**Ex no 3 a**

1. Start.
2. Define a struct Node for a doubly linked list.
3. Create a function createNode to allocate and initialize a new Node.
4. Create an insertEnd function to add a new Node to the end of the list.
5. Create a deleteEnd function to remove the last Node.
6. Create a function printList to display the list.
7. Initialize the list head as NULL in the main function.
8. Add four Nodes to the end of the list (1, 2, 3, 4).
9. Print the list.
10. Delete the last Node and print the updated 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 == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;

}

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

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

struct Node\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

}

}

void deleteEnd(struct Node\*\* head) {

if (\*head == NULL) {

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

return;

}

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

free(\*head);

\*head = NULL;

} else {

struct Node\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->prev->next = NULL;

free(current);

}

}

void printList(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

printf("%d <-> ", current->data);

current = current->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

insertEnd(&head, 1);

insertEnd(&head, 2);

insertEnd(&head, 3);

insertEnd(&head, 4);

printf("Doubly Linked List: ");

printList(head);

deleteEnd(&head);

printf("Doubly Linked List after deleting the last node: ");

printList(head);

return 0;

**Ex no 3 b**

Start.  
2. Define a struct Node for a doubly linked list with data, next, and prev pointers.  
3. Create a function createNode to allocate memory for a new Node and initialize its values.  
4. Create an insertEnd function to add a new Node to the end of the linked list.  
5. Check if the head is NULL, and if so, set it to the new Node; otherwise, traverse the list to find the end and insert the new Node.  
6. Create an isPrime function to determine if a given number is prime.  
7. Iterate through the linked list, summing the values of nodes that are prime.  
8. Print the doubly linked list.  
9. Print the sum of prime numbers.  
10. End.

#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 == NULL) {

printf("Memory allocation failed\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;

}

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

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

struct Node\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

}

}

int isPrime(int n) {

if (n <= 1) return 0;

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0) return 0;

}

return 1;

}

int sumOfPrimes(struct Node\* head) {

int sum = 0;

struct Node\* current = head;

while (current != NULL) {

if (isPrime(current->data)) {

sum += current->data;

}

current = current->next;

}

return sum;

}

void printList(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

printf("%d <-> ", current->data);

current = current->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

insertEnd(&head, 2);

insertEnd(&head, 3);

insertEnd(&head, 4);

insertEnd(&head, 5);

insertEnd(&head, 6);

insertEnd(&head, 7);

printf("Doubly Linked List: ");

printList(head);

int sum = sumOfPrimes(head);

printf("Sum of prime numbers: %d\n", sum);

return 0;

**Ex no 4**

1. Start.  
   2. Define a struct Term to represent a term in a polynomial with coefficient, exponent, and a next pointer.  
   3. Create a typedef to make "Term" synonymous with "struct Term".  
   4. Create a function createTerm to allocate memory for a new Term and initialize its values.  
   5. Create an addTerm function to add a new Term to the end of the polynomial.  
   6. Check if the polynomial is empty, and if so, set it to the new Term; otherwise, traverse the list to find the end and insert the new Term.  
   7. Create a function displayPolynomial to print the polynomial in a human-readable form.  
   8. Check if the polynomial is empty; if not, iterate through it, printing each term with coefficients and exponents.  
   9. End the polynomial printing with a new line.  
   10. In the main function, initialize a polynomial as NULL.  
   11. Add three terms to the polynomial with coefficients and exponents: 3x^4, -2x^2, and 5x^0.  
   12. Print the polynomial.  
   13. Iterate through the polynomial to free the allocated memory for each term.  
   14. End.

#include <stdio.h>

#include <stdlib.h>

struct Term {

int coefficient;

int exponent;

struct Term\* next;

};

typedef struct Term Term;

Term\* createTerm(int coefficient, int exponent) {

Term\* term = (Term\*)malloc(sizeof(Term));

term->coefficient = coefficient;

term->exponent = exponent;

term->next = NULL;

return term;

}

void addTerm(Term\*\* polynomial, int coefficient, int exponent) {

Term\* newTerm = createTerm(coefficient, exponent);

if (\*polynomial == NULL) {

\*polynomial = newTerm;

} else {

Term\* current = \*polynomial;

while (current->next != NULL) {

current = current->next;

}

current->next = newTerm;

}

}

void displayPolynomial(Term\* polynomial) {

if (polynomial == NULL) {

printf("Polynomial is empty\n");

return;

}

while (polynomial != NULL) {

printf("%dx^%d ", polynomial->coefficient, polynomial->exponent);

if (polynomial->next != NULL) {

printf("+ ");

}

polynomial = polynomial->next;

}

printf("\n");

}

int main() {

Term\* polynomial = NULL;

addTerm(&polynomial, 3, 4);

addTerm(&polynomial, -2, 2);

addTerm(&polynomial, 5, 0);

printf("Polynomial: ");

displayPolynomial(polynomial);

while (polynomial != NULL) {

Term\* temp = polynomial;

polynomial = polynomial->next;

free(temp);

}

return 0;

}