L15 - Stacks

November 20, 2019

1 Stacks

1.0.1 Administrivia

- graduate attributes at the start of lab 12
- bonus marks
- no class on Dec 4; Dec 6 is on Monday schedule

1.1 Stacks

A *stack* is a linear collection, like a queue, where elements are maintained in the same order as they were added. However, a stack is a **last in first out** (LIFO): the most recently added element is the first one removed.

1.1.1 Fundamental operations

`push()` Add a new value to the top of the stack.

`peek()` Return the value in the item on the top of the stack, without removing it from the stack.

`pop()` Remove the top item from the stack, and return its value.

1.1.2 Additional operations

- determine if the stack is empty
- empty the stack of all items
- destroy the stack
- determine the length of a stack
- compare two stacks
- print the contents of a stack

1.1.3 Unsupported operations

Operation on specific elements (by value) or on specific positions on the stack contradict the LIFO nature of the stack and should be avoided.

1.2 Implementing Stacks

In C we can implement stacks by:

- **Arrays**
    - this is probably gonna be annoying
        - we will have to shift a lot of values around if we `pop()` a value
        - we will have a limit on the max values in the stack
- **Linked Lists**
    - use a singly-linked list
    - `*top` will point to the top of the stack
        - `pop()`, `push()` and `peek()` will operate on the `top` node

```
In [ ]: // i think this is right
        #include <stdlib.h>
        #include <stdio.h>
        #include <assert.h>

        // node structure
        struct node {
            int value;             // list payload
            struct node* next;     // pointer to the somewhere IDK
        };
        typedef struct node node_t;

        struct stack {
            node_t* top;

        }

        void push(node_t* top, int data)
        {
            node_t* oldtop = top;
            top = malloc(sizeof(node_t));
            top->data = data;
            top->next = oldtop;
        }

        int peek(node_t* top)
        {
```

```
    return top->value;
}

int pop( node_t* top )
{
    int k = top->value; // save the value
    node_t* oldtop = top; // a pointer to the old top of the list
    top = top->next;
    free(oldtop);
    return k;
}
```