2. Polynomial Manipulation

Program :

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

struct Node {
  int coeff;
  int exp;
  struct Node * next;
}* poly1 = NULL, * poly2 = NULL, * result = NULL;

void create1() {
  struct Node * t, * last = NULL;
  int num, i;

  printf("Enter number of terms: ");
  scanf("%d", & num);
  printf("Enter each term with coeff and exp:\n");

  for (i = 0; i < num; i++) {
    t = (struct Node * ) malloc(sizeof(struct Node));
    scanf("%d%d", & t -> coeff, & t -> exp);
    t -> next = NULL;
    if (poly1 == NULL) {

      poly1 = last = t;
    } else {
      last -> next = t;
      last = t;
    }
  }
}

void create2() {
  struct Node * t, * last = NULL;
  int num, i;

  printf("Enter number of terms: ");
  scanf("%d", & num);
  printf("Enter each term with coeff and exp:\n");

  for (i = 0; i < num; i++) {
    t = (struct Node * ) malloc(sizeof(struct Node));
    scanf("%d%d", & t -> coeff, & t -> exp);
    t -> next = NULL;
    if (poly2 == NULL) {
      poly2 = last = t;
    } else {
      last -> next = t;
      last = t;
    }
```

```
  }
}

void Display(struct Node * p) {
  printf("(%dx^%d) ", p -> coeff, p -> exp);
  p = p -> next;

  while (p) {
    printf("+ (%dx^%d)", p -> coeff, p -> exp);
    p = p -> next;
  }
  printf("\n");
}

void normalize() {
  struct Node * ptr = result;
  while (ptr) {
    struct Node * temp = ptr;
    while (temp -> next) {
      struct Node * erase;
      if (ptr -> exp == temp -> next -> exp) {
        ptr -> coeff = ptr -> coeff + temp -> next -> coeff;
        erase = temp -> next;
        temp -> next = temp -> next -> next;
        free(erase);

      }
      temp = temp -> next;
    }
    ptr = ptr -> next;
  }
}

void add(struct Node * p1, struct Node * p2) {
  struct Node * t, * last = NULL;
  int num1, num2;
  int p;
  if (p1 -> exp > p2 -> exp)
    p = p1 -> exp;
  else
    p = p2 -> exp;

  while (p) {
    p--;
    if (p1 != NULL && p2 != NULL) {
      t = (struct Node * ) malloc(sizeof(struct Node));
      if (p1 -> exp == p2 -> exp) {
        num1 = p1 -> exp;
        num2 = p1 -> coeff + p2 -> coeff;
        p1 = p1 -> next;
        p2 = p2 -> next;
      } else if (p1 -> exp > p2 -> exp) {
        num1 = p1 -> exp;
        num2 = p1 -> coeff;
        p1 = p1 -> next;
      } else if (p1 -> exp < p2 -> exp) {
```

```
        num1 = p2 -> exp;
        num2 = p2 -> coeff;
        p2 = p2 -> next;
      }
      t -> exp = num1;
      t -> coeff = num2;

      t -> next = NULL;
      if (result == NULL) {
        result = last = t;
      } else {
        last -> next = t;
        last = t;
      }
    }
    if (p1 != NULL && p2 == NULL) {
      t = (struct Node * ) malloc(sizeof(struct Node));
      num1 = p1 -> exp;
      num2 = p1 -> coeff;
      p1 = p1 -> next;

      t -> exp = num1;
      t -> coeff = num2;

      t -> next = NULL;
      if (result == NULL) {
        result = last = t;
      } else {
        last -> next = t;
        last = t;
      }
    }
    if (p1 == NULL && p2 != NULL) {
      t = (struct Node * ) malloc(sizeof(struct Node));
      num1 = p2 -> exp;
      num2 = p2 -> coeff;
      p2 = p2 -> next;
      t -> exp = num1;
      t -> coeff = num2;

      t -> next = NULL;
      if (result == NULL) {
        result = last = t;
      } else {
        last -> next = t;
        last = t;
      }

    }
  }
  normalize(result);
  Display(result);
  result = NULL;
}

void sub(struct Node * p1, struct Node * p2) {
  struct Node * t, * last = NULL;
```

```
int num1, num2;
int p;
if (p1 -> exp > p2 -> exp)
  p = p1 -> exp;
else
  p = p2 -> exp;

while (p) {
  p--;
  if (p1 != NULL && p2 != NULL) {
    t = (struct Node * ) malloc(sizeof(struct Node));
    if (p1 -> exp == p2 -> exp) {
      num1 = p1 -> exp;
      num2 = p1 -> coeff - p2 -> coeff;
      p1 = p1 -> next;
      p2 = p2 -> next;
    } else if (p1 -> exp > p2 -> exp) {
      num1 = p1 -> exp;
      num2 = -p1 -> coeff;
      p1 = p1 -> next;
    } else if (p1 -> exp < p2 -> exp) {
      num1 = p2 -> exp;
      num2 = -p2 -> coeff;
      p2 = p2 -> next;
    }
    t -> exp = num1;
    t -> coeff = num2;

    t -> next = NULL;
    if (result == NULL) {
      result = last = t;
    } else {
      last -> next = t;
      last = t;
    }
  }
  if (p1 != NULL && p2 == NULL) {
    t = (struct Node * ) malloc(sizeof(struct Node));
    num1 = p1 -> exp;
    num2 = p1 -> coeff;
    p1 = p1 -> next;

    t -> exp = num1;
    t -> coeff = num2;

    t -> next = NULL;
    if (result == NULL) {
      result = last = t;
    } else {
      last -> next = t;
      last = t;
    }
  }
  if (p1 == NULL && p2 != NULL) {
    t = (struct Node * ) malloc(sizeof(struct Node));
    num1 = p2 -> exp;
    num2 = -p2 -> coeff;
```

```
      p2 = p2 -> next;
      t -> exp = num1;
      t -> coeff = num2;

      t -> next = NULL;
      if (result == NULL) {
        result = last = t;
      } else {
        last -> next = t;
        last = t;
      }

    }
  }
  normalize(result);
  Display(result);
  result = NULL;
}

void mul(struct Node * p1, struct Node * p2) {
  int i = 1;
  struct Node * t, * last = NULL;
  while (p1) {
    struct Node * temp = p2;
    while (temp) {
      t = malloc(sizeof(struct Node));
      t -> coeff = (temp -> coeff) * (p1 -> coeff);
      t -> exp = temp -> exp + p1 -> exp;
      if (result == NULL) {
        result = last = t;
      } else {
        last -> next = t;
        last = t;
      }
      temp = temp -> next;
    }
    p1 = p1 -> next;
  }

  normalize(result);
  Display(result);
  result = NULL;
}

int main() {
  int x;
  printf("Enter first polynomial:\n");
  create1();
  Display(poly1);
  create2();
  Display(poly2);
  printf("\nADD: ");
  add(poly1, poly2);
  printf("\nSUB: ");
  sub(poly1, poly2);
  printf("\nMUL: ");
  mul(poly1, poly2);
```

```
  return 0;
}
```

Output :

```
Enter first polynomial:
Enter number of terms: 3
Enter each term with coeff and exp:
1 2
2 3
3 4
(1x^2) + (2x^3)+ (3x^4)
Enter number of terms: 3
Enter each term with coeff and exp:
2 2
3 3
4 4
(2x^2) + (3x^3)+ (4x^4)

ADD: (3x^2) + (5x^3)

SUB: (-1x^2) + (-1x^3)

MUL: (2x^4) + (7x^5)+ (16x^6)+ (17x^7)+ (12x^8)
```