Práctica 02

DOCENTE	CARRERA	CURSO
MSc. Vicente Enrique Machaca Arceda	Escuela Profesional de Ingeniería de Software	Compiladores

PRÁCTICA	TEMA	DURACIÓN
02	Scanner, Parser y Analizador semántico	3 horas

1. Datos de los estudiantes

- Grupo: 1
- Integrantes:
  - Guillermo Aleman Zambrano
  - Daniel Mendiguri Chavez
  - Daniela Vilchez Silva
  - Marvik Del Carpio Lazo
- Link Repositorio GitHub: https://github.com/vsdaniela/Compiladores-G1/tree/master/Practica

2. Ejercicios

1. Redacte el siguiente código en dos archivos distintos, luego compile cada archivo y explique a qué se debe el tipo de error del segundo código. (2 puntos).

```
1
2       int main(){
3           char* c = "abcdef";
4           int m = 11148;
5           return 0;
6       }
```

```
1
2       int main(){
3           char* c = "abcdef";
4           int 4m = 11148;
5           return 0;
6       }
```

Solución

Porque el nombre de la variable no debe iniciar con un número, porque así está determinado en el lenguaje.

2. Explique cuál es la función del Scanner o Analizador Léxico. (2 puntos)

   Solución

   Leer el código fuente e identificar errores léxicos y retornar los tokens.

3. Dado el siguiente código, en una tabla escriba todos los Tokens que un Scanner encontraría, detalle la clase y el lexema. Usted tiene la libertad de escoger el nombre de las clases. (4 puntos).

```
1
2     int main(){
3         char* c = "abcdef";
4         int m = 11148;
5         int x = m/8;
6         int y = m/4;
7         int z = m/2;
8         return 0;
9     }
```

Solución

```
1                 type_int:               int
2                 type_main:              main
3                 type_(:                 (
4                 type_):                 )
5                 type_{:                 {
6                 type_salto:             '\n'
7                 type_char:              char
8                 type_character:         *
9                 type_op_igual:          =
10                type_texto:             "abcdef"
11                dot_com:                ;
12                Salto:                  '\n'
13                type_int:               int
14                type_name:              m
15                type_op_equal:          =
16                type_num:               11148
17                dot_com:                ;
18                Salto:                  '\n'
19                type_int:               int
20                type_name:              x
21                type_op_equal:          =
22                type_name:              m
23                operator:               /
24                num:                    8
25                dot_com:                ;
26                Salto:                  '\n'
27                type_int:               int
28                type_name:              x
29                type_op_equal:          =
30                type_name:              m
31                type_op_div:            /
32                type_num:               8
33                dot_com:                ;
34                Salto:                  '\n'
```

```
35                  type_int:               int
36                  type_name:              y
37                  type_op_equal:          =
38                  type_name:              m
39                  type_op_div:            /
40                  type_num:               4
41                  dot_com:                ;
42                  Salto:                  '\n'
43                  type_int:               int
44                  type_name:              z
45                  type_op_equal:          =
46                  type_name:              m
47                  type_op_div:            /
48                  type_num:               2
49                  dot_com:                ;
50                  Salto:                  '\n'
51                  type_return:            return
52                  type_num:               0
53                  dot_com:                ;
54                  Salto:                  '\n'
55                  type_}:                 }
```

4. Redacte el siguiente código en dos archivos distintos, luego compile cada archivo y explique a qué se debe el tipo de error del segundo código. (2 puntos).

```
1       int main(){
2           int m = 11148;
3           int x = m + 100;
4           return 0;
5       }
```

```
1
2       int main(){
3           int m = 11148;
4           int x = m + ;
5           return 0;
6       }
```

Solución

Se debe a un error de sintaxis, porque el operador suma es un operador binario, y este sólo está recibiendo un elemento. El parser identifica elemento inicial seguido del operador, mas no del segundo sumando, por lo cual manda error.

5. Explique cuál es la función del Parser o Analizador Sintáctico. (2 puntos)

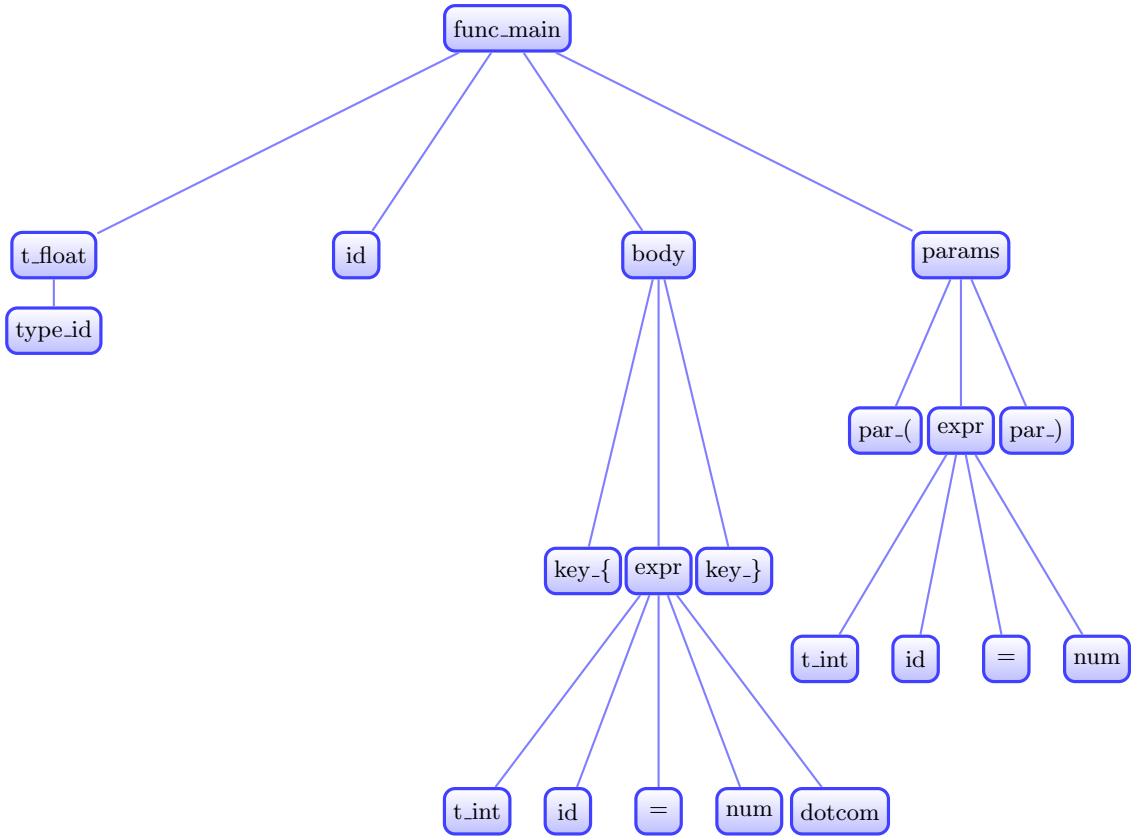
Solución

Leer los tokens, idetenfica errores sintácticos y retorna un árbol sintáctico.

6. Genere el árbol sintáctico para el siguiente código. Puede tomar como ejemplo el mostrado en clases. (4 puntos).

```
1       float main(int a = 5){
2           int x = 100;
3       }
```

Solución



7. Redacte el siguiente código en dos archivos distintos, luego compile cada archivo y explique a qué se debe el tipo de error del segundo código. (2 puntos).

```
1        int main(){
2            int m = 8;
3            int y = 3;
4            int x = m + y;
5            return 0;
6        }
```

```
1        int main(){
2            int m = "ceee";
3            int x = m + y;
4            return 0;
5        }
```

Solución
El primer error es semántico ya que le estamos asignando una cadena de caracteres a una variable de tipo int,y el segundo del mismo tipo ya que queremos hacer uso de una variable no declarada.

8. Explique cuál es la función del Analizador Semántico.(2 puntos).

   Solución

   Hace uso del árbol sintáctico y la información en la tabla de tokens para comprobar la coherencia del código fuente con la definición del lenguaje. En otras palabras, verifica que el código escrito concuerde en significado con el alfabeto del lenguaje.