1a Guía Compiladores

Nombre: **Rodríguez Martínez Heber Emanuel** Grupo: **3CM15** Fecha: **13 de septiembre de 2021**

**-Defina compilador**

Un compilador es un programa que lee un programa escrito en un lenguaje, el lenguaje fuente, y lo traduce a un programa equivalente en otro lenguaje, el lenguajeobjeto.

**-Cuales son las dos partes de la compilación**

Análisis y Síntesis

**-Describa las 6 fases de un compilador**

Analizador léxico, analizador sintáctico, analizador semántico, generador de código intermedio, optimador de código y generador de código.

**-Cuales son los 8 módulos de un compilador**

1. Administrador de la tabla de símbolos.
2. Analizador léxico.
3. Analizador sintáctico.
4. Analizador semántico.
5. Generador de código intermedio.
6. Optimizador de código.
7. Generador de código.
8. Manejador de errores**.**

**A partir de hoc4 se usan dos etapas en hoc. ¿Cuáles son y qué hacen?**

1.-Generación de código

2.- Ejecución

Falso o verdadero (F/V)

|  |  |
| --- | --- |
| 0.-A los terminales se les llama así porque no pueden ser sustituidos | ( V ) |
| 1.-Que es una secuencia de caracteres concreta sea un token depende del lenguaje | ( F ) |
| 2.-Las cadenas que pertenecen al lenguaje generado por una gramática están hechas solo de terminales | ( F ) |
| 3.-El **análisis léxico** lee la cadena de entrada de derecha a izquierda | ( V ) |
| 4.-El **análisis léxico** construye el árbol de análisis sintáctico | ( F ) |
| 5.-La secuencia de caracteres que forma un componente léxico es el **lexema** del componente | ( V ) |
| 6.-La gramática S → **a**S |S**a** | **a** se puede analizar con un **analizador sintáctico predictivo descendente recursivo** | ( V ) |
| 7.-El tipo de **yylval** no es el mismo que el de los elementos en la pila de YACC | ( V ) |
| 8.-La unica forma de indicar el tipo de los elementos en la pila de YACC es usando #define YYSTYPE | ( V ) |
| 9.-El **código intermedio** debe ser fácil de generar | ( V ) |
| 10.- Un **esquema de traducción** es una **GLC** + **reglas semánticas** | ( F ) |
| 11.- Árbol de análisis sintáctico con anotaciones es sinónimo de **árbol decorado** | ( V ) |
| 12-Análisis sintáctico **descendente** es donde la construcción del árbol de análisis sintáctico se inicia en las hojas y avanza hacia la raíz | ( F ) |
| 13-Análisis sintáctico **ascendente** es donde la construcción del árbol de análisis sintáctico se  inicia en las hojas y avanza hacia la raíz | ( V ) |
| *14.-*yylex() llama a yyparse() | ( F ) |
| 15.-yyparse() llama a yylex() | ( V ) |
| *16.-*yylex() retorna el **tipo** de **token** | ( V ) |
| 17.-yylval almacena el **lexema** | ( V ) |
| 18-HOC1 es una **calculadora** | ( V ) |
| 19-Las **variables** en HOC son de tipo entero | ( F ) |
| 20.-La notación **posfija** es una notación matemática libre de paréntesis y en esta notación los operadores aparecen después de los operandos | ( V ) |
| 21.-La **raiz** del árbol de análisis sintáctico se etiqueta con el símbolo inicial | ( V ) |
| 22.- Las **hojas** del árbol de análisis sintáctico se etiquetan con no terminales | ( F ) |
| 23.-En la notación **infija** la **asociatividad** y la **precedencia** se usan para determinar en que orden hay que realizar las operaciones para **evaluar** una expresion | ( V ) |

**Para que sirve el Análisis Léxico**

|  |  |  |
| --- | --- | --- |
| a) Para generar el código en lenguaje objeto | b) Nos dice si una cadena pertenece al lenguaje generado por una gramática | ( **c**  ) |
| c) Para dividir una cadena en tokens | d) Los compiladores no lo necesitan nunca |

**El \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ comprueba que el orden en que el analizador léxico le va entregando los tokens es válido.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) analizador semantico | b) analizador sintáctico | c) optimizador | d) generador de codigo | ( **b** ) |

**Es una gramática que tiene cuatro componentes:**

1. Un conjunto de componentes léxicos*.*

1. Un conjunto de no terminales.
2. Un conjunto de producciones, en el que cada producción consta de un no terminal, llamado *lado izquierdo* de la producción, una flecha y una secuencia de componentes léxicos y no terminales, o ambos, llamado *lado derecho* de la producción.
3. .La denominación de uno de los no terminales como símbolo *inicial.*

|  |  |  |
| --- | --- | --- |
| a) Gramática Asociativa por la izquierda | b) Gramática recursiva | (  **c**  ) |
| c) Gramática libre de contexto (GLC) | d) Gramática ambigua |  |

**¿Cuál de las siguientes opciones no es sinónimo de las otras?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Componente léxico | b) no terminal | c) token | d) Símbolo gramatical | ( **b**  ) |

**Es una gramática donde en el lenguaje que genera existe una cadena que tiene más de un árbol de análisis sintáctico.**

|  |  |  |
| --- | --- | --- |
| *a)* Gramática recursiva por la izquierda | *b) Gramática recursiva* | ( **d**  ) |
| c) Gramática libre de contexto | d) Gramática *ambigua* |  |

**Si una gramática contiene una regla de producción de la forma A *→* A α entonces es una**

|  |  |  |
| --- | --- | --- |
| *a)* Gramática recursiva por la izquierda | *b) Gramática ambigua* | ( **a**  ) |
| c) Gramática libre de contexto | d) ninguna de las anteriores |  |

**Considere la siguiente gramática**

*S→* ***0*** *S* ***1*** *|* ***01***

a) Mostrar una derivación de ***00001111***

**S→ 0S1**

**→ 00S11**

**→ 000S111**

**→ 00001111**

b ) Dibuje el árbol de análisis sintáctico para la entrada ***00001111***

Gráfico

Descripción generada automáticamente

Considere la siguiente gramática

S → **b**A

A → **b**B

B → **b**C

C → **ε**

a ) Mostrar una derivación de **bbb**

**S → bA**

**→ bbB**

**→ bbbC**

**→ bbb**

b ) Dibuje el árbol de análisis sintáctico para la entrada **bbb**

Gráfico, Gráfico radial

Descripción generada automáticamente

Considere la siguiente gramática

*S* → *A*

*A* → *A***+***A* | *B****++***

*B* → ***y***

a ) Mostrar una derivación de **y + + + y + +**

**S → A**

**→ A + A**

**→ B + + + B + +**

**→ y + + + y + +**

b ) Dibuje el árbol de análisis sintáctico para la entrada **y + + + y + +**

Forma

Descripción generada automáticamente con confianza media

Considere la siguiente gramática

*l→ l* ***,*** *d | d*

*d→* ***0*** *|* ***1*** *|* ***2*** *|* ***3*** *|* ***4*** *|* ***5*** *|* ***6*** *|* ***7*** *|****8*** *|* ***9***

a ) Mostrar una derivación de ***9,8,7,6,5,4,3,2,1,0***

**l → l, d**

**→ l, d, d**

**→ l, d, d, d**

**→ l, d, d, d, d**

**→ l, d, d, d, d, d**

**→ l, d, d, d, d, d, d**

**→ l, d, d, d, d, d, d, d**

**→ l, d, d, d, d, d, d, d, d**

**→ l, d, d, d, d, d, d, d, d, d**

**→ d, d, d, d, d, d, d, d, d, d**

**...**

**→ 9, 8, 7, 6, 5, 4, 3, 2, 1, 0**

b ) Dibuje el árbol de análisis sintáctico para la entrada ***9,8,7,6,5,4,3,2,1,0***

Gráfico

Descripción generada automáticamente

Dada la gramática

T={**a**, **b**, +, -, \*, /, (,)}, N={E, T, F} S={E}

P={ E->T | E+T | E-T

T->F | T\*F | T/F

F-> **a** | **b** | (E) }

y la cadena **(a+b)/b**

**a) Obtenga una derivación de dicha cadena**

F→ T

→ T / F

→ F / b

→ (E)/b

→ (E+T)/b

→ (T+F)/b

→(F+b)/b

→(a+b)/b

**b) Dibuje el árbol de análisis sintáctico que corresponde a la cadena mencionada**

Diagrama

Descripción generada automáticamente

**Análisis sintáctico predictivo descendente recursivo**

Considere la siguiente gramática

S →**a** | **(** S **)**

Escriba el analizador sintáctico predictivo descendente recursivo

**void parea(complex t){**

**if(preana == t) preana = sigcomplex();**

**else error();**

**}**

**void S(){**

**if(preana == ‘a’)**

**parea(‘a’);**

**else if(preana == ‘(’){**

**parea(‘(’);S();parea(‘)’);**

**}else**

**error();**

**}**

**void main(){**

**preana = sigcomplex();**

**}**

**Ambigüedad**

Demostrar que la siguiente gramática es ambigua

S → **a**S |S**a** | **a**

usando la cadena **aa**

**Se generan 2 A.A.S**

Un dibujo de una persona

Descripción generada automáticamente con confianza baja

Demostrar que la siguiente gramática es ambigua

A →A **x** B | **x**

B → **x** B | **x**

usando la cadena **xxxxx**

**Se generan 2 A.A.S**

Forma

Descripción generada automáticamente con confianza media

Demostrar que la siguiente gramática es ambigua

S→ **a** S **b** S | **b** S **a** S | **ε**

usando la cadena **abab**

**Se generan 2 o más A.A.S.**

**Imagen en blanco y negro

Descripción generada automáticamente con confianza baja**

Verificar si las siguientes gramáticas son ambiguas

S → S **+** S | S **-** S | **a**

S → S S **+** | S S **-** | **a**

**Esta cadena no es ambigua, ya que los terminales ‘+’ y ‘-‘, siempre van a estar al final de la cadena, lo que provoca que las cadenas generadas solo tengan un A.A.S.**

**Diagrama, Forma, Esquemático

Descripción generada automáticamente**

**Recursividad por** la **izquierda**

Para eliminar la recursividad por la izquierda

A → A**a** | **b**

se transforma en

A → **b** | **b**R

R → **a**R | **ε**

Ahora considere las siguientes gramáticas

A→ **1** | A **0**

y

S→ **(** L **)** | **a**

L→ L **,** S | S

Elimine la recursividad por la izquierda de dichas gramáticas.

**A→ 1 | A 0**

**Pasa a**

**A→ 1R | 1**

**R → 0R | ε**

**S→ ( L ) | a**

**L→ L , S | S**

**Pasa a**

**L → S R | S**

**R → , S R | ε**

**S → (L) | a**

Escriba el analizador sintáctico predictivo descendente recursivo para dichas gramáticas.

**1era gramática**

|  |
| --- |
| **void A(){  if(preana=='1R')  parea('1'); R();  elseif(preana=='1')  parea('1');  else  error(); } void R(){  if(preana=='0')  parea('0'); R();  else; }** |

**2nda gramática**

|  |
| --- |
| **void L(){  if(preana=='SR')  S(); R();  elseif(preana=='S')  parea('S');  else  error(); } void R(){  if(preana=='S')  S(); R();  else; } void S(){ if(preana=='(')  parea('('); L();parea(')');  elseif(preana=='a')  parea('a');  else  error();**  **}** |

Escriba la sección de reglas de la especificación de YACC para dichas gramáticas

**Definiciones dirigidas por la sintaxis**

|  |  |
| --- | --- |
| Producción | Regla Semántica |
| *sec →* **comienza** | *sec.x = 0*  *sec.y = 0* |
| *sec →* *sec1 instr* | *sec.x = sec1.x + instr.dx*  *sec. y = sec1.y + instr.dy* |
| *instr →* **este** | *instr.dx = 1*  *instr.dy = 0* |
| *instr →* **norte** | *instr.dx = 0*  *instr.dy = 1* |
| *instr →* **oeste** | *instr.dx = -1*  *instr.dy = 0* |
| instr → **sur** | *instr.dx = 0*  *instr.dy = -1* |

**Dibuje el árbol de análisis sintácticocon anotaciones para la sig cadena**

c n e ss oo nnn eee ssss oooo

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Escribir la sección de reglas de la especificación de yacc para calcular la posición final del robot.

Escriba una definición dirigida por la sintaxis para evaluar expresiones booleanas.

**Esquemas de traducción**

Escriba un esquema de traducción para convertir una expresión en:

|  |  |
| --- | --- |
| **infijo a postfijo**  **expr -> expr + termino { print (‘ + ’)}**  **expr -> expr - termino { print (‘ - ’)}**  **expr -> termino**  **termino -> 0 { print (‘ 0 ’)}**  **termino -> 1 { print (‘ 1 ’)}**  **…**  **termino -> 9 { print (‘ 9 ’)}** | **postfijo a infijo**  **expr -> expr { print (‘ + ’)} termino +**  **expr -> expr { print (‘ - ’)} termino -**  **expr -> termino**  **termino -> 0 { print (‘ 0 ’)}**  **termino -> 1 { print (‘ 1 ’)}**  **…**  **termino -> 9 { print (‘ 9 ’)}** |
| **infijo a prefijo**  **expr -> { print (‘ + ’)} expr + termino**  **expr -> { print (‘ - ’)} expr - termino**  **expr -> termino**  **termino -> 0 { print (‘ 0 ’)}**  **termino -> 1 { print (‘ 1 ’)}**  **…**  **termino -> 9 { print (‘ 9 ’)}** | **prefijo a infijo**  **expr -> + expr { print (‘ + ’)} termino**  **expr -> - expr { print (‘ - ’)} termino**  **expr -> termino**  **termino -> 0 { print (‘ 0 ’)}**  **termino -> 1 { print (‘ 1 ’)}**  **…**  **termino -> 9 { print (‘ 9 ’)}** |

Escriba un esquema de traducción para evaluar expresiones booleanas

Para cada esquema de traducción de arriba escriba la sección de reglas de la especificación de YACC

**Escritura de Gramaticas**

Escribir una gramática que genere todas las cadenas de longitud 4 formadas con los símbolos del alfabeto {a,b,c}

**T={a,b,c}**

**NT={I,X}**

**S={I}**

**I → X X X X**

**X → a | b | c**

|  |  |  |  |
| --- | --- | --- | --- |
| **Escribir una gramática que sirva para generar las siguientes cadenas** | | | |
| **Especie perro**  **Edad 1**  **Sexo macho**  **Tamaño grande**  **Colores negro , blanco**  **Soy rápido , activo, alegre**  **Aficiones correr, comer** | **Especie gato**  **Edad 2**  **Sexo macho**  **Tamaño mediano**  **Colores negro , blanco , café**  **Soy tranquilo , sociable**  **Aficiones dormir, parrandear, comer** | **Especie perro**  **Edad 2**  **Sexo hembra**  **Tamaño pequeño**  **Colores canela , gris**  **Soy fuerte , alegre, activo.**  **Aficiones aullar** | **Especie gato**  **Edad 2**  **Sexo macho**  **Tamaño grande**  **Colores blanco**  **Soy listo , obediente**  **Aficiones jugar, haraganear** |

**(┘ = Enter)**

**I → text I | ε**

**text → text ┘| text,**

**text→ Especie | 1 | 2 | … | haraganea**r

|  |  |  |  |
| --- | --- | --- | --- |
| 12.-Escribir una gramática que sirva para generar las siguientes cadenas | | | |
| Etiquetado Nerd  Nivel Junior  Sexo Hombre  Lenguajes Java , C , Logo  Aficiones programar, videogames, comics, hackear, googlear | Etiquetado Geek  Nivel Senior  Sexo Mujer  Lenguajes Pascal , Prolog , SQL  Aficiones chatear, videogames, programar | Etiquetado Nerd  Nivel Junior  Sexo Mujer  Lenguajes PHP , Perl, Java  Aficiones hackear, googlear, gotcha, dormir | Etiquetado Freak  Nivel Senior  Sexo Hombre  Lenguajes Ensamblador, C  Aficiones gotcha, dormir, chatear, comics |

**I → text I | ε**

**text → text ┘| text,**

**text→ Etiquetado | Nerd | … | comics**

**YACC**

.-Para qué sirve $$,

**Asigna valores de una producción**

.-Dentro de una accion gramatical $n se refiere

**Variable asociada al elemento n del lado derecho de una producción.**

1.-Los %% se usan para indicar

|  |  |  |
| --- | --- | --- |
| a)inicio de la sección de declaraciones | b)inicio de la sección de reglas | ( **b** ) |
| c)precedencia de los operadores | d)fin del código de soporte |

2.-%token sirve para indicar

|  |  |  |
| --- | --- | --- |
| a)inicio de la sección de declaraciones | d)los no terminales de la gramática | ( **d** ) |
| c)precedencia de los operadores | d)los terminales de la gramática |

3.-Como le indica el analizador léxico (yylex) al analizador sintáctico (yyparse) que ya no hay mas tokens en la entrada

|  |  |  |
| --- | --- | --- |
| a) retornando cero | b) retornando -1 | ( **d** ) |
| c) almacenando -1 en yylval | d) almacenando 0 en yylval |

4.-Una acción gramatical debe ir entre

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) comillas | b) paréntesis | c) corchetes | d) llaves | ( **d** ) |

5.-Considere la producción

S : S ' a' S 'b'

$4 a cual de los miembros del lado derecho de la producción se refiere?

|  |  |  |
| --- | --- | --- |
| a)la 'a' | b)la 1er S | ( **d** ) |
| c)la segunda S | d)la 'b' |

Si el codigo de yylex es el siguiente

int yylex() { return getchar(); }

de cuantos caracteres son los tokens

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) 0 | b) 1 | c) 2 | d) la cantidad de caracteres del token varia | ( **b** ) |

Considere la siguiente gramática (los terminales se indican en negritas)

L-> L**,** D | D

D-> **0** | **1**

Escriba la sección de reglas de la especificación de yacc para dicha gramática

**%%**

**L: L ‘,’ D**

**| D**

**;**

**D: 0**

**| 1**

**;**

**%%**

Escriba la especificación de yacc para la gramática

S → U | V

U → T**a**U | T**a**T

V → T**b**V | T**b**T

T → **a**T**b**T | **b**T**a**T | **ε**

**%%**

**S: U**

**| V**

**;**

**U: T ‘a’ U**

**| T ‘a’ T**

**;**

**V: T ‘b’ V**

**| T ‘b’ T**

**;**

**T /\* ε\*/**

**| ‘a’ T ‘b’ T**

**| ‘b’ T ‘a’ T**

**;**

%%

Escriba las acciones gramaticales para que imprima el numero de b's en la cadena de entrada

%{

/\*escriba el tipo de los elementos en la pila de yacc \*/

#define YYSTYPE

%}

%%

S : ’(’ B ’)’ {}

;

B : ’(’ B ’)’ { }

| D { $$=$1; }

;

D : { }

| ’b’ D { }

;

%%

Considere la siguiente gramática (los terminales se indican en negritas)

lista->lista **,** figura | figura

figura-> triangulo | cuadrilatero

triangulo-> **lado lado lado**

cuadrilatero-> **lado lado lado lado**

Escriba la sección de reglas de la especificación de yacc para dicha gramática y las acciones semánticas respectivas para que se imprima si un triangulo es equilátero y si un cuadrilátero es un cuadrado.

**%%**

**lista: lista ‘,’ figura**

**| figura**

**;**

**figura: triangulo**

**| cuadrilátero**

**;**

**triangulo: lado lado lado {if($1==$2 && $2==$3) printf(“Equilatero”);}**

**;**

**cuadrilátero: lado lado lado lado {if($1 == $2 && $2 == $3 && $3 == $4) printf(“Cuadrilatero”);}**

**;**

**%%**