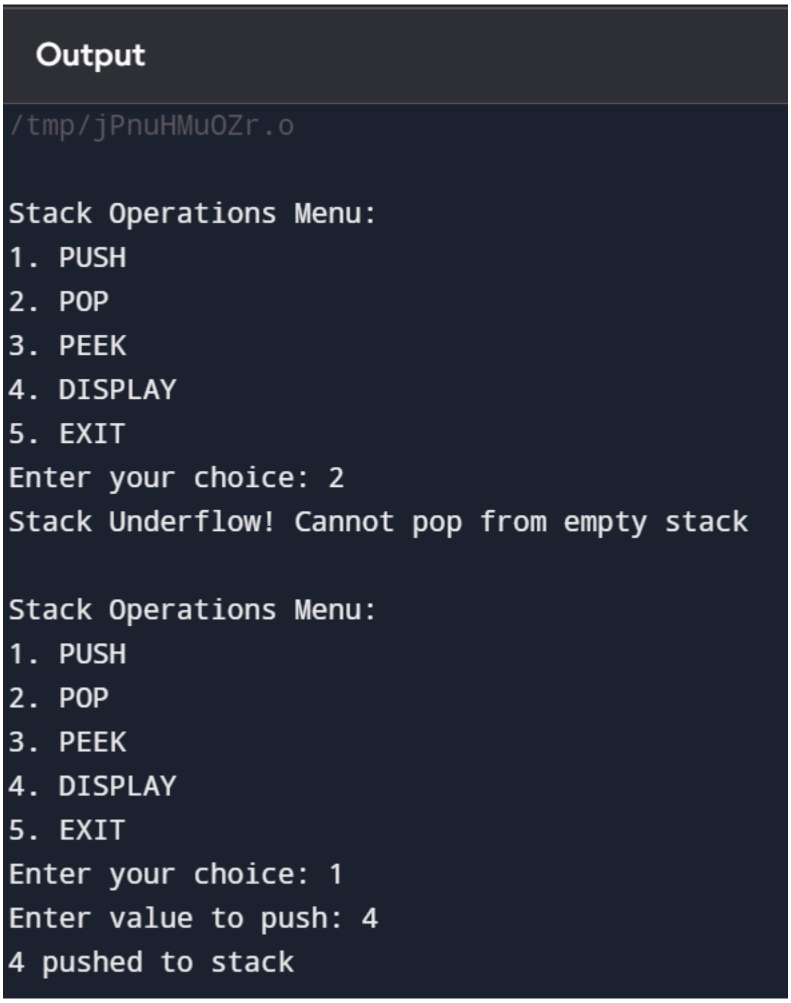
DAY-3-LAB

1) Write a C program to implement Stack operations using array such as PUSH, POP and PEEK.

```
#include <stdio.h>

#include <stdlib.h>


#define MAX 100 // Maximum size of the stack


// Stack structure

struct Stack {

  int top;

  int items[MAX];

};


// Function to create a stack and initialize its top

struct Stack* createStack() {

  struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));

  stack->top = -1; // Stack is initially empty

  return stack;

}


// Function to check if the stack is full

int isFull(struct Stack* stack) {

  return stack->top == MAX - 1;

}


// Function to check if the stack is empty
```

```
int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

// Function to add an element to the stack (PUSH)
void push(struct Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack Overflow! Cannot push %d\n", value);
    } else {
        stack->items[++stack->top] = value;
        printf("%d pushed to stack\n", value);
    }
}

// Function to remove an element from the stack (POP)
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow! Cannot pop from empty stack\n");
        return -1; // Return -1 to indicate stack is empty
    } else {
        return stack->items[stack->top--];
    }
}

// Function to return the top element of the stack (PEEK)
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! Cannot peek\n");
        return -1; // Return -1 to indicate stack is empty
    } else {
        return stack->items[stack->top];
    }
}

// Function to display the stack elements
void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
    } else {
        printf("Stack elements: ");
        for (int i = stack->top; i >= 0; i--) {
            printf("%d ", stack->items[i]);
        }
        printf("\n");
    }
}

// Main function to demonstrate stack operations
int main() {
    struct Stack* stack = createStack();
    int choice, value;

    while (1) {
        printf("\nStack Operations Menu:\n");
        printf("1. PUSH\n");
        printf("2. POP\n");
        printf("3. PEEK\n");
        printf("4. DISPLAY\n");
        printf("5. EXIT\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
               printf("Enter value to push: ");
               scanf("%d", &value);
               push(stack, value);
               break;
            case 2:
               value = pop(stack);
               if (value != -1) {
                  printf("%d popped from stack\n", value);
               }
               break;
            case 3:
               value = peek(stack);
               if (value != -1) {
                  printf("Top element is %d\n", value);
               }
               break;
            case 4:
               display(stack);
               break;
            case 5:
```

```
            free(stack);

            printf("Exiting...\n");

            exit(0);

          default:

            printf("Invalid choice! Please try again.\n");

        }

    }


    return 0;

}
```



```
/tmp/jPnuHMuOZr.o

Stack Operations Menu:
1. PUSH
2. POP
3. PEEK
4. DISPLAY
5. EXIT
Enter your choice: 2
Stack Underflow! Cannot pop from empty stack

Stack Operations Menu:
1. PUSH
2. POP
3. PEEK
4. DISPLAY
5. EXIT
Enter your choice: 1
Enter value to push: 4
4 pushed to stack
```

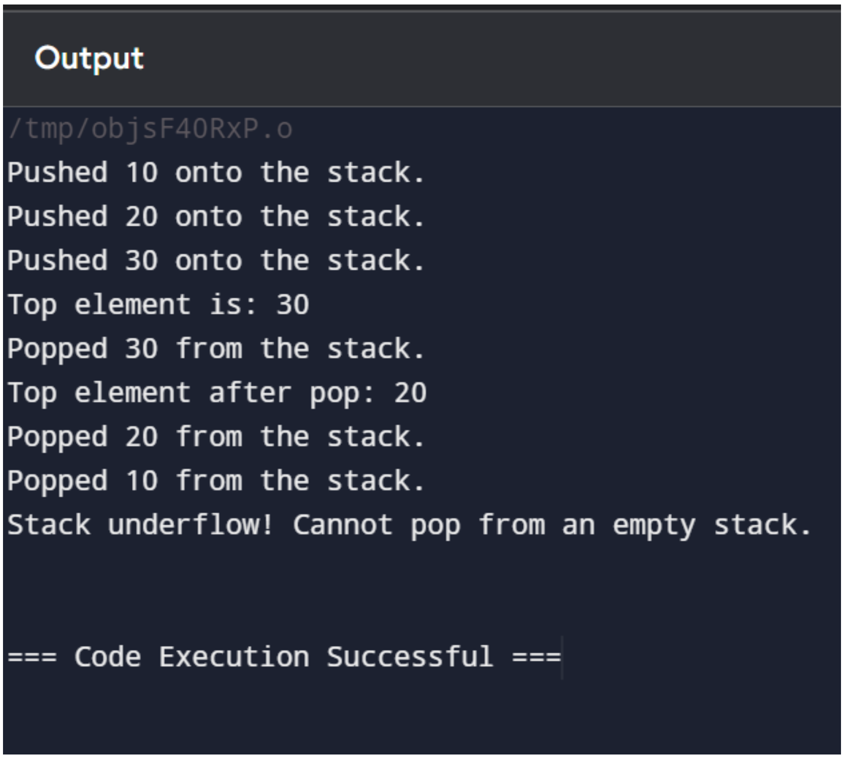
2) Write a C program to implement Stack operations using linked list such as PUSH, POP and PEEK.

```
#include <stdio.h>
```

```
#include <stdlib.h>

// Define a structure for the stack node
struct Node {
   int data;
   struct Node* next;
};

// Define a structure for the stack
struct Stack {
   struct Node* top;
};

// Function to create a new stack
struct Stack* createStack() {
   struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
   stack->top = NULL;
   return stack;
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
   return stack->top == NULL;
}

// Function to add an item to the stack (PUSH operation)
void push(struct Stack* stack, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    newNode->data = data;

    newNode->next = stack->top;

    stack->top = newNode;

    printf("Pushed %d onto the stack.\n", data);

}


// Function to remove an item from the stack (POP operation)

int pop(struct Stack* stack) {

    if (isEmpty(stack)) {

        printf("Stack underflow! Cannot pop from an empty stack.\n");

        return -1; // Return -1 to indicate stack underflow

    }

    struct Node* temp = stack->top;

    int poppedData = temp->data;

    stack->top = stack->top->next;

    free(temp);

    printf("Popped %d from the stack.\n", poppedData);

    return poppedData;

}


// Function to get the top item of the stack (PEEK operation)

int peek(struct Stack* stack) {

    if (isEmpty(stack)) {

        printf("Stack is empty! Cannot peek.\n");

        return -1; // Return -1 to indicate empty stack

    }

    return stack->top->data;

}
```

```
// Function to free the stack
void freeStack(struct Stack* stack) {
    while (!isEmpty(stack)) {
        pop(stack);
    }
    free(stack);
}

// Main function to demonstrate stack operations
int main() {
    struct Stack* stack = createStack();

    // Perform stack operations
    push(stack, 10);
    push(stack, 20);
    push(stack, 30);

    printf("Top element is: %d\n", peek(stack));

    pop(stack);
    printf("Top element after pop: %d\n", peek(stack));

    pop(stack);
    pop(stack);
    pop(stack); // Attempt to pop from an empty stack

    // Free the stack
    freeStack(stack);


    return 0;
}
```



Output

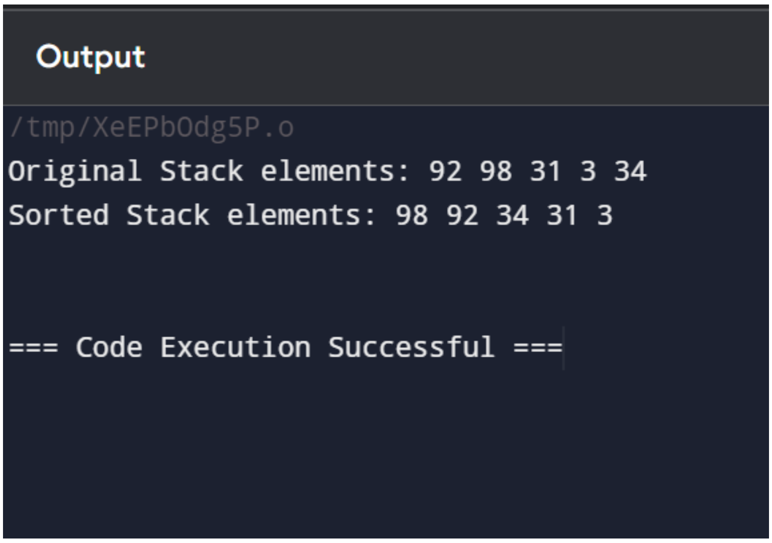
```
/tmp/objsF40RxP.o
Pushed 10 onto the stack.
Pushed 20 onto the stack.
Pushed 30 onto the stack.
Top element is: 30
Popped 30 from the stack.
Top element after pop: 20
Popped 20 from the stack.
Popped 10 from the stack.
Stack underflow! Cannot pop from an empty stack.


=== Code Execution Successful ===
```

3) Write a C program for Sorting elements using a stack (e.g., sorting a stack using recursion).

```
#include <stdio.h>

#include <stdlib.h>


// Define a structure for the stack node

struct Node {

    int data;

    struct Node* next;

};


// Define a structure for the stack

struct Stack {
    struct Node* top;
};

// Function to create a new stack
struct Stack* createStack() {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->top = NULL;
    return stack;
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
    return stack->top == NULL;
}

// Function to add an item to the stack (PUSH operation)
void push(struct Stack* stack, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = stack->top;
    stack->top = newNode;
}

// Function to remove an item from the stack (POP operation)
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        return -1; // Return -1 to indicate stack underflow
    }
    struct Node* temp = stack->top;
    int poppedData = temp->data;
    stack->top = stack->top->next;
    free(temp);
    return poppedData;
}

// Function to get the top item of the stack (PEEK operation)
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        return -1; // Return -1 to indicate empty stack
    }
    return stack->top->data;
}

// Function to sort the stack using recursion
void sortedInsert(struct Stack* stack, int element) {
    // Base case: If stack is empty or the element is greater than the top
    if (isEmpty(stack) || peek(stack) <= element) {
        push(stack, element);
        return;
    }

    // If the top element is greater, pop it and sort the remaining stack
    int topElement = pop(stack);
    sortedInsert(stack, element);

    // Push the top element back
    push(stack, topElement);
}

// Function to sort the stack
void sortStack(struct Stack* stack) {
    // Base case: If stack is not empty
    if (!isEmpty(stack)) {
        // Pop the top element
        int topElement = pop(stack);

        // Sort the remaining stack
        sortStack(stack);

        // Insert the popped element back in sorted order
        sortedInsert(stack, topElement);
    }
}

// Function to print the stack
void printStack(struct Stack* stack) {
    struct Node* current = stack->top;
    printf("Stack elements: ");
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
```

```
// Function to free the stack
void freeStack(struct Stack* stack) {
   while (!isEmpty(stack)) {
      pop(stack);
   }
   free(stack);
}

// Main function to demonstrate sorting a stack
int main() {
   struct Stack* stack = createStack();

   // Push elements onto the stack
   push(stack, 34);
   push(stack, 3);
   push(stack, 31);
   push(stack, 98);
   push(stack, 92);

   printf("Original ");
   printStack(stack);

   // Sort the stack
   sortStack(stack);

   printf("Sorted ");
   printStack(stack);
```

```
    // Free the stack
    freeStack(stack);

    return 0;
}
```



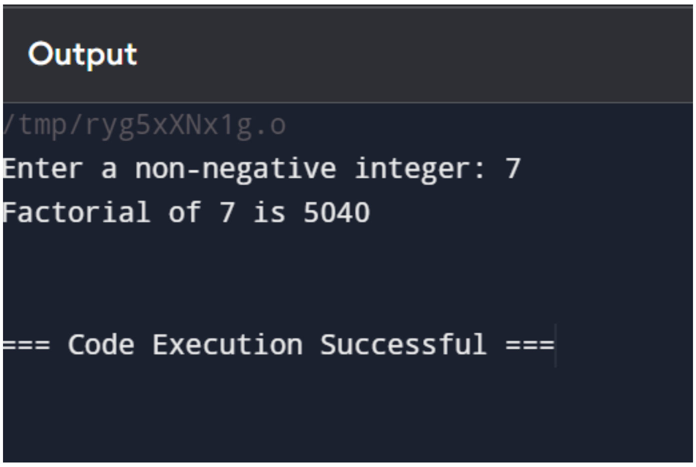
```
/tmp/XeEPbOdg5P.o
Original Stack elements: 92 98 31 3 34
Sorted Stack elements: 98 92 34 31 3


=== Code Execution Successful ===
```

4) Write a C Program to Simulate Recursive Function Calls Using a Stack

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

#define MAX_STACK_SIZE 100

// Stack structure
typedef struct {
    int top;
    int items[MAX_STACK_SIZE];
} Stack;

// Function to create a stack
```

```
Stack* createStack() {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->top = -1;
    return stack;
}

// Function to check if the stack is empty
int isEmpty(Stack* stack) {
    return stack->top == -1;
}

// Function to push an item onto the stack
void push(Stack* stack, int item) {
    if (stack->top < MAX_STACK_SIZE - 1) {
        stack->items[++stack->top] = item;
    } else {
        printf("Stack overflow\n");
    }
}

// Function to pop an item from the stack
int pop(Stack* stack) {
    if (!isEmpty(stack)) {
        return stack->items[stack->top--];
    } else {
        printf("Stack underflow\n");
        return -1; // Return an invalid value
    }
```

```
}

// Function to simulate factorial calculation
int factorial(int n) {
    Stack* stack = createStack();
    int result = 1;

    // Push initial value onto the stack
    push(stack, n);

    while (!isEmpty(stack)) {
        n = pop(stack);

        if (n == 0 || n == 1) {
            result *= 1; // Base case
        } else {
            result *= n; // Multiply result by n
            push(stack, n - 1); // Push next value onto the stack
        }
    }

    free(stack);
    return result;
}

int main() {
    int number;
    printf("Enter a non-negative integer: ");
    scanf("%d", &number);

    if (number < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    } else {
        int result = factorial(number);
        printf("Factorial of %d is %d\n", number, result);
    }

    return 0;
}
```

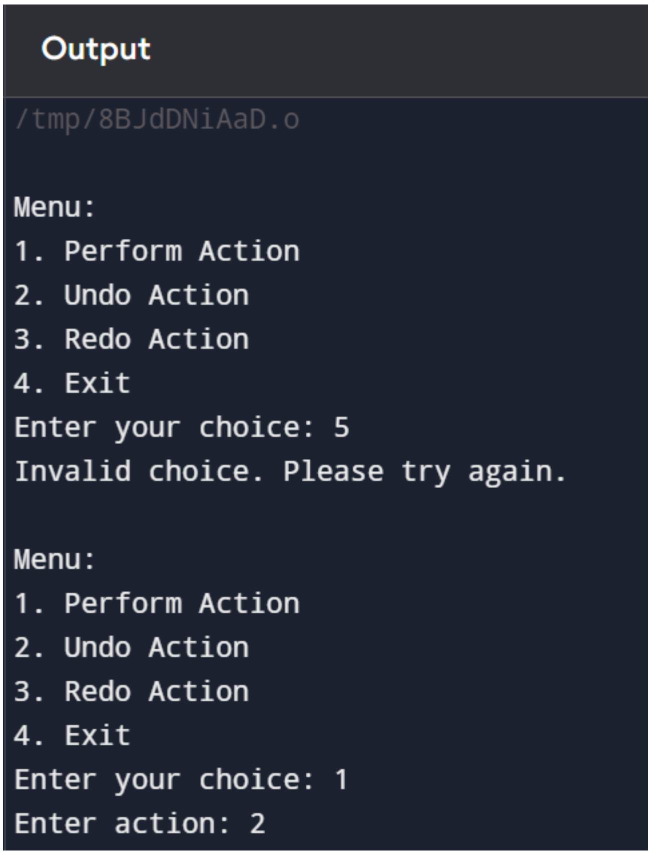


```
/tmp/ryg5xXNx1g.o
Enter a non-negative integer: 7
Factorial of 7 is 5040


=== Code Execution Successful ===
```

5) Write a C program to Implement undo and redo functionality using two stacks.

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

#define MAX_STACK_SIZE 100

// Stack structure
typedef struct {
```

```c
    int top;

    char items[MAX_STACK_SIZE][100]; // Store actions as strings

} Stack;


// Function to create a stack

Stack* createStack() {

    Stack* stack = (Stack*)malloc(sizeof(Stack));

    stack->top = -1;

    return stack;

}


// Function to check if the stack is empty

int isEmpty(Stack* stack) {

    return stack->top == -1;

}


// Function to push an item onto the stack

void push(Stack* stack, const char* item) {

    if (stack->top < MAX_STACK_SIZE - 1) {

        strcpy(stack->items[++stack->top], item);

    } else {

        printf("Stack overflow\n");

    }

}


// Function to pop an item from the stack

char* pop(Stack* stack) {

    if (!isEmpty(stack)) {
        return stack->items[stack->top--];
    } else {
        printf("Stack underflow\n");
        return NULL; // Return NULL if stack is empty
    }
}

// Function to perform an action
void performAction(Stack* undoStack, Stack* redoStack, const char* action) {
    push(undoStack, action); // Push action to undo stack
    // Clear the redo stack since a new action is performed
    redoStack->top = -1;
}

// Function to undo the last action
void undo(Stack* undoStack, Stack* redoStack) {
    if (!isEmpty(undoStack)) {
        char* action = pop(undoStack);
        printf("Undid action: %s\n", action);
        push(redoStack, action); // Push the undone action to redo stack
    } else {
        printf("Nothing to undo\n");
    }
}

// Function to redo the last undone action
void redo(Stack* undoStack, Stack* redoStack) {
    if (!isEmpty(redoStack)) {
        char* action = pop(redoStack);
        printf("Redid action: %s\n", action);
        push(undoStack, action); // Push the redone action back to undo stack
    } else {
        printf("Nothing to redo\n");
    }
}

int main() {
    Stack* undoStack = createStack();
    Stack* redoStack = createStack();
    char action[100];
    int choice;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Perform Action\n");
        printf("2. Undo Action\n");
        printf("3. Redo Action\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar(); // Consume newline character

        switch (choice) {
            case 1:
                printf("Enter action: ");
                fgets(action, sizeof(action), stdin);
                action[strcspn(action, "\n")] = 0; // Remove newline character
                performAction(undoStack, redoStack, action);
                break;
            case 2:
                undo(undoStack, redoStack);
                break;
            case 3:
                redo(undoStack, redoStack);
                break;
            case 4:
                free(undoStack);
                free(redoStack);
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```



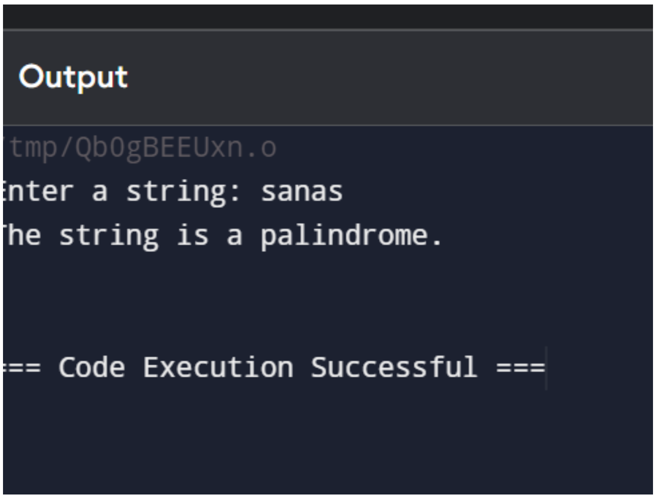
```
/tmp/8BJdDNiAaD.o

Menu:
1. Perform Action
2. Undo Action
3. Redo Action
4. Exit
Enter your choice: 5
Invalid choice. Please try again.

Menu:
1. Perform Action
2. Undo Action
3. Redo Action
4. Exit
Enter your choice: 1
Enter action: 2
```

6) Write a C program to Check if a string is a palindrome using a stack.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>



#define MAX 100



// Stack structure

typedef struct {

  char items[MAX];

  int top;

} Stack;



// Function to initialize the stack
```

```
void initStack(Stack *s) {
    s->top = -1;
}

// Function to check if the stack is full
int isFull(Stack *s) {
    return s->top == MAX - 1;
}

// Function to check if the stack is empty
int isEmpty(Stack *s) {
    return s->top == -1;
}

// Function to push an element onto the stack
void push(Stack *s, char item) {
    if (!isFull(s)) {
        s->items[++s->top] = item;
    }
}

// Function to pop an element from the stack
char pop(Stack *s) {
    if (!isEmpty(s)) {
        return s->items[s->top--];
    }
    return '\0'; // Return null character if stack is empty
}
```

```
// Function to check if a string is a palindrome
int isPalindrome(char *str) {
    Stack s;
    initStack(&s);

    // Normalize the string: convert to lowercase and ignore non-alphanumeric characters
    char normalized[MAX];
    int j = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (isalnum(str[i])) {
            normalized[j++] = tolower(str[i]);
            push(&s, tolower(str[i])); // Push to stack
        }
    }
    normalized[j] = '\0'; // Null-terminate the normalized string

    // Check for palindrome
    for (int i = 0; i < j; i++) {
        if (normalized[i] != pop(&s)) {
            return 0; // Not a palindrome
        }
    }
    return 1; // Is a palindrome
}

int main() {
    char str[MAX];
    printf("Enter a string: ");

    fgets(str, sizeof(str), stdin);


    // Remove newline character if present

    str[strcspn(str, "\n")] = 0;


    if (isPalindrome(str)) {

        printf("The string is a palindrome.\n");

    } else {

        printf("The string is not a palindrome.\n");

    }


    return 0;

}
```



Output

```
tmp/QbOgBEEUxn.o
nter a string: sanas
he string is a palindrome.


== Code Execution Successful ===
```