LENGUAJE : ARTURITO

Creadores: Macarena Prado Torres, Ignacio Matías Vidaurreta y Natali Lilienthal

Índice

Idea subyacente y objetivo del lenguaje

La idea fue la creación de un lenguaje que sea fácil de aprender para los programadores que les cuesta un poco la sintaxis de CSS, los que están más acostumbrados a programar en lenguajes más “comunes”, así se evita tener que googlear toda la sintaxis por internet.
El lenguaje es capaz de crear un archivo css de lo programado.

Consideraciones realizadas (no previstas en el enunciado)

- Selectores: Los selectores en css son lo más importante es por eso que dedicamos un tipo de dato para los mismos, este tipo de dato tiene “hijos” que como si fuera Java “heredan” las características que tienen los mismos selectores.
- Tipos HTML: como el css se usa comúnmente en conjunto con HTML creamos tipos de datos propios de etiquetas HTML (p, body, h1, h2, div)
- Manejo de memoria: No quisimos que el usuario pueda manejar memoria ya que pensamos que este lenguaje debería ser lo más simple posible así que la memoria se crea y se libera internamente.
- Crear funciones de librería estándar (de nuestro lenguaje) para poder crear el css.
- El usuario debe hacer el llamado al “addSelector” una vez que hay a agregado toda la información pertinente al selector. No puede hacerlo antes, esta es una limitación de nuestro lenguaje.
- Para el cálculo del color (setColor()) Se debe pasar el color en formato **INT** que luego se pasará a hexadecimal.

Descripción del desarrollo del TP

1. Creación de la gramática
   Primero creamos la gramática en papel para que sea más fácil pasarla a código.
2. Creación del archivo yacc y lex
   Armamos los archivos para definir la gramática
3. Diseño del árbol
   Empezamos planteando toda la gramática como mencionamos en el paso 1 pero después era difícil diseñar el árbol entre tantos errores. Por ende, tomamos un approach más escalonado en el que íbamos agregando las producciones a medida que las necesitábamos.

Se utilizó un tipo de dato Node que dentro tiene un campo “node_type” que marca qué tipo de nodo es para luego extraer el comportamiento a través del union “NodeKind”. De esta manera a lo largo de todo el árbol podemos usar un tipo de dato sólo para mover entre nodo y nodo.

4. Creación de la “librería estándar”
   Creamos las diferentes funcionalidades especiales que iba a tener, en conjunto con los tipos especiales y sus respectivas funciones.
   También la creación del archivo css que se iba a crear internamente en el código.
5. Creación de la tabla de símbolos
   Se creó la funcionalidad de una hash table aplicada a los tipos de datos que necesitábamos para definir variables y funciones. Para el algoritmo de hashing se utilizó el hash djb2 ya que es un algoritmo sencillo con poca probabilidad de colisiones. También se decidió precargar las funciones admitidas por nuestro lenguaje Arturito almacenándolas en nuestra tabla de símbolos para agilizar la compilación .
6. Generación de código
   Generamos código en “c” como código de salida, para luego poder ejecutarlo con el gcc y que él mismo cree el archivo css.

Descripción de la gramática

- Tipos de datos:
  - Numérico (int, double)
  - String
  - Selector
  - Div (html)
  - P (html)
  - Body(html)
  - H1 (html)
  - H2 (html)
  - Id
  - Class

Es obligatorio el uso de ‘;’ para delimitar cada instrucción

- Constantes
- Operadores aritméticos “+, -, *, /”
- Operadores relacionales “==”, “!=”, “>=”, “>” “<=”, “<”
- Operadores lógicos “and, not, or ”

- Bloque Condicional (if).
    - if(condición)
      {

      }
- Bloque Do-While (repetición hasta condición).
    - While(condición)
      {
      }

Declaración:
Hay dos tipos de declaraciones

- Declaración de tipos especiales:
    - Selector l = id;
    - Selector s = div;
    - etc
- Declaración de tipos comunes:
    - int n = 2;
    - double d = 2.2;
    - String h = "hello";

Funciones propias de la librería especial:

- setColor(int n, Selector s);
- setFontSize
- setMargin(int n,Selector s);
- setMarginTop(int n,Selector s);
- setMarginBottom(int n,Selector s);
- setMarginRight(int n,Selector s);
- setMarginLeft(int n,Selector s);
- setPadding(int n,Selector s);
- setPaddingTop(int n,Selector s);
- setPaddingBottom(int n,Selector s);
- setPaddingRight(int n,Selector s);
- setPaddingLeft(int n,Selector s);
- setName(“poner un string asi”,Selector s);

Casos de Uso

TEST 1:
Codigo en file.art:

```
Selector l = CLASS;
setName("hola",l);
setColor(13340764, l);
addSelectors(l);
double dou = 2.2;
String h = "hello";
int w = 8;
```

Output:

StyleSheet.css:

```
#hola{
	color: #cb905c;
	margin: 0px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
}
```

TEST 2:
Codigo en file.art:

```
Selector l = CLASS;
setName("hola",l);
setColor(13340764, l);
addSelectors(l);

Selector h = ID;
setName("id",h);
setColor(13340764, h);
setMarginTop(2,h);
addSelectors(h);
double dou = 2.2;
```

Output:

```
#hola{
```

```
	color: #cb905c;
	margin: 0px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
}
.id{
	color: #cb905c;
	margin: 2px 0px 0px 0px;
	padding: 2px 0px 0px 0px;
}
```

TEST 3

Codigo en file.art:

```
int n = 3;
Selector h = H1;
setColor(13340764, h);
setMarginTop(n,h);
setMarginLeft(n,h);
setPadding(n,h);
addSelectors(h);
```

Output:

```
h1{
	color: #cb905c;
	margin: 3px 0px 0px 3px;
	padding: 3px 3px 3px 3px;
}
```

TEST 4

Codigo en file.art:

```
if(3>2 and 3>1){
Selector l = CLASS;
setName("hola",l);
setColor(13340764, l);
setFontSize(4,l);
addSelectors(l);

Selector h = ID;
setName("id",h);
setColor(13340764, h);
setMarginTop(2,h);
```

```
addSelectors(h);
double dou = 2.2;


}
```

Output:

```
#hola{
	color: #cb905c;
	font-size: 4px;
	margin: 0px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
}
.id{
	color: #cb905c;
	margin: 2px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
```

TEST 5

```
int x = 3;
while(x > 0){
Selector l = CLASS;
setName("hola",l);
setColor(13340764, l);
addSelectors(l);

Selector h = ID;
setName("id",h);
setColor(13340764, h);
setMarginTop(2,h);
addSelectors(h);
x = x - 1;

}
```

```
#hola{
	color: #cb905c;
	margin: 0px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
}
.id{
	color: #cb905c;
	margin: 2px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
}
#hola{
	color: #cb905c;
	margin: 0px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
}
.id{
	color: #cb905c;
	margin: 2px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
}
#hola{
	color: #cb905c;
	margin: 0px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
}
.id{
	color: #cb905c;
	margin: 2px 0px 0px 0px;
	padding: 0px 0px 0px 0px;
}
```

Dificultades encontradas en el desarrollo del TP

1. Aprendizaje y uso de yacc y lex:
   Para nosotros fue complejo entender el uso de lex y yacc y luego pasarlo al árbol.
2. Decisión de si ejecutar el código línea por línea o crear código intermedio
   Fue una decisión complicada, pero terminamos optando por crear código en C intermedio ya que nos pareció lo más simple y claro

3. Encontrar errores

    Nos fue complicado encontrar los errores y debuguear. Los mensajes de error sobre todos los dados por yacc eran en general poco descriptivos, nos sirvió bastante encontrar la forma de poner yacc en modo *debug* pero aún así resultaba críptico. GDB también fue de gran utilidad.

4. Creación del archivo .bnf

    Nos fue difícil automatizar la creación de este archivo. Por el poco tiempo restante nos vimos obligados a crearlo manualmente

Futuras extensiones, con una breve descripción de la complejidad de cada una

- Soporte de funciones

    Queremos que el usuario pueda poder crear sus propias funciones, no creo que esta funcionalidad sea muy compleja de soportar en alguna versión siguiente, seria agregar algunas reglas más en la gramática y agregar algunas funciones más para la creación de estas funciones.

- Soporte de colecciones

    Queremos soportar listas, arrays, mapas para poder manejar los selectores de manera más eficiente y poder recorrer los mismos para cambiar todos al mismo tiempo.

- Soporte de varios archivos

    Queremos que el usuario se pueda manejar en varios archivos así puede tener un código más modular.

- Soporte de más funcionalidad de CSS

    Queremos que el usuario pueda manejar todos los tipos que hay en css, para eso hay que agregar más funciones en la librería estándar de nuestro lenguaje y más tipos

Referencias

- Symbol table
- Hash table implementation
- Variable arguments
- Hash function: djb2
- Ejemplo de Flex y Bison
- Yacc y Flex
- How can I debug in YACC
- Generating a parser using yacc - IBM
- Oracle Docs - Lexical Analysis
- Documentación Bison