Parte I: Analizador léxico-sintáctico

Objetivo:

Aprender a implementar analizadores léxico sintácticos usando las herramientas Flex y Bison.

Fecha límite entrega Parte I (A. Léxico-Sintáctico):

8/11/2020

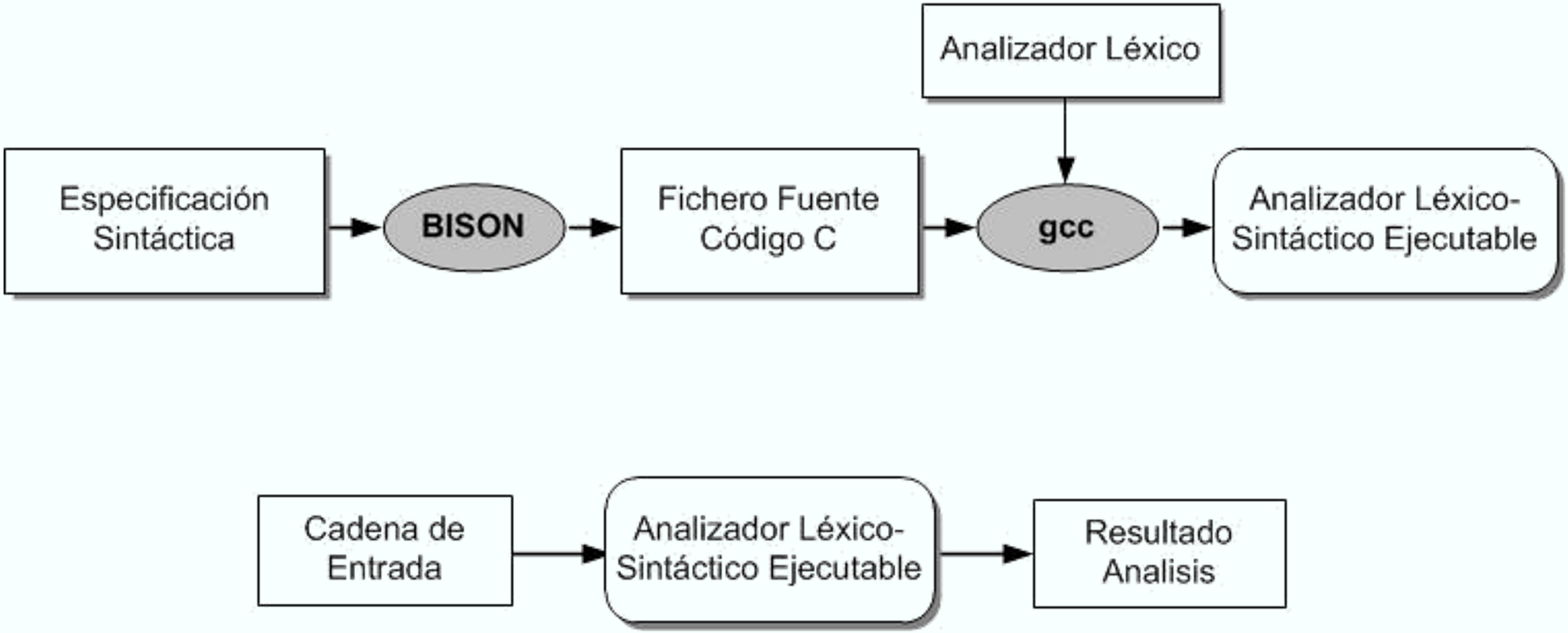
Modo de entrega:

- Individual
- A través de una tarea PoliformaT

Bison I: Introducción

Lenguajes de Programación y Procesadores de Lenguajes

Uso de Bison


Analizador Léxico
Especificación Sintáctica
BISON
Fichero Fuente Código C
gcc
Analizador Léxico-Sintáctico Ejecutable
Cadena de Entrada
Analizador Léxico-Sintáctico Ejecutable
Resultado Analisis

Especificación Sintáctica

- Fichero de texto con extensión .y

- Dividido en 3 partes (separadas por %%):

  1. Definiciones: Declaraciones C y Bison

  2. Reglas: Reglas gramaticales

  3. Funciones de usuario: Funciones C opcionales

Sección de definiciones

Preámbulo C:

```
%{
  #include <stdio.h>
  extern int yylineno
%}
```

Declaraciones Bison

Identificadores para cada símbolo terminal (token)

```
%token nombre_token   nombre_token   …
```

Ejemplo:

```
%token PARA_ PARC_ MAS_ MENOS_ POR_ DIV_
%token CTE_
```

Sección de reglas

Contiene reglas de la forma:

noterminal **:** *lado_derecho* **{** *acciones* **}**

- Primera regla es la del símbolo inicial de la gramática.

- Símbolos no-terminales: Minúsculas
- Símbolos terminales:
  - Identificador en mayúsculas (declarado con %token)
  - Carácter entre comillas simples (‘+’)
  - Cadena de caracteres (“<=“)

6

Sección de reglas

- Reglas de mismo no-terminal separadas por barra vertical (**|**)
- Última regla de un no-terminal acaba en punto y coma (**;**)

Ejemplo:

```
expMat: exp
      ;
exp: exp MAS_ term
   | exp MENOS_ term
   | term
   ;
term: term POR_ fac
    |  term DIV_ fac
    |  fac
    ;
fac : PARA_ exp PAR_
     | CTE_
     ;
```

Sección código usuario

- Opcional
- El código se copiará en el fichero de salida.
- Se usa para incorporar funciones que aparecen en las acciones de las reglas.

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

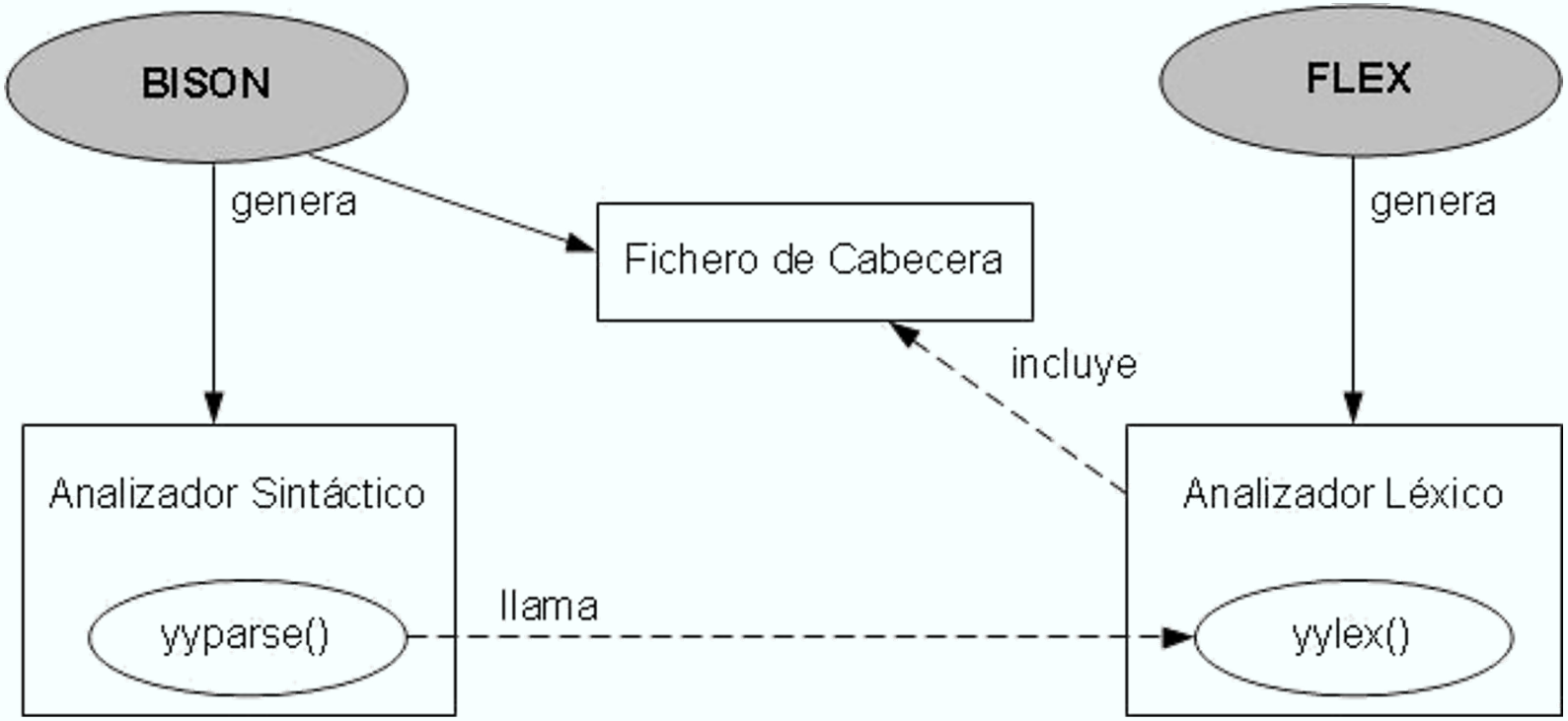
Analizador sintáctico Bison

- *El AS generado por Bison llama a **yylex()** para obtener un token.*
  - Escrita por el programador o usar la generada por Flex

- *Incluir una llamada a **yyparse()** en **main()** para comenzar el análisis.*

- ***yyparse()** devuelve 0 si llega a fin de fichero yel análisis fue correcto.*

Integración Bison - Flex



Integración Bison - Flex

- Compilar Bison con la opción **'-d'**.

  Genera fichero (por defecto <nombre>.tab.h) con las definiciones de los tokens de las declaraciones %token de Bison

  Ejemplo:

  ```
  bison -d  asin.y
  ```

  ```
  asin.tab.c
  asin.tab.h
  ```

- Incluir este fichero en Flex

  Escribir en la sección de preámbulo de C de flex:

  ```
  #include "asin.tab.h"
  ```

Integración Bison - Flex

Resumen modificaciones a realizar para la integración Bison-Flex

1. Incluir el fichero de cabecera generado por Bison.

2. Las acciones de las reglas léxicas deben ejecutar la sentencia `return` seguida del token detectado.

3. La funcion main debe llamar a `yyparse()` en lugar de llamar a `yylex()`.

Ejemplo Flex

```
%{
#include <stdio.h>
#include "header.h"
#include "asin.h”
#define  retornar(x) {if (verbosidad) ECHO; return x ; }
%}
delimitador     [ \t\n]+
digito          [0-9]
entero          {digito}+
%%
{delimitador}   {if (verbosidad) ECHO ; }
"+"             { retornar (MAS_) ; }
"-"             { retornar (MENOS_) ; }
"*"             { retornar (POR_) ; }
"/"             { retornar (DIV_) ; }
"("             { retornar (OPAR_) ; }
")"             { retornar (DIV_) ; }
{entero}        { retornar (CTE_) ; }
.               { yyerror("Caracter desconocido");     }
%%
```

Ejemplo Bison (1/2)

```
%{
  #include <stdio.h>
  #include "header.h"
%}
  %token PARA_ PARC_ MAS_ MENOS_ POR_ DIV_
  %token CTE_
%%
expMat : exp
       ;
exp    : exp MAS_   term
       | exp MENOS_ term
       | term
       ;
term   : term POR_ fac
       | term DIV_ fac
       | fac
       ;
fac    : PARA_ exp PARC_
       | CTE_
       ;
```

Ejemplo Bison (2/2)

```
int verbosidad = FALSE;

void yyerror(const char *msg){
   fprintf(stderr, "\nError en la linea %d: %s\n", yylineno, msg);
 }


int main(int argc, char **argv) {
   int i, n=1 ;

   for (i=1; i<argc; ++i)
      if (strcmp(argv[i], "-v")==0) { verbosidad = TRUE; n++; }
   if (argc == n+1)
      if ((yyin = fopen (argv[n], "r")) == NULL)
       fprintf (stderr, "El fichero '%s' no es valido\n", argv[n]) ;
      else yyparse ();
   else fprintf (stderr, "Uso: cmc [-v] fichero\n");
   return (0);
}
```