Iván Bruno Muñoz Yhmoff

A01366756

Compilador del lenguaje C-

Tabla de contenido

[Introducción 2](#_Toc42089865)

[Manual de Usuario 2](#_Toc42089866)

[Apéndice A: Expresiones Regulares 8](#_Toc42089867)

[Apéndice B: Forma EBNF de la GLC del lenguaje de programación C- 9](#_Toc42089868)

[Apéndice C: Analizador Semántico 10](#_Toc42089869)

[Apéndice D: Definición del lenguaje 12](#_Toc42089870)

# Introducción

El código generado a través del compilador para el lenguaje C- es del lenguaje ensamblador para el procesador MIPS, sin embargo, se utilizó el simulador SPIM en su versión más reciente QtSpim, para la implementación del compilador.

Fue seleccionada este tipo de procesador porque usa una arquitectura RISC (Reduced instruction set computing) y este tipo de arquitectura es común en los procesadores actuales; Por otro lado, se escogió este tipo de procesador por la disponibilidad de éste debido al simulador antes mencionado.

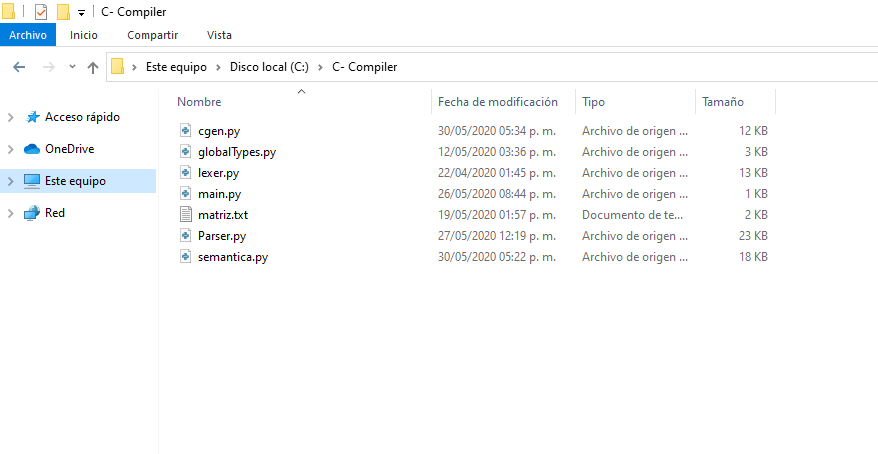
# Manual de Usuario

Requisitos:

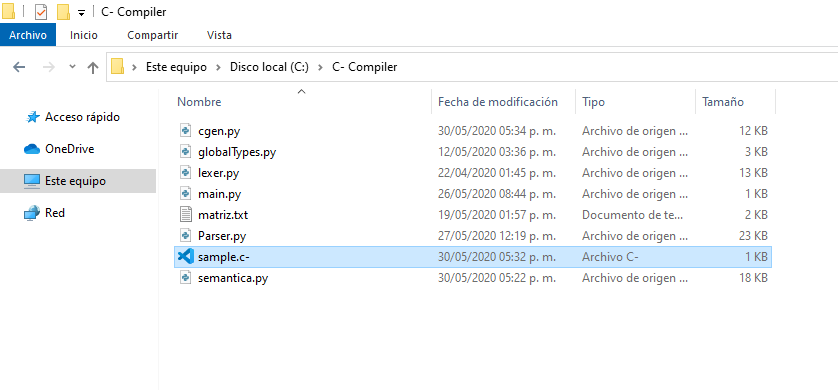
* Tener Python instalado en su computadora y listo para funcionar.
* Tener descargado y funcionando el simulador SPIM , de preferencia su última versión: QtSpim.

Este es el procedimiento para compilar un programa escrito en el lenguaje C-:

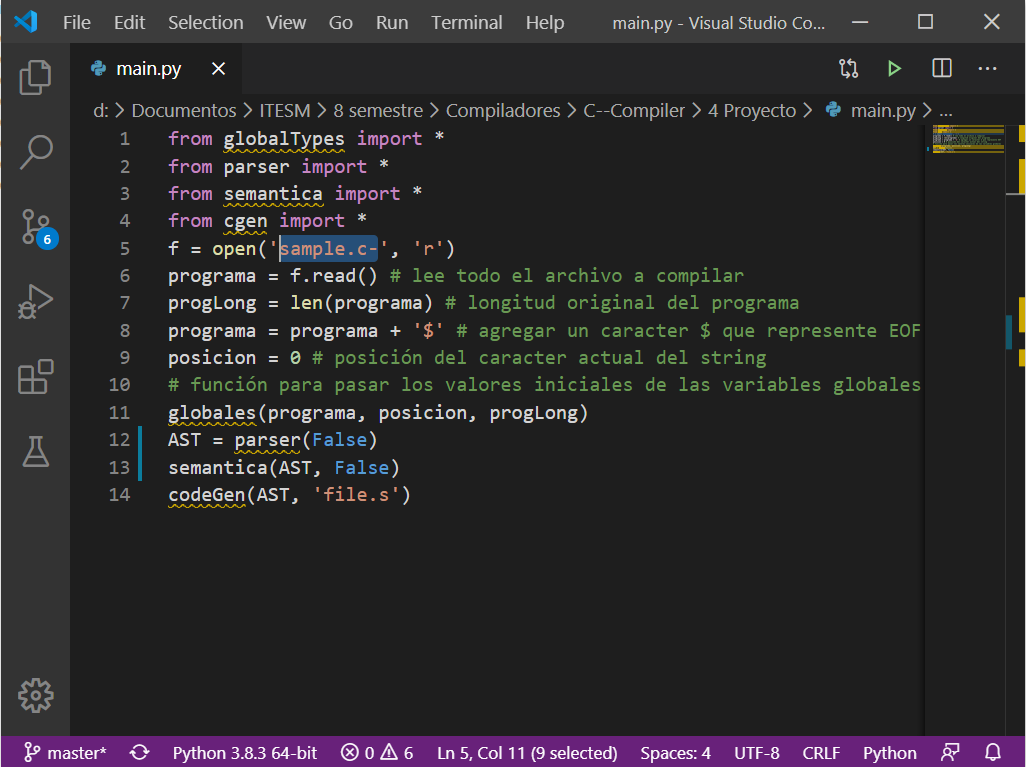
1. Obtener el paquete de archivos que componen el compilador:
   1. matriz.txt
   2. globalTypes.py
   3. lexer.py
   4. Parser.py
   5. semántica.py
   6. cgen.py
   7. main.py



1. Debe tener en la misma carpeta el programa que quiere compilar (en este caso se llama “sample.c-“):



1. En caso de que su archivo tenga otro nombre debe cambiarlo por “sample.c-“ o en su defecto entrar al archivo llamado “main.py” y editar la línea 5, poniendo el nombre de su archivo donde dice “sample.c-“:

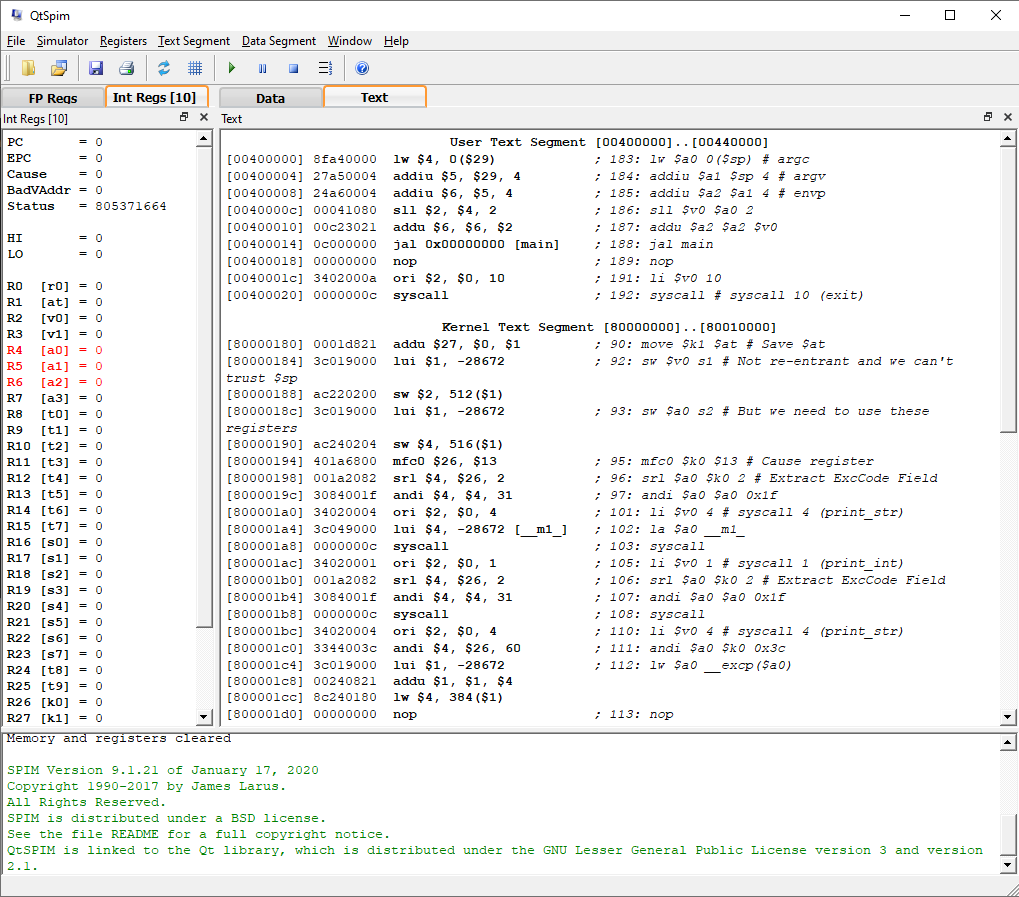


1. Ahora debe ejecutar el programa “main.py” entrando desde una terminal de comandos, abierta en la carpeta donde se ubican todos los archivos del compilador y escribir el siguiente comando:
   1. python main.py

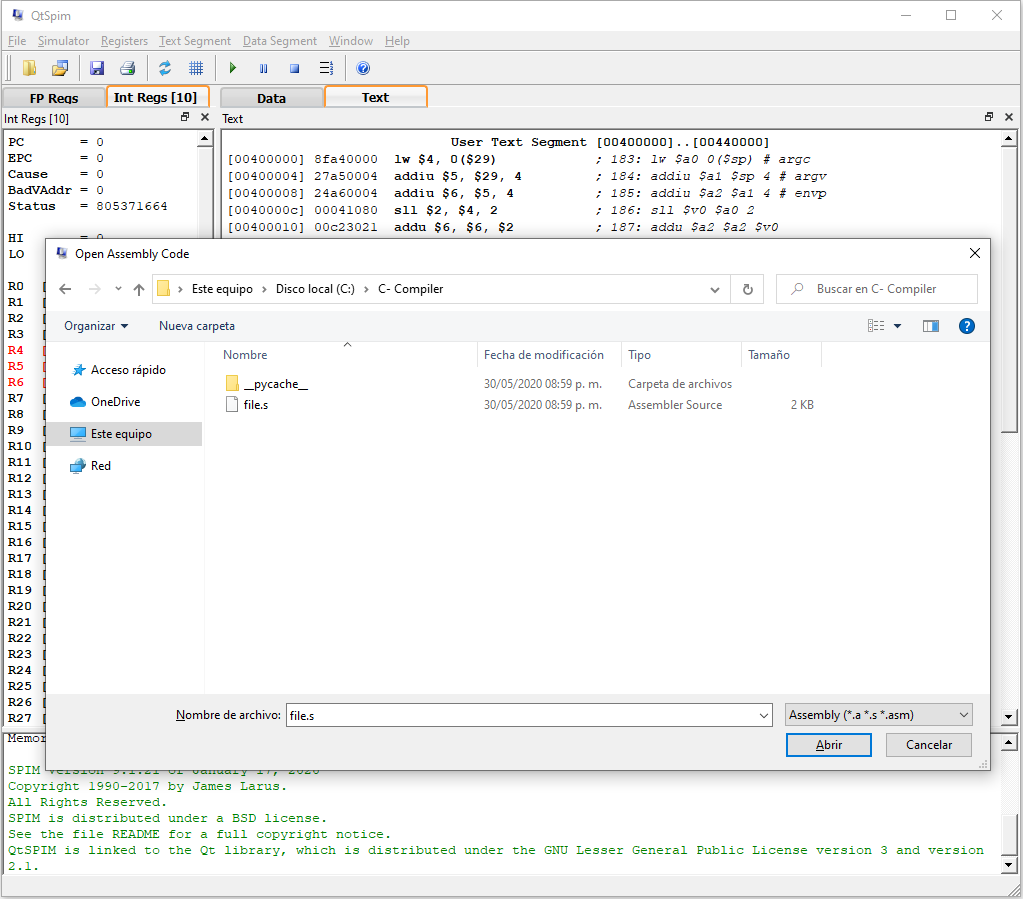


* 1. En caso de que su programa tenga algún error se le notificará a través de la misma terminal.

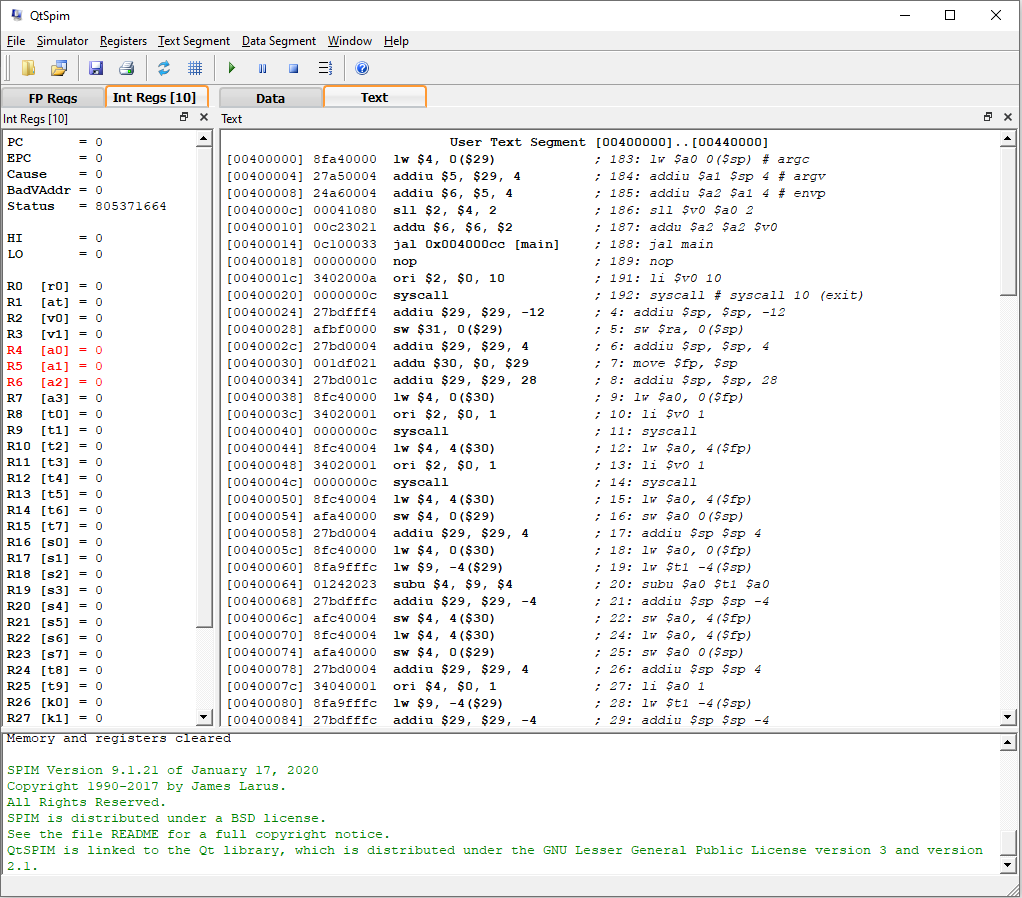
1. En la misma carpeta se habrá generado un archivo nuevo con el nombre de “file.s” y lo deberá abrir desde su simulador QtSpim:
   1. Dando click en el botón “Load File”:



* 1. Seleccionar el archivo “file.s” que se generó dentro de su carpeta:



* 1. Dar click en el botón de “play”:



Al completar estos pasos habrá compilado su programa escrito en el lenguaje C- y podrá ver sus resultados en la consola del simulador.

# Apéndice A: Expresiones Regulares

LETRA = A|…|Z|a|…|z

DIGITO = 0|…|9

ID = LETRA LETRA\*

NUM = DIGITO DIGITO\*

COMENTARIO = /\* (LETRA|DIGITO|^(\*/))\* \*/

IF = if

ELSE = else

INT = int

RETURN = return

VOID = void

WHILE = while

PLUS = +

LESS = -

MULT = \*

DIV = /

LT = <

MT = >

LTEQUAL = <=

MTEQUAL = >=

EQUAL = ==

DIFF = !=

ASSIGN = =

SEMICOL = ;

COMMA = ,

OPENB = (

CLOSEB = )

OPENSB = [

CLOSESB = ]

OPENCB = {

CLOSECB = }

# Apéndice B: Forma EBNF de la GLC del lenguaje de programación C-

1. program -> declaration {declaration}
2. declaration -> var-decalaration | fun-decalaration
3. var-declaration -> type-specifier ID [“[“NUM “]”]";"
4. type-specifier -> int | void
5. fun-decalaration -> type-specifier ID "("params")" compound-stmt
6. params -> param {"," param}
7. param -> type-specifier ID [“[“NUM “]”]
8. compound-stmt -> "{" local-declarations {statement}"}"
9. local-declarations -> {var-decalaration}
10. statement -> expression-stmt | compound-stmt | selection-stmt | iteration-stmt | return-stmt
11. expression-stmt -> [expression] ";"
12. selection-stmt -> if "("expression")"statement [else statement]
13. iteration-stmt -> while "("expression")" statement
14. return-stmt -> return [expression]";"
15. expression -> var "=" expression | simple-expression
16. var -> ID ["["expression"]"]
17. simple-expression -> additive-expression [relop additive-expression]
18. relop -> <= | < | > | >= | == |!=
19. additive-expression -> term {addop term}
20. addop -> +|-
21. term -> factor {mulop factor}
22. mulop -> \*|/
23. factor -> "("expression")" | var | call | NUM
24. call -> ID "("args")"
25. args - [expression {"," expression}]

# Apéndice C: Analizador Semántico

**Estructura del stack:**

El stack tendrá la estructura de una lista, y sólo contendrá los números que representen cada scope de la tabla de símbolos, además en la primera posición siempre contendrá el número 0 que representará la tabla de símbolos global y éste no podrá ser sacado del stack.

Un ejemplo del stack sería este:

* Stack = [0,2,3]

En donde significa que el scope 3 está dentro del scope 2, y a su vez éste está dentro del scope 0.

**Estructura de la tabla de símbolos:**

La tabla de símbolos correspondiente a cada scope, se encontrará dentro de un diccionario de diccionarios, siendo identificada cada una mediante un valor entero, que además representa el orden en el que fueron creadas.

Cada una de estas tablas tendrá la siguiente estructura:

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre del atributo | Tipo de variable | Notas | Ejemplo |
| Identificador del atributo | Cadena de caracteres | Con este se va a encontrar cada variable o función dentro de la tabla. | “main” |
| Tipo de variable o función | Cadena de caracteres | Este va a representar el tipo de la variable o función, pudiendo ser “void” o “int”. | “int” |
| Si es arreglo | Booleano | Este atributo será “True” en caso de que la variable que representa sea un arreglo y “False” en cualquier otro caso. | False |
| Tamaño del arreglo o número de parámetros de la función | Entero | Este atributo tendrá un valor de -1 cuando el identificador en cuestión no sea arreglo ni función.  En caso de ser función tendrá el número de parámetros correspondientes y en caso de ser arreglo tendrá el tamaño del arreglo. | 0 |
| Scope de la función | Entero | En caso de que el atributo en cuestión sea una función, aquí estará el scope correspondiente a esta.  En caso de que sea cualquier otro tipo de atributo, tendrá el valor de -1. | 5 |

**Reglas lógicas de inferencia de tipos:**

1. Identificadores:
   1. TablaDeSimbolos(Id) = INTEGER → ⊢ Id: INTEGER
2. Arreglos:
   1. TablaDeSimbolos(Id) = ARRAY → ⊢ Id: ARRAY
3. Identificadores:
   1. Identificador es una literal entera → ⊢ Identificador: INTEGER
4. Constantes:
   1. Constante → ⊢ Constante: INTEGER
5. Arreglos:
   1. Identificador es un arreglo → ⊢ Identificador: ARRAY
6. Operadores lógicos:
   1. String está dentro de los operadores lógicos → ⊢ String: BOOLEAN
7. Operadores enteros:
   1. String está dentro de los operadores → ⊢ String: INTEGER
8. Asignación:
   1. String es de tipo assign → ⊢ String: INTEGER
9. Constante dentro de los corchetes de un arreglo:
   1. ⊢ Identificador: ARRAY ^ ⊢ Constante: INTEGER → ⊢ Identificador [Constante]: INTEGER
10. Variable dentro de los corchetes de un arreglo:
    1. ⊢ Identificador1: ARRAY ^ ⊢ Identificador2: INTEGER → ⊢Identificador1[Identificador2]: INTEGER
11. Suma:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1+Identificador2: INTEGER
12. Resta:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1-Identificador2: INTEGER
13. División:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1/Identificador2: INTEGER
14. Multiplicación:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1\*Identificador2: INTEGER
15. Mayor que:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1>Identificador2: BOOLEAN
16. Menor que:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1<Identificador2: BOOLEAN
17. Menor o igual que:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1<=Identificador2: BOOLEAN
18. Mayor o igual que:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1>=Identificador2: BOOLEAN
19. Igual que:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1==Identificador2: BOOLEAN
20. Diferente que:
    1. ⊢ Id1: INTEGER ^ ⊢ Id2: INTEGER → ⊢Identificador1 !=Identificador2: BOOLEAN
21. Estatuto If:
    1. ⊢ OP: BOOLEAN → ⊢ if(OP):BOOLEAN
22. Estatuto while:
    1. ⊢ OP: BOOLEAN → ⊢ while(OP):BOOLEAN
23. Estatuto return:
    1. ⊢ EXP: INTEGER→ ⊢ return EXP:INTEGER
24. Estatuto call:
    1. ⊢TablaDeSimbolos(parametro) = INTEGER ^ ⊢TablaDeSimbolos(id) = INTEGER → ⊢
25. Estatuto call:
    1. ⊢TablaDeSimbolos(parametro) = INTEGER ^ ⊢TablaDeSimbolos(id) = INTEGER → ⊢Es correcto
26. Estatuto call:
    1. ⊢TablaDeSimbolos(parametro) = ARRAY ^ ⊢TablaDeSimbolos(id) = ARRAY → ⊢Es correcto

# Apéndice D: Definición del lenguaje

[Lenguaje C-.pdf](Lenguaje%20C-.pdf)