CptS 122 – Data Structures

Lab 4: Data Structures and Dynamic Linked Stacks in C

Assigned: Week of February 5, 2024
Due: At the end of the lab session

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Design, implement and test a dynamic stack in C
- Compare and contrast dynamic linked lists and dynamic stacks
- Summarize the advantages of applying a stack within certain applications
- Describe the operations applied to a stack including
    1. push ( )
    2. pop ( )
    3. top ( ) or peek ( )
    4. isEmpty ( )

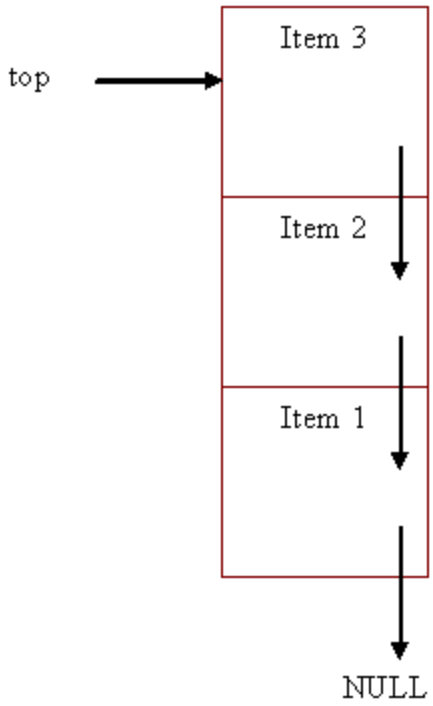
II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Analyze a basic set of requirements for a problem
- Compose a small C language program
- Create test cases for a program
- Apply and implement structs in C
- Apply and implement pointers in C
- Apply and implement dynamic memory in C
- Design and implement a dynamic singly linked list

III. Overview & Requirements:

This lab, along with your TA, will help you navigate through designing, implementing, and testing a dynamic stack. **Recall, a stack data structure is a restricted list, where only the top item in the stack may be accessed at any given time. A stack is referred to as a last-in, first-out (LIFO) structure as a result of this constraint. Furthermore, the operation of a stack must adhere to this restriction. A push ( ) operation adds an item to the top of the stack, a pop ( ) operation removes an item from the top of the stack, and a top ( ) or peek ( ) operation returns the data in the node at the top of the stack.** We will visualize a stack in the following way:



Labs are held in a "closed" environment such that you may ask your TA questions. Please use your TAs knowledge to your advantage. You are required to move at the pace set forth by your TA. Please help other students in need when you are finished with a task. You must work in teams of 2-4 students. However, I encourage you to compose your own solution to each problem. Have a great time! Labs are a vital part to your education in CptS 122 so work diligently.

Tasks:

1. For the following problem define a `stackNode` struct with data of type double. Implement the following operations for your stack data structure:
    1. `isEmpty()` - a predicate function which checks to see if the stack is empty; returns true if the stack is empty; false otherwise
    2. `push()` - inserts a node, with a double precision value, to the top of the stack; the node is allocated dynamically; the double precision value should be passed in as an argument, along with a double pointer to the top of the stack; returns true if the Node is allocated successfully; false otherwise
    3. `pop()` - deletes a node from the top of the stack; accepts a double pointer to the top of the stack; does not return a value; this function should only be called after isEmpty ( ) is called
    4. `top()` or `peek()` - returns the data in the node at the top of the stack; does not modify the stack

2. Test your application. In the same project, create one more header file `testStack.h` and source file `testStack.c` (for a total of at least five files). The `testStack.h` file should contain function prototypes for test functions you will use on your stack functions. The `testStack.c` source file should contain the implementations for these test functions. You will have at least one test function per application function. For example, you will have an application function called `push()` (or a function very similar) that is used to insert a node on the top of the stack. In this task, you will need to create a test function called `testPush()` that passes in various double precision data directly into `push()` to see if it works correctly. Your test should check to see that the top node has the correct value. Does the top pointer change?

3. Tower of Hanoi: A very popular mathematical game or puzzle is referred to as the Tower of Hanoi. The idea behind the game is to find an efficient method for moving disks between *three* posts. Each disk has a different diameter, but all of them can be placed on the available posts. The goal of the game is to move all of the disks from one post to the another according the following rules:
    1. Only one disk may be transferred at a time
    2. Only the top disk on any post may be accessed at a given time
    3. No disk may be placed on top of a smaller disk at any point

    At the start of the game, all of the disks must originally be placed such that the largest disk is on the bottom of the stack of one post, and the smallest is on the top of the stack on the same post. The disks should form a cone shape. Write a program to simulate the Tower of Hanoi game. For each move print the post number (1 – 3) from which the disk is taken, the diameter of the disk, and the resulting post on which the disk is placed. Also, show the current diameter of the disks on each post. You must use stacks to solve this problem! Initially start with three disks in your game. Note: if you visit http://towersofhanoi.info/Animate.aspx, you will find an animation of how Tower of Hanoi should run.

4. Maze: Generate a maze with a start to end path. The maze be represented by a two dimensional array of integers, where a wall may be represented by a 0 and a door may be represented by a 1. Find a path in your maze by using a stack. Modify your `stackNode` to store `Point` data for a path in a maze. A `Point` should be defined as a struct with `row` and `column` fields. Whenever a fork in the maze is encountered, store the coordinates of the fork on the stack. If the current path does not provide a path to the end of the maze, then the last forking point can be popped and a different path may be taken. This is called backtracking.

IV. Submitting Labs:

- You are not required to submit your lab solutions. However, you should keep them in a folder that you may continue to access throughout the semester.

V. Grading Guidelines:

- This lab is worth 10 points. Your lab grade is assigned based on completeness and effort. To receive full credit for the lab you must show up on time, work in a team, continue to work on the problems until the TA has dismissed you, and complete at least 2/3 of the problems.