

if (isempty(q)) {

q->Front = q->Rear = nptr;

} else {

q->Rear->next = nptr;

q->Rear = nptr;

}

printf("\n Value inserted in queue: %d", val);

}

/\* Deletes a value from the queue \*/

int qdelete(que \*q) {

if (isempty(q)) {

return -1; // Returning -1 to indicate underflow

}

int del\_val;

Node \*temp;

del\_val = q->Front->info;

temp = q->Front;

if (q->Front == q->Rear) {

q->Front = q->Rear = NULL; // Queue becomes empty

} else {

q->Front = q->Front->next;

}

free(temp); // Free the deleted node

return del\_val;

}

**//C program that demonstrate insertion & deletion cases on single linked list**

#include<stdio.h>

#include<stdlib.h>

typedef struct node {

int info;

struct node \*next;

} Node;

/\* Function Prototype \*/

Node \*create\_node();

void insert\_beg(Node \*\*);

void insert\_end(Node \*\*);

void insert\_pos(Node \*\*);

void delete\_beg(Node \*\*);

void delete\_end(Node \*\*);

void delete\_pos(Node \*\*);

void display(Node \*);

int count\_node(Node \*);

/\* Main function \*/

void main() {

int item, ch, ch1;

Node \*Start = NULL;

do {

printf("\n\t Main Menu");

printf("\n1 Insert");

printf("\n2 Delete");

printf("\n3 Display");

printf("\n4 Exit");

printf("\n Enter your choice: ");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("\n1. Insert at the beginning");

printf("\n2. Insert at the End");

printf("\n3. Insert at a specific position");

printf("\n4. Back to main menu");

printf("\n Enter your choice: ");

scanf("%d", &ch1);

switch (ch1) {

case 1: insert\_beg(&Start); break;

case 2: insert\_end(&Start); break;

case 3: insert\_pos(&Start); break;

case 4: break;

default: printf("\n Invalid choice");

}

break;

case 2:

printf("\n1. Delete from the beginning");

printf("\n2. Delete at the end");

printf("\n3. Delete from a specific position");

printf("\n4. Back to main menu");

printf("\n Enter your choice: ");

scanf("%d", &ch1);

switch (ch1) {

case 1: delete\_beg(&Start); break;

case 2: delete\_end(&Start); break;

case 3: delete\_pos(&Start); break;

case 4: break;

default: printf("\n Invalid choice");

}

break;

case 3:

display(Start);

break;

case 4:

exit(0);

default:

printf("\n Invalid choice");

}

} while (1);

}

/\* Create a new node \*/

Node \*create\_node() {

Node \*nptr;

int item;

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

if (nptr == NULL) {

printf("\n Overflow");

exit(1);

}

printf("\n Enter the value to be inserted: ");

scanf("%d", &item);

nptr->info = item;

nptr->next = NULL;

return nptr;

}

/\* Insert node at the beginning \*/

void insert\_beg(Node \*\*Start) {

Node \*nptr = create\_node();

if (\*Start != NULL)

nptr->next = \*Start;

\*Start = nptr;

printf("\n Node inserted at the beginning");

}

/\* Insert node at the end \*/

void insert\_end(Node \*\*Start) {

Node \*temp;

Node \*nptr = create\_node();

if (\*Start == NULL) {

\*Start = nptr;

} else {

temp = \*Start;

while (temp->next != NULL)

temp = temp->next;

temp->next = nptr;

}

printf("\n Node inserted at the end");

}

/\* Insert node at a specific position \*/

void insert\_pos(Node \*\*Start) {

int i, pos, count;

Node \*nptr = create\_node();

Node \*temp = \*Start;

printf("\n Enter the position: ");

scanf("%d", &pos);

count = count\_node(temp);

if (pos > count + 1 || pos == 0) {

printf("\n Invalid position");

return;

}

if (pos == 1) {

nptr->next = \*Start;

\*Start = nptr;

} else {

for (i = 1; i < pos - 1; i++)

temp = temp->next;

nptr->next = temp->next;

temp->next = nptr;

}

printf("\n Node inserted at position %d", pos);

}

/\* Delete node from the beginning \*/

void delete\_beg(Node \*\*Start) {

Node \*temp = \*Start;

if (\*Start == NULL) {

printf("Underflow");

return;

}

\*Start = temp->next;

free(temp);

printf("\n Node deleted from the beginning");

}

/\* Delete node from the end \*/

void delete\_end(Node \*\*Start) {

Node \*temp = \*Start;

Node \*save;

if (\*Start == NULL) {

printf("\n Underflow");

return;

}

if (temp->next == NULL) {

\*Start = NULL;

} else {

while (temp->next != NULL) {

save = temp;

temp = temp->next;

}

save->next = NULL;

}

free(temp);

printf("\n Node deleted from the end");

}

/\* Delete node from a specific position \*/

void delete\_pos(Node \*\*Start) {

Node \*temp = \*Start, \*save;

int i, pos, count;

if (\*Start == NULL) {

printf("\n Underflow");

return;

}

printf("\n Enter position of node to be deleted: ");

scanf("%d", &pos);

count = count\_node(temp);

if (pos > count || pos == 0) {

printf("\n Invalid position");

return;

}

if (pos == 1) {

\*Start = temp->next;

} else {

for (i = 1; i < pos; i++) {

save = temp;

temp = temp->next;

}

save->next = temp->next;

}

free(temp);

printf("\n Node deleted from position %d", pos);

}

/\* Display the list \*/

void display(Node \*Start) {

Node \*temp = Start;

if (temp == NULL) {

printf("\n List is empty");

return;

}

printf("\n Linked list: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

/\* Count the number of nodes \*/

int count\_node(Node \*Start) {

Node \*temp = Start;

int count = 0;

while (temp != NULL) {

count++;

temp = temp->next;

}

return count;

}

**//Program to perform various operations on Circular Linked**

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

    int info;

    struct node\* next;

} Node;

/\* Function Prototypes \*/

Node\* create\_node();

void insert\_end(Node\*\*);

void insert\_beg(Node\*\*);

void delete\_beg(Node\*\*);

void delete\_end(Node\*\*);

void display(Node\*);

int main() {

    int choice, sub\_choice;

    Node\* start = NULL;

    while (1) {

        printf("\n\*\*\* Circular Linked List Menu \*\*\*\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:  // Insertion Menu

                printf("\nInsertion Options:\n");

                printf("1. Insert at the beginning\n");

                printf("2. Insert at the end\n");

                printf("3. Back to main menu\n");

                printf("Enter your choice: ");

                scanf("%d", &sub\_choice);

                switch (sub\_choice) {

                    case 1: insert\_beg(&start); break;

                    case 2: insert\_end(&start); break;

                    case 3: break;

                    default: printf("Invalid choice! Returning to main menu.\n");

                }

                break;

            case 2:  // Deletion Menu

                printf("\nDeletion Options:\n");

                printf("1. Delete from the beginning\n");

                printf("2. Delete from the end\n");

                printf("3. Back to main menu\n");

                printf("Enter your choice: ");

                scanf("%d", &sub\_choice);

                switch (sub\_choice) {

                    case 1: delete\_beg(&start); break;

                    case 2: delete\_end(&start); break;

                    case 3: break;

                    default: printf("Invalid choice! Returning to main menu.\n");

                }

                break;

            case 3:

                display(start);

                break;

            case 4:

                printf("Exiting the program. Goodbye!\n");

                exit(0);

            default:

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

        }

    }

    return 0;

}

/\* Create a new node \*/

Node\* create\_node() {

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

    if (new\_node == NULL) {

        printf("Memory allocation failed. Exiting program.\n");

        exit(1);

    }

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

    scanf("%d", &new\_node->info);

    new\_node->next = NULL;

    return new\_node;

}

/\* Insert node at the beginning \*/

void insert\_beg(Node\*\* start) {

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

    if (\*start == NULL) {

        \*start = new\_node;

        new\_node->next = new\_node;  // Self-loop for the first node

    } else {

        Node\* temp = \*start;

        while (temp->next != \*start)

            temp = temp->next;

        new\_node->next = \*start;

        temp->next = new\_node;

        \*start = new\_node;

    }

    printf("Node inserted at the beginning.\n");

}

/\* Insert node at the end \*/

void insert\_end(Node\*\* start) {

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

    if (\*start == NULL) {

        \*start = new\_node;

        new\_node->next = new\_node;  // Self-loop for the first node

    } else {

        Node\* temp = \*start;

        while (temp->next != \*start)

            temp = temp->next;

        temp->next = new\_node;

        new\_node->next = \*start;

    }

    printf("Node inserted at the end.\n");

}

/\* Delete node from the beginning \*/

void delete\_beg(Node\*\* start) {

    if (\*start == NULL) {

        printf("List is empty. Cannot delete.\n");

        return;

    }

    Node\* temp = \*start;

    if (temp->next == \*start) {  // Single node case

        \*start = NULL;

    } else {

        Node\* last = \*start;

        while (last->next != \*start)

            last = last->next;

        \*start = temp->next;

        last->next = \*start;

    }

    free(temp);

    printf("Node deleted from the beginning.\n");

}

/\* Delete node from the end \*/

void delete\_end(Node\*\* start) {

    if (\*start == NULL) {

        printf("List is empty. Cannot delete.\n");

        return;

    }

    Node\* temp = \*start;

    Node\* prev = NULL;

    if (temp->next == \*start) {  // Single node case

        \*start = NULL;

    } else {

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

            prev = temp;

            temp = temp->next;

        }

        prev->next = \*start;

    }

    free(temp);

    printf("Node deleted from the end.\n");

}

/\* Display the list \*/

void display(Node\* start) {

    if (start == NULL) {

        printf("List is empty.\n");

        return;

    }

    Node\* temp = start;

    printf("Circular Linked List: ");

    do {

        printf("%d -> ", temp->info);

        temp = temp->next;

    } while (temp != start);

    printf("(back to start)\n");

}