```cpp
#include <iostream>
#include <cstdlib>
#include <cassert>

// Non modificare questa parte sotto del codice
typedef struct Stack {
    int data;
    struct Stack * next;
} Stack;

struct Stack * initStack() {
    return nullptr;
}

bool isEmpty(struct Stack * s) {
    return (s == nullptr);
}

void push(struct Stack * &s, int value) {
    struct Stack * newElement = new Stack;
    newElement->data = value;
    newElement->next = s;
    s = newElement;
}

int top(struct Stack * s) {
    if (isEmpty(s)) {
        std::cerr << "top Error: stack is empty" << std::endl;
        assert(false);
        exit(1);
    }
    return s->data;
}

int pop(struct Stack * &s) {
    if (isEmpty(s)) {
        std::cerr << "pop Error: stack is empty" << std::endl;
        assert(false);
        exit(1);
    }
    int value = s->data;
    struct Stack * temp = s;
    s = s->next;
    delete temp;
    return value;
}

int lenght(struct Stack * s) {
    int count = 0;
    struct Stack * temp = s;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}

void deleteStack(struct Stack * &s) {
    while (!isEmpty(s)) {
        pop(s);
    }
}

void printStack(struct Stack * s, const char * message = "Stack: ") {
    if (isEmpty(s)) {
        std::cout << "Stack is empty" << std::endl;
    }
    else
    {
        std::cout << message;
        struct Stack * temp = s;
        while (temp != nullptr) {
            std::cout << temp->data << " ";
            temp = temp->next;
        }
        std::cout << std::endl;
    }
}
```

```
// Non modificare questa parte sopra del codice

// Inserire qui sotto la dichiarazione della funzione calcola
void calcola(Stack *&s1, Stack * &s2);
// Inserire qui sopra la dichiarazione della funzione calcola


int main() {
    struct Stack *s1, *s2;
    unsigned int seed = (unsigned int)time(NULL);
    // seed = 60000
    seed = 1697033220;
    srand(seed);

    s1 = initStack();
    s2 = initStack();
    for (int i = 0; i < 10; i++) {
        if (i % 2 == 0) push(s1, 10-i);
        else push(s2, 10-i);
    }
    printStack(s1, "S1: ");
    printStack(s2, "S2: ");
    calcola(s1, s2);
    printStack(s1, "NS1: ");
    printStack(s2, "NS2: ");
    deleteStack(s1);
    deleteStack(s2);

    s1 = initStack();
    s2 = initStack();
    for (int i = 0; i < 10; i++) {
        push(s1, rand() % 10);
        push(s2, rand() % 10);
    }
    printStack(s1, "S1: ");
    printStack(s2, "S2: ");
    calcola(s1, s2);
    printStack(s1, "NS1: ");
    printStack(s2, "NS2: ");
    deleteStack(s1);
    deleteStack(s2);

    s1 = initStack();
    s2 = initStack();
    printStack(s1, "S1: ");
    printStack(s2, "S2: ");
    calcola(s1, s2);
    printStack(s1, "NS1: ");
    printStack(s2, "NS2: ");
    deleteStack(s1);
    deleteStack(s2);

    return 0;
}

// Inserire qui sotto la definizione della funzione calcola

int getValue(Stack * & s, int i) {
    if (i == 1) {
        return top(s);
    }
    int value = pop(s);
    int result = getValue(s, i-1);
    push(s, value);
    return result;
}

void calcolaRecur(Stack *&s1, Stack * &s2, int i, int n) {
    if (isEmpty(s1)) {
        return;
    }
    int value1 = pop(s1);
    int vi = getValue(s2, n - i);
    calcolaRecur(s1, s2, i + 1, n);
    push(s1, value1 * vi);
}

void calcola(Stack *&s1, Stack * &s2) {
```

```
    int n1 = lenght(s1);

    calcolaRecur(s1, s2, 0, n1);
}
// Inserire qui sopra la definizione della funzione stackOperator
```