Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_MCQ

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : MCQ

1. 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***

2. 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***

3. 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

15 -&gt; 16 -&gt; 6

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

4. 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***

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

Answer

Binary search

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

6. 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***

7. 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***

8. 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***

9. 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 -&gt; 16 -&gt; 22 -&gt; 45 -&gt; 16

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

10. 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***

Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

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

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

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

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

void insertTerm(Node** head, int coef, int exp) {
   Node* newNode = createNode(coef, exp);
   if (*head == NULL) {
       *head = newNode;
   } else {
       Node* current = *head;
       while (current->next != NULL) {
           current = current->next;
       }
       current->next = newNode;
   }
}

Node* addPolynomials(Node* poly1, Node* poly2) {
   Node* result = NULL;
   Node* current = NULL;

   while (poly1 != NULL && poly2 != NULL) {
       if (poly1->exponent == poly2->exponent) {
           int sumCoefficients = poly1->coefficient + poly2->coefficient;
           if (sumCoefficients != 0) {
               insertTerm(&result, sumCoefficients, poly1->exponent);
               if (current == NULL) {
                   current = result;
               } else {
                   current = current->next;
               }
           }
           poly1 = poly1->next;
           poly2 = poly2->next;
       } else if (poly1->exponent > poly2->exponent) {
           insertTerm(&result, poly1->coefficient, poly1->exponent);
           if (current == NULL) {
               current = result;
           } else {
               current = current->next;
           }
           poly1 = poly1->next;
       } else {
           insertTerm(&result, poly2->coefficient, poly2->exponent);
           if (current == NULL) {
               current = result;
           } else {
               current = current->next;
           }
           poly2 = poly2->next;
       }
   }

   while (poly1 != NULL) {
       insertTerm(&result, poly1->coefficient, poly1->exponent);
       if (current == NULL) {
           current = result;
       } else {
           current = current->next;
       }
       poly1 = poly1->next;
   }

   while (poly2 != NULL) {
       insertTerm(&result, poly2->coefficient, poly2->exponent);
       if (current == NULL) {
           current = result;
       } else {
           current = current->next;
       }
       poly2 = poly2->next;
   }

   return result;
}

void printPolynomial(Node* poly) {
   if (poly == NULL) {
       printf("0\n");
       return;
   }

   while (poly != NULL) {
       if (poly->coefficient != 0) {
           if (poly->exponent > 1) {
               printf("%dx^%d", poly->coefficient, poly->exponent);
           } else if (poly->exponent == 1) {
               printf("%dx", poly->coefficient);
           } else {
               printf("%d", poly->coefficient);
           }
           if (poly->next != NULL && poly->next->coefficient > 0) {
               printf(" + ");
           }
       }
       poly = poly->next;
   }

   printf("\n");
}

void freeLinkedList(Node* head) {
   Node* current = head;
   while (current != NULL) {
       Node* temp = current;
       current = current->next;
       free(temp);
   }
}

int main() {
   Node* poly1 = NULL;
   Node* poly2 = NULL;

   int num_terms_poly1, num_terms_poly2;

   scanf("%d", &num_terms_poly1);

   for (int i = 0; i < num_terms_poly1; i++) {
       int coef, exp;
       scanf("%d%d", &coef, &exp);
       insertTerm(&poly1, coef, exp);
   }

   scanf("%d", &num_terms_poly2);

   for (int i = 0; i < num_terms_poly2; i++) {
       int coef, exp;
       scanf("%d%d", &coef, &exp);
       insertTerm(&poly2, coef, exp);
   }
   Node* result = addPolynomials(poly1, poly2);

   int sum = 0;
   Node* current = result;
   while (current != NULL) {
       sum += current->coefficient;
       current = current->next;
   }

   printf("%d", sum);

   freeLinkedList(poly1);
   freeLinkedList(poly2);
   freeLinkedList(result);

   return 0;
}
```

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

Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Arun is learning about data structures and algorithms. He needs your help in solving a specific problem related to a singly linked list.

Your task is to implement a program to delete a node at a given position. If the position is valid, the program should perform the deletion; otherwise, it should display an appropriate message.

Input Format

The first line of input consists of an integer N, representing the number of elements in the linked list.

The second line consists of N space-separated elements of the linked list.

The third line consists of an integer x, representing the position to delete.

Position starts from 1.

Output Format

The output prints space-separated integers, representing the updated linked list after deleting the element at the given position.

If the position is not valid, print "Invalid position. Deletion not possible."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
8 2 3 1 7
2
Output: 8 3 1 7

Answer

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

void insert(int);
void display_List();
void deleteNode(int);

struct node {
   int data;
   struct node* next;
} *head = NULL, *tail = NULL;

void deleteNode(int pos) {
   if (pos <= 0) {
      printf("Invalid position. Deletion not possible.");
      return;
   }

   struct node* temp = head;
   struct node* prev = NULL;
   int i;
```

```
    for (i = 1; i < pos && temp != NULL; i++) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Invalid position. Deletion not possible.");
        return;
    }

    if (prev == NULL) {
        head = head->next;
        free(temp);
    } else {
        prev->next = temp->next;
        free(temp);
    }

    display_List();
    return;
}

void insert(int value) {
    struct node* newnode;
    newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = value;
    newnode->next = NULL;

    if (head == NULL) {
        head = newnode;
        tail = newnode;
    } else {
        tail->next = newnode;
        tail = newnode;
    }
    return;
}

void display_List() {
    struct node* temp;
    temp = head;
    while (temp != NULL) {
        if (temp->next == NULL) {
            printf("%d ", temp->data);
        } else {
            printf("%d ", temp->data);
        }
        temp = temp->next;
    }
    return;
}


int main() {
    int num_elements, element, pos_to_delete;

    scanf("%d", &num_elements);

    for (int i = 0; i < num_elements; i++) {
        scanf("%d", &element);
        insert(element);
    }

    scanf("%d", &pos_to_delete);

    deleteNode(pos_to_delete);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Imagine you are working on a text processing tool and need to implement a feature that allows users to insert characters at a specific position.

Implement a program that takes user inputs to create a singly linked list of characters and inserts a new character after a given index in the list.

Input Format

The first line of input consists of an integer N, representing the number of characters in the linked list.

The second line consists of a sequence of N characters, representing the linked list.

The third line consists of an integer index, representing the index(0-based) after which the new character node needs to be inserted.

The fourth line consists of a character value representing the character to be inserted after the given index.

Output Format

If the provided index is out of bounds (larger than the list size):

1. The first line of output prints "Invalid index".
2. The second line prints "Updated list: " followed by the unchanged linked list values.

Otherwise, the output prints "Updated list: " followed by the updated linked list after inserting the new character after the given index.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
a b c d e
2
X
Output: Updated list: a b c X d e

Answer

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

struct Node {
   char data;
   struct Node* next;
};

struct Node* createNode(char value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;
```

```
    return newNode;
}

struct Node* insertAfterIndex(struct Node* head, int index, char value) {
    struct Node* newNode = createNode(value);

    if (index == -1) {
        newNode->next = head;
        return newNode;
    }

    struct Node* current = head;
    for (int i = 0; i < index; i++) {
        if (current == NULL) {
            printf("Invalid index\n");
            return head;
        }
        current = current->next;
    }

    if (current != NULL) {
        newNode->next = current->next;
        current->next = newNode;
    } else {
        printf("Invalid index\n");
    }

    return head;
}

void displayList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%c ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int n;
    char value;

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        scanf(" %c", &value);

        struct Node* newNode = createNode(value);
        if (head == NULL) {
            head = newNode;
        } else {
            struct Node* current = head;
            while (current->next != NULL) {
                current = current->next;
            }
            current->next = newNode;
        }
    }

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

    scanf(" %c", &value);

    head = insertAfterIndex(head, index, value);

    printf("Updated list: ");
    displayList(head);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

As part of a programming assignment in a data structures course, students are required to create a program to construct a singly linked list by inserting elements at the beginning.

You are an evaluator of the course and guide the students to complete the task.

Input Format

The first line of input consists of an integer N, which is the number of elements.

The second line consists of N space-separated integers.

Output Format

The output prints the singly linked list elements, after inserting them at the beginning.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
78 89 34 51 67

Output: 67 51 34 89 78

Answer

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

struct Node {
   int data;
   struct Node* next;
};


void insertAtFront(struct Node** head, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = *head; // Make the new node point to the current head
   *head = newNode;      // Update the head to point to the new node
}

// Function to print the linked list
void printList(struct Node* head) {
   struct Node* current = head;
   while (current != NULL) {
      printf("%d ", current->data);
      current = current->next;
   }
   printf("\n");
}

int main(){
   struct Node* head = NULL;
```

```
    int n;
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        int activity;
        scanf("%d", &activity);
        insertAtFront(&head, activity);
    }

    printList(head);
    struct Node* current = head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Imagine you are tasked with developing a simple GPA management system using a singly linked list. The system allows users to input student GPA values, insertion should happen at the front of the linked list, delete record by position, and display the updated list of student GPAs.

Input Format

The first line of input contains an integer n, representing the number of students.

The next n lines contain a single floating-point value representing the GPA of each student.

The last line contains an integer position, indicating the position at which a student record should be deleted. Position starts from 1.

Output Format

After deleting the data in the given position, display the output in the format "GPA: " followed by the GPA value, rounded off to one decimal place.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
3.8
3.2
3.5
4.1
2
Output: GPA: 4.1
GPA: 3.2
GPA: 3.8

Answer

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

typedef struct Node {
   float gpa;
   struct Node* next;
} Node;

Node* insertAtFront(Node* head, float gpa) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->gpa = gpa;
   newNode->next = head;
   return newNode;
}

Node* deleteAtPosition(Node* head, int position, int n) {
  if (head == NULL) return head;
   if (position < 1 || position > n) return head;
```

```
 if (position == 1) {
        Node *temp = head;
        head = head->next;
        free(temp);
        return head;
    }

    Node* current = head;
    Node* previous = NULL;
    int count = 1;

    while (current != NULL && count < position) {
        previous = current;
        current = current->next;
        count++;
    }


    if (current == NULL) return head;

    previous->next = current->next;
    free(current);
    return head;
}

void displayList(Node* head) {
    Node* current = head;
    while (current != NULL) {
        printf("GPA: %.1f\n", current->gpa);
        current = current->next;
    }
}

void freeList(Node* head) {
    Node* current = head;
    Node* nextNode;
    while (current != NULL) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
```

```
}

int main() {
    int n, position;
    float gpa;
    Node* head = NULL;

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        scanf("%f", &gpa);
        head = insertAtFront(head, gpa);
    }

    scanf("%d", &position);

    head = deleteAtPosition(head, position, n);


    int count = 0;
    Node* temp = head;
    while(temp!=NULL){
      count++;
      temp = temp->next;
    }

    displayList(head);

    freeList(head);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 6

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

John is tasked with creating a program to manage student roll numbers using a singly linked list.

Write a program for John that accepts students' roll numbers, inserts them at the end of the linked list, and displays the numbers.

Input Format

The first line of input consists of an integer N, representing the number of students.

The second line consists of N space-separated integers, representing the roll numbers of students.

Output Format

The output prints the space-separated integers singly linked list, after inserting the roll numbers of students at the end.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
23 85 47 62 31

Output: 23 85 47 62 31

Answer

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

struct Node {
   int data;
   struct Node* next;
};

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

void insertEnd(struct Node** head, int data) {
   struct Node* newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
      return;
   }
   struct Node* temp = *head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
}
```

```
void displayList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int n, roll;
    struct Node* head = NULL;

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        scanf("%d", &roll);
        insertEnd(&head, roll);
    }

    displayList(head);

    // Free the allocated memory (optional but good practice)
    struct Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_COD_Question 7

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Dev is tasked with creating a program that efficiently finds the middle element of a linked list. The program should take user input to populate the linked list by inserting each element into the front of the list and then determining the middle element.

Assist Dev, as he needs to ensure that the middle element is accurately identified from the constructed singly linked list:

If it's an odd-length linked list, return the middle element.If it's an even-length linked list, return the second middle element of the two elements.

Input Format

The first line of input consists of an integer n, representing the number of elements in the linked list.

The second line consists of n space-separated integers, representing the elements of the list.

Output Format

The first line of output displays the linked list after inserting elements at the front.

The second line displays "Middle Element: " followed by the middle element of the linked list.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50

Output: 50 40 30 20 10
Middle Element: 30

Answer

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};


struct Node* push(struct Node* head, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = head;
   return newNode;
}

// Function to find the middle element of the linked list
int printMiddle(struct Node* head) {
   struct Node *slowPtr = head, *fastPtr = head;
```

```
    while (fastPtr != NULL && fastPtr->next != NULL) {
        slowPtr = slowPtr->next;
        fastPtr = fastPtr->next->next;
    }
    return slowPtr->data;
}


int main() {
    struct Node* head = NULL;
    int n;

    scanf("%d", &n);
    int value;

    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        head = push(head, value);
    }

    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");


    int middle_element = printMiddle(head);
    printf("Middle Element: %d\n", middle_element);


    current = head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 1_PAH_modified

Attempt : 1
Total Mark : 5
Marks Obtained : 4

Section 1 : Coding

1. Problem Statement

Write a program to manage a singly linked list. The program should allow users to perform various operations on the linked list, such as inserting elements at the beginning or end, deleting elements from the beginning or end, inserting before or after a specific value, and deleting elements before or after a specific value. After each operation, the updated linked list should be displayed.

Input Format

The first line contains an integer choice, representing the operation to perform:

- For choice 1 to create the linked list. The next lines contain space-separated integers, with -1 indicating the end of input.
- For choice 2 to display the linked list.
- For choice 3 to insert a node at the beginning. The next line contains an integer

data representing the value to insert.
- For choice 4 to insert a node at the end. The next line contains an integer data representing the value to insert.
- For choice 5 to insert a node before a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 6 to insert a node after a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 7 to delete a node from the beginning.
- For choice 8 to delete a node from the end.
- For choice 9 to delete a node before a specific value. The next line contains an integer value representing the node before which deletion occurs.
- For choice 10 to delete a node after a specific value. The next line contains an integer value representing the node after which deletion occurs.
- For choice 11 to exit the program.

Output Format

For choice 1, print "LINKED LIST CREATED".

For choice 2, print the linked list as space-separated integers on a single line. If the list is empty, print "The list is empty".

For choice 3, 4, 5, and 6, print the updated linked list with a message indicating the insertion operation.

For choice 7, 8, 9, and 10, print the updated linked list with a message indicating the deletion operation.

For any operation that is not possible print an appropriate error message such as "Value not found in the list".

For choice 11 terminate the program.

For any invalid option, print "Invalid option! Please try again".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
5

3
7
-1
2
11
Output: LINKED LIST CREATED
5 3 7

Answer

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

struct Node {
   int data;
   struct Node* next;
};

struct Node* head = NULL;

void insertAtBeginning(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = head;
   head = newNode;
}

void insertAtEnd(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
      return;
   }

   struct Node* temp = head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
```

```
}

void insertBefore(int value, int data) {
    if (head == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    if (head->data == value) {
        insertAtBeginning(data);
        return;
    }

    struct Node* temp = head;
    struct Node* prev = NULL;

    while (temp != NULL && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = temp;
    prev->next = newNode;
}

void insertAfter(int value, int data) {
    struct Node* temp = head;

    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = temp->next;
    temp->next = newNode;
}

void deleteFromBeginning() {
    if (head == NULL) return;

    struct Node* temp = head;
    head = head->next;
    free(temp);
}

void deleteFromEnd() {
    if (head == NULL) return;

    if (head->next == NULL) {
        free(head);
        head = NULL;
        return;
    }

    struct Node* temp = head;
    struct Node* prev = NULL;

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

    prev->next = NULL;
    free(temp);
}

void deleteBefore(int value) {
    if (head == NULL || head->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }
    if (head->next->data == value) {
        deleteFromBeginning();
        return;
    }

    struct Node* temp = head;
    struct Node* prev = NULL;
    struct Node* prev2 = NULL;

    while (temp->next != NULL && temp->next->data != value) {
        prev2 = prev;
        prev = temp;
        temp = temp->next;
    }

    if (temp->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    if (prev2 == NULL) {
      head = temp->next;
      free(temp);
    } else {
      prev2->next = temp->next;
      free(temp);
    }
}

void deleteAfter(int value) {
    struct Node* temp = head;

    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }
    struct Node* toDelete = temp->next;
    temp->next = temp->next->next;
    free(toDelete);
}

void displayList() {
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }

    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, data, value;

    while (1) {
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int input;
                while (1) {
                    scanf("%d", &input);
                    if (input == -1) break;
                    insertAtEnd(input);
                }
                printf("LINKED LIST CREATED\n");
                break;
            }
            case 2:
                displayList();
                break;
            case 3:
                scanf("%d", &data);
                insertAtBeginning(data);
                printf("The linked list after insertion at the beginning is:\n");
                displayList();
                break;
            case 4:
                scanf("%d", &data);
                insertAtEnd(data);
                printf("The linked list after insertion at the end is:\n");
                displayList();
                break;
            case 5:
                scanf("%d %d", &value, &data);
                insertBefore(value, data);
                printf("The linked list after insertion before a value is:\n");
                displayList();
                break;
            case 6:
                scanf("%d %d", &value, &data);
                insertAfter(value, data);
                printf("The linked list after insertion after a value is:\n");
                displayList();
                break;
            case 7:
                deleteFromBeginning();
                printf("The linked list after deletion from the beginning is:\n");
                displayList();
                break;
            case 8:
                deleteFromEnd();
                printf("The linked list after deletion from the end is:\n");
                displayList();
                break;
            case 9:
                scanf("%d", &value);
                deleteBefore(value);
                printf("The linked list after deletion before a value is:\n");
                displayList();
                break;
            case 10:
                scanf("%d", &value);
                deleteAfter(value);
                printf("The linked list after deletion after a value is:\n");
                displayList();
```

```
                break;
            case 11:
                exit(0);
            default:
                printf("Invalid option! Please try again\n");
        }
    }

    return 0;
}
```

Status : Partially correct ***Marks : 0.5/1***

2. Problem Statement

Emily is developing a program to manage a singly linked list. The program should allow users to perform various operations on the linked list, such as inserting elements at the beginning or end, deleting elements from the beginning or end, inserting before or after a specific value, and deleting elements before or after a specific value. After each operation, the updated linked list should be displayed.

Your task is to help Emily in implementing the same.

Input Format

The first line contains an integer choice, representing the operation to perform:

- For choice 1 to create the linked list. The next lines contain space-separated integers, with -1 indicating the end of input.
- For choice 2 to display the linked list.
- For choice 3 to insert a node at the beginning. The next line contains an integer data representing the value to insert.
- For choice 4 to insert a node at the end. The next line contains an integer data representing the value to insert.
- For choice 5 to insert a node before a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 6 to insert a node after a specific value. The next line contains two integers: value (existing node value) and data (value to insert).
- For choice 7 to delete a node from the beginning.
- For choice 8 to delete a node from the end.

- For choice 9 to delete a node before a specific value. The next line contains an integer value representing the node before which deletion occurs.
- For choice 10 to delete a node after a specific value. The next line contains an integer value representing the node after which deletion occurs.
- For choice 11 to exit the program.

Output Format

For choice 1, print "LINKED LIST CREATED".

For choice 2, print the linked list as space-separated integers on a single line. If the list is empty, print "The list is empty".

For choice 3, 4, 5, and 6, print the updated linked list with a message indicating the insertion operation.

For choice 7, 8, 9, and 10, print the updated linked list with a message indicating the deletion operation.

For any operation that is not possible print an appropriate error message such as "Value not found in the list".

For choice 11 terminate the program.

For any invalid option, print "Invalid option! Please try again".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
5
3
7
-1
2
11

Output: LINKED LIST CREATED
5 3 7

Answer

```
// You are using GCC

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

struct Node {
   int data;
   struct Node* next;
};

struct Node* head = NULL;

void insertAtBeginning(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = head;
   head = newNode;
}

void insertAtEnd(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
      return;
   }

   struct Node* temp = head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
}

void insertBefore(int value, int data) {
   if (head == NULL) {
      printf("Value not found in the list\n");
      return;
   }
```

```
    if (head->data == value) {
        insertAtBeginning(data);
        return;
    }

    struct Node* temp = head;
    struct Node* prev = NULL;

    while (temp != NULL && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = temp;
    prev->next = newNode;
}

void insertAfter(int value, int data) {
    struct Node* temp = head;

    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = temp->next;
    temp->next = newNode;
}
```

```
void deleteFromBeginning() {
    if (head == NULL) {
        return;
    }

    struct Node* temp = head;
    head = head->next;
    free(temp);
}

void deleteFromEnd() {
    if (head == NULL) {
        return;
    }

    if (head->next == NULL) {
        free(head);
        head = NULL;
        return;
    }

    struct Node* temp = head;
    struct Node* prev = NULL;

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

    prev->next = NULL;
    free(temp);
}

void deleteBefore(int value) {
    if (head == NULL || head->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    if (head->next->data == value) {
        deleteFromBeginning();
        return;
    }

    struct Node* temp = head;
    struct Node* prev = NULL;
    struct Node* prev2 = NULL;

    while(temp != NULL && temp->next != NULL && temp->next->data != value){
        prev2 = prev;
        prev = temp;
        temp = temp->next;
    }

    if(temp == NULL || temp->next == NULL){
        printf("Value not found in the list\n");
        return;
    }

    if(prev == head){
        head = temp->next;
        free(temp);
    }
    else{
    prev2->next = temp->next;
    free(temp);
    }
}

void deleteAfter(int value) {
    struct Node* temp = head;

    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    struct Node* toDelete = temp->next;
    temp->next = toDelete->next;
    free(toDelete);
```

```
}

void displayList() {
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }

    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, data, value;

    do {
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int num;
                while (1) {
                    scanf("%d", &num);
                    if (num == -1)
                        break;
                    insertAtEnd(num);
                }
                printf("LINKED LIST CREATED\n");
                break;
            }
            case 2:
                displayList();
                break;
            case 3:
                scanf("%d", &data);
                insertAtBeginning(data);
                printf("The linked list after insertion at the beginning is:\n");
                displayList();
                break;
            case 4:
                scanf("%d", &data);
                insertAtEnd(data);
                printf("The linked list after insertion at the end is:\n");
                displayList();
                break;
            case 5:
                scanf("%d %d", &value, &data);
                insertBefore(value, data);
                printf("The linked list after insertion before a value is:\n");
                displayList();
                break;
            case 6:
                scanf("%d %d", &value, &data);
                insertAfter(value, data);
                printf("The linked list after insertion after a value is:\n");
                displayList();
                break;
            case 7:
                deleteFromBeginning();
                printf("The linked list after deletion from the beginning is:\n");
                displayList();
                break;
            case 8:
                deleteFromEnd();
                printf("The linked list after deletion from the end is:\n");
                displayList();
                break;
            case 9:
                scanf("%d", &value);
                deleteBefore(value);
                printf("The linked list after deletion before a value is:\n");
                displayList();
                break;
            case 10:
                scanf("%d", &value);
                deleteAfter(value);
                printf("The linked list after deletion after a value is:\n");
                displayList();
                break;
            case 11:
            break;
          default:
            printf("Invalid option! Please try again\n");
        }
    } while (choice != 11);

    return 0;
}
```

Status : Partially correct ***Marks : 0.5/1***

3. Problem Statement

John is working on evaluating polynomials for his math project. He needs to compute the value of a polynomial at a specific point using a singly linked list representation.

Help John by writing a program that takes a polynomial and a value of x as input, and then outputs the computed value of the polynomial.

Example

Input:

2

13

12

11

1

Output:

36

Explanation:

The degree of the polynomial is 2.

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

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

Calculate the value of x0: 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 the degree of the polynomial.

The second line consists of the coefficient x2.

The third line consists of the coefficient of x1.

The fourth line consists of the coefficient x0.

The fifth line consists of the value of x, at which the polynomial should be evaluated.

Output Format

The output is the 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

```
// You are using GCC
// You are using Java

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

```
int main() {
  int degree, x;
  int coeff[3];

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

  int result = 0;
  for (int i = degree; i >= 0; i--) {
    int termValue = coeff[degree - i];
    for (int j = 0; j < i; j++) {
      termValue *= x;
    }
    result += termValue;
  }

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

  return 0;
}
```

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

4. Problem Statement

Bharath is very good at numbers. As he is piled up with many works, he decides to develop programs for a few concepts to simplify his work. As a first step, he tries to arrange even and odd numbers using a linked list. He stores his values in a singly-linked list.

Now he has to write a program such that all the even numbers appear before the odd numbers. Finally, the list is printed in such a way that all even numbers come before odd numbers. Additionally, the even numbers should be in reverse order, while the odd numbers should maintain their original order.

Example

Input:

6

3 1 0 4 30 12

Output:

12 30 4 0 3 1

Explanation:

Even elements: 0 4 30 12

Reversed Even elements: 12 30 4 0

Odd elements: 3 1

So the final list becomes: 12 30 4 0 3 1

Input Format

The first line consists of an integer n representing the size of the linked list.

The second line consists of n integers representing the elements separated by space.

Output Format

The output prints the rearranged list separated by a space.

The list is printed in such a way that all even numbers come before odd numbers and the even numbers should be in reverse order, while the odd numbers should maintain their original order.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 6
3 1 0 4 30 12
Output: 12 30 4 0 3 1

Answer

```
// You are using GCC

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node** headRef, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *headRef;
    *headRef = newNode;
}

int main() {
    int n, data;
    scanf("%d", &n);

    struct Node* head = NULL;
    struct Node* evenHead = NULL;
    struct Node* oddHead = NULL;
    struct Node* oddTail = NULL;

    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        if (data % 2 == 0) {
            insertAtBeginning(&evenHead, data);
        } else {
            struct Node* newNode = createNode(data);
            if (oddHead == NULL) {
                oddHead = newNode;
                oddTail = newNode;
            } else {
                oddTail->next = newNode;
                oddTail = newNode;
            }
        }
    }

    struct Node* current = evenHead;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }

    current = oddHead;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }

    printf("\n");


    struct Node* temp;
    current = evenHead;
    while(current != NULL){
        temp = current;
        current = current->next;
        free(temp);
    }
    current = oddHead;
    while(current != NULL){
        temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

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

5. Problem Statement

Imagine you are managing the backend of an e-commerce platform. Customers place orders at different times, and the orders are stored in two separate linked lists. The first list holds the orders from morning, and the second list holds the orders from the evening.

Your task is to merge the two lists so that the final list holds all orders in sequence from the morning list followed by the evening orders, in the same order

Input Format

The first line contains an integer n , representing the number of orders in the morning list.

The second line contains n space-separated integers representing the morning orders.

The third line contains an integer m , representing the number of orders in the evening list.

The fourth line contains m space-separated integers representing the evening orders.

Output Format

The output should be a single line containing space-separated integers representing the merged order list, with morning orders followed by evening orders.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
101 102 103
2
104 105
Output: 101 102 103 104 105

Answer

```
// You are using GCC

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

struct Node* insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        return newNode;
    }
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    return head;
}

struct Node* mergeLists(struct Node* morningList, struct Node* eveningList) {
    if (morningList == NULL) {
        return eveningList;
    }
    if (eveningList == NULL) {
        return morningList;
    }

    struct Node* temp = morningList;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = eveningList;
    return morningList;
}

void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int n, m, order;
    struct Node* morningList = NULL;
    struct Node* eveningList = NULL;

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &order);
        morningList = insertAtEnd(morningList, order);
    }

    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d", &order);
        eveningList = insertAtEnd(eveningList, order);
    }

    struct Node* mergedList = mergeLists(morningList, eveningList);
    printList(mergedList);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Rahulan R
Email: 240701412@rajalakshmi.edu.in
Roll no: 240701412
Phone: 9444114186
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_week 1_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 21.5

Section 1 : Coding

1. Problem Statement

Hayley loves studying polynomials, and she wants to write a program to compare two polynomials represented as linked lists and display whether they are equal or not.

The polynomials are expressed as a series of terms, where each term consists of a coefficient and an exponent. The program should read the polynomials from the user, compare them, and then display whether they are equal or not.

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 representing 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 representing the coefficient and the exponent of the term in the second polynomial.

Output Format

The first line of output prints "Polynomial 1: " followed by the first polynomial.

The second line prints "Polynomial 2: " followed by the second polynomial.

The polynomials should be displayed in the format ax^b, where a is the coefficient and b is the exponent.

If the two polynomials are equal, the third line prints "Polynomials are Equal."

If the two polynomials are not equal, the third line prints "Polynomials are Not Equal."

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 2
1 2
2 1
2
1 2
2 1
Output: Polynomial 1: (1x^2) + (2x^1)
Polynomial 2: (1x^2) + (2x^1)
Polynomials are Equal.

Answer

```
// You are using GCC

#include <stdio.h>
```

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

typedef struct Term {
   int coefficient;
   int exponent;
   struct Term* next;
} Term;

Term* createTerm(int coefficient, int exponent) {
   Term* newTerm = (Term*)malloc(sizeof(Term));
   newTerm->coefficient = coefficient;
   newTerm->exponent = exponent;
   newTerm->next = NULL;
   return newTerm;
}

Term* insertTerm(Term* head, int coefficient, int exponent) {
   Term* newTerm = createTerm(coefficient, exponent);
   if (head == NULL) {
      return newTerm;
   }
   Term* current = head;
   while (current->next != NULL) {
      current = current->next;
   }
   current->next = newTerm;
   return head;
}

void displayPolynomial(Term* head) {
   Term* current = head;
   while (current != NULL) {
      printf("(%dx^%d)", current->coefficient, current->exponent);
      if (current->next != NULL) {
         printf(" + ");
      }
      current = current->next;
   }
   printf("\n");
}
```

```
bool arePolynomialsEqual(Term* poly1, Term* poly2) {
   Term* current1 = poly1;
   Term* current2 = poly2;
   while (current1 != NULL && current2 != NULL) {
      if (current1->coefficient != current2->coefficient || current1->exponent !=
current2->exponent) {
         return false;
      }
      current1 = current1->next;
      current2 = current2->next;
   }
   if (current1 != NULL || current2 != NULL) {
      return false;
   }
   return true;
}

void freePolynomial(Term* head) {
   Term* current = head;
   while (current != NULL) {
      Term* temp = current;
      current = current->next;
      free(temp);
   }
}

int main() {
   int n, m, coefficient, exponent;
   Term* polynomial1 = NULL;
   Term* polynomial2 = NULL;

   scanf("%d", &n);
   for (int i = 0; i < n; i++) {
      scanf("%d %d", &coefficient, &exponent);
      polynomial1 = insertTerm(polynomial1, coefficient, exponent);
   }

   scanf("%d", &m);
   for (int i = 0; i < m; i++) {
      scanf("%d %d", &coefficient, &exponent);
      polynomial2 = insertTerm(polynomial2, coefficient, exponent);
   }
```

```
    printf("Polynomial 1: ");
    displayPolynomial(polynomial1);
    printf("Polynomial 2: ");
    displayPolynomial(polynomial2);

    if (arePolynomialsEqual(polynomial1, polynomial2)) {
        printf("Polynomials are Equal.\n");
    } else {
        printf("Polynomials are Not Equal.\n");
    }

    freePolynomial(polynomial1);
    freePolynomial(polynomial2);

    return 0;
}
```

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

2. Problem Statement

Lisa is studying polynomials in her class. She is learning about the multiplication of polynomials.

To practice her understanding, she wants to write a program that multiplies two polynomials and displays the result. Each polynomial is represented as a linked list, where each node contains the coefficient and exponent of a term.

Example

Input:

4 3

y

3 1

y

1 0

n

2 2

y

3 1

y

2 0

n

Output:

8x^5 + 12x^4 + 14x^3 + 11x^2 + 9x + 2

Explanation

1. Poly1: 4x^3 + 3x + 1
2. Poly2: 2x^2 + 3x + 2

Multiplication Steps:

1. Multiply 4x^3 by Poly2:

   -> 4x^3 * 2x^2 = 8x^5

   -> 4x^3 * 3x = 12x^4

   -> 4x^3 * 2 = 8x^3

2. Multiply 3x by Poly2:

   -> 3x * 2x^2 = 6x^3

   -> 3x * 3x = 9x^2

   -> 3x * 2 = 6x

3. Multiply 1 by Poly2:

   -> 1 * 2x^2 = 2x^2

-> 1 * 3x = 3x

-> 1 * 2 = 2

Combine the results: 8x^5 + 12x^4 + (8x^3 + 6x^3) + (9x^2 + 2x^2) + (6x + 3x) + 2

The combined polynomial is: 8x^5 + 12x^4 + 14x^3 + 11x^2 + 9x + 2

Input Format

The input consists of two sets of polynomial terms.

Each polynomial term is represented by two integers separated by a space:

- The first integer represents the coefficient of the term.
- The second integer represents the exponent of the term.

After entering a polynomial term, the user is prompted to input a character indicating whether to continue adding more terms to the polynomial.

If the user inputs 'y' or 'Y', the program continues to accept more terms.

If the user inputs 'n' or 'N', the program moves on to the next polynomial.

Output Format

The output consists of a single line representing the resulting polynomial after multiplying the two input polynomials.

Each term of the resulting polynomial is formatted as follows:

- The coefficient and exponent are separated by 'x^' if the exponent is greater than 1.
- If the exponent is 1, only 'x' is displayed without the exponent.
- If the exponent is 0, only the coefficient is displayed.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 4 3
y
3 1
y
1 0
n
2 2
y
3 1
y
2 0
n
Output: 8x^5 + 12x^4 + 14x^3 + 11x^2 + 9x + 2

Answer

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

typedef struct Term {
   int coeff;
   int exp;
} Term;

Term* readPolynomial(int* size) {
   Term* poly = (Term*)malloc(10 * sizeof(Term));
   *size = 0;
   char choice;

   do {
      scanf("%d %d", &poly[*size].coeff, &poly[*size].exp);
      (*size)++;
      scanf(" %c", &choice);
   } while (choice == 'y' || choice == 'Y');
```

```
    return poly;
}

Term* multiplyPolynomials(Term* poly1, int size1, Term* poly2, int size2, int*
resultSize) {
    Term* result = (Term*)malloc(100 * sizeof(Term)); // Allocate enough space
    *resultSize = 0;

    for (int i = 0; i < size1; i++) {
        for (int j = 0; j < size2; j++) {
            int coeff = poly1[i].coeff * poly2[j].coeff;
            int exp = poly1[i].exp + poly2[j].exp;

            int found = 0;
            for (int k = 0; k < *resultSize; k++) {
                if (result[k].exp == exp) {
                    result[k].coeff += coeff;
                    found = 1;
                    break;
                }
            }

            if (!found) {
                result[*resultSize].coeff = coeff;
                result[*resultSize].exp = exp;
                (*resultSize)++;
            }
        }
    }

    // Sort the result in descending order of exponents
    for (int i = 0; i < *resultSize - 1; i++) {
        for (int j = 0; j < *resultSize - i - 1; j++) {
            if (result[j].exp < result[j + 1].exp) {
                Term temp = result[j];
                result[j] = result[j + 1];
                result[j + 1] = temp;
            }
        }
    }

    // Remove zero terms
    int newSize = 0;
    Term* tempResult = (Term*)malloc(100 * sizeof(Term));

    for (int i = 0; i < *resultSize; i++) {
        if (result[i].coeff != 0) {
            tempResult[newSize].coeff = result[i].coeff;
            tempResult[newSize].exp = result[i].exp;
            newSize++;
        }
    }

    *resultSize = newSize;
    free(result);
    return tempResult;
}

void printPolynomial(Term* poly, int size) {
    for (int i = 0; i < size; i++) {
        if (poly[i].coeff == 0) continue;
        if (i > 0 && poly[i].coeff > 0) {
            printf(" + ");
        } else if (i > 0 && poly[i].coeff < 0) {
            printf(" - ");
            poly[i].coeff = abs(poly[i].coeff);
        } else if (poly[i].coeff < 0) {
             printf("-");
             poly[i].coeff = abs(poly[i].coeff);
        }

        if (poly[i].exp == 0) {
            printf("%d", poly[i].coeff);
        } else if (poly[i].exp == 1) {
            if (poly[i].coeff != 0) {
               printf("%dx", poly[i].coeff);
            } else {
                printf("x");
            }

        } else {
            if (poly[i].coeff != 1) {
                printf("%dx^%d", poly[i].coeff, poly[i].exp);
            } else {
```

```
            printf("x^%d", poly[i].exp);
        }
    }
  }

  printf("\n");
}

int main() {
  Term* poly1, * poly2, * result;
  int size1, size2, resultSize;

  poly1 = readPolynomial(&size1);
  poly2 = readPolynomial(&size2);

  result = multiplyPolynomials(poly1, size1, poly2, size2, &resultSize);

  printPolynomial(result, resultSize);

  free(poly1);
  free(poly2);
  free(result);

  return 0;
}
```

Status : Partially correct   ***Marks : 1.5/10***

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

```
// You are using GCC

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

int main() {
  int degree, coeff2, coeff1, coeff0, x;
  scanf("%d", &degree);
  scanf("%d", &coeff2);
  scanf("%d", &coeff1);
  scanf("%d", &coeff0);
  scanf("%d", &x);

  int result = coeff2 * pow(x, 2) + coeff1 * pow(x, 1) + coeff0 * pow(x, 0);
  printf("%d\n", result);

  return 0;
}
```

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