```
Alexs-MacBook-Air:26 alex$ cat stack.h
#ifndef _STACK_H_
#define _STACK_H_

#include <stdlib.h>

typedef int data_type;

typedef struct {
    data_type *data;
    size_t size;
    size_t top;
} Stack;

Stack* stack_create(void);
void stack_delete (Stack **stack);
int stack_is_empty(Stack *stack);
void stack_push(Stack *stack, data_type value);
data_type stack_pop(Stack *stack);
void stack_print(Stack *stack);
size_t stack_size(Stack *stack);

#endif
Alexs-MacBook-Air:26 alex$ cat stack.c
#include <stdio.h>
#include "stack.h"

#define INIT_SIZE 10
#define STACK_OVERFLOW  -100
#define STACK_UNDERFLOW -101
#define OUT_OF_MEMORY   -102
#define MULTIPLIER 2

Stack* stack_create(void)
{
    Stack *out = NULL;
    out = malloc(sizeof(Stack));
    if (out == NULL) {
        exit(OUT_OF_MEMORY);
    }
    out->size = INIT_SIZE;
    out->data = malloc(out->size * sizeof(data_type));
    if (out->data == NULL) {
        free(out);
        exit(OUT_OF_MEMORY);
    }
    out->top = 0;
    return out;
}

void stack_delete (Stack **stack) {
    free((*stack)->data);
    free(*stack);
    *stack = NULL;
}

void resize(Stack *stack) {
    stack->size *= MULTIPLIER;
    stack->data = realloc(stack->data, stack->size * sizeof(data_type));
    if (stack->data == NULL) {
        exit(STACK_OVERFLOW);
    }
}

int stack_is_empty(Stack *stack)
{
```

```
return stack->top == 0;
}

void stack_push(Stack *stack, data_type value)
{
     if (stack->top >= stack->size) {
        resize(stack);
    }
    stack->data[stack->top] = value;
    stack->top++;
}

data_type stack_pop(Stack *stack)
{
     if (stack->top == 0) {
        exit(STACK_UNDERFLOW);
    }
    stack->top--;
    return stack->data[stack->top];
}

void stack_print(Stack *stack)
{
     for(int i = 0; i + 1 <= stack->top; i++)
     {
          printf("%d\n", stack->data[i]);
     }
}

size_t stack_size(Stack *stack)
{
     return stack->top;
}
Alexs-MacBook-Air:26 alex$ cat sort.h
#ifndef _SORT_H_
#define _SORT_H_

void sort(Stack *A);

#endif
Alexs-MacBook-Air:26 alex$ cat sort.c
#include "stack.h"
#include "sort.h"

void stack_concatenation(Stack *A, Stack *B)
{
     Stack *T = stack_create();
     while(!stack_is_empty(B)) {
          stack_push(T, stack_pop(B));
     }
     while(!stack_is_empty(T)) {
          stack_push(A, stack_pop(T));
     }
     stack_delete(&T);
}


void sort(Stack *A)
{
     int a;
     int key = stack_pop(A);
     Stack *L = stack_create();
     Stack *G = stack_create();
     while(!stack_is_empty(A)) {
          a = stack_pop(A);
          if ( a < key) {
```

```
            stack_push(L, a);
        } else {
            stack_push(G, a);
        }
    }
    if (stack_size(L) > 1) {
        sort(L);
    }
    if (stack_size(G) > 1) {
        sort(G);
    }
    stack_push(L, key);
    stack_concatenation(L, G);
    stack_concatenation(A, L);
    stack_delete(&L);
    stack_delete(&G);
}
Alexs-MacBook-Air:26 alex$ cat main.c
#include <stdio.h>
#include "stack.h"
#include "sort.h"

int main(void)
{
    int a;
    Stack *A = stack_create();
    while(scanf("%d", &a) == 1) {
        stack_push(A, a);
    }
    puts("-------------------------");
    stack_print(A);
    puts("-------------------------");
    sort(A);
    stack_print(A);
    return 0;
}
Alexs-MacBook-Air:26 alex$ cat makefile
CC = gcc
LD = gcc
CCFLAGS = -Wall -pedantic  -std=c99
LDFLAGS =

main.out: main.o sort.o stack.o
    $(LD) $(LDFLAGS) -o main.out main.o sort.o stack.o
main.o: main.c stack.h sort.h
    $(CC) $(CCFLAGS) -c main.c
sort.o: sort.c sort.h
    $(CC) $(CCFLAGS) -c sort.c
stack.o: stack.c stack.h
    $(CC) $(CCFLAGS) -c stack.c
sort.o: stack.h
Alexs-MacBook-Air:26 alex$ make
gcc -Wall -pedantic  -std=c99 -c main.c
gcc -Wall -pedantic  -std=c99 -c sort.c
gcc -Wall -pedantic  -std=c99 -c stack.c
gcc  -o main.out main.o sort.o stack.o
Alexs-MacBook-Air:26 alex$ touch sort.c
Alexs-MacBook-Air:26 alex$ make
gcc -Wall -pedantic  -std=c99 -c sort.c
gcc  -o main.out main.o sort.o stack.o
Alexs-MacBook-Air:26 alex$ touch main.c
Alexs-MacBook-Air:26 alex$ make
gcc -Wall -pedantic  -std=c99 -c main.c
gcc  -o main.out main.o sort.o stack.o
Alexs-MacBook-Air:26 alex$ touch stack.h
Alexs-MacBook-Air:26 alex$ make
```

```
gcc -Wall -pedantic  -std=c99 -c main.c
gcc -Wall -pedantic  -std=c99 -c sort.c
gcc -Wall -pedantic  -std=c99 -c stack.c
gcc  -o main.out main.o sort.o stack.o
Alexs-MacBook-Air:26 alex$ ./main.out
2 3 88 -8 9 0 84 77 2 -4 -4
-------------------------
2
3
88
-8
9
0
84
77
2
-4
-4
-------------------------
-8
-4
-4
0
2
2
3
9
77
84
88
Alexs-MacBook-Air:26 alex$
```