**UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA**

**FACULTAD DE CIENCIAS QUIMICAS EN INGENIERIA**

**Algoritmos y estructura de datos**

Practica 6. Pilas estáticas

**Alumno:** Caudillo Sánchez Diego

**Matricula**: 1249199

**Grupo:** 551

**Docente:** Alma Leticia Palacio Guerrero

**Fecha de entrega:** 05/Abril/2019

**Introducción**

Las pilas son un Tipo de Dato Abstracto (TAD) que sirven como una colección de elementos, con dos funciones principales:

* Push: agrega un elemento a la colección de datos.
* Pop: remueve el dato que haya ingresado recientemente.

El orden en el cual los elementos salen de una pila, recibe el nombre de LIFO (Last In First Out), que se refiere al último elemento que entra será el primero en salir.

Si consideramos a la pila como una estructura de datos linear o más abstracto, una colección secuencial, la operación de *push* y *pop* solo ocurren en un extremo de la estructura, a la cual se refiere como el *tope de pila.* Esto hace posible implementar una pila como una simple lista enlazada y un apuntador al tope de pila. Una pila puede ser implementada para obtener una capacidad encerrada. Si la pila está llena y ya no contiene espacio suficiente para que a una entidad se le haga un push (insertar dato a la pila), a la pila se le considera en un estado de *desbordamiento (overflow).* La operación pop remueve un dato del tope de pila.

**Competencia**

Comprender el principio LIFO mediante el diseño e implementación de las funciones básicas de entrada y salida de datos en una estructura pila, para generar soluciones creativas a problemas de ingeniería con creatividad y responsabilidad.

**Problema**

Suponga que unos libros están organizados en dos pilas ordenadas ascendentemente por título. Elabore un programa que fusione ambas pilas en una tercera ordenada descendentemente. NOTA: no debe utilizar más de 3 pilas, pero si puede utilizar como base las funciones y métodos de pila vistos en clase. Por ejemplo, si la pila 1 contiene los títulos:

* Del amor y otros demonios
* El Perfume
* Marianela

y la pila 2 contiene:

* Casa de campo
* Las batallas en el desierto
* Mujeres de ojos grandes
* Rayuela

La pila resultante debe ser:

* Casa de campo
* Del amor y otros demonios
* El perfume
* Las batallas en el desierto
* Marianela
* Mujeres de ojos grandes
* Rayuela

**Código**

*/\**

*Elabore un programa que fusione ambas pilas en una tercera ordenada*

*descendentemente. NOTA: no debe utilizar más de 3 pilas, pero si puede*

*utilizar como base las funciones y métodos de pila vistos en clase.*

*Por ejemplo, si la pila 1 contiene los títulos:*

*♣ Del amor y otros demonios,*

*♣ El Perfume,*

*♣ Marianela*

*y la pila 2 contiene*

*♣ Casa de campo,*

*♣ Las batallas en el desierto,*

*♣ Mujeres de ojos grandes,*

*♣ Rayuela*

*la pila resultante debe ser*

*• Casa de campo*

*• Del amor y otros demonios*

*• El perfume*

*• Las batallas en el desierto*

*• Marianela*

*• Mujeres de ojos grandes*

*• Rayuela*

*\*/*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

*/\*Headers\*/*

void printData(char stack[][30]);

int push (char stack[][30], int\* sp, char\*data);

int pop (char stack[][30], int \*sp, char \*data);

void clrRow(char stack[][30], int sp);

void clrData(char\* data);

int main(int argc, char const \*argv[])

{

*/\*Declaración de pilas\*/*

char stack1[5][30] = {"Del amor y otros demonios", "El perfume", "Marianela"};

char stack2[5][30] = {"Casa de campo", "Las botellas en el desierto", "Mujeres de ojos grandes", "Rayuela"};

char stack3[10][30] = {0};

char data[30] = {0};

int sp\_s1 = 0, sp\_s2 = 0, sp\_s3 = 9, ov; *//Stack Pointer*

if((ov =pop(stack2, &sp\_s2, data)) == -1) exit(-1);

printf("Dato que sale de la pila 2: %s\tOverflow: %d\tStack Pointer: %d\n\n", data, ov, sp\_s2);

push(stack3, &sp\_s3, data);

if((ov =pop(stack1, &sp\_s1, data)) == -1) exit(-1);

printf("Dato que sale de la pila 1: %s\tOverflow: %d\tStack Pointer: %d\n\n", data, ov, sp\_s1);

push(stack3, &sp\_s3, data);

if((ov =pop(stack1, &sp\_s1, data)) == -1) exit(-1);

printf("Dato que sale de la pila 1: %s\tOverflow: %d\tStack Pointer: %d\n\n", data, ov, sp\_s1);

push(stack3, &sp\_s3, data);

if((ov =pop(stack2, &sp\_s2, data)) == -1) exit(-1);

printf("Dato que sale de la pila 2: %s\tOverflow: %d\tStack Pointer: %d\n\n", data, ov, sp\_s2);

push(stack3, &sp\_s3, data);

if((ov =pop(stack1, &sp\_s1, data)) == -1) exit(-1);

printf("Dato que sale de la pila 1: %s\tOverflow: %d\tStack Pointer: %d\n\n", data, ov, sp\_s1);

push(stack3, &sp\_s3, data);

if((ov =pop(stack2, &sp\_s2, data)) == -1) exit(-1);

printf("Dato que sale de la pila 2: %s\tOverflow: %d\tStack Pointer: %d\n\n", data, ov, sp\_s2);

push(stack3, &sp\_s3, data);

if((ov =pop(stack2, &sp\_s2, data)) == -1) exit(-1);

printf("Dato que sale de la pila 2: %s\tOverflow: %d\tStack Pointer: %d\n\n", data, ov, sp\_s2);

push(stack3, &sp\_s3, data);

puts("\*\*\* Pila resultante \*\*\*");

printData(stack3);

return 0;

}

*/\**

*Descripción*

*Función que muestra los datos dentro de una pila, la función recibe*

*como parámetro la pila que se desea mostrar sus datos. La función no*

*devuelve ningún valor.*

*Parámetros*

*- stack: Pila la cual se quiere mostrar sus datos.*

*\*/*

void printData(char stack[][30])

{

for(int i = 9; i >= 0; i--)

{

for(int j = 0; stack[i][j] != 0; j++)

{

printf("%c", stack[i][j]);

}

printf("\n");

if(stack[i][0] == 0) break;

}

}

*/\**

*Descripción:*

*Función que mete un dato hacia a la pila siguiendo el formato de LIFO.*

*la función devuelve un -1 si el STACK POINTER excede el valor de los datos que puede*

*almacenar indicando que existe un OVERFLOW o desbordamiento. Caso contrario devuelve*

*un 1 indicando que el dato se ha almacenado de manera correctamente.*

*Parámetros*

*-stack: pila donde se va almacenar el dato*

*-sp: sp (stack pointer) es la posición donde se encuentra actualmente*

*apuntando la pila.*

*-data: arreglo del cual el dato va ser extraído para ser insertado en la pila.*

*\*/*

int push(char stack[][30], int\* sp, char\* data)

{

int i = 0;

if(\*sp < 0) return -1; *// underflow*

while(data[i] != 0){

stack[\*sp][i] = data[i];

i++;

} \*sp-=1;

clrData(data);

return 0; *// Push completed!*

}

*/\**

*Descripción:*

*Función que remueve un dato de la pila siguiendo el formato de LIFO*

*el cual consiste en el último que entra va ser el primer dato en salir. SP se aumenta*

*para pasar al siguiente dato de la pila.*

*La función devuelve un -1 indicando que ha habido un UNDERFLOW el cual se da*

*cuando el STACK POINTER ha llegado a un valor menor que 0. Se devuelve un 1 cuando*

*se haya removido un dato de manera exitosa.*

*Parámetros*

*-stack: pila de la cual va ser removido el dato indicado por SP.*

*-sp: sp (stack pointer) es la posición donde se encuentra actualmente*

*apuntando la pila.*

*-data: es un arreglo donde el dato extraído de la pila va ser almacenado.*

*\*/*

int pop(char stack[][30], int\* sp, char\* data)

{

int i = 0;

if(\*sp >= 10) return -1; *//Overflow*

while(stack[\*sp][i] != 0)*/\*se reemplaza por 0 la palabra que ha salido de la pila\*/*

{

data[i] = stack[\*sp][i];

i++;

}

clrRow(stack, \*sp);

\*sp+=1;

return 0; *//dato que salió de la pila*

}

*/\**

*Descripción*

*Función que limpia un renglón de la pila indicado por sp.*

*Esta función se utiliza para asegure que al remover un*

*dato con la instrucción pop haya sido limpiado todo el*

*dato completo reemplazándolos por 0.*

*Parámetros*

*-stack: pila a la cual se le borra un dato*

*-sp: apuntador de pila que posición va ser limpiado.*

*\*/*

void clrRow(char stack[][30], int sp)

{

int i = 0;

while(stack[sp][i] != 0)

{

stack[sp][i] = 0;

i++;

}

}

void clrData(char\* data)

{

int i =0 ;

while(data[i] != 0){

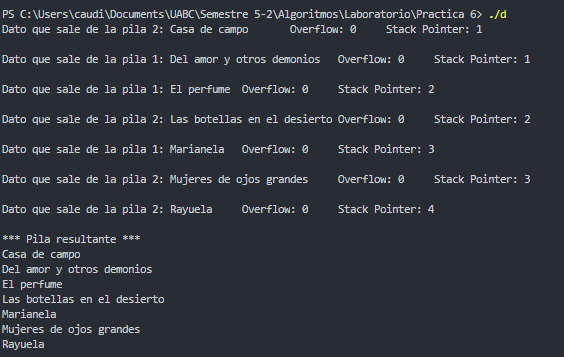
data[i] = 0;

i++;

}

}

**Evidencia de ejecución**



**Conclusión**

Con la resolución de esta práctica se puede ver de qué manera se puede acomodan las pilas y como podemos acomodar de cierta manera en orden. Esto ayuda a complementar la práctica anterior para reforzar los conocimientos. Y al estar usando pilas de dos dimensiones el grado de dificultad se eleva un poco más.

**Bibliografía**

Donal E. Knuth (1997). *The Art of Computer Programming Volume 1. Fundamental Algorithms.* Massachusetts:Addison-Wesley.