**(6 point) Implement the addLargeNumbers function with the following prototype: void addLargeNumbers(const char \*pNum1, const char \*pNum2); This function should output the result of adding the two numbers passed in as strings. Here is an example call to this function with the expected output: /\* Sample call to addLargeNumbers \*/ addLargeNumbers("592", "3784"); /\* Expected output \*/ 4376**

1. void addLargeNumbers(const char \*pNum1, const char \*pNum2) {
2. Stack operandStack1, operandStack2, resultStack;
3. void \*data;
4. int carry = 0, sum;
5. stack\_init(&operandStack1, free);
6. stack\_init(&operandStack2, free);
7. stack\_init(&resultStack, free);
8. // Push numerals of the first and second numbers onto their respective stacks
9. for (int i = 0; pNum1[i] != '\0'; i++) {
10. int \*val = (int \*)malloc(sizeof(int));  // Added cast here
11. \*val = pNum1[i] - '0';
12. stack\_push(&operandStack1, val);
13. }
14. for (int i = 0; pNum2[i] != '\0'; i++) {
15. int \*val = (int \*)malloc(sizeof(int));  // Added cast here
16. \*val = pNum2[i] - '0';
17. stack\_push(&operandStack2, val);
18. }
19. while (stack\_size(&operandStack1) > 0 || stack\_size(&operandStack2) > 0) {
20. sum = carry;
21. if (stack\_size(&operandStack1) > 0) {
22. stack\_pop(&operandStack1, &data);
23. sum += \*(int \*)data;
24. free(data);
25. }
26. if (stack\_size(&operandStack2) > 0) {
27. stack\_pop(&operandStack2, &data);
28. sum += \*(int \*)data;
29. free(data);
30. }
31. carry = sum / 10;
32. int \*resultVal = (int \*)malloc(sizeof(int));
33. \*resultVal = sum % 10;
34. stack\_push(&resultStack, resultVal);
35. }
36. if (carry != 0) {
37. int \*resultVal = (int \*)malloc(sizeof(int));
38. \*resultVal = carry;
39. stack\_push(&resultStack, resultVal);
40. }
41. // Pop and display the result
42. while (stack\_size(&resultStack) > 0) {
43. stack\_pop(&resultStack, &data);
44. printf("%d", \*(int \*)data);
45. free(data);
46. }
47. printf("\n");
48. stack\_destroy(&operandStack1);
49. stack\_destroy(&operandStack2);
50. stack\_destroy(&resultStack);
51. }

**(3 points) Implement a test program that demonstrates adding at least three pairs of large numbers (numbers larger than can be represented by a long).**

1. int main() {
2. // Sample call
3. addLargeNumbers("592", "3784"); // Expected output: 4376
4. // Test cases for numbers larger than can be represented by a long
5. addLargeNumbers("12345678901234567890", "98765432109876543210");// Expected output: 111111111111111111100
6. addLargeNumbers("91827364581263485128364851", "91827364581263485128364851");
7. addLargeNumbers("1234567890123456789001234567890", "9876543210987654321098706543210");
8. return 0;
9. }

**Source file:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "list.h"

#include "stack.h"

//Prototype

void addLargeNumbers(const char \*pNum1, const char \*pNum2);

//Main program

int main() {

    // Sample call

    addLargeNumbers("592", "3784"); // Expected output: 4376

    // Test cases for numbers larger than can be represented by a long

    addLargeNumbers("12345678901234567890", "98765432109876543210");// Expected output: 111111111111111111100

    addLargeNumbers("91827364581263485128364851", "91827364581263485128364851");

    addLargeNumbers("1234567890123456789001234567890", "9876543210987654321098706543210");

    return 0;

}

void addLargeNumbers(const char \*pNum1, const char \*pNum2) {

    Stack operandStack1, operandStack2, resultStack;

    void \*data;

    int carry = 0, sum;

    stack\_init(&operandStack1, free);

    stack\_init(&operandStack2, free);

    stack\_init(&resultStack, free);

    // Push numerals of the first and second numbers onto their respective stacks

    for (int i = 0; pNum1[i] != '\0'; i++) {

    int \*val = (int \*)malloc(sizeof(int));  // Added cast here

    \*val = pNum1[i] - '0';

    stack\_push(&operandStack1, val);

}

    for (int i = 0; pNum2[i] != '\0'; i++) {

    int \*val = (int \*)malloc(sizeof(int));  // Added cast here

    \*val = pNum2[i] - '0';

    stack\_push(&operandStack2, val);

}

    while (stack\_size(&operandStack1) > 0 || stack\_size(&operandStack2) > 0) {

        sum = carry;

        if (stack\_size(&operandStack1) > 0) {

            stack\_pop(&operandStack1, &data);

            sum += \*(int \*)data;

            free(data);

        }

        if (stack\_size(&operandStack2) > 0) {

            stack\_pop(&operandStack2, &data);

            sum += \*(int \*)data;

            free(data);

        }

        carry = sum / 10;

        int \*resultVal = (int \*)malloc(sizeof(int));

        \*resultVal = sum % 10;

        stack\_push(&resultStack, resultVal);

    }

    if (carry != 0) {

        int \*resultVal = (int \*)malloc(sizeof(int));

        \*resultVal = carry;

        stack\_push(&resultStack, resultVal);

    }

    // Pop and display the result

    while (stack\_size(&resultStack) > 0) {

        stack\_pop(&resultStack, &data);

        printf("%d", \*(int \*)data);

        free(data);

    }

    printf("\n");

    stack\_destroy(&operandStack1);

    stack\_destroy(&operandStack2);

    stack\_destroy(&resultStack);

}

**Output:**

**4376**

**36649002859721140271138110**

**183654729162526970256729702**

**1111111110111111111011111111100**

**Stack.h**

/\*

 \* stack.h

 \*/

#ifndef STACK\_H

#define STACK\_H

#include <stdlib.h>

#include "list.h"

/\*

 \* Implement stacks as linked lists.

 \*/

typedef List Stack;

/\*

 \* Public Interface

 \*/

// Initialize the stack

void stack\_init(Stack \*stack, void (\*destroy)(void \*data));

// Destroy the stack

void stack\_destroy(Stack \*stack);

// Push an element onto the stack

int stack\_push(Stack \*stack, const void \*data);

// Pop an element off the stack

int stack\_pop(Stack \*stack, void \*\*data);

// Get the element at the top of the stack without removing it

void \*stack\_peek(Stack \*stack);

// Get the current size of the stack

int stack\_size(Stack \*stack);

#endif

**Stack.c**

/\*

 \* stack.c

 \*/

#include <stdlib.h>

#include "list.h"

#include "stack.h"

// Initialize the stack

void stack\_init(Stack \*stack, void (\*destroy)(void \*data)) {

    list\_init(stack, destroy);

}

// Destroy the stack

void stack\_destroy(Stack \*stack) {

    list\_destroy(stack);

}

// Push data onto the stack

int stack\_push(Stack \*stack, const void \*data) {

    // Insert the data at the beginning of the list

    return list\_ins\_next(stack, NULL, data);

}

// Pop data off the stack

int stack\_pop(Stack \*stack, void \*\*data) {

    // Remove the first element from the list

    return list\_rem\_next(stack, NULL, data);

}

// Peek at the top of the stack

void \*stack\_peek(Stack \*stack) {

    // Return the data at the beginning of the list if it's not empty

    return (stack->head == NULL ? NULL : stack->head->data);

}

// Return the size of the stack

int stack\_size(Stack \*stack) {

    return list\_size(stack);

}