Instituto Tecnológico y de Estudios Superiores de Monterrey

Proyecto: Generador de código

Víctor Ignacio Cano Salazar

A01366074

Índice

Introducción

En estos días, y gracias a los avances tecnológicos que hemos presenciado, podemos decir que cada vez se vuelve más común hablar de lenguajes de programación, ya que la humanidad en sí, se ha dado cuenta de que está siendo la base de nuestro futuro, y que, aunque aún no sea necesario que todas las personas aprendan algún lenguaje de programación, estoy seguro que en unos cuantos años será una herramienta indispensable para las personas.
Dicho lo anterior, al tener la oportunidad de escribir un programa en algún lenguaje, nos podemos dar cuenta de que es algo que podemos entender; a la mejor no a la primera, pero con práctica se vuelve familiar el código que escribimos o que vemos de alguna otra persona.
Por otro lado, para la computadora es más fácil si, en lugar de símbolos, transforma todo a números. Es por eso por lo que se escogió que dicha transformación la hiciera MIPS, ya que el ensamblador va a leer un archivo fuente (el que se va a generar) en lenguaje ensamblador, y nos va a producir un 'archivo objeto' que va a contener instrucciones máquina donde se almacenará el resultado final.

Manual de usuario

Software necesario

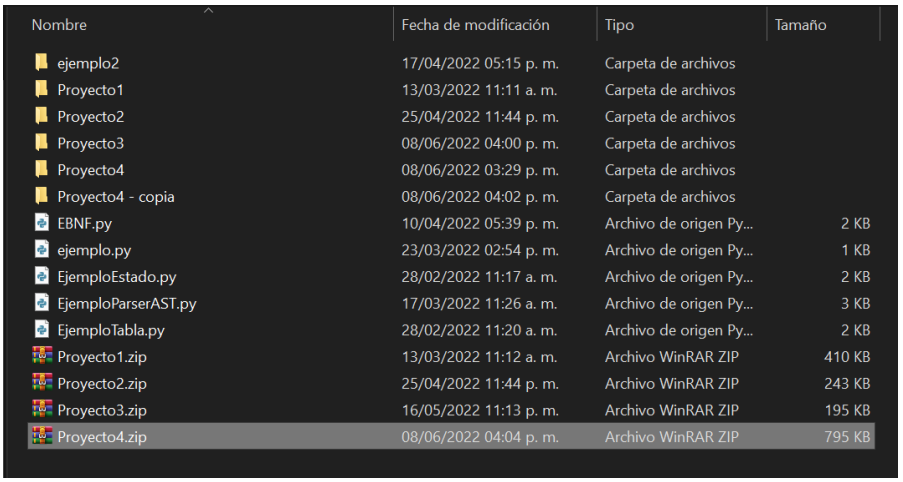
Para poder ejecutar los programas que contiene el compilador es necesario que tenga instalado Python: https://www.python.org/downloads/release/python-392/
La versión que se utilizó para el desarrollo del proyecto fue:

```
C:\Users\Víctor Cano>py --version
Python 3.9.2
```

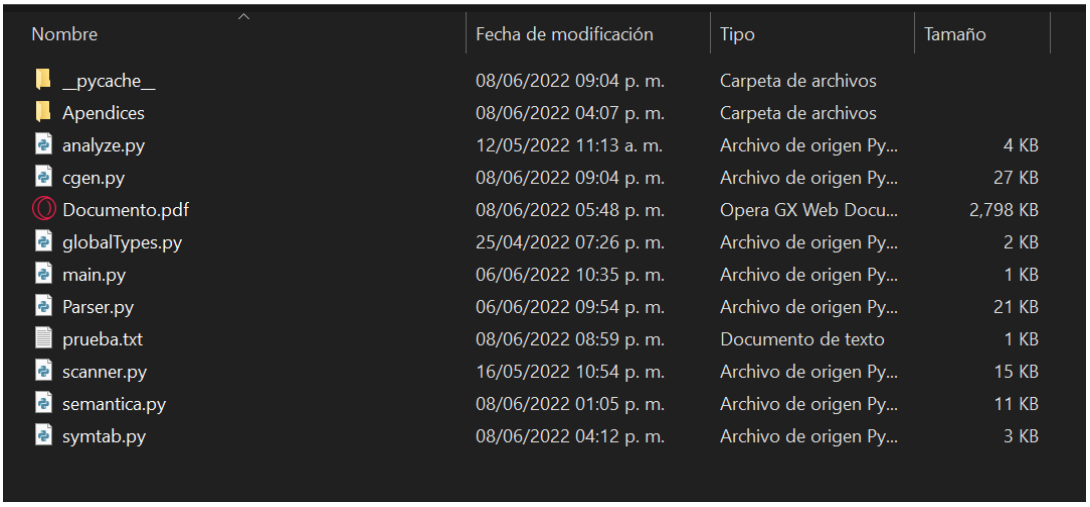
Del mismo modo, su equipo debe de contar con QtSpim que es un simulador que corre programas en MIPS (nuestro lenguaje ensamblador): http://spimsimulator.sourceforge.net

Descarga del compilador

Al descargar el archivo 'Proyecto4.zip' deberemos descomprimir el archivo en una carpeta que ubiquemos dentro de nuestra computadora:



Podremos observar que tenemos estos archivos:



Nombre	Fecha de modificación	Tipo	Tamaño
__pycache__	08/06/2022 09:04 p. m.	Carpeta de archivos	
Apendices	08/06/2022 04:07 p. m.	Carpeta de archivos	
analyze.py	12/05/2022 11:13 a. m.	Archivo de origen Py...	4 KB
cgen.py	08/06/2022 09:04 p. m.	Archivo de origen Py...	27 KB
Documento.pdf	08/06/2022 05:48 p. m.	Opera GX Web Docu...	2,798 KB
globalTypes.py	25/04/2022 07:26 p. m.	Archivo de origen Py...	2 KB
main.py	06/06/2022 10:35 p. m.	Archivo de origen Py...	1 KB
Parser.py	06/06/2022 09:54 p. m.	Archivo de origen Py...	21 KB
prueba.txt	08/06/2022 08:59 p. m.	Documento de texto	1 KB
scanner.py	16/05/2022 10:54 p. m.	Archivo de origen Py...	15 KB
semantica.py	08/06/2022 01:05 p. m.	Archivo de origen Py...	11 KB
symtab.py	08/06/2022 04:12 p. m.	Archivo de origen Py...	3 KB

Escribir un ejemplo de código

Para poder compilar de manera exitosa, el primer paso será escribir un código a compilar. Debemos de tener en cuenta que se estará utilizando el lenguaje C-. En caso de no conocer dicho lenguaje o, no recordar su sintaxis, se puede consultar la siguiente liga, con el fin de que no se genere ningún error al momento de compilar: http://www.ii.uib.no/~wolter/teaching/h09-inf225/project/Syntax-C-Minus.pdf

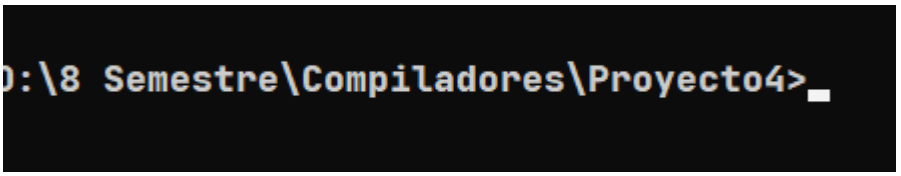
Una vez familiarizados, podemos escribir nuestro código de ejemplo. Para este paso, nos debemos de dirigir al archivo llamado 'prueba.txt' que va a fungir como nuestro editor de texto. En él podemos escribir alguna operación en nuestra función principal main, definir algunas variables, entre otros ejemplos. Para este caso, y, guiándonos de los ejemplos de C- de la liga anterior utilizaremos:

```
int a[5];

int compare (int x, int y, int z){
    a[2] = x;
    if(a[2] > z){
        a[1] = x + y;
    }else{
        a[1] = x - y;
    }
    x = a[1];
    return x;
}

void main(void){
    compare(5,4,2);
}
```

Posterior a escribir el código debemos de guardarlo.

Correr el ejemplo de código

Para poder compilar nuestro código, debemos de abrir una terminal en nuestra computadora (en caso de no saber como hacer dicho paso puede guiarse de la siguiente liga: https://es.wikihow.com/abrir-la-terminal-en-Windows), y dirigirnos a la ubicación de nuestro proyecto:



Una vez estando en el proyecto, debemos de ejecutar cualquiera de los comandos:

py main.py

python main.py

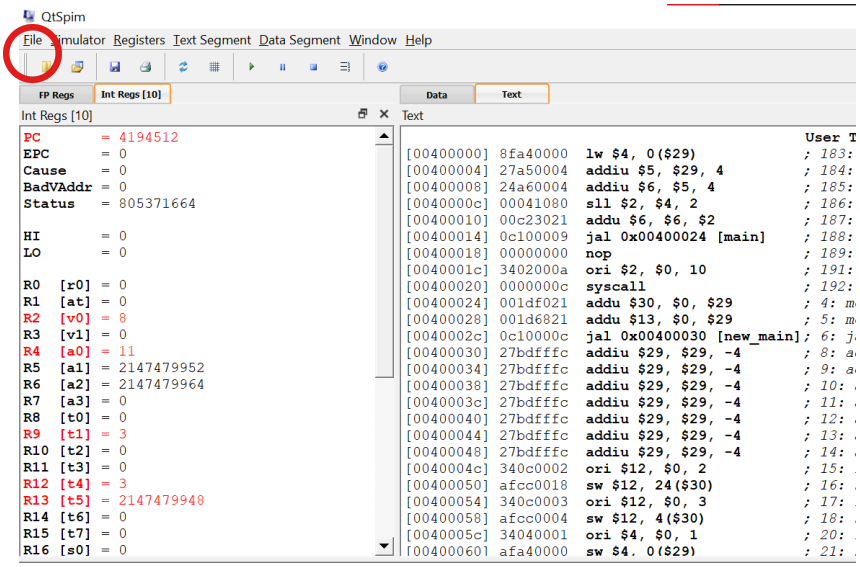
Nuestra consola se verá de la siguiente manera (siempre y cuando hayamos hecho el código de ejemplo descrito anteriormente):

```
D:\8 Semestre\Compiladores\Proyecto4>py main.py
  Program:
    List Declaration:
      Declaration:  int
        ID:  a
        Const:  5
      Declaration:  int
        ID:  compare
        Function:
          Params:
            Param List:
              Declaration:  int
                ID:  x
              Declaration:  int
                ID:  y
              Declaration:  int
                ID:  z
          Group:
            List Declaration:
            ListStatement:
              Assign:  =
                ID:  a
                  Const:  2
                ID:  x
              If:
                Op:  >
                  ID:  a
                    Const:  2
                  ID:  z
                Group:
                  List Declaration:
                  ListStatement:
                    Assign:  =
                      ID:  a
                        Const:  1
                      Op:  TokenType.PLUS
                        ID:  x
                        ID:  y
                Group:
                  List Declaration:
                  ListStatement:
                    Assign:  =
                      ID:  a
                        Const:  1
                      Op:  TokenType.MINUS
                        ID:  x
                        ID:  y
              Assign:  =
                ID:  x
                ID:  a
                  Const:  1
              Return:
                ID:  x
      Declaration:  void
        ID:  main
                    Const:  2


SCOPE 2

Key             Type            Structure       Value
---             ----            ---------       ------


SCOPE 1

Key             Type            Structure       Value
---             ----            ---------       ------
z               int             Const           None
y               int             Const           None
x               int             Const           None


GLOBAL TABLE

Key             Type            Structure       Value
---             ----            ---------       ------
main            void            Function        void
compare         int             Function        ['Const', 'Const', 'Const']
a               int             Array           5

D:\8 Semestre\Compiladores\Proyecto4>
```

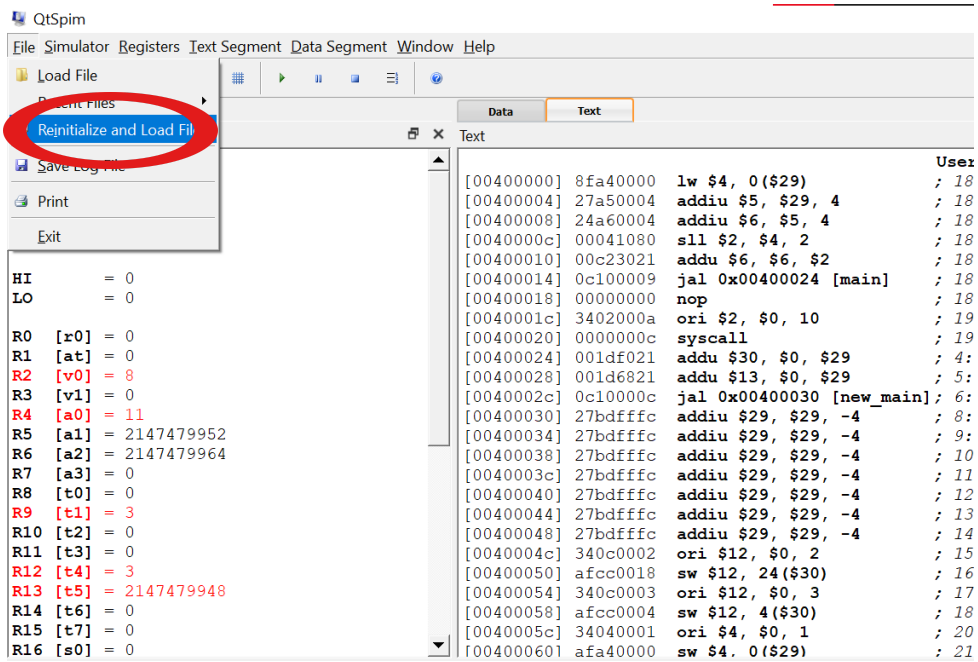
Al mismo tiempo podremos notar que se nos creó un archivo llamado ‘file.asm’, que es el que ocuparemos para mostrar el resultado de nuestro programa.

MIPS

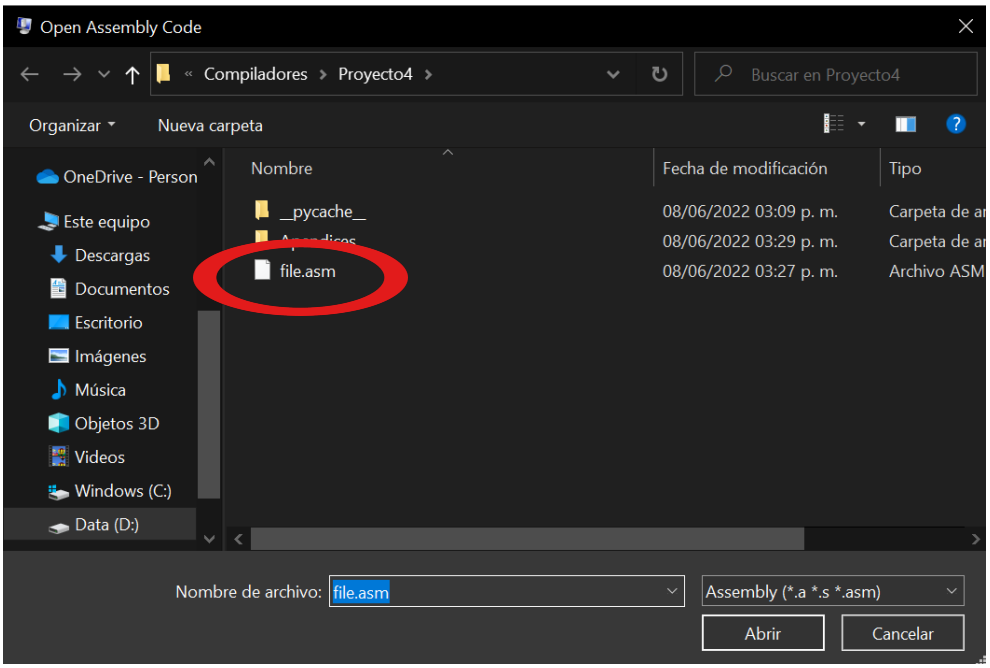
Para poder ver el resultado de nuestra operación descrita en ‘prueba.txt’ el resultado es 5, por lo que en QtSpim, del lado izquierdo y en a0, nos debe de almacenar el resultado.
Para esto, lo que debemos de hacer es, abrir nuestro QtSpim, y debemos de ubicarnos en la esquina superior derecha en la parte de ‘File’:



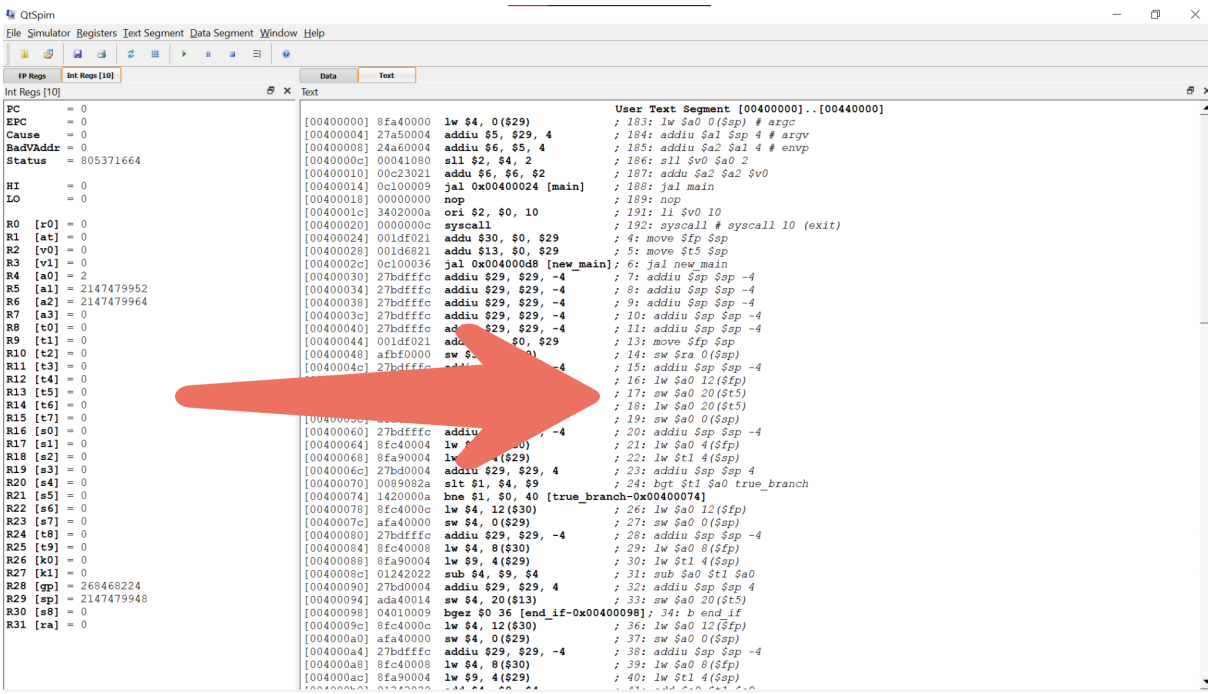
Daremos clic, y en el recuadro que nos aparece daremos clic en donde dice ‘Reinitialized and Load File’:



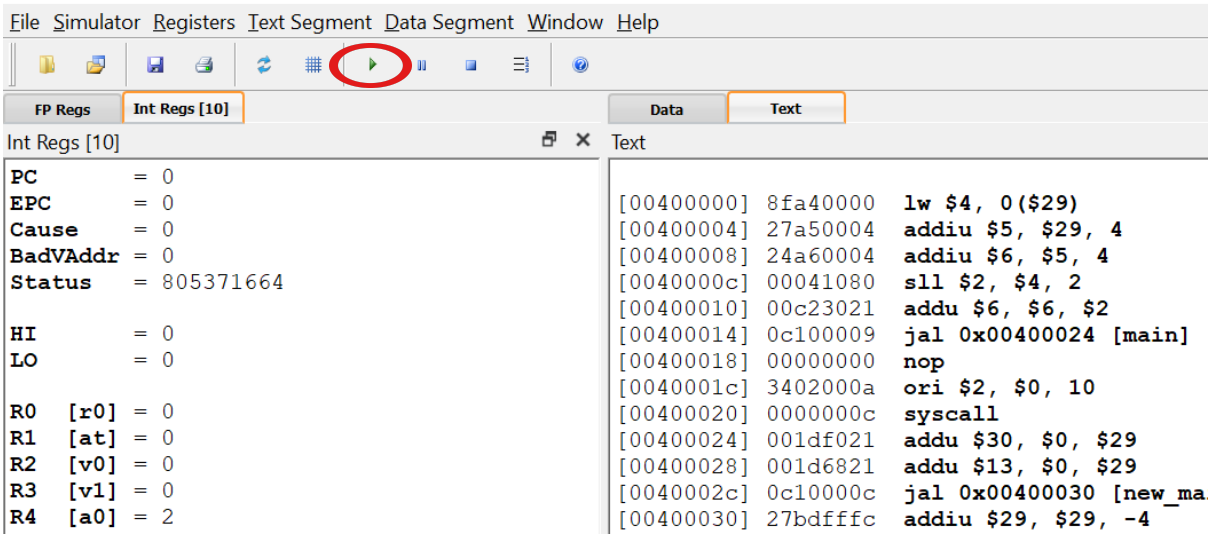
Esto abrirá nuestro buscador de archivos, y nos debemos de ubicar en la carpeta donde guardamos el .zip, y encontraremos el archivo que previamente se nos creó llamado ‘file.asm’:



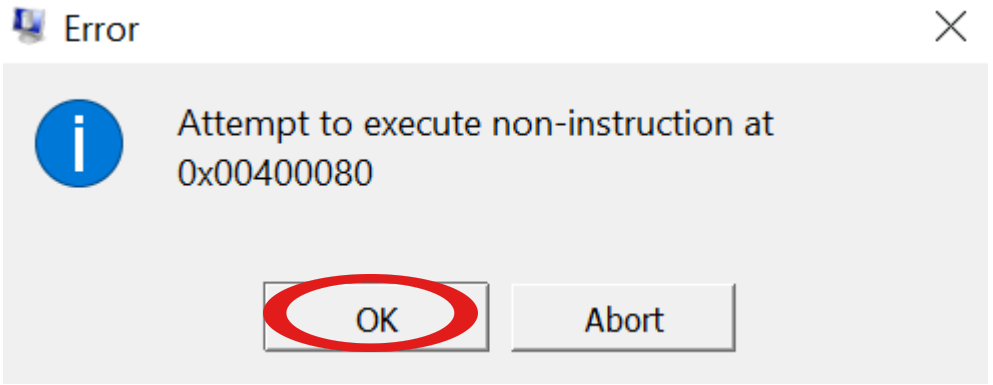
Una vez seleccionado el archivo, nos mandara de nuevo al inicio de QtSpim, solo que esta vez ya vamos a tener el archivo cargado, y nos podemos dar cuenta si seleccionamos ‘Text’ y observamos en la columna de la derecha que está cargado nuestro código generado a ensamblador:



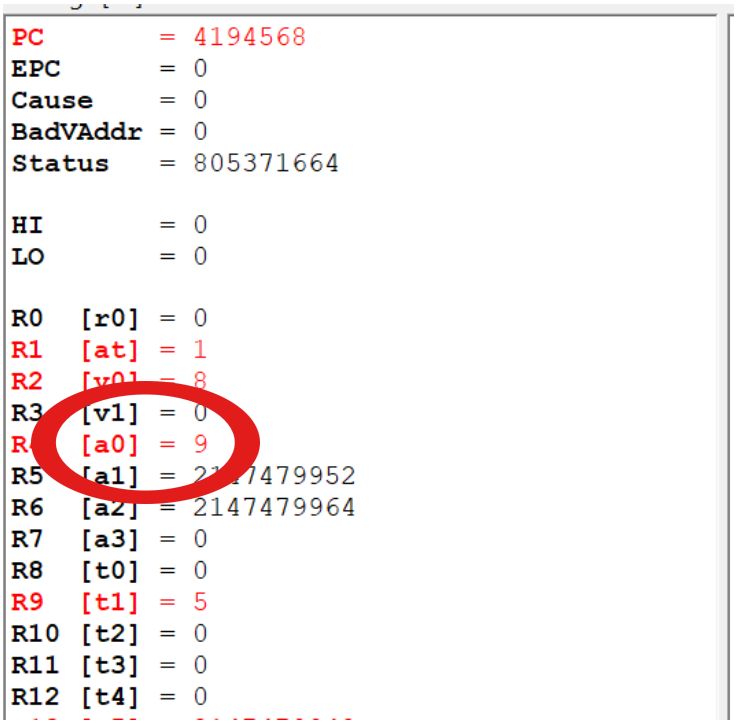
Para correrlo solo basta con dar clic en el ícono de play:



Luego dar ‘Ok’ al recuadro que aparece:

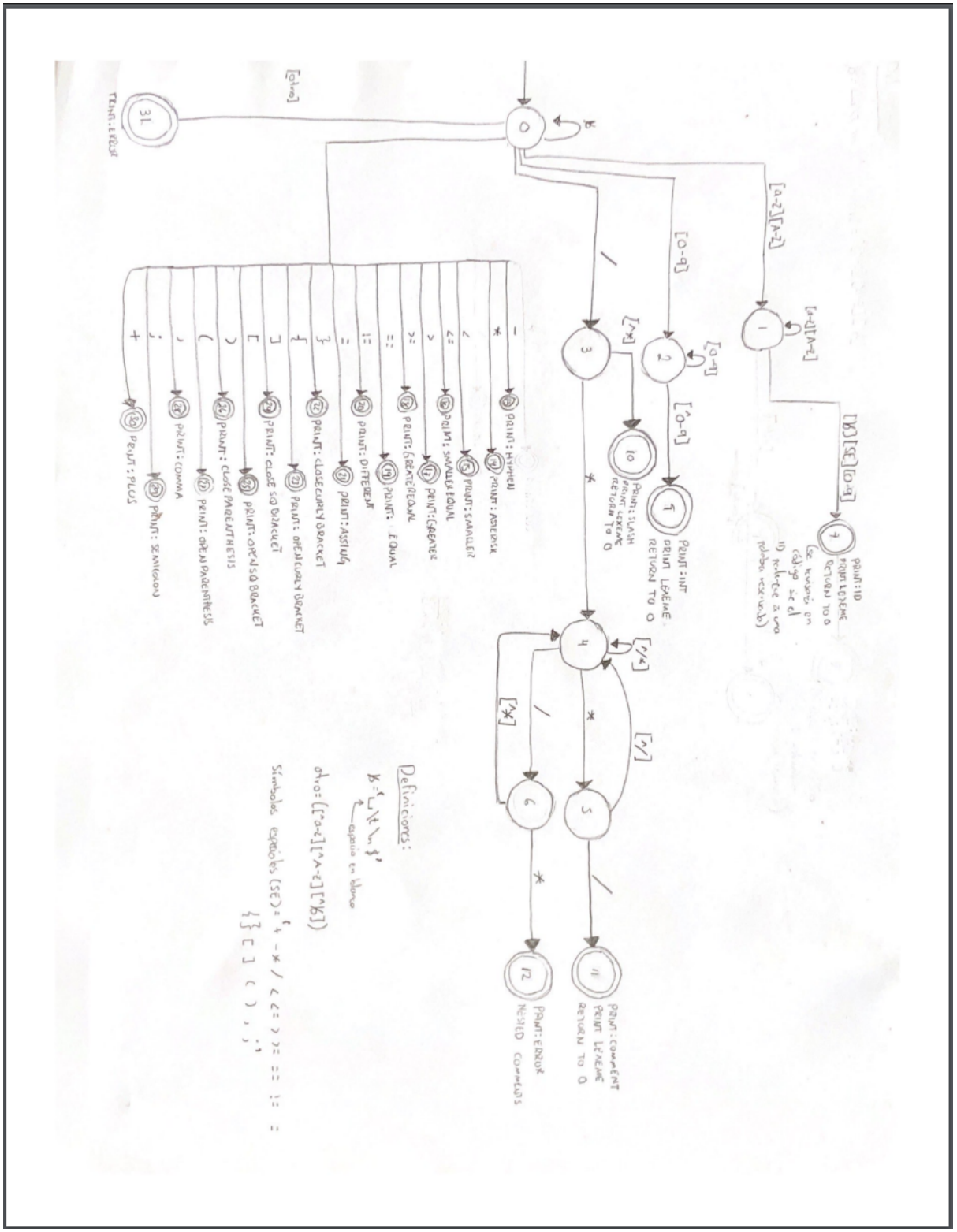


Y, nuestro resultado final se encontrará del lado izquierdo, en a0:



Apéndices

Lexer



ID:

([a-z|A-Z][a-z]*[A-Z]*)

INT:

([0-9][0-9]*)

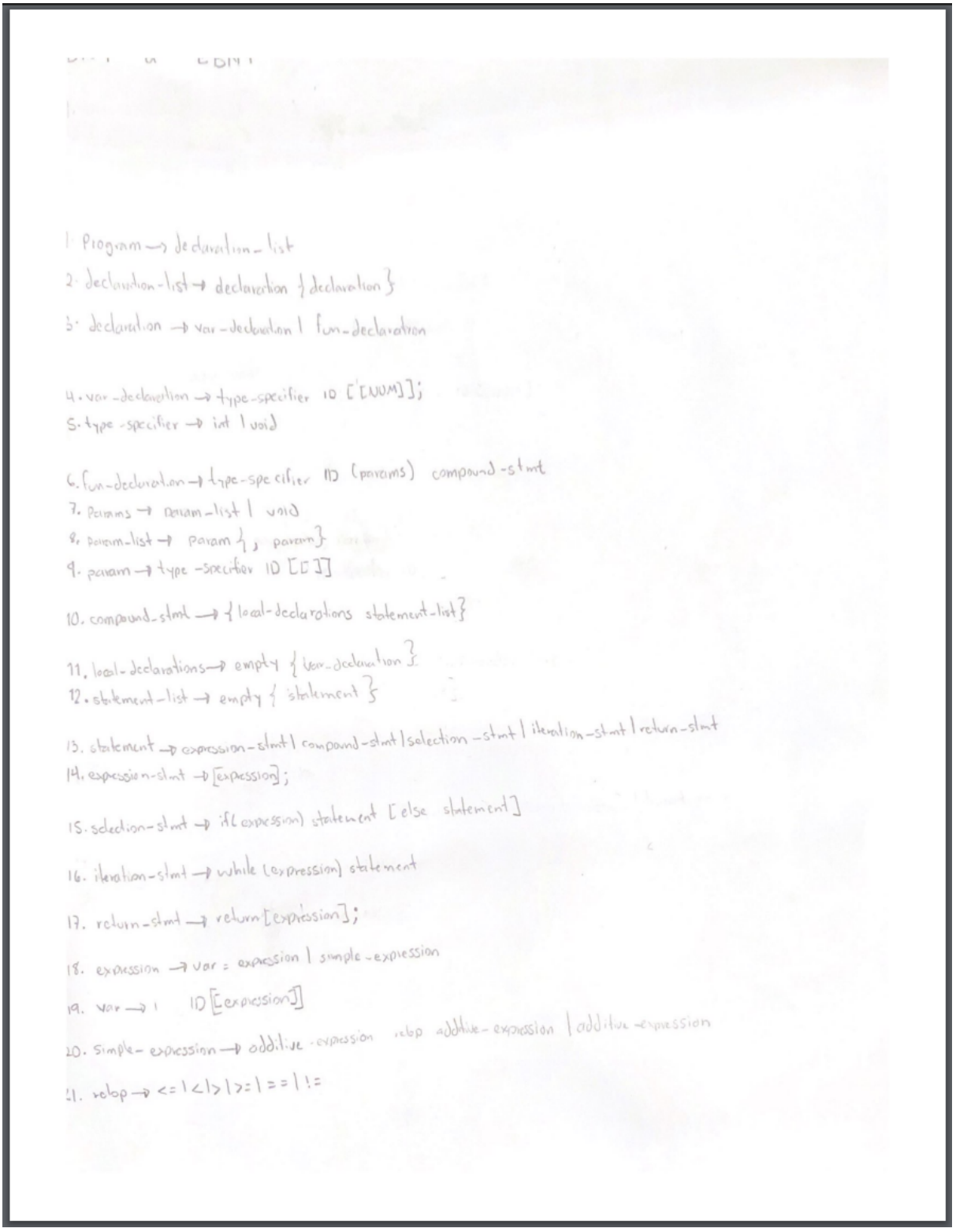
COMMENT:

(/*[^/*]* */)

SPECIAL SYMBOLS:

([+|-|*|/|>|>=|<|<=|==|!=|=|;|,|(|)|[|]|{|}]*)

Parser



1. Program → declaration_list
2. declaration-list → declaration { declaration }
3. declaration → var-declaration | fun-declaration

4. var-declaration → type-specifier ID ['['NUM']'];
5. type-specifier → int | void

6. fun-declaration → type-specifier ID (params) compound-stmt
7. Params → param-list | void
8. param-list → param { , param }
9. param → type-specifier ID [[]]

10. compound-stmt → { local-declarations statement-list }

11. local-declarations → empty { var-declaration }
12. statement-list → empty { statement }

13. statement → expression-stmt | compound-stmt | selection-stmt | iteration-stmt | return-stmt
14. expression-stmt → [expression];

15. selection-stmt → if(expression) statement [else statement]

16. iteration-stmt → while (expression) statement

17. return-stmt → return [expression];

18. expression → var = expression | simple-expression

19. var → | ID [[expression]]

20. simple-expression → additive-expression relop additive-expression | additive-expression

21. relop → <= | < | > | >= | == | !=

22. additive_expression → term {addop term}

23. addop → + | -

24. term → factor {mulop factor}

25. mulop → * | /

26. factor → (expression) | var | call | NUM

27. call → ID (args)

28. args → arg-list | empty

29. arg-list → expression {, expression}

REGLAS LÓGICAS DE INFERENCIA

Regla	Expresión
Declaración:	
Constante entera [INT]	i es una literal entera / ⊢i: int
Asignación de valores a una variable	O(x) = T / O⊢x: T
Arreglo	O[v]⊢v[i]: int / ⊢i: int
Argumento	O[param]⊢param: int / ⊢args: int
Símbolos especiales:	
Suma	⊢e1: int, ⊢e2: int / ⊢e1+e2: int
Resta	⊢e1: int, ⊢e2: int / ⊢e1-e2: int
División	⊢e1: int, ⊢e2: int / ⊢e1/e2: int
Multiplicación	⊢e1: int, ⊢e2: int / ⊢e1*e2: int
Mayor que	⊢e1: int, ⊢e2: int / ⊢e1<e2: int
Menor que	⊢e1: int, ⊢e2: int / ⊢e1>e2: int
Mayor igual que	⊢e1: int, ⊢e2: int / ⊢e1<=e2: int
Menor igual que	⊢e1: int, ⊢e2: int / ⊢e1>=e2: int
Diferente	⊢e1: int, ⊢e2: int / ⊢e1!=e2: int
Igual	⊢e1: int, ⊢e2: int / ⊢e1==e2: int
Función:	
Return [INT]	⊢función: int / ⊢return: int
Return [VOID]	⊢función: void / ⊢return: void

Explicación de la estructura de la tabla de símbolos

```
GlOBAL TABLE

Key             Type            Structure       Value
---             ----            ---------       ------
main            void            Function        void
sort            void            Function        ['Array', 'Const', 'Const']
minloc          int             Function        ['Array', 'Const', 'Const']
x               int             Array           10
hight           int             Const           None
low             int             Const           None
a               int             Array           None
```

Para la tabla me basé en el ejemplo del profesor, donde la Key, sirve para poder accesar a la información de cada fila además de que sirve para identificar el nombre de la variable. El Type lo utilizo para poder validar los tipos de cada variable, y me sirve para checar si una variable está bien declarada (gracias a su tipo), si la función es del mismo tipo que su return, o si los argumentos de una llamada son iguales a los parámetros que recibe la función. La Structure además de que nos ayuda a identificar el tipo de estructura del que se trata, ayuda para la declaración y llamada de funciones, ya que, si la función tiene como parámetros una structure de tipo 'Array', se iría a la parte de Value a buscar el arreglo que se crea, y en la posición en la que se manda el argumento, es la posición que se tiene en el array de value. Por último, el Value sirve para ver qué es lo que está almacenando la variable.

Estructura de datos

No ocupe el stack, sin embargo, utilicé una double linked list para poder recorrer por delante y por detrás los nodos (tablas) que se iban creando y fuera un tanto más fácil utilizar esto.

Definición del lenguaje

La definición del lenguaje se encuentra en el documento llamado ‘Lenguaje C-.pdf’ dentro de la carpeta previamente descargada en la carpeta ‘Apendices’.

Nota final

Para los archivos previamente descritos, se puede encontrar su archivo con extensión .pdf en la carpeta llamada ‘Apendices’. Estos se llamarán Lexer, Parser y Semantica.