# Singly 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 == NULL)

{

printf("Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

struct Node\* insertB(struct Node \*head, int data)

{

struct Node \*newNode = createNode(data);

newNode->next = head;

head = newNode;

return head;

}

struct Node\* insertE(struct Node \*head, int data)

{

struct Node \*newNode = createNode(data);

if (head == NULL)

{

head = newNode;

return head;

}

struct Node \*lastNode = head;

while (lastNode->next != NULL)

{

lastNode = lastNode->next;

}

lastNode->next = newNode;

return head;

}

void insertNodeAtN(struct Node\* head, int data, int position)

{

if (position <= 0)

{

printf("Invalid position\n");

return;

}

struct Node\* newNode = createNode(data);

if (position == 1)

{

newNode->next = head;

head = newNode;

return;

}

struct Node\* temp = head;

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

{

temp = temp->next;

}

if (temp == NULL)

{

printf("Invalid position\n");

return;

}

newNode->next = temp->next;

temp->next = newNode;

}

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

{

if (head == NULL) {

printf("List is empty\n");

return head;

}

if (position <= 0)

{

printf("Invalid position\n");

return head;

}

struct Node \*temp = head;

if (position == 1)

{

head = temp->next;

free(temp);

return head;

}

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

{

temp = temp->next;

}

if (temp == NULL || temp->next == NULL)

{

printf("Position out of range\n");

return head;

} else

{

struct Node \*nextNode = temp->next->next;

free(temp->next);

temp->next = nextNode;

return head;

}

}

struct Node\* search(struct Node \*head, int key)

{

struct Node \*current = head;

while (current != NULL) {

if (current->data == key)

{

return current;

}

current = current->next;

}

return NULL;

}

struct Node\* revlist(struct Node \*head)

{

struct Node \*prevNode = NULL;

struct Node \*currentNode = head;

struct Node \*nextNode = NULL;

while (currentNode != NULL)

{

nextNode = currentNode->next;

currentNode->next = prevNode;

prevNode = currentNode;

currentNode = nextNode;

}

head = prevNode;

return head;

}

void display(struct Node \*head)

{

struct Node \*temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

printf("\n");

}

int main()

{

struct Node \*head = NULL;

int choice, data, position;

do {

printf("\nMenu:\n");

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

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

printf("3. Insert at position\n");

printf("4. Delete at position\n");

printf("5. Search\n");

printf("6. Reverse\n");

printf("7. Display\n");

printf("8. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

head = insertB(head, data);

break;

case 2:

printf("Enter data to insert at end: ");

scanf("%d", &data);

head = insertE(head, data);

break;

case 3:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter position to insert: ");

scanf("%d", &position);

insertNodeAtN(head, data, position);

break;

case 4:

printf("Enter position to delete: ");

scanf("%d", &position);

head = deletion(head, position);

break;

case 5:

printf("Enter data to search: ");

scanf("%d", &data);

if (search(head, data))

printf("Data found in the list\n");

else

printf("Data not found in the list\n");

break;

case 6:

head = revlist(head);

printf("List reversed\n");

break;

case 7:

printf("Current list: ");

display(head);

break;

case 8:

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

break;

default:

printf("Invalid choice!\n");

}

} while (choice != 8);

return 0;

}

# Doubly linked list

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*prev;

struct Node \*next;

};

struct Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

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

struct Node \*newNode = createNode(data);

if (head == NULL) {

head = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

return head;

}

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

struct Node \*newNode = createNode(data);

if (head == NULL) {

head = newNode;

} else {

struct Node \*temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

return head;

}

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

if (position <= 0) {

printf("Invalid position\n");

return;

}

struct Node\* newNode = createNode(data);

if (position == 1) {

newNode->next = head;

if (head != NULL) {

head->prev = newNode;

}

head = newNode;

return;

}

struct Node\* temp = head;

int i;

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

temp = temp->next;

}

if (temp == NULL) {

printf("Invalid position\n");

return;

}

newNode->next = temp->next;

if (temp->next != NULL) {

temp->next->prev = newNode;

}

temp->next = newNode;

newNode->prev = temp;

}

struct Node\* deletion(struct Node \*head, int position) {

if (head == NULL) {

printf("List is empty\n");

return head;

}

if (position <= 0) {

printf("Invalid position\n");

return head;

}

struct Node \*temp = head;

if (position == 1) {

head = temp->next;

if (head != NULL) {

head->prev = NULL;

}

free(temp);

return head;

}

int i;

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

temp = temp->next;

}

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

printf("Position out of range\n");

return head;

} else {

struct Node \*nextNode = temp->next->next;

free(temp->next);

temp->next = nextNode;

if (nextNode != NULL) {

nextNode->prev = temp;

}

return head;

}

}

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

struct Node \*current = head;

while (current != NULL) {

if (current->data == key) {

return current;

}

current = current->next;

}

return NULL;

}

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

struct Node \*temp = NULL;

struct Node \*current = head;

while (current != NULL) {

temp = current->prev;

current->prev = current->next;

current->next = temp;

current = current->prev;

}

if (temp != NULL) {

head = temp->prev;

}

return head;

}

void display(struct Node \*head) {

struct Node \*temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

int main() {

struct Node \*head = NULL;

int choice, data, position;

do {

printf("\nMenu:\n");

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

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

printf("3. Insert at position\n");

printf("4. Delete at position\n");

printf("5. Search\n");

printf("6. Reverse\n");

printf("7. Display\n");

printf("8. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

head = insertB(head, data);

break;

case 2:

printf("Enter data to insert at end: ");

scanf("%d", &data);

head = insertE(head, data);

break;

case 3:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter position to insert: ");

scanf("%d", &position);

insertNodeAtN(head, data, position);

break;

case 4:

printf("Enter position to delete: ");

scanf("%d", &position);

head = deletion(head, position);

break;

case 5:

printf("Enter data to search: ");

scanf("%d", &data);

if (search(head, data))

printf("Data found in the list\n");

else

printf("Data not found in the list\n");

break;

case 6:

head = revlist(head);

printf("List reversed\n");

break;

case 7:

printf("Current list: ");

display(head);

break;

case 8:

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

break;

default:

printf("Invalid choice!\n");

}

} while (choice != 8);

return 0;

}

# CSLL

#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 == NULL) {

printf("Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

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

struct Node \*newNode = createNode(data);

if (head == NULL) {

newNode->next = newNode;

head = newNode;

} else {

struct Node \*temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

head = newNode;

}

return head;

}

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

struct Node \*newNode = createNode(data);

if (head == NULL) {

newNode->next = newNode;

head = newNode;

} else {

struct Node \*temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

}

return head;

}

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

if (position <= 0) {

printf("Invalid position\n");

return;

}

struct Node\* newNode = createNode(data);

if (position == 1) {

struct Node \*temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

head = newNode;

return;

}

struct Node\* temp = head;

int i;

for (i = 1; i < position - 1 && temp->next != head; i++) {

temp = temp->next;

}

if (temp->next == head && i < position - 1) {

printf("Invalid position\n");

return;

}

newNode->next = temp->next;

temp->next = newNode;

}

struct Node\* deletion(struct Node \*head, int position) {

if (head == NULL) {

printf("List is empty\n");

return head;

}

if (position <= 0) {

printf("Invalid position\n");

return head;

}

struct Node \*temp = head;

if (position == 1) {

struct Node \*temp2 = head;

while (temp2->next != head) {

temp2 = temp2->next;

}

if (temp->next == head) {

free(temp);

return NULL;

}

head = head->next;

temp2->next = head;

free(temp);

return head;

}

int i;

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

temp = temp->next;

}

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

printf("Position out of range\n");

return head;

}

struct Node \*nextNode = temp->next->next;

free(temp->next);

temp->next = nextNode;

return head;

}

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

struct Node \*current = head;

do {

if (current->data == key) {

return current;

}

current = current->next;

} while (current != head);

return NULL;

}

void display(struct Node \*head) {

if (head == NULL) {

printf("List is empty\n");

return;

}

struct Node \*temp = head;

do {

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

temp = temp->next;

} while (temp != head);

printf("\n");

}

int main() {

struct Node \*head = NULL;

int choice, data, position;

do {

printf("\nMenu:\n");

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

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

printf("3. Insert at position\n");

printf("4. Delete at position\n");

printf("5. Search\n");

printf("6. Display\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

head = insertB(head, data);

break;

case 2:

printf("Enter data to insert at end: ");

scanf("%d", &data);

head = insertE(head, data);

break;

case 3:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter position to insert: ");

scanf("%d", &position);

insertNodeAtN(head, data, position);

break;

case 4:

printf("Enter position to delete: ");

scanf("%d", &position);

head = deletion(head, position);

break;

case 5:

printf("Enter data to search: ");

scanf("%d", &data);

if (search(head, data))

printf("Data found in the list\n");

else

printf("Data not found in the list\n");

break;

case 6:

printf("Current list: ");

display(head);

break;

case 7:

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

break;

default:

printf("Invalid choice!\n");

}

} while (choice != 7);

return 0;

}

# CDLL

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node \*prev;

struct Node \*next;

};

struct Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed\n");

exit(EXIT\_FAILURE);

}

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

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

struct Node \*newNode = createNode(data);

if (head == NULL) {

newNode->next = newNode;

newNode->prev = newNode;

head = newNode;

} else {

struct Node \*last = head->prev;

newNode->next = head;

newNode->prev = last;

head->prev = newNode;

last->next = newNode;

head = newNode;

}

return head;

}

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

struct Node \*newNode = createNode(data);

if (head == NULL) {

newNode->next = newNode;

newNode->prev = newNode;

head = newNode;

} else {

struct Node \*last = head->prev;

newNode->next = head;

newNode->prev = last;

head->prev = newNode;

last->next = newNode;

}

return head;

}

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

if (position <= 0) {

printf("Invalid position\n");

return;

}

struct Node\* newNode = createNode(data);

if (position == 1) {

head = insertB(head, data);

return;

}

struct Node\* temp = head;

int i;

for (i = 1; i < position - 1 && temp->next != head; i++) {

temp = temp->next;

}

if (temp->next == head && i < position - 1) {

printf("Invalid position\n");

return;

}

newNode->next = temp->next;

newNode->prev = temp;

temp->next->prev = newNode;

temp->next = newNode;

}

struct Node\* deletion(struct Node \*head, int position) {

if (head == NULL) {

printf("List is empty\n");

return head;

}

if (position <= 0) {

printf("Invalid position\n");

return head;

}

if (position == 1) {

struct Node \*temp = head;

if (temp->next == temp) {

free(temp);

return NULL;

}

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

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

head = temp->next;

free(temp);

return head;

}

struct Node \*temp = head;

int i;

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

temp = temp->next;

}

if (temp == NULL) {

printf("Position out of range\n");

return head;

}

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

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

free(temp);

return head;

}

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

struct Node \*current = head;

do {

if (current->data == key) {

return current;

}

current = current->next;

} while (current != head);

return NULL;

}

void display(struct Node \*head) {

if (head == NULL) {

printf("List is empty\n");

return;

}

struct Node \*temp = head;

do {

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

temp = temp->next;

} while (temp != head);

printf("\n");

}

int main() {

struct Node \*head = NULL;

int choice, data, position;

do {

printf("\nMenu:\n");

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

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

printf("3. Insert at position\n");

printf("4. Delete at position\n");

printf("5. Search\n");

printf("6. Display\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at beginning: ");

scanf("%d", &data);

head = insertB(head, data);

break;

case 2:

printf("Enter data to insert at end: ");

scanf("%d", &data);

head = insertE(head, data);

break;

case 3:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter position to insert: ");

scanf("%d", &position);

insertNodeAtN(head, data, position);

break;

case 4:

printf("Enter position to delete: ");

scanf("%d", &position);

head = deletion(head, position);

break;

case 5:

printf("Enter data to search: ");

scanf("%d", &data);

if (search(head, data))

printf("Data found in the list\n");

else

printf("Data not found in the list\n");

break;

case 6:

printf("Current list: ");

display(head);

break;

case 7:

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

break;

default:

printf("Invalid choice!\n");

}

} while (choice != 7);

return 0;

}