Universidad Nacional Autónoma de México

Facultad de Ingeniería

Compiladores

Examen Parcial 3

Integrantes:
Díaz González Rivas Ángel Iñaqui
Perez Delgado Erandy Estefanya

Profesora:
Sáenz García Elba Karen

Grupo: 1

Semestre: 2025-1

Fecha de entrega: 02/Diciembre/2024

Desarrollar las siguientes actividades

Actividad 1: 35 puntos

Utilizando la gramática que define a las cadenas de la forma $a^n b^m c^m$ donde $n \geq 0$,m > 0

1: $S \rightarrow aS$

2: $S \rightarrow bAc$

3: A→bAc

4: A→b

Colocar a la gramática modificada los símbolos de acción necesarios para que obtenga como traducción cadenas de la forma $c^m b^m a^n$.

Gramática modificada:

0: F → S

1: S → aS{a}

2: S → b{c}A

3: A → b{c}Ac{b}

4: A → c{b}

Ejemplo: aabbcc

0 → S

1→ aS{a}

1→ aaS{a}{a}

2 → aab{c}A{a}{a}

3→ aab{c}b{c}Ac{b}{a}{a}

4→ aab{c}b{c}c{b}c{b}{a}{a}

Resultado: aabbcc Símbolos de acción: {c}{c}{b}{b}{a}{a}

Ejemplo: bbcc

0 → S

2 → b{c}A

3 → b{c}b{c}Ac{b}

4 → b{c}b{c}c{b}c{b}

Resultado: bbcc Símbolos de acción: {c}{c}{b}{b}

Elaborar un analizador léxico y sintáctico traductor utilizando flex y yacc/bison .El analizador debe recibir un archivo de entrada con la cadena a analizar y mostrar la traducción correspondiente.

Para poder correr los programas es necesario instalar flex (para la construcción del analizador léxico) y bison (para la fusión y conexión con el analizador sintáctico).

El código bison sería el siguiente:

```
// Inclusión de bibliotecas estándar necesarias para la funcionalidad del programa.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Declaración de funciones externas
void yyerror(const char *s);    // Función para manejar errores de Bison
int yylex(void);                // Función generada por Flex para el análisis léxico
extern FILE *yyin;               // Variable global para asociar el archivo de entrada
al analizador léxico
%}
```

Este bloque contiene las inclusiones de bibliotecas estándar necesarias para el funcionamiento del programa (como `stdio.h`, `stdlib.h`, `string.h`). También declara las funciones externas como `yyerror` (para manejo de errores) y `yylex` (generada por Flex para el análisis

léxico). Además, declara `yyin`, una variable global que se usa para asociar el archivo de entrada al analizador léxico.

```
%token a b c  // Definición de los tokens que serán utilizados en el analizador
```

Aquí se definen los **tokens** que serán reconocidos durante el análisis léxico. En este caso, los tokens `a`, `b`, y `c` se corresponden con caracteres específicos que serán utilizados en la gramática definida en la siguiente sección.

```
%%

F: S ;

S: 'a' S { printf("a"); }
  | 'b' { printf("c"); } A
  ;

A: 'b' { printf("c"); } A 'c' { printf("b"); }
  | 'c' { printf("b"); }
  ;

%%
```

Esta sección define la **gramática** que será utilizada para analizar la entrada. Especifica cómo se pueden combinar los tokens para formar frases válidas.

- `F` es el símbolo inicial, que se deriva en `S`.
- `S` es una regla que puede ser un `a` seguido de otra `S` o un `b` seguido de una producción `A`.
- `A` describe una serie de alternativas: puede ser un `b` seguido de otra `A` y un `c`, o simplemente un `c`.

Además, cada producción tiene **acciones asociadas** (por ejemplo, `printf("a");`), que se ejecutan cuando se detecta un patrón específico en la entrada.

```c
void manejar_error(const char *mensaje) {
    fprintf(stderr, "Se ha producido un error: %s\n", mensaje);
    exit(EXIT_FAILURE);
}
```

Esta función maneja errores generales que puedan ocurrir durante el procesamiento. Si se detecta un error (como no poder abrir el archivo de entrada), muestra un mensaje de error y termina la ejecución del programa.

```c
void procesar_archivo_linea_a_linea(const char *nombre_archivo) {
    FILE *archivo = fopen(nombre_archivo, "r");
    if (!archivo) {
        manejar_error("No se pudo abrir el archivo de entrada");
    }

    char buffer[256];
    while (fgets(buffer, sizeof(buffer), archivo)) {
        size_t longitud = strlen(buffer);
        if (longitud > 0 && buffer[longitud - 1] == '\n') {
            buffer[longitud - 1] = '\0';
        }

        if (strlen(buffer) == 0) {
            continue;
        }

        FILE *archivo_temporal = tmpfile();
        if (!archivo_temporal) {
            manejar_error("No se pudo crear el archivo temporal");
        }

        fputs(buffer, archivo_temporal);
        rewind(archivo_temporal);

        yyin = archivo_temporal;
        printf("\nProcesando Cadena: %s\n", buffer);
        printf("Simbolo de Accion: ");
        yyparse();

        fclose(archivo_temporal);
        printf("\n");
    }
```

```
    fclose(archivo);
}
```

Este bloque se encarga de **procesar el archivo de entrada línea por línea**. Abre el archivo y lo lee línea por línea, luego procesa cada línea en un archivo temporal para ser analizado por el analizador léxico y sintáctico (`yylex` y `yyparse`).

- **Cada línea leída** es procesada de manera independiente, eliminando los saltos de línea y saltando líneas vacías.
- **El archivo temporal** es utilizado para permitir que Flex lea la entrada línea por línea.
- Después de cada línea procesada, se llama a `yyparse()`, que realiza el análisis sintáctico con la gramática definida.

```
int main(int argc, char **argv) {
    if (argc != 2) {
        fprintf(stderr, "Uso incorrecto: %s <archivo_de_entrada.txt>\n", argv[0]);
        return 1;
    }

    procesar_archivo_linea_a_linea(argv[1]);

    return 0;
}
```

La **función principal** es el punto de entrada del programa.

- **Verifica los argumentos de entrada** para asegurarse de que se ha proporcionado un archivo de entrada.
- Si el archivo es válido, llama a la función `procesar_archivo_linea_a_linea` para iniciar el procesamiento del archivo.

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

Esta función es llamada por Bison cuando se detecta un error de sintaxis en el análisis. Simplemente imprime el mensaje de error proporcionado por Bison.

Se ejecuta de la siguiente forma:

```
C:\Users\Erand\OneDrive - UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO\7° SEMESTRE\Compiladores\EXAMEN3>flex Actividad1.l

C:\Users\Erand\OneDrive - UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO\7° SEMESTRE\Compiladores\EXAMEN3>bison -d Actividad1.y

C:\Users\Erand\OneDrive - UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO\7° SEMESTRE\Compiladores\EXAMEN3>gcc Actividad1.tab.c lex.yy.c -o Actividad1

C:\Users\Erand\OneDrive - UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO\7° SEMESTRE\Compiladores\EXAMEN3>Actividad1.exe cadenasAct1.txt
```

Salida:

```
Procesando Cadena: aabbcc
Simbolo de Accion: ccbbaa

Procesando Cadena: bbcc
Simbolo de Accion: ccbb

Procesando Cadena: abbcc
Simbolo de Accion: ccbba

Procesando Cadena: aabc
Simbolo de Accion: cbaa

Procesando Cadena: bc
Simbolo de Accion: cb

Procesando Cadena: bbbccc
Simbolo de Accion: cccbbb

Procesando Cadena: aabc
Simbolo de Accion: cbaa

Procesando Cadena: aa
Simbolo de Accion: syntax error
```

Actividad 2: 65 puntos

De la gramática que define las sentencias declarativas del lenguaje del analizador que se realizó como proyecto 2.

1)Sentencia declarativa:

Ejemplos:

caracter x_~'h ':

real a1_, a2_, b3x_~3.62:

Gramática:

D → <Tipo>L:

<Tipo>→ g

<Tipo>→n

<Tipo>→d

<Tipo>→h

L → a<Valor> I'

I'→,a<Valor> I'

I'→ ε

<Valor> → = V

<Valor>→ ε

V→c

V→s

V→z

V→r

A. Incluir los símbolos de acción y los atributos (con sus reglas de asignación por producción) para que el analizador semántico realice la actualización de la columna tipo en las variables que se declaren.

Gramática modificada:

D → <Tipo>L: {asignarTipo(Tipo)}

<Tipo>→ g {Tipo = "carácter"}

<Tipo>→n {Tipo = "entero"}

<Tipo>→d {Tipo = "real"}

<Tipo>→h {Tipo = "cadena"}

L → a<Valor> I' {asignarValor(a)}

I'→,a<Valor> I' {asignarValor(Valor)}

I'→ ε

<Valor> → = V {asignarValor(V)}

<Valor>→ ε

V→c {Valor = "caracter"}

V→s {Valor = "cadena"}

V→z {Valor = "entero"}

V→r {Valor = "real"}

B. Elaborar un analizador sintáctico traductor en yacc/bison de la gramática de traducción con atributos definida para que escriba qué tipo de variables se está declarando. Dada una o varias sentencias declarativas de entrada, escriba "Las variables son de tipo (indicar tipo)"

El código bison es el siguiente:

```
#include "sintac2.tab.h"  // Incluir la cabecera generada por Bison que contiene las definiciones de tokens y tipos.
#include <stdio.h>         // Incluir la biblioteca estándar para funciones de entrada y salida.
#include <stdlib.h>          // Incluir la biblioteca para funciones estándar como `exit()`.
#include <string.h>       // Incluir la biblioteca para manipulación de cadenas.

extern int yylex();       // Declaración de la función de análisis léxico generada por Flex.
extern char *yytext;       // Declaración de la variable que contiene el texto de la última coincidencia de token.
extern FILE *yyin;          // Declaración del archivo de entrada para el análisis léxico.

void yyerror(const char *s);   // Declaración de la función de manejo de errores de Bison.
void imprimir_tipo(char *tipo);   // Declaración de la función que imprime el tipo de una variable.
%}
```

Este bloque de código contiene la inclusión de bibliotecas necesarias (como las de entrada/salida, manejo de cadenas, etc.), así como las funciones y variables externas necesarias para que el analizador léxico y sintáctico funcione. Se incluye también el archivo generado por **Bison** (`sintac2.tab.h`), que contiene las definiciones de los tokens y tipos que se usan en las reglas de la gramática.

```
%union {
   char *str;  // Unión para almacenar cadenas (usada para tokens de tipo string).
}

%token <str> IDENTIFICADOR CARACTER ENTERO REAL CADENA c s z r    // Definición de tokens con sus tipos de valor.
%token ':'   // Token para el símbolo ":"
%token ','   // Token para el símbolo ","

%type <str> D tipo I_prima L Valor V  // Definición de los tipos de las reglas no terminales.
```

Aquí se definen los **tokens** que serán reconocidos por el analizador léxico (`IDENTIFICADOR`, `CARACTER`, `ENTERO`, etc.). Cada uno de estos tokens tiene un valor asociado (en este caso de tipo `char*` para representar cadenas).

También se declara un **`%union`** que indica que los valores semánticos de los tokens y no terminales se almacenarán como cadenas (strings). Además, se definen los **tipos de las reglas no terminales** que se utilizarán en la gramática.

```
%%

D: tipo L ':' {
   imprimir_tipo($1);   // Imprime el tipo de la variable (valor de tipo).
   printf("\n");
}
| /* vacío */ {
   printf("Cadena vacía detectada\n");  // Si se detecta una cadena vacía, se imprime
este mensaje.
   $$ = "";  // Asigna una cadena vacía a la variable semántica.
};

tipo: CARACTER { $$ = "CARACTER"; }   // Si se encuentra 'CARACTER', asigna el valor
"CARACTER".
    | ENTERO { $$ = "ENTERO"; }        // Si se encuentra 'ENTERO', asigna el valor
"ENTERO".
   | REAL { $$ = "REAL"; }           // Si se encuentra 'REAL', asigna el valor "REAL".
    | CADENA { $$ = "CADENA"; }        // Si se encuentra 'CADENA', asigna el valor
"CADENA".

L: IDENTIFICADOR Valor I_prima { $$ = $1; }   // Si se encuentra un identificador
seguido de su valor y posibles valores adicionales, asigna el identificador.

I_prima: ',' IDENTIFICADOR Valor I_prima { $$ = $2; }  // Si se encuentra una coma,
asigna el segundo valor (se continúa la lista).
      | { $$ = ""; }  // Si no se encuentra nada, asigna una cadena vacía.

Valor: '~'V { $$ = $2; }  // Si se encuentra el símbolo '~', asigna el valor V.
    | { $$ = ""; }  // Si no se encuentra nada, asigna una cadena vacía.
```

```
V: c { $$ = "c"; }  // Si se encuentra 'c', asigna el valor "c".
| s { $$ = "s"; }  // Si se encuentra 's', asigna el valor "s".
| z { $$ = "z"; }  // Si se encuentra 'z', asigna el valor "z".
| r { $$ = "r"; }  // Si se encuentra 'r', asigna el valor "r".

%%
```

Estas son las **reglas de producción** de la gramática, donde se describe cómo se combinan los tokens para formar construcciones válidas en el lenguaje que se está analizando. Aquí se definen varias reglas:

- **`D`**: Define cómo se declara una variable de un tipo específico.
- **`tipo`**: Define los posibles tipos de una variable (como `CARACTER`, `ENTERO`, etc.).
- **`L`**: Define cómo se maneja un identificador seguido de su valor.
- **`I_prima`**: Regla recursiva que maneja la continuación de una lista de identificadores separados por comas.
- **`Valor`** y **`V`**: Describen los valores posibles que una variable puede tomar, como `c`, `s`, `z`, `r`.

Cada regla puede incluir **acciones semánticas**, que son bloques de código que se ejecutan cuando se aplica una producción (por ejemplo, imprimir el tipo de variable).

```
void manejar_error(const char *mensaje) {
   fprintf(stderr, "Se ha producido un error: %s\n", mensaje);
   exit(EXIT_FAILURE);  // Finaliza el programa si ocurre un error.
}


void yyerror(const char *s) {
```

```
    fprintf(stderr, "Error de sintaxis: %s\n", s);  // Mensaje de error de sintaxis.

}
```

Estas funciones se encargan de mostrar mensajes de error. `manejar_error()` maneja errores generales (como problemas al abrir archivos), mientras que `yyerror()` maneja errores de sintaxis detectados por **Bison**.

```
void procesar_archivo_linea_a_linea(const char *nombre_archivo) {

    FILE *archivo = fopen(nombre_archivo, "r");

    if (!archivo) {

        manejar_error("No se pudo abrir el archivo de entrada");

    }



    char buffer[256];  // Buffer para almacenar cada línea.

    while (fgets(buffer, sizeof(buffer), archivo)) {

        size_t longitud = strlen(buffer);

        if (longitud > 0 && buffer[longitud - 1] == '\n') {

            buffer[longitud - 1] = '\0';  // Elimina el salto de línea al final.

        }



        if (strlen(buffer) == 0) continue;  // Si la línea está vacía, la omite.



        FILE *archivo_temporal = tmpfile();  // Crea un archivo temporal.

        if (!archivo_temporal) {

            manejar_error("No se pudo crear el archivo temporal");
```

```
        }



        fputs(buffer, archivo_temporal);  // Escribe la línea en el archivo temporal.

        rewind(archivo_temporal);  // Rewind para poder leer desde el inicio.



        yyin = archivo_temporal;  // Asocia el archivo temporal al analizador léxico.

        yyparse();  // Realiza el análisis sintáctico.



        fclose(archivo_temporal);  // Cierra el archivo temporal.

    }



    fclose(archivo);  // Cierra el archivo de entrada.

}
```

Esta función se encarga de abrir un archivo de entrada, leerlo línea por línea, y procesar cada línea. Cada línea se escribe en un archivo temporal, que luego es analizado por el analizador léxico y sintáctico. Este proceso permite que el programa analice cada línea independientemente.

El código se ejecuta de la siguiente manera:

```
angelinaquidiazgonzalezrivas@Air-de-Angel tercer % flex lex2.l
angelinaquidiazgonzalezrivas@Air-de-Angel tercer % bison -d sintac2.y
angelinaquidiazgonzalezrivas@Air-de-Angel tercer % gcc lex.yy.c sintac2.tab.c -L/opt/homebrew/Cellar/flex/2.6.4_2/lib -lfl

angelinaquidiazgonzalezrivas@Air-de-Angel tercer % ./a.out act2.txt
```

Archivo act2.txt de ejemplo

real num ~ 50, x:

caracter x_~'h ':

real a1_, a2_, b3x_~3.62:

entero a1_,b3x_ ~ 35515 :

real num ~ 50, x:

real x, num ~ 50:

caracter x_~'h ':

cadena letra ~"cadena hecha":

entero a1_,b3x_ ~35515:

real a1_,b3x_ ~ 35.515 :

Salida:

```
angelinaquidiazgonzalezrivas@Air-de-Angel tercer % ./a.out act2.txt

Procesando Cadena: real num ~ 50, x:
Token: REAL
Token: IDENTIFICADOR, Valor: num
Token: ~
Token: ENTERO, Valor: 50
Token: ,
Token: IDENTIFICADOR, Valor: x
Token: :
Las variables son de tipo REAL


Procesando Cadena: caracter x_~'h ':
Token: CARACTER
Token: IDENTIFICADOR, Valor: x_
Token: ~
Token: CARACTER, Valor: 'h '
Token: :
Las variables son de tipo CARACTER



Procesando Cadena: real a1_, a2_, b3x_~3.62:
Token: REAL
Token: IDENTIFICADOR, Valor: a1_
Token: ,
Token: IDENTIFICADOR, Valor: a2_
Token: ,
Token: IDENTIFICADOR, Valor: b3x_
Token: ~
Token: REAL, Valor: 3.62
Token: :
Las variables son de tipo REAL



Procesando Cadena:
Cadena vacía detectada


Procesando Cadena:
Cadena vacía detectada


Procesando Cadena: entero a1_,b3x_ ~ 35515 :
Token: ENTERO
Token: IDENTIFICADOR, Valor: a1_
Token: ,
Token: IDENTIFICADOR, Valor: b3x_
Token: ~
Token: ENTERO, Valor: 35515
Token: :
Las variables son de tipo ENTERO



Procesando Cadena:
Cadena vacía detectada
```

```
Procesando Cadena: real num ~ 50, x:
Token: REAL
Token: IDENTIFICADOR, Valor: num
Token: ~
Token: ENTERO, Valor: 50
Token: ,
Token: IDENTIFICADOR, Valor: x
Token: :
Las variables son de tipo REAL



Procesando Cadena: real  x, num ~ 50:
Token: REAL
Token: IDENTIFICADOR, Valor: x
Token: ,
Token: IDENTIFICADOR, Valor: num
Token: ~
Token: ENTERO, Valor: 50
Token: :
Las variables son de tipo REAL



Procesando Cadena: caracter x_~'h ':
Token: CARACTER
Token: IDENTIFICADOR, Valor: x_
Token: ~
Token: CARACTER, Valor: 'h '
Token: :
Las variables son de tipo CARACTER



Procesando Cadena: cadena letra ~"cadena hecha":
Token: CADENA
Token: IDENTIFICADOR, Valor: letra
Token: ~
Token: CADENA, Valor: "cadena hecha"
Token: :
Las variables son de tipo CADENA



Procesando Cadena: entero a1_,b3x_ ~35515:
Token: ENTERO
Token: IDENTIFICADOR, Valor: a1_
Token: ,
Token: IDENTIFICADOR, Valor: b3x_
Token: ~
Token: ENTERO, Valor: 35515
Token: :
Las variables son de tipo ENTERO
```

```
Procesando Cadena: real a1_,b3x_ ~ 35.515 :
Token: REAL
Token: IDENTIFICADOR, Valor: a1_
Token: ,
Token: IDENTIFICADOR, Valor: b3x_
Token: ~
Token: REAL, Valor: 35.515
Token: :
Las variables son de tipo REAL
```

Tabla de actividades

Integrantes que participaron	Actividades
Iñaqui y Erandy	Actividad 1 Colocar a la gramática modificada los símbolos de acción necesarios para que obtenga como traducción cadenas de la forma $c^m b^m a^n$.
Erandy	Actividad 1 Elaborar un analizador léxico y sintáctico traductor utilizando flex y yacc/bison .El analizador debe recibir un archivo de entrada con la cadena a analizar y mostrar la traducción correspondiente.
Iñaqui	Actividad 2 A Incluir los símbolos de acción y los atributos (con sus reglas de asignación por producción)

	para que el analizador semántico realice la actualización de la columna tipo en las variables que se declaren.
Iñaqui y Erandy	Actividad 2 B Elaborar un analizador sintáctico traductor en yacc/bison de la gramática de traducción con atributos definida para que escriba qué tipo de variables se está declarando. Dada una o varias sentencias declarativas de entrada, escriba “Las variables son de tipo (indicar tipo)”
Iñaqui y Erandy	Documento