Compilador

Lenguaje de programación: python.

Paquete de python : ply

Módulo : lex.py

Módulo: yacc.py

Compilador:

* El lenguaje de programación basado en python.
* Funcionalidad permite sumar, restar, multiplicar, dividir, asignar y agrupar , imprimir e igualar.
* Estructura:

Lexer.py: Definición de los tokens.

Parser.py: Definición sixtasis y AST.

Apartir de este parser se origina la estructura de la pila que es el parser.out y

parsetab.py

CompiladorPython.py:

Generador.py: Programa1.hc , salida.ll

Lexer.py:

Módulo Lex.py :

* Apartir de la expresión como entrada genera cada toke individual .
* Devuelve tokens
* Continuidad de tokens.
* Escribe el archivo:

lextab.py: Reglas y tablas de expresión regular.

Parser.py:

Módulo yacc.py. : yacc.py se utiliza para analizar la sintaxis del lenguaje

* Análisis sintáctico por medio del algoritmo LALR1: Left: Permite construir todas las tablas de análisis de LR para la gramática.
* Sintaxis:

Salida árbol abstracto.

* First:

Expresiónes aritmética

* LR parsing (Análisis de reducción de desplazamiento. ) : El análisis LR es una técnica ascendente que reconoce el lado derecho de varias reglas gramaticales. Siempre que se encuentra un lado derecho válido en la entrada, se activa el código de acción apropiado y los símbolos gramaticales se reemplazan por el símbolo gramatical en el lado izquierdo. cambiando símbolos gramaticales a una pila y mirando la pila y el siguiente token de entrada en busca de patrones que coincidan con una de las reglas gramaticales
* Shift\_reduce parsing: uso de la stack.
* LR parsing: Botton
* Escribe el archivo:

parser.out (stack).

* Shift : LR parsing.
* Archivo parsetab.py : La tabla de análisis resultante
* yacc genera tablas en modo de depuración (lo que produce el archivo parser.out ).Al recibir tokens de entrada, el analizador crea una pila y busca reglas coincidentes.

Construcción AST:

Crear un conjunto de estructuras de datos para diferentes tipos de nodos de árbol de sintaxis abstracta.

Generacion de codigo

* la libreria LLVMLITE: uso del modulo IR (itermediate repredentation ), al cual se le pasa el AST , con las operaciones para transformar el arbol a codigo intermedio
* Pasaremos el archivo .hc a .ll
* Archivo .ll lo compilamos con el GCC y generamos el codigo compilado.