RESUMEN PRIMER PARCIAL

Proceso de compilación

Involucra 4 etapas sucesivas:

- *Preprocesamiento:* El preprocesador toma el archivo fuente para dejar el archivo en condiciones para el proceso de compilación. Resuelve directivas para el compilador, elimina comentarios y empalma las líneas eliminando todos los saltos de línea y tabulaciones del código.
- *Compilación:* El compilador tomará el archivo fuente preprocesado para hacer un análisis léxico, sintáctico y semántico del mismo (fase de análisis) y luego generará un programa en lenguaje ensamblador (fase de síntesis).
- *Ensamblado:* Se encarga de traducir un fichero escrito en un lenguaje ensamblador a un fichero objeto que contiene código máquina, ejecutable directamente por el microprocesador.
- *Enlazado:* El enlazador agrupa los módulos objeto (que a menudo tienen extensiones como .o) , generados por el ensamblador, dando como salida un programa ejecutable que el sistema operativo puede cargar y ejecutar.

Para pasar de un programa fuente escrito por un humano a un archivo ejecutable es necesario realizar estas cuatro etapas en forma sucesiva.

Compilador: Programa que permite traducir el código fuente de un programa en lenguaje de alto nivel a otro lenguaje de nivel inferior.

Gramáticas Formales y Jerarquía de Chomsky

Lenguaje formal: Conjunto de palabras con una connotación sintáctica.

Gramática formal: Conjunto de producciones o reglas de reescritura que se aplican para obtener palabras. Genera todas las palabras de un lenguaje formal y no genera cadenas que estén fuera del lenguaje.

Producciones: Están formados por tres partes, el lado izquierdo, el lado derecho y el operador de producción. Requieren 3 tipos de símbolos:

- Productores
- Los que forman las palabras
- Metasímbolos

Producción-épsilon: Se escribe como S → ε y significa que "S es reemplazada por nada" o genera la palabra vacía.

Definición formal: Toda gramática formal debe poseer:

- V_N: Vocabulario de noterminales.
- V_T: Vocabulario de terminales (caracteres del alfabeto).
- P: Conjunto de producciones.
- S e V_N: Noterminal especial llamado "axioma" a partir del cual siempre deben comenzarse las producciones.

Jerarquía de Chomsky: Clasificación de 4 tipos de gramáticas formales que a su vez generan 4 tipos diferentes de lenguajes formales:

- Regulares o tipo 3.
- Independientes del contexto o tipo 2.
- Sensibles al contexto o tipo 1.
- Irrestrictas o tipo 0.

Gramática regular (GR): Sus producciones tienen las siguientes restricciones:

- El lado izquierdo debe tener un solo noterminal.
- El lado derecho debe estar formado por:
  - un solo terminal, o
  - un terminal seguido por un noterminal (o al revés, pero solo uno de los casos),
  - o "épsilon".

Se consideran equivalentes si generan el mismo lenguaje.

Lenguaje regular infinito: Se logran mediante las producciones recursivas, es decir, aquellas en las que el noterminal que figura en el lado izquierdo también figura en el lado derecho.

Gramática quasi-regular (GQR): Gramáticas similares a una GR en donde un conjunto de terminales es reemplazado por un no terminal. Abrevia la escritura de una GR y siempre puede ser reescrita como una GR.

Gramática independiente del contexto (GIC):

- No tienen restricciones con respecto a la forma del lado derecho de sus producciones.
- El lado izquierdo requiere que siga siendo un único noterminal.
- Toda GR también es una GIC, pero no toda GIC es una GR.

Gramática irrestricta: Son gramáticas más amplias que pueden contener cualquier cosa de cualquiera de los dos lados.

Gramática sensible al contexto: Es una gramática irrestricta cuya única condición es que la longitud de lado izquierdo debe ser menor o igual que la del lado derecho.

Derivación: Proceso que permite obtener cada una de las palabras de un Lenguaje Formal, aplicando las producciones. Diferentes formas de representarla:

- Horizontal: Utilizando el símbolo →.
- Vertical: Armando una tabla con una columna dedicada a cada derivación y una segunda columna con comentarios. Reemplazando únicamente de a un noterminal.
- Árbol: El axioma es la raíz de el mismo.

Lenguajes Regulares y Expresiones Regulares

Lenguaje regular: Un lenguaje es regular si puede ser generado por una gramática regular. Si un lenguaje formal es finito, entonces siempre es un lenguaje regular, pero hay lenguajes infinitos que son LRs y otros que no.

Expresión regular: Forma más precisa de representar a los LRs. Describe un conjunto de cadenas que son palabras.

Para *lenguajes regulares finitos*, se obtiene mediante la utilización de:

- los caracteres del alfabeto,
- el símbolo épsilon,
- el operador concatenación (. / no conmutativo) y
- el operador unión (+ / conmutativo).

Se dice que dos expresiones regulares son EQUIVALENTES si representan el mismo LR.

Operador potencia: Se utiliza para simplificar la escritura, lectura y comprensión. Ej. (a+b)3 = (a+b)(a+b)(a+b)

Para *lenguajes regulares infinitos*, se utilizan, además:

Clausura de Kleene (*): Se utiliza para representar estas repeticiones indeterminadas. Representa a la palabra vacía y a todas las palabras que se forman con la repetición de su operando.

Clausura positiva (+): Se ubica el símbolo + como supraíndice (a+) y cumple la misma función que el *, con la diferencia de que, en este caso, no se incluye la palabra vacía.

Autómatas finitos

Autómatas: Tienen la capacidad de reconocer a los lenguajes formales, es decir, reconocer o aceptar cada palabra del lenguaje y rechazar toda cadena que no es una palabra de ese lenguaje. Existen tres tipos de autómatas:

- Finitos: Reconocen a los lenguajes regulares.
- Finitos con pila: Reconocen a los lenguajes independientes del contexto.
- Máquina de Turing: Reconocen a los lenguajes sensibles al contexto y a los lenguajes irrestrictos.

Son útiles, por ejemplo, en la construcción de un módulo del compilador llamado "Analizador Léxico".

Autómata finito:

- Herramienta abstracta que se utiliza para reconocer un determinado LR.
- Recorre cada uno de los caracteres y por cada uno de ellos produce un *cambio de estado*.
- Si al terminar, se encuentra en un ESTADO FINAL (supraíndice +), entonces el AF ha reconocido la cadena, caso contrario, la cadena es rechazada.
- Poseen un ESTADO INICIAL (supraíndice -) único en el que se encuentra antes de comenzar su actividad.

- Tienen asociado un dígrafo llamado DIAGRAMA DE TRANSICIONES (DT) que permite visualizar el funcionamiento. Los nodos son los diferentes estados y las flechas son las transiciones, que estarán etiquetadas con símbolos del alfabeto.
- Se dice que tiene éxito si finaliza en un estado final.
- Se dice que fracasa por alguno de los siguientes motivos:
    - Lee todos los caracteres, pero finaliza su actividad en un estado no final.
    - No puede leer todos los caracteres de la cadena.
    - Llega al estado final pero la cadena tiene más caracteres.

Se dice que son *determinísticos* si:

- Para cualquier estado en que se encuentre el autómata en un momento dado, la lectura de un carácter determina, sin ambigüedades, cuál será el estado de llegada en la próxima transición.
- Cada estado del AF tiene cero o una transición por cada carácter del alfabeto reconocido.

Definición formal: Un AFD contiene:
- Q: Conjunto finito de estados.
- $\sum$: Alfabeto de caracteres.
- q_0: Estado inicial.
- F: Conjunto de estados finales.
- T: Q x $\sum$ → Q: Función de transiciones. Se representa mediante una TABLA DE TRANSICIONES (TT).

Se dice que es *completo* si cada estado tiene exactamente una transición por cada carácter del alfabeto. Su tabla de transiciones no tiene huecos. Para completar una TT, se siguen los siguientes pasos:

1. Se agrega un nuevo estado llamado "*Estado de rechazo*".
2. Se reemplaza cada hueco por una transición a este nuevo estado.
3. Se incorpora una nueva entrada para el estado de rechazo en la que se representaran ciclos para todos los caracteres del alfabeto.

Sintaxis y BNF

Lenguaje de programación (LP): Notación utilizada para describir algoritmos y estructuras de datos que resuelven problemas computacionales. Están formados por un conjunto de LRs y un conjunto de GICs.
- Tokens: Identificadores, números enteros, números reales, constituyen diferentes LRs. Algunos finitos y otros infinitos. Es la unidad mínima indivisible de un lenguaje de programación.
- Categorías Sintácticas: Las expresiones y sentencias son en general LICs. No pueden ser generados por GRs, sino que requieren ser generados por GICs.
- La sintaxis de un LP debe describirse con precisión utilizando una notación sin ambigüedades.

Identificador: Elemento más utilizado en todo LP. Secuencia de uno o más caracteres que nombra diferentes entidades. La descripción de estos siempre posee ambigüedades al hacerla utilizando un lenguaje natural, por lo que es más útil utilizar una GIC.

```
Identificador -> Letra |
                 Identificador Letra |
                 Identificador GuiónBajo Letra
GuiónBajo -> _
Letra -> A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
         P | Q | R | S | T | U | V | W | X | Y | Z
```

- Una GIC para expresiones no solo debe producir todas las expresiones válidas, sino que también debe preocuparse por la precedencia y la asociatividad.
- Las PRIORIDADES de los operadores en una GIC se pueden ver analizando que tan cerca del axioma se encuentran. Mientras más cerca, entonces menor es la prioridad.
- La ASOCIATIVIDAD de la operación "Identificador" es a izquierda porque se encuentra del lado izquierdo. (Para evaluar, en el caso de tener la expresión 1-2-3, si es a izquierda, primero se hace 1-2 y si es a derecha, primero se hace 2-3).

Evaluación: Proceso inverso al de la derivación, que llega al axioma. Debemos hacer una reducción de la tabla, siempre en el orden inverso a la derivación. Se construye la "TABLA DE EVALUACIÓN". Es indispensable PARTIR DE LA CORRESPONDIENTE TABLA DE DERIVACIÓN y proceder en orden inverso.

Expresiones:
- Secuencia de operandos y operadores más el posible uso de paréntesis.

- No es suficiente con definir una BNF para estas mismas, sino que también debemos agregarles RESTRICCIONES. Si no las tenemos en cuenta, podríamos llegar a conclusiones absurdas, por ejemplo, que 1 = 2.
- La BNF debe representar correctamente la precedencia y la asociatividad de los operadores.

Objeto: Región de la memoria compuesta por una secuencia de uno o más bytes con la capacidad de contener la representación de valores.

Lvalue: Son expresiones que hacen referencia a ubicaciones de memoria. Aquellos que hacen referencia a ubicaciones modificables se denominan "valores L modificables".

Definición: Provoca una reserva de memoria para el objeto o la función definidos.

Sentencias: Especifican las acciones que llevará a cabo la computadora en tiempo de ejecución.

Expresiones Regulares (ER) y Expresiones Regulares Extendidas (ERX)

Expresión regular universal (ERU): Es la ER que representa al Lenguaje Universal sobre un alfabeto dado, es decir, a aquel que contiene la palabra vacía y todas las palabras que se pueden formar con caracteres del alfabeto dado.

La ER original se construye utilizando solo los operadores básicos unión (+), concatenación (.) y clausura de Kleene y se define:

- $\emptyset$ es una ER que representa al LR vacío.
- ε es una ER que representa al LR que solo contiene la palabra vacía.
- Todo carácter x de un alfabeto corresponde a una ER x que representa a un LR que solo tiene una palabra formada por ese carácter x.
- Una cadena s es una ER s que representa a un LR que solo contiene la palabra S.

Toda ER más compleja se construye de esta manera:

- Si R_1 y R_2 son ERs, entonces $R_1 + R_2$ es una ER.
- Si R_1 y R_2 son ERs, entonces $R_1 * R_2$ es una ER.
- Si R_1 es una ER, entonces R_1* es una ER.
- Si R_1 es una ER, entonces (R_1) es una ER.

Esta es la definición formal oficial de las ERs. Agregando dos operadores más podemos simplificar aún más su escritura:

- Si R_1 es una ER, entonces $R1^+$ es una ER.
- Si R_1 es una ER, entonces $R1^n$ (con $n \geq 0$ y entero) es una ER.

La precedencia de los operadores es la siguiente:

1. Los tres operadores unarios tienen prioridad máxima.
2. El operador "concatenación" tiene prioridad media.
3. El operador "unión" tiene prioridad mínima.

Como los lenguajes formales son conjuntos, las operaciones de conjuntos, también se pueden aplicar a los LRs, sumados a las propias operaciones de los Lenguajes Formales.

Unión: Sean R_1 y R_2 dos LRs. La unión se escribe como $L_1 \cup L_2$ y es un LR que contiene todas las palabras que pertenecen a cualquiera de los dos LRs. Es representada por la ER $R_1 + R_2$.

Concatenación: Se escribe como $(L_1.L_2)$ y es un LR en el que cada palabra está formada por la concatenación de una palabra de L_1 con una palabra de L_2. La cardinalidad es el producto de las cardinalidades. Es representada por la ER R_1R_2.

Clausura de Kleene: Si L es un LR, su clausura de Kleene (L*) es un LR infinito formado por la palabra vacía, las palabras de L y todas aquellas palabras que se obtienen concatenando las palabras de L. Si L es representado por la ER R, L* es representado por $R*$.

Clausura Positiva: Si L es un LR, su clausura positiva (L^+) es un LR formado por las palabras de L y todas aquellas palabras que se obtienen concatenando palabras de L. Si L es representado por R, L^+ es representado por R^+.

Complemento: Se escribe como L^c y es un LR que está formado por todas aquellas palabras que no pertenecen al LR original.

Intersección: Es un LR constituido por todas aquellas palabras que pertenecen a los dos lenguajes dados.

Los componentes léxicos de un LP constituyen diferentes LRs, por lo que los podremos representar mediante ERs.

Metalenguaje: Es un lenguaje que se usa para describir otro lenguaje. Agrega símbolos que ayudan a la representación llamados METACARACTERES. El lenguaje de las ERs está formado por:

- Operandos (caracteres del alfabeto).
- Operadores.
- Ciertos caracteres especiales, llamados metacaracteres, que colaboran en la descripción.
- Paréntesis para agrupar.

Deben cumplir las siguientes dos características:

- No pueden existir subíndices ni supraíndices.
- Los metacaracteres y los operadores deben representar las operaciones básicas para la escritura de ERs más otras operaciones que simplifican la escritura.

A los ERs escritos en este metalenguaje los llamaremos metaERs.

Metacaracteres Operadores	Explicación y Ejemplo
. (punto)	Se corresponde con cualquier carácter, excepto el "nueva línea" (\n). *Ejemplo*: a.a representa cualquier cadena de tres caracteres en la que el primer carácter y el tercer carácter son a.
| (barra vertical)	Es el operador *unión* de ERs. *Ejemplo*: ab|b representa la ER ab+b.
[ ] (corchetes)	Describe un conjunto de caracteres. Simplifica el uso del operador "unión". *Ejemplo*: [abx] representa la ER a+b+x.
[–]	Describe una clase de caracteres en uno o más intervalos. *Ejemplo 1*: [a–d] representa la ER a+b+c+d. *Ejemplo 2*: [0-9a-z] representa cualquier dígito decimal o cualquier letra minúscula (del alfabeto inglés).
{ } (llaves)	Es el operador *potencia*, una repetición *determinada* del patrón que lo precede como operando. *Ejemplo 1*: a{3} representa la ER aaa. *Ejemplo 2*: (ab){4} representa la ER abababab.
{,}	Es el operador *potencia* extendido a un intervalo. *Ejemplo*: a{1,3} representa la ER a+aa+aaa.
?	Es un operador que indica cero o una ocurrencia de la ER que lo precede. *Ejemplo*: a? representa la ER a+ε.
*	Es el operador *clausura de Kleene*: cero o más ocurrencias de la ER que lo precede. *Ejemplo*: a* representa la ER a^*.
+	Es el operador *clausura positiva*: una o más ocurrencias de la ER que lo precede. *Ejemplo*: a+ representa la ER a^+.
( ) (paréntesis)	Se utilizan para agrupar una ER. *Ejemplo*: ((ab)? | b)+ representa la ER $(ab+\varepsilon+b)^+$.

A esta tabla le agregamos la utilización del carácter \ para representar caracteres "no imprimibles" y también para convertir y usar como caracteres a los metacaracteres.

Puntuadores y Operadores de C

Carácter ',': Es utilizado como **carácter de puntuación** para separar la lista de elementos en la invocación de una función, para separar elementos al inicializar un arreglo o estructura de datos o para separar la declaración de variables.

```c
int a=3,b,c=5;
//Aquí está separando las declaraciones de variables

char array[3]={'a','b','c'};
//Aquí separa los elementos de un arreglo

Persona alumno={"Santiago","Ferreiros",32,64.5};
//Aquí se separan los elementos que componente la estructura

printf("Nombre: %s \n",alumno.nombre);
printf("Apellido: %s \n",alumno.nombre);
printf("Edad: %d \n",alumno.edad);
printf("Peso: %f \n",alumno.peso);
//Aquí estamos seprando los parametros al invocar a la función printf
```

Es utilizado como **operador** en expresiones separadas por coma. Pueden mezclarse ambos usos, pero se debe poner entre paréntesis para distinguir entre ellos.

```c
for (i = 0, j = 8 ; i + j < 9;  i++, j /= 2)
    printf("El valor de i es: %d y el valor de j es: %f \n",i,j);

/*Aquí pueden verse expresiones separadas por coma dentro de
la estructura de la sentencia for */
```

Carácter '[' ']': Se comportan como un **operador** de acceso a los elementos de un arreglo. O bien se comportan como un **carácter de puntuación** en la declaración de variables de tipo arreglo, ya que en este caso no estamos accediendo sino definiendo el tamaño del arreglo.

```c
int a[5]={56,76,43,32,11};
/* Aquí los corchetes están actuando como caracter de puntuación
definiendo el tamaño del arreglo*/

int b= a[1];
//Aquí está actuando como operador de acceso al elemento del arreglo
```

Carácter '(' ')': Los paréntesis se comportan como un **operador** en la invocación de funciones, para la agrupación de expresiones cuando es necesario cambiar la prioridad de los operadores y para la conversión de tipos (typecasting). Por otro lado, se comporta como un **carácter de puntuación** a la hora de declarar y definir funciones, ya que en este caso no las estamos invocando.

```c
int sumar(int,int);

/* Declaración de prototipo de la función sumar,
aquí actúa como caracter de puntuación */

int sumar(int a, int b){
    return a+b;
}

/* Definición de la función sumar, aquí actua como
caracter de puntuación */
```

```c
int a = sumar(1,3);

/* Aquí los paréntesis () actúan como operador
en la invocación de la función sumar */

int b = (3+5)*2;

/* Aquí los paréntesis () acutúa como un operador
para agrupar la expresiones y modificar la precedencia
de los operadores */

float c = (float) b;

/* Aquí los paréntesis actúan como un operador
conversión de datos explícita (typecasting)*/
```

Carácter '=': El carácter "=" se considera un **carácter de puntuación** en los siguientes casos particulares:

1. Definir una constante de enumeración.
2. Inicializar las variables en una declaración.

En otros casos, actúa como un operador. Por ejemplo, en una sentencia de asignación.

```c
int main()
{
    enum colors {Blue = 1, Red = 2, Green = 4, Light = 8};
    return Red;
}
```

```c
int main()
{
    int array[5] = { 1, 2, 3, 4, 5 };
    char *name = "Fred";
    int x = 12;
    return x;
}
```

Operador Incremento "++" / Decremento "--": El operador de incremento puede ser utilizado como postincremento o como preincremento. Puede aplicarse sobre expresiones que referencien a tipos de datos aritméticos o punteros y deben ser valorL modificables. El incremento siempre es acorde al tipo de operando.

- *expr ++ (postincremento):* el valor de la expresión se calcula de forma previa a realizar el incremento
- *++ expr (preincremento):* lo primero que sucede es que el operador es incrementado en 1, antes de que la expresión sea evaluada.

Operadores "&&" y "||" : Reciben dos expresiones como operandos de tipo escalar. Se van evaluando las expresiones de izquierda a derecha.

expr1 && expr2
expr1 || expr2

Si expresión1 vale 0 (False), y se trata de un operador &&, de forma automática expr1 && expr2 vale 0 (False) , y además la expresión2 no es evaluada.

De modo similar, si expresión1 vale 1 (True), y se trata de un operador ||, de forma automática expr1 || expr2 vale 1 (True), y además la expresión2 no es evaluada.

Operador ternario condicional '?' : Expr1 debe ser de tipo escalar dado que deberá tener que poder asociarse a un valor de verdad (True o False/0). Siempre se evalúa primero expresión1. Si el valor de Expr1 es distinto de cero (True), entonces Expr2 es evaluado y Expr3 es ignorado por completo. Caso contrario, si el valor deExpr1 es cero (False), entonces Expr3 es evaluado y Expr2 se ignora por completo.

Expr1 ? Expr2 : Expr3

Operador unario '&' y '*': El carácter '&', llamado Operador de referenciación , sigue la siguiente sintaxis:

&exp

Retorna como resultado la dirección de la expresión exp. Para que pueda operar, el operando exp debe ser de tipo lvalue (no necesariamente modificable), ya que debe designar a un objeto alojado en memoria que posea una dirección. O bien el operando exp puede estar designado a una función. Este último caso se verá más adelante.

El carácter '*' como operador de desreferenciación posee la siguiente sintaxis:

*exp

El operador de indirección retorna el objeto apuntado por exp. Para que pueda operar, el operando exp debe ser de tipo puntero.

Caso particular: Tener presente que el carácter '*' se considera como carácter de puntuación para la creación / declaración de tipos de datos puntero. Por ejemplo int* p.

Puntuador Ellipsis "...": Puede ser utilizado para representar un número variable de parámetros que recibe la función en su declaración y definición. También es utilizado para denotar rangos de valores enteros dentro de una sentencia case o al inicializar elementos de un arreglo.

Operadores a nivel bit '&' , '|' , '^': Dichos operadores trabajan bit a bit con los operandos. Ejemplo:

```c
int a= 6; //110
int b= 5; //101

printf("La operación bit a bit entre entre a y b para el operador & es: %d \n",a&b);

/* 110 & 101 = 100 , representa el valor 4
Recordar que es como si se tratase del AND, deben tener ambos bit valor 1
para que el resultado para dicha posición de bit sea 1. */

printf("La operación bit a bit entre entre a y b para el operador | es: %d \n",a|b);

/* 110 | 101 = 111 , representa el valor 7
Recordar que es como si se tratase del OR inclusivo, con que uno de los dos bits
evaluados sea 1, el resultado para dicha posición de bit es 1. */

printf("La operación bit a bit entre entre a y b para el operador ^ es: %d \n",a^b);

 /* 110 | 101 = 011 , representa el valor 3
Recordar que es como si se tratase del OR exluyente, para que el resultado para
dicha posición de bit sea 1, úncamente uno de los dos bits evaluados debe ser 1. */
```

Semántica

Definición de un LP: Documento que debe cumplir las siguientes funciones:

- Manual de referencia para los programadores que debe describir el LP con precisión y sin ambigüedades.
- Especificar la semántica y las restricciones de cada constructo.

- Punto de partida para quienes desarrollen compiladores.

Tipo: Significado de un valor almacenado en un objeto o retornado por una función. El conjunto de tipos básicos incluye a los char, short int, int, long int, float, double, long double, etc.

Identificador: Un identificador denota un objeto, una función y otras entidades.

Constantes: Cada constante tiene un tipo, determinado por su forma y su valor. Dada una lista de tipos predeterminada, el tipo de una constante entera es el primero en el que su valor puede ser representado.

Reales: Estas pueden ser:
- Constantes reales en Punto Fijo.
- Constantes reales en Punto Flotante (mantisa, exponente).

En general, una constante real tiene una parte significativa (número racional en base 10), que puede estar seguida de una parte exponente (entero en base 10) y de un sufijo real que especifica su tipo. No siempre es exacta, sino la representación más cercana a la exacta. La diferencia entre los tipos se basa en su *precisión* y su *rango*.

Carácter: Delimitada por apóstrofos y su valor es de tipo int. Esta misma es la representación numérica de ese carácter en una tabla como la tabla ASCII.

Literal cadena: Secuencia de cero o más caracteres delimitada por apóstrofos.

Operadores: Especifica la realización de una operación que producirá un valor. Un *operando* es la entidad sobre la que actúa un operador.

Caracteres de puntuación: Es un símbolo que tiene un significado semántico determinado, no especifica una operación y no produce un valor. Dependiendo del contexto, el mismo símbolo puede representar un carácter de puntuación o un operador.

Expresiones: Una expresión puede tener varias aplicaciones, como el cálculo de un valor.

Arreglo: Un objeto arreglo se suele expresar como un identificador y expresiones entre corchetes.

Expresiones Booleanas: El resultado de su evaluación debe representar verdadero o falso. La expresión debe tener un operador de relación o un operador booleano.

Expresión de asignación: El valor del operando derecho es convertido al tipo del operando izquierdo, que debe ser un ValorL.

Sentencias: Una sentencia especifica la realización de una acción.

Sentencias Compuestas: Una sentencia compuesta, también llamada bloque, permite que un grupo de sentencias sea tratado como una unidad. Puede tener su propio conjunto de declaraciones.

Formato de la Condición en las Sentencias de Selección e Iterativas: Cualquier expresión puede ser utilizada como condición. Si el valor es distinto de cero, entonces la condición es verdadera y si el valor es cero, la condición es falsa.

Una SECUENCIA DE TOKENS puede generar una expresión, una sentencia o bien una declaración.

Secuencia de caracteres → Categorías léxicas (tokens)
Secuencia de tokens → Categorías sintácticas

Análisis Léxico y Flex

Scanner: Es el encargado de realizar un análisis léxico de un programa. lo lleva a cabo mediante un análisis secuencial de caracteres, que lee de un archivo de entrada, identificando categorías léxicas/tokens. Dicha secuencia de tokens será procesada posteriormente por el parser en la fase del análisis sintáctico identificando una estructura denominada “sentencia de declaración”.

Flex: Herramienta que genera scanners de forma automática a través de un archivo de especificación que le permita saber que se necesita reconocer (categorías léxicas). Dicho archivo de especificación describe los patrones utilizando expresiones regulares extendidas (ERX / metaER) para describirlos.

¿Cómo funciona?

Primero, FLEX lee la especificación del scanner de un archivo de entrada con extensión .L y genera como salida un archivo .c con el nombre lex.yy.c.

Luego, este archivo lex.yy.c es compilado y enlazado con la librería libfl.a para producir un archivo ejecutable .exe. Finalmente, dicho ejecutable realiza la lectura de un archivo de entrada caracter a caracter produciendo una secuencia de tokens como salida.

Formato del archivo:

%{
DEFINICIONES Y DECLARACIONES DE C
%}

DEFINICIONES:

%%

REGLAS (PRODUCCIONES GIC - CÓDIGO C) -> Conocido como PATRÓN/ACCIÓN

%%

CÓDIGO C DE USUARIO -> Código C que ejecuta el analizador léxico

Variables

- **yytext**: Es un puntero a la cadena que coincidió con la regla actual. Va variando a lo largo del tiempo dado que el analizador léxico estará apuntando a distintas cadenas a medida que avanza en el análisis ya que ingresará por distintas reglas.

- **yyin**: Es una variable de tipo puntero que apunta a la cadena de entrada a ser procesada. Puede ser un archivo de texto completo.

- **yyout**: Es una variable de tipo puntero que apunta a una cadena en la cual queda toda la secuencia de entrada que no pudo aplicar a ninguna de las reglas especificadas.

Complemento e Intersección de AFDs

Los algoritmos descriptos no pueden ser aplicados a AFNs.

Complemento: Es un AFD que se obtiene invirtiendo los estados finales y no finales. Por lo tanto, la única diferencia en la definición de mi autómata va a ser el *conjunto de estados finales*. Es muy útil cuando debemos hallar un AFD que reconozca un LR cuyas palabras no tienen cierto patrón de caracteres.

Intersección: Es un AFD que reconoce las palabras comunes a ambos LRs. Sus estados son *pares ordenados de estados*.

Buena técnica para armar el AFD:
1. Obtener el estado inicial y colocarlo en la primera fila de la TT.
2. Determinar las transiciones desde el estado inicial y agregar nuevas entradas en la tabla a medida que surjan nuevos estados.
3. Repetir este proceso para cada nuevo estado.
4. Los únicos estados finales son aquellos en lo que ambos estados del par ordenado son finales.

Se dice que dos o más estados tiene el mismo comportamiento si todos son estados no finales o todos son estados finales y, además, sus filas son idénticas. Esto significa que son EQUIVALENTES y, por lo tanto, es suficiente con que exista uno de ellos y podemos simplificar la TT.

Pasaje de AF a ER: A partir del AF se genera un sistema de ecuaciones que al resolverse obtenemos la ER correspondiente al AF.

- Cada estado del autómata posee una única ecuación asociada: qi = ...
- cada estado final posee "+ ε" en el lado derecho de su ecuación.
- Cada transición (carácter) desde un mismo estado se separa dentro de la ecuación con la operación + (unión de ER).

Cada ESTADO es pensado como una incógnita, el resto son expresiones regulares ER. Es decir que cada estado, al despejarse del sistema de ecuaciones va a quedar igualado a una expresión regular.

Se deben ir despejando los ESTADOS en función de los otros, hasta despejar por último el estado inicial 0. Allí obtendremos la ER que representa las palabras que reconoce el AFD.

Autómatas Finitos con Pila

Autómatas finitos con pila (AFPs): Reconocen a los Lenguajes Regulares y también a los LICs, por lo que son más poderosos que los AFs. Además de tener estados y transiciones, tiene una memoria en forma de PILA que permite almacenar, retirar y consultar cierta información.

Un AFP está constituido por:
1. FLUJO DE ENTRADA: Infinito en una dirección, contiene la secuencia de caracteres que debe analizar.
2. CONTROL FINITO: Formado por estados y transiciones etiquetadas.
3. PILA ABSTRACTA: Se representa como una secuencia de símbolos o caracteres tomados de cierto alfabeto, cuyo primer carácter se encuentra en el tope de esta.

Entonces, un AFP queda formalmente definido como M = (E, A, A', T, e0, p0, F), en donde:
- E es el conjunto finito de estados.
- A es el alfabeto de entrada.
- A' es el alfabeto de pila.
- e_0 es el estado inicial.
- P_0 es el símbolo inicial de la pila.
- F es el conjunto de estados finales.
- T es la función T: E x (A ∪ { ε }) x A' → conjuntos finitos de E x A'*.

Ejemplo de notación de la función T:

$$T(4,a,Z) = \{ (4, RPZ), (5, \varepsilon) \}$$

Si el AFP se encuentra en el estado 4, en el tope de la pila tiene el símbolo Z y lee el carácter a → Se queda en el estado 4 y agrega RP a la pila o se mueve al estado 5 y quita el símbolo Z.

Para procesar la pila, se realizan dos acciones:
1. El AFP realiza un pop del símbolo que está en el tope.
2. El AFP realiza un push de los símbolos indicados. ε significa que no agrega símbolos a la pila, sino que solo hace el pop.

Autómata Finito con Pila Determinístico (AFPD): Un AFPD contiene en su definición los mismos 7 elementos anteriores y también un "flujo de entrada" infinito en una dirección que contiene la cadena a analizar.

Existen dos formas de movimiento:

Primera forma de movimiento: T (e,a,R) = (e', α) indica que, si el AFPD se encuentra en el estado e, lee el símbolo a y tiene el símbolo R en el tope de la pila, entonces pasa al estado e' y reemplaza, en la pila, el símbolo R por la secuencia de símbolos α, además, adelanta una posición en el flujo de entrada.

Segunda forma de movimiento: T (e, ε, R) = (e', α) indica que, si el AFPD se encuentra en el estado y tiene el símbolo R en el tope, entonces pasa al estado e' y reemplaza en la pila el símbolo R por la secuencia de símbolos α, además, no adelanta ninguna posición en el flujo de entrada.

PARA QUE SEA DETERMINISTICO, SOLO DEBE EXISTIR UN TIPO DE MOVIMIENTO PARA CUALQUIER PAR (e,R)

Tabla de Movimientos (TM): Similar a la Tabla de Transiciones, pero contiene el par "estado y símbolo en el tope de la pila".

Siempre que se puede construir un AFPD que reconozca por estado final, se podrá construir un AFPD que reconozca por pila vacía, y viceversa.

TM	a	b	fdc
e0,$	e1,R$		
e1,R	e1,RR	e2,ε	
e2,R		e2,ε	
e2,$			e3,$

por estado final

TM	a	b	fdc
e0,$	e1,R$		
e1,R	e1,RR	e2,ε	
e2,R		e2,ε	
e2,$			e2,ε

por pila vacía

Análisis sintáctico

- Llevado a cabo por el Parser, que verifica si los tokens forman secuencias o construcciones validas.
- Convierte el flujo de tokens en un árbol de análisis sintáctico. Representa la secuencia analizada, en donde los tokens que forman la construcción son las hojas del árbol.

Al derivar una secuencia de tokens, si existe más de un noterminal debemos elegir cuál es el próximo noterminal que se va a expandir. Existen dos tipos de derivación: a IZQUIERDA o a DERECHA. Gracias a estas, podemos "obtener" dos tipos de análisis sintácticos:

- Análisis Sintáctico Descendente: produce una derivación por Izquierda que comienza por el axioma y finaliza con los terminales que forman la construcción analizada.
- Análisis Sintáctico Ascendente: Utiliza la derivación a derecha, pero en un orden inverso, es decir, que la última producción aplicada en la derivación a derecha es la primera producción utilizada y la que involucra al axioma es la última producción en ser descubierta.

Hay reglas sintácticas que no pueden ser expresadas empleando únicamente GICs, es por eso, que son considerados parte de la semántica estática y son chequeados por rutinas semánticas.

Análisis semántico

- Se encarga de la realización de tareas de verificación y de conversión que no puede realizar el Análisis Sintáctico.
- Debe verificar que se cumplan todas las reglas sensibles al contexto.
- Utiliza mucho la Tabla de Símbolos.

Algunos ejemplos de uso:

- Definir si una variable utilizada es del tipo correcto.
- Definir si una variable fue declarada antes de su utilización.
- Definir si existen declaraciones múltiples en un mismo bloque.

Tipos de errores

Errores Léxicos:

- Asociados al análisis léxico.
- El compilador realiza una lectura carácter a carácter para reconocer los tokens. Asociado a la búsqueda de las palabras que pertenezcan al lenguaje regular que reconoce cada categoría.
- El error está asociado a cuando el AFD reconoce una palabra y queda en un estado de rechazo.

Errores Sintácticos:

- Asociados al análisis sintáctico.
- El compilador analiza la secuencia de tokens e intenta reconocer cada estructura dentro del archivo de entrada y ubicarlo en su categoría sintáctica. Asociado a la búsqueda de estructuras pertenecientes a los lenguajes independientes de contexto.
- El AFP nos indicará únicamente si una estructura es DERIVABLE de la BNF. Pero luego se deben cumplir las RESTRICCIONES SINTÁCTICAS para que dicha estructura que se derivó sea SINTÁCTICAMENTE CORRECTA.

- El error puede ser porque una secuencia de tokens no es derivable de la BNF o porque no cumple alguna restricción sintáctica.

Errores Semánticos:

- Asociados al análisis semántico, que trabaja de forma conjunta con el análisis semántico.
- Se encarga de validar que aquellas estructuras que son sintácticamente correctas tengan sentido en el contexto donde se encuentran dentro del programa. Para esto, se utilizan los lenguajes sensibles al contexto, que se analizan haciendo uso de las "Tablas de Símbolos".
- Se ocupa de:
    - Control de tipos de datos en las operaciones presentes en las expresiones.
    - Cantidad y tipo de dato de parámetros en la invocación de funciones.
    - Tipo de dato que retornado por las funciones.
    - Declaración de variables.
    - Declaración de funciones.

Sintácticamente vs. Semánticamente equivalentes:

- Dos acciones son semánticamente equivalentes si poseen el mismo significado. Ej. *P u P[0].
- Dos acciones son sintácticamente equivalentes si poseen la misma estructura.