1.Write a C program to search for an element in the singly linked list by specifying the position and the item.

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 6
//PROGRAM 1
//Date : 17/02/2024

/*
1.Write a C program to search for an element in the singly linked list
by specifying the position and the item.
*/


#include <stdio.h>
#include <stdlib.h>

typedef struct node {
   int data;
   struct node *next;
} node;

void traverse(node *head) {
   int count = 1;
   node *p = head;
   while (p != NULL) {
       printf("\nNode_%d _data: %d", count, p->data);
       count++;
       p = p->next;
   }
}

void insert_begin(node **head, int item) {
   node *new_node = (node*) malloc(sizeof(node));
   new_node->data = item;
   new_node->next = *head;
   *head = new_node;
}
```

```
void insert_end(node **head, int item) {
   node *new_node = (node*) malloc(sizeof(node));
   new_node->data = item;
   new_node->next = NULL;
   if (*head == NULL) {
       *head = new_node;
       return;
   }
   node *p = *head;
   while(p->next != NULL) {
       p = p->next;
   }
   p->next = new_node;
}

void insert_any_position(node **head, int item, int pos) {
   node *new_node = (node*) malloc(sizeof(node));
   new_node->data = item;
   if (pos == 1) {
       new_node->next = *head;
       *head = new_node;
       return;
   }
   node *p = *head;
   for(int i = 1; i < pos - 1 && p != NULL; i++) {
       p = p->next;
   }
   if (p == NULL) {
       printf("Position out of range\n");
       return;
   }
   new_node->next = p->next;
   p->next = new_node;
}

void delete_begin(node **head) {
   if (*head == NULL) {
       printf("List is empty\n");
       return;
```

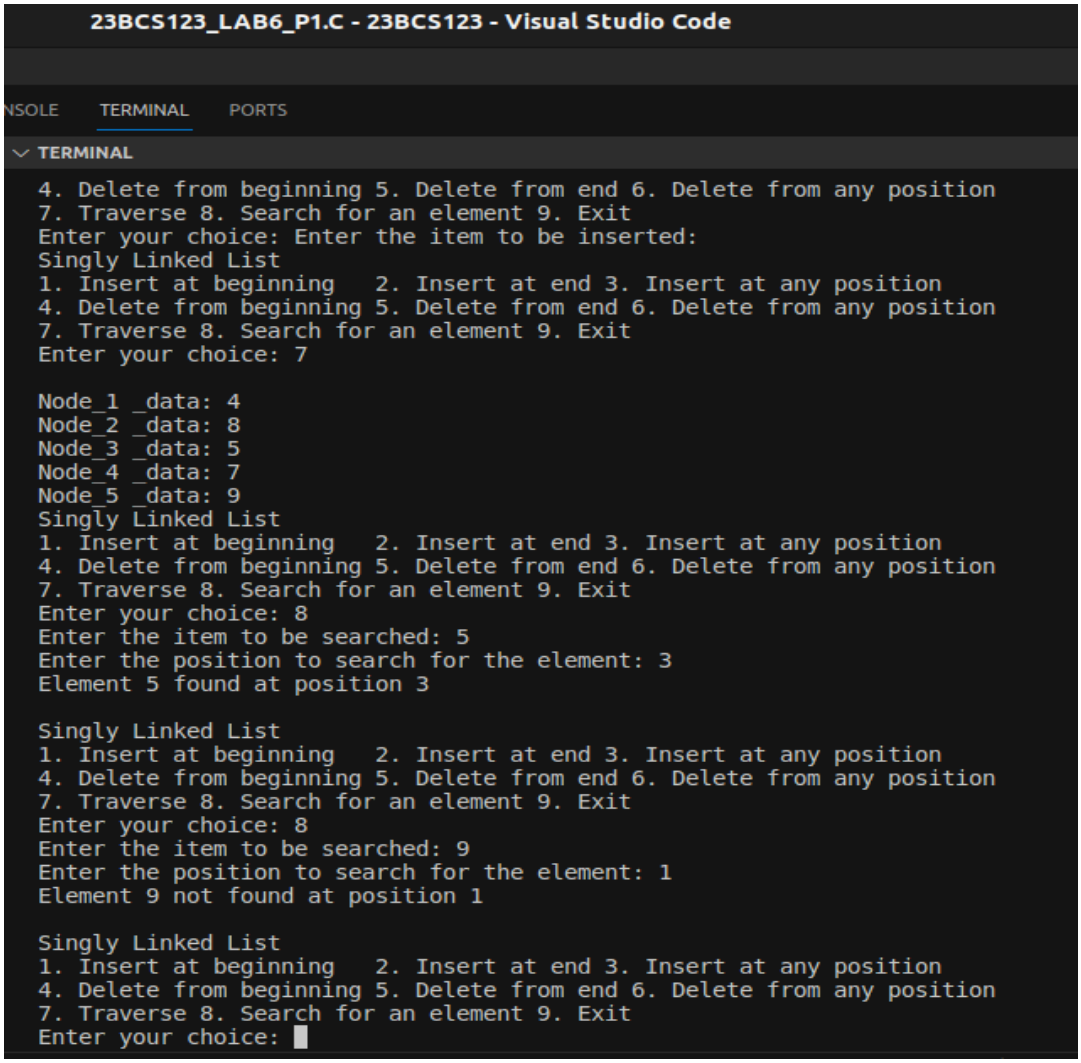
```
    }
    node *temp = *head;
    *head = (*head)->next;
    free(temp);
}

void delete_end(node **head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    node *p = *head, *q = NULL;
    while(p->next != NULL) {
        q = p;
        p = p->next;
    }
    if (q != NULL) {
        q->next = NULL;
    } else {
        *head = NULL;
    }
    free(p);
}

void delete_any_pos(node **head, int pos) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    node *p = *head, *q = NULL;
    if (pos == 1) {
        *head = p->next;
        free(p);
        return;
    }
    for(int i = 1; i < pos && p != NULL; i++) {
        q = p;
        p = p->next;
    }
    if (p == NULL) {
```

```
        printf("Position out of range\n");
        return;
    }
    q->next = p->next;
    free(p);
}

void search_element(node *head, int item, int pos) {
    node *ptr = head;
    int position = 1;

    while (ptr != NULL) {
        if (item == ptr->data && pos == position) {
            printf("Element %d found at position %d\n", item, position);
            return;
        }
        ptr = ptr->next;
        position++;
    }

    printf("Element %d not found at position %d\n", item, pos);
}

void checkPalindrome

int main() {
    node *head = NULL;
    int choice, item, pos;

    while(1) {
        printf("\nSingly Linked List\n");
        printf("1. Insert at beginning   ");
        printf("2. Insert at end ");
        printf("3. Insert at any position\n");
        printf("4. Delete from beginning ");
        printf("5. Delete from end ");
        printf("6. Delete from any position\n");
        printf("7. Traverse ");
        printf("8. Search for an element ");
        printf("9. Exit\n");
```

```
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                insert_begin(&head, item);
                break;
            case 2:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                insert_end(&head, item);
                break;
            case 3:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                printf("Enter the position at which to insert: ");
                scanf("%d", &pos);
                insert_any_position(&head, item, pos);
                break;
            case 4:
                delete_begin(&head);
                break;
            case 5:
                delete_end(&head);
                break;
            case 6:
                printf("Enter the position at which to delete: ");
                scanf("%d", &pos);
                delete_any_pos(&head, pos);
                break;
            case 7:
                traverse(head);
                break;
            case 8:
                printf("Enter the item to be searched: ");
                scanf("%d", &item);
                printf("Enter the position to search for the element: ");
                scanf("%d", &pos);
```

```
                search_element(head, item, pos);
                break;
            case 9:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT:



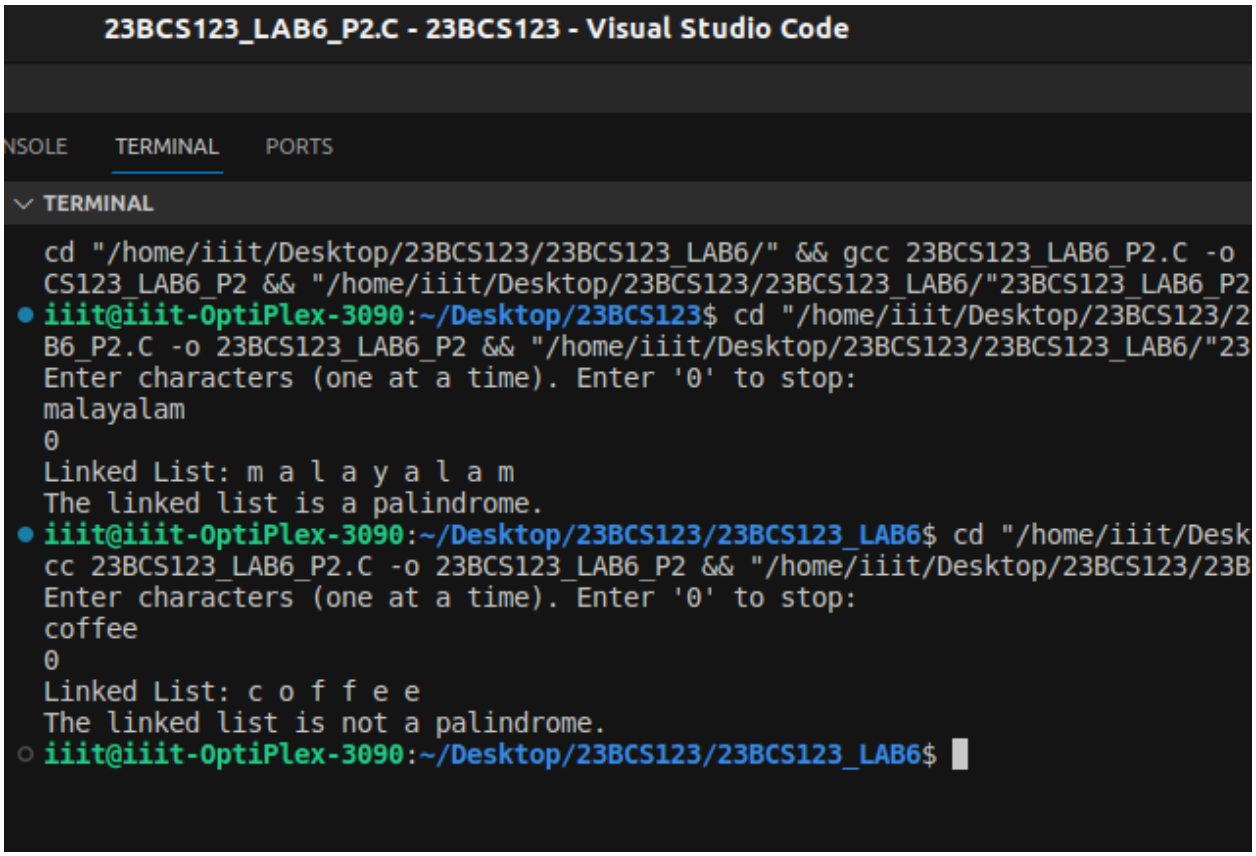
2.Write a C program to check whether a given string is a palindrome or not using the linked lists.

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 6
//PROGRAM 2
//Date : 17/02/2024

/*
2.Write a C program to check whether a given string is a palindrome or not
using the linked lists.
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct Node {
   char data;
   struct Node* next;
}node;



void insert_end(node **head, char item) {
   node* new_node = (node*)malloc(sizeof(node));
   new_node->data = item;
   new_node->next = NULL;
   if (*head == NULL) {
       *head = new_node;
       return;
   }
   node* p = *head;
   while (p->next != NULL) {
       p = p->next;
   }
   p->next = new_node;
}
```

```
void display(node* node) {
   printf("Linked List: ");
   while (node != NULL) {
       printf("%c ", node->data);
       node = node->next;
   }
   printf("\n");
}

bool palindrome_ll(node* head, int count) {
   int i = 0, j;
   node* front, * rear;
   while (i != count / 2) {
       front = rear = head;
       for (j = 0; j < i; j++) {
           front = front->next;
       }
       for (j = 0; j < count - (i + 1); j++) {
           rear = rear->next;
       }
       if (front->data != rear->data) {
           return false;
       } else {
           i++;
       }
   }
   return true;
}

int main() {
   node* head = NULL;
   char data;
   int counter = 0;

   printf("Enter characters (one at a time). Enter '0' to stop:\n");
   while (1) {
       scanf(" %c", &data);
       if (data == '0') {
           break;
       }
```

```
        insert_end(&head, data);
        counter++;
    }

    display(head);

    if (counter == 0) {
        printf("No characters entered. The linked list is empty.\n");
    } else {
        bool result = palindrome_ll(head, counter);
        if (result) {
            printf("The linked list is a palindrome.\n");
        } else {
            printf("The linked list is not a palindrome.\n");
        }
    }
    return 0;
}
```

OUTPUT:



3.Write a C program to search for an element in the circular linked list by specifying the position and the item.

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 6
//PROGRAM 3
//Date : 17/02/2024

/*
3.Write a C program to search for an element in the circular linked list
by specifying the position and the item.
*/

#include <stdio.h>
#include <stdlib.h>

typedef struct node {
   int data;
   struct node *next;
} node;

void insert_begin(node **head, int value) {
   node *newNode = (node*)malloc(sizeof(node));
   newNode->data = value;

   if (*head == NULL) {
       *head = newNode;
       newNode->next = *head;
   } else {
       node *temp = *head;
       while (temp->next != *head) {
           temp = temp->next;
       }
       newNode->next = *head;
       temp->next = newNode;
       *head = newNode;
   }
}
```

```c
void insert_end(node **head, int value) {
   node *newNode = (node*)malloc(sizeof(node));
   newNode->data = value;

   if (*head == NULL) {
       *head = newNode;
       newNode->next = *head;
   } else {
       node *temp = *head;
       while (temp->next != *head) {
           temp = temp->next;
       }
       temp->next = newNode;
       newNode->next = *head;
   }
}

void insert_at_position(node **head, int value, int position) {
   node *newNode = (node*)malloc(sizeof(node));
   newNode->data = value;

   if (*head == NULL || position <= 1) {
       newNode->next = *head;
       *head = newNode;
       return;
   }

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

   newNode->next = temp->next;
   temp->next = newNode;
}

void delete_begin(node **head) {
   if (*head == NULL) {
       printf("List is Empty!!! Deletion is not possible\n");
```

```
        return;
    }

    node *temp1 = *head, *temp2 = *head;

    if (temp1->next == *head) {
        *head = NULL;
        free(temp1);
    } else {
        while (temp1->next != *head) {
            temp1 = temp1->next;
        }

        *head = temp2->next;
        temp1->next = *head;
        free(temp2);
    }
}

void delete_end(node **head) {
    if (*head == NULL) {
        printf("List is Empty!!! Deletion is not possible\n");
        return;
    }

    node *temp1 = *head, *temp2 = *head;

    if (temp1->next == *head) {
        *head = NULL;
        free(temp1);
    } else {
        while (temp1->next != *head) {
            temp2 = temp1;
            temp1 = temp1->next;
        }

        temp2->next = *head;
        free(temp1);
    }
}
```

```c
void delete_at_position(node **head, int position) {
    if (*head == NULL || position < 1) {
        printf("Invalid position or empty list\n");
        return;
    }

    if (position == 1) {
        node *temp = *head;
        if ((*head)->next == *head) {
            free(*head);
            *head = NULL;
        } else {
            (*head) = temp->next;
            free(temp);
        }
        return;
    }

    node *temp1 = *head, *temp2 = NULL;
    for (int i = 1; i < position && temp1->next != *head; i++) {
        temp2 = temp1;
        temp1 = temp1->next;
    }

    if (temp1 == *head) {
        printf("Position out of range\n");
        return;
    }

    temp2->next = temp1->next;
    free(temp1);
}

void traverse(node *head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
```

```
    node *temp = head;
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != head);

    printf(" (head)\n");
}

void search_element(node *head, int item, int pos) {
    if (head == NULL || pos < 1) {
        printf("Invalid position or empty list\n");
        return;
    }

    node *temp = head;
    int position = 1;

    do {
        if (item == temp->data && pos == position) {
            printf("Element %d found at position %d\n", item, pos);
            return;
        }
        temp = temp->next;
        position++;
    } while (temp != head);

    printf("Element %d not found at position %d\n", item, pos);
}

int main() {
    node *head = NULL;
    int choice, item, pos;

    while (1) {
        printf("\nCircular Linked List \n");
        printf("1. Insert at beginning   ");
        printf("2. Insert at end ");
        printf("3. Insert at any position\n");
        printf("4. Delete from beginning ");
```

```
        printf("5. Delete from end ");
        printf("6. Delete from any position\n");
        printf("7. Traverse ");
        printf("8. Search for an element ");
        printf("9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                insert_begin(&head, item);
                break;
            case 2:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                insert_end(&head, item);
                break;
            case 3:
                printf("Enter the item to be inserted: ");
                scanf("%d", &item);
                printf("Enter the position at which to insert: ");
                scanf("%d", &pos);
                insert_at_position(&head, item, pos);
                break;
            case 4:
                delete_begin(&head);
                break;
            case 5:
                delete_end(&head);
                break;
            case 6:
                printf("Enter the position at which to delete: ");
                scanf("%d", &pos);
                delete_at_position(&head, pos);
                break;
            case 7:
                traverse(head);
                break;
```

```
            case 8:
                printf("Enter the item to be searched: ");
                scanf("%d", &item);
                printf("Enter the position to search for the element: ");
                scanf("%d", &pos);
                search_element(head, item, pos);
                break;
            case 9:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT:

```
Circular Linked List
1. Insert at beginning   2. Insert at end 3. Insert at any position
4. Delete from beginning 5. Delete from end 6. Delete from any position
7. Traverse 8. Search for an element 9. Exit
Enter your choice: 7
3 -> 1 -> 6 -> 7 -> 4 -> 8 ->  (head)

Circular Linked List
1. Insert at beginning   2. Insert at end 3. Insert at any position
4. Delete from beginning 5. Delete from end 6. Delete from any position
7. Traverse 8. Search for an element 9. Exit
Enter your choice: 8
Enter the item to be searched: 6
Enter the position to search for the element: 3
Element 6 found at position 3

Circular Linked List
1. Insert at beginning   2. Insert at end 3. Insert at any position
4. Delete from beginning 5. Delete from end 6. Delete from any position
7. Traverse 8. Search for an element 9. Exit
Enter your choice: 8
Enter the item to be searched: 1
Enter the position to search for the element: 7
Element 1 not found at position 7
```

4.Write a C program to add/subtract two ordered polynomial expressions using the linked list.

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 6
//PROGRAM 4
//Date : 17/02/2024

/*
4.Write a C program to add/subtract two ordered polynomial expressions
using the linked list.
*/

#include <stdio.h>
#include <stdlib.h>


typedef struct Term {
   int coeff;
   int power;
   struct Term* next;
} Term;


Term* createTerm(int coeff, int power) {
   Term* newTerm = (Term*)malloc(sizeof(Term));
   if (newTerm == NULL) {
       printf("Memory allocation error\n");
       exit(EXIT_FAILURE);
   }
   newTerm->coeff = coeff;
   newTerm->power = power;
   newTerm->next = NULL;
   return newTerm;
}
void insertTerm(Term** head, int coeff, int power) {
   Term* newTerm = createTerm(coeff, power);
   if (*head == NULL || power > (*head)->power) {
       newTerm->next = *head;
```

```
        *head = newTerm;
    } else {
        Term* current = *head;
        while (current->next != NULL && power < current->next->power) {
            current = current->next;
        }
        newTerm->next = current->next;
        current->next = newTerm;
    }
}


void display(Term* head) {
    if (head == NULL) {
        printf("Polynomial is empty\n");
        return;
    }


    while (head != NULL) {
        printf("%dx^%d", head->coeff, head->power);
        head = head->next;
        if (head != NULL) {
            printf(" + ");
        }
    }
    printf("\n");
}


Term* addPol(Term* poly1, Term* poly2) {
    Term* result = NULL;


    while (poly1 != NULL && poly2 != NULL) {
        if (poly1->power > poly2->power) {
            insertTerm(&result, poly1->coeff, poly1->power);
            poly1 = poly1->next;
        } else if (poly1->power < poly2->power) {
            insertTerm(&result, poly2->coeff, poly2->power);
```

```
            poly2 = poly2->next;
        } else {
            int sumCoeff = poly1->coeff + poly2->coeff;
            if (sumCoeff != 0) {
                insertTerm(&result, sumCoeff, poly1->power);
            }
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
    }


    while (poly1 != NULL) {
        insertTerm(&result, poly1->coeff, poly1->power);
        poly1 = poly1->next;
    }


    while (poly2 != NULL) {
        insertTerm(&result, poly2->coeff, poly2->power);
        poly2 = poly2->next;
    }


    return result;
}


Term* subPol(Term* poly1, Term* poly2) {
    Term* result = NULL;


    while (poly1 != NULL && poly2 != NULL) {
        if (poly1->power > poly2->power) {
            insertTerm(&result, poly1->coeff, poly1->power);
            poly1 = poly1->next;
        } else if (poly1->power < poly2->power) {
            insertTerm(&result, -poly2->coeff, poly2->power);
            poly2 = poly2->next;
        } else {
```

```
            int diffCoeff = poly1->coeff - poly2->coeff;
            if (diffCoeff != 0) {
                insertTerm(&result, diffCoeff, poly1->power);
            }
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
    }


    while (poly1 != NULL) {
        insertTerm(&result, poly1->coeff, poly1->power);
        poly1 = poly1->next;
    }


    while (poly2 != NULL) {
        insertTerm(&result, -poly2->coeff, poly2->power);
        poly2 = poly2->next;
    }


    return result;
}


void freePol(Term* polynomial) {
    while (polynomial != NULL) {
        Term* temp = polynomial;
        polynomial = polynomial->next;
        free(temp);
    }
}


int main() {
    Term* poly1 = NULL;
    Term* poly2 = NULL;
    Term* res_add = NULL;
    Term* res_sub = NULL;
```

```
    int coeff, power;
    char choice;


    printf("Enter the terms of the first polynomial in descending order of
powers:\n");
    do {
        printf("Enter the coeff and power of the term: ");
        scanf("%d %d", &coeff, &power);
        insertTerm(&poly1, coeff, power);


        printf("Do you want to add another term to the first polynomial?
(y/n): ");
        scanf(" %c", &choice);
    } while (choice == 'y' || choice == 'Y');


    printf("\nEnter the terms of the second polynomial in descending order
of powers:\n");
    do {
        printf("Enter the coeff and power of the term: ");
        scanf("%d %d", &coeff, &power);
        insertTerm(&poly2, coeff, power);


        printf("Do you want to add another term to the second polynomial?
(y/n): ");
        scanf(" %c", &choice);
    } while (choice == 'y' || choice == 'Y');


    printf("\nFirst Polynomial: ");
    display(poly1);


    printf("\nSecond Polynomial: ");
    display(poly2);
```

```
    res_add = addPol(poly1, poly2);
    res_sub = subPol(poly1, poly2);


    printf("\nResultant Polynomial (Sum): ");
    display(res_add);


    printf("\nResultant Polynomial (Subtraction): ");
    display(res_sub);

    freePol(poly1);
    freePol(poly2);
    freePol(res_add);
    freePol(res_sub);

    return 0;
}
```

OUTPUT:

```
Enter the terms of the first polynomial in descending order of powers:
Enter the coeff and power of the term: 2 4
Do you want to add another term to the first polynomial? (y/n): y
Enter the coeff and power of the term: 3 5
Do you want to add another term to the first polynomial? (y/n): y
Enter the coeff and power of the term: 4 2
Do you want to add another term to the first polynomial? (y/n): n

Enter the terms of the second polynomial in descending order of powers:
Enter the coeff and power of the term: 3 5
Do you want to add another term to the second polynomial? (y/n): y
Enter the coeff and power of the term: 1 2
Do you want to add another term to the second polynomial? (y/n): y
Enter the coeff and power of the term: 3 4
Do you want to add another term to the second polynomial? (y/n): n

First Polynomial: 3x^5 + 2x^4 + 4x^2

Second Polynomial: 3x^5 + 3x^4 + 1x^2

Resultant Polynomial (Sum): 6x^5 + 5x^4 + 5x^2

Resultant Polynomial (Subtraction): -1x^4 + 3x^2
```