1. **Activity: Add two polynomials using linked list**

**Algorithm:**

1. Create 2 linked lists for insertion of polynomial. (Note: Polynomial power should be in descending order, hence added automation for power in the program. User now only have to enter coefficient)
2. Pass the polynomials to addpoly() function
   1. Start from head of both polynomial lists, start traversing lists
   2. If powers are mismatched, enter the term with higher power directly into the resultant polynomial
   3. If powers are equal, coefficient of resultant = coeff. of poly1 + coeff. of poly2
3. Print the resultant polynomial

**Test Case Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| Polynomial 1 | Polynomial 2 | Resultant | Pass/Fail |
| 3x^5 + x^4 + 2x^3 – 2x +5 | 2x^5 + 3x^3 + 7x + 2 | 5x^5 + 1x^4 + 5x^3 + 5x + 7 | Pass |
| 2x^2 + 3x - 4 | 3x^2 – 3x + 4 | 5x^2 | Pass |

**Program:**

#include <stdio.h>

#include <stdlib.h>

struct node{

    int coef;

    int power;

    struct node \*next;

} \*first = NULL, \*second = NULL, \*res = NULL, \*ptr = NULL;

void create\_node(int po, int co, struct node \*\*head){

    //Temporary node

    struct node \*add = NULL;

    add = (struct node \*)malloc(sizeof(struct node));

    //Check for memory allocation

    if (add == NULL)

    {

        printf("Out of memory");

        return;

    }

    add->coef = co;

    add->power = po;

    add->next = NULL;

    //Check if list is empty or not

    if (\*head == NULL)

    {

        \*head = add;

    }

    //Add node to the end of list

    else{

        for (ptr = \*head; ptr->next != NULL; ptr = ptr->next);

        ptr->next = add;

    }

}

void addpoly(struct node \*p1, struct node \*p2, struct node \*r){

    //Start from head pointers of both polynomials and compare their powers.

    //Move to next node if powers do not match

    while (p1 != NULL && p2 != NULL)

    {

        //Check is power is mismatched.

        //Enter into resultant directly the highest power

        if (p1->power > p2->power)

        {

            r->power = p1->power;

            r->coef = p1->coef;

            p1 = p1->next;

        }

        else if(p1->power < p2->power){

            r->power = p2->power;

            r->coef = p2->coef;

            p2 = p2->next;

        }

        //Sum the coefficients is the powers are equal

        else{

            r->power = p1->power;

            r->coef = (p1->coef) + (p2->coef);

            p1 = p1->next;

            p2 = p2->next;

        }

        if (p1 != NULL && p2 != NULL)

        {

            //Insert a node for resultant polynomial

            r->next = (struct node \*)malloc(sizeof(struct node));

            r = r->next;

            r->next = NULL;

        }

    }

    //Entering the remaining terms to polynomial as it is.

    //Not required due to our driver code automation, still added for stability

    while (p1 != NULL || p2 != NULL)

    {

        r->next = (struct node \*)malloc(sizeof(struct node));

        r = r->next;

        r->next = NULL;

        if (p1 != NULL)

        {

            r->power = p1->power;

            r->coef = p1->coef;

            p1 = p1->next;

        }

        if (p2 != NULL)

        {

            r->power = p2->power;

            r->coef = p2->coef;

            p2 = p2->next;

        }

    }

}

void printPoly(struct node \*head){

    //Traverse through given list

    while (head != NULL)

    {

        //Print term only if coefficient in non-zero

        if (head->coef != 0)

        {

            printf("%dx^%d ",head->coef, head->power);

            //Print '+' if a next term is present

            if (head->next != NULL && head->next->coef != 0)

            {

                printf("+ ");

            }

        }

        head = head->next;

    }

}

int main(){

    int n, po1, co1;

    printf("Enter degree of polynomial 1: ");

    scanf("%d", &n);

    n++;

    // Creation of polynomial 1

    while (n--)

    {

        printf("Enter coefficient of x^%d: ", n);

        scanf("%d", &co1);

        create\_node(n, co1, &first);

    }

    printf("\nPolynomial 1 ->\n");

    printPoly(first);

    printf("\n\nEnter degree of polynomial 2: ");

    scanf("%d", &n);

    n++;

    //Creation of polynomial 2

    while (n--)

    {

        printf("Enter coefficient of x^%d: ", n);

        scanf("%d", &co1);

        create\_node(n, co1, &second);

    }

    printf("\nPolynomial 2 ->\n");

    printPoly(second);

    //Intitalise resultant polynomial

    res = (struct node \*)malloc(sizeof(struct node));

    printf("\n");

    addpoly(first, second, res);

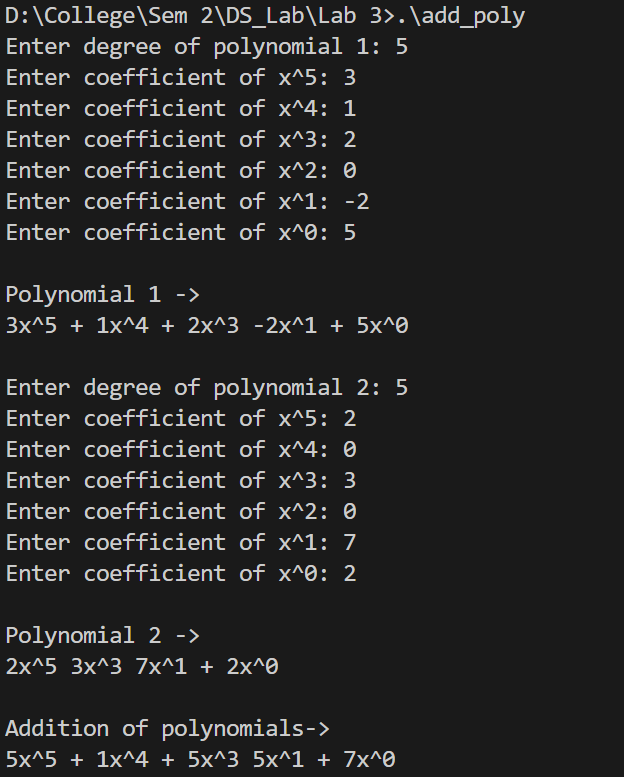
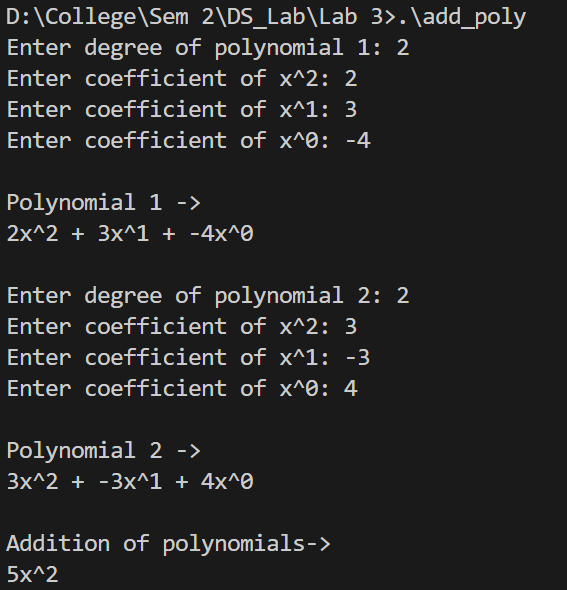
    printf("\nAddition of polynomials->\n");

    printPoly(res);

    return 0;

}

**Screenshot of compilation and execution:**

** **

1. **Activity:** Implement circular linked list and its implementation

**Algorithm:**

1. Algorithm to display list:
   1. Check if the list is empty. (head == NULL)
      1. If list empty. Inform user and exit function
   2. Print head->data
   3. Node = head->next
   4. While node->next is not head:
      1. Print node->data
2. Algorithm to insert element
   1. Create a new block and check is memory is assigned
   2. New->data = given data
   3. If list is empty
      1. Head = new
      2. New->next = head
   4. Else
      1. Traverse till node->next = head. i.e. node before head
      2. New->next = head
      3. Node->next = new
3. Algorithm to delete
   1. Check if list exist
   2. Traverse to index just before index we need to delete
   3. Unlink node from list
      1. Temp = ptr->next
      2. Ptr->next = temp->next
      3. Temp->next = NULL
   4. Free temp

**Program:**

#include <stdio.h>

#include <stdlib.h>

struct block

{

    int data;

    struct block \*next;

} \*head = NULL, \*ptr = NULL;;

//Clears the buffer memory

void clear (void)

{

  while ( getchar() != '\n' );

}

void display(){

    if (head == NULL)

    {

        printf("List is empty..\n");

        return;

    }

    printf("...-->\n%d -->\n", head->data);

    for(ptr = head->next; ptr != head; ptr = ptr->next){

        printf("%d -->\n", ptr->data);

    }

    printf("...");

}

void insert(int n){

    struct block \*new;

    new = (struct block \*)malloc(sizeof(struct block));

    if (new == NULL)

    {

        printf("Out of memory");

    }

    new->data = n;

    if (head == NULL)

    {

        head = new;

        new->next = head;

    }

    else{

        for(ptr = head; ptr->next != head; ptr = ptr->next);

        new->next = head;

        ptr->next = new;

    }

    printf("Node Successfully inserted.\n");

}

void delete(int n){

    if(head == NULL){

        printf("List does not exist make a list first.\n");

        return;

    }

    if (head->next == head)

    {

        printf("Single element in list.... Deleted.\n");

        head = NULL;

        return;

    }

    int counter = 0;

    for (ptr = head; counter < n-2; ptr = ptr->next, counter++);

    printf("Deleted Element: %d", ptr->next->data);

    struct block \*temp;

    temp = ptr->next;

    ptr->next = temp->next;

    temp->next = NULL;

free(temp);

}

int main(){

    int n;

    MAIN\_MENU: system("cls");

    printf("Circular linked list\n");

    printf("--------------------\n");

    printf("1. Display list\n");

    printf("2. Insert element(s)\n");

    printf("3. Delete at a given index\n");

    printf("4. Exit\n");

    printf("\nEnter your choice: ");

    scanf("%d", &n);

    switch (n)

    {

    case 1:

        display();

        printf("\n\nPress any key to continue...");

        clear();

        getchar();

        goto MAIN\_MENU;

        break;

    case 2:

        system("cls");

        int num;

        printf("Enter number of elements you want to enter: ");

        clear();

        scanf("%d", &num);

        for (int i = 1; i <= num; i++)

        {

            int d;

            printf("New data %d: ", i);

            clear();

            scanf("%d", &d);

            insert(d);

        }

        printf("\n\nSuccessfully inserted. Press any key to continue...");

        clear();

        getchar();

        goto MAIN\_MENU;

        break;

    case 3:

        system("cls");

        int index;

        printf("Note: In circular list, indexes are overlapped if list size is smaller than index\n");

        printf("Enter index of elements you want to delete: ");

        clear();

        scanf("%d", &index);

        delete(index);

        printf("\n\nPress any key to continue...");

        clear();

        getchar();

        goto MAIN\_MENU;

        break;

    case 4:

        free(head);

        free(ptr);

        return 0;

        break;

    default:

        printf("Invalid selection. Press any key to continue..");

        clear();

        getchar();

        goto MAIN\_MENU;

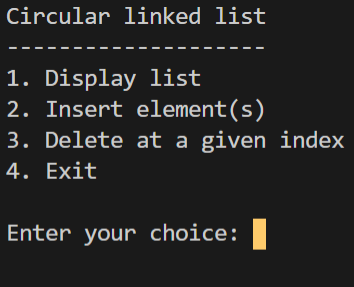
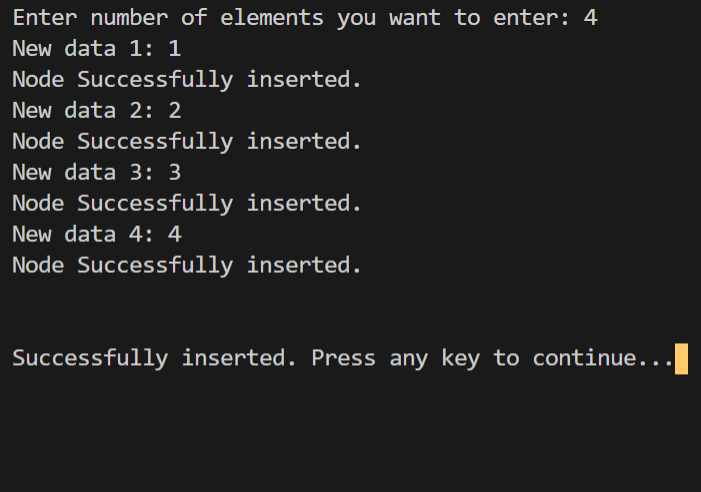
        break;

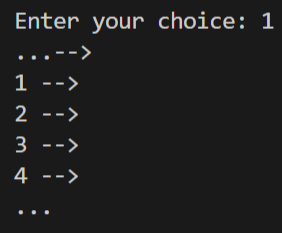
    }

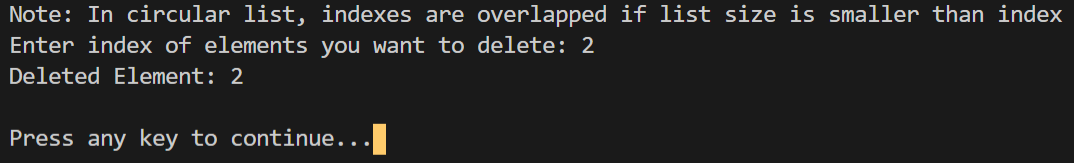
    return 0;

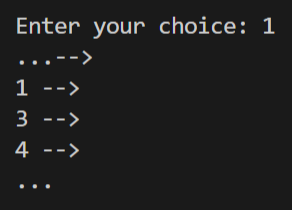
}

**Screenshot of compilation and execution:**

** **

****

****

****

1. **Activity:** Implement doubly linked list and its operations.

**Algorithm:**

1. Algorithm for insertion:
   1. To insert in beginning
      1. Check for memory allocation to new node
      2. New->data = data
      3. New->prev = NULL
      4. If head is NULL
         1. Head = new
         2. Last = new
         3. New->next = NULL
      5. Else
         1. New->next = head
         2. Head->prev = new
         3. Head = new
   2. To insert at end
      1. Check for memory allocation to new node
      2. New->data = data
      3. New->next = NULL
      4. If head is NULL (list empty)
         1. Head = new
         2. Last = new
         3. New->prev = NULL
      5. Else
         1. New->prev = last
         2. Last->next = new
         3. Last = new
   3. To insert after an element
      1. Check for small or empty list
      2. Get list size
      3. Get index from user
      4. Create new node
      5. Traverse to index given by user
         1. If index is before mid, traverse from head
         2. Else traverse from last
      6. New->prev = ptr
      7. New->next = ptr->next
      8. Ptr->next->prev = new
      9. Ptr->next = new
2. Algorithm to print list
   1. To print list normally
      1. Check for empty list
      2. Ptr = head
      3. While ptr is not NULL
         1. Print ptr->data
         2. Ptr = ptr->next
   2. To print list in reverse
      1. Check for empty list
      2. Ptr = last
      3. While ptr is not NULL
         1. Print ptr->data
         2. Ptr = ptr->prev
3. Algorithm to delete elements
   1. To delete element from beginning
      1. Check for small lists
      2. Ptr = head->next
      3. Ptr->prev = NULL
      4. Free head
      5. Head = ptr
   2. To delete element from end
      1. Check for small lists
      2. Ptr = last->prev
      3. Ptr->next = NULL
      4. Free last
      5. Last = ptr
   3. To delete from end
      1. Check for small lists
      2. Get index whose next element is to be deleted
      3. Traverse to index
         1. From head if index before mid
         2. From last if index after mid
      4. Temp = ptr->next
      5. Ptr->next = temp->next
      6. Temp->next->prev = ptr
      7. Free temp

**Program:**

#include <stdio.h>

#include <stdlib.h>

struct node{

    int data;

    struct node \*prev, \*next;

} \*head=NULL, \*ptr=NULL, \*last=NULL;

//Clears the buffer memory

void clear (void)

{

  while ( getchar() != '\n' );

}

int listsize(){

    int size = 0;

    for(ptr = head; ptr != NULL; ptr = ptr->next){

        size++;

    }

    return size;

}

void insertstart(int n){

    struct node \*temp = NULL;

    temp = (struct node \*)malloc(sizeof(struct node));

    if (temp == NULL)

    {

        printf("Out of memory. Can not add.");

    }

    temp->data = n;

    temp->prev = NULL;

    if (head == NULL)

    {

        head = temp;

        last = temp;

        temp->next = NULL;

    }

    else{

        temp->next = head;

        head->prev = temp;

        head = temp;

    }

}

void insertend(int n){

    struct node \*temp = NULL;

    temp = (struct node \*)malloc(sizeof(struct node));

    if (temp == NULL)

    {

        printf("Out of memory. Can not add.");

    }

    temp->data = n;

    temp->next = NULL;

    if (head == NULL)

    {

        head = temp;

        last = temp;

        temp->prev = NULL;

    }

    else{

        temp->prev = last;

        last->next = temp;

        last = temp;

    }

}

void insertindex(int n){

    if (head == NULL)

    {

        printf("List is empty.. Use other two insert options\n");

        return;

    }

    if (head->next == NULL)

    {

        printf("Single element in list. Can not insert in middle.\n");

        return;

    }

    int l\_size = listsize();

    printf("Insert element after index (1-%d): ", l\_size-1);

    int index;

    clear();

    scanf("%d", &index);

    struct node \*temp = NULL;

    temp = (struct node \*)malloc(sizeof(struct node));

    if (temp == NULL)

    {

        printf("Out of memory. Can not add.");

    }

    temp->data = n;

    if ((int)(l\_size/2) >= index)

    {

        int i = 1;

        for(ptr=head; i < index ; ptr = ptr->next, i++);

        temp->prev = ptr;

        temp->next = ptr->next;

        ptr->next->prev = temp;

        ptr->next = temp;

    }

    else{

        int i = l\_size - index;

        for(ptr = last; i > 0; ptr = ptr->prev, i--);

        temp->prev = ptr;

        temp->next = ptr->next;

        ptr->next->prev = temp;

        ptr->next = temp;

    }

}

void printlist(){

    system("cls");

    if (head == NULL)

    {

        printf("List is empty..");

        return;

    }

    for(ptr = head; ptr != NULL; ptr=ptr->next){

        printf("%d <-->\n", ptr->data);

    }

}

void printrev(){

    system("cls");

    if (head == NULL)

    {

        printf("List is empty..");

        return;

    }

    for(ptr = last; ptr != NULL; ptr=ptr->prev){

        printf("%d <-->\n", ptr->data);

    }

}

void deletestart(){

    if (head == NULL)

    {

        printf("List is empty..\n");

        return;

    }

    if (head->next == NULL)

    {

        printf("Single element in list. Deleted\n");

        head = NULL;

        free(last);

        last = NULL;

        return;

    }

    ptr = head->next;

    if (ptr != NULL)

    {

        ptr->prev = NULL;

    }

    free(head);

    head = ptr;

    printf("Deletion Successful.\n");

}

void deleteend(){

   if (head == NULL)

    {

        printf("List is empty..\n");

        return;

    }

    if (head->next == NULL)

    {

        printf("Single element in list. Deleted\n");

        head = NULL;

        free(last);

        last = NULL;

        return;

    }

    ptr = last->prev;

    if (ptr != NULL)

    {

        ptr->next = NULL;

    }

    free(last);

    last = ptr;

    printf("Deletion Successful.\n");

}

void deleteindex(){

    if (head == NULL)

    {

        printf("List is empty..\n");

        return;

    }

    if (head->next == NULL)

    {

        printf("Single element in list. Deleted\n");

        head = NULL;

        free(last);

        last = NULL;

        return;

    }

    int l\_size = listsize();

    if (l\_size == 2)

    {

        printf("Can not delete like this.. List too small\nUse other 2 options\n");

        return;

    }

    printf("Insert element after index (1-%d): ", l\_size-2);

    clear();

    int ind;

    scanf("%d", &ind);

    if ((int)(l\_size/2) >= ind){

        int i = 1;

        for(ptr=head; i < ind ; ptr = ptr->next, i++);

    }

    else{

        int i = l\_size - ind;

        for(ptr = last; i > 0; ptr = ptr->prev, i--);

    }

    struct node \*temp = ptr->next;

    ptr->next = temp->next;

    ptr->next->prev = ptr;

    free(temp);

    printf("Deletion Successful\n");

}

int main(){

    int n;

    insertend(1);

    insertend(2);

    insertend(3);

    insertend(4);

    insertend(5);

    insertend(6);

    MAIN\_MENU: system("cls");

    printf("Doubly linked list\n");

    printf("--------------------\n");

    printf("1. Display list\n");

    printf("2. Insert element\n");

    printf("3. Delete element\n");

    printf("4. Exit\n");

    printf("\nEnter your choice: ");

    scanf("%d", &n);

    switch (n)

    {

    case 1:

        DISPLAY\_MENU: system("cls");

        printf("Display Menu\n");

        printf("--------------------\n");

        printf("1. Display normally\n");

        printf("2. Display in reverse\n");

        printf("\nEnter your choice: ");

        clear();

        scanf("%d", &n);

        switch (n)

        {

        case 1:

            printlist();

            printf("\n\nPress any key to continue...");

            clear();

            getchar();

            goto MAIN\_MENU;

            break;

        case 2:

            printrev();

            printf("\n\nPress any key to continue...");

            clear();

            getchar();

            goto MAIN\_MENU;

            break;

        default:

            printf("Invalid selection. Press any key to continue..");

            clear();

            getchar();

            goto DISPLAY\_MENU;

            break;

        }

        break;

    case 2:

        INSERT\_MENU: system("cls");

        printf("Insert Menu\n");

        printf("--------------------\n");

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

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

        printf("3. Insert at after element.\n");

        printf("\nEnter your choice: ");

        clear();

        scanf("%d", &n);

        int num;

        printf("\nEnter element to insert: ");

        clear();

        scanf("%d", &num);

        switch (n)

        {

        case 1:

            insertstart(num);

            printf("Successfully inserted. Press enter to continue.");

            clear();

            getchar();

            goto MAIN\_MENU;

            break;

        case 2:

            insertend(num);

            printf("Successfully inserted. Press enter to continue.");

            clear();

            getchar();

            goto MAIN\_MENU;

            break;

        case 3:

            insertindex(num);

            printf("Press enter to continue.");

            clear();

            getchar();

            goto MAIN\_MENU;

            break;

        default:

            printf("Invalid selection. Press any key to continue..");

            clear();

            getchar();

            goto INSERT\_MENU;

            break;

        }

        break;

    case 3:

        DELETE\_MENU: system("cls");

        printf("Delete Menu\n");

        printf("--------------------\n");

        printf("1. Delete element at beginning.\n");

        printf("2. Delete element at end.\n");

        printf("3. Delete element after an index.\n");

        printf("\nEnter your choice: ");

        clear();

        scanf("%d", &n);

        switch(n){

            case 1:

                deletestart();

                printf("Press any key to continue.");

                clear();

                getchar();

                goto MAIN\_MENU;

                break;

            case 2:

                deleteend();

                printf("Press any key to continue.");

                clear();

                getchar();

                goto MAIN\_MENU;

                break;

            case 3:

                deleteindex();

                printf("Press any key to continue.");

                clear();

                getchar();

                goto MAIN\_MENU;

                break;

            default:

                printf("Invalid selection. Press enter to continue.");

                clear();

                getchar();

                goto DELETE\_MENU;

                break;

        }

        break;

    case 4:

        free(head);

        free(last);

        free(ptr);

        return 0;

        break;

    default:

        printf("Invalid selection. Press any key to continue..");

        clear();

        getchar();

        goto MAIN\_MENU;

        break;

    }

    return 0;

}

**Screenshot of compilation and execution:**

