UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

Algoritmos y estructura de datos

Practica 5. Pilas estáticas

Alumno: Caudillo Sánchez Diego

Matricula: 1249199

Grupo: 551

Docente: Alma Leticia Palacios Guerrero

Fecha de entrega: 29/Marzo/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 ultimo 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 esta 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

Aplicar eficientemente el principio LIFO para generar soluciones creativas a problemas de ingeniería.

Problema

Un palíndromo es una expresión que se lee de igual manera de izquierda a derecha que de derecha a izquierda. Se pide diseñar e implementar un algoritmo basado en *pilas estáticas* que reciba una y evalúe una expresión dada por el usuario y determine si es un palíndromo o no.

Código

```
/*
Alumno: Caudillo Sanchez Diego
Matricula: 1249199
Grupo: 551
Asignatura: Algoritmos y estructuas de datos
Docente: Alma Leticia Palacios Guerrero
Fecha de entrega: 29-marzo-2019

DESCRIPCION DEL PROGRAMA
Se pide diseñar e implementar un algoritmo basado en pilas estáticas que
reciba una y evalúe una expresión dada por el usuario y determine si es
un palíndromo o no.
*/

/*Librerias utilizadas*/
#include <stdio.h>
#include <time.h>
```

```c
#include <string.h>
#include <ctype.h>

/*Headers*/
void deleteSpace(char* array);
int palindrome(char* array);
void push(char* array, char* stack);
char pop(char* stack, int index);
void printStack(int size, char* stack);

/*Funcion main, donde se ejecutan todas los metodos creados*/
int main(int argc, char const *argv[])
{
    char array[20] = {0}, opc;
    clock_t exe_time;

    while(opc != '0')
    {
        //system("cls");
        printf("Ingrese palabra: ");
        gets(array);
        deleteSpace(array);
        // si devuelve -1 no es palindrome, caso contrario si lo es.
        exe_time = clock();
        if(palindrome(array) == -1) printf("==========================\n==== No
es palindrome ====\n==========================\n");
        else printf("==========================\n===== Es palindrome
=====\n==========================\n");
        exe_time = clock() - exe_time;
        printf("Tiempo de ejecucion %lf segundos\n",
((double)exe_time)/CLOCKS_PER_SEC);
        setbuf(stdin, 0);
        printf("Presiones '0' si quiere salir... ");
        opc = getchar();
        setbuf(stdin, 0);
    }
    return 0;
}
/*
    Descripcion
        Funcion que recorre un vector buscando espacios para borrarlos.
    Parametros
        -array: arreglo de caracteres el cual se van a borrar los espacios.
*/
void deleteSpace(char* array)
```

```
{
    for(int i=0; array[i] != 0; i++ )
    {
        if(array[i] == ' ')
        {
            for(int j = i; array[j] != 0; j++ )
                array[j] = array[j+1];
        }
    }
}
/*
    Descripcion
        Funcion que realiza un push de un vector hacia la pila. La funcion mete
el
        vector entero a la pila.
    Parametros
        -array: es el arreglo el cual se va introducir a la pila.
        -stack: un arreglo el cual se reserva como la pila y donde los datos que
        recibe son del "array"
*/
void push(char* array, char* stack)
{
    //array[position] = 0;
    for(int i=strlen(array)-1, j=0; i >= 0; i--,j++)
        stack[j] = array[i];
}
/*
    Descripcion
        Funcion que realiza un pop a la pila, quiere decir que extrae un dato de
ella.
        la funcion solo va removiendo datos de uno en uno. Devuelve el dato
extraido.
    Parametros
        -stack: array el cual actua como la pila y es la cual de donde se extraen
los
        datos.
        -index: la posicion actual donde se la pila quedo apuntada.
*/
char pop(char* stack, int index)
{
    char data_out;
    data_out = stack[index]; // guarda el dato que salio
    stack[index] = 0; // coloca un cero en la posicion del dato que salio,
indicando
    // que esa posicion se ha vaciado.
```

```
    return data_out; // regresa el dato sacado de la pila.
}
/*
    Descripcion
        Funcion que imprime los datos actuales en la pila.
    Parametros
        -size: entero el cual tiene el tamano de la pila
        -stack: la pila, de la cual sevan a extraer los datos
*/
void printStack(int size, char* stack)
{
    for(int i = 0; i < size; i++)
           printf("|_%c_|\n", stack[i]);
}
/*
    Descripcion
        Funcion que verifica un arreglo si son palindromes con la ayuda de pila.
        Primeramente el arreglo se carga en la pila (LIFO) y se van comparando
        los caracteres de la pila y del arreglo, si todos coinciden, quiere decir
        que la palabra es palindrome y se retorna un valor de 1, caso contrario
se
        devuelve un -1 indicando que la palabra o frase no es palindrome.
    Parametros
        -array: arreglo que contiene la palabra o frase que se quiere verificar
si
        es palindromo o no.
*/
int palindrome(char* array)
{
    char stack[20], letra;
    int index=0;
    push(array,stack); // se realiza un push del arreglo.
    puts("STACK");
    while(index <= strlen(array))
    {
        printStack(strlen(array),stack); // impresion del stack con datos
actualizados
        if(index < strlen(array))
        {
            printf("\n==>Actualizado<==\n");
            letra = pop(stack, index); // letra contiene el dato de cuando se
realiza un pop.
            printf("Dato que salio: [%c]\n", letra);
            // Si el dato extraido de la pila no coincide con el dato del arreglo
```

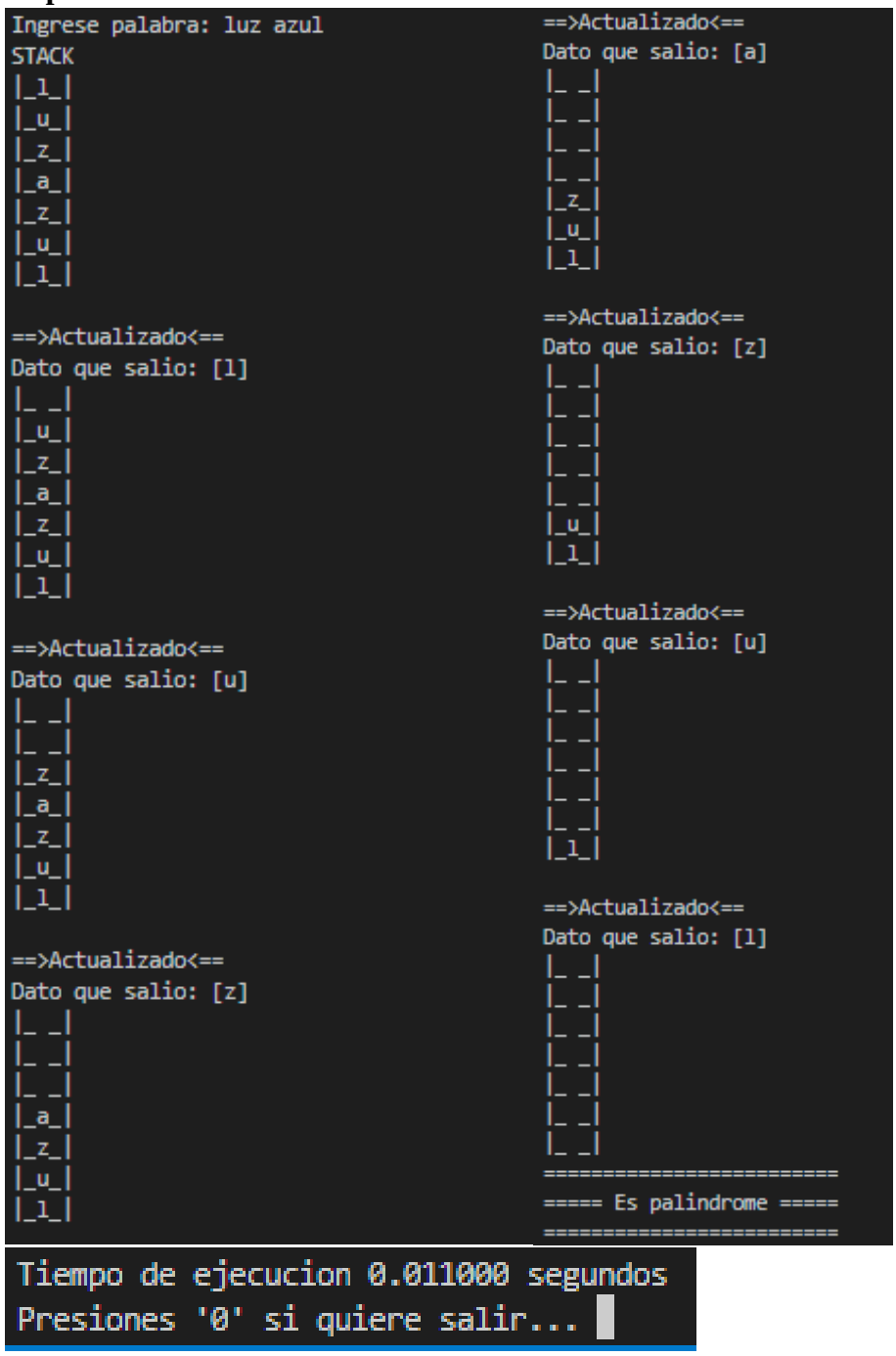
```
            // entonces se devuelve un -1 indicando que la palabra no es
palindrome.
            if(toupper(letra) != toupper(array[index]))
            {
                printf("[%c] es diferente de [%c]\n", letra, array[index]);
                return -1;
            }
        }index++;
    }return 1; // si al recorrer todo el arreglo y las letras coinciden, entonces
la palabra en palindrome.
}
```

Desarrollo

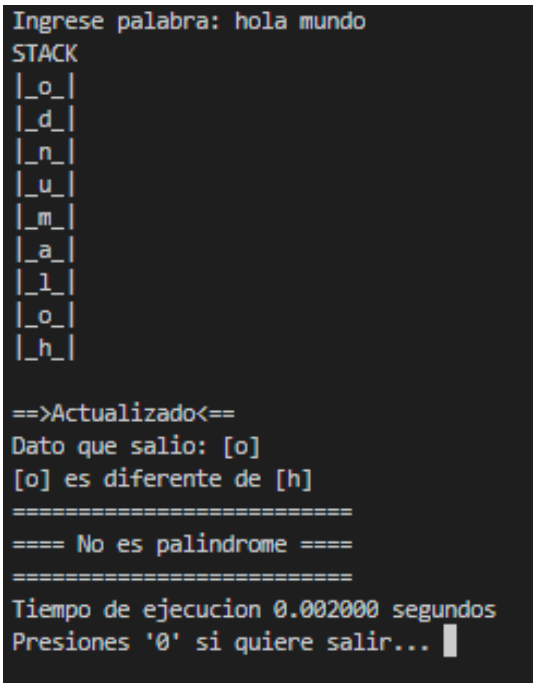
- La frase debe empezar a evaluarse después de haber sido capturada.
- El programa debe mostrar el estado de la(s) pila(s) durante el proceso de evaluación de cada carácter de la expresión.
- El programa debe presentar cada pareja de caracteres que se esté comparando durante el proceso de evaluación de la expresión.
- El tamaño máximo de la expresión será preestablecido por el programador, pero el usuario podrá introducir expresiones de menor tamaño.
- El lenguaje de programación a utilizar es libre, C o Java.
- El programa debe repetirse N veces, hasta que el usuario elija la opción de salida.
- Deben eliminarse los espacios de las frases.
- Las expresiones deben tener al menos un carácter.
- La interfaz del programa debe estar bien organizada.

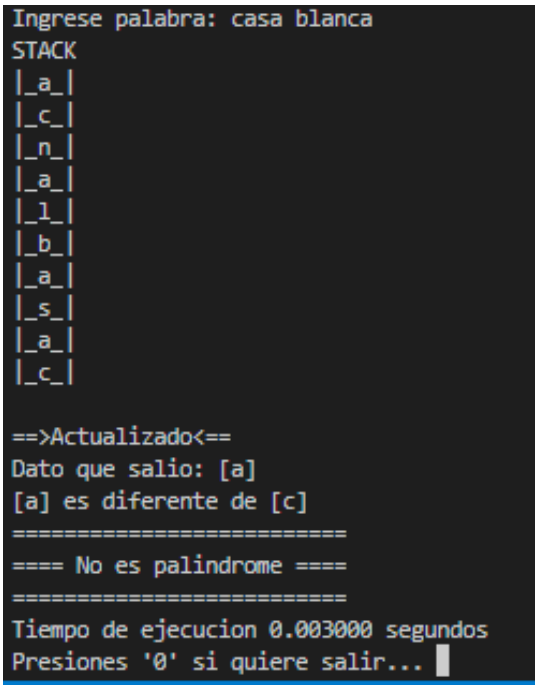
Evidencia de ejecución

- **Es palíndromo**

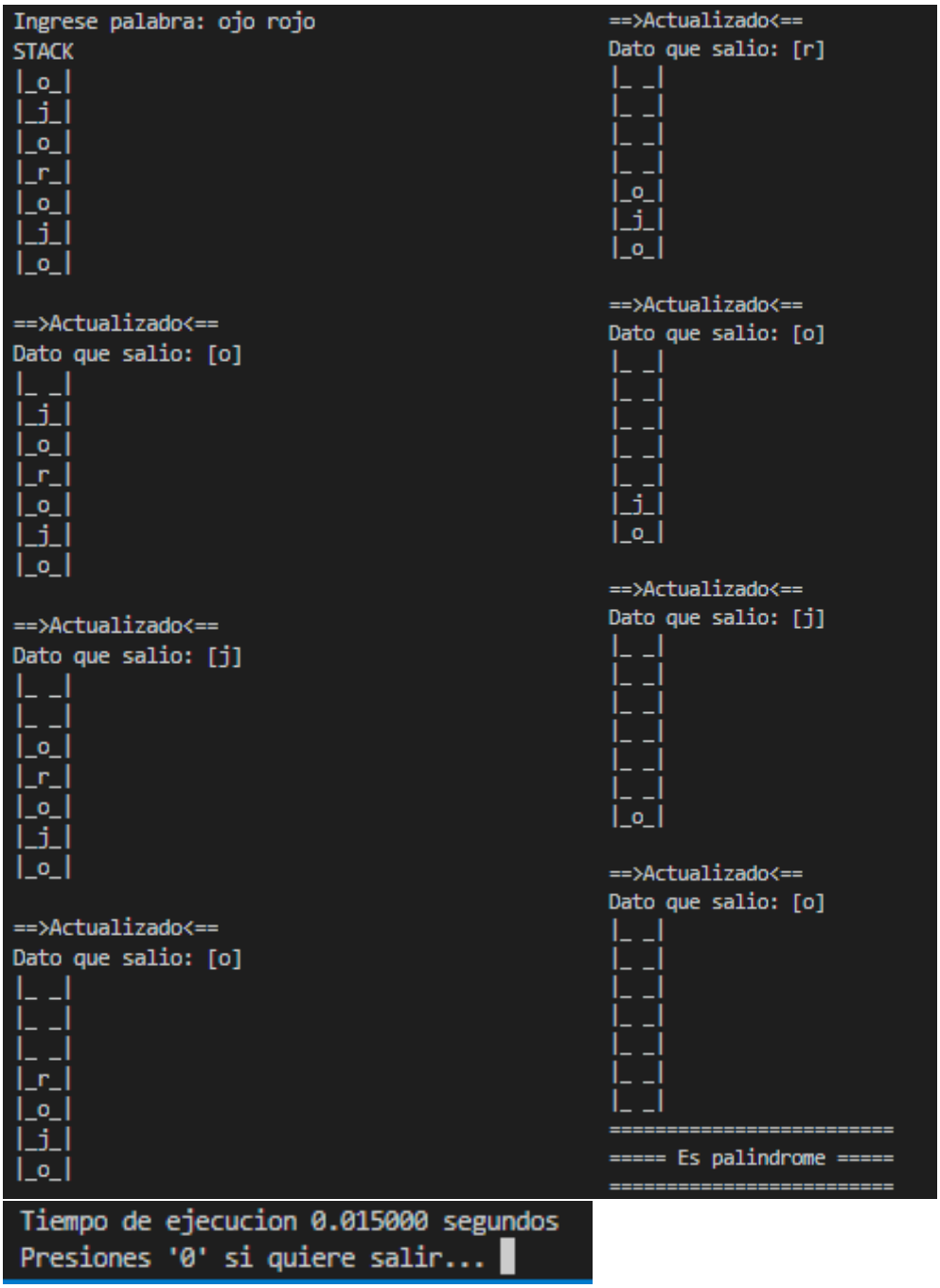


- **No es palíndromo**





Algunos caracteres son iguales



Conclusión

Con la realización de la practica se pudo observar de qué manera funciona la pila. Nos ayuda a comprender mejor en como esta estructurada. Con el ejercicio pudimos ver una de las muchas aplicaciones que se pueden hacer con la pila.

Entre mas grande fuera la palabra, mayor es el crecimiento del tiempo de ejecución. Y si las letras se repiten, parece ser que el tiempo de ejecución es menor. Por otra parte, cuando la palabra no es palíndroma el tiempo de ejecución es mucho menor, porque el programa se trunca y ya no sigue la ejecución.

Bibliografía

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