

```

#include <stdio.h>

int main() {
    int n, i, j;
    float x, sum = 0.0, term;

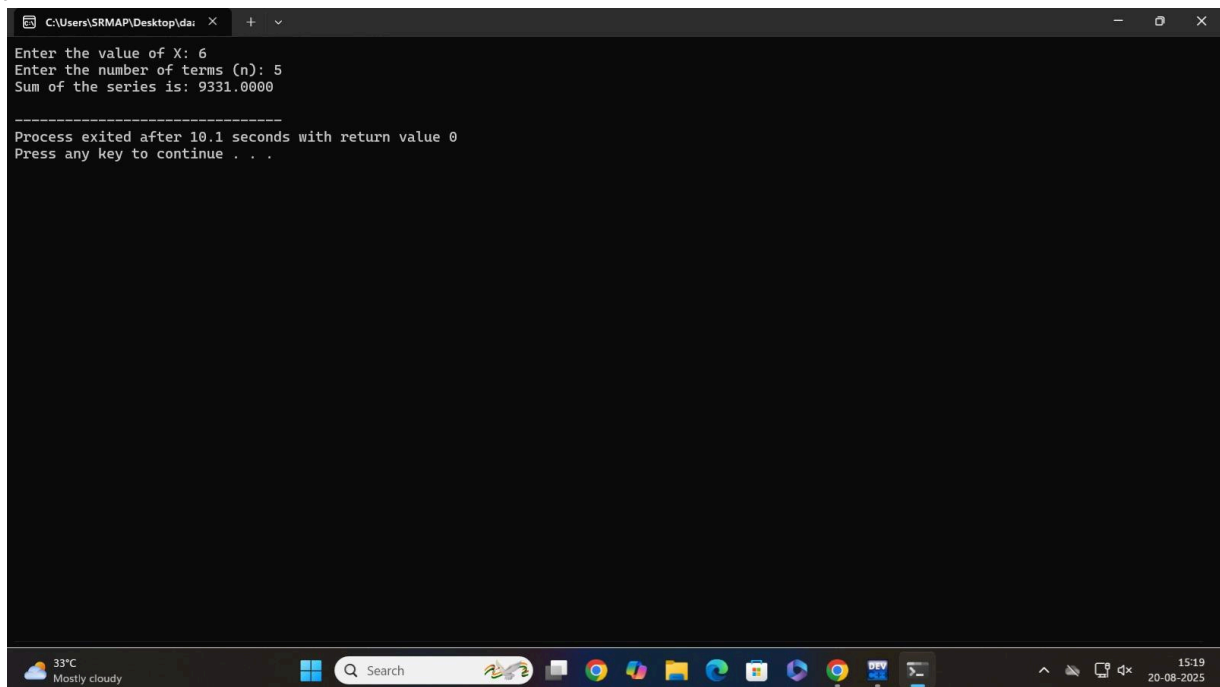
    printf("Enter the value of X: ");
    scanf("%f", &x);
    printf("Enter the number of terms (n): ");
    scanf("%d", &n);

    for (i = 0; i <= n; i++) {
        term = 1.0;
        // Calculate X^i
        for (j = 0; j < i; j++) {
            term *= x;
        }
        sum += term;
    }

    printf("Sum of the series is: %.4f\n", sum);

    return 0;
}

```



```

C:\Users\SRMAP\Desktop\da...
Enter the value of X: 6
Enter the number of terms (n): 5
Sum of the series is: 9331.0000

-----
Process exited after 10.1 seconds with return value 0
Press any key to continue . . .

```

```

#include <stdio.h>

```

```

int main() {
    int n, i;
    float x, sum = 0.0, term = 1.0;

    printf("Enter the value of X: ");
    scanf("%f", &x);
    printf("Enter the number of terms (n): ");
    scanf("%d", &n);

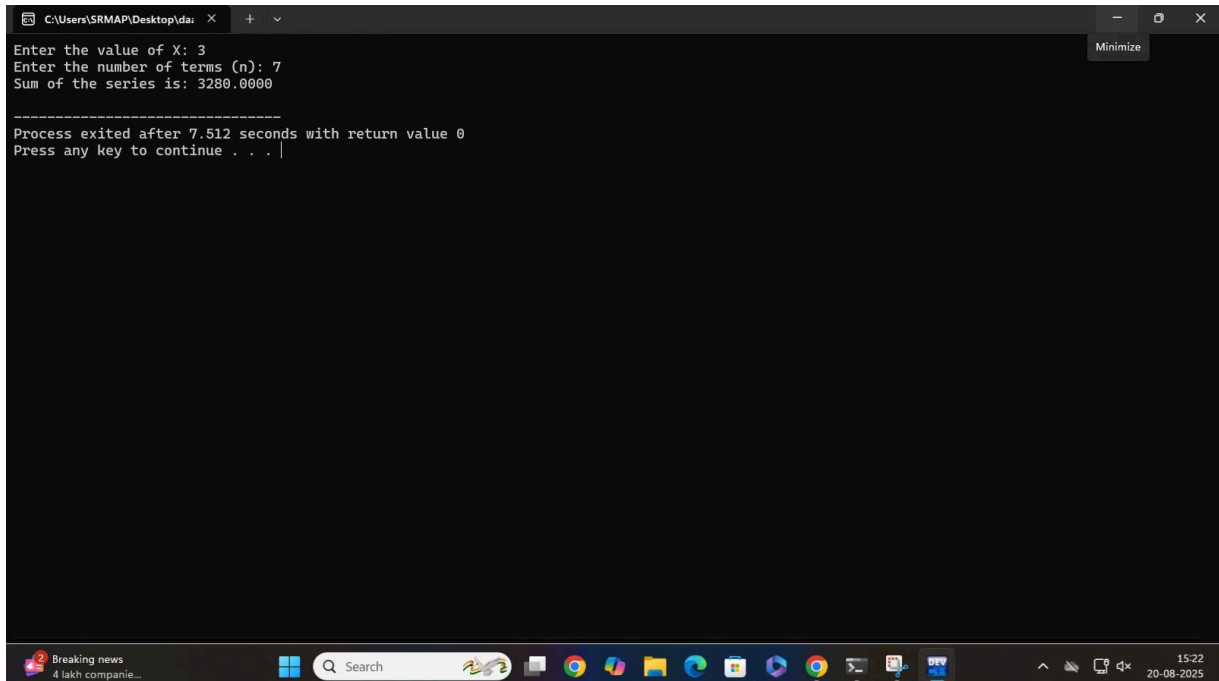
    // First term is 1 (X^0)
    sum = 1.0;

    for (i = 1; i <= n; i++) {
        term *= x; // Calculate the next term by multiplying the previous term by X
        sum += term;
    }

    printf("Sum of the series is: %.4f\n", sum);

    return 0;
}

```



```

C:\Users\SRMAP\Desktop\da...
Enter the value of X: 3
Enter the number of terms (n): 7
Sum of the series is: 3280.0000

-----
Process exited after 7.512 seconds with return value 0
Press any key to continue . . .

```

```

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

struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node* newNode(int data) {
    struct Node* node = (struct Node*) malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) return newNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);
    return root;
}

struct Node* delNode(struct Node* root, int data) {
    if (root == NULL) return NULL;

    if (data < root->data)
        root->left = delNode(root->left, data);
    else if (data > root->data)
        root->right = delNode(root->right, data);
    else {
        // Node with only one child or no child
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        // Node with two children: inorder successor
    }
}

```

```

    struct Node* temp = root->right;
    while (temp->left != NULL) {
        temp = temp->left;
    }
    root->data = temp->data;
    root->right = delNode(root->right, temp->data);
}
return root;
}

```

```

void inorder(struct Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

```

```

int main() {
    struct Node* root = NULL;
    root = insert(root, 29);
    insert(root, 39);
    insert(root, 66);
    insert(root, 98);
    insert(root, 70);
    insert(root, 21);
    insert(root, 82);

    printf("elements before deletion: ");
    inorder(root);

    root = delNode(root, 82);
    root = delNode(root, 39);

    printf("\nelements after deletion: ");
    inorder(root);

    return 0;
}

```

C++

C

C#

Go

HTML

Java

JS

Jupyter

Kotlin

Perl

C++ Online Compiler

↺ ↻ ↷

Share

Run Code

Input

Output

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Node {
5      int data;
6      struct Node *left;
7      struct Node *right;
8  };
9
10 struct Node* newNode(int data) {
11     struct Node* node = (struct Node*) malloc(sizeof(struct Node));
12     node->data = data;
13     node->left = node->right = NULL;
14     return node;
15 }
16
17 struct Node* insert(struct Node* root, int data) {
18     if (root == NULL) return newNode(data);
19     if (data < root->data)
20         root->left = insert(root->left, data);
21     else if (data > root->data)
22         root->right = insert(root->right, data);
23     return root;
24 }
25
26 struct Node* delNode(struct Node* root, int data) {
27     if (root == NULL) return NULL;
```

elements before deletion: 21 29 39 66 70 82 98

elements after deletion: 21 29 66 70 98