1.Write a C Program to Sort a stack using a temporary stack

We follow this algorithm.

1.Create a temporary stack say tmpStack.

2.While input stack is NOT empty do this:

• Pop an element from input stack call it temp  
• while temporary stack is NOT empty and top of temporary stack is greater than temp,  
pop from temporary stack and push it to the input stack  
• push temp in temporary stack

3.The sorted numbers are in tmpStack

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct stack

{

int data;

struct stack \*next;

};

void initStack(struct stack \*\*s)

{

\*s = NULL;

}

int isEmpty(struct stack \*s)

{

if (s == NULL)

return 1;

return 0;

}

void push(struct stack \*\*s, int x)

{

struct stack \*p = (struct stack \*)malloc(sizeof(\*p));

if (p == NULL)

{

fprintf(stderr, "Memory allocation failed.\n");

return;

}

p->data = x;

p->next = \*s;

\*s = p;

}

int pop(struct stack \*\*s)

{

int x;

struct stack \*temp;

x = (\*s)->data;

temp = \*s;

(\*s) = (\*s)->next;

free(temp);

return x;

}

int top(struct stack \*s)

{

return (s->data);

}

void sortedInsert(struct stack \*\*s, int x)

{

if (isEmpty(\*s) || x > top(\*s))

{

push(s, x);

return;

}

int temp = pop(s);

sortedInsert(s, x);

push(s, temp);

}

void sortStack(struct stack \*\*s)

{

if (!isEmpty(\*s))

{

int x = pop(s);

sortStack(s);

sortedInsert(s, x);

}

}

void printStack(struct stack \*s)

{

while (s)

{

printf("%d ", s->data);

s = s->next;

}

printf("\n");

}

int main(void)

{

struct stack \*top;

initStack(&top);

push(&top, 30);

push(&top, -5);

push(&top, 18);

push(&top, 14);

push(&top, -3);

printf("Stack elements before sorting:\n");

printStack(top);

sortStack(&top);

printf("\n\n");

printf("Stack elements after sorting:\n");

printStack(top);

return 0;

}

**Output:**

