Rajalakshmi Engineering College

Name: KEERTHI PRIYA T
Email: 240701258@rajalakshmi.edu.in
Roll no: 2116240701258
Phone: 7397397221
Branch: REC
Department: I CSE FC
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 : 1

Section 1 : Coding

1. 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 LinkedList {
    struct Node* head;
};

void createLinkedList(struct LinkedList* list, int values[], int size) {
    struct Node* newNode;
    struct Node* current;

    for (int i = 0; i < size; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = values[i];
        newNode->next = NULL;

        if (list->head == NULL) {
            list->head = newNode;
        } else {
            current = list->head;
            while (current->next) {
                current = current->next;
            }
            current->next = newNode;
```

```
        }
    }
    printf("LINKED LIST CREATED\n");
}

void display(struct LinkedList* list) {
    if (list->head == NULL) {
        printf("The list is empty\n");
        return;
    }

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

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

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

    if (list->head == NULL) {
        list->head = newNode;
    } else {
        struct Node* current = list->head;
        while (current->next) {
            current = current->next;
        }
        current->next = newNode;
    }
    display(list);
```

```
}

void insertBeforeValue(struct LinkedList* list, int target, int data) {
    if (list->head == NULL) {
        printf("Value not found in the list\n");
        return;
    }

    if (list->head->data == target) {
        insertAtBeginning(list, data);
        return;
    }

    struct Node* current = list->head;
    while (current->next && current->next->data != target) {
        current = current->next;
    }

    if (current->next) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = current->next;
        current->next = newNode;
        display(list);
    } else {
        printf("Value not found in the list\n");
    }
}

void insertAfterValue(struct LinkedList* list, int target, int data) {
    struct Node* current = list->head;
    while (current && current->data != target) {
        current = current->next;
    }

    if (current) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = current->next;
        current->next = newNode;
        display(list);
    } else {
```

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

void deleteFromBeginning(struct LinkedList* list) {
    if (list->head == NULL) {
        printf("The list is empty\n");
        return;
    }

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

void deleteFromEnd(struct LinkedList* list) {
    if (list->head == NULL) {
        printf("The list is empty\n");
        return;
    }

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

    struct Node* current = list->head;
    while (current->next && current->next->next) {
        current = current->next;
    }

    free(current->next);
    current->next = NULL;
    display(list);
}

void deleteBeforeValue(struct LinkedList* list, int target) {
    if (list->head == NULL || list->head->data == target) {
        printf("Value not found in the list\n");
```

```
        return;
    }

    if (list->head->next && list->head->next->data == target) {
        deleteFromBeginning(list);
        return;
    }

    struct Node* current = list->head;
    while (current->next && current->next->next && current->next->next->data !=
target) {
        current = current->next;
    }

    if (current->next && current->next->next) {
        struct Node* temp = current->next;
        current->next = current->next->next;
        free(temp);
        display(list);
    } else {
        printf("Value not found in the list\n");
    }
}

void deleteAfterValue(struct LinkedList* list, int target) {
    struct Node* current = list->head;
    while (current && current->data != target) {
        current = current->next;
    }

    if (current && current->next) {
        struct Node* temp = current->next;
        current->next = current->next->next;
        free(temp);
        display(list);
    } else {
        printf("Value not found in the list\n");
    }
}

int main() {
    struct LinkedList list = { NULL };
```

```
int choice, data, target;

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

   if (choice == 1) {
      int values[100], size = 0;
      while (1) {
         scanf("%d", &values[size]);
         if (values[size] == -1) break;
         size++;
      }
      createLinkedList(&list, values, size);
   } else if (choice == 2) {
      display(&list);
   } else if (choice == 3) {
      printf("The linked list after insertion at the beginning is : \n");
      scanf("%d", &data);
      insertAtBeginning(&list, data);
   } else if (choice == 4) {
      printf("The linked list after insertion at the end is : \n");
      scanf("%d", &data);
      insertAtEnd(&list, data);
   } else if (choice == 5) {
      printf("Enter the value before which to insert and the data to insert: ");
      scanf("%d %d", &target, &data);
      insertBeforeValue(&list, target, data);
   } else if (choice == 6) {
      printf("Enter the value after which to insert and the data to insert: ");
      scanf("%d %d", &target, &data);
      insertAfterValue(&list, target, data);
   } else if (choice == 7) {
      printf("The linked list after deletion from the beginning is: \n");

      deleteFromBeginning(&list);
   } else if (choice == 8) {
      printf("The linked list after deletion from the end is: \n");

      deleteFromEnd(&list);
   } else if (choice == 9) {
      printf("Enter the value before which to delete: ");
      scanf("%d", &target);
```

```
            deleteBeforeValue(&list, target);
        } else if (choice == 10) {
            printf("Enter the value after which to delete: ");
            scanf("%d", &target);
            deleteAfterValue(&list, target);
        } else if (choice == 11) {
            break;
        } else {
            printf("Invalid option! Please try again.\n");
        }
    }

    return 0;
}
```

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

2. 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>

// Define the structure for the polynomial term (node)
struct Node {
    int coefficient;
    int exponent;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int coefficient, int exponent) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = NULL;
    return newNode;
}

// Function to add a term to the polynomial linked list (terms are added in
descending order of exponent)
void addTerm(struct Node** head, int coefficient, int exponent) {
    struct Node* newNode = createNode(coefficient, exponent);
    if (*head == NULL || (*head)->exponent < exponent) {
        newNode->next = *head;
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL && current->next->exponent > exponent) {
            current = current->next;
        }
        newNode->next = current->next;
        current->next = newNode;
    }
}

// Function to evaluate the polynomial at a given value of x
int evaluatePolynomial(struct Node* head, int x) {
    int result = 0;
    struct Node* current = head;
    while (current != NULL) {
```

```
        result += current->coefficient * pow(x, current->exponent); // Compute term
value
        current = current->next;
    }
    return result;
}

// Function to print the polynomial (for debugging)
void printPolynomial(struct Node* head) {
    if (head == NULL) {
        printf("0\n");
        return;
    }
    struct Node* current = head;
    int first = 1;
    while (current != NULL) {
        if (first) {
            printf("%dx^%d", current->coefficient, current->exponent);
            first = 0;
        } else {
            if (current->coefficient >= 0) {
                printf(" + %dx^%d", current->coefficient, current->exponent);
            } else {
                printf(" - %dx^%d", -current->coefficient, current->exponent);
            }
        }
        current = current->next;
    }
    printf("\n");
}

int main() {
    struct Node* polynomial = NULL;

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

    int coefficient, exponent;

    // Read the coefficients and exponents for the polynomial
    for (int i = degree; i >= 0; i--) {
        scanf("%d", &coefficient);
```

```
        addTerm(&polynomial, coefficient, i);
    }

    // Read the value of x at which the polynomial needs to be evaluated
    int x;
    scanf("%d", &x);

    // Evaluate the polynomial at the given value of x
    int result = evaluatePolynomial(polynomial, x);

    // Output the result
    printf("%d\n", result);

    return 0;
}
```

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

3. 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

// You are using GCC

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

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

```
// You are using GCC
#include<stdio.h>
int main()
{
    int n,m;
    scanf("%d",&n);
    int morning[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d", &morning[i]);
    }

    scanf("%d",&m);
    int evening[m];

    for(int i=0;i<n;i++)
    {
        scanf("%d", &evening[i]);

    }
    for(int i=0;i<n;i++)
    {
        printf("%d ", &morning[i]);
        if(i!= n-1 || m>0)
        {
            printf(" ");

        }
    }

    for(int i=0;i<n;i++)
    {
        printf("%d ", &evening[i]);
```

```
        if (i!= m-1)
        {
            printf(" ");
        }

    }
    printf("\n");
    return 0;
}
```

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

5. 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>

// Define a node structure for the linked list
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
```

```
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to append a node at the end of the linked list
void append(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

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

// Function to rearrange the linked list as per the problem's requirements
void rearrangeList(struct Node* head) {
    struct Node *evenHead = NULL, *oddHead = NULL;
    struct Node *evenTail = NULL, *oddTail = NULL;
    struct Node *temp = head;

    // Traverse the list and separate even and odd numbers
    while (temp != NULL) {
        if (temp->data % 2 == 0) {
            // Append to even list
            if (evenHead == NULL) {
                evenHead = evenTail = createNode(temp->data);
            } else {
```

```
            evenTail->next = createNode(temp->data);
            evenTail = evenTail->next;
        }
    } else {
        // Append to odd list
        if (oddHead == NULL) {
            oddHead = oddTail = createNode(temp->data);
        } else {
            oddTail->next = createNode(temp->data);
            oddTail = oddTail->next;
        }
    }
    temp = temp->next;
}

// Reverse the even list
struct Node *prev = NULL, *current = evenHead, *next = NULL;
while (current != NULL) {
    next = current->next;
    current->next = prev;
    prev = current;
    current = next;
}
evenHead = prev; // Now evenHead points to the reversed even list

// Merge the even and odd lists
if (evenHead == NULL) {
    head = oddHead;
} else {
    head = evenHead;
    evenTail = evenHead;
    while (evenTail->next != NULL) {
        evenTail = evenTail->next;
    }
    evenTail->next = oddHead;
}

// Print the rearranged list
printList(head);
}

int main() {
```

```
    int n;
    scanf("%d", &n); // Size of the linked list
    int value;
    struct Node* head = NULL;

    // Input the elements into the linked list
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        append(&head, value);
    }

    // Rearrange and print the list
    rearrangeList(head);

    return 0;
}
```

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