Stack.c

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include "stack.h"

static int mockUp = 0;
static int *stackTop = &mockUp;

static int default_stack[DEFAULT_STACK_SIZE];
static int *stack = default_stack;
static int stack_size = DEFAULT_STACK_SIZE;

static int pop_counter = 0;
static int push_counter = 0;

int getStackSize()
{
  return stack_size;
}

void setStackSize( int size)
{
	int old_stack_size = getStackSize();
	stack_size = size;
	int * new_stack = calloc(size, sizeof(int));

	if(old_stack_size > size){
		//printf("Number of elements in the old stack is %d. Number of elements in the new stack is %d. Will result
in lose of elements. Operation cancelled.\n", (push_counter-pop_counter), size);
		//exit(0);
		pop_counter = pop_counter + (old_stack_size - size);
	}

	int x = 0;
	while(x < old_stack_size && x < size){
		new_stack[x] = stack[x];
		printf("%d\n", new_stack[x]);
		x++;
	}

	(size != 0 && old_stack_size != 0) ? (stackTop = &new_stack[x-1]) : (stackTop = &mockUp);

	if(stack != default_stack){
		free(stack);
	}

	stack = new_stack;
}

void deleteStack()
{
	stack = default_stack;
	stackTop = &mockUp;
	pop_counter = 0;
	push_counter = 0;
}
```

```
int top()
{
  return *stackTop;
}

int pop( int *val)
{
		*val = 0;
		if (stackTop >= &stack[0] && stackTop <= &stack[getStackSize()]) {
			*val = *stackTop;
			(stackTop-1 < &stack[0]) ? (stackTop = &mockUp) : (stackTop = stackTop - 1);
			return ++pop_counter;
		} else{
			return pop_counter;
		}
}

int push( int val)
{
if(stackTop == &mockUp && getStackSize() != 0){
		stackTop = &stack[0];
		*stackTop = val;
   return ++push_counter;
} else if(stackTop >= &stack[0] && stackTop <= &stack[getStackSize()]){
   if (stackTop+1 <= &stack[getStackSize()]){
		stackTop = stackTop + 1;
		*stackTop = val;
		return ++push_counter;
		}
}
return push_counter;
}
```

Stack.h

```
#define DEFAULT_STACK_SIZE 10

extern void setStackSize( int size);
extern int getStackSize();
extern void deleteStack();
extern int top();
extern int pop(  int* val);
extern int push( int val);
```

Stacktest.c

```
s#include <stdlib.h>
#include <stdio.h>
#include "stack.h"
#include "stack.c"

void printState( )
{
  printf( "Size: %d; Top-element: %d.\n", getStackSize(), top());
}
```

```
void printPush( int val)
{
  int res;
  res = push(val);
  printf( "attempting push( %d)...", val);
  if( res ==0) {
    printf(" failed.\n");
  } else {
    printf(" succeeded.\n");
  }
  printState();
}

void printSetSize( int size)
{
  printf( "executing setStackSize( %d).\n", size);
  setStackSize( size);
  printState();
}

void printPop( )
{
  int val;
  int res;
  res = pop( &val);
  printf( "attempting pop( ...) ...");
  if( res ==0) {
    printf(" failed.\n");
  } else {
    printf(" yields %d.\n", val);
  }
  printState();
}

int main()
{
  int i;

  printState();
  for( i=1; i<15; i++) {
    printPush(i);
  }

  printSetSize( 9);
  printSetSize( 17);

  for( i=0; i<15; i++) {
    printPop();
  }

  printSetSize( 0);
  printPush(42);
  printPop();

  printSetSize( 1);
  printPush(42);
  printPop();
```

```
 return 0;
}
```

Main.c

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

#include "linkedlist.c"

int main()
{
        int i;
        printf("pushing stage\n");
        for( i=1; i<15; i++) {
        push(i);
        printf("%d\n", top());
        }


printf("\n\n");
printf("popping stage\n");
        int x;
        for( i=0; i<19; i++) {
        printf("top - %d\n", top());
        pop(&x);
        printf("popped - %d\n", x);
        }
  return 0;
}
```

LinkedList.c

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

struct node {
   int value;
   struct node * next;
};

static struct node * top_node = NULL;
static int push_counter = 0;
static int pop_counter = 0;

int push(int value)
{
   struct node* tmp = (struct node*) malloc(sizeof(struct node));
   if(tmp == NULL){exit(0);}

   tmp->value = value;

   if(top_node == NULL){
        tmp->next = NULL;
        } else{
          tmp->next = top_node;
```

```
        }

    top_node = tmp;

    return ++push_counter;
}

int pop(int* val)
{
        *val = 0;
        if(top_node == NULL){return pop_counter;}
    struct node* tmp = top_node;
    *val = top_node->value;
    top_node = top_node->next;
    free(tmp);
    return ++pop_counter;
}

int top()
{
        if (top_node == NULL){return 0;}
   return top_node->value;
}
```