Ankush Jha DSA Assignment 1 BT22CSH023

Assignment Question

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

structNode {

    floatcoeff;

    int exp;

    structNode\*next;

};

typedefstruct Node Node;

// Function to create a new node

Node\*createNode(floatcoeff, intexp) {

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

    if (newNode==NULL) {

        printf("Memory allocation failed\n");

        exit(1);

    }

    newNode->coeff=coeff;

    newNode->exp= exp;

    newNode->next=NULL;

    returnnewNode;

}

// Function to insert a term into the polynomial

voidinsertTerm(Node\*\*header, floatcoeff, intexp) {

    Node\*newNode=createNode(coeff, exp);

    if (\*header ==NULL) {

        \*header =createNode(0, 0);  // Initialize the circular linked list with a header node

        newNode->next=\*header;

        (\*header)->next=newNode;

        return;

    }

    Node\* current =\*header;

    while (current->next!=\*header &&current->next->exp> exp) {

        current =current->next;

    }

    newNode->next=current->next;

    current->next=newNode;

}

// Function to read a polynomial and convert it to circular representation

Node\*Pread() {

    Node\* header =NULL;

    intnumTerms;

    printf("Enter the number of terms in the polynomial: ");

    scanf("%d", &numTerms);

    for (inti=0; i<numTerms; ++i) {

        floatcoeff;

        int exp;

        printf("Enter coefficient and exponent for term %d: ", i+1);

        scanf("%f%d", &coeff, &exp);

        insertTerm(&header, coeff, exp);

    }

    return header;

}

// Function to output the polynomial in mathematical form

voidPwrite(Node\*header) {

    Node\* current =header->next;

    while (current != header) {

        printf("%.2fx^%d", current->coeff, current->exp);

        current =current->next;

        if (current != header) {

            printf(" + ");

        }

    }

    printf("\n");

}

// Function to add two polynomials

Node\*Padd(Node\*a, Node\*b) {

    Node\* result =NULL;

    Node\*currentA=a->next;

    Node\*currentB=b->next;

    while (currentA!= a &&currentB!= b) {

        if (currentA->exp>currentB->exp) {

            insertTerm(&result, currentA->coeff, currentA->exp);

            currentA=currentA->next;

        } elseif (currentA->exp<currentB->exp) {

            insertTerm(&result, currentB->coeff, currentB->exp);

            currentB=currentB->next;

        } else {

            floatsumCoeff=currentA->coeff+currentB->coeff;

            if (fabs(sumCoeff) >1e-6) {

                insertTerm(&result, sumCoeff, currentA->exp);

            }

            currentA=currentA->next;

            currentB=currentB->next;

        }

    }

    while (currentA!= a) {

        insertTerm(&result, currentA->coeff, currentA->exp);

        currentA=currentA->next;

    }

    while (currentB!= b) {

        insertTerm(&result, currentB->coeff, currentB->exp);

        currentB=currentB->next;

    }

    return result;

}

// Function to subtract two polynomials

Node\*Psub(Node\*a, Node\*b) {

    Node\* result =NULL;

    Node\*currentA=a->next;

    Node\*currentB=b->next;

    while (currentA!= a &&currentB!= b) {

        if (currentA->exp>currentB->exp) {

            insertTerm(&result, currentA->coeff, currentA->exp);

            currentA=currentA->next;

        } elseif (currentA->exp<currentB->exp) {

            insertTerm(&result, -currentB->coeff, currentB->exp);

            currentB=currentB->next;

        } else {

            floatdiffCoeff=currentA->coeff-currentB->coeff;

            if (fabs(diffCoeff) >1e-6) {

                insertTerm(&result, diffCoeff, currentA->exp);

            }

            currentA=currentA->next;

            currentB=currentB->next;

        }

    }

    while (currentA!= a) {

        insertTerm(&result, currentA->coeff, currentA->exp);

        currentA=currentA->next;

    }

    while (currentB!= b) {

        insertTerm(&result, -currentB->coeff, currentB->exp);

        currentB=currentB->next;

    }

    return result;

}

// Function to multiply two polynomials

Node\*Pmult(Node\*a, Node\*b) {

    Node\* result =NULL;

    Node\*currentA=a->next;

    while (currentA!= a) {

        Node\*currentB=b->next;

        while (currentB!= b) {

            insertTerm(&result, currentA->coeff\*currentB->coeff, currentA->exp+currentB->exp);

            currentB=currentB->next;

        }

        currentA=currentA->next;

    }

    return result;

}

// Function to evaluate the polynomial at a specific point

floatPeval(Node\*header, floatx) {

    float result =0.0;

    Node\* current =header->next;

    while (current != header) {

        result +=current->coeff\*pow(x, current->exp);

        current =current->next;

    }

    return result;

}

// Function to erase a term with a specific exponent

voidPearse(Node\*\*header, intexp) {

    Node\* current =\*header;

    while (current->next!=\*header) {

        if (current->next->exp== exp) {

            Node\* temp =current->next;

            current->next=temp->next;

            free(temp);

            return;

        }

        current =current->next;

    }

}

// Function to free memory occupied by the linked list

voidfreeList(Node\*header) {

    Node\* current =header->next;

    while (current != header) {

        Node\* temp = current;

        current =current->next;

        free(temp);

    }

    free(header);

}

intmain() {

    Node\* a =NULL;

    Node\* b =NULL;

    printf("Enter polynomial A:\n");

    a =Pread();

    printf("Enter polynomial B:\n");

    b =Pread();

    Node\* c =Padd(a, b);

    Node\* d =Psub(a, b);

    Node\* e =Pmult(a, b);

    printf("A: ");

    Pwrite(a);

    printf("B: ");

    Pwrite(b);

    printf("A + B: ");

    Pwrite(c);

    printf("A - B: ");

    Pwrite(d);

    printf("A \* B: ");

    Pwrite(e);

    // Evaluate polynomials at a specific point

    float x;

    printf("Enter the value of x to evaluate polynomials: ");

    scanf("%f", &x);

    printf("A(%f) = %f\n", x, Peval(a, x));

    printf("B(%f) = %f\n", x, Peval(b, x));

    // Erase a term from a polynomial

    int exp;

    printf("Enter the exponent of the term to erase from polynomial A: ");

    scanf("%d", &exp);

    Pearse(&a, exp);

    printf("A after erasing term with exponent %d: ", exp);

    Pwrite(a);

    // Free memory

    freeList(a);

    freeList(b);

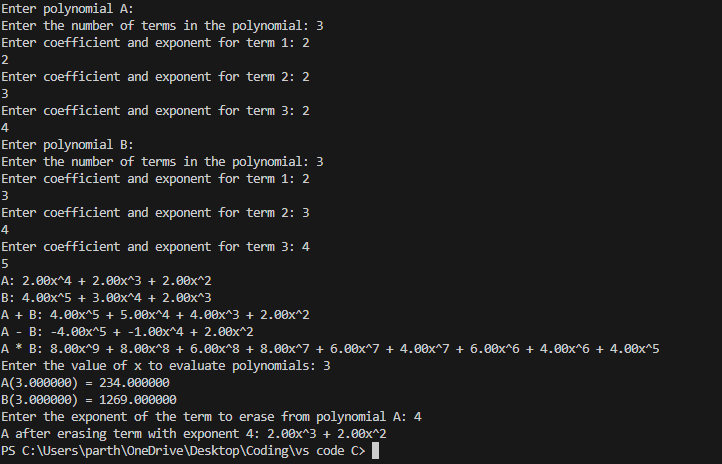
    freeList(c);

    freeList(d);

    freeList(e);

    return0;

}



HW Question 1)

#include<stdio.h>

#include<stdlib.h>

// Define the structure for a polynomial term

typedefstructTerm {

    intcoefficient;

    intexponent;

    structTerm\*next;

} Term;

// Function to insert a term into the linked list

voidinsertTerm(Term\*\*poly, intcoef, intexp) {

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

    newTerm->coefficient=coef;

    newTerm->exponent=exp;

    newTerm->next=NULL;

    if (\*poly==NULL) {

        \*poly=newTerm;

    } else {

        Term\*current=\*poly;

        while (current->next!=NULL) {

            current=current->next;

        }

        current->next=newTerm;

    }

}

// Function to add two polynomials

Term\*addPolynomials(Term\*poly1, Term\*poly2) {

    Term\*result=NULL;

    while (poly1!=NULL||poly2!=NULL) {

        if (poly1==NULL|| (poly2!=NULL&&poly1->exponent<poly2->exponent)) {

            insertTerm(&result, poly2->coefficient, poly2->exponent);

            poly2=poly2->next;

        } elseif (poly2==NULL||poly1->exponent>poly2->exponent) {

            insertTerm(&result, poly1->coefficient, poly1->exponent);

            poly1=poly1->next;

        } else {

            intsumCoeff=poly1->coefficient+poly2->coefficient;

            if (sumCoeff!=0) {

                insertTerm(&result, sumCoeff, poly1->exponent);

            }

            poly1=poly1->next;

            poly2=poly2->next;

        }

    }

    returnresult;

}

// Function to display a polynomial

voiddisplayPolynomial(Term\*poly) {

    if (poly==NULL) {

        printf("0\n");

        return;

    }

    while (poly!=NULL) {

        printf("%d", poly->coefficient);

        if (poly->exponent!=0) {

            printf("x^%d", poly->exponent);

        }

        if (poly->next!=NULL) {

            printf(" + ");

        }

        poly=poly->next;

    }

    printf("\n");

}

intmain() {

    Term\*poly1=NULL;

    Term\*poly2=NULL;

    // Adding terms to the first polynomial using linked list

    insertTerm(&poly1, 3, 2);

    insertTerm(&poly1, 4, 1);

    insertTerm(&poly1, 5, 0);

    // Adding terms to the second polynomial using direct input

    intcoef, exp;

    printf("Enter coefficients and exponents for the second polynomial (enter -1 to stop):\n");

    while (1) {

        printf("Coefficient: ");

        scanf("%d", &coef);

        if (coef==-1) {

            break;

        }

        printf("Exponent: ");

        scanf("%d", &exp);

        insertTerm(&poly2, coef, exp);

    }

    printf("First polynomial: ");

    displayPolynomial(poly1);

    printf("Second polynomial: ");

    displayPolynomial(poly2);

    Term\*result=addPolynomials(poly1, poly2);

    printf("Resultant polynomial: ");

    displayPolynomial(result);

    // Clean up memory

    while (poly1!=NULL) {

        Term\*temp=poly1;

        poly1=poly1->next;

        free(temp);

    }

    while (poly2!=NULL) {

        Term\*temp=poly2;

        poly2=poly2->next;

        free(temp);

    }

    while (result!=NULL) {

        Term\*temp=result;

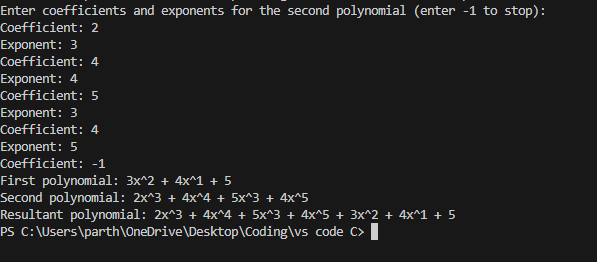
        result=result->next;

        free(temp);

    }

    return0;

}



HW Question 2)

#include<stdio.h>

#include<stdlib.h>

// Define the structure for a polynomial term

typedefstructTerm {

    intcoefficient;

    intexponent[3]; // Assuming at most 3 variables (x, y, z)

    structTerm\*next;

} Term;

// Function to insert a term into the linked list

voidinsertTerm(Term\*\*poly, intcoef, intexp\_x, intexp\_y, intexp\_z) {

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

    newTerm->coefficient=coef;

    newTerm->exponent[0] =exp\_x;

    newTerm->exponent[1] =exp\_y;

    newTerm->exponent[2] =exp\_z;

    newTerm->next=NULL;

    if (\*poly==NULL) {

        \*poly=newTerm;

    } else {

        Term\*current=\*poly;

        while (current->next!=NULL) {

            current=current->next;

        }

        current->next=newTerm;

    }

}

// Function to add two polynomials

Term\*addPolynomials(Term\*poly1, Term\*poly2) {

    Term\*result=NULL;

    while (poly1!=NULL||poly2!=NULL) {

        if (poly1==NULL||

            (poly2!=NULL&&

             (poly1->exponent[0] <poly2->exponent[0] ||

              (poly1->exponent[0] ==poly2->exponent[0] &&

               (poly1->exponent[1] <poly2->exponent[1] ||

                (poly1->exponent[1] ==poly2->exponent[1] &&

                 poly1->exponent[2] <poly2->exponent[2])))))) {

            insertTerm(&result, poly2->coefficient, poly2->exponent[0], poly2->exponent[1], poly2->exponent[2]);

            poly2=poly2->next;

        } elseif (poly2==NULL||

                   (poly1->exponent[0] >poly2->exponent[0] ||

                    (poly1->exponent[0] ==poly2->exponent[0] &&

                     (poly1->exponent[1] >poly2->exponent[1] ||

                      (poly1->exponent[1] ==poly2->exponent[1] &&

                       poly1->exponent[2] >poly2->exponent[2]))))) {

            insertTerm(&result, poly1->coefficient, poly1->exponent[0], poly1->exponent[1], poly1->exponent[2]);

            poly1=poly1->next;

        } else {

            intsumCoeff=poly1->coefficient+poly2->coefficient;

            if (sumCoeff!=0) {

                insertTerm(&result, sumCoeff, poly1->exponent[0], poly1->exponent[1], poly1->exponent[2]);

            }

            poly1=poly1->next;

            poly2=poly2->next;

        }

    }

    returnresult;

}

// Function to display a polynomial

voiddisplayPolynomial(Term\*poly) {

    if (poly==NULL) {

        printf("0\n");

        return;

    }

    while (poly!=NULL) {

        printf("%d", poly->coefficient);

        if (poly->exponent[0] !=0) {

            printf("x^%d", poly->exponent[0]);

        }

        if (poly->exponent[1] !=0) {

            printf("y^%d", poly->exponent[1]);

        }

        if (poly->exponent[2] !=0) {

            printf("z^%d", poly->exponent[2]);

        }

        if (poly->next!=NULL) {

            printf(" + ");

        }

        poly=poly->next;

    }

    printf("\n");

}

intmain() {

    Term\*poly1=NULL;

    Term\*poly2=NULL;

    // Adding terms to the first polynomial using linked list

    insertTerm(&poly1, 10, 1, 0, 0);

    insertTerm(&poly1, 2, 0, 1, 1);

    // Adding terms to the second polynomial using direct input

    intcoef, exp\_x, exp\_y, exp\_z;

    printf("Enter coefficients and exponents for the second polynomial (enter -1 to stop):\n");

    while (1) {

        printf("Coefficient: ");

        scanf("%d", &coef);

        if (coef==-1) {

            break;

        }

        printf("Exponent for x: ");

        scanf("%d", &exp\_x);

        printf("Exponent for y: ");

        scanf("%d", &exp\_y);

        printf("Exponent for z: ");

        scanf("%d", &exp\_z);

        insertTerm(&poly2, coef, exp\_x, exp\_y, exp\_z);

    }

    printf("First polynomial: ");

    displayPolynomial(poly1);

    printf("Second polynomial: ");

    displayPolynomial(poly2);

    Term\*result=addPolynomials(poly1, poly2);

    printf("Resultant polynomial: ");

    displayPolynomial(result);

    // Clean up memory

    while (poly1!=NULL) {

        Term\*temp=poly1;

        poly1=poly1->next;

        free(temp);

    }

    while (poly2!=NULL) {

        Term\*temp=poly2;

        poly2=poly2->next;

        free(temp);

    }

    while (result!=NULL) {

        Term\*temp=result;

        result=result->next;

        free(temp);

    }

    return0;

}

