Rajalakshmi Engineering College

Name: Ravi Sankar
Email: 240701424@rajalakshmi.edu.in
Roll no: 240701424
Phone: 8122932671
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 : 3.9

Section 1 : Coding

1. 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

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

```
#include <stdlib.h>

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

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

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

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

Node* rearrangeList(Node* head) {
    Node* evenHead = NULL;
    Node* oddHead = NULL;
    Node* evenTail = NULL;
    Node* oddTail = NULL;
```

```
    Node* current = head;
    while (current != NULL) {
        Node* nextNode = current->next;
        current->next = NULL;
        if (current->data % 2 == 0) {
            if (evenHead == NULL) {
                evenHead = current;
                evenTail = current;
            } else {
                current->next = evenHead;
                evenHead = current;
            }
        } else {
            if (oddHead == NULL) {
                oddHead = current;
                oddTail = current;
            } else {
                oddTail->next = current;
                oddTail = current;
            }
        }
        current = nextNode;
    }

    if (evenHead == NULL) {
        return oddHead;
    }

    evenTail->next = oddHead;
    return evenHead;
}

int main() {
    int n;
    if (scanf("%d", &n) != 1) {
        fprintf(stderr, "Error reading input\n");
        return 1;
    }

    Node* head = NULL;
    for (int i = 0; i < n; i++) {
        int num;
```

```
        if (scanf("%d", &num) != 1) {
            fprintf(stderr, "Error reading input\n");
            return 1;
        }
        append(&head, num);
    }

    head = rearrangeList(head);
    printList(head);

    return 0;
}
```

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

2. 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

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

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

Node* head = NULL;

void createList() {
    int data;
    while (1) {
        scanf("%d", &data);
        if (data == -1) {
            break;
        }
        Node* newNode = (Node*)malloc(sizeof(Node));
        newNode->data = data;
        newNode->next = NULL;
        if (head == NULL) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }
    printf("LINKED LIST CREATED\n");
}

void displayList() {
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
```

```
    }
    printf("\n");
}

void insertAtBeginning(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
    printf("The linked list after insertion at the beginning is:\n");
    displayList();
}

void insertAtEnd(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("The linked list after insertion at the end is:\n");
    displayList();
}

void insertBeforeValue(int value, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    if (head == NULL) {
        printf("Value not found in the list\n");
        return;
    }
    if (head->data == value) {
        newNode->next = head;
        head = newNode;
        printf("The linked list after insertion before a value is:\n");
        displayList();
```

```
        return;
    }
    Node* temp = head;
    while (temp->next != NULL && temp->next->data != value) {
        temp = temp->next;
    }
    if (temp->next == NULL) {
        printf("Value not found in the list\n");
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
    printf("The linked list after insertion before a value is:\n");
    displayList();
}

void insertAfterValue(int value, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    Node* temp = head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Value not found in the list\n");
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
    printf("The linked list after insertion after a value is:\n");
    displayList();
}

void deleteFromBeginning() {
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }
    Node* temp = head;
    head = head->next;
    free(temp);
    printf("The linked list after deletion from the beginning is:\n");
```

```
    displayList();
}

void deleteFromEnd() {
    if (head == NULL) {
        printf("The list is empty\n");
        return;
    }
    if (head->next == NULL) {
        free(head);
        head = NULL;
    } else {
        Node* temp = head;
        while (temp->next->next != NULL) {
            temp = temp->next;
        }
        free(temp->next);
        temp->next = NULL;
    }
    printf("The linked list after deletion from the end is:\n");
    displayList();
}

void deleteBeforeValue(int value) {
    if (head == NULL || head->next == NULL || head->data == value) {
        printf("No node to delete before the specified value\n");
        return;
    }
    if (head->next->data == value) {
        Node* temp = head;
        head = head->next;
        free(temp);
        printf("The linked list after deletion before a value is:\n");
        displayList();
        return;
    }
    Node* temp = head;
    while (temp->next->next != NULL && temp->next->next->data != value) {
        temp = temp->next;
    }
    if (temp->next->next == NULL) {
        printf("Value not found in the list\n");
```

```
        return;
    }
    Node* toDelete = temp->next;
    temp->next = temp->next->next;
    free(toDelete);
    printf("The linked list after deletion before a value is:\n");
    displayList();
}

void deleteAfterValue(int value) {
    Node* temp = head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("No node to delete after the specified value\n");
        return;
    }
    Node* toDelete = temp->next;
    temp->next = temp->next->next;
    free(toDelete);
    printf("The linked list after deletion after a value is:\n");
    displayList();
}

int main() {
    int choice, data, value;
    while (1) {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                createList();
                break;
            case 2:
                displayList();
                break;
            case 3:
                scanf("%d", &data);
                insertAtBeginning(data);
                break;
            case 4:
                scanf("%d", &data);
```

```
            insertAtEnd(data);
            break;
        case 5:
            scanf("%d %d", &value, &data);
            insertBeforeValue(value, data);
            break;
        case 6:
            scanf("%d %d", &value, &data);
            insertAfterValue(value, data);
            break;
        case 7:
            deleteFromBeginning();
            break;
        case 8:
            deleteFromEnd();
            break;
        case 9:
            scanf("%d", &value);
            deleteBeforeValue(value);
            break;
        case 10:
            scanf("%d", &value);
            deleteAfterValue(value);
            break;
        case 11:
            exit(0);
        default:
            printf("Invalid option! Please try again\n");
    }
  }
  return 0;
}
```

Status : Partially correct **Marks : 0.4/1**

3. 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

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

// Define the node structure
struct Node {
    int data;
    struct Node* next;
};

// Function to insert a node at the beginning
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

// Function to insert a node at the end
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    struct Node* temp = *head;
    newNode->data = data;
    newNode->next = NULL;

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

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

// Function to insert a node before a specific value
void insertBeforeValue(struct Node** head, int value, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;
    newNode->data = data;

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

    // If the value is at the head
    if (temp->data == value) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

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

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

```
    newNode->next = temp->next;
    temp->next = newNode;
}

// Function to insert a node after a specific value
void insertAfterValue(struct Node** head, int value, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;
    newNode->data = data;

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

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

    newNode->next = temp->next;
    temp->next = newNode;
}

// Function to delete a node from the beginning
void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty\n");
        return;
    }

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

// Function to delete a node from the end
void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty\n");
        return;
    }
```

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

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

// Function to delete a node before a specific value
void deleteBeforeValue(struct Node** head, int value) {
    if (*head == NULL || (*head)->next == NULL) {
        printf("The list is empty or too small to delete before a value\n");
        return;
    }

    struct Node* temp = *head;

    // Handle case where the value is at the head (no node before it)
    if (temp->next->data == value) {
        printf("The list is empty or too small to delete before a value\n");
        return;
    }

    while (temp->next != NULL && temp->next->next != NULL) {
        if (temp->next->next->data == value) {
            struct Node* toDelete = temp->next;
            temp->next = temp->next->next;
            free(toDelete);
            return;
        }
        temp = temp->next;
    }

    printf("Value not found in the list\n");
}
```

```
// Function to delete a node after a specific value
void deleteAfterValue(struct Node** head, int value) {
    if (*head == NULL) {
        printf("The list is empty\n");
        return;
    }

    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);
}

// Function to display the linked list
void display(struct Node* head) {
    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");
}

// Main function to handle operations
int main() {
    struct Node* head = NULL;
```

```
int choice, data, value;

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

   switch (choice) {
      case 1:
         head = NULL;
         while (1) {
            scanf("%d", &data);
            if (data == -1) break;
            insertAtEnd(&head, data);
         }
         printf("LINKED LIST CREATED\n");
         break;

      case 2:
         display(head);
         break;

      case 3:
         scanf("%d", &data);
         insertAtBeginning(&head, data);
         printf("The linked list after insertion at the beginning is:\n");
         display(head);
         break;

      case 4:
         scanf("%d", &data);
         insertAtEnd(&head, data);
         printf("The linked list after insertion at the end is:\n");
         display(head);
         break;

      case 5:
         scanf("%d %d", &value, &data);
         insertBeforeValue(&head, value, data);
         printf("The linked list after insertion before a value is:\n");
         display(head);
         break;

      case 6:
```

```
            scanf("%d %d", &value, &data);
            insertAfterValue(&head, value, data);
            printf("The linked list after insertion after a value is:\n");
            display(head);
            break;

        case 7:
            deleteFromBeginning(&head);
            printf("The linked list after deletion from the beginning is:\n");
            display(head);
            break;

        case 8:
            deleteFromEnd(&head);
            printf("The linked list after deletion from the end is:\n");
            display(head);
            break;

        case 9:
            scanf("%d", &value);
            deleteBeforeValue(&head, value);
            printf("The linked list after deletion before a value is:\n");
            display(head);
            break;

        case 10:
            scanf("%d", &value);
            deleteAfterValue(&head, value);
            printf("The linked list after deletion after a value is:\n");
            display(head);
            break;

        case 11:
            return 0;

        default:
            printf("Invalid option! Please try again\n");
    }
  }

  return 0;
}
```

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

4. 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

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

int main() {
    int n, m;
    scanf("%d", &n);
    int *morning_orders = (int *)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        scanf("%d", &morning_orders[i]);
    }
    scanf("%d", &m);
    int *evening_orders = (int *)malloc(m * sizeof(int));
    for (int i = 0; i < m; i++) {
        scanf("%d", &evening_orders[i]);
    }

    int *merged_orders = (int *)malloc((n + m) * sizeof(int));
    for (int i = 0; i < n; i++) {
        merged_orders[i] = morning_orders[i];
    }
    for (int i = 0; i < m; i++) {
        merged_orders[n + i] = evening_orders[i];
    }

    for (int i = 0; i < n + m; i++) {
        printf("%d ", merged_orders[i]);
    }
    printf("\n");

    free(morning_orders);
    free(evening_orders);
    free(merged_orders);

    return 0;
}
```

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

5. 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

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

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

    int coefficients[degree + 1];
    for (int i = degree; i >= 0; i--) {
        scanf("%d", &coefficients[i]);
    }

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

```
    int result = 0;
    for (int i = degree; i >= 0; i--) {
        result += coefficients[i] * (int)pow(x, i);
    }

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

    return 0;
}
```

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