Compiladores

Compilador: El código se traduce a código objeto y ese archivo es ejecutable.

Intérprete: Traduce y ejecuta.

Etapas de un compilador

Primero es la etapa de análisis y después la de síntesis. Son 6 fases agrupadas en 2 etapas.

Etapa de análisis

- Análisis léxico (que pertenezcan al lenguaje: palabras reservadas, operadores, etc.) (TOKENS)
- Análisis sintáctico (Estructura, orden) (Gramática libre de contexto)
- Análisis semántico (Sentido o significado) (Llamar a funcion si params, usar variable no declarada, cast o tipos equivocados)

Etapa de síntesis

- Generador de código intermedio
- Optimización de código
- Generador de código objeto

El manejo de errores está en las 6 fases de compilación.

Tabla de símbolos

Toda la info referentes a los identificadores del programa. La tabla de símbolos está involucrada en las 6 fases de la compilación.

Ejemplo:

int a = 5 + 3;

TOKEN: Todas las palabras válidas dentro de un lenguaje de programación

Requerimientos para hacer análisis léxico y sintáctico:

- Expresiones regulares
- Gramáticas libres de contexto

Análisis léxico

Detectar que no hayan caracteres no válidos. Se usan autómatas y regex.

Los compiladores no se paran cuando encuentran un error, lo reportan y sigue analizando en busca de más errores.

Hace un barrido secuencial del código fuente

Todos los tokens que sean de tipo identificador van a la tabla de símbolos.

Tipos de errores

- Símbolo no permitido
- Identificador mal construido o que excede cierta longitud permitida
- Constante numérica al construida o que excede cierto número de caracteres

Tarea

Archivo como entrada, identificar los números de control y dar como salida los correoss de la siguiente forma: "no. de control"@itmorelia.edu.mx

Tarea (Miércoles 28 de Agosto)

Clave de materia: A4O5 (grupo, semestre, carrera(O(info), L(sistemas), T(tics)), (0-9)) Carrera(No es token, se obtiene a partir de la clave de materia) Días: Horario: Salón: F o K

Empezamos a derivar con el axioma principal

Poner tokens en parte léxica.

matchedToken.beginLine: Sirve para saber el tamaño del lexema "|" (or) para dentro de los tokens

Si un lexema tiene la misma longitud, se toma el primero que aparece en los tokens definidos.

TOKEN: Crea un token ue coincida con la regex y lo envía al analizador sináctico

SKIP: No pasa la coincidencia a la fase sintáctica

MORE: Esta cadena será un prefijo para la nueva cadena coincidencia.

(MORE) Estado léxico: sirve para encontrar string.

DEFAULT: Primer token que nosotros definamos (en orden) en javacc.

“image” es un string buffer (se puede hacer cast a string).

TOKEN_MGR_DECLS:{

}

Todo dentro de de TOKEN_MGR_DECLS puede ser accesible desde cualquier acción léxica.

Acción léxica: Lo que se ejecutará cuando encuentre un lexema.

C- o C-Minus (lo que haremos con javacc)

- Subconjunto del lenguaje C
- Palabras reservadas: else, if, int, return, void, while
- Símbolos: los símbolos
- Símbolo no válido (Con número de línea)
- Id mal construido (Con número de línea)
- Decimales mal construidos (Con número de línea)

Equipos de 3: Ángel, Antonio y Julia

Análisis sintáctico

En esta fase se necesita las gramáticas libres de contexto.

Una gramática libre de contexto se componen de 4 cosas:

- Terminal
- No terminal
- Axioma
- Reglas