// // Q. Represent a 2-variable polynomial using array. Use this representation to implement the addition of polynomials.
// // Also use a header-linked list to manage polynomials stored in an array that can help in efficiently handling dynamic polynomial terms,
// // allowing for easy addition and removal of terms without worrying about fixed array sizes.

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


// // struct Term {
// //     int coefficient;
// //     int exponentX;
// //     int exponentY;
// //     struct Term* next;
// // };


// // struct Polynomial {
// //     struct Term* head;
// // };


// // struct Term* createTerm(int coef, int expX, int expY) {
// //     struct Term* newTerm = (struct Term*)malloc(sizeof(struct Term));
// //     newTerm->coefficient = coef;
// //     newTerm->exponentX = expX;
// //     newTerm->exponentY = expY;
// //     newTerm->next = NULL;
```

```
// //     return newTerm;
// // }


// // struct Polynomial* createPolynomial() {
// //     struct Polynomial* poly = (struct Polynomial*)malloc(sizeof(struct Polynomial));
// //     poly->head = NULL;
// //     return poly;
// // }


// // void insertTerm(struct Polynomial* poly, int coef, int expX, int expY) {
// //     struct Term* newTerm = createTerm(coef, expX, expY);
// //     if (poly->head == NULL || (poly->head->exponentX < expX ||
// //         (poly->head->exponentX == expX && poly->head->exponentY < expY))) {
// //         newTerm->next = poly->head;
// //         poly->head = newTerm;
// //     } else {
// //         struct Term* current = poly->head;
// //         while (current->next != NULL && (current->next->exponentX > expX ||
// //                (current->next->exponentX == expX && current->next->exponentY > expY))) {
// //             current = current->next;
// //         }
// //         newTerm->next = current->next;
// //         current->next = newTerm;
// //     }
// // }


// // struct Polynomial* addPolynomials(struct Polynomial* poly1, struct Polynomial* poly2) {
// //     struct Polynomial* result = createPolynomial();
// //     struct Term* term1 = poly1->head;
// //     struct Term* term2 = poly2->head;

// //     while (term1 != NULL && term2 != NULL) {
// //         if (term1->exponentX == term2->exponentX && term1->exponentY ==
term2->exponentY) {
// //             int sum = term1->coefficient + term2->coefficient;
// //             if (sum != 0) insertTerm(result, sum, term1->exponentX, term1->exponentY);
// //             term1 = term1->next;
// //             term2 = term2->next;
// //         } else if (term1->exponentX > term2->exponentX ||
// //                    (term1->exponentX == term2->exponentX && term1->exponentY >
term2->exponentY)) {
// //             insertTerm(result, term1->coefficient, term1->exponentX, term1->exponentY);
// //             term1 = term1->next;
// //         } else {
// //             insertTerm(result, term2->coefficient, term2->exponentX, term2->exponentY);
```

```
// //             term2 = term2->next;
// //         }
// //     }


// //     while (term1 != NULL) {
// //         insertTerm(result, term1->coefficient, term1->exponentX, term1->exponentY);
// //         term1 = term1->next;
// //     }


// //     while (term2 != NULL) {
// //         insertTerm(result, term2->coefficient, term2->exponentX, term2->exponentY);
// //         term2 = term2->next;
// //     }

// //     return result;
// // }


// // void printPolynomial(struct Polynomial* poly) {
// //     struct Term* current = poly->head;
// //     if (current == NULL) {
// //         printf("0\n");
// //         return;
// //     }
// //     while (current != NULL) {
// //         printf("%d*x^%d*y^%d", current->coefficient, current->exponentX,
current->exponentY);
// //         current = current->next;
// //         if (current != NULL) printf(" + ");
// //     }
// //     printf("\n");
// // }

// // int main() {

// //     struct Polynomial* poly1 = createPolynomial();
// //     struct Polynomial* poly2 = createPolynomial();


// //     insertTerm(poly1, 3, 2, 2);
// //     insertTerm(poly1, 4, 1, 1);


// //     insertTerm(poly2, 5, 2, 2);
// //     insertTerm(poly2, 2, 0, 0);
```

```
// //     printf("Polynomial 1: ");
// //     printPolynomial(poly1);
// //     printf("Polynomial 2: ");
// //     printPolynomial(poly2);


// //     struct Polynomial* sum = addPolynomials(poly1, poly2);

// //     printf("Sum: ");
// //     printPolynomial(sum);


// //     return 0;
// // }









// // #include<stdio.h>

// // struct Node
// // {
// //     int coeff;
// //     int exp;
// //     struct Node* next;
// // };

// // struct Node* createHeaderNode(){
// //     struct Node * header=(struct Node*)malloc(sizeof(struct Node));


// //     if(!header){
// //         printf("Not Defined");
// //     }
// //     header->coeff=0;
// //     header->exp=-1;
// //     header->next=NULL;
```

```
// //     void addPolynomial(struct Node* p1, struct Node* p2){
// //         struct Node* temp1=p1->next;
// //         struct Node* temp2=p2->next;
// //         struct Node* result=createHeaderNode();
// //     }
// // }

// // int main(){
// //     struct Node* p1=createHeaderNode();
// //     struct Node* p2=createHeaderNode();

// //     printf("\nFOR POLYNOMIAL 1:");
// //     insert(p1,4,3);
// //     insert(p1,3,2);
// //     insert(p1,2,1);
// //     insert(p1,4,0);



// //     printf("\nFOR POLYNOMIAL 2:");
// //     insert(p1,4,3);
// //     insert(p1,3,2);
// //     insert(p1,2,1);
// //     insert(p1,4,0);
// // }







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

// // Define the structure for a node representing a term in the polynomial
// struct Node {
//     int coeff;     // Coefficient of the term
//     int exp;       // Exponent of the term
//     struct Node* next;  // Pointer to the next node (term)
// };

// // Function to create a header node for a polynomial
// struct Node* createHeaderNode() {
//     struct Node* header = (struct Node*)malloc(sizeof(struct Node));
```

```
//     if (!header) {
//         printf("Memory allocation failed!\n");
//         return NULL;
//     }
//     header->coeff = 0;
//     header->exp = -1;
//     header->next = NULL;
//     return header;
// }

// // Function to insert a term into the polynomial
// void insert(struct Node* header, int coeff, int exp) {
//     struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
//     temp->coeff = coeff;
//     temp->exp = exp;
//     temp->next = NULL;

//     // Find the correct position to insert the term in decreasing order of exponents
//     struct Node* current = header;
//     while (current->next != NULL && current->next->exp > exp) {
//         current = current->next;
//     }

//     // If the term with the same exponent already exists, add the coefficients
//     if (current->next != NULL && current->next->exp == exp) {
//         current->next->coeff += coeff;
//     } else {
//         // Insert the new term
//         temp->next = current->next;
//         current->next = temp;
//     }
// }

// // Function to add two polynomials
// struct Node* addPolynomial(struct Node* p1, struct Node* p2) {
//     struct Node* result = createHeaderNode();
//     struct Node* temp1 = p1->next;  // Pointer to traverse the first polynomial
//     struct Node* temp2 = p2->next;  // Pointer to traverse the second polynomial

//     // Traverse both polynomials and add the corresponding terms
//     while (temp1 != NULL && temp2 != NULL) {
//         if (temp1->exp == temp2->exp) {
//             insert(result, temp1->coeff + temp2->coeff, temp1->exp);
//             temp1 = temp1->next;
//             temp2 = temp2->next;
//         } else if (temp1->exp > temp2->exp) {
//             insert(result, temp1->coeff, temp1->exp);
//             temp1 = temp1->next;
```

```
//         } else {
//             insert(result, temp2->coeff, temp2->exp);
//             temp2 = temp2->next;
//         }
//     }

//     // If there are remaining terms in the first polynomial
//     while (temp1 != NULL) {
//         insert(result, temp1->coeff, temp1->exp);
//         temp1 = temp1->next;
//     }

//     // If there are remaining terms in the second polynomial
//     while (temp2 != NULL) {
//         insert(result, temp2->coeff, temp2->exp);
//         temp2 = temp2->next;
//     }

//     return result;
// }

// // Function to print a polynomial
// void printPolynomial(struct Node* poly) {
//     struct Node* temp = poly->next; // Skip the header node
//     if (temp == NULL) {
//         printf("0\n");
//         return;
//     }

//     while (temp != NULL) {
//         printf("%d*x^%d", temp->coeff, temp->exp);
//         temp = temp->next;
//         if (temp != NULL) {
//             printf(" + ");
//         }
//     }
//     printf("\n");
// }

// int main() {
//     // Create two polynomials
//     struct Node* p1 = createHeaderNode();
//     struct Node* p2 = createHeaderNode();

//     // Add terms to polynomial 1 (4*x^3 + 3*x^2 + 2*x^1 + 4*x^0)
//     printf("\nFOR POLYNOMIAL 1:\n");
//     insert(p1, 4, 3);
//     insert(p1, 3, 2);
```

```
//     insert(p1, 2, 1);
//     insert(p1, 4, 0);
//     printPolynomial(p1);

//     // Add terms to polynomial 2 (4*x^3 + 3*x^2 + 2*x^1 + 4*x^0)
//     printf("\nFOR POLYNOMIAL 2:\n");
//     insert(p2, 4, 3);
//     insert(p2, 3, 2);
//     insert(p2, 2, 1);
//     insert(p2, 4, 0);
//     printPolynomial(p2);

//     // Add the two polynomials
//     struct Node* sum = addPolynomial(p1, p2);

//     // Print the result
//     printf("\nSum of Polynomial 1 and Polynomial 2:\n");
//     printPolynomial(sum);

//     // Free memory (for simplicity, freeing not shown here)

//     return 0;
// }
```