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

# Stack Operations Using Array

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

struct Stack {

int top;

int items[MAX];

};

void initStack(struct Stack\* s) {

s->top = -1;

}

int isFull(struct Stack\* s) {

return s->top == MAX - 1;

}

int isEmpty(struct Stack\* s) {

return s->top == -1;

}

void push(struct Stack\* s, int item) {

if (isFull(s)) {

printf("Stack Overflow\n");

} else {

s->items[++s->top] = item;

printf("%d pushed to stack\n", item);

}

}

int pop(struct Stack\* s) {

if (isEmpty(s)) {

printf("Stack Underflow\n");

return -1;

} else {

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

}

}

int peek(struct Stack\* s) {

if (isEmpty(s)) {

printf("Stack is empty\n");

return -1;

} else {

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

}

}

int main() {

struct Stack stack;

initStack(&stack);

push(&stack, 10);

push(&stack, 20);

push(&stack, 30);

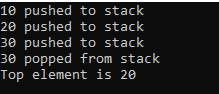
printf("%d popped from stack\n", pop(&stack));

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

return 0;

}

Output:



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

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Stack {

struct Node\* top;

};

struct Stack\* createStack() {

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

stack->top = NULL;

return stack;

}

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;

}

int pop(struct Stack\* stack) {

if (stack->top == NULL) {

printf("Stack Underflow\n");

return -1;

}

struct Node\* temp = stack->top;

int popped = temp->data;

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

free(temp);

return popped;

}

int peek(struct Stack\* stack) {

if (stack->top == NULL) {

printf("Stack is empty\n");

return -1;

}

return stack->top->data;

}

int main() {

struct Stack\* stack = createStack();

push(stack, 10);

push(stack, 20);

push(stack, 30);

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

printf("Popped element is %d\n", pop(stack));

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

return 0;

}

Output:



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

#include <stdio.h>

#include <stdlib.h>

struct Stack {

int top;

unsigned capacity;

int\* array;

};

struct Stack\* createStack(unsigned capacity) {

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

stack->capacity = capacity;

stack->top = -1;

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

return stack;

}

int isFull(struct Stack\* stack) {

return stack->top == stack->capacity - 1;

}

int isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

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

if (isFull(stack)) return;

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

}

int pop(struct Stack\* stack) {

if (isEmpty(stack)) return -1;

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

}

int peek(struct Stack\* stack) {

if (isEmpty(stack)) return -1;

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

}

void sortedInsert(struct Stack\* stack, int element) {

if (isEmpty(stack) || element > peek(stack)) {

push(stack, element);

return;

}

int temp = pop(stack);

sortedInsert(stack, element);

push(stack, temp);

}

void sortStack(struct Stack\* stack) {

if (!isEmpty(stack)) {

int temp = pop(stack);

sortStack(stack);

sortedInsert(stack, temp);

}

}

void printStack(struct Stack\* stack) {

for (int i = 0; i <= stack->top; i++) {

printf("%d ", stack->array[i]);

}

printf("\n");

}

int main() {

struct Stack\* stack = createStack(100);

push(stack, 34);

push(stack, 3);

push(stack, 31);

push(stack, 98);

push(stack, 92);

printf("Original Stack:\n");

printStack(stack);

sortStack(stack);

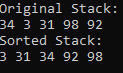
printf("Sorted Stack:\n");

printStack(stack);

return 0;

}

Output:



1. Write a C Program to Simulate Recursive Function Calls Using a Stack:

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct {

int top;

int items[MAX];

} Stack;

void initStack(Stack\* s) {

s->top = -1;

}

int isFull(Stack\* s) {

return s->top == MAX - 1;

}

int isEmpty(Stack\* s) {

return s->top == -1;

}

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

if (!isFull(s)) {

s->items[++s->top] = item;

}

}

int pop(Stack\* s) {

if (!isEmpty(s)) {

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

}

return -1; // Return -1 if stack is empty

}

void simulateRecursiveFunction(int n) {

Stack stack;

initStack(&stack);

while (n > 0) {

push(&stack, n);

n--;

}

while (!isEmpty(&stack)) {

int value = pop(&stack);

printf("Simulated recursive call with value: %d\n", value);

}

}

int main() {

int n;

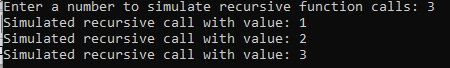
printf("Enter a number to simulate recursive function calls: ");

scanf("%d", &n);

simulateRecursiveFunction(n);

return 0;

}



1. Write a C program to Implement undo and redo functionality using two stacks:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

typedef struct Stack {

char\* items[MAX];

int top;

} Stack;

void initStack(Stack\* s) {

s->top = -1;

}

int isFull(Stack\* s) {

return s->top == MAX - 1;

}

int isEmpty(Stack\* s) {

return s->top == -1;

}

void push(Stack\* s, char\* item) {

if (!isFull(s)) {

s->items[++(s->top)] = strdup(item);

}

}

char\* pop(Stack\* s) {

if (!isEmpty(s)) {

return s->items[(s->top)--];

}

return NULL;

}

char\* peek(Stack\* s) {

if (!isEmpty(s)) {

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

}

return NULL;

}

void freeStack(Stack\* s) {

while (!isEmpty(s)) {

free(pop(s));

}

}

void undo(Stack\* undoStack, Stack\* redoStack) {

char\* lastAction = pop(undoStack);

if (lastAction) {

printf("Undo: %s\n", lastAction);

push(redoStack, lastAction);

} else {

printf("Nothing to undo.\n");

}

}

void redo(Stack\* undoStack, Stack\* redoStack) {

char\* lastUndone = pop(redoStack);

if (lastUndone) {

printf("Redo: %s\n", lastUndone);

push(undoStack, lastUndone);

} else {

printf("Nothing to redo.\n");

}

}

int main() {

Stack undoStack, redoStack;

initStack(&undoStack);

initStack(&redoStack);

push(&undoStack, "Action 1");

push(&undoStack, "Action 2");

push(&undoStack, "Action 3");

undo(&undoStack, &redoStack);

undo(&undoStack, &redoStack);

redo(&undoStack, &redoStack);

undo(&undoStack, &redoStack);

redo(&undoStack, &redoStack);

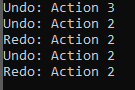
freeStack(&undoStack);

freeStack(&redoStack);

return 0;

}

Output:



1. Write a C program to Check if a string is a palindrome using a stack:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX 100

typedef struct Stack {

char arr[MAX];

int top;

} Stack;

void initStack(Stack\* s) {

s->top = -1;

}

int isFull(Stack\* s) {

return s->top == MAX - 1;

}

int isEmpty(Stack\* s) {

return s->top == -1;

}

void push(Stack\* s, char c) {

if (!isFull(s)) {

s->arr[++(s->top)] = c;

}

}

char pop(Stack\* s) {

if (!isEmpty(s)) {

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

}

return '\0';

}

int isPalindrome(char\* str) {

Stack s;

initStack(&s);

int len = strlen(str);

for (int i = 0; i < len; i++) {

push(&s, str[i]);

}

for (int i = 0; i < len; i++) {

if (str[i] != pop(&s)) {

return 0;

}

}

return 1;

}

int main() {

char str[MAX];

printf("Enter a string: ");

fgets(str, MAX, stdin);

str[strcspn(str, "\n")] = 0; // Remove newline character

if (isPalindrome(str)) {

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

} else {

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

}

return 0;

}

Output:

