Sesión 06: **Gramáticas independientes de contexto con Bison**

I. OBJETIVOS

- Utilizar bison para crear gramáticas independientes de contexto
- Manejar los errores al evaluar una gramatica

II. TEMAS A TRATAR

- Gramáticas libres de contexto
- Manejo de errores

III. MARCO TEORICO

Gramáticas Independientes del Contexto

Una Gramática independientes del contexto (GIC) es una gramática formal en la que cada regla de producción es de la forma:

Exp → x

Donde Exp es un símbolo no terminal y x es una cadena de terminales y/o no terminales. El término independiente del contexto se refiere al hecho de que el no terminal Exp puede siempre ser sustituido por x sin tener en cuenta el contexto en el que ocurra. Un lenguaje formal es independiente de contexto si hay una gramática libre de contexto que lo genera, este tipo de gramática fue creada por Backus-Naur y se utiliza para describir la mayoría de los lenguajes de programación.

Una GIC está compuesta por 4 elementos:

1. Símbolos terminales (elementos que no generan nada)

2. No terminales (elementos del lado izquierdo de una producción, antes de la flecha "->")

3. Producciones (sentencias que se escriben en la gramática)

4. Símbolo inicial (primer elemento de la gramática)

Manejo de errores en Bison

El analizador de Bison detecta un "error de análisis" o un "error de sintaxis" siempre que lee un token que no puede satisfacer ninguna regla de sintaxis. Una acción en la gramática también puede proclamar explícitamente un error, usando la macro `YYERROR. El analizador de Bison espera informar el error llamando a un función de error denominada 'yyerror', esta da un informe que debe proporcionar. Está llamado por 'yyparse' cada vez que se encuentra un error de sintaxis, y recibe un argumento. Para un error de análisis, la cadena es normalmente `" error de análisis "'.

Si define la macro 'YYERROR_VERBOSE' en las declaraciones de Bison, entonces Bison proporciona una cadena de mensaje de error más detallada y específica en lugar de simplemente "error de análisis". No importa que definición use para 'YYERROR_VERBOSE', basta con definirla.

El analizador puede detectar otro tipo de error: desbordamiento de pila. Esta sucede cuando la entrada contiene construcciones que son muy profundamente anidadas. No es probable que encuentres esto, ya que el Bison el analizador extiende su pila automáticamente hasta un límite muy grande. Pero si se produce un desbordamiento, 'yyparse' llama a 'yyerror' de la manera habitual, excepto que la cadena de argumento es `" desbordamiento de pila de analizador ''.

La siguiente definición es suficiente en programas simples:

```
yyerror (es)
    char * s;
{
 fprintf (stderr, "% s \ n", s);
}
```

Después de que 'yyerror' regrese a 'yyparse', este último intentará recuperar el error si ha escrito reglas adecuadas de gramática de recuperación de errores. Si la recuperación es imposible, 'yyparse' regresar inmediatamente 1.

La variable 'yynerrs' contiene el número de errores de sintaxis encontrado hasta ahora. Normalmente esta variable es global; pero si solicitastes un analizador puro entonces es una variable local a la que solo pueden acceder acciones.

IV. ACTIVIDADES (La práctica tiene una duración de 2 horas)

1. Crear una programa que acepte solamente un token

Primero el archivo lexico.l

```
%{
   #include "sintactico.tab.h"
%}

%%

El {return ARTICULO;}

%%
int yywrap(){ return 1;}
```

Segundo el archivo sintactico.y

```
%{
```

```
#include <stdio.h>
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token ARTICULO

%%

cadena: ARTICULO {printf("Se imprimio una cadena \n");
                  exit(0);}
;

%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

2. Crear un programa que rechace cualquier carácter o palabra que no sea “El”

primero el lexico

```
%{
   #include "sintactico.tab.h"
%}

%%

El {return ARTICULO;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

segundo el sintactico

```
%{

#include <stdio.h>
```

```
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token ARTICULO
%token NL

%%

cadena: ARTICULO NL {printf("Se imprimio una cadena \n");
                     exit(0);}
;

%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

3.Agregar un mensaje de inicio al programa

Primero el lexico

```
%{
   #include "sintactico.tab.h"
%}

%%

El {return ARTICULO;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

Segundo el sintactico

```
%{

#include <stdio.h>
```

```
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token ARTICULO
%token NL

%%

cadena: ARTICULO NL {printf("Se imprimio una cadena \n");
                     exit(0);}
;

%%
int yyerror (char *s)
{

 printf ("cadena invalida\n", s);
 return 0;
}
void main(){
   printf("ingresa la cadena\n");
   yyparse();
   yylex();
}
```

4.Agregar un mensaje personalizado al mensaje de error

Primero el léxico

```
%{
   #include "sintactico.tab.h"
%}

%%

El {return ARTICULO;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

Segundo el sintactico, asegurarse de guardarlos como sintactico1.y

```
%{
```

```
#include <stdio.h>
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token ARTICULO
%token NL

%%

cadena: ARTICULO NL {printf("Se imprimio una cadena \n");
                     exit(0);}
;

%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

5. Utilizando bison, crear las siguientes reglas de producción de gramaticas A => aA, A=>a

Primero lexico.l

```
%{
   #include "sintactico.tab.h"
%}

%%

[aA] {return A;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

Segundo sintactico.y

```
%{

#include <stdio.h>
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token A
%token NL

%%

cadena: S NL {printf("Se imprimio una cadena \n");
                    exit(0);}
;
S: S A |
;

%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

6. Utilizando bison, crear las siguientes reglas de producción de gramaticas A => aA, A=>AA, A=>a

Primero lexico.l

```
%{
   #include "sintactico.tab.h"
%}

%%

[aA] {return A;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

Segundo sintactico.y

```
%{

#include <stdio.h>
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token A
%token NL

%%

cadena: S NL {printf("Se imprimio una cadena \n");
                exit(0);}
;
S: S A A |
;

%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

7. Utilizando bison, crear las siguientes reglas de producción de gramaticas en un archivo sintactico.y para que reconozca cadenas que acepten las letras a o b o la combinación de ambos

Primero lexico.l

```
%{
    #include "sintactico.tab.h"
%}

%%

[aA] {return A;}
[Bb] {return B;}
\n {return NL;}
.  {return yytext[0];}
```

```
%%
int yywrap(){ return 1;}
```

Segundo sintactico.y

```
%{

#include <stdio.h>
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token A
%token B
%token NL

%%

cadena: S NL {printf("Se imprimio una cadena \n");
                    exit(0);}
;
S: A S | B S |
;

%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

8. Utilizando bison, crear reglas de producción de gramaticas en un archivo sintactico.y para que reconozca cadenas de cualquier cantidad de letras “a” pero que terminen en la letra “b”

Primero lexico.l

```
%{
    #include "sintactico.tab.h"
%}

%%
```

```
[aA] {return A;}
[Bb] {return B;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

Segundo sintactico.y

```
%{

#include <stdio.h>
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token A
%token B
%token NL

%%

cadena:  S B NL {printf("Se imprimio una cadena \n");
                 exit(0);}
;
S: A S |
;

%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

9. Utilizando bison, crear reglas de producción de gramáticas que reconozcan cualquier cantidad de letras “b” pero solo una letra “a” al inicio

Primero lexico.l

```
%{
   #include "sintactico.tab.h"
%}

%%

[aA] {return A;}
[Bb] {return B;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

Segundo sintactico.y

```
%{

#include <stdio.h>
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token A
%token B
%token NL

%%

cadena: A S NL {printf("Se imprimio una cadena \n");
                    exit(0);}
;
S: S B | B
;

%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
```

```
  yylex();
}
```

10. Utilizando bison, crear reglas de producción de gramaticas que reconozcan solo cadenas de “a” o solo cadenas de letras “b”

Primero lexico.l

```
%{
   #include "sintactico.tab.h"
%}

%%

[aA] {return A;}
[bB] {return B;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

Segundo sintactico.y

```
%{

#include <stdio.h>
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token A
%token B
%token NL

%%

cadena: P NL {printf("Se imprimio una cadena 'b' \n");
                    exit(0);}
      |S NL {printf("Se imprimio una cadena 'a' \n");
                    exit(0);}
;
S: S A | A
;
P: P B | B
;
```

```
%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

11. Uitlizando bison, crear reglas de producción de gramaticas que reconozcan cadenas del letras “a” y “b” intercaladas pero no cadenas juntas de letras “a” o “b”

Primero lexico.l

```
%{
    #include "sintactico.tab.h"
%}

%%

[aA] {return A;}
[bB] {return B;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

Segundo sintactico.y

```
%{

#include <stdio.h>
int yylex();
int yyerror (char *s);


%}

/* Declaraciones de BISON */

%token A
%token B
%token NL

%%

cadena: P NL {printf("Termino en la producción 'P' \n");
```

```
                exit(0);}
      |S NL {printf("Termino en la producción 'S' \n");
                exit(0);}
;
S: P A | A
;
P: S B | B
;

%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

12. Utilizando bison, crear reglas de producción de gramaticas que muestren cadenas pares de letras “a” junto con cadenas pares de letras “b’

Primero lexico.l

```
%{
   #include "sintactico.tab.h"
%}

%%

[aA] {return A;}
[bB] {return B;}
\n {return NL;}
.  {return yytext[0];}

%%
int yywrap(){ return 1;}
```

Segundo sintactico.y

```
%{

#include <stdio.h>
int yylex();
int yyerror (char *s);


%}
```

```
/* Declaraciones de BISON */

%token A
%token B
%token NL

%%

cadena: P NL {printf("Termino en la producción 'P' \n");return 0;}
      |S NL {printf("Termino en la producción 'S' \n");return 0;}
;
S: A A P | A A
;
P: B B S | B B
;


%%
int yyerror (char *s)
{

 printf ("%s\n", s);
 return 0;
}
void main(){
   yyparse();
   yylex();
}
```

13. Utilizando bison, crear reglas de producción de gramaticas que reconozcan igual cantidad de a y también igual cantidad de b

Primero lexico.l

```
%{
/* Sección de definición */
#include "sintactico.tab.h"
%}

/* Sección de reglas*/
%%
[aA] {return A;}
[bB] {return B;}
\n {return NL;}
.  {return yytext[0];}
%%

int yywrap()
{
   return 1;
}
```

Segundo sintactico.y

```
%{
/* Definition section */
  #include<stdio.h>
  #include<stdlib.h>
%}

%token A B NL
/* Sección de reglas */
%%
stmt: S NL  { printf("cadena validad\n");
          exit(0); }
;
S: A S B |
;
%%

int yyerror(char *msg)
{
 printf("cadena invalida\n");
 exit(0);
}
//driver code
main()
{
printf("ingresa la cadena\n");
yyparse();
}
```

14.Utilizando bison, crear reglas de producción de gramaticas que reconozcan la expresión número + número

Primero lexico.l

```
%{
#include "sintactico.tab.h"
#include "string.h"
int yyparse();

%}
numero [0-9]+
%%
{numero}     {return(NUM);};
"+"          {return('+');};
"\n"         {return('\n');};
. ;
%%
int yywrap(){ return 0;}
```

```
void main(){
yyparse();
}
```

Segundo sintactico.y

```
%{

  #include <stdio.h>
  extern int yylex(void);
  extern int linea;
  void yyerror(char *s);
%}

%union {
char cadena[100];
int numero;
}
%token <numero> NUM
%token '+'
%token '\n'

%%
entrada: expnum '\n' {printf("Expresion reconocida \n");}
       |expnum '+' expnum {printf("Es una operacion entre numeros\n");}
 ;
expnum: NUM {printf("Es un numero \n");}
 ;

%%

void yyerror(char *s)
{
  printf("%s",s);
}
```

15.Utilizando bison, crear reglas de producción de gramaticas que reconozcan la expresión variable + variable

Primero lexico.l

```
%{
#include "sintactico.tab.h"
#include "string.h"
int yyparse();

%}
cadena [a-zA-Z]+
%%
{cadena}     {return(CAD);};
"+"          {return('+');};
"\n"         {return('\n');};
. ;
%%
int yywrap(){ return 0;}

void main(){
yyparse();
}
```

Segundo sintactico.y

```
%{

  #include <stdio.h>
  extern int yylex(void);
  extern int linea;
  void yyerror(char *s);
%}

%union {
char cadena[100];
int numero;
}
%token <cadena> CAD
%token '+'
%token '\n'

%%
entrada: expcad '\n' {printf("Expresion reconocida \n");}
       |expcad '+' expcad {printf("Es una operacion entre variables\n");}
;
```

```
expcad: CAD {printf("Es una variable \n");}
;

%%

void yyerror(char *s)
{
  printf("%s",s);
}
```

16.Utilizando bison, crear reglas de producción de gramaticas que reconozcan la expresión número + variable

Primero lexico.l

```
%{
#include "sintactico.tab.h"
#include "string.h"
int yyparse();

%}
cadena [a-zA-Z]+
numero [0-9]+
%%
{cadena}     {return(CAD);};
{numero}     {return(NUM);};
"+"          {return('+');};
"\n"         {return('\n');};
. ;
%%
int yywrap(){ return 0;}

void main(){
yyparse();
}
```

Segundo sintactico.y

```
%{

  #include <stdio.h>
  extern int yylex(void);
  extern int linea;
```

```
  void yyerror(char *s);
%}

%union {
char cadena[100];
int numero;
}
%token <numero> NUM
%token <cadena> CAD
%token '+'
%token '\n'

%%
entrada: expcad '\n' {printf("Expresion reconocida \n");}
       |expnum '\n' {printf("Expresion reconocida \n");}
       |expcad '+' expnum {printf("Es una operacion entre numeros y variables\n");}
       |expnum '+' expcad {printf("Es una operacion entre numeros y variables\n");}
 ;
 expcad: CAD {printf("Es una variable \n");}
 ;
expnum: NUM {printf("Es un numero \n");}
 ;

%%

void yyerror(char *s)
{
  printf("%s",s);
}
```

17.Utilizando bison, crear reglas de producción de gramaticas que reconozcan la expresión número - número
Primero lexico.l

```
%{
#include "sintactico.tab.h"
#include "string.h"
int yyparse();
```

```
%}
numero [0-9]+
%%
{numero}     {return(NUM);};
"-"          {return('-');};
"\n"         {return('\n');};
. ;
%%
int yywrap(){ return 0;}

void main(){
yyparse();
}
```

Segundo sintactico.y

```
%{

  #include <stdio.h>
  extern int yylex(void);
  extern int linea;
  void yyerror(char *s);
%}

%union {
char cadena[100];
int numero;
}
%token <numero> NUM
%token '-'
%token '\n'

%%
entrada: expnum '\n' {printf("Expresion reconocida \n");}
       |expnum '-' expnum {printf("Es una operacion entre numeros\n");}
 ;
expnum: NUM {printf("Es un numero \n");}
 ;

%%
```

```
void yyerror(char *s)
{
  printf("%s",s);
}
```

18.Utilizando bison, crear reglas de producción de gramaticas que reconozcan la expresión número - variable

Primero lexico.l

```
%{
#include "sintactico.tab.h"
#include "string.h"
int yyparse();

%}
cadena [a-zA-Z]+
numero [0-9]+
%%
{cadena}      {return(CAD);};
{numero}      {return(NUM);};
"-"           {return('-');};
"\n"          {return('\n');};
. ;
%%
int yywrap(){ return 0;}

void main(){
yyparse();
}
```

Segundo sintactico.y

```
%{

  #include <stdio.h>
  extern int yylex(void);
  extern int linea;
  void yyerror(char *s);
%}

%union {
char cadena[100];
int numero;
}
```

```
%token <numero> NUM
%token <cadena> CAD
%token '-'
%token '\n'

%%
entrada: expcad '\n' {printf("Expresion reconocida \n");}
       |expnum '\n' {printf("Expresion reconocida \n");}
       |expcad '-' expnum {printf("Es una operacion entre numeros y variables\n");}
       |expnum '-' expcad {printf("Es una operacion entre numeros y variables\n");}
 ;
 expcad: CAD {printf("Es una variable \n");}
 ;
expnum: NUM {printf("Es un numero \n");}
 ;

%%

void yyerror(char *s)
{
  printf("%s",s);
}
```

19.Utilizando bison, crear reglas de producción de gramaticas que reconozcan la expresión número * número

Primero lexico.l

```
%{
#include "sintactico.tab.h"
#include "string.h"
int yyparse();

%}
numero [0-9]+
%%
{numero}     {return(NUM);};
"*"          {return('*');};
"\n"         {return('\n');};
. ;
%%
int yywrap(){ return 0;}
```

```
void main(){
yyparse();
}
```

Segundo sintactico.y

```
%{

  #include <stdio.h>
  extern int yylex(void);
  extern int linea;
  void yyerror(char *s);
%}

%union {
char cadena[100];
int numero;
}
%token <numero> NUM
%token '*'
%token '\n'

%%
entrada: expnum '\n' {printf("Expresion reconocida \n");}
       |expnum '*' expnum {printf("Es una operacion entre numeros\n");}
 ;
expnum: NUM {printf("Es un numero \n");}
 ;

%%

void yyerror(char *s)
{
  printf("%s",s);
}
```

20.Utilizando bison, crear reglas de producción de gramaticas que reconozcan la expresión número * variable

Primero lexico.l

```
%{
#include "sintactico.tab.h"
#include "string.h"
int yyparse();

%}
cadena [a-zA-Z]+
numero [0-9]+
%%
{cadena}     {return(CAD);};
{numero}     {return(NUM);};
"*"          {return('*');};
"\n"         {return('\n');};
. ;
%%
int yywrap(){ return 0;}

void main(){
yyparse();
}
```

Segundo sintactico.y

```
%{

  #include <stdio.h>
  extern int yylex(void);
  extern int linea;
  void yyerror(char *s);
%}

%union {
char cadena[100];
int numero;
}
%token <numero> NUM
%token <cadena> CAD
%token '-'
%token '\n'
```

```
%%
entrada: expcad '\n' {printf("Expresion reconocida \n");}
       |expnum '\n' {printf("Expresion reconocida \n");}
       |expcad '*' expnum {printf("Es una operacion entre numeros y variables\n");}
       |expnum '*' expcad {printf("Es una operacion entre numeros y variables\n");}
 ;
 expcad: CAD {printf("Es una variable \n");}
 ;
expnum: NUM {printf("Es un numero \n");}
 ;

%%

void yyerror(char *s)
{
  printf("%s",s);
}
```

21.Utilizando bison, crear reglas de producción de gramaticas que reconozcan la expresión número / número

Primero lexico.l

```
%{
#include "sintactico.tab.h"
#include "string.h"
int yyparse();

%}
numero [0-9]+
%%
{numero}     {return(NUM);};
"/"          {return('/');};
"\n"         {return('\n');};
. ;
%%
int yywrap(){ return 0;}

void main(){
yyparse();
}
```

Segundo sintactico.y

```
%{

  #include <stdio.h>
  extern int yylex(void);
  extern int linea;
  void yyerror(char *s);
%}

%union {
char cadena[100];
int numero;
}
%token <numero> NUM
%token '/'
%token '\n'

%%
entrada: expnum '\n' {printf("Expresion reconocida \n");}
       |expnum '/' expnum {printf("Es una operacion entre numeros\n");}
 ;
expnum: NUM {printf("Es un numero \n");}
 ;

%%

void yyerror(char *s)
{
  printf("%s",s);
}
```

22.Utilizando bison, crear reglas de producción de gramaticas que reconozcan la expresión número / variable

Primero lexico.l

```
%{
#include "sintactico.tab.h"
```

```
#include "string.h"
int yyparse();

%}
cadena [a-zA-Z]+
numero [0-9]+
%%
{cadena}     {return(CAD);};
{numero}     {return(NUM);};
"/"          {return('/');};
"\n"         {return('\n');};
. ;
%%
int yywrap(){ return 0;}

void main(){
yyparse();
}
```

Segundo sintactico.y

```
%{

  #include <stdio.h>
  extern int yylex(void);
  extern int linea;
  void yyerror(char *s);
%}

%union {
char cadena[100];
int numero;
}
%token <numero> NUM
%token <cadena> CAD
%token '-'
%token '\n'

%%
entrada: expcad '\n' {printf("Expresion reconocida \n");}
       |expnum '\n' {printf("Expresion reconocida \n");}
```

```
      |expcad '/' expnum {printf("Es una operacion entre numeros y variables\n");}
      |expnum '/' expcad {printf("Es una operacion entre numeros y variables\n");}
;
 expcad: CAD {printf("Es una variable \n");}
 ;
expnum: NUM {printf("Es un numero \n");}
 ;

%%

void yyerror(char *s)
{
  printf("%s",s);
}
```

VII. Ejercicios

1. Describir que pasa si no se considera el carácter "." en el analizador léxico.
2. ¿Cual es el mejor lugar para definir la función main, en el lexer o en el parser? ¿Porque?
3. Crear un programa que solo reconozca "bb"
4. Crear un programa que haga la operación asignación de valor a una variable
5. Redactar un informe con los ejercicios propuestos

VII. Bibliografía y referencias

1. https://www.gnu.org/software/bison/
2. https://es.qwe.wiki/wiki/Context-free_grammar
3. https://cnx.org/contents/YuBtnLIS@1/Gramaticas-Independientes-del-Contexto-ejemplos-y-ejercicios
4. https://users.dcc.uchile.cl/~gnavarro/apunte.pdf
5. https://docs.freebsd.org/info/bison/bison.info.Error_Reporting.html