Rajalakshmi Engineering College

Name: MITHUN CHAKRAVARTHY
Email: 241801158@rajalakshmi.edu.in
Roll no: 241801158
Phone: 9360341516
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_MCQ

Attempt : 1
Total Mark : 10
Marks Obtained : 9

Section 1 : MCQ

1. Linked lists are not suitable for the implementation of?

Answer

Binary search

Status : Correct ***Marks : 1/1***

2. Which of the following statements is used to create a new node in a singly linked list?

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

NODE *ptr;

Answer

ptr = (NODE*)malloc(sizeof(NODE));

Status : Correct ***Marks : 1/1***

3. Given the linked list: 5 -> 10 -> 15 -> 20 -> 25 -> NULL. What will be the output of traversing the list and printing each node's data?

Answer

5 10 15 20 25

Status : Correct ***Marks : 1/1***

4. Consider the singly linked list: 13 -> 4 -> 16 -> 9 -> 22 -> 45 -> 5 -> 16 -> 6, and an integer K = 10, you need to delete all nodes from the list that are less than the given integer K.

What will be the final linked list after the deletion?

Answer

13 -> 16 -> 22 -> 45 -> 16

Status : Correct ***Marks : 1/1***

5. Given a pointer to a node X in a singly linked list. If only one point is given and a pointer to the head node is not given, can we delete node X from the given linked list?

Answer

Possible if X is not last node.

Status : Correct ***Marks : 1/1***

6. The following function reverse() is supposed to reverse a singly linked list. There is one line missing at the end of the function.

What should be added in place of "/*ADD A STATEMENT HERE*/", so that the function correctly reverses a linked list?

```
struct node {
   int data;
   struct node* next;
};
static void reverse(struct node** head_ref) {
   struct node* prev   = NULL;
   struct node* current = *head_ref;
   struct node* next;
   while (current != NULL) {
      next  = current->next;
      current->next = prev;
      prev = current;
      current = next;
   }
   /*ADD A STATEMENT HERE*/
}
```

Answer

*head_ref = prev;

Status : Correct ***Marks : 1/1***

7. In a singly linked list, what is the role of the "tail" node?

Answer

It stores the last element of the list

Status : Correct ***Marks : 1/1***

8. Consider the singly linked list: 15 -> 16 -> 6 -> 7 -> 17. You need to delete all nodes from the list which are prime.

What will be the final linked list after the deletion?

Answer

16 -> 6

Status : Wrong **Marks : 0/1**

9. The following function takes a singly linked list of integers as a parameter and rearranges the elements of the lists.

The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution?

```
struct node {
   int value;
   struct node* next;
};

void rearrange (struct node* list) {
   struct node *p,q;
   int temp;
   if (! List || ! list->next) return;
   p=list; q=list->next;
   while(q) {
      temp=p->value; p->value=q->value;
      q->value=temp;p=q->next;
      q=p?p->next:0;
   }
}
```

Answer

2, 1, 4, 3, 6, 5, 7

Status : Correct **Marks : 1/1**

10. Consider an implementation of an unsorted singly linked list. Suppose it has its representation with a head pointer only. Given the representation, which of the following operations can be implemented in O(1) time?

i) Insertion at the front of the linked list

ii) Insertion at the end of the linked list

iii) Deletion of the front node of the linked list

iv) Deletion of the last node of the linked list

Answer

I and III

Status : Correct ***Marks : 1/1***

241801158

Rajalakshmi Engineering College

Name: MITHUN CHAKRAVARTHY
Email: 241801158@rajalakshmi.edu.in
Roll no: 241801158
Phone: 9360341516
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Janani is a tech enthusiast who loves working with polynomials. She wants to create a program that can add polynomial coefficients and provide the sum of their coefficients.

The polynomials will be represented as a linked list, where each node of the linked list contains a coefficient and an exponent. The polynomial is represented in the standard form with descending order of exponents.

Input Format

The first line of input consists of an integer n, representing the number of terms in the first polynomial.

The following n lines of input consist of two integers each: the coefficient and the exponent of the term in the first polynomial.

The next line of input consists of an integer m, representing the number of terms in the second polynomial.

The following m lines of input consist of two integers each: the coefficient and the exponent of the term in the second polynomial.

Output Format

The output prints the sum of the coefficients of the polynomials.

Sample Test Case

Input: 3
2 2
3 1
4 0
3
2 2
3 1
4 0

Output: 18

Answer

```
// You are using GCC
#include<stdio.h>
#include<stdlib.h>
typedef struct Poly{
    int coeff;
    int expon;
    struct Poly* next;
}Node;
Node* newnode(int coeff,int expon){
    Node* new_node = (Node*) malloc(sizeof(Node));
    new_node->coeff = coeff;
    new_node->expon = expon;
    new_node->next = NULL;
    return new_node;
}
void insertNode(Node** head, int coeff,int expon){
    Node* temp = *head;
    if(temp == NULL){
        *head = newnode(coeff,expon);
        return;
```

```
    }
    while(temp->next != NULL){
        temp = temp->next;

    }
    temp->next = newnode(coeff,expon);

}
int main()
{
    int n,coeff,expon;
    scanf("%d",&n);
    Node* poly1;
    Node* poly2;
    for(int i=0; i<n ; i++)
    {
        scanf("%d %d",&coeff,&expon);
        insertNode(&poly1,coeff,expon);
    }
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d %d",&coeff,&expon);
        insertNode(&poly2,coeff,expon);
    }
    int sum = 0;
    while(poly1 != NULL)
    {
        sum += poly1->coeff;
        poly1 = poly1->next;
    }
    while(poly2 != NULL)
    {
        sum += poly2->coeff;
        poly2 = poly2->next;
    }
    printf("%d",sum);
}
```

Status : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: MITHUN CHAKRAVARTHY
Email: 241801158@rajalakshmi.edu.in
Roll no: 241801158
Phone: 9360341516
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_week 1_CY

Attempt : 2
Total Mark : 30
Marks Obtained : 30

Section 1 : Coding

1. Problem Statement

John is working on a math processing application, and his task is to simplify polynomials entered by users. The polynomial is represented as a linked list, where each node contains two properties:

Coefficient of the term.

Exponent of the term.

John's goal is to combine all the terms that have the same exponent, effectively simplifying the polynomial.

Input Format

The first line of input consists of an integer representing the number of terms in the polynomial.

The next n lines of input consist of two integers, representing the coefficient and exponent of the polynomial in each line separated by space.

Output Format

The first line of output prints the original polynomial in the format 'cx^e + cx^e + ...' (where c is the coefficient and e is the exponent of each term).

The second line of output displays the simplified polynomial in the same format as the original polynomial.

If the polynomial is 0, then only '0' will be printed.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
5 2
3 1
6 2
Output: Original polynomial: 5x^2 + 3x^1 + 6x^2
Simplified polynomial: 11x^2 + 3x^1

Answer

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

struct Node {
    int coefficient;
    int exponent;
    struct Node* next;
};

struct Node* createNode(int coef, int exp) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coefficient = coef;
    newNode->exponent = exp;
    newNode->next = NULL;
    return newNode;
}
```

```
struct Polynomial {
    struct Node* head;
};

void initPolynomial(struct Polynomial* polynomial) {
    polynomial->head = NULL;
}

void addTerm(struct Polynomial* polynomial, int coef, int exp) {
    struct Node* newNode = createNode(coef, exp);

    if (polynomial->head == NULL) {
        polynomial->head = newNode;
    } else {
        struct Node* ptr = polynomial->head;
        while (ptr->next != NULL) {
            ptr = ptr->next;
        }
        ptr->next = newNode;
    }
}

void displayPolynomial(struct Polynomial* polynomial) {
    if (polynomial->head == NULL) {
        printf("0");
        return;
    }
    struct Node* current = polynomial->head;
    while (current != NULL) {
        printf("%dx^%d", current->coefficient, current->exponent);
        if (current->next != NULL) {
            printf(" + ");
        }
        current = current->next;
    }
}

void simplifyPolynomial(struct Polynomial* polynomial) {
    if (polynomial->head == NULL || polynomial->head->next == NULL) {
        return;
    }
```

```
    struct Node* ptr1 = polynomial->head;
    struct Node* ptr2;
    struct Node* prev = NULL;

    while (ptr1 != NULL) {
        ptr2 = ptr1->next;
        prev = ptr1;
        while (ptr2 != NULL) {
            if (ptr1->exponent == ptr2->exponent) {
                ptr1->coefficient += ptr2->coefficient;
                prev->next = ptr2->next;
                free(ptr2);
                ptr2 = prev->next;
            } else {
                prev = ptr2;
                ptr2 = ptr2->next;
            }
        }
        ptr1 = ptr1->next;
    }
}

int main() {
    struct Polynomial polynomial;
    initPolynomial(&polynomial);

    int terms;
    scanf("%d", &terms);

    for (int i = 0; i < terms; i++) {
        int coefficient, exponent;
        scanf("%d %d", &coefficient, &exponent);
        addTerm(&polynomial, coefficient, exponent);
    }

    printf("Original polynomial: ");
    displayPolynomial(&polynomial);

    printf("\nSimplified polynomial: ");
    simplifyPolynomial(&polynomial);
    displayPolynomial(&polynomial);
```

241801158

```
    return 0;
}
```

Status : Correct ***Marks : 10/10***

2. Problem Statement

Akila is a tech enthusiast and wants to write a program to add two polynomials. Each polynomial is represented as a linked list, where each node in the list represents a term in the polynomial.

A term in the polynomial is represented in the format ax^b, where a is the coefficient and b is the exponent.

Akila needs your help to implement a program that takes two polynomials as input, adds them, and stores the result in ascending order in a new polynomial-linked list. Write a program to help her.

Input Format

The input consists of lines containing pairs of integers representing the coefficients and exponents of polynomial terms.

Each line represents a single term, with the coefficient and exponent separated by a space.

The input for each polynomial ends with a line containing "0 0".

Output Format

The output consists of three lines representing the first, second, and resulting polynomial after the addition operation, with terms sorted in ascending order of exponents.

Each line contains terms of the polynomial in the format "coefficientx^exponent", separated by " + ".

If the resulting polynomial is zero, the output is "0".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 3 4
2 3
1 2
0 0
1 2
2 3
3 4
0 0

Output: 1x^2 + 2x^3 + 3x^4
1x^2 + 2x^3 + 3x^4
2x^2 + 4x^3 + 6x^4

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

struct PolynomialTerm {
    int coefficient;
    int exponent;
    struct PolynomialTerm* next;
};

typedef struct PolynomialTerm PolynomialTerm;

PolynomialTerm* createTermNode(int coefficient, int exponent) {
    PolynomialTerm* newNode =
(PolynomialTerm*)malloc(sizeof(PolynomialTerm));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;
    return newNode;
}

void insertTerm(PolynomialTerm** poly, int coefficient, int exponent) {
    PolynomialTerm* newNode = createTermNode(coefficient, exponent);
    if (*poly == NULL) {
        *poly = newNode;
```

```
        return;
    }

    if (exponent < (*poly)->exponent) {
        newNode->next = *poly;
        *poly = newNode;
        return;
    }

    PolynomialTerm* current = *poly;
    PolynomialTerm* prev = NULL;
    while (current && current->exponent <= exponent) {
        prev = current;
        current = current->next;
    }
    prev->next = newNode;
    newNode->next = current;
}

PolynomialTerm* addPolynomials(PolynomialTerm* poly1, PolynomialTerm*
poly2) {
    PolynomialTerm* result = NULL;
    PolynomialTerm* tail = NULL;

    while (poly1 || poly2) {
        int coeff = 0;
        int exp = 0;

        if (poly1 && poly2) {
            if (poly1->exponent == poly2->exponent) {
                coeff = poly1->coefficient + poly2->coefficient;
                exp = poly1->exponent;
                poly1 = poly1->next;
                poly2 = poly2->next;
            } else if (poly1->exponent > poly2->exponent) {
                coeff = poly1->coefficient;
                exp = poly1->exponent;
                poly1 = poly1->next;
            } else {
                coeff = poly2->coefficient;
                exp = poly2->exponent;
                poly2 = poly2->next;
```

```
        }
    } else if (poly1) {
        coeff = poly1->coefficient;
        exp = poly1->exponent;
        poly1 = poly1->next;
    } else {
        coeff = poly2->coefficient;
        exp = poly2->exponent;
        poly2 = poly2->next;
    }

    if (coeff != 0) {
        insertTerm(&result, coeff, exp);
    }
}

return result;
}

void printPolynomial(PolynomialTerm* polynomial) {
    if (polynomial == NULL) {
        printf("0");
        return;
    }

    while (polynomial != NULL) {
        if (polynomial->coefficient != 0) {
            printf("%dx^%d", polynomial->coefficient, polynomial->exponent);
            if (polynomial->next != NULL) {
                printf(" + ");
            }
        }
        polynomial = polynomial->next;
    }
    printf("\n");
}

void freePolynomial(PolynomialTerm* polynomial) {
    while (polynomial != NULL) {
        PolynomialTerm* temp = polynomial;
        polynomial = polynomial->next;
        free(temp);
```

```
    }
}

int main() {
    int coefficient, exponent;
    PolynomialTerm* poly1 = NULL;
    PolynomialTerm* poly2 = NULL;

    while (1) {
        scanf("%d %d", &coefficient, &exponent);
        if (coefficient == 0 && exponent == 0) {
            break;
        }
        insertTerm(&poly1, coefficient, exponent);
    }

    while (1) {
        scanf("%d %d", &coefficient, &exponent);
        if (coefficient == 0 && exponent == 0) {
            break;
        }
        insertTerm(&poly2, coefficient, exponent);
    }

    printPolynomial(poly1);
    printPolynomial(poly2);

    PolynomialTerm* result = addPolynomials(poly1, poly2);
    printPolynomial(result);

    freePolynomial(poly1);
    freePolynomial(poly2);
    freePolynomial(result);

    return 0;
}
```

Status : Correct ***Marks : 10/10***

3. Problem Statement

Timothy wants to evaluate polynomial expressions for his mathematics homework. He needs a program that allows him to input the coefficients of a polynomial based on its degree and compute the polynomial's value for a given input of x. Implement a function that takes the degree, coefficients, and the value of x, and returns the evaluated result of the polynomial.

Example

Input:

degree of the polynomial = 2

coefficient of x2 = 13

coefficient of x1 = 12

coefficient of x0 = 11

x = 1

Output:

36

Explanation:

Calculate the value of 13x2: 13 * 12 = 13.

Calculate the value of 12x1: 12 * 11 = 12.

Calculate the value of 11x0: 11 * 10 = 11.

Add the values of x2, x1, and x0 together: 13 + 12 + 11 = 36.

Input Format

The first line of input consists of an integer representing the degree of the polynomial.

The second line consists of an integer representing the coefficient of x2.

The third line consists of an integer representing the coefficient of x1.

The fourth line consists of an integer representing the coefficient of x0.

The fifth line consists of an integer representing the value of x, at which the

polynomial should be evaluated.

Output Format

The output is an integer value obtained by evaluating the polynomial at the given value of x.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 2
13
12
11
1
Output: 36

Answer

```
#include <stdio.h>
#include <math.h>

int main() {
    int degree, x;

    scanf("%d", &degree);


    int coefficients[degree + 1];


    for (int i = 0; i <= degree; i++) {
        scanf("%d", &coefficients[i]);
    }


    scanf("%d", &x);


    int result = 0;
    for (int i = 0; i <= degree; i++) {
```

2418011158

```
        result += coefficients[i] * pow(x, degree - i);
    }

    printf("%d\n", result);

    return 0;
}
```

Status : Correct ***Marks : 10/10***

2418011158