Rajalakshmi Engineering College

Name: Nishal Pandi
Email: 240801224@rajalakshmi.edu.in
Roll no: 240801230
Phone: 6374294588
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

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

Section 1 : Coding

1. 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>

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

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

// Function to insert node at end
void insertEnd(Node** head, int coeff) {
    Node* newNode = createNode(coeff);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
}

// Function to evaluate polynomial
int evaluatePolynomial(Node* head, int degree, int x) {
    int result = 0;
    Node* temp = head;
    while (temp != NULL) {
```

```
        result += temp->coeff * pow(x, degree);
        degree--;
        temp = temp->next;
    }
    return result;
}

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

    Node* head = NULL;

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

    scanf("%d", &x);

    int result = evaluatePolynomial(head, degree, x);
    printf("%d\n", result);

    // Free allocated memory
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

    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

-

Status : Skipped ***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

-

Status : Skipped ***Marks : 0/1***

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>

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

```
void push(Node** head, int data) {
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->data = data;
    new_node->next = *head;
    *head = new_node;
}

void append(Node** head, int data) {
    Node* new_node = (Node*)malloc(sizeof(Node));
    new_node->data = data;
    new_node->next = NULL;
    if (*head == NULL) {
        *head = new_node;
        return;
    }
    Node* temp = *head;
    while (temp->next)
        temp = temp->next;
    temp->next = new_node;
}

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

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

    Node* even = NULL;
    Node* odd = NULL;

    for (int i = 0; i < n; i++) {
        scanf("%d", &x);
        if (x % 2 == 0) {
            push(&even, x); // Reverse order
        } else {
```

```
            append(&odd, x); // Preserve order
        }
    }

    // Merge even and odd lists
    Node* temp = even;
    if (temp == NULL) {
        printList(odd);
    } else {
        while (temp->next)
            temp = temp->next;
        temp->next = odd;
        printList(even);
    }

    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>

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

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

// Function to append a node to the list
void append(Node** head, int order_id) {
```

```
    Node* newNode = createNode(order_id);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
}

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

int main() {
    int n, m, order;
    Node* head = NULL;

    // Read morning orders
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%d", &order);
        append(&head, order);
    }

    // Read evening orders
    scanf("%d", &m);
    for (int i = 0; i < m; ++i) {
        scanf("%d", &order);
        append(&head, order);
    }

    // Print merged list
```

```
    printList(head);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Nishal Pandi
Email: 240801224@rajalakshmi.edu.in
Roll no: 240801230
Phone: 6374294588
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 2_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

Bala is a student learning about the doubly linked list and its functionalities. He came across a problem where he wanted to create a doubly linked list by appending elements to the front of the list.

After populating the list, he wanted to delete the node at the given position from the beginning. Write a suitable code to help Bala.

Input Format

The first line contains an integer N, the number of elements in the doubly linked list.

The second line contains N integers separated by a space, the data values of the nodes in the doubly linked list.

The third line contains an integer X, the position of the node to be deleted from the doubly linked list.

Output Format

The first line of output displays the original elements of the doubly linked list, separated by a space.

The second line prints the updated list after deleting the node at the given position X from the beginning.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 20 30 40 50
2
Output: 50 40 30 20 10
50 30 20 10

Answer

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

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

// Function to insert a node at the front
void insertFront(struct Node** head_ref, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = *head_ref;

    if (*head_ref != NULL)
```

```
        (*head_ref)->prev = newNode;

    *head_ref = newNode;
}

// Function to delete a node at position X
void deleteAtPosition(struct Node** head_ref, int pos) {
    if (*head_ref == NULL || pos <= 0)
        return;

    struct Node* temp = *head_ref;

    // Traverse to the position
    for (int i = 1; temp != NULL && i < pos; i++)
        temp = temp->next;

    if (temp == NULL) return;

    if (temp->prev != NULL)
        temp->prev->next = temp->next;
    else
        *head_ref = temp->next; // if head is being removed

    if (temp->next != NULL)
        temp->next->prev = temp->prev;

    free(temp);
}

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

int main() {
    int N, X, value;
    struct Node* head = NULL;
```

```
    // Read number of elements
    scanf("%d", &N);

    // Insert elements at front
    for (int i = 0; i < N; i++) {
        scanf("%d", &value);
        insertFront(&head, value);
    }

    // Read the position to delete
    scanf("%d", &X);

    // Print original list
    printList(head);

    // Delete node at position X
    deleteAtPosition(&head, X);

    // Print updated list
    printList(head);

    return 0;
}
```

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

2. Problem Statement

Pranav wants to clockwise rotate a doubly linked list by a specified number of positions. He needs your help to implement a program to achieve this. Given a doubly linked list and an integer representing the number of positions to rotate, write a program to rotate the list clockwise.

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 linked list elements.

The third line consists of an integer k, representing the number of places to

rotate the list.

Output Format

The output displays the elements of the doubly linked list after rotating it by k positions.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 4 5
1
Output: 5 1 2 3 4

Answer

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

// Node structure
struct Node {
    int data;
    struct Node* prev;
    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->prev = newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end
void insertEnd(struct Node** head_ref, int data) {
    struct Node* newNode = createNode(data);

    if (*head_ref == NULL) {
```

```
        *head_ref = newNode;
        return;
    }

    struct Node* temp = *head_ref;
    while (temp->next)
        temp = temp->next;

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

// Function to rotate the list clockwise by k
void rotateClockwise(struct Node** head_ref, int k) {
    if (k == 0 || *head_ref == NULL)
        return;

    struct Node* last = *head_ref;
    int count = 1;

    // Find last node and count length
    while (last->next) {
        last = last->next;
        count++;
    }

    // Make the list circular
    last->next = *head_ref;
    (*head_ref)->prev = last;

    // Move (count - k) steps to find new head
    int steps = count - k;
    struct Node* temp = *head_ref;
    while (steps--)
        temp = temp->next;

    // Break the circular link
    *head_ref = temp;
    (*head_ref)->prev->next = NULL;
    (*head_ref)->prev = NULL;
}
```

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

// Main function
int main() {
    int n, k, value;
    struct Node* head = NULL;

    // Input number of elements
    scanf("%d", &n);

    // Input elements and build list
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        insertEnd(&head, value);
    }

    // Input number of positions to rotate
    scanf("%d", &k);

    // Rotate list
    rotateClockwise(&head, k);

    // Output result
    printList(head);

    return 0;
}
```

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

3. Problem Statement

Tom is a software developer working on a project where he has to check if a doubly linked list is a palindrome. He needs to write a program to solve

this problem. Write a program to help Tom check if a given doubly linked list is a palindrome or not.

Input Format

The first line 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 linked list elements.

Output Format

The first line displays the space-separated integers, representing the doubly linked list.

The second line displays one of the following:

1. If the doubly linked list is a palindrome, print "The doubly linked list is a palindrome".
2. If the doubly linked list is not a palindrome, print "The doubly linked list is not a palindrome".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 2 1

Output: 1 2 3 2 1
The doubly linked list is a palindrome

Answer

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


struct Node {
    int data;
    struct Node* prev;
```

```
    struct Node* next;
};

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

void insertEnd(struct Node** head_ref, int data) {
    struct Node* newNode = createNode(data);
    if (*head_ref == NULL) {
        *head_ref = newNode;
        return;
    }

    struct Node* temp = *head_ref;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}

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

int isPalindrome(struct Node* head) {
    if (head == NULL) return 1;

    struct Node* start = head;
```

```
    struct Node* end = head;

    while (end->next) {
        end = end->next;
    }

    while (start != end && start->prev != end) {
        if (start->data != end->data) {
            return 0;
        }
        start = start->next;
        end = end->prev;
    }

    return 1;
}

int main() {
    int N, value;
    struct Node* head = NULL;

    scanf("%d", &N);

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

    printList(head);

    if (isPalindrome(head)) {
        printf("The doubly linked list is a palindrome\n");
    } else {
        printf("The doubly linked list is not a palindrome\n");
    }
```

```
    return 0;
}
```

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

4. Problem Statement

Rohan is a software developer who is working on an application that processes data stored in a Doubly Linked List. He needs to implement a feature that finds and prints the middle element(s) of the list. If the list contains an odd number of elements, the middle element should be printed. If the list contains an even number of elements, the two middle elements should be printed.

Help Rohan by writing a program that reads a list of numbers, prints the list, and then prints the middle element(s) based on the number of elements in the list.

Input Format

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

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

Output Format

The first line prints the elements of the list separated by space. (There is an extra space at the end of this line.)

The second line prints the middle element(s) based on the number of elements.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
20 52 40 16 18

Output: 20 52 40 16 18
40

Answer

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


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


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


void append(struct Node** head_ref, int data) {
    struct Node* newNode = createNode(data);
    if (*head_ref == NULL) {
        *head_ref = newNode;
        return;
    }

    struct Node* temp = *head_ref;
    while (temp->next != NULL)
        temp = temp->next;

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


void printList(struct Node* head) {
    struct Node* temp = head;
```

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


void printMiddle(struct Node* head, int n) {
    struct Node* temp = head;
    int mid = n / 2;
    int count = 0;


    while (count < mid) {
        temp = temp->next;
        count++;
    }

    if (n % 2 == 1) {

        printf("%d\n", temp->data);
    } else {

        printf("%d %d\n", temp->prev->data, temp->data);
    }
}

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


    scanf("%d", &n);


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

```
    printList(head);

    printMiddle(head, n);

    return 0;
}
```

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

5. Problem Statement

Riya is developing a contact management system where recently added contacts should appear first. She decides to use a doubly linked list to store contact IDs in the order they are added. Initially, new contacts are inserted at the front of the list. However, sometimes she needs to insert a new contact at a specific position in the list based on priority.

Help Riya implement this system by performing the following operations:

Insert contact IDs at the front of the list as they are added.Insert a new contact at a given position in the list.

Input Format

The first line of input consists of an integer N, representing the initial size of the linked list.

The second line consists of N space-separated integers, representing the values of the linked list to be inserted at the front.

The third line consists of an integer position, representing the position at which the new value should be inserted (position starts from 1).

The fourth line consists of integer data, representing the new value to be inserted.

Output Format

The first line of output prints the original list after inserting initial elements to the front.

The second line prints the updated linked list after inserting the element at the

specified position.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
10 20 30 40
3
25

Output: 40 30 20 10
40 30 25 20 10

Answer

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


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


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


void insertFront(struct Node** head_ref, int data) {
    struct Node* newNode = createNode(data);
    if (*head_ref == NULL) {
        *head_ref = newNode;
    } else {
        newNode->next = *head_ref;
```

```
        (*head_ref)->prev = newNode;
        *head_ref = newNode;
    }
}


void insertAtPosition(struct Node** head_ref, int position, int data) {
    if (position == 1) {
        insertFront(head_ref, data);
        return;
    }

    struct Node* newNode = createNode(data);
    struct Node* temp = *head_ref;
    int count = 1;


    while (temp != NULL && count < position - 1) {
        temp = temp->next;
        count++;
    }

    if (temp == NULL) {
        printf("Position out of bounds\n");
        return;
    }


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


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, position, data;
    struct Node* head = NULL;


    scanf("%d", &N);


    for (int i = 0; i < N; i++) {
        scanf("%d", &data);
        insertFront(&head, data);
    }


    printList(head);


    scanf("%d", &position);
    scanf("%d", &data);


    insertAtPosition(&head, position, data);


    printList(head);

    return 0;
}
```

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