Lab 05 Stack Implementation with Applications

Learning Outcomes

After completing the lab students will be able to:

- Implement stacks using linked lists.
- Implement stacks of varying data types using linked lists
- Solve different problems using stacks

Pre-Lab Task:

A stack is a special case of a singly-linked list which works according to LIFO algorithm. Access is restricted to one end, called the top of the stack. Its operations are:

push	− push an element onto the top of the stack.
pop	− pop an element from the top of the stack.
peek	− read the element at the top of the stack without removing it.

Read Chapter 7 from "Data Structures using C", by Reema Thareja, 2nd edition for more information on stacks, their implementation and applications.

Go through the code listings for an implementation of stack using linked lists. Push and pop functions are already implemented. Implement the **peek** function to familiarize yourself with this implementation of stack.

In-lab Task 1: Reversing an array of numbers.

You have to write a function to reverse the order of integers in an array. Call this function from your main function to demonstrate that this functions successfully reverses (in-place) the contents of the input array. You should declare and use a stack within this function. The functions for stack implementation are already given to you. The prototype for the function is given below;

```
void reverse_num_array(int * num_array);
```

In-Lab Task 2: Testing if a mathematical expression is balanced or not.

Write a function that checks whether parentheses in a mathematical expression are balanced or not. The input to the function will be a pointer to a null-terminated array of characters (string). Assume that all the operands in the expression are single digit numbers. You should call this function from 'main' function to demonstrate that it works as intended. The function should return 0 if the expression is balanced, and index of mismatch otherwise. The function prototype is given below:

```
int isBalanced(char * ptr_array);
```

Post-Lab Task: Infix to postfix conversion

Write a function that converts a mathematical expression containing parentheses from infix form to postfix form. The function should take pointers to both source infix array and destination postfix array as arguments. You may assume that the maximum array size is fixed. The infix expression will contain multi digit positive integers only. The function must first check if the brackets in the expression are balanced. The function prototype is given below:

```
void infixToPostfix(char * src, char * dst);
```

```c
/* Program to demonstrate that different data types can be pushed onto the
stack using structures. Each element contains an extra field 'd_type' to tell
which data type is being pushed onto the stack.
*/
#include <stdio.h>
#include <stdlib.h>
#include "node.h"
#include "stack_functions.h"

int main()
{
    struct node * top = NULL;   /// This is the top of the stack
    struct element d1, d2, d3;
    d1.d = 10;
    d1.d_type = 0;
    d2.ch = 'A';
    d2.d_type = 1;
    d3.d = 45;
    d3.d_type = 0;

    push(&top, d1);
    push(&top, d3);
    push(&top, d2);

    for(int i = 0; i<3; i++)
    {
        struct element temp;
        temp = pop(&top);
        if(temp.d_type == 0)
            printf("\nThe data popped is %d", temp.d);
        else
            printf("\nThe data popped is %c", temp.ch);
    }
    return 0;
}
```

CodeListing 1: main.c ***Stack usage example***

```c
#include <stdio.h>
#include <stdlib.h>
#include "node.h"
#include "stack_functions.h"

struct element pop(struct node ** top)
{
    struct element temp = (*top)->data;   /// I copy the data at the top node into a temporary variable
    struct node * ptr_temp = (*top)->next;
    free(*top);
    *top = ptr_temp;
    return(temp);
}
void push(struct node ** top, struct element new_data)
{
    struct node * new_node = (struct node *) malloc(sizeof(struct node));
    new_node->data = new_data;      /// I can assign one struct to another if the type is the same
    new_node->next = * top;
    * top = new_node;
}
```

CodeListing 2: stack_functions.c ***A linked list implementation of stack***

```c
struct element
{
    int d;      /// To store data
    char ch;    /// To store, brackets and operators
    int d_type; /// To tell whether an integer (0) or
                /// a charachter (1) is pushed onto
                /// the stack
};
struct node
{
    struct element data;

    struct node * next;
};
```

CodeListing 3: node.h ***Structure definitions for stack elements***