5 . Write a program Implement a stack data structure and perform its operations.

// C program for array implementation of stack

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

// A structure to represent a stack

struct Stack {

int top;

unsigned capacity;

int\* array;

};

// function to create a stack of given capacity. It initializes size of

// stack as 0

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;

}

// Stack is full when top is equal to the last index

int isFull(struct Stack\* stack)

{

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

}

// Stack is empty when top is equal to -1

int isEmpty(struct Stack\* stack)

{

return stack->top == -1;

}

// Function to add an item to stack. It increases top by 1

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

{

if (isFull(stack))

return;

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

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

}

// Function to remove an item from stack. It decreases top by 1

int pop(struct Stack\* stack)

{

if (isEmpty(stack))

return INT\_MIN;

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

}

// Function to return the top from stack without removing it

int peek(struct Stack\* stack)

{

if (isEmpty(stack))

return INT\_MIN;

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

}

// Driver program to test above functions

int main()

{

struct Stack\* stack = createStack(100);

push(stack, 10);

push(stack, 20);

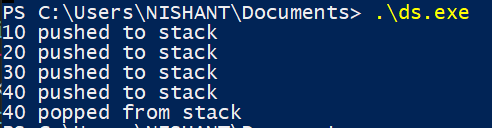
push(stack, 30);

push(stack, 40);

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

return 0;

}



# 6. Write a program Implement two stack using single array.

//This is a C Program to Implement two Stacks using a Single Array & Check for Overflow & Underflow

#include <stdio.h>

#define SIZE 10

int ar[SIZE];

int top1 = -1;

int top2 = SIZE;

//Functions to push data

void push\_stack1 (int data)

{

if (top1 < top2 - 1)

{

ar[++top1] = data;

}

else

{

printf ("Stack Full! Cannot Push\n");

}

}

void push\_stack2 (int data)

{

if (top1 < top2 - 1)

{

ar[--top2] = data;

}

else

{

printf ("Stack Full! Cannot Push\n");

}

}

//Functions to pop data

void pop\_stack1 ()

{

if (top1 >= 0)

{

int popped\_value = ar[top1--];

printf ("%d is being popped from Stack 1\n", popped\_value);

}

else

{

printf ("Stack Empty! Cannot Pop\n");

}

}

void pop\_stack2 ()

{

if (top2 < SIZE)

{

int popped\_value = ar[top2++];

printf ("%d is being popped from Stack 2\n", popped\_value);

}

else

{

printf ("Stack Empty! Cannot Pop\n");

}

}

//Functions to Print Stack 1 and Stack 2

void print\_stack1 ()

{

int i;

for (i = top1; i >= 0; --i)

{

printf ("%d ", ar[i]);

}

printf ("\n");

}

void print\_stack2 ()

{

int i;

for (i = top2; i < SIZE; ++i)

{

printf ("%d ", ar[i]);

}

printf ("\n");

}

int main()

{

int ar[SIZE];

int i;

int num\_of\_ele;

printf ("We can push a total of 10 values\n");

//Number of elements pushed in stack 1 is 6

//Number of elements pushed in stack 2 is 4

for (i = 1; i <= 6; ++i)

{

push\_stack1 (i);

printf ("Value Pushed in Stack 1 is %d\n", i);

}

for (i = 1; i <= 4; ++i)

{

push\_stack2 (i);

printf ("Value Pushed in Stack 2 is %d\n", i);

}

//Print Both Stacks

print\_stack1 ();

print\_stack2 ();

//Pushing on Stack Full

printf ("Pushing Value in Stack 1 is %d\n", 11);

push\_stack1 (11);

//Popping All Elements From Stack 1

num\_of\_ele = top1 + 1;

while (num\_of\_ele)

{

pop\_stack1 ();

--num\_of\_ele;

}

//Trying to Pop From Empty Stack

pop\_stack1 ();

return 0;

}

