COMPILADORES
REPORTE #2
HOC2 -- Canvas de figuras

1 de Noviembre del 2018

PROFESOR: TECLA PARRA ROBERTO

ALUMNO: SALDAÑA AGUILAR ANDRÉS

GRUPO: 3CM8

HOC2 tiene como propósito hacer una programa gráfico ocupando Java Swing, además del uso de BYACCJ y LEX para la creación del analizador sintáctico y léxico respectivamente.

Yacc
Es un programa para generar analizadores sintácticos. Las siglas del nombre significan Yet Another Compiler-Compiler, es decir, "Otro generador de compiladores más". Genera un analizador sintáctico (la parte de un compilador que comprueba que la estructura del código fuente se ajusta a la especificación sintáctica del lenguaje) basado en una gramática analítica escrita en una notación similar a la BNF. Yacc genera el código para el analizador sintáctico en el Lenguaje de programación C, pero en este caso BYACCJ es para lenguaje Java.

Flex
Lex es un programa para generar analizadores léxicos (en inglés scanners o lexers). Lex se utiliza comúnmente con el programa yacc que se utiliza para generar análisis sintáctico. Lex, escrito originalmente por Eric Schmidt y Mike Lesk, es el analizador léxico estándar en los sistemas Unix, y se incluye en el estándar de POSIX. Lex toma como entrada una especificación de analizador léxico y devuelve como salida el código fuente implementando el analizador léxico en C.

Desarrollo

En este caso, nos apoyamos en un código ya existente para su creación, se trata de un programa que dibuja círculos, cuadrados y líneas pero tomando como punto inicial el origen (0,0).

El objetivo de esta práctica es, modificar el código del HOC2 para que pueda dibujar estas figuras en cualquier punto del canvas.

En la gramática, agregue en la creación de las figuras más parámetros de números para que, se pueda conocer la coordenada (X,Y) de inicio y la coordenada (X,Y) de fin para el dibujo de la figura, además de agregar como acción gramatical, la llamada al método que guarda en la pila valores (“Constpush”), y el valor que se va a ingresar en ese orden.

```
%token NUMBER LINE CIRCULO RECTANGULO COLOR PRINT IMAGEN FILTRO
%start list
%%
list :
    | list ';'
    | list inst ';'   {
            maq.code("print"); maq.code("STOP"); return 1 ;
    }
    ;
inst:  NUMBER  { ((Algo)$$.obj).inst=maq.code("constpush");
```

```
            maq.code(((Algo)$1.obj).simb); }
    | RECTANGULO NUMBER NUMBER NUMBER NUMBER{
            maq.code("constpush");
            maq.code(((Algo)$2.obj).simb);
            maq.code("constpush");
            maq.code(((Algo)$3.obj).simb);
            maq.code("constpush");
            maq.code(((Algo)$4.obj).simb);
            maq.code("constpush");
            maq.code(((Algo)$5.obj).simb);
            maq.code("rectangulo"); }
    | LINE NUMBER NUMBER NUMBER NUMBER {
            maq.code("constpush");
            maq.code(((Algo)$2.obj).simb);
            maq.code("constpush");
            maq.code(((Algo)$3.obj).simb);
            maq.code("constpush");
            maq.code(((Algo)$4.obj).simb);
            maq.code("constpush");
            maq.code(((Algo)$5.obj).simb);
            maq.code("line");}
    | CIRCULO NUMBER NUMBER NUMBER {
            maq.code("constpush");
            maq.code(((Algo)$2.obj).simb);
            maq.code("constpush");
            maq.code(((Algo)$3.obj).simb);
            maq.code("constpush");
            maq.code(((Algo)$4.obj).simb);
            maq.code("circulo");}
    | COLOR NUMBER { maq.code("constpush");
            maq.code(((Algo)$2.obj).simb); maq.code("color");}
    ;
%%
```

En el código de máquina, la única modificación que debemos hacer es conseguir los parámetros para la creación de las figuras por medio de la pila, donde sacamos cada parámetro, teniendo en cuenta que el primer elemento en salir fue el último en ingresar, por lo tanto los parámetros se van consiguiendo del último al primero, para después castear el objeto al tipo de dato necesario, en este caso Double.

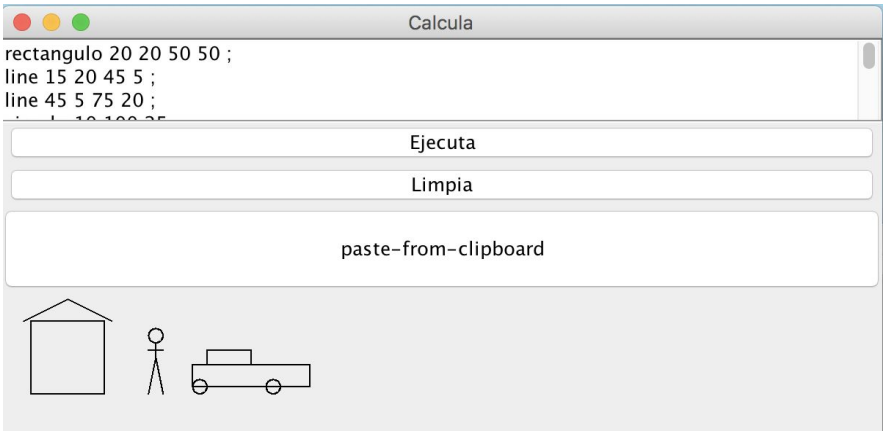
```
void line(){
    double d1=0,d2=0,d3=0,d4=0;
    d4=((Double)pila.pop()).doubleValue();
    d3=((Double)pila.pop()).doubleValue();
    d2=((Double)pila.pop()).doubleValue();
    d1=((Double)pila.pop()).doubleValue();

    if(g!=null){
      (new Linea((int)d1,(int)d2,(int)(d3),(int)(d4))).dibuja(g);
```

```
    }
  }
  void circulo(){
    double d1=0,d2=0,d3=0;
    d3=((Double)pila.pop()).doubleValue();
    d2=((Double)pila.pop()).doubleValue();
    d1=((Double)pila.pop()).doubleValue();

    if(g!=null){
      (new Circulo((int)d2, (int)d3, (int)d1)).dibuja(g);
    }
  }
  void rectangulo(){
   double d1=0,d2=0,d3=0,d4=0;
   d4=((Double)pila.pop()).doubleValue();
   d3=((Double)pila.pop()).doubleValue();
   d2=((Double)pila.pop()).doubleValue();
   d1=((Double)pila.pop()).doubleValue();

    if(g!=null){
      (new Rectangulo((int)d1, ((int)d2), (int)d3, ((int)d4) )).dibuja(g);
    }
  }
```

Resultados

Una vez corriendo el programa, procedemos a crear un dibujo ingresando las siguientes instrucciones:

```
rectangulo 20 20 50 50 ;
line 15 20 45 5 ;
line 45 5 75 20 ;
circulo 10 100 25 ;
line 105 35 105 45 ;
line 100 40 110 40 ;
line 100 70 105 45 ;
line 110 70 105 45 ;
rectangulo 130 50 80 15 ;
circulo 10 130 60 ;
circulo 10 180 60 ;
rectangulo 140 40 30 10 ;
```



Conclusión

En esta práctica vimos la introducción a lo que es una máquina virtual, que hasta el momento lo vemos como una caja negra que guarda cadenas con nombre de métodos que realizan una acción, variables o constantes y nos permite obtenerlas en la lógica para poder realizar las acciones pertinentes.