**Programming Fundamentals III**

**COSC 2436**

**Lab 3: Stacks**

# Lab 17.1 Learning about Stacks and Stack Operations

A **stack** is a very useful data structure in computer science; for example, computers use stacks to implement function calls. A stack is a list of homogeneous elements in which the addition and deletion of elements occur only at one end, called the **top** of the stack.

Elements at the bottom of the stack have been in the stack the longest. The top element of the stack is the last element added to the stack. Because the elements are added and removed from one end (that is, the top), it follows that the item that is added last will be removed first. For this reason, a stack is also called a **Last In First Out (LIFO)** data structure.

Certain kinds of operations can be performed on a stack. Because new items can be added to the stack, we can perform the add operation, called **push**, to add an element to the stack. To retrieve the top element of the stack, perform the **top** operation, and to remove the top element from the stack, perform the **pop** operation. An element can be added to the stack only if there is room and an element can be removed from the stack only if there is something in the stack. These operations are performed with **isFullStack** and **isEmptyStack**, respectively. Depending on the specific implementation, one might also need other operations on a stack.

## Objectives

In this lab, you become acquainted with stacks and how stack operations are implemented.

## After completing this lab, you will be able to:

* Define and implement the stack data structure.
* Identify and define stack operations.
* Represent a stack graphically. Estimated completion time: **15–20 minutes**

# Lab 17.1 Steps: Learning About Stacks and Stack Operations

Match the following stack operations terms with the appropriate definitions.

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | top | a) | Initializes the stack to an empty state. |
| 2. | initializeStack | b) | Removes the top element of the stack. Prior to this operation, the stack must exist and must not be empty. |
| 3. | pop | c) | Returns the top element of the stack. Prior to this operation, the stack must exist and must not be full. |
| 4. | isEmptyStack | d) | Determines whether the stack is full. If the stack is full, it returns the value true; otherwise, it returns the value false. |
| 5. | push | e) | Adds a new element to the top of the stack. The input to this operation consists of the stack and the new element. Prior to this operation, the stack must exist and must not be full. |
| 6. | isFullStack | f) | Determines whether the stack is empty. If the stack is empty, it returns the value true; otherwise, it returns the value false. |

Draw the states of the stack after each operation below, given the following instructions:

Suppose a stack of books (Applied Math, World History, C++ Programming, English, and Chemistry) are lying on a table and need to be stacked neatly on a bookshelf. Initially, all the books are on the table and the stack is empty.

7a. initializeStack:

7b. push the World History book:

7c. push the C++ Programming book:

7d. top and peek at the element in the stack:

7e. push the English book:

7f. pop stack:

7g. push the Chemistry book:

7h. push the Applied Math book:

# Lab 17.2 Implementation of Stacks as Arrays and Linked Lists

In Chapter 8, you learned about one-dimensional and multi-dimensional arrays, how to declare and manipulate data into arrays, became familiar with restrictions on processing arrays, as well as how to search and sort an array. Because all of the elements of a stack are the same type, you can use an array to implement a stack. However, although an array is a random access data structure (that is, you can directly access any element of the array), by definition, a stack is a data structure in which the elements are accessed (pushed or popped) at only one end. This is a critical feature of the stack data structure and must be recognized at the outset of any implementation.

Because an array size is fixed, in the linear array representation of a stack, only a fixed number of elements can be pushed onto the stack. If in a program, the number of elements to be pushed exceeds the size of the array, the program may terminate in an error. To overcome these problems, we can use dynamically organize data using linked lists (such as an ordered list), while dynamically allocating and deallocating memory by using pointer variables.

## Objectives

In this lab, you become acquainted with implementing stack methods backed by an array and a linked list.

## After completing this lab, you will be able to:

* Determine possible cases that may arise while a program is reading input.
* Determine the necessary variables in implementing a stack based on given data input.
* Implement various functions of an array stack.
* Implement various functions of a linked stack. Estimated completion time: **45–60 minutes**

# Lab 17.2 Steps: Implementation of Stacks as Linked Lists

1. Use the linkedStackType class described in this chapter of your text to implement the following functions backed by a linked list. Use comments to indicate the names and types of variables used.

4a. Initialize the stack:

template <class Type>

void linkedStackType<Type>::initializeStack()

{

nodeType<Type> \*tempNode; //pointer to delete the node

} //end initializeStack

4b. Push newStackElement onto the stack:

template <class Type>

void linkedStackType<Type>::push(const Type& newStackElement)

{

} //end push

# Lab 17.3 Fun with Stacks

Stacks have numerous applications in computer science. You can also use stacks to convert recursive algorithms into nonrecursive algorithms, especially recursive algorithms that are not tail recursive.

## Objectives

In this lab, you become acquainted with converting recursive algorithms into nonrecursive algorithms.

## After completing this lab, you will be able to:

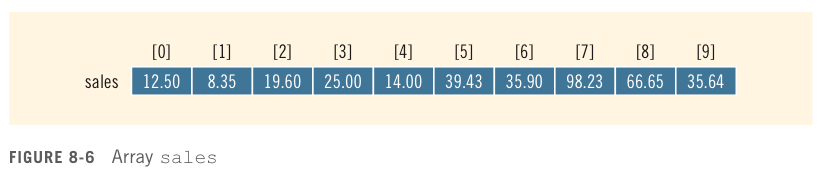
* Design and implement various nonrecursive stack programs. Estimated completion time: **60–90 minutes**

# Lab 17.3 Steps: Fun with Stacks

**Do one of the three tasks described below**

**Submit a copy of your .cpp file in text format**

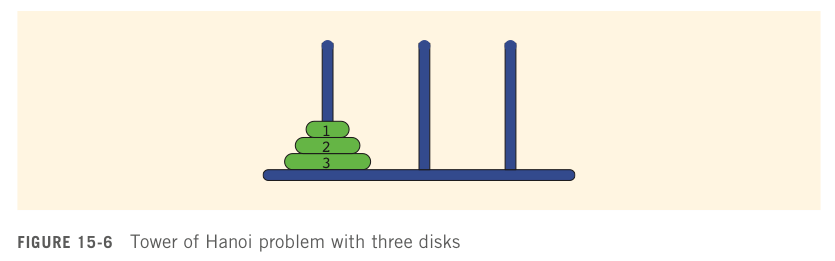
**Task 1**: In this part of the lab, you will write a C++ program to print the top three highest value of an array.



1. Suppose that you read in a data file consisting of the array sales (given in Figure 8-6 above) as input. The program will display the top three highest values of sales. The program will scan the input file only once. (No sorting algorithm can be used). List the possible cases that may arise while the input data is being read by your program.
2. Assume you have a stack variable already declared. Write an algorithm to accomplish the above specifications.
3. Write your program and test it. Demo to your instructor.

**Task 2:** In Chapter 5 of your text, you designed a program to determine the desired Fibonacci number. In Chapter 15, *Fibonacci Number* uses a recursive function, rFibNum, to determine the desired Fibonacci number. A stack is an excellent data structure to use in a program for calculating a Fibonacci number. Write a program that uses a stack to determine the desired Fibonacci number. Save your program as **fibNum.cpp**.

**Task 3:** The Tower of Hanoi problem was also introduced in Chapter 15 of your text. The tower consists of *n* disks and there are three needles.



The objective is to move the disks from needle one to needle three. The rules for moving the disks are as follows:

* 1. Only one disk can be moved at a time.
  2. The removed disk must be placed on one of the needles.
  3. A larger disk cannot be placed on top of a smaller disk.

In this example, we designed a recursive program that printed the sequence of moves needed to transfer the disks from the first needle to the third needle. This program can be converted to a nonrecursive function using stack data structures. Figure 15-6 shows the Tower of Hanoi Problem with three disks. Write a program for the Tower of Hanoi problem with eight disks that uses stacks to solve the puzzle. Save your program as **towerHanoi.cpp**.