Rajalakshmi Engineering College

Name: Priyan S
Email: 240701402@rajalakshmi.edu.in
Roll no: 240701402
Phone: 9150170939
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 : 4.5

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

Node* append(Node* head, int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
   newNode->next = NULL;

   if (!head)
      return newNode;

   Node* temp = head;
   while (temp->next)
      temp = temp->next;

   temp->next = newNode;
   return head;
}

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

   while (head) {
      if (head->data % 2 == 0)
         evenHead = insertAtBeginning(evenHead, head->data);
      else
         oddHead = append(oddHead, head->data);

      head = head->next;
   }
```

```
    if (!evenHead)
        return oddHead;

    Node* temp = evenHead;
    while (temp->next)
        temp = temp->next;

    temp->next = oddHead;
    return evenHead;
}

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

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

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

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

    return 0;
}
```

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

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

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

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

struct Node* head = NULL;

void create() {
   int value;
   while (1) {
      scanf("%d", &value);
      if (value == -1) break;
      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
      newNode->data = value;
```

```
        newNode->next = NULL;

        if (head == NULL)
            head = newNode;
        else {
            struct Node* temp = head;
            while (temp->next != NULL)
                temp = temp->next;
            temp->next = newNode;
        }
    }
    printf("LINKED LIST CREATED\n");
}

void display() {
    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");
}

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

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

    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
    printf("The linked list after insertion at the end is:\n");
    display();
}

void insert_before(int value, int data) {
    if (head == NULL) {
        printf("Value not found in the list\n");
        printf("The linked list after insertion before a value is:\n"); // Fix: Ensure
correct message format
        display();
        return;
    }
    if (head->data == value) {
        insert_begin(data);
        return;
    }
    struct 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");
        printf("The linked list after insertion before a value is:\n"); // Fix: Print correct
message after failure
        display();
        return;
    }
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = temp->next;
    temp->next = newNode;
    printf("The linked list after insertion before a value is:\n");
    display();
}
```

```
void insert_after(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");
      printf("The linked list after insertion after a value is:\n");
      display();
      return;
   }
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = temp->next;
   temp->next = newNode;
   printf("The linked list after insertion after a value is:\n");
   display();
}

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

void delete_end() {
   if (head == NULL) {
      printf("The list is empty\n");
      return;
   }
   if (head->next == NULL) {
      free(head);
      head = NULL;
      printf("The linked list after deletion from the end is:\n");
      display();
      return;
   }
```

```
    struct 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");
    display();
}

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

    if (head->next && head->next->data == value) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
        printf("The linked list after deletion before a value is:\n");
        display();
        return;
    }

    struct Node* temp = head;
    while (temp->next && temp->next->next && temp->next->next->data != value)
        temp = temp->next;

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

    struct Node* delNode = temp->next;
    temp->next = delNode->next;
    free(delNode);
    printf("The linked list after deletion before a value is:\n");
    display();
}


void delete_after(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* delNode = temp->next;
    temp->next = delNode->next;
    printf("The linked list after deletion after a value is:\n");
    display();
}

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

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

        switch (choice) {
            case 1:
                create();
                break;
            case 2:
                display();
                break;
            case 3:
                scanf("%d", &data);
                insert_begin(data);
                break;
            case 4:
                scanf("%d", &data);
                insert_end(data);
                break;
            case 5:
                scanf("%d %d", &value, &data);
                insert_before(value, data);
                break;
            case 6:
                scanf("%d %d", &value, &data);
                insert_after(value, data);
                break;
            case 7:
                delete_begin();
                break;
            case 8:
                delete_end();
                break;
            case 9:
                scanf("%d", &value);
                delete_before(value);
                break;
            case 10:
                scanf("%d", &value);
                delete_after(value);
                break;
            case 11:
                return 0;
            default:
                printf("Invalid option! Please try again\n");
        }
    }
}
```

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

3. 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;
};

// Function to insert elements into a linked list
void insert(struct Node** head, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;

   if (*head == NULL) {
      *head = newNode;
```

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

// Function to merge two linked lists
struct Node* mergeLists(struct Node* head1, struct Node* head2) {
    if (!head1) return head2;
    if (!head2) return head1;

    struct Node* mergedHead = head1;
    struct Node* temp = head1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = head2;

    return mergedHead;
}

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

int main() {
    int n, m, data;
    struct Node* morningHead = NULL;
    struct Node* eveningHead = NULL;

    // Read morning list size
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        insert(&morningHead, data);
    }

    // Read evening list size
    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d", &data);
        insert(&eveningHead, data);
    }

    // Merge lists and display the result
    struct Node* mergedHead = mergeLists(morningHead, eveningHead);
    display(mergedHead);

    return 0;
}
```

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

4. 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
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

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

// Function to insert a node into the linked list
void insert(struct Node** head, int coefficient, int exponent) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;

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

// Function to evaluate the polynomial at a given value of x
int evaluate(struct Node* head, int x) {
    int result = 0;
    struct Node* temp = head;

    while (temp != NULL) {
        result += temp->coefficient * pow(x, temp->exponent);
        temp = temp->next;
    }

    return result;
}
```

```
int main() {
    int degree, coefficient, x;
    struct Node* head = NULL;

    // Read degree of the polynomial
    scanf("%d", &degree);

    // Read coefficients and insert into the linked list
    for (int i = degree; i >= 0; i--) {
        scanf("%d", &coefficient);
        insert(&head, coefficient, i);
    }

    // Read value of x
    scanf("%d", &x);

    // Evaluate the polynomial and print the result
    printf("%d\n", evaluate(head, x));

    return 0;
}
```

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

5. 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>

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

Node *head = NULL;

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

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

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

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

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

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

void insertBefore(int target, int val) {
    int inserted = 0;
    if (head == NULL) {
        printf("Value not found in the list\n");
            printf("The linked list after insertion before a value is: ");
            displayList();
    } else if (head->data == target) {
        insertAtBeginning(val);
        return;
    } else {
        Node *temp = head;
        while (temp->next && temp->next->data != target)
            temp = temp->next;

        if (!temp->next) {
            printf("Value not found in the list\n");
        } else {
            Node *newNode = (Node *)malloc(sizeof(Node));
            newNode->data = val;
            newNode->next = temp->next;
            temp->next = newNode;
            inserted = 1;
        }
    }

    printf("The linked list after insertion before a value is: ");
    displayList();
}


void insertAfter(int target, int val) {
    Node *temp = head;
    int inserted = 0;
    while (temp && temp->data != target)
        temp = temp->next;

    if (!temp) {
        printf("Value not found in the list\n");
    } else {
        Node *newNode = (Node *)malloc(sizeof(Node));
        newNode->data = val;
        newNode->next = temp->next;
        temp->next = newNode;
        inserted = 1;
    }

    printf("The linked list after insertion after a value is: ");
    displayList();
}
```

```
void deleteFromBeginning() {
    if (!head) {
        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) {
        printf("The list is empty\n");
        return;
    }

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

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

    printf("The linked list after deletion from the end is: \n");
    displayList();
}

void deleteBefore(int target) {
    if (!head || !head->next) {
        printf("Value not found in the list\n");
        return;
    }
    if (head->next->data == target) {
        Node *temp = head;
        head = head->next;
        free(temp);
        printf("The linked list after deletion before a value is: \n");
        displayList();
        return;
    }

    Node *prev = NULL, *curr = head, *next = head->next;
    while (next && next->next && next->next->data != target) {
        prev = curr;
        curr = next;
        next = next->next;
    }

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

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

    printf("The linked list after deletion before a value is: \n");
    displayList();
}

void deleteAfter(int target) {
    Node *temp = head;
    while (temp && temp->data != target)
        temp = temp->next;

    if (!temp || !temp->next) {
        printf("Value not found in the list\n");
        printf("The linked list after deletion after a value is: \n");
        displayList();
        return;
    }

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

    printf("The linked list after deletion after a value is: \n");
    displayList();
}

int main() {
    int choice;

    while (scanf("%d", &choice)) {
        if (choice == 1) {
            createList();
        } else if (choice == 2) {
            displayList();
        } else if (choice == 3) {
            int val;
            scanf("%d", &val);
            insertAtBeginning(val);
        } else if (choice == 4) {
            int val;
            scanf("%d", &val);
            insertAtEnd(val);
        } else if (choice == 5) {
            int value, data;
            scanf("%d %d", &value, &data);
            insertBefore(value, data);
        } else if (choice == 6) {
            int value, data;
            scanf("%d %d", &value, &data);
            insertAfter(value, data);
        } else if (choice == 7) {
            deleteFromBeginning();
        } else if (choice == 8) {
            deleteFromEnd();
        } else if (choice == 9) {
            int value;
            scanf("%d", &value);
            deleteBefore(value);
        } else if (choice == 10) {
            int value;
            scanf("%d", &value);
            deleteAfter(value);
        } else if (choice == 11) {
            break;
        } else {
            printf("Invalid option! Please try again\n");
        }
    }

    return 0;
}
```

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