TEORÍA DE LA COMPUTACIÓN
Prof: Dr. Hiram Calvo
Alumno: José Antonio M Faustinos
COnvertidor de un AP a una CFG

El ejercicio propuesto consiste en que a partir de un autómata de pila, y aplicando el teorema 2.3 del libro de Brookshear, se generen las reglas de una gramática libre de contexto que acepte el mismo lenguaje que acepta el autómata.

Para la implementación de este proceso, tenemos lo siguiente:

- El autómata de pila, se presenta en un archivo de texto *.ap* , el cual contiene la siguiente estructura:

```
inicial=f
aceptación=h
g,c,c;h,/
f,c,/;g,c
g,b,/;g,/
```

  - **inicial:** Identificador del estado inicial del AP.
  - **aceptación:** Identificadores de los estados de aceptación del AP, separados por comas.

Cada línea subsecuente representa una transición del autómata de la forma (p,x,s;q,y) donde:

- p es el estado actual
- x es el caracter que se lee de la cadena
- s es el símbolo que se saca de la pila
- q es el estado destino de la transición
- y es el símbolo que se introduce a la pila

A partir de esto, la información del autómata se organiza en las estructuras siguientes:

```
estados=[]
aceptacion=[]
pila=[]
inicial
transiciones=[]
```

En donde **estados**, **aceptación** y ***pila*** son listas que incluyen los identificadores de todos los estados, los identificadores de los estados de aceptación y los símbolos de la pila respectivamente. (El conjunto total de estados y de símbolos de pila se obtienen de las transiciones del autómata :). ***inicial*** almacena el identificador del estado inicial y ***transiciones*** es una lista de las transiciones del autómata, modeladas a su vez como una lista de 5 elementos [p,x,s,q,y], donde cada elemento es análogo al de la transición (p,x,s;q,y) explicada anteriormente.

La transición λ está representada por el caracter / (diagonal normal), esta es agregada a los símbolos de la pila para considerarla como parte de las reglas de la CFG.

```
pila.append("/")
```

Estas reglas serán almacenadas en una lista.

```
reglasCFG=[]
```

Y serán modeladas a su vez como una lista de dos elementos, donde el primero representa el no terminal a la izquierda de la producción y el segundo el conjunto de terminales y no terminales a la derecha de la producción, por ejemplo:

```
['S', '<f,/,h>']
```

Es importante mencionar que el procedimiento planteado por Brookshear nos indica que los no terminales de la CFG generada están representados por una 3-tupla de la forma `<a,b,c>`, donde a,b y c tomarán valores de los identificadores de estados y símbolos de la pila según el procedimiento. Cada 3-tupla representará un símbolo no terminal a lo largo de toda la CFG (al final de la ejecución, se realiza la sustitución de estas 3 -tuplas por símbolos no terminales usuales).

Las reglas generadas por cada uno de los 4 pasos que indica el teorema son agregados a la lista `reglasCFG`, la cual, contiene al final del paso 4 todas las reglas que conforman a la CFG equivalente al AP introducido.

Adicionalmente, y con el propósito de facilitar la comprensión de la gramática generada y posteriormente ser utilizada para analizar una cadena, se realizó la sustitución de la 3-tupla que representa cada símbolo no terminal por una mayúscula (símbolo no terminal usual. Este procedimiento se ejecuta únicamente cuando la gramática resultante tiene menos de 27 símbolos no terminales.

Ambas gramáticas, la del formato Brookshear y la “reducida” a símbolos no terminales usuales son impresas en pantalla y escritas a un archivo . cfg

Ejemplos de ejecución:

- Usando el archivo “apila1.ap” se realizaron las siguientes pruebas:

apila1.ap

```
inicial=f
aceptacion=h
g,c,c;h,/
f,c,/;g,c
g,b,/;g,/
```

Output de la CFG en formato Brookshear (3-tupla):

```
(base) el_faus@maverick:~/Documents/TeoriaC/ap2cfg$ python ap2cfg.py apila1.ap
S-><f,/,h>
<f,/,f>->/
<g,/,g>->/
<h,/,h>->/
<g,c,f>->c<h,/,f>
<g,c,g>->c<h,/,g>
<g,c,h>->c<h,/,h>
<f,c,f>->c<g,c,f><f,c,f>
<f,c,f>->c<g,c,g><g,c,f>
<f,c,f>->c<g,c,h><h,c,f>
<f,c,g>->c<g,c,f><f,c,g>
<f,c,g>->c<g,c,g><g,c,g>
<f,c,g>->c<g,c,h><h,c,g>
<f,c,h>->c<g,c,f><f,c,h>
<f,c,h>->c<g,c,g><g,c,h>
<f,c,h>->c<g,c,h><h,c,h>
<f,/,f>->c<g,c,f><f,/,f>
<f,/,f>->c<g,c,g><g,/,f>
<f,/,f>->c<g,c,h><h,/,f>
<f,/,g>->c<g,c,f><f,/,g>
<f,/,g>->c<g,c,g><g,/,g>
<f,/,g>->c<g,c,h><h,/,g>
<f,/,h>->c<g,c,f><f,/,h>
<f,/,h>->c<g,c,g><g,/,h>
<f,/,h>->c<g,c,h><h,/,h>
<g,c,f>->b<g,/,f><f,c,f>
<g,c,f>->b<g,/,g><g,c,f>
<g,c,f>->b<g,/,h><h,c,f>
<g,c,g>->b<g,/,f><f,c,g>
<g,c,g>->b<g,/,g><g,c,g>
<g,c,g>->b<g,/,h><h,c,g>
<g,c,h>->b<g,/,f><f,c,h>
<g,c,h>->b<g,/,g><g,c,h>
<g,c,h>->b<g,/,h><h,c,h>
<g,/,f>->b<g,/,f><f,/,f>
<g,/,f>->b<g,/,g><g,/,f>
<g,/,f>->b<g,/,h><h,/,f>
<g,/,g>->b<g,/,f><f,/,g>
<g,/,g>->b<g,/,g><g,/,g>
<g,/,g>->b<g,/,h><h,/,g>
```

Output de la CFG en formato convencional:

S->A
B->/
C->/
D->/
E->cF
G->cH
I->cD
J->cEJ
J->cGE
J->cIK
L->cEL
L->cGG
L->cIM
N->cEN
N->cGI
N->cIO
B->cEB
B->cGP
B->cIF
Q->cEQ
Q->cGC
Q->cIH
A->cEA
A->cGR
A->cID
E->bPJ
E->bCE
E->bRK
G->bPL
G->bCG
G->bRM
I->bPN
I->bCI
I->bRO
P->bPB
P->bCP
P->bRF
C->bPQ
C->bCC
C->bRH
R->bPA
R->bCR
R->bRD

Usando apila2.ap

```
inicial=r
aceptacion=t
r,c,/;s,c
s,g,/;s,g
s,f,g;s,/
s,/,c;t,/
```

Output:

```
(base) el_faus@maverick:~/Documents/TeoriaC/ap2cfg$ python ap2cfg.py apila2.ap
S-><r,/,t>
<r,/,r>->/
<s,/,s>->/
<t,/,t>->/
<s,g,r>->f<s,/,r>
<s,g,s>->f<s,/,s>
<s,g,t>->f<s,/,t>
<s,c,r>->/<t,/,r>
<s,c,s>->/<t,/,s>
<s,c,t>->/<t,/,t>
<r,c,r>->c<s,c,r><r,c,r>
<r,c,r>->c<s,c,s><s,c,r>
<r,c,r>->c<s,c,t><t,c,r>
<r,c,s>->c<s,c,r><r,c,s>
<r,c,s>->c<s,c,s><s,c,s>
<r,c,s>->c<s,c,t><t,c,s>
<r,c,t>->c<s,c,r><r,c,t>
<r,c,t>->c<s,c,s><s,c,t>
<r,c,t>->c<s,c,t><t,c,t>
<r,g,r>->c<s,c,r><r,g,r>
<r,g,r>->c<s,c,s><s,g,r>
<r,g,r>->c<s,c,t><t,g,r>
<r,g,s>->c<s,c,r><r,g,s>
<r,g,s>->c<s,c,s><s,g,s>
<r,g,s>->c<s,c,t><t,g,s>
<r,g,t>->c<s,c,r><r,g,t>
<r,g,t>->c<s,c,s><s,g,t>
<r,g,t>->c<s,c,t><t,g,t>
<r,/,r>->c<s,c,r><r,/,r>
<r,/,r>->c<s,c,s><s,/,r>
<r,/,r>->c<s,c,t><t,/,r>
<r,/,s>->c<s,c,r><r,/,s>
<r,/,s>->c<s,c,s><s,/,s>
<r,/,s>->c<s,c,t><t,/,s>
<r,/,t>->c<s,c,r><r,/,t>
<r,/,t>->c<s,c,s><s,/,t>
```

<r,/,t>->c<s,c,t><t,/,t>
<s,c,r>->g<s,g,r><r,c,r>
<s,c,r>->g<s,g,s><s,c,r>
<s,c,r>->g<s,g,t><t,c,r>
<s,c,s>->g<s,g,r><r,c,s>
<s,c,s>->g<s,g,s><s,c,s>
<s,c,s>->g<s,g,t><t,c,s>
<s,c,t>->g<s,g,r><r,c,t>
<s,c,t>->g<s,g,s><s,c,t>
<s,c,t>->g<s,g,t><t,c,t>
<s,g,r>->g<s,g,r><r,g,r>
<s,g,r>->g<s,g,s><s,g,r>
<s,g,r>->g<s,g,t><t,g,r>
<s,g,s>->g<s,g,r><r,g,s>
<s,g,s>->g<s,g,s><s,g,s>
<s,g,s>->g<s,g,t><t,g,s>
<s,g,t>->g<s,g,r><r,g,t>
<s,g,t>->g<s,g,s><s,g,t>
<s,g,t>->g<s,g,t><t,g,t>
<s,/,r>->g<s,g,r><r,/,r>
<s,/,r>->g<s,g,s><s,/,r>
<s,/,r>->g<s,g,t><t,/,r>
<s,/,s>->g<s,g,r><r,/,s>
<s,/,s>->g<s,g,s><s,/,s>
<s,/,s>->g<s,g,t><t,/,s>
<s,/,t>->g<s,g,r><r,/,t>
<s,/,t>->g<s,g,s><s,/,t>
<s,/,t>->g<s,g,t><t,/,t>

Usando apila5.ap

apila5.ap

```
inicial=a
aceptacion=b
a,x,/;a,x
a,y,x;b,/
b,y,x;b,/
```

Output:

```
(base) el_faus@maverick ~/Documents/TeoriaC/ap2cfg$ python ap2cfg.py apila5.ap
S-><a,/,b>
<a,/,a>->/
<b,/,b>->/
<a,x,a>->y<b,/,a>
<a,x,b>->y<b,/,b>
<b,x,a>->y<b,/,a>
<b,x,b>->y<b,/,b>
<a,x,a>->x<a,x,a><a,x,a>
<a,x,a>->x<a,x,b><b,x,a>
<a,x,b>->x<a,x,a><a,x,b>
<a,x,b>->x<a,x,b><b,x,b>
<a,/,a>->x<a,x,a><a,/,a>
<a,/,a>->x<a,x,b><b,/,a>
<a,/,b>->x<a,x,a><a,/,b>
<a,/,b>->x<a,x,b><b,/,b>
S->A
B->/
C->/
D->yE
F->yC
G->yE
H->yC
D->xDD
D->xFG
F->xDF
F->xFH
B->xDB
B->xFE
A->xDA
A->xFC
```