PROYECTO - *SSH* COMPILER

Nelson Sanabio
nsanabiom@uni.pe

Angela Serrano
angelaserrano301@gmail.com

Angel Huiza
ahflores20@gmail.com

RESUMEN

El presente trabajo tiene como objetivo la aplicación del conocimiento obtenido en clase en la realización de un compilador, el cual deberá aceptar sentencias válidas según el léxico y gramática definidas previamente por nuestro compilador. Para la realización del proyecto se usar ciertas reglas, patrones, palabras reservadas y variables los cuales ayudarán a un mejor manejo del mismo. El compilador realizado es mostrado a través de una interfaz gráfica, en la cuál se podrá escribir la sentencia deseada y se verificará si lo escrito es válido o no haciendo clic en el botón RUN establecido para ese trabajo.

Keywords Compilador · Bison · Gramática · Variables · Token · Analizador Léxico

1 Características

- Posee interfaz gráfica.
- Es un programa estandarizado.
- Soporta un lenguaje estructurado que incluye declaración de funciones, condicionales y loops.
- Software basado en lenguaje C/Python y usa herramientas externas como BISON.

2 Ventajas

El proyecto fue realizado con la finalidad de hacer más eficiente la codificación de un lenguaje pre-definido.

- La interfaz es una ayuda para que sea de fácil entendimiento y es amigable es decir cualquier persona la puede entender.
- Es para el público en general.
- Es estructurado porque esta orientado a mejorar la claridad, calidad y tiempo de desarrollo.
- El software es portable.
- Es open-source.

3 Reglas y Patrones

Las reglas y patrones que deben ser seguidas por el usuario también están definidas por el compilador las cuales pueden ser vistas en el código del programa.

Mostraremos unos ejemplos de lo establecido en el programa:

- `include:raiz | libreria include;`
  el cual establece el inicio de cualquier programa que se vaya a codificar en el compilador. Siendo `raiz` una función dentro del programa y `libreria` la importación de una librería o paquete que quiera agregarse al programa que se desea compilar.
- `libreria: '%' SINTLIB '"' LIBRERIA '"';`
  Como podemos apreciar aquí se define la estructura que debe tener `libreria`, mencionada en el punto anterior, en donde podemos observar que toda importación de librerias o paquetes debe iniciar con el símbolo de

porcentaje (“%”) seguido de la palabra reservada SINTLIB, la cuál está definida en el código del programa de la siguiente manera: if(!strcmp(lexema,"import")) return SINTLIB;, aquí podemos apreciar que la palabra reservada que hará referencia a SINTLIB será import, a lo que luego seguirá el nombre de la librería o paquete que vayamos a tomar, para nuestro caso hemos definido, de momento, las siguiente librerias:

- if(!strcmp(lexema,"math")) return LIBRERIA;
- if(!strcmp(lexema,"io")) return LIBRERIA;
- if(!strcmp(lexema,"string")) return LIBRERIA;

fuera de estas librerías no habrán otras que sean aceptadas.

- raiz: | comment raiz | sentencia raiz | print raiz
  | bucles CA raiz CC raiz | condicional CA raiz CC raiz
  | funcion CA raiz CC;

Como podemos observar, aquí tenemos el otro patrón llamado raiz, el cual tiene más posibilidades de ser mencionado dentro de la compilación.

3.1 Palabras Reservadas

Las palabras reservadas por nuestro compilador son las siguientes:

- script
- when
- repeatWhen
- do
- for
- each
- in
- pt
- import
- math
- io
- string

Los cuales, por su misma condición de reservados, no deben ser tomados para nombrar alguna variables o función a crear.

3.2 Inicialización de variables

Ejemplo:

```
variable1 = 3
variable2 = 4
variable3 = variable2
```

3.3 Estructuras de Control

Ejemplo:

```
repeatWhen  variable1 < 5  [
    variable2 = variable1+2
]
```

Ejemplo:

```
when  variable2 = 2  do [
    pt "Hola mundo!"
]
```

4 Gramática

Para la gramática se ha tomado lo siguiente:

```
include:raiz
      | libreria include;
raiz:
      | comment raiz
      | sentencia raiz
      | print raiz
      | bucles CA raiz CC raiz
      | condicional CA raiz CC raiz
      | funcion CA raiz CC raiz;

libreria: '%' SINTLIB '"' LIBRERIA '"';

bucles: MIENTRAS cond
      | FOR EACH variable IN NUM;

condicional: SI  cond  DO;

funcion: FUNCION VAR '(' parametros ')' '$' parametros;

parametros: VAR
          | VAR ',' parametros;

variable: VAR variable
        | VAR;

comment: '&' variable '&';

print: PRINT '"' variable '"';

cond: two MAYOR two
    | two MENOR two
    | two IGUAL two;

two: NUM
   | VAR;

sentencia:VAR IGUAL expr;

expr: expr MAS term
    | expr MENOS term
    | term;

term: fact
    | term MULT fact
    | term DIV fact;

fact: NUM
    | VAR;
```

5 Codificación del Analizador Léxico

El código del programa se encuentra en el archivo `SHS.y`, el cual mostraremos a continuación:

```
%{
#include<ctype.h>
#include<stdio.h>
#include<string.h>
char lexema[255];
void yyerror(char *);
int yylex();
%}
%token VAR NUM
%token FUNCION PRINT
%token SI MAYOR MENOR IGUAL DO
%token MIENTRAS FOR EACH IN
%token SINTLIB LIBRERIA
%token CA CC
```

```
15
16  %%
17
18  include:
19              | libreria include
20              | comment raiz ;
21
22  raiz:
23          | comment raiz
24          | sentencia raiz
25          | print raiz
26          | expr raiz
27          | bucles CA raiz CC
28          | condicional CA raiz CC
29          | funcion CA raiz CC;
30
31  libreria: '%' SINTLIB '"' LIBRERIA '"';
32
33  bucles: MIENTRAS cond
34              | FOR EACH variable IN NUM;
35
36  condicional: SI  cond  DO;
37
38  funcion: FUNCION VAR '(' parametros ')' '$' parametros
39
40  parametros:
41              | VAR
42              | VAR ',' parametros;
43
44
45  variable: VAR variable
46              | VAR;
47
48
49  comment: '&' variable '&';
50
51  print: PRINT '"' variable '"';
52
53  cond: VAR MAYOR VAR
54          | VAR MENOR VAR
55          | VAR IGUAL VAR;
56
57  sentencia: VAR IGUAL VAR
58              | VAR IGUAL NUM;
59
60  expr: expr '+' term
61          | expr '-' term
62          | term;
63
64  term: fact
65          | term '*' fact
66          | term '/' fact;
67
68  fact: NUM;
69
70  %%
71  void yyerror(char *mgs){
72      printf("\n... Error: %s ",mgs);
73      }
74
75  int yylex(){
76   char c;
77   while(1){
78      memset( lexema, 0, sizeof(lexema));
79      c=getchar();
```

```
80      if(c=='\n') continue;
81      if(isspace(c)) continue;
82          if(c == '>') return MAYOR;
83          if(c == '<') return MENOR;
84          if(c == '=') return IGUAL;
85          if(c == '[') return CA;
86          if(c == ']') return CC;
87
88      if(isalpha(c)){
89          int i=0;
90              do{
91                  lexema[i++]=c;
92                  c=getchar();
93              }while(isalnum(c));
94
95              ungetc(c,stdin);
96              lexema[i] = 0;
97
98              if(!strcmp(lexema,"script")) return FUNCION;
99              if(!strcmp(lexema,"when")) return SI;
100             if(!strcmp(lexema,"repeatWhen")) return MIENTRAS;
101             if(!strcmp(lexema,"do")) return DO;
102             if(!strcmp(lexema,"for")) return FOR;
103             if(!strcmp(lexema,"each")) return EACH;
104             if(!strcmp(lexema,"in")) return IN;
105             if(!strcmp(lexema,"pt")) return PRINT;
106             if(!strcmp(lexema,"import")) return SINTLIB;
107             if(!strcmp(lexema,"math")) return LIBRERIA;
108             if(!strcmp(lexema,"io")) return LIBRERIA;
109             if(!strcmp(lexema,"string")) return LIBRERIA;
110
111             return VAR;
112     }
113     if( c >= 48 && c <= 57){
114         int i=0;
115             do{
116                 lexema[i++]=c;
117                 c=getchar();
118             }while((isdigit(c)));
119
120             ungetc(c,stdin);
121             lexema[i]=0;
122             return NUM;
123             }
124
125         return c;
126     }
127 }
128 int main(){
129     if(!yyparse()){
130         printf("cadena valida\n");
131         return 1;
132     }else{
133         printf("cadena invalida\n");
134         return 0;
135     }
136     return 0;
137 }
```

El cuál para ser usado deberá compilarse de la siguiente manera usando BISON:

```
$ bison  SHS.y
```

Cabe aclarar que podemos agregar la opción `-v` para generar el archivo `SHS.output` el cuál contiene la relación de estados en nuestro compilador:

```
$ bison  -v SHS.y
```

Luego pasaremos a compilar el archivo `SHS.tab.c` generado en el punto anterior con ayuda de `gcc`:

```
$ gcc SHS.tab.c
```

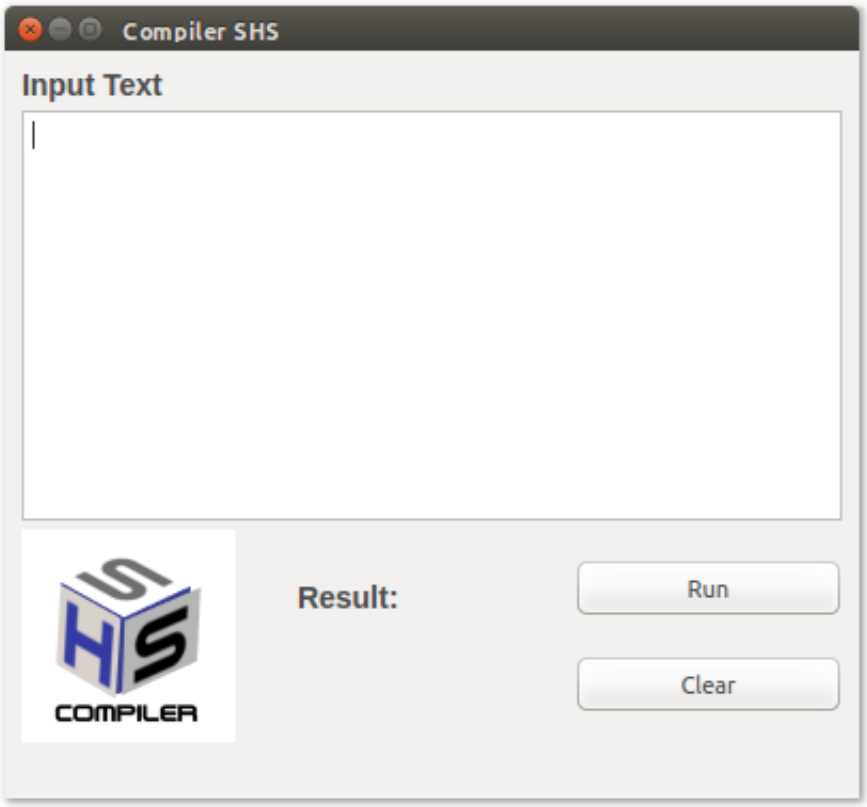
Luego de hacer esto pasaremos a ejecutar el programa `a.out` generado, de la siguiente manera:

```
$ ./a.out
```

Adicional a todo esto ya mencionado, hemos desarrollado una interfaz que hará más agradable el uso del compilador para esto usaremos lo siguiente en el terminal:

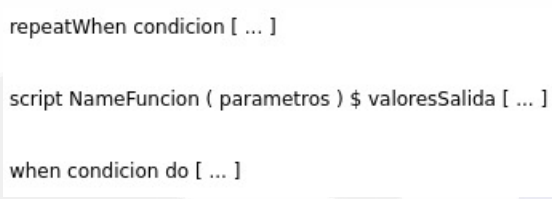
```
$ python ProyectoCompiladores.py
```

Mostrándose de la siguiente manera:



6 Definición de la Sintáxis de las Sentencias

Dado todo lo ya mostrado hasta el momento pasemos a revisar las *sentencias selectivas y repetitivas* que han sido implementadas en el compilador. Como podemos ver en la siguiente imagen:



- **Sentencia Repetitiva.** Referido a la sentencia que permitirá al compilador aceptar bucles (o loops).

  - `repeatWhen condicion [...]`
    La función realizará la tarea encomendada en el espacio entre corchetes *multiples veces* miéntras se cumpla una condición establecida por el usuario.

- **Sentencia Selectiva.** Referido a la sentencia que permitirá al compilador aceptar funciones.

  - `script NameFunction (parametros) $ valoresSalida [...]`
    la función realizará la tarea encomendada en el espacio entre corchetes cuando sea llamada en el programa creado por el usuario; escribiendo previamente la palabra `script` la cual identificará que se está creando una función, luego dando el nombre de la función en `NameFunction` y estableciendo los `parametros` con los que trabajará dicha función, seguidamente se usará el símbolo $ el cual indicará que valores de variables retornará dicha función en `valoresSalida`.

  - `when condicion [...]`
    La función realizará la tarea encomendada en el espacio entre corchetes *una sola vez* cuando cumpla la condición establecida por el usuario.

7 Aplicación del Método Shift-Reduce

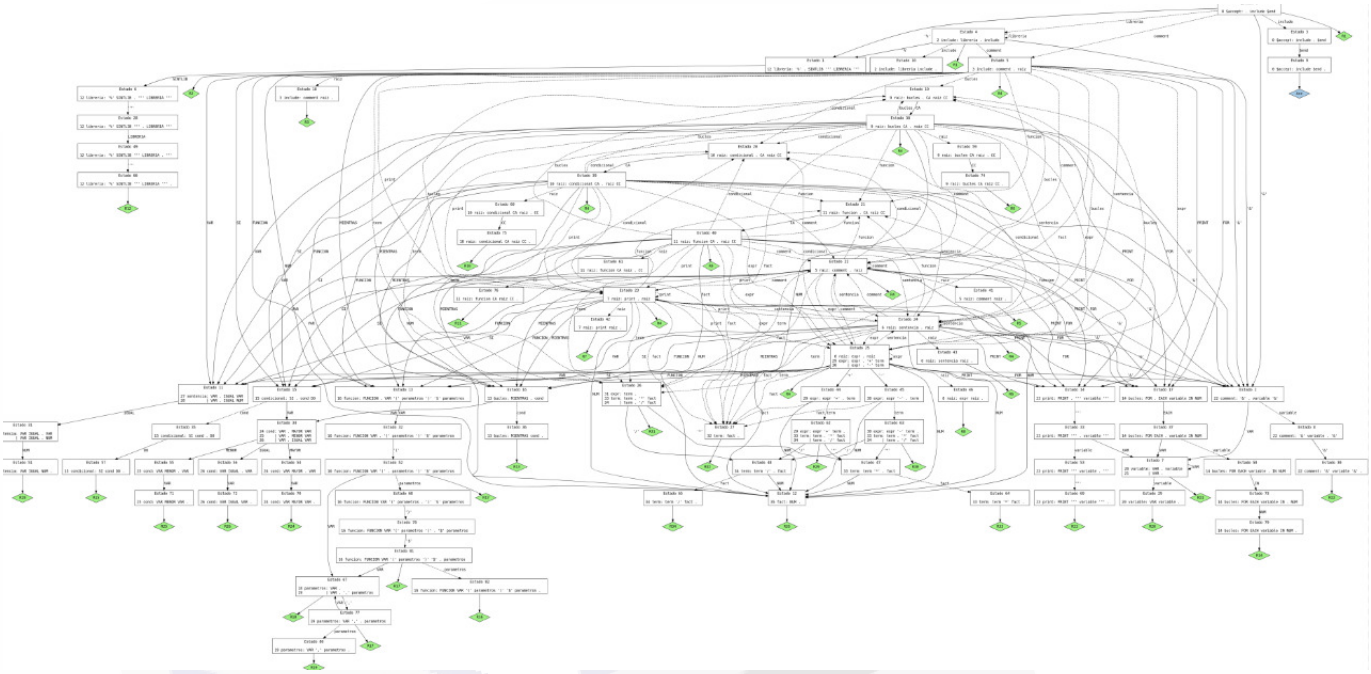
Antes de pasar a realizar el método, queremos mostrar una aplicación[1] encontrada en un repositorio github el cual ayuda en la observación del árbol de estados que se genera en un archivo bison.

Si deseamos visualizar el árbol de estados se usa el siguiente comando que generará un archivo `.dot` y con ayuda de un convertidor se visualizará como imagen.

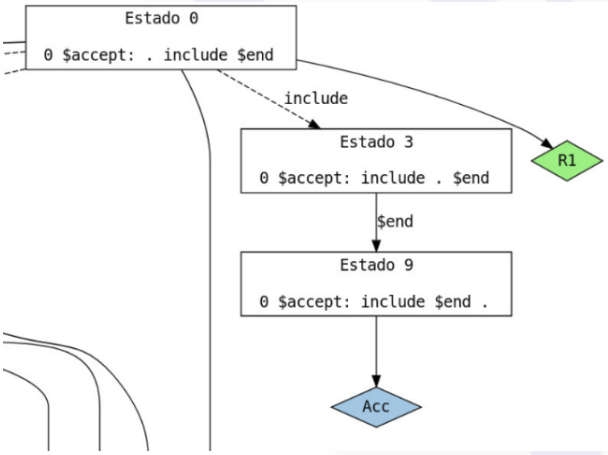
```
$ bison --graph SSH.y
```

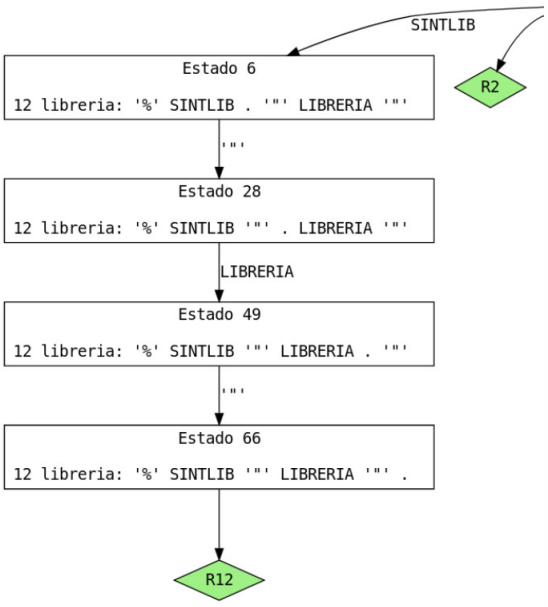
Este nuevo archivo `.dot` generado será pasado a la aplicación mencionada (**GraphvizOnline**[2]) y nos brindará como respuesta el gráfico de los 83 estados que posee nuestro compilador como se muestra a continuación:

[1]Página web: `https://github.com/dreampuf/GraphvizOnline`
[2]Página web: `https://github.com/dreampuf/GraphvizOnline`



Como bien podemos observar, al tener un total de 83 estados en nuestro compilador, este arbol de relación de estados se hace muy complejo y enmarañado para su completa comprensión, así que demos un vistazo más de cerca:





Dicho y mostrado esto, pasaremos a tomar la siguiente sentencia para verificar el correcto funcionamiento del programa:

when var1 > var2 do [var3 = 8]

la cual, para nuestro compilador, estaría siendo vista de la siguiente manera:

SI VAR MAYOR VAR DO CA VAR IGUAL NUM CC

Ahora, pasemos a realizar la tabla de estados, guiándonos del archivo `SHS.output`:

Estados	Sentencia
0	~~SI~~ VAR MAYOR VAR DO CA VAR IGUAL NUM CC
04	~~VAR~~ MAYOR VAR DO CA VAR IGUAL NUM CC
04~~(21)~~	MAYOR VAR DO CA VAR IGUAL NUM CC R25 two: NUM goto(4,two)
04(24)	~~MAYOR~~ VAR DO CA VAR IGUAL NUM CC
04(24)(46)	~~VAR~~ DO CA VAR IGUAL NUM CC
04(24)(46)~~(21)~~	DO CA VAR IGUAL NUM CC R25 two: NUM goto(46,two)
04~~(24)(46)(63)~~	DO CA VAR IGUAL NUM CC R21 cond: two MAYOR two goto(4,cond)
04(23)	~~DO~~ CA VAR IGUAL NUM CC
04~~(23)(45)~~	CA VAR IGUAL NUM CC R13 condicional: SI cond DO goto(0,condicional)
0(13)	~~CA~~ VAR IGUAL NUM CC
0(13)(33)	~~VAR~~ IGUAL NUM CC
0(13)(33)1	~~IGUAL~~ NUM CC
0(13)(33)1(18)	~~NUM~~ CC
0(13)(33)1(18)~~(39)~~	CC R34 fact: NUM goto(18,fact)
0(13)(33)1(18)~~(42)~~	CC R31 term: fact goto(18,term)
0(13)(33)1(18)~~(41)~~	CC R30 expr: term goto(18,expr)
0(13)(33)~~1(18)(40)~~	CC R27 sentencia: VAR IGUAL expr goto(33,sentencia)
0(13)(33)(17)	CC R3 raiz: ir al estado 37
0(13)(33)~~(17)(37)~~	CC R5 raiz: sentencia raiz goto(33,raiz)
0(13)(33)(54)	~~CC~~
0(13)(33)(54)(69)	ϕ R3 raiz: ir al estado 80
0~~(13)(33)(54)(69)(80)~~	ϕ R8 raiz: condicional CA raiz CC raiz goto(0,raiz)
0~~(10)~~	ϕ R1 include: raiz -> goto(0,include)
09	ϕ
09(30)	ϕ ACEPTAR

Table 1: Tabla de Estados para el Método Shift-Reduce

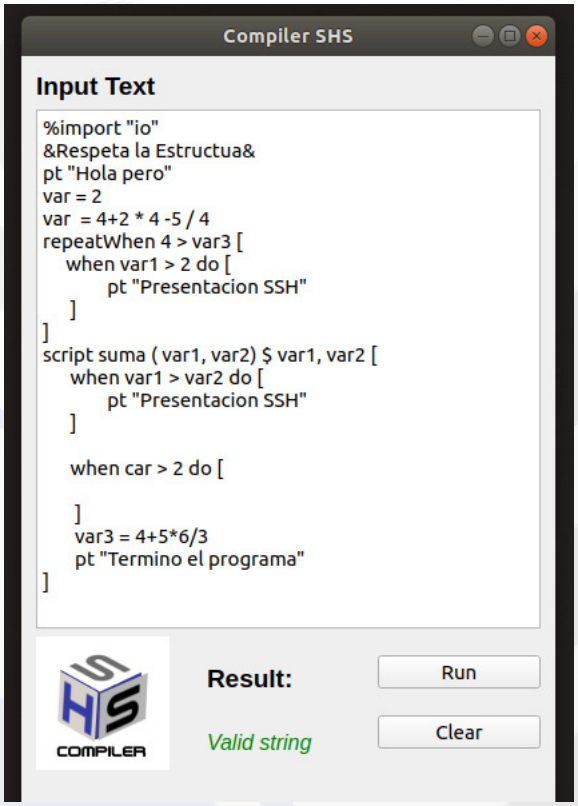
La cadena ingresada es válida, dando a entender que cumple con todas las características para que se pueda usar en nuestro compilador.

8 Pruebas Realizadas

Aquí algunas pruebas realizadas en el compilador.

Sentencia Válida

Presenta una correcta introducción de los patrones y variables.



Sentencia Inválida

Presenta una incorrecta inctroducción de sináxis dado que puede estar tomando una palabra reservada, digitando mal la sintáxis, etc

En este caso el error es debido a que la impresión de texto se realiza con el comando `pt` y no con `print` como se muestra en la imagen.