Documentación Externa Segunda Tarea Programada

Instituto Tecnológico de Costa Rica

Compiladores e Intérpretes

Jean Carlo Cervantes Rodríguez
Luis Prado Rodríguez
Joseph Araya Rojas

Tabla de Contenido

Descripción del Problema

El objetivo de esta tarea programada consiste en desarrollar un parser o analizador sintáctico para xhtml, utilizando herramientas automatizadas como Bison, además se puede hacer uso del scanner que se desarrolló para la tarea programada anterior, el cual se encargará de ir enviando los tokens necesarios para que se realice el análisis sintáctico.

Para el desarrollo de esta tarea se tendrá que establecer la gramática que permitirá reconocer el xhtml, además se debe poder obtener el árbol de análisis sintáctico resultante, así como reportar los errores que se presenten en esta etapa con su descripción y número de línea correspondiente.

Diseño del Programa

Tokens

Con respecto a la gramática se definieron los siguientes tokens con el fin de realizar un análisis sintáctico a un documento xhtml y generar el parse tree:

- T_dl, T_html, T_script, T_dt, T_img, T_span, T_a , T_dd, T_input, T_strong, T_b, T_em, T_li, T_style, T_blockquote, T_embed, T_link, T_table, T_body, T_footer, T_meta, T_td, T_br, T_form, T_object, T_th, T_button, T_h1, T_h2, T_h3, T_h4, T_h5, T_h6, T_ol, T_tr, T_caption, T_head, T_option, T_textarea, T_code, T_header ,T_p, T_title, T_div, T_hr, T_pre, T_ul: Son los tokens correspondientes a los elementos que se deben manejar definidos en la especificación del proyecto.
- T_Atributte: Este token contiene información del atributo leído por el scanner.
- T_Values: Es el token donde se almacena la información del valor de un atributo.
- T_Text : Es el token donde se guarda el texto entre una etiqueta.
- T_PublicIdentifier: Token de la palabra reservada “PUBLIC”.
- T_DOCTYPE: Token de la palabra reservada “DOCTYPE”.

Gramática

La producción inicial de la gramática es la siguiente:

"xhtml: doctype html_tag”: Esta producción contiene la estructura inicial que debe tener un xhtml, que es la etiqueta doctype de primera, luego la etiqueta html.

La gramática de las etiquetas de los elementos se manejo de la siguiente manera:

- Se creó una producción por cada elemento, de apertura de todas las etiquetas de los elementos. Estas producciones "head" tienen el formato <NombreElemento>_h.
- También se hizo una producción "tag" por cada elemento que contiene la estructura de la información dentro de una etiqueta, ya sea que contenga texto u otra etiqueta. Estas producciones tienen el formato <NombreElement>_tag.
- Para el manejo del cierre de una etiqueta se creó una producción "tail" por cada elemento que tiene la estructura del cierre de una etiqueta.
- Para manejar las etiquetas de los elementos se hizo una producción que contiene las producciones "head".
- Con respecto al elemento "body" la diferencia de la producción que se hizo es que tiene la estructura para manejar una o varias etiquetas dentro del "body" como debe hacerlo un documento "xhtml".
- Para las etiquetas de html, head, title y meta se hizo una producción para cada una correspondiente a la estructura correcta que deben tener para un documento "xhtml".

La gramática de los atributos y texto se manejo de la siguiente manera:

- Para los atributos se creó una producción que contiene la estructura de un atributo.
- Para el texto se creó una producción que maneja el texto dentro de una etiqueta.

Árbol Sintáctico

Elementos

El árbol sintáctico se diseño mediante typedef y struct de c:
Se crearon los siguientes elementos:

Atributo_t

```
typedef struct Atributo{//Representa atributos de ele
    char* nombreAtributo;
    char* valorAtributo;
    struct Atributo* siguiente;
}Atributo_t;
```

Representa los atributos HTML para cualquier elemento HTML en general (Excluyendo DocType).

Está compuesto por

1. **nombreAtributo:** Representa el nombre del atributo.
2. **nombreValor:** Representa el valor que tiene el atributo
3. **siguiente:** Es un puntero al siguiente atributo hermano.

Ejemplo:

id="validation-icon"

ListaAtributos_t

```
typedef struct ListaAtributos{
    Atributo_t* primerAtributo;
}ListaAtributos_t;
```

Representa la lista de atributos que puede contener un elemento HTML (Excluyendo DocType).

Está compuesta por:

1. **primerAtributo:** Puntero al primer elemento de la lista de atributos de un elemento HTML.

Ejemplo:

name="pdf-object" type="application/pdf"

Texto_t

```
typedef struct Texto{
    char* texto;
    struct Texto* siguiente;
}Texto_t;
```

Representa una palabra que puede estar dentro del contenido de un elemento HTML (Excluyendo DocType).

Está compuesto por:

1. **texto:** palabra que puede estar dentro del contenido HTML.
2. **Siguiente:** Puntero al siguiente texto hermano.

Ejemplo:

textohtml

ListaTextos_t

```
typedef struct ListaTextos{
    Texto_t* primerTexto;
}ListaTextos_t;
```

Representa todas las palabras que puede tener el contenido de un elemento html.

Esta compuesto por:

1. **primerTexto:** Puntero al primer texto del contenido de una etiqueta html.

DocType_t

```
typedef struct DocType{
    char* valor1;
    char* valor2;
}DocType_t;
```

Representa al doctype de un documento html.

Esta compuesto por:

1. **valor1:** Primer valor del doctype
2. **valor2:** Segundo valor del doctype.

Ejemplo:

<!DOCTYPE html PUBLIC valor1
Valor 2>

Nodo_t

```
typedef struct Nodo //Representa cada elemento del arb
{
    char* nombreNodo;
    ListaAtributos_t* listaAtributos;
    ListaTextos_t* listaTextos;
    struct Nodo* padre;
    struct Nodo* siguiente;//Siguiente nodo hermano(Es
    struct ListaHijos* listaHijos;
}Nodo_t;
```

Representa a los elementos html(Excluyendo Doctype).

Está compuesto: por

1. **nombreNodo:** Es el nombre del elemento html.
2. **listaAtributos:** Representa la lista de los atributos que puede contener un elemento html.
3. **listaTextos:** Representa el contenido de un elemento HTML
4. **padre:** Es un puntero al padre del elemento HTML actual.
5. **siguiente:** Es un puntero al hermano más cercano del elemento HTML actual
6. **listaHijos:** Es un puntero a la lista de hijos del elemento HTML actual

struct ListaHijos

```
struct ListaHijos{//Representa la lista de hijos que ti
    Nodo_t* primerHijo;//Primer hijo
};
```

Represnta la lista de hijos que un elemento html puede tener.

Esta compuesto por:

1. **primerHijo:** Puntero al primer hijo de un elemento html.

Arbol_t

```
typedef struct Arbol{
    Nodo_t* raizHtml;
    DocType_t* doctype;
}Arbol_t;
```

Representa al árbol sintáctico:

Está compuesto por

1. **raizHtml**: Representa al elemento principal del documento HTML
2. **doctype**: Representa al doctype del documento HTML.

Creación del árbol:

Variables

Para la creación del árbol se usan las siguientes variables

1. **listaActualAtributos:** Representa la lista de todos los atributos que ha leído el parser
2. **listaActualTextos**:: Representa la lista de todos los textos que ha leído el parser
3. **nodoActual:** Representa al último elemento HTML leído.

Proceso

El árbol sintáctico se va formando conforme el parser se va ejecutando:

1. **Al encontrar un atributo:** Se crea un Atributo_t * y se agrega a listaActualAtributos
2. **Al encontrar un token:** Texto Se crea un Texto_t* y se agrega a listaActualTextos
3. **Al encontrar una etiqueta Inicio:** Se re direcciona la lista temporal de textos al nodo actual. Se crea un nuevo Nodo_t * que apuntara a nodo actual y se le asigna la lista Actual de Atributos. Se crea una nueva lista Actual de Atributos
4. **Al encontrar una etiqueta de cierre:** Se asigna a nodoActual la lista Actual de textos, y se re direcciona nodoActual al padre de nodoActual.

Librerías Externas

Para el desarrollo de esta tarea programada no se hizo uso de ninguna librería externa, únicamente se utilizaron los componentes de Bison.

Análisis de Resultados

Rubro	Porcentaje Alcanzado
Especificación de la gramática sin conflictos	100%
Generación del árbol sintáctico correctamente	100%
Detección e informe de errores (detalle y fila)	100%

Manual de Usuario

1. Asegurarse de tener instalado flex y bison.
2. Abrir la Terminal.
3. Ubicarse en la dirección donde se encuentran los archivos del parser.
4. Ejecutar make en la terminal.
5. Ejecutar ./xhtml <nombre_del_archivo_de_prueba.

Conclusión Personal

Jean Carlo:

El uso de headers hace la programación más ordenada y útil, para detectar problemas de forma más sencilla y rápida.

Joseph:

La definición una gramática para un lenguaje es un factor clave y complejo, ya que se deben evitar los conflictos en la construcción de la misma, y al mismo tiempo asegurar la consistencia de dicha gramática a la hora de corregir los errores que se van presentando a lo largo de su creación.

Luis:

La creación del árbol de análisis sintáctico no es un proceso trivial ni sencillo, pues implica conocer ampliamente el lenguaje del cual se va a realizar el árbol para que este tenga todos los diferentes atributos necesarios para representar al lenguaje, además este no se va a crear en un solo proceso, su creación ocurre conjuntamente a la ejecución del parser, por lo tanto se debe entender y

conocer bastante bien la gramática para poder escoger las correctas producciones en su creación.