Homework3: CMSI-662: SECURE SOFTWARE DEVELOPMENT

Name: Peter Bukenya

1. **Give the titles and URLs of *three more* Tom Scott security-related videos you watched (to completion), together with a sentence or two on the purpose or lesson if each video.**

   Here are the three Tom Scott videos about security, along with a brief description of each:

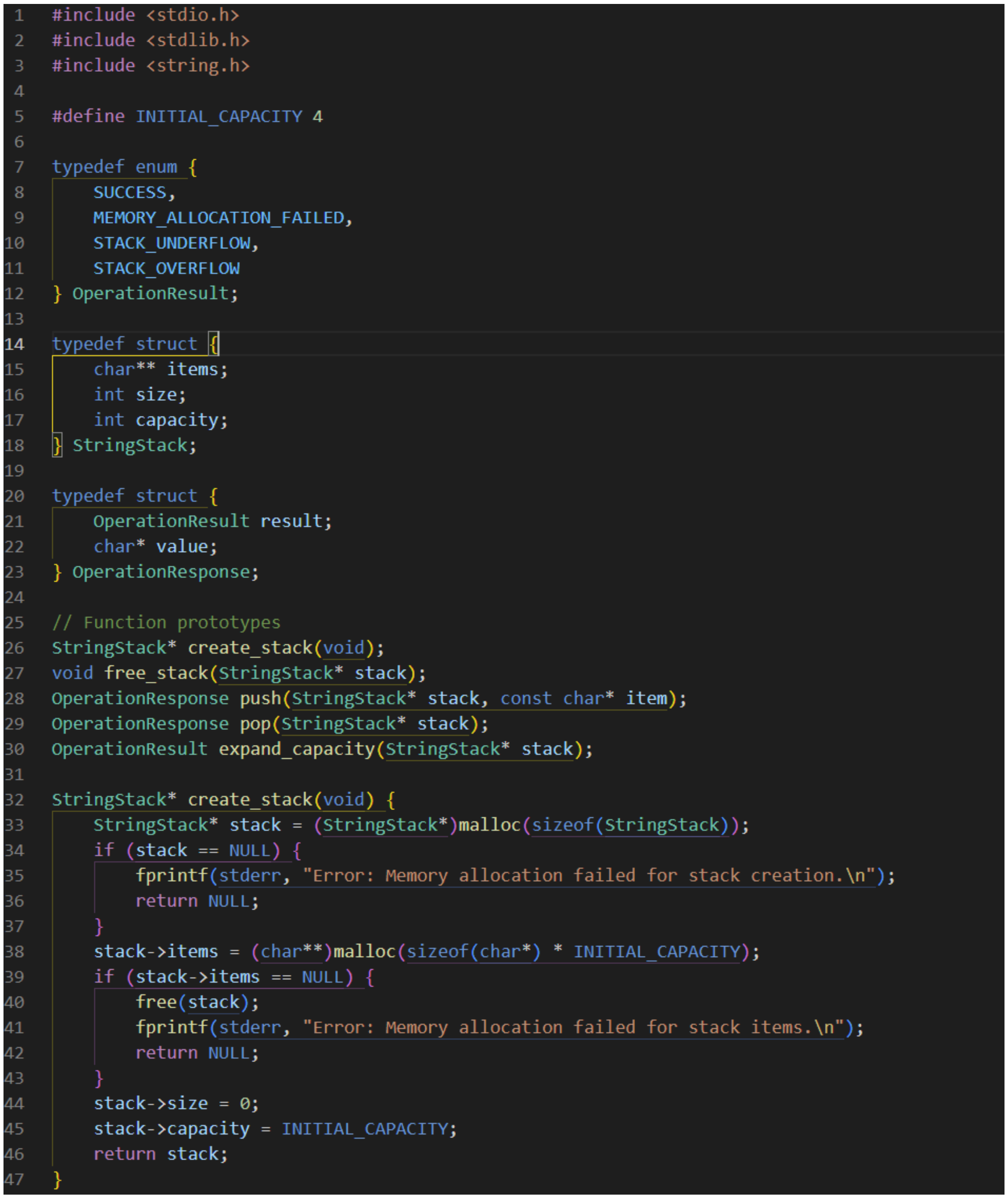
   a) The Moonpig Bug: How 3,000,000 Customers' Details Were Exposed https://www.youtube.com/watch?v=CgJudU_jlZ8 This explores a significant security flaw in Moonpig's API that inadvertently allowed unauthorized access to the personal details of millions of its customers. Through this video, Tom Scott highlights the importance of secure API design and the potential consequences of neglecting cybersecurity measures, demonstrating the ease with which hackers could exploit such vulnerabilities to access sensitive information.

   b) 2030: Privacy's Dead. What happens next?" https://www.youtube.com/watch?v=_kBIH-DQsEg Tom Scott explores the potential future where privacy as we know it has been completely eroded. He discusses the technological, social, and political changes that lead to this point, and speculates on how society will adapt to an era where every action is visible and every moment potentially monitored. The video aims to provoke thought on the importance of privacy and what we stand to lose if it disappears.

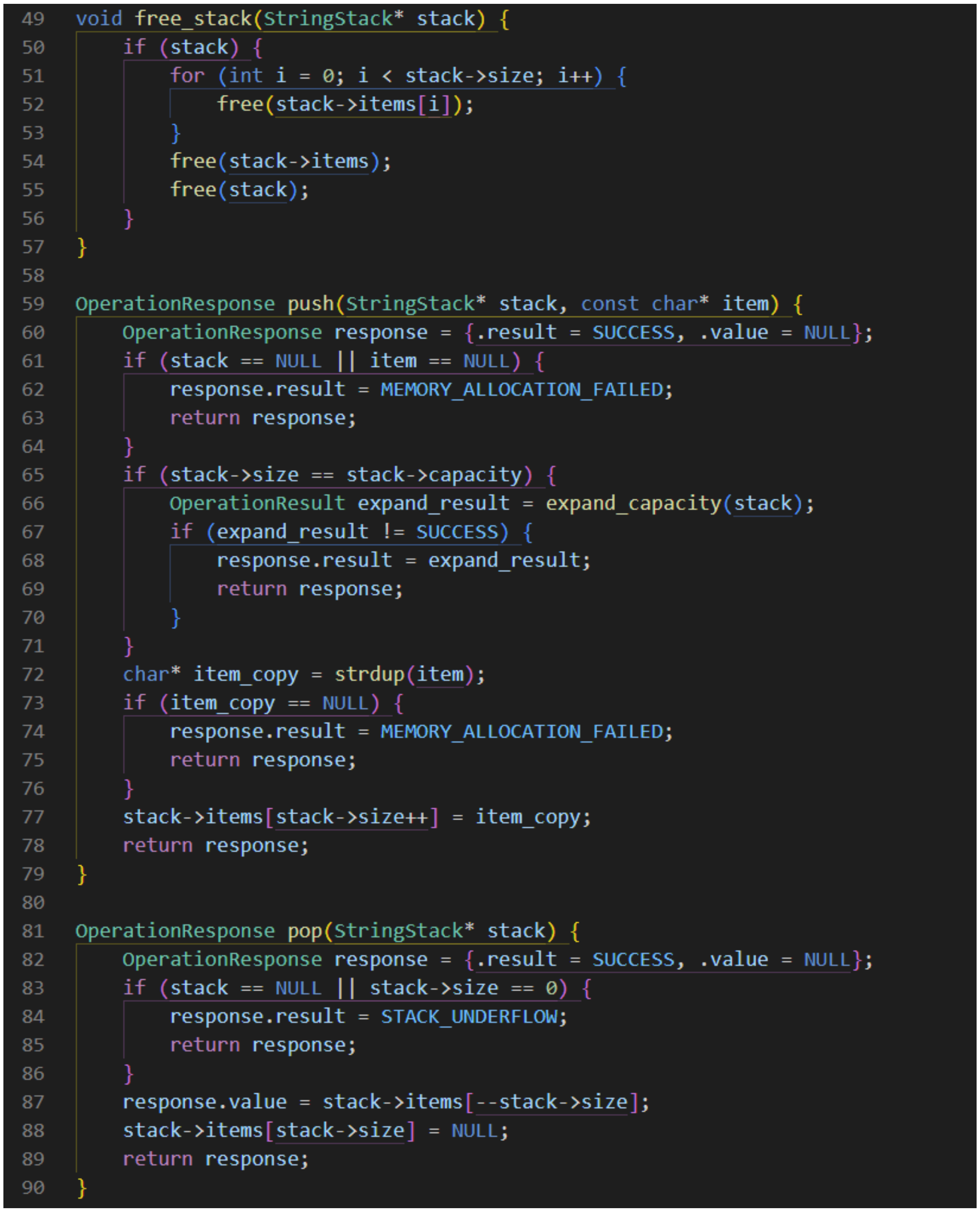
c) Why The Web Is Such A Mess https://www.youtube.com/watch?v=OFRjZtYs3wY provides a comprehensive look at the intricate and often chaotic nature of the web's infrastructure. This video explains the complexities and vulnerabilities inherent in the web's infrastructure, highlighting how legacy systems and backward compatibility contribute to security issues.

2. **Write a module defining a secure, expandable array-based stack of strings in C. (We will do a non-expandable version in class.) Fail fast by crashing with an error code (a different code for each type of failure), or return a "response object" that the caller can use to determine whether the operation succeeded, or if it did not, what happened.**

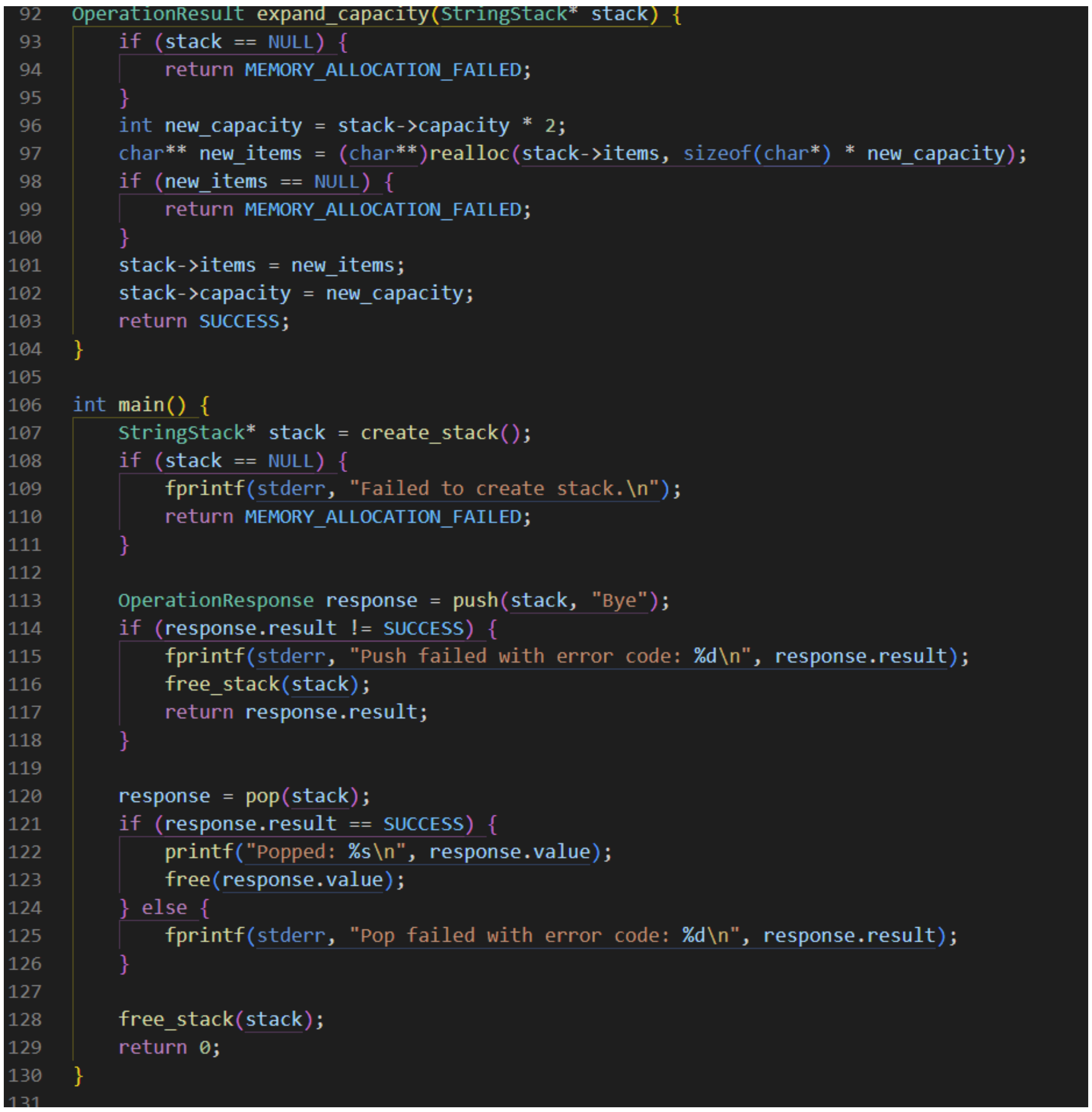
   https://github.com/buke2016/buke2016/tree/buke2016-patch-3/CMSI-662/Homework/Homework/Hoemwork3



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define INITIAL_CAPACITY 4

typedef enum {
    SUCCESS,
    MEMORY_ALLOCATION_FAILED,
    STACK_UNDERFLOW,
    STACK_OVERFLOW
} OperationResult;

typedef struct {
    char** items;
    int size;
    int capacity;
} StringStack;

typedef struct {
    OperationResult result;
    char* value;
} OperationResponse;

// Function prototypes
StringStack* create_stack(void);
void free_stack(StringStack* stack);
OperationResponse push(StringStack* stack, const char* item);
OperationResponse pop(StringStack* stack);
OperationResult expand_capacity(StringStack* stack);

StringStack* create_stack(void) {
    StringStack* stack = (StringStack*)malloc(sizeof(StringStack));
    if (stack == NULL) {
        fprintf(stderr, "Error: Memory allocation failed for stack creation.\n");
        return NULL;
    }
    stack->items = (char**)malloc(sizeof(char*) * INITIAL_CAPACITY);
    if (stack->items == NULL) {
        free(stack);
        fprintf(stderr, "Error: Memory allocation failed for stack items.\n");
        return NULL;
    }
    stack->size = 0;
    stack->capacity = INITIAL_CAPACITY;
    return stack;
}
```



```c
void free_stack(StringStack* stack) {
    if (stack) {
        for (int i = 0; i < stack->size; i++) {
            free(stack->items[i]);
        }
        free(stack->items);
        free(stack);
    }
}

OperationResponse push(StringStack* stack, const char* item) {
    OperationResponse response = {.result = SUCCESS, .value = NULL};
    if (stack == NULL || item == NULL) {
        response.result = MEMORY_ALLOCATION_FAILED;
        return response;
    }
    if (stack->size == stack->capacity) {
        OperationResult expand_result = expand_capacity(stack);
        if (expand_result != SUCCESS) {
            response.result = expand_result;
            return response;
        }
    }
    char* item_copy = strdup(item);
    if (item_copy == NULL) {
        response.result = MEMORY_ALLOCATION_FAILED;
        return response;
    }
    stack->items[stack->size++] = item_copy;
    return response;
}

OperationResponse pop(StringStack* stack) {
    OperationResponse response = {.result = SUCCESS, .value = NULL};
    if (stack == NULL || stack->size == 0) {
        response.result = STACK_UNDERFLOW;
        return response;
    }
    response.value = stack->items[--stack->size];
    stack->items[stack->size] = NULL;
    return response;
}
```

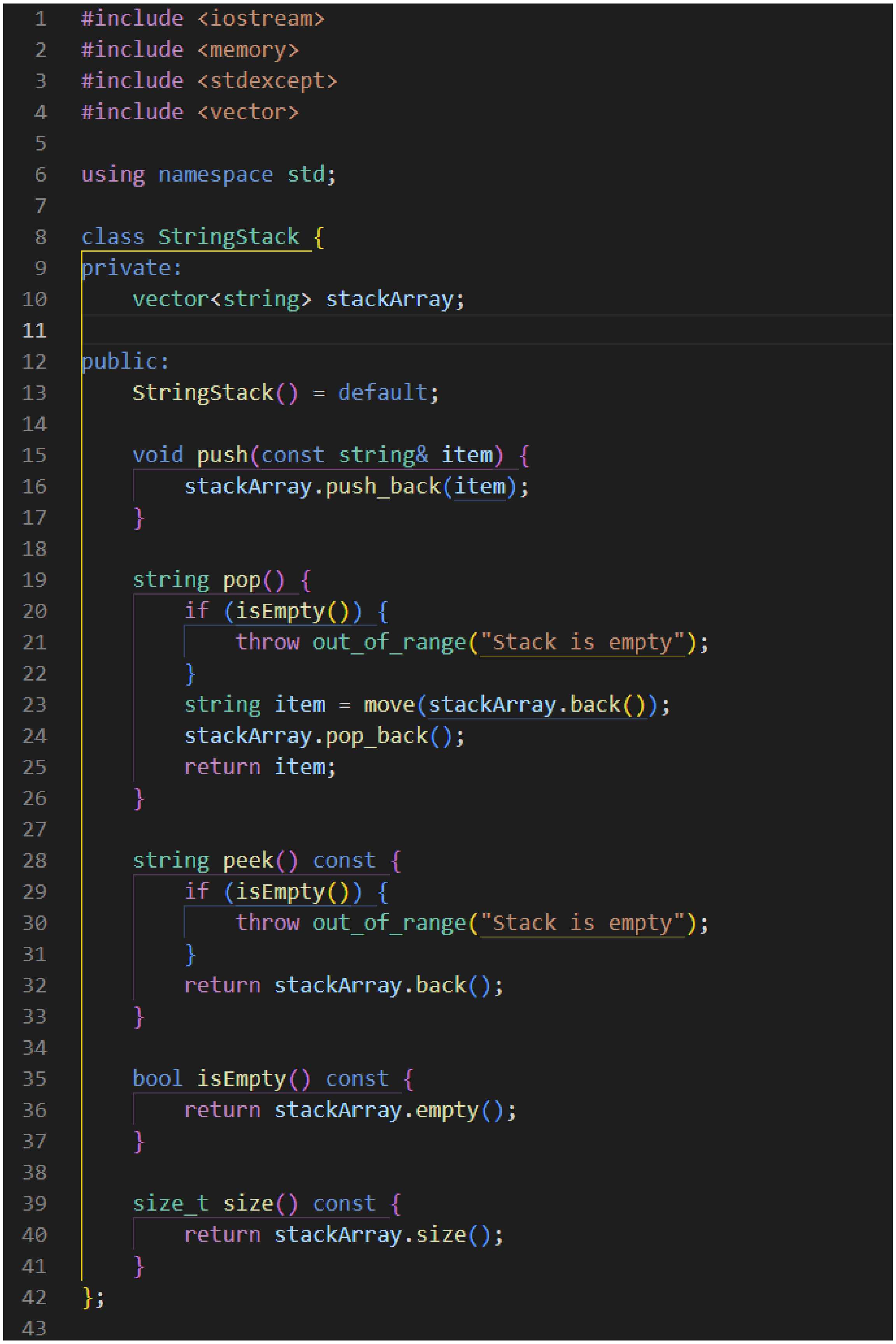


```c
OperationResult expand_capacity(StringStack* stack) {
    if (stack == NULL) {
        return MEMORY_ALLOCATION_FAILED;
    }
    int new_capacity = stack->capacity * 2;
    char** new_items = (char**)realloc(stack->items, sizeof(char*) * new_capacity);
    if (new_items == NULL) {
        return MEMORY_ALLOCATION_FAILED;
    }
    stack->items = new_items;
    stack->capacity = new_capacity;
    return SUCCESS;
}

int main() {
    StringStack* stack = create_stack();
    if (stack == NULL) {
        fprintf(stderr, "Failed to create stack.\n");
        return MEMORY_ALLOCATION_FAILED;
    }

    OperationResponse response = push(stack, "Bye");
    if (response.result != SUCCESS) {
        fprintf(stderr, "Push failed with error code: %d\n", response.result);
        free_stack(stack);
        return response.result;
    }

    response = pop(stack);
    if (response.result == SUCCESS) {
        printf("Popped: %s\n", response.value);
        free(response.value);
    } else {
        fprintf(stderr, "Pop failed with error code: %d\n", response.result);
    }

    free_stack(stack);
    return 0;
}
```

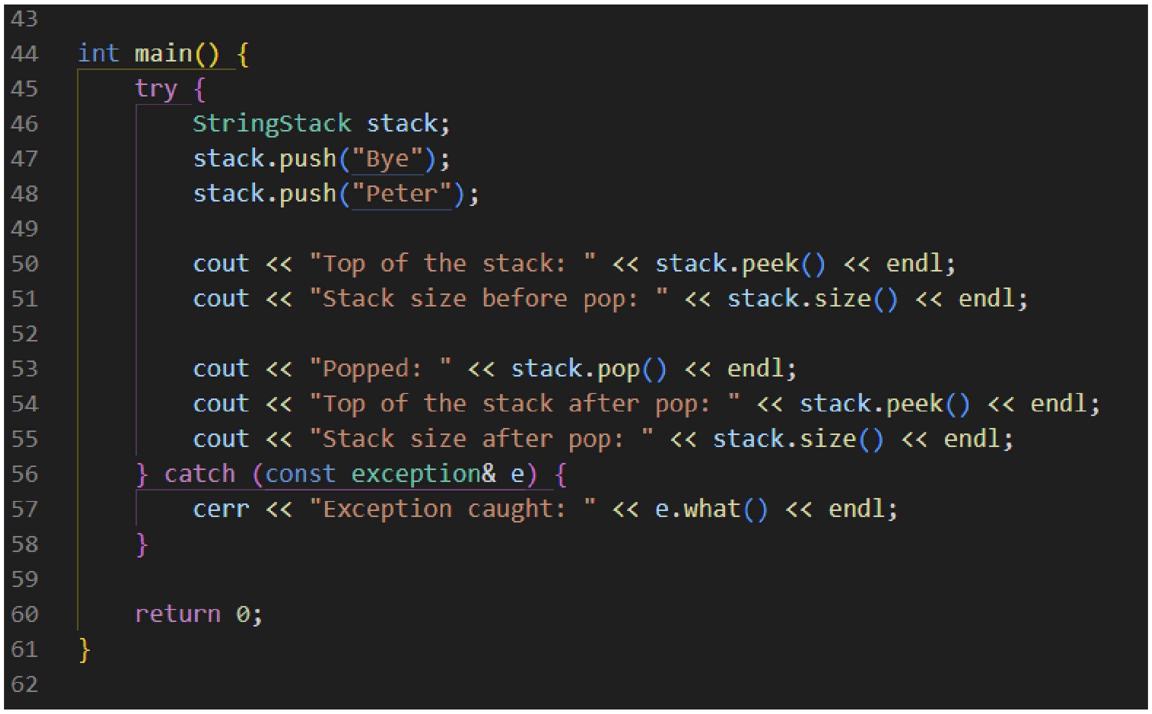
3. **Write a class for a secure expandable array-based stack of strings in C++, using a raw array of smart pointers for the stack. In practice, C++ programmers have a standard stack class, but in this course we are interested in building secure structures from first principles and getting practice with all**

the various features (and warts) of C++. Fail fast by throwing exceptions.

https://github.com/buke2016/buke2016/tree/buke2016-patch-3/CMSI-662/Homework/Homework/Hoemwork3



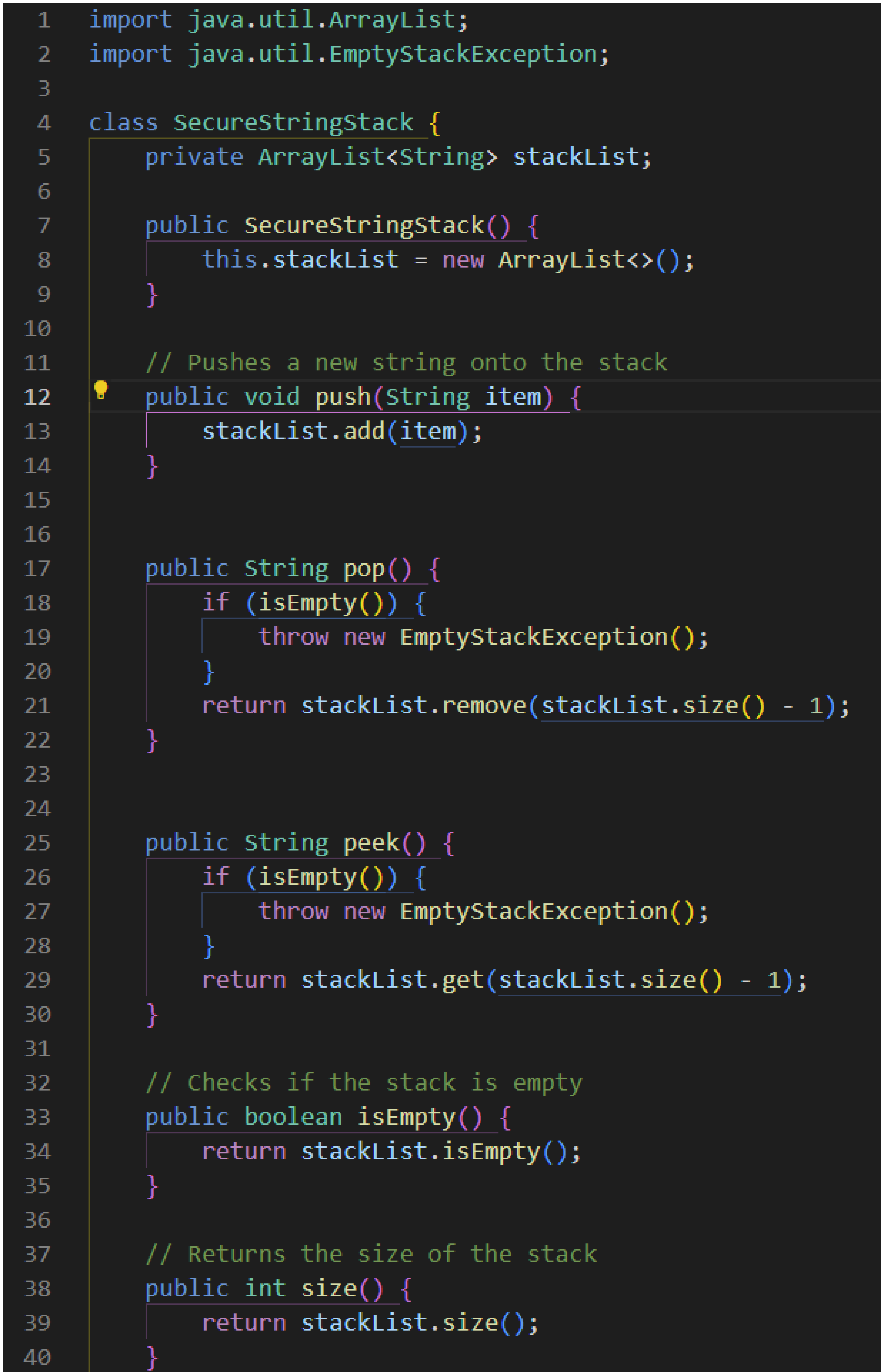
```cpp
#include <iostream>
#include <memory>
#include <stdexcept>
#include <vector>

using namespace std;

class StringStack {
private:
    vector<string> stackArray;

public:
    StringStack() = default;

    void push(const string& item) {
        stackArray.push_back(item);
    }

    string pop() {
        if (isEmpty()) {
            throw out_of_range("Stack is empty");
        }
        string item = move(stackArray.back());
        stackArray.pop_back();
        return item;
    }

    string peek() const {
        if (isEmpty()) {
            throw out_of_range("Stack is empty");
        }
        return stackArray.back();
    }

    bool isEmpty() const {
        return stackArray.empty();
    }

    size_t size() const {
        return stackArray.size();
    }
};

```



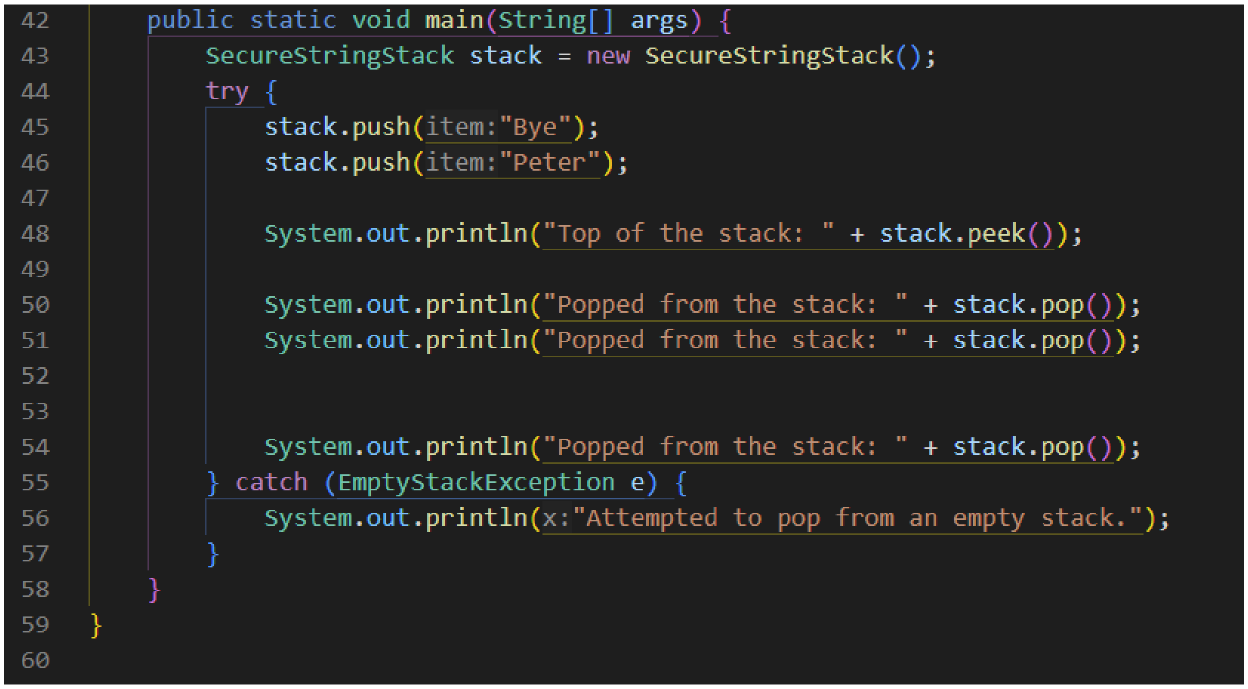
```cpp

int main() {
    try {
        StringStack stack;
        stack.push("Bye");
        stack.push("Peter");

        cout << "Top of the stack: " << stack.peek() << endl;
        cout << "Stack size before pop: " << stack.size() << endl;

        cout << "Popped: " << stack.pop() << endl;
        cout << "Top of the stack after pop: " << stack.peek() << endl;
        cout << "Stack size after pop: " << stack.size() << endl;
    } catch (const exception& e) {
        cerr << "Exception caught: " << e.what() << endl;
    }

    return 0;
}

```

4. **Write a class for a secure expandable array-based stack of strings in Java. Fail fast by throwing exceptions.**

   https://github.com/buke2016/buke2016/tree/buke2016-patch-3/CMSI-662/Homework/Homework/Hoemwork3



```java
import java.util.ArrayList;
import java.util.EmptyStackException;

class SecureStringStack {
    private ArrayList<String> stackList;

    public SecureStringStack() {
        this.stackList = new ArrayList<>();
    }

    // Pushes a new string onto the stack
    public void push(String item) {
        stackList.add(item);
    }


    public String pop() {
        if (isEmpty()) {
            throw new EmptyStackException();
        }
        return stackList.remove(stackList.size() - 1);
    }


    public String peek() {
        if (isEmpty()) {
            throw new EmptyStackException();
        }
        return stackList.get(stackList.size() - 1);
    }

    // Checks if the stack is empty
    public boolean isEmpty() {
        return stackList.isEmpty();
    }

    // Returns the size of the stack
    public int size() {
        return stackList.size();
    }
```



```java
    public static void main(String[] args) {
        SecureStringStack stack = new SecureStringStack();
        try {
            stack.push(item:"Bye");
            stack.push(item:"Peter");

            System.out.println("Top of the stack: " + stack.peek());

            System.out.println("Popped from the stack: " + stack.pop());
            System.out.println("Popped from the stack: " + stack.pop());


            System.out.println("Popped from the stack: " + stack.pop());
        } catch (EmptyStackException e) {
            System.out.println(x:"Attempted to pop from an empty stack.");
        }
    }
}

```