**1.implement Stack operations using array such as PUSH, POP and PEEK.**

#include <stdio.h>

#define MAX 100

typedef struct {

int top;

int arr[MAX];

} Stack;

void push(Stack \*s, int value) {

if (s->top < MAX - 1) {

s->arr[++s->top] = value;

printf("%d pushed\n", value);

} else {

printf("Stack overflow\n");

}

}

int pop(Stack \*s) {

if (s->top >= 0) {

return s->arr[s->top--];

} else {

printf("Stack underflow\n");

return -1; // Error value

}

}

int peek(Stack \*s) {

if (s->top >= 0) {

return s->arr[s->top];

} else {

printf("Stack is empty\n");

return -1; // Error value

}

}

int main() {

Stack s = { .top = -1 }; // Initialize stack

push(&s, 10);

push(&s, 20);

push(&s, 30);

printf("Top element is %d\n", peek(&s));

printf("%d popped\n", pop(&s));

return 0;

}

**Output:**



**2.implement Stack operations using linked list such as PUSH, POP and PEEK.**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

void push(Node \*\*top, int value) {

Node \*newNode = (Node \*)malloc(sizeof(Node));

if (!newNode) {

printf("Memory allocation failed\n");

return;

}

newNode->data = value;

newNode->next = \*top;

\*top = newNode;

printf("%d pushed\n", value);

}

int pop(Node \*\*top) {

if (\*top == NULL) {

printf("Stack underflow\n");

return -1; // Error value

}

Node \*temp = \*top;

int value = temp->data;

\*top = (\*top)->next;

free(temp);

return value;

}

int peek(Node \*top) {

if (top == NULL) {

printf("Stack is empty\n");

return -1; // Error value

}

return top->data;

}

int main() {

Node \*stack = NULL; // Initialize the stack

push(&stack, 10);

push(&stack, 20);

push(&stack, 30);

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

printf("%d popped\n", pop(&stack));

return 0;

}

**Output:**



**3.Sorting elements using a stack:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct {

int arr[MAX];

int top;

} Stack;

void initStack(Stack \*s) {

s->top = -1;

}

void push(Stack \*s, int value) {

if (s->top < MAX - 1) {

s->arr[++s->top] = value;

}

}

int pop(Stack \*s) {

return s->top == -1 ? -1 : s->arr[s->top--];

}

int peek(Stack \*s) {

return s->top == -1 ? -1 : s->arr[s->top];

}

void sortedInsert(Stack \*s, int value) {

if (s->top == -1 || peek(s) <= value) {

push(s, value);

} else {

int temp = pop(s);

sortedInsert(s, value);

push(s, temp);

}

}

void sortStack(Stack \*s) {

if (s->top != -1) {

int value = pop(s);

sortStack(s);

sortedInsert(s, value);

}

}

void printStack(Stack \*s) {

for (int i = s->top; i >= 0; i--) {

printf("%d ", s->arr[i]);

}

printf("\n");

}

int main() {

Stack s;

initStack(&s);

push(&s, 30);

push(&s, 10);

push(&s, 50);

push(&s, 20);

push(&s, 40);

printf("Original stack:\n");

printStack(&s);

sortStack(&s);

printf("Sorted stack:\n");

printStack(&s);

return 0;

}

**Output:**



**4.Simulate Recursive Function Calls Using a Stack:**

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int \*data;

int top;

int capacity;

} Stack;

Stack\* createStack(int capacity) {

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

stack->capacity = capacity;

stack->top = -1;

stack->data = (int \*)malloc(capacity \* sizeof(int));

return stack;

}

void push(Stack \*stack, int value) {

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

stack->data[++stack->top] = value;

}

}

int pop(Stack \*stack) {

if (stack->top == -1) return -1;

return stack->data[stack->top--];

}

int isEmpty(Stack \*stack) {

return stack->top == -1;

}

int factorial(int n) {

Stack \*stack = createStack(n + 1);

int result = 1;

while (n > 1) {

push(stack, n);

n--;

}

while (!isEmpty(stack)) {

result \*= pop(stack);

}

free(stack->data);

free(stack);

return result;

}

int main() {

int num = 5;

printf("Factorial of %d is %d\n", num, factorial(num));

return 0;

}

**Output:**



**5.Implement undo and redo functionality using two stacks:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

int undoStack[MAX\_SIZE];

int redoStack[MAX\_SIZE];

int undoTop = -1;

int redoTop = -1;

void pushUndo(int item) {

if (undoTop >= MAX\_SIZE - 1) {

printf("Undo Stack Overflow\n");

} else {

undoStack[++undoTop] = item;

}

}

void pushRedo(int item) {

if (redoTop >= MAX\_SIZE - 1) {

printf("Redo Stack Overflow\n");

} else {

redoStack[++redoTop] = item;

}

}

int popUndo() {

if (undoTop < 0) {

printf("Undo Stack Underflow\n");

return -1;

} else {

return undoStack[undoTop--];

}

}

int popRedo() {

if (redoTop < 0) {

printf("Redo Stack Underflow\n");

return -1;

} else {

return redoStack[redoTop--];

}

}

int main() {

pushUndo(1);

pushUndo(2);

pushUndo(3);

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

while (undoTop >= 0) {

int item = popUndo();

printf("Undo: %d\n", item);

pushRedo(item);

}

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

while (redoTop >= 0) {

int item = popRedo();

printf("Redo: %d\n", item);

}

return 0;

}

**Output:**



**6.Check if a string is a palindrome using a stack:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX 100

struct Stack {

int top;

char array[MAX];

};

struct Stack\* createStack() {

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

stack->top = -1;

return stack;

}

void push(struct Stack\* stack, char item) {

stack->array[++stack->top] = item;

}

char pop(struct Stack\* stack) {

return stack->array[stack->top--];

}

int isPalindrome(char str[]) {

int n = strlen(str);

struct Stack\* stack = createStack();

int i, mid = n / 2;

for (i = 0; i < mid; i++) {

push(stack, str[i]);

}

if (n % 2 != 0) {

i++;

}

while (str[i] != '\0') {

char item = pop(stack);

if (item != str[i])

return 0;

i++;

}

return 1;

}

int main() {

char str[MAX];

printf("Enter a string: ");

scanf("%s", str);

if (isPalindrome(str))

printf("%s is a palindrome.\n");

else

printf("%s is not a palindrome.\n");

return 0;

}

**Output:**

