INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

TEORÍA COMPUTACIONAL

PROFA. LUZ MARÍA GARCÍA SÁNCHEZ

AUTÓMATA DE PILA

PRÁCTICA 6

2CM4

COLABORADORES:

OLEDO ENRIQUEZ GILBERTO IRVING

NO. BOLETA 2014170825

RAYA TOLENTINO PAOLA

NO. BOLETA 2017630182

Introducción

El propósito de esta práctica es implementar un programa que represente como es que funciona el autómata de pila, que es un modelo matemático de un sistema que recibe una cadena constituida por símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce.

El lenguaje que reconoce un autómata con pila pertenece al grupo de los lenguajes libres de contexto en la clasificación de jerarquía de Chomski.

En esta práctica se implementará el Autómata Finito con Pila Determinista (AFPD), con el objetivo de verificar si determinadas cadenas pertenecen al lenguaje generado por la GLC que representa al autómata.

Definición de AFPD

El AFPD es el modelo de autómata requerido para aceptar los lenguajes libres de contexto, que a diferencia de un AFD normal, solo acepta lenguajes regulares.

Un AFPD es una tupla de 7 elementos: $M = (Q, q0, F, \Sigma, \Gamma, z0, \delta)$, donde:

- Q es el conjunto finito de estados.
- $q0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto no vacío de estados finales o de aceptación.
- Σ es el alfabeto finito de entrada, también llamado alfabeto de cinta.
- Γ es el alfabeto finito de pila.
- $z0 \in \Gamma$ es el símbolo inicial de la pila o el marcador de fondo, donde $z0$ no pertenece a Σ.
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma *$ es la función de transición del autómata. Es de la forma $\delta(q, a, s) = (q\ 0\ , \alpha)$.

Descripción

Un AFPD procesa cadenas sobre una cinta de entrada (la cadena de entrada) justo como en un AFD, pero hay una cinta adicional (la pila) que es utilizada por el autómata como lugar de almacenamiento. En un momento determinado estando en un estado q, se escanea un símbolo a sobre la cinta de entrada y el símbolo s en el tope de la pila. De esta forma la transición $\delta(q, a, s) = (q\ 0\ , \alpha)$ representa un paso computacional: pasamos al estado $q0$, extraemos el carácter del tope de la pila e insertamos la cadena α en el tope de la pila, carácter por carácter.

Recordemos que la pila es una estructura de datos del tipo $LIFO$ $(Last\ In\ First\ Out)$, por lo que en todo momento el aut´omata solo tiene acceso al s´ımbolo que est´a en el tope de la pila, y el contenido de la pila siempre se lee de arriba (tope) hacia abajo (fondo). También, por definición de δ, nunca podemos realizar alguna transición si la pila est´a vacía.

En caso de que alguna transición $\delta(q, a, s)$ no exista o no está definida, se aborta el procesamiento de la cadena de entrada, tal y como ocurr´ıa en el AFD.

Algunos casos especiales de transiciones son:

- $\delta(q, a, s) = (q\ 0\ , s)$: el contenido de la pila no se altera, pues luego de borrar el símbolo s lo volvemos a insertar.
- $\delta(q, a, s) = (q\ 0\ , \varepsilon)$: se borra el símbolo del tope de la pila y en el siguiente paso se escanea el nuevo tope de la pila, que es el símbolo colocado justo debajo de s.
- $\delta(q, \varepsilon, s) = (q\ 0\ , \alpha)$: esta es una ε-transición o transición espontanea, en donde no procesamos el símbolo actual de la cadena de entrada pero en la pila borramos s e insertamos la nueva cadena α. De esta forma podemos modificar el contenido de la pila sin consumir símbolos de la cadena de entrada. Notemos que para que el autómata sea determinista, no podemos tener al mismo tiempo las transiciones $\delta(q, a, s)\ y\ \delta(q, \varepsilon, s)$.

Planteamiento del problema

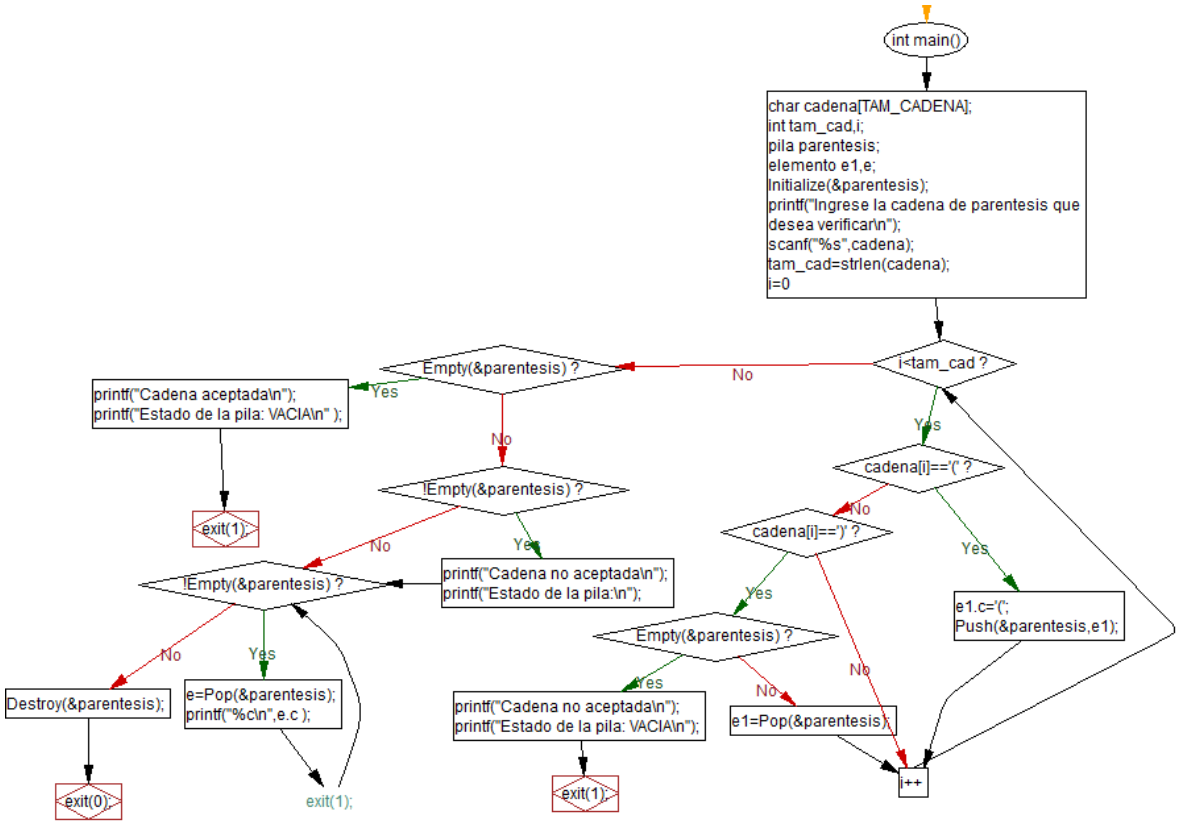
Construir un programa en ANSI C capaz de recibir como entrada una cadena que pertenezca al lenguaje del autómata de pila, como lo puede ser el balanceo de paréntesis.

Se debe de mostrar el contenido de la pila después de haber leído una decana de entrada, ya sea que la cadena haya sido aceptada o rechazada.

Se hizo uso del TAD Pila ya anteriormente implementado es por eso que aunque el la solución haya sido implementada en C podemos hacer uso de las siguientes funciones:

```
void Initialize(pila *s);  //Inicializar pila (Iniciar una pila para su uso)

void Push(pila *s, elemento e); //Empilar (Introducir un elemento a la pila)

elemento Pop (pila *s);    //Desempilar (Extraer un elemento de la pila)

boolean Empty(pila *s);    //Vacia (Preguntar si la pila esta vacia)

elemento Top(pila *s);     //Tope (Obtener el "elemento" del tope de la pila si extraerlo de la pila)

int Size(pila *s);   //Tamaño de la pila (Obtener el número de elementos en la pila)

void Destroy(pila *s);     //Elimina pila (Borra a todos los elementos y a la pila de memoria)
```

Diseño de la solución



Ejemplo del funcionamiento del Autómata de Pila con paréntesis correctos:

(A+B)*(C/D)

PILA1
A+B)*(C/D)
Paréntesis que abre
Push: (
(

PILA1
A+B*(C/D)
Paréntesis que cierra
Pop: (

PILA1
A+B*C/D)
Paréntesis que abre
Push: (
(

PILA1
A+B*C/D)
Paréntesis que cierra
Pop: (

¿La pila1 está *Vacía*? Si
Paréntesis Ok

Ejemplo del funcionamiento del Autómata de Pila con paréntesis incorrectos:

(((A+B)*(C/D)

	PILA1
	((A+B)*(C/D)
	Paréntesis que abre
	Push: (
(	

	PILA1
	(A+B)*(C/D)
	Paréntesis que abre
(	
(	Push: (

	PILA1
	A+B)*(C/D)
(	Paréntesis que abre
(	
(	Push: (

	PILA1
	A+B*C/D)
	Paréntesis que cierra
(	
(	Pop: (

	PILA1
	A+B*C/D)
(	Paréntesis que abre
(	
(	Push: (

	PILA1
	A+B*C/D
	Paréntesis que cierra
(	
(	Pop: (

¿La pila1 está *Vacía*? No
Paréntesis Incorrectos

Implementación de la solución

Programa principal

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "TADPilaDin.h"
#define TAM_CADENA 100
#include <math.h>

int main(){
      char cadena[TAM_CADENA];
      int tam_cad,i;

      pila parentesis;
      elemento e1,e;
      Initialize(&parentesis);

      printf("Ingrese la cadena de parentesis que desea
verificar\n");
      scanf("%s",cadena);

      tam_cad=strlen(cadena);
```

```
	for(i=0;i<tam_cad;i++)
	{

		if(cadena[i]=='(')
			{
				e1.c='(';
				Push(&parentesis,e1);
			}


		else if(cadena[i]==')')
			{

				if(Empty(&parentesis))
				{
					printf("Cadena no aceptada\n");
					printf("Estado de la pila: VACIA\n");
					exit(1);
				}
				e1=Pop(&parentesis);
			}

	}

	if (Empty(&parentesis)){
		printf("Cadena aceptada\n");
		printf("Estado de la pila: VACIA\n" );
		exit(1);
	}


	if(!Empty(&parentesis))
	{
		printf("Cadena no aceptada\n");
		printf("Estado de la pila:\n");
	}
	while(!Empty(&parentesis)){
		e=Pop(&parentesis);
		printf("%c\n",e.c );
		//exit(1);
	}


	Destroy(&parentesis);
	exit(0);
}
```

TAD Pila.c

```c
#include <stdlib.h>
#include "TADPilaDin.h"

//DEFINICIÓN DE FUNCIONES

/*
void Initialize(pila *s);
Descripción: Inicializar pila (Iniciar una pila para su uso)
Recibe: int *s (Referencia a la pila "s" a operar)
Devuelve:
Observaciones: El usuario a creado una pila y s tiene la
referencia a ella,
si esto no ha pasado se ocasionara un error.
*/
void Initialize(pila *s)
{
	s->tope=NULL; //(*s).tope=NULL;
	return;
}

/*
void Push(pila *s, elemento e);
Descripción: Empilar (Introducir un elemento a la pila)
Recibe: int *s (Referencia a la pila "s" a operar), elemento e
(Elemento a introducir en la pila)
Devuelve:
Observaciones: El usuario a creado una pila y s tiene la
referencia a ella, s ya ha sido inicializada.
Ademas no se valida si el malloc() pudo o no apartar memoria, se
idealiza que siempre funciona bien
y no se acaba la memoria
*/
void Push(pila *s, elemento e)
{
	nodo *aux;
	aux=malloc(sizeof(nodo));
	(*aux).e=e;  //aux->e=e;
	aux->abajo=s->tope;
	s->tope=aux;
	return;
}


/*
void Pop(pila *s);
```

```
Descripción: Desempilar (Extraer un elemento de la pila)
Recibe: int *s (Referencia a la pila "s" a operar)
Devuelve: elemento (Elemento e extraido de la pila)
Observaciones: El usuario a creado una pila y s tiene la
referencia a ella, s ya ha sido inicializada.
Ademas no se valida si la pila esta vacia antes de desempilar
(causa error desempilar si esta esta vacía),
tampoco se valida si free() pudo o no liberar la memoria, se
idealiza que siempre funciona bien
*/
elemento Pop (pila *s)
{
	elemento r;
	nodo *aux;
	r=s->tope->e;
	aux=s->tope;
	s->tope=s->tope->abajo;
	free(aux);
	return r;
}

/*
boolean Empty(pila *s);
Descripción: //Vacia (Preguntar si la pila esta vacia)
Recibe: int *s (Referencia a la pila "s" a operar)
Devuelve: boolean (TRUE o FALSE según sea el caso)
Observaciones: El usuario a creado una pila y s tiene la
referencia a ella, s ya ha sido inicializada.
*/
boolean Empty(pila *s)
{
	boolean r;
	if(s->tope==NULL)
	{
		r=TRUE;
	}
	else
	{
		r=FALSE;
	}
	return r;
}

/*
elemento Top(pila *s);
```

```
Descripción: Tope (Obtener el "elemento" del tope de la pila si
extraerlo de la pila)
Recibe: int *s (Referencia a la pila "s" a operar)
Devuelve: elemento (Elemento del tope de la pila)
Observaciones: El usuario a creado una pila y s tiene la
referencia a ella, s ya ha sido inicializada.
Ademas no se valida si la pila esta vacia antes de consultar al
elemnto del tope (causa error si esta esta vacía).
*/
elemento Top(pila *s)
{
	return s->tope->e;
}

/*
int Size(pila *s);
Descripción: Tamaño de la pila (Obtener el número de elementos en
la pila)
Recibe: int *s (Referencia a la pila "s" a operar)
Devuelve: int (Tamaño de la pila -1->Vacia, 1->1 elemento, 2->2
elementos, ...)
Observaciones: El usuario a creado una pila y s tiene la
referencia a ella, s ya ha sido inicializada.
*/
int Size(pila *s)
{
	nodo *aux;
	int tam_pila=0;
	aux=s->tope;
	if(aux!=NULL)
	{
		tam_pila++;
		while(aux->abajo!=NULL)
		{
			tam_pila++;
			aux=aux->abajo;
		}
	}
	return tam_pila;
}

/*
void Destroy(pila *s);
Descripción: Elimina pila (Borra a todos los elementos en a
la pila de memoria)
Recibe: int *s (Referencia a la pila "s" a operar)
```

```
Devuelve:
Observaciones: El usuario a creado una pila y s tiene la
referencia a ella, s ya ha sido inicializada.
*/
void Destroy(pila *s)
{
	nodo *aux;
	if(s->tope!=NULL)
	{
		while(s->tope!=NULL)
		{
			aux=s->tope->abajo;
			free(s->tope);
			s->tope=aux;
		}
	}
	return;
}
```

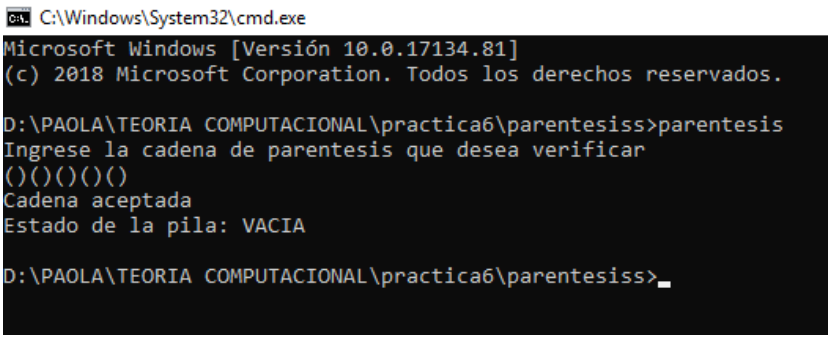
TAD Pila.h

```
//DEFINICIONES DE CONSTANTES
#define TRUE    1
#define FALSE   0

//DEFINICIONES DE TIPOS DE DATO

//Definir un boolean (Se modela con un "char")
typedef unsigned char boolean;

//Definir un elemento (Se modela con una estructura
"elemento")
typedef struct elemento
{
	//Variables de la estructura "elemento" (El usuario
puede modificar)
	char c;
	int q;
	float f;
	//***
	//***
	//***
}elemento;
```

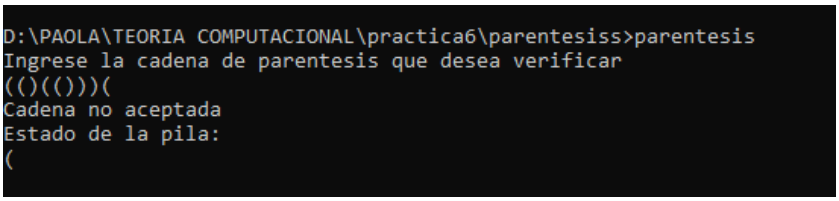
```
//Definir un nodo que será utilizado para almacenar una posición
de la pila (Nodo), lo que incluira a un elemento y a un apuntador
al siguiente nodo
typedef struct nodo
{
      //Elemento a almacenar en cada nodo de la pila
      elemento e;
      //Apuntador al elemento de debajo (Requerido por ser una
implementación dinámica -Usuario: No modificar)
      struct nodo *abajo;
}nodo;

//Definir una pila (Se modela con una estructura que unicamente
incluye un puntero a "elemento")
typedef struct pila
{
      nodo *tope;
}pila;

//DECLARACIÓN DE FUNCIONES
void Initialize(pila *s);            //Inicializar pila (Iniciar
una pila para su uso)
void Push(pila *s, elemento e);      //Empilar (Introducir un
elemento a la pila)
elemento Pop (pila *s);                    //Desempilar (Extraer
un elemento de la pila)
boolean Empty(pila *s);                    //Vacia (Preguntar si
la pila esta vacia)
elemento Top(pila *s);                     //Tope (Obtener el
"elemento" del tope de la pila si extraerlo de la pila)
int Size(pila *s);                         //Tamaño de la pila
(Obtener el número de elementos en la pila)
void Destroy(pila *s);                     //Elimina pila
(Borra a todos los elementos y a la pila de memoria)
```

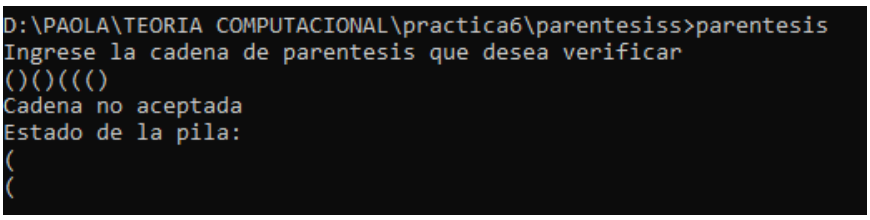
Funcionamiento

Se ingresa una cadena que esta correctamente escrita:



Se ingresan cadenas que no estan correctamente escritas:





Conclusiones

Paola Raya Tolentino:

Con esta práctica entendí mejor como es que funciona un autómata de pila, ya con los conocimientos previos que se tenía de la materia de estructuras de datos pudo ser más fácil al igual que con lo explicado en clase.

Gilberto Irving Oledo Enriquez:

Puedo concluir que desde mi punto de vista fue una práctica sencilla porque anteriormente había trabajado con pilas pero esto mismo permitió que comprendiera rápidamente el tema visto en clase y empleado en esta práctica además se conocer otro uso importante del este TAD.