2413 Assignment 2

Due Date
Before Start of Lab Next Week

For this assignment you will be implementing a Stack ADT with a singly linked list substrate. The header file will be provided to you which specifies the data structures and functions involved. In the future you will create the header file, but for now I want you to see the pattern first. You will need to implement both the main.c file and the stack.c file.

There are several test examples provided along with sample output. Your output should match the sample output exactly in order to pass through the grading script properly. Partial credit will reflect completing part of the necessary functionality.

1 Linked List Constructs

In the header file you will find the following structure and typedef declarations.

```
typedef int element;

struct _node{
  element e;
  struct _node* next;
};

typedef struct _node node_type;
typedef struct _node* node;

struct _linked_list{
  node head;
};

typedef struct _linked_list stack_type;
typedef struct _linked_list* stack;
```

A node in a linked list contains an element and a pointer to the next node in the list. It is not sufficient to simply have a list of nodes. You need some way of keeping track of the first node on the list. Typically this means having a second data structure which keeps track of the meta-data for the list. At minimum we need to know where the list starts (often called the head of the list). Other aspects you might save in meta data is the size of the list and where the last node on the list is located in memory.

All memory occupied by a linked list, including the meta data, should be allocated by malloc. You will also need to free the memory when you are done with it.

Here is a list of the functions which need to be implemented:

```
stack newStack();
void deleteStack(stack);
int isEmpty(stack);
element pop(stack);
void push(stack, element);
element peek(stack);
void display(stack);
```

2 newStack

```
stack newStack();
```

This function, when called, will malloc a chunk of memory large enough to hold the size of the "stack_type". You will need to cast the pointer returned to be a stack pointer. (note that from the typedef "stack" is a pointer to "struct _linked_list".) Once you've carved out the memory, you will need to initialize it with proper values. The head pointer should be NULL, indicating an empty list and an empty stack.

3 deleteStack

```
void deleteStack(stack);
```

For every node still in the stack's list, the node must be freed. Do not lose pointers to nodes. When you are ready to free a node be sure you've saved the node's next pointer so it can be freed.

Once all nodes have been freed, then you will free the stack.

4 isEmpty

```
int isEmpty(stack);
```

Given a stack, return 1 when the stack is empty, and 0 otherwise. You can check if the stack is empty by checking that the head pointer inside of it is NULL or points to a node.

5 pop

```
element pop(stack);
```

pop returns the element at the top of the stack. But before returning the element, the node at the top must be freed and the stack head must point to the next element in the stack. If the stack becomes empty, the head pointer in the stack must be NULL.

This function assumes that the stack is not already empty. The isEmpty function must be called prior to calling this function to make sure its safe to call. Your main will have to check before calling pop.

6 push

```
void push(stack, element);
```

This function creates a new node to contain the element and places it at the top of the stack. The new node's next pointer should point to the old top of the stack, and the stack's head pointer should point to the new node.

7 peek

```
element peek(stack);
```

This function assumes the stack is not empty. It returns the element at the top of the stack without destroying the node or changing the structure of the stack. Be sure your main function tests the isEmpty function before calling this function.

8 display

```
void display(stack);
```

This function assumes the stack is not empty. First "Display" is written on its own line to the output using printf. Then you will walk the linked list and print the elements one per line. So the output will display the stack contents from top to bottom.

9 main

Apart from the display function, your main should perform all output to the console. The commands will be read in from the console as in the prior assignment. You will continually read in commands until end of file is reached.

Main Pseudocode

```
1. initialize new stack
2. while able to read in stack comman
    (a) if command is push, read in an integer, call the push function and
        write appropriate output.
    (b) if command is pop, if stack is empty, print indicating so, otherwise
        call pop and perform proper output.
    (c) if command is peek, if stack is empty, print indicating so, otherwise
        call peek and perform proper output.
    (d) if command is display, if stack is empty, print indicating so, otherwise
        call display function to print proper output.
3. delete the stack.
```

10 Input and Output From Console

Input Commands From Console

```
push <integer>
pop
peek
display
```

Expected Output For push $< integer >$

```
Pushed <integer>
```

Expect Output For pop

```
Stack Is Empty
```

or

```
Popped <integer>
```

Expect Output For peek

```
Stack Is Empty
```

or

```
Peeked <integer>
```

Expect Output For display

```
Stack Is Empty
```

or

```
Display
<integer>
<integer>
<integer>
...
```

Number of integers printed is equivalent to number of integers in stack. Printed from top of stack to bottom.

11 Submission

Submit all and only the necessary .h and .c files, including the stdheader.h file. Put your submission into a ”lastname2.zip” file where lastname is your lastname and zip is any standard archive extension. Do not submit your executable or any of the input and output test cases.

Expected Files

1. main.c
2. stdheader.h
3. stack.h
4. stack.c