Compilador para C-

*Diseño de compiladores - Doc. Víctor Manuel de la Cueva*

Entrega Final

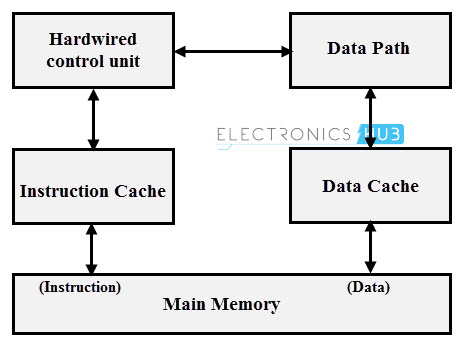
José Manuel Beauregard Méndez

A01021716

Introducción

Para la parte final de este proyecto se tuvo como reto traducir el código de C- a ensamblador con la finalidad de que la computadora pueda entender dicho código.

En clase se revisó la implementación con un procesador MIPS (Microprocessor without Interlocked Pipeline Stages) el cual es una versión reducida de un RISC.



Utilizando dicho procesador es posible nos permite completar el último paso de un compilador, generación de código. Se utilizó dicho procesador debido a que en una investigación que se realizó de otras opciones de procesadores como TM, y en dicha indagación encontré una fuente que comentaba lo común que es la incorporación de un MIPS en compiladores escolares gracias a su flexibilidad y facilidad de escribir código ensamblador.

Manual de usuarios

Utilizando dicho procesador es posible nos permite completar el último paso de un compilador, generación Un compilador está compuesto principalmente por tres partes, un analizador léxico, semántico, por último, un que recorra todo el programa identificando lo que esta pasando en cada línea. En este proyecto se abarco dichas partes, donde cada “módulo” se encarga de llevar a cabo su trabajo en el todo el proceso de compilación.

A continuación se mostrará el flujo por el cual los módulos interactúan entre sí:

**Input => LEXER -> PARSER (AST) -> SEMANTICA -> Code-Generation => Output**

Cabe destacar que este compilador fue producido para fines didácticos, si este compilador se quisiera llevar a un nivel profesional, antes se debería implementar mejores practicas para optimizar muchas de las tareas que se realizan. Para poder correr el programa es necesario tener la librería de ply en python3.x. Si no se tiene dicho módulo correr la siguiente línea en una terminal con python3.x:

**pip3 install ply**

Dicha librería es utilizada en las primeras dos partes del compilador, es decir, lexer y parser. Se escogió esta librería por su facilidad y flexibilidad al momento de escribir un compilador con todas sus partes. Este seria otro cambio importante para hacer en caso de que quisiera producir de manera profesional.

Considerando que el usuario ya tiene instalando python3.x y la librería de ply, entonces será posible correr el código de la siguiente forma:

from globalTypes import \*

from parser import \*

from semantica import \*

from cgen import \*

f = open('./sample.c-', 'r')

programa = f.read() *# lee todo el archivo a compilar*

progLong = len(programa) *# longitud original del programa*

programa = programa + '$' *# agregar un caracter $ que represente EOF*

posicion = 0 *# posición del caracter actual del string*

*# función para pasar los valores iniciales de las variables globales*

globales(programa, posicion, progLong)

AST = parser(False)

semantica(AST, True)

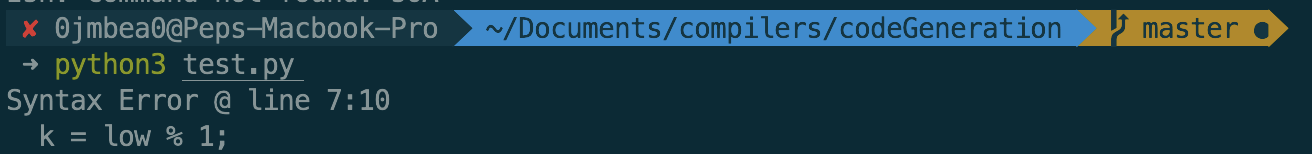
codeGen(AST, 'file.s')

Este es un ejemplo de un script que fusione las cuatro partes del código. En caso de querer imprimir los resultados de cada paso es importante pasar True en el caso de parser(...) y semántica(AST, …). Supongamos que le pongamos como nombre “test.py” al script anterior, entonces así se corre en la terminal dicho código:

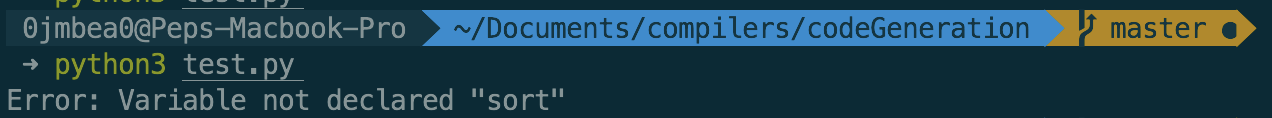
**python3 test.py**

Si en la terminal no hay ningún mensaje del compilador, entonces ya podremos encontrar un archivo llamado “file.s” donde podemos encontrar el código ensamblador generado. A continuación se mostrarán resultados posibles que el compilador pudiese arrojar:

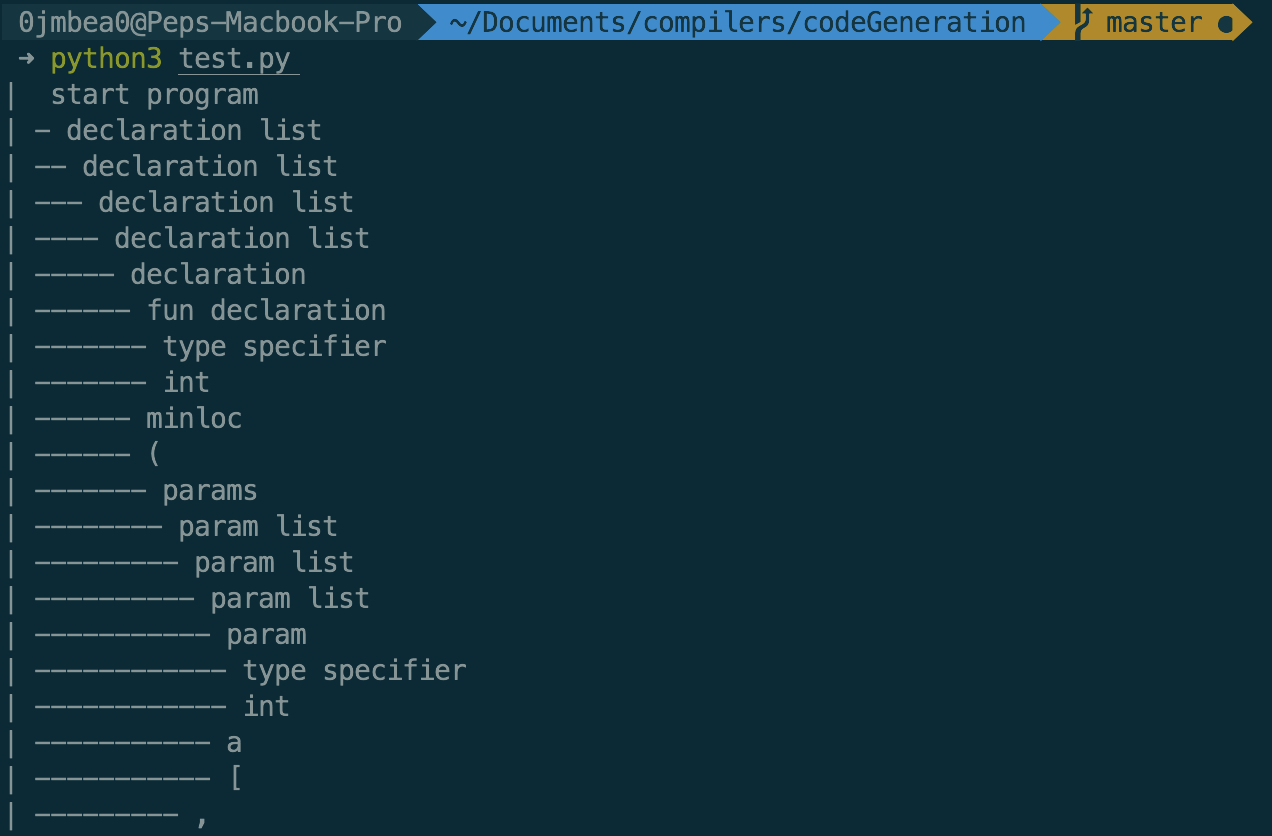
**Lexer error:**

****

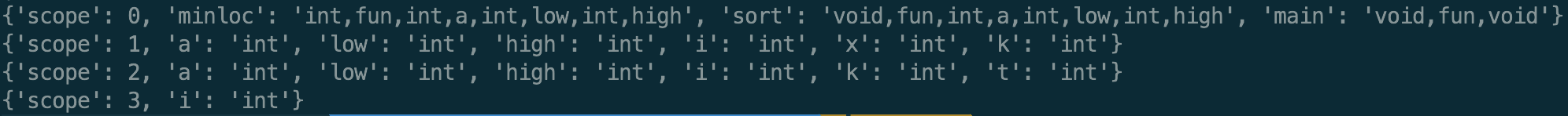
**Semantic error:**

****

**AST**

****

**Symbol tables**

****

En las siguientes secciones se mostrarán diferentes entregables necesarios para la elaboración de un compilador.

LEXER

*Diseño de compiladores - Doc. Víctor Manuel de la Cueva*

Primera Entrega

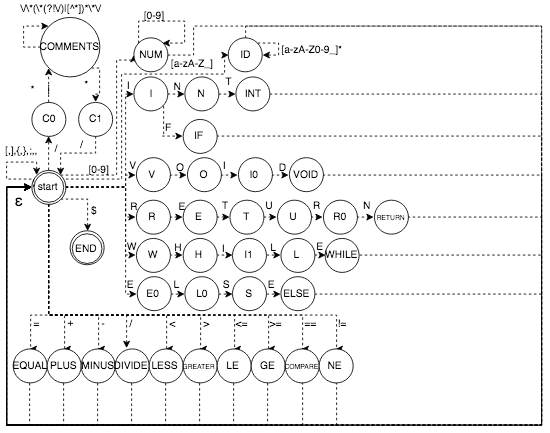
José Manuel Beauregard Méndez

A01021716

Expresiones regulares

|  |  |
| --- | --- |
| **Expresión Regular** | **Token** |
| \d+ | NUM |
| [a-zA-Z\_][a-zA-Z0-9\_]\* | ID |
| \/\\*(\\*(?!\/)|[^\*])\*\\*\/ | COMMENT |
| “=“ | EQUAL |
| \+ | PLUS |
| - | MINUS |
| \\* | TIMES |
| / | DIVIDE |
| < | LESS |
| > | GREATER |
| ; | SEMICOLOM |
| , | COMMA |
| \( | LPAREN |
| \) | RPAREN |
| \[ | LBRACKET |
| \] | RBRACKET |
| \{ | LBLOCK |
| } | RBLOCK |
| \$ | ENDFILE |

DFA



PARSER

*Diseño de compiladores - Doc. Víctor Manuel de la Cueva*

Segunda Entrega

José Manuel Beauregard Méndez

A01021716

Gramática

Rule 0 S' -> start

Rule 1 start -> declaration\_list

Rule 2 declaration\_list -> declaration\_list declaration

Rule 3 declaration\_list -> declaration

Rule 4 declaration -> var\_declaration

Rule 5 declaration -> fun\_declaration

Rule 6 declaration -> ENDFILE

Rule 7 var\_declaration -> type\_specifier ID SEMICOLON

Rule 8 var\_declaration -> type\_specifier ID LBRACKET NUMBER RBRACKET SEMICOLON

Rule 9 type\_specifier -> INT

Rule 10 type\_specifier -> VOID

Rule 11 fun\_declaration -> type\_specifier ID LPAREN params RPAREN compound\_stmt

Rule 12 params -> param\_list

Rule 13 params -> VOID

Rule 14 param\_list -> param\_list COMMA param

Rule 15 param\_list -> param

Rule 16 param -> type\_specifier ID

Rule 17 param -> type\_specifier LBRACKET RBRACKET

Rule 18 compound\_stmt -> LBLOCK local\_declarations statement\_list RBLOCK

Rule 19 local\_declarations -> local\_declarations var\_declaration

Rule 20 local\_declarations -> <empty>

Rule 21 statement\_list -> statement\_list statement

Rule 22 statement\_list -> <empty>

Rule 23 statement -> expression\_stmt

Rule 24 statement -> compound\_stmt

Rule 25 statement -> selection\_stmt

Rule 26 statement -> iteration\_stmt

Rule 27 statement -> return\_stmt

Rule 28 expression\_stmt -> expression SEMICOLON

Rule 29 expression\_stmt -> SEMICOLON

Rule 30 selection\_stmt -> IF LPAREN expression RPAREN statement

Rule 31 selection\_stmt -> IF LPAREN expression RPAREN statement ELSE statement

Rule 32 iteration\_stmt -> WHILE LPAREN expression RPAREN statement

Rule 33 return\_stmt -> RETURN SEMICOLON

Rule 34 return\_stmt -> RETURN expression SEMICOLON

Rule 35 expression -> var EQUAL expression

Rule 36 expression -> simple\_expression

Rule 37 var -> ID

Rule 38 var -> ID LBRACKET expression RBRACKET

Rule 39 simple\_expression -> additive\_expression relop additive\_expression

Rule 40 simple\_expression -> additive\_expression

Rule 41 relop -> LE

Rule 42 relop -> LT

Rule 43 relop -> GREATER

Rule 44 relop -> LESS

Rule 45 relop -> COMPARE

Rule 46 relop -> NE

Rule 47 additive\_expression -> additive\_expression addop term

Rule 48 additive\_expression -> term

Rule 49 addop -> PLUS

Rule 50 addop -> MINUS

Rule 51 term -> term mulop factor

Rule 52 term -> factor

Rule 53 mulop -> TIMES

Rule 54 mulop -> DIVIDE

Rule 55 factor -> LPAREN expression RPAREN

Rule 56 factor -> ID

Rule 57 factor -> call

Rule 58 factor -> NUMBER

Rule 59 call -> ID LPAREN args RPAREN

Rule 60 args -> arg\_list

Rule 61 args -> <empty>

Rule 62 arg\_list -> arg\_list COMMA expression

Rule 63 arg\_list -> expression

SEMANTICA

*Diseño de compiladores - Doc. Víctor Manuel de la Cueva*

Tercera Entrega

José Manuel Beauregard Méndez

A01021716

Symbol Tables

|  |  |
| --- | --- |
| **Symbols** | **Order** |
| Program | 0 |
| Declaration List | x |
| Declaration var | 2 |
| Declaration | 1 |
| Function Declaration | 2 |
| Void | 2 |
| Compound Statement | 0 |
| Local Declarations | 0 |
| Statement List | 0 |
| Expression Statement | 0 |
| Expression | 0 |
| Var | 0 |
| Selection Statement | 0 |
| Iteration Statement | 0 |
| Return Statement | 0 |
| Simple or Additive Statement | 0 |
| Simple/Additive Statement | 0 |
| Arg List | 0 |

Inference Rules

S' -> program  
program -> declaration\_list  
declaration\_list -> declaration\_list declaration  
declaration\_list -> declaration  
declaration -> var\_declaration  
declaration -> fun\_declaration  
var\_declaration -> type\_specifier ID SEMICOLON  
var\_declaration -> type\_specifier ID LBRACK NUM RBRACK SEMICOLON type\_specifier -> INT  
type\_specifier -> VOID  
fun\_declaration -> type\_specifier ID LPAREN params RPAREN compound\_stmt params -> param\_list  
params -> VOID  
param\_list -> param\_list COMA param  
param\_list -> param  
param -> type\_specifier ID  
param -> type\_specifier LBRACK RBRACK  
compound\_stmt -> LCURLY local\_declarations statement\_list RCURLY local\_declarations -> local\_declarations var\_declaration  
local\_declarations -> <empty>  
statement\_list -> statement\_list statement  
statement\_list -> <empty>  
statement -> expression\_stmt  
statement -> compound\_stmt  
statement -> selection\_stmt  
statement -> iteration\_stmt  
statement -> return\_stmt  
expression\_stmt -> expression SEMICOLON

expression\_stmt -> SEMICOLON  
selection\_stmt -> IF LPAREN expression RPAREN statement  
selection\_stmt -> IF LPAREN expression RPAREN statement ELSE statement iteration\_stmt -> WHILE LPAREN expression RPAREN statement return\_stmt -> RETURN SEMICOLON  
return\_stmt -> RETURN expression SEMICOLON  
expression -> var EQUALS expression  
expression -> simple\_expression  
var -> ID  
var -> ID LBRACK expression RBRACK  
simple\_expression -> additive\_expression relop additive\_expression simple\_expression -> additive\_expression  
relop -> LTHANEQ  
relop -> LTHAN  
relop -> GTHAN  
relop -> GTHANEQ  
relop -> EQUALTO  
relop -> NOTEQUALTO  
additive\_expression -> additive\_expression addop term  
additive\_expression -> term  
addop -> PLUS  
addop -> MINUS  
term -> term mulop factor  
term -> factor  
mulop -> TIMES  
mulop -> DIVIDE  
factor -> LPAREN expression RPAREN  
factor -> ID  
factor -> NUM