UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 8

Stacks

Submitted by:
Palmes, Lewis Clark L.

Instructor:
Engr. Maria Rizette H. Sayo

October 04, 2025

I. Objectives

Introduction

A stack is a collection of objects that are inserted and removed according to the last-in, first-out (LIFO) principle.

A user may insert objects into a stack at any time, but may only access or remove the most recently inserted object that remains (at the so-called “top” of the stack)

This laboratory activity aims to implement the principles and techniques in:

- Writing Python program using Stack
- Writing a Python program that will implement Stack operations

II. Methods

Instruction: Type the python codes below in your Colab. After running your codes, answer the questions below.

```
# Stack implementation in python


# Creating a stack
def create_stack():
    stack = []
    return stack


# Creating an empty stack
def is_empty(stack):
    return len(stack) == 0

# Adding items into the stack
def push(stack, item):
    stack.append(item)
    print("Pushed Element: " + item)

# Removing an element from the stack
def pop(stack):
    if (is_empty(stack)):
        return "The stack is empty"
    return stack.pop()

stack = create_stack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
push(stack, str(5))

print("The elements in the stack are:"+ str(stack))
```

Answer the following questions:

1 Upon typing the codes, what is the name of the abstract data type? How is it implemented?

Answer: The abstract data type is a Stack. It is implemented using a Python list ([]), where append() simulates push and pop()

2 What is the output of the codes?

Output:
Pushed Element: 1
Pushed Element: 2
Pushed Element: 3
Pushed Element: 4
Pushed Element: 5
The elements in the stack are:['1', '2', '3', '4', '5']

3 If you want to type additional codes, what will be the statement to pop 3 elements from the top of the stack?

```
for i in range(3):
    print("Popped:", pop(stack))
```

4 If you will revise the codes, what will be the statement to determine the length of the stack? (Note: You may add additional methods to count the no. of elements in the stack)

```
def size(stack):
        return len(stack)
print("The size of the stack is:", size(stack))
```

III. Results

```python
# Stack implementation in python
# Creating a stack
def create_stack():
    stack = []
    return stack

# Creating an empty stack
def is_empty(stack):
    return len(stack) == 0

# Adding items into the stack
def push(stack, item):
    stack.append(item)
    print("Pushed Element: " + item)

# Removing an element from the stack
def pop(stack):
    if (is_empty(stack)):
        return "The stack is empty"
    return stack.pop()

stack = create_stack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
push(stack, str(5))

print("The elements in the stack are:"+ str(stack))
```

```
Pushed Element: 1
Pushed Element: 2
Pushed Element: 3
Pushed Element: 4
Pushed Element: 5
The elements in the stack are:['1', '2', '3', '4', '5']
```

FIgure 1: Screenshot of output

The program demonstrates the basic implementation of a stack using a Python list. The code pushes five elements—1, 2, 3, 4, and 5—onto the stack, and after each insertion, it prints a confirmation message. Finally, it displays the complete stack, showing that the elements are stored in the order they were pushed. This highlights the fundamental operation of adding items into the stack.

```python
# Stack implementation in python
# Creating a stack
def create_stack():
    stack = []
    return stack

# Creating an empty stack
def is_empty(stack):
    return len(stack) == 0

# Adding items into the stack
def push(stack, item):
    stack.append(item)
    print("Pushed Element: " + item)

# Removing an element from the stack
def pop(stack):
    if (is_empty(stack)):
        return "The stack is empty"
    return stack.pop()

stack = create_stack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
push(stack, str(5))

for i in range(3):
    print("Popped:", pop(stack))


print("The elements in the stack are:"+ str(stack))
```

```
Pushed Element: 1
Pushed Element: 2
Pushed Element: 3
Pushed Element: 4
Pushed Element: 5
Popped: 5
Popped: 4
Popped: 3
The elements in the stack are:['1', '2']
```

Figure 2: Screenshot of output for question 3

The program extends the functionality by including the process of removing elements from the stack. After pushing the same five elements, it uses a loop to pop three items from the top of the stack. Since stacks follow the Last-In, First-Out (LIFO) principle, the elements removed are 5, 4, and 3, leaving only 1 and 2 in the stack. This clearly illustrates how the pop operation works in stacks by always removing the most recently added elements first.

```python
# Stack implementation in python
# Creating a stack
def create_stack():
    stack = []
    return stack

# Creating an empty stack
def is_empty(stack):
    return len(stack) == 0

# Adding items into the stack
def push(stack, item):
    stack.append(item)
    print("Pushed Element: " + item)

# Removing an element from the stack
def pop(stack):
    if (is_empty(stack)):
        return "The stack is empty"
    return stack.pop()

stack = create_stack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
push(stack, str(5))

print("The elements in the stack are:"+ str(stack))

def size(stack):
    return len(stack)
print("The size of the stack is:", size(stack))
```

```
Pushed Element: 1
Pushed Element: 2
Pushed Element: 3
Pushed Element: 4
Pushed Element: 5
The elements in the stack are:['1', '2', '3', '4', '5']
The size of the stack is: 5
```

Figure 3: Screenshot of output for question 5

The program introduces a new method called size(stack), which determines the number of elements currently in the stack. After pushing the five elements, the code prints both the content of the stack and its size. The output shows that the stack contains five elements, confirming the accuracy of the method. This addition emphasizes how programmers can extend stack implementations with useful utility functions such as checking the length of the stack.

IV. Conclusion

In conclusion, since I already knew how a stack works, this activity felt quite easy for me to follow. But even if I was already familiar with the concept, actually coding it step by step helped me understand it on a much deeper level. Seeing how the push, pop, and size operations worked in Python made me realize how simple but good data structure stacks really are. It also gave me more confidence in applying what I learn, not just in theory but also in practice. Overall, this activity may have been straightforward, but it still made me appreciate stacks more and strengthened my skills in both programming and problem-solving.

References

[1] Co Arthur O.. “University of Caloocan City Computer Engineering Department Honor Code,” UCC-CpE Departmental Policies, 2020.

[2] “Google Colab.” https://colab.research.google.com/drive/1l5JlnkkOLY7Nc73bJq_1hTf0du9M6cno#scrollTo=VFQ4mZxlLLZz

[3] Lewis-Clark-Palmes, “CPE-201L-DSA-2-A/LAB-8.ipynb at main · Lewis-Clark-Palmes/CPE-201L-DSA-2-A,” *GitHub*. https://github.com/Lewis-Clark-Palmes/CPE-201L-DSA-2-A/blob/main/LAB-8.ipynb