Print the linked list in reverse order

Given a pointer to the head of a singly-linked list, print each data value from the reversed list. If the given list is empty, do not print anything.

Function Description

Complete the *reversePrint* function in the editor below.

reversePrint has the following parameters:

- *SinglyLinkedListNode pointer head:* a reference to the head of the list

Prints

The data values of each node in the reversed list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int data;
  struct Node* next;
};

typedef struct Node Node;

// Function to create a new node
Node* create_node(int data) {
  Node* new_node = (Node*)malloc(sizeof(Node));
  new_node->data = data;
  new_node->next = NULL;
  return new_node;
}

// Function to insert a node at the end of the list
```

```
void insert_node(Node** head, int data) {
   Node* new_node = create_node(data);
   if (*head == NULL) {
      *head = new_node;
   } else {
      Node* temp = *head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = new_node;
   }
}
// Function to print the list
void print_list(Node* head) {
   Node* temp = head;
   printf("The linked list is: ");
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
   printf("\n");
}

// Recursive function to print the list in reverse order
//Complete the print_reverse function below
void print_reverse(Node* head) {
*Write your code here
}
```

```
int main() {
    Node* head = NULL;
    int n, data;

    // Reading the number of nodes
    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    // Reading the data for each node and inserting into the list
    for (int i = 0; i < n; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        insert_node(&head, data);
    }
    // Printing the list before reversing
    print_list(head);

    // Printing the list in reverse order
    printf("The linked list in reverse order is: ");
    print_reverse(head);
    printf("\n");

    // Freeing the allocated memory
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
```

```
        free(temp);
    }
    return 0;
}
```