Universidad de San Carlos de Guatemala
Facultad de ingeniería
Escuela de ciencias y sistemas
Área de ciencias de la computación
Organización de lenguajes y compiladores 1
Segundo semestre 2019
Catedráticos: Ing. Manuel Castillo, Ing. Mario Bautista, Ing. Kevin Lajpop
Auxiliares: Nery Gálvez, Miguel Ruano, Erick Tejaxún

Primer proyecto de laboratorio 1

Usac Front End framework

Descripción

En la actualidad la mayoría de aplicaciones que son desarrolladas son de tipo web, es decir que no requiere instalación alguna y son accesibles desde un navegador web a través del internet. Por eso mismo existe la necesidad de crear herramientas que permitan a los desarrolladores de front end desarrollar aplicaciones web de manera modular, es decir poder realizar elementos haciendo así el desarrollo ágil.

El primer proyecto del curso se tratará de una aplicación que pueda desplegar aplicaciones gráficas al ejecutar un proyecto 'usac front end' [.ufe].

Objetivos

General

Que el estudiante pueda aplicar los conocimientos las fases de análisis léxico y sintáctico para poder desarrollar aplicaciones funcionales.

Específicos

- Que el estudiante sea capaz de utilizar herramientas para generación de analizadores léxicos para poder aplicarlo a soluciones de sistemas computacionales.
- Que el estudiante sea capaz de utilizar herramientas para generación de analizadores sintácticos para poder aplicarlo a soluciones de sistemas computacionales.
- Que el estudiante sea capaz de integrar ambas fases del proceso de compilación para generar soluciones de software.

Contenido

Descripción de la solución

El sistema OLC1 framework cuenta contará con una interfaz gráfica que permitirá la creación y edición de proyectos. Un proyecto constará de tres tipos de archivos:

- HTML set básico de html para la creación de la estructura principal de la vista de la aplicación.
- UFE lenguaje de programación que permitirá crear la vista y los componentes de la vista de la aplicación.
- CSS archivo que contendrá las hojas de estilo en cascada para que la interfaz pueda tener mejor apariencia y entregar una mejor experiencia al usuario.

Flujo de la aplicación

Al crear un proyecto 'usac front end' se creará una carpeta que será la raíz del proyecto que deberá desplegar una interfaz con un mensaje de bienvenida que será el número de carnet del estudiante. Un nuevo proyecto contendrá las siguientes rutas.

- ./public contendrá los archivos públicos que se mostrarán al usuario
- ./public/index.html Contendrá la vista principal de la aplicación.
- ./src contendrá los archivos que la aplicación ejecutará para generar la aplicación, tanto los archivos por defecto como los archivos que el usuario necesite.
- ./src/App.ufe Contendrá el archivo principal desde donde se ejecutará la aplicación
- ./src/App.css Contendrá hojas de estilos aplicables a los componentes declarados en el archivo app.ufe

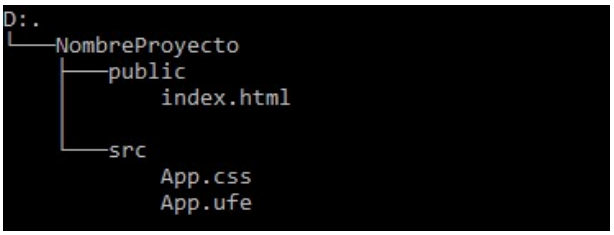


Ilustración 1 Archivos creados por defecto al iniciar un nuevo proyecto

```
D:.
└───NombreProyecto
    ├───public
    │       index.html
    │
    └───src
        │   App.css
        │   App.ufe
        │
        └───Components
            ├───Boton
            │       Boton.css
            │       Boton.ufe
            │
            ├───Display
            │       Display.css
            │       Display.ufe
            │
            ├───Formulario
            │       Formulario.css
            │       Formulario.ufe
            │
            └───Panel
                    Panel.css
                    Panel.ufe
```

```
D:.
└───NombreProyecto
    ├───public
    │       index.html
    │
    └───src
            App.css
            App.ufe
            Boton.css
            Boton.ufe
            Display.css
            Display.ufe
            Formulario.css
            Formulario.ufe
            Panel.css
            Panel.ufe
```

Ilustración 2 Ejemplos de directorio de proyecto personalizado

Flujo de ejecución

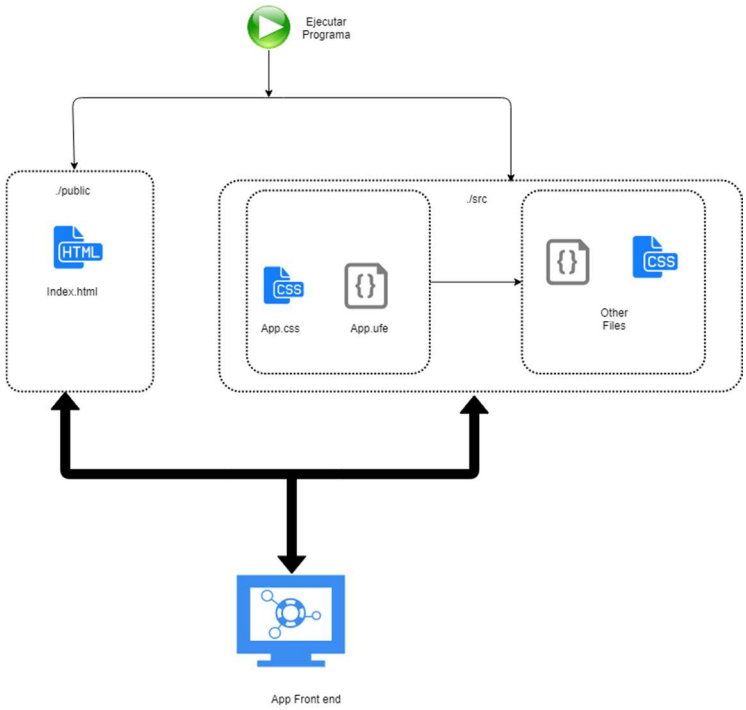


Ilustración 3 Flujo de ejecución de una aplicación

1. El usuario inicia la ejecución de su aplicación previamente desarrollada.
2. El programa analiza el archivo *./public/index.html* que contiene las directrices de la aplicación, es decir el encabezado y declaración del cuerpo.
3. El programa analiza el archivo *./public/app.ufe* que contiene las directrices para la generación de la interfaz desde el cual se pueden importar componentes de otros archivos en cualquier lugar de la carpeta *./src*.

4. El programa analiza el archivo *./public/app.css* que contiene hojas de estilos que el usuario podría utilizar desde el archivo *app.ufe*
5. El programa genera la aplicación y su interfaz mezclando los resultados del análisis del archivo *index.html* y el análisis del archivo *app.css*

Interfaz gráfica

Se deberá desarrollar una aplicación que cuente con una interfaz gráfica capaz de proveer al desarrollador las herramientas para poder desarrollar sus proyectos.

Se deberá de contar con las siguientes secciones:

1. Área de menú
    a. Crear Archivo
    b. Crear Proyecto
    c. Guardar Archivo
    d. Guardar como
    e. Cerrar Archivo
    f. Cerrar Proyecto
    g. Compilar proyecto
2. Área de edición: La herramienta debe permitir editar varios archivos a la vez.
3. Área de Reportes
    a. Área de errores: Debe mostrarse tabulados los errores encontrados durante la ejecución.
    b. Área de Consola: Consola de salida para los programas ejecutados.
4. Área de despliegue de interfaz de la aplicación: esta puede ser desplegada en una nueva ventana o bien en la interfaz como una nueva ventana.

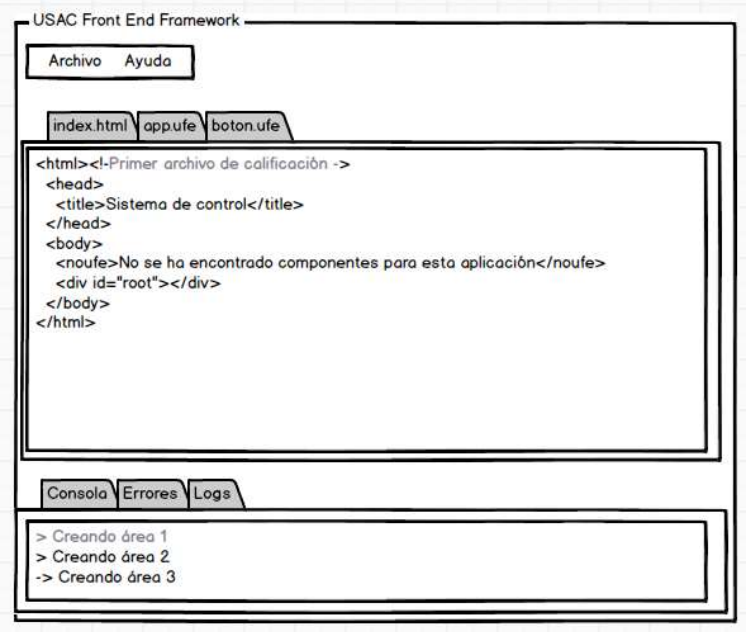


Ilustración 4 Sugerencia de interfaz gráfica

Descripción de los lenguajes

HTML reducido

Este lenguaje de marcado será un derivado de HTML el cual permitirá definir la estructura inicial de la vista de la aplicación. Es un lenguaje de etiquetas jerárquico, es decir una etiqueta pertenece a un nivel en el cuál puede existir más de una etiqueta. Este lenguaje será insensitive case, es decir no hará distinción entre minúsculas y mayúsculas.

Etiquetas permitidas

Como todos los componentes de la aplicación serán declarados a través del lenguaje [ufe], el set de etiquetas que contará este html reducido serán las siguientes:

Etiqueta	Descripción
HTML	Etiqueta inicial, en ella estará contenido toda la especificación de la interfaz de la aplicación. Solo podrá contener las etiquetas *head* y *body* .
Head	Esta etiqueta contendrá la etiqueta *title* que almacena el título de la aplicación. Este título debe ser mostrado en la parte superior de la ventana a mostrar.
Title	Esta etiqueta contendrá el título de la aplicación a desarrollar.
Body	Etiqueta donde contiene toda la interfaz gráfica de la aplicación. Solo contendrá las etiquetas *noufe* y la etiqueta *div*.
Noufe	Etiqueta que contiene un mensaje que se mostrará en caso de que la etiqueta div contenga errores.
Div	Etiqueta que sirve para marcar divisiones, es decir fragmentos que agrupan contenido. Esta será la única etiqueta que tendrá una propiedad, que se llamará id que recibirá como valor una cadena entre comillas que le dará nombre a esa sección que contendrá todo el contenido de la interfaz de la aplicación. En caso de que ese id no se encuentre en el archivo app.efu, se mostrará únicamente un texto con el contenido de la etiqueta *Noufe*.

Tabla 1 Set de etiquetas permitidas para el lenguaje html reducido

```
<html><!--Primer archivo de calificación -->
  <head>
    <title>Sistema de control</title>
  </head>
  <body>
    <noufe>No se ha encontrado componentes para esta aplicación</noufe>
    <div id="root"></div>
  </body>
</html>
```

Cuadro 1 Ejemplo de archivo index.html

Comentarios

Este lenguaje de maquetado permitirá comentarios. Para iniciar comentario se utiliza la palabra reservada <!-- y terminará con la palabra reservada -->

Lenguaje CSS

Es un lenguaje de diseño gráfico que nos permite definir y crear presentación de un documento, en este caso, la interfaz que se generará a través de los archivos ufe. Este lenguaje será insensitive case, es decir no hará distinción entre minúsculas y mayúsculas.

Un archivo CSS está compuesto de una lista de estilos que son agrupaciones de propiedades con un valor definido. Estas a su vez también pueden contener (adyacentemente) otros sub estilos dentro de ellos, es decir más sub estilos enlistados después de este. Estos sub estilos tienen la misma estructura que un estilo normal con la diferencia de comenzar con un punto antes del selector.

```
Selector: { (Propiedad: valor) (,Propiedad: valor)+}
(Adyacente)*
```

Cuadro 2 Estructura de un estilo

- Selector: Nombre (id) con el que se guardará el estilo.
- Propiedad: Nombre de la propiedad de la cual se definirá un valor.
- Valor: Valor de la propiedad. El valor dependerá de la propiedad a establecer.

```css
h1
{
  font-color: #282c34;
  font: 'Lucida Sans';
  background: #67ff00;
  align: center;
  border-color: #282c34;
  border-width: max;
}
```

Cuadro 3 Ejemplo de estilo CSS

Propiedades

Propiedad	Palabra reservada	Descripción	Valores posibles
Fondo	Background	Indica el color del fondo del elemento.	Color en formato hexadecimal. Nombre del color en inglés. Color en formato rgb.
Borde	Border	Indica la existencia del borde en el elemento.	Palabra reservada *true* y palabra reservada *false*.
Color de borde	Border-color	Indica el color del borde del elemento.	Color en formato hexadecimal. Nombre del color en inglés. Color en formato rgb.
Grosor de borde	Border-width	Indica el grosor del borde del elemento.	Número entero.
Alineación	Align	Indica la alineación del contenido del elemento.	Palabras reservadas *left*, *right*, *center*.
Fuente	Font	Indica la fuente a utilizar para mostrar textos.	Nombres de fuentes conocidas entre comillas simples.
Tamaño de fuente	Font-size	Indica el tamaño de los textos a mostrar.	Número entero.
Color de fuente	Font-color	Indica el color de los textos a mostrar.	Color en formato hexadecimal. Nombre del color en inglés. Color en formato rgb.
Alto	hight	Indica la altura del elemento.	Número entero.
Ancho	Width	Indica la anchura del elemento.	Número entero.

Tabla 2 Propiedades existentes en el lenguaje de diseño CSS

```
panel
{
     border: true;
     border-color: rgb(#333333, green, blue);
     border-width: 2;
}
.panelInput /*Sub estilo 1*/
{
     background: blue;
}
.panelOutput /*Sub estilo 2*/
{
     background: rgb(red, green, blue);
}
.Auxiliar //Sub estilo 3
{
     background: #ff7709;
}
```

Cuadro 4 Ejemplo de archivo CSS

Comentarios de una sola línea

El lenguaje de diseño CSS permite comentarios de una línea que comienzan con el lexema consistente en dos barras seguidas [//] y termina con un salto de línea [\n].

Comentarios multilíneas

El lenguaje de diseño CSS permite comentarios de varias líneas que comienza con el lexema consistente en una barra en una barra seguida de un asterisco [/*] y termina con el lexema consistente en un asterisco seguido de una barra [*/].

```
panel
{    /**Comentario
      *multi línea**/
     border: true; //Este panel tendrá activo el borde
     border-color: rgb(#333333, green, blue); //Se espera un color azul
}
```

Cuadro 5 Ejemplo de comentarios en archivo CSS

Nota

Los elementos a los cuales se le aplican estos estilos a través del lenguaje de diseño CSS pueden o no tener ciertos atributos que definirán su estilo desde su declaración, en ese caso se aplicarán primero esos estilos y luego se aplicará el estilo css que se la haya indicado, es decir el estilo CSS es el que prevalecerá.

Lenguaje Usac Front End (UFE)

Este será un lenguaje de programación que será el encargado de generar toda la interfaz gráfica de nuestras aplicaciones. Este lenguaje de programación estará compuesto por un conjunto de instrucciones que hará posible la creación de aplicaciones utilizando la modularidad para poder reutilizar componentes y hacer mucho más sencillo el desarrollo.

Insensitive case

Sera case issensitive, es decir, nohará distinción entre minúsculas y mayúsculas. Por ejemplo la palabra reservada "var" será igual al identificador "VAR".

Comentarios

Este lenguaje de programación soportará comentarios de una sola línea comenzando con la palabra reservada consistente en dos diagonales seguidas [//] y terminando con el carácter de salto de línea [\n]. También soportará comentarios multi línea que comienzan con la palabra reservada consistente en diagonal seguido de un asterisco [/*] y termina con la palabra reservada consistente en asterisco seguido de un diagonal [*/].

Tipos de datos permitidos

UFE es un lenguaje de programación con tipado débil, es decir que el tipo de una variable dependerá del valor que se le asigna, por lo tanto puede ir variando su tipo cuantas veces se le asigne un valor. Los tipos de datos permitidos serán tipos de datos primitivos, como lo muestra el cuadro siguiente:

Tipo	Descripción	Ejemplo
Carácter	Un único carácter encerrado entre comillas simples.	'a' 'x'
Cadena	Conjunto de caracteres encerrados entre comillas dobles.	"a" "compi 1"
Entero	Valor de numérico sin decimales con tamaño de 4 bits, es decir con un mínimo de -2147483647 y un máximo de 2147483648.	33 66 1693
Doble	Valor de tipo numérico con decimales con un tamaño de 32 bits.	3.1416 666.661 777.122340
Booleano	Valor que representa un valor lógico.	true false

Tabla 3 Tipos de datos primitivos soportados por UFE

Variables

El lenguaje de programación UFE soportará la declaración, asignación y utilización de variables con valores de tipo primitivo.

Sentencias UFE

Declaración de variables

Para declarar una variable (es decir apartar un espacio en memoria para una variable) se sigue la siguiente sintaxis:

```
Declaracion -> var listaAsignaciones;

listaAsignaciones -> listaAsignaciones  , identificador ( = Valor)?
                                      | identificador ( = Valor)?
```

Cuadro 6 Sintaxis declaración de variables

Asignación a variable

Para asignarle un valora una variable esta debe estar previamente declarada, y sigue la siguiente sintaxis:

```
Asignación ->  identificador  = Valor ;
```

Cuadro 7 Sintaxis asignación a variables

Operaciones

UFE permitirá realizar operaciones entre valores para poder obtener un resultado final.

Aritméticas

UFE soportará las siguientes operaciones aritméticas

Adición (Suma)

Se podrá realizar adición, una operación binaria, utilizando la palabra reservada +, y para esta se tendrá el siguiente sistema de tipos:

Operando	Tipo Resultante	Ejemplo
Cadena + cadena Cadena + entero Cadena + doble Cadena + carácter Cadena + booleano Entero + cadena Doble + cadena Carácter + cadena Booleano + cadena	Cadena	“hola ” + “ olc1” = “hola olc1” “hola” + 33 = “hola33” “hola”+3.33 = “hola3.33” “hola” + ‘a’ = “holaa” “hola” + true = “holatrue” 33 + “hola” = “33hola”
Doble + entero Entero + doble Doble + carácter Carácter + Doble	Doble	10 + 3.1416 = 13.1416 3.1416 + 10 = 13.1416 10.0 + ‘a’ = 107.0 ‘a’ + 10.0 = 107.0
Entero + carácter Carácter + entero Carácter + carácter	Entero	10 + ‘a’ = 107 ‘a’ + 10 = 107 ‘a’ + ‘a’ = 194

Tabla 4 Sistema de tipos para la adición

Cualquier otra combinación dará como resultado un error y deberá ser reportado.

Resta

Se podrá realizar resta, una operación binaria, utilizando la palabra reservada - , y para esta se tendrá el siguiente sistema de tipos:

Operando	Tipo Resultante	Ejemplo
Doble - entero Entero - doble Doble - carácter Carácter - Doble	Doble	100.05- 3 = 87.95 3 – 2.5 = 0.5 10.0 - ‘a’ = -87.0 ‘a’ - 10.0 = 87.0
Entero - carácter Carácter – entero Carácter – carácter	Entero	10 - a = 107 a - 10 = 107 a –b = -1

Tabla 5 Sistema de tipos para la resta

Cualquier otra combinación dará como resultado un error y deberá ser reportado.

Producto

Se podrá realizar multiplicaciones, una operación binaria, utilizando la palabra reservada *, y para esta se tendrá el siguiente sistema de tipos:

Operando	Tipo Resultante	Ejemplo
Doble * entero Entero * doble Doble * carácter Carácter*Doble	Doble	100.05* 3 = 300.15 3 * 2.5 = 7.5 10.0 * 'a' = 970.0 'a' * 10.0 = 970.0
Entero * carácter Carácter * entero Carácter * carácter	Entero	10 * a = 970 a * 10 = 970 a * b = 9506

Tabla 6 Sistema de tipos para el producto

Cualquier otra combinación dará como resultado un error y deberá ser reportado.

División

Se podrá realizar resta, una operación binaria, utilizando la palabra reservada /, y para esta se tendrá el siguiente sistema de tipos:

Operando	Tipo Resultante	Ejemplo
Doble / entero Entero / doble Doble / carácter Carácter/ Doble	Doble	100.00/ 3 = 33.333333 3 / 1.0 = 3.0 100.0 / 'a' = 1.0309278 'a' / 10.0 = 0.97
Entero / carácter Carácter / entero Carácter / carácter	Entero	10 / a = 1 a / 10 = 9 a / b = 0

Tabla 7 Sistema de tipos división

Cualquier otra combinación dará como resultado un error y deberá ser reportado.

Potencia

Se podrá realizar resta, una operación binaria, utilizando la palabra reservada *pow*, y para esta se tendrá el siguiente sistema de tipos:

Operando	Tipo Resultante	Ejemplo
Doble pow entero Entero pow doble Doble pow carácter Carácter pow Doble Entero pow carácter Carácter pow entero Carácter pow carácter	Doble	100.00 pow 3 = 100000.00 3 pow 1.0 = 3.0 1.0 pow 'a' = 1.0 'a' pow 2.0 = 9409.0 a pow 1 = 97.0

Tabla 8 Sistema de tipos potencia

Cualquier otra combinación dará como resultado un error y deberá ser reportado.

Relacionales

Estas operaciones son operaciones binarias. Estas tienen la finalidad de comparar los valores de los operando y su resultado es un valor booleano. Las operaciones relacionales soportadas por el lenguaje UFE son las siguientes:

- Igualdad (==)
- Desigualdad (!=)
- Mayor que (>)
- Menor que (<)
- Menor o igual que (<=)
- Mayor o igual que (>=)

Operando	Operador	Ejemplo
Entero [operador] entero Entero [operador] doble doble [operador] Entero doble [operador] doble entero [operador] carácter carácter [operador] entero carácter [operador] doble doble [operador] carácter carácter [operador] caracter	>,< , <=, >=	10 > 2 = true 10.0 > 33 = false 10>= 10.0 = true 10<= 10.0 = true 'a' < 'b' = true 'a' > 'b' = false
Todas las operaciones de la fila anterior. Cadena [operador] cadena	= , !=	"hola" == "hola = true "hola" != "hola" = false

Tabla 9 Sistema de tipos para operaciones relacionales

Lógicas

Estas operaciones son operaciones binarias que soportan únicamente como operando valores booleanos. Las operaciones lógicas soportadas por el lenguaje UFE son las siguientes:

- Operación lógica and: Se utilizará la palabra reservada &&.
- Operación lógica or: Se utilizará la palabra reservada ||
- Operación lógica xor: Se utilizará la palabra reservada ^
- Operación lógica not: Se utilizará la palabra reservada ! , esta es una operación unaria.

Operando A	Operando B	&& (and)	|| (or)	^ (xor)	! (not)
true	True	True	True	False	False
True	false	False	True	True	False
False	True	false	True	True	True
False	False	False	False	false	True

Tabla 10 Tabla de verdad para operaciones lógicas

Renderizado

El método *Render* es el encargado de iniciar el despliegue de la interfaz gráfica de nuestra aplicación. Este método recibe como parámetros el nombre del componente que contiene el contenido de la aplicación gráfica en formato ufex, el segundo parámetro indica el nombre del elemento html que tendrá que buscar el compilador para insertar el contenido.

```
Render(< idComponente />, idDiv);
```

Cuadro 8 Sintaxis llamada a función render

- idComponente: Nombre del componente que será insertado.
- idDiv: Nombre del div (separador) que se encuentra en el archivo html para insertar el contenido en esta parte. De no existir debe mostrarse el mensaje de error que se encuentra en la etiqueta *noufe* .

Componente

Los componentes serán funciones todas las instrucciones. La instrucción *return* es obligatoria y es la cual contendrá como parámetro todo el contenido de ese componente. Siguiendo la siguiente sintaxis:

```
Component idComponente
{
  return (contendido );
}
```

Cuadro 9 Sintaxis de declaración de un componente

- component: palabra reservada que indica que se trata de un componente.
- idComponente: Identificador que se le dará a ese componente.
- Contenido: es el contenido del componente en formato ufex.

Formato UFEX

El formato ufex es un formato que combina un lenguaje de etiquetas (en este caso html) con el lenguaje UFE. En este formato se indicará la estructura de cada componente que formará la aplicación.

El lenguaje predominante es el html pero se podrá incrustar porciones de código UFE para poder utilizar valores de variables o bien indicar que se insertarán más componentes.

Componentes html soportados por UFE

UFE soportará muchos componentes HTML que seguirán el siguiente formato

```
< idComponenteHTML (atributos)?>
Contenido
</ idComponenteHTML>
```

Cuadro 10 Sintaxis Componente HTML en formato UFEX

- idComponenteHTML: Es la palabra reservada que indica el tipo del componente a insertar.
- Contenido: Este será el contenido del componente a insertar. Este variará según sea el componente a insertar y será indicado a detalle en las secciones siguientes.
- Nameclass: Palabra reservada que indica que estilo CSS se aplicará.
- CSSEstilo: Indica el nombre del estilo que se aplicará. En caso de ser un sub estilo se indicará el nombre del estilo y luego el sub estilo a aplicar.
  Por ejemplo:
  - "estilo1 subestilo"
  - "botton alerta"
- Atributos: Indica una lista de asignaciones tipo <atributo = valor> para establecer las propiedades de los distintos componentes. Estas varían según el componente. Se indicará a detalle cuando se explique cada componente.

Los atributos comunes a todos los elementos se muestran en la tabla siguiente, ninguno es obligatorio y en caso de no estar, el elemento tendrá el valor por defecto indicado en dicha tabla.

Descripción	Palabra reservada	Valor	Ejemplo	Valor por defecto
Identificar único para el elemento	id	identificador	Id = panel1 Id = panelMain	No aplica
Posición en el eje x donde comenzará a dibujarse el elemento.	X	Numero entero	X = 100 X = {posX}	0
Posición en el eje y donde comenzará a dibujarse el elemento.	y	Numero entero	y = 100 y = {posX}	0
Alto del elemento.	height	Numero entero	High = 200 High = {alto}	100
Ancho del elemento.	width	Numero entero	width= 200 width= {widthB}	100

Color de fondo.	Color	Nombre del color encerrado entre comillas.
		Color en formato Hexadecimal
			Color = "blue"
			Color = "BLUE"
			Color = #ffffff
			Color = #ff00aa
				"white"

Grosor del borde que rodea el elemento. El color del borde siempre será de color negro. Estará en un rango de 0 a 4.	Border	Numero entero	Border = 1 Border = 2 Border = 3	0
Nombre del estilo recolectado en el archivo CSS a aplicar.	ClassName	Indica el nombre del estilo que se aplicará. En caso de ser un sub estilo se indicará el nombre del estilo y luego el sub estilo a aplicar.	className = "panelito" className = "pan sub1"	No aplica

Tabla 11 Descripción de los atributos de los elementos

El cuadro anterior muestra los atributos en común de todos los componentes pero estos pueden tener componentes especiales, esos serán indicados en la presentación de cada uno de los componentes.

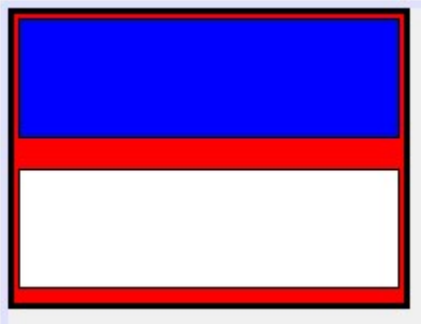
Esta es la lista de componentes soportados por el lenguaje UFE:

Componente panel

Este componente tiene como objetivo ser un contenedor de elementos.

```
component App() {
     return (
          <Panel id= primary x = 0  y = 0 height = 150 width = 200 color = "red" border = 2>
               <Panel id = second x = 5  y = 5 height = 60 width = 190 color = "blue" border = 1>
               </Panel>
               <Panel id = tercero x = 5  y = 80 height =60  width = 190 className = “panel ter”>
               </Panel>
          </Panel>
     );
   }
```

Cuadro 11 Ejemplo de declaración de un panel



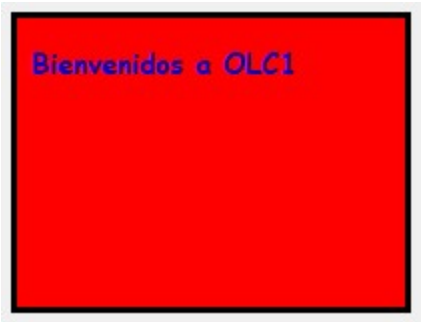
Cuadro 12 Elemento creado basado en el cuadro 11

Componente Text

Este componente tiene como objetivo ser un contenedor de una cadena de texto. El único contenido que puede tener este componente es una cadena que puede o no estar vacía

```
component panelConText() {
     return (
          <Panel id= primary x = 0  y = 0 height = 150 width = 200 color = "red" border = 2>
                <text x = 10 y = 10 height = 10 width = 20 className = “titulo1”>
                {mensaje_bienvenida}
                </text>
          </Panel>
     );
   }
```

Cuadro 12 Ejemplo de declaración de un componente text



Cuadro 13 Elemento creado basado en el cuadro 12

Componente TextField

Este componente representa a una caja de texto desde el cual el usuario puede ingresar datos. El único contenido que puede tener este componente es una cadena que puede o no estar vacía que será el texto que tendrá por defecto el elemento.

```
component PanelConTexField() {
     return (
          <Panel id= primary x = 0  y = 0 height = 150 width = 200 color = "red" border = 2>
                <text x = 10 y = 10 height = 10 width = 20 className = “titulo1”>
                {mensaje_bienvenida}
                </text>
                <textfield y = 40 x = 10 classNmae = “field”>{mensaje_bienvenida
                </textfield>
          </Panel>
     );
   }
```

Cuadro 14 Ejemplo de declaración de un textfield

Componente Button

Este componente representa un botón que podrá ser presionado y que únicamente desplegará una alerta con un mensaje indicado.

```
component PanelConBoton() {
     return (
          <Panel id= primary x = 0  y = 0 height = 150 width = 200 color = "red" border = 2>
                <button x = 10 y = 10 height = 30 width = 30 onclick = {variable10}>{nombreBoton}
                </button>
          </Panel>
     );
   }
```

Cuadro 15 Ejemplo de declaración de un button

En el área de contenido del botón únicamente podrá ir el texto que se mostrará en el botón. Este componente cuenta con un atributo llamado onclick al cual se le asignará un valor que se imprimirá en el cuadro de texto que se muestre cuando se presione el botón. Este valor puede ser de cualquier tipo y se convertirá en una cadena texto.

Componente List

Este componente representa un componente desplegable que mostrará una lista de valores posibles indicados en su declaración.

```
component ListaElementos() {
    return (
        <Panel id= primary x = 0  y = 0 height = 150 width = 200 color = "red" border = 2>
            <list x = 100 y = 100 className = “lista”>
                <elements>
                    <ítem> {item1}</ítem>
                    <ítem> “Física aplicada”</ítem>
                    <ítem> “Organización computacional”</ítem>
                </element>
                <default> 3 </default>
            </list>
        </Panel>
    );
  }
```

Cuadro 16 Ejemplo de declaración de un List

En el área de contenido de este componente podrá tener uno o elementos que son los siguientes:

- Elements: Este elemento es obligatorio y contiene una lista de ítems como se muestra en la figura anterior.
- Default: Este elemento es opcional, en caso de no estar el valor por defecto será el primer elemento. Pero en caso de estar indicará el índice del elemento que estará seleccionado por defecto en la lista desplegable.

Componente Spinner

Este componente representa un componente que permite la selección de un valor numérico. Este elemento contará con dos atributos especiales, los cuales son:

- Min: Indica el valor más pequeño que puede ser seleccionado a través de ese elemento. Puede ser un número negativo. En caso de no venir el valor por defecto será -100.
- Max: Indica el valor más grande que puede ser seleccionado a través de este elemento. Puede ser un número negativo. En caso de no venir el valor por defecto será 100.

El valor por defecto estará en el área de contenido del componente.

```
component PanelConEspiner() {
     return (
          <Panel id= primary x = 0  y = 0 height = 150 width = 200 color = "red" border = 2>
                <spinner x = 100 y = 100 className = “spinnerPro” max = 1000 min = -100> 666
                </ spinner >
          </Panel>
     );
   }
```

Cuadro 17 Ejemplo de declaración de un Spinner

Componente Image

Este componente tiene como finalidad mostrar una imagen. Este elemento tendrá un atributo especial llamado src al cual se le podrá asignar un valor de tipo cadena que indicará la ruta de la imagen a mostrar. Este será el único elemento que no contará con una etiqueta de cierre ya que constará únicamente

```
component App() {
     return (
          <Panel id= primary x = 0  y = 0 height = 500 width = 500 color = "red" border = 2>
                <image x = 100 y = 100  height = 100 width = 200 src = “profile.jpg”>
          </Panel>
     );
   }
```

Cuadro 18 Ejemplo de declaración de un Image

Sentencia import

Esta sentencia permitirá traer elementos de otros archivos indicando su ruta relativa es decir que la ruta comienza desde la localización del archivo donde se use la sentencia.

Esta sentencia tiene dos utilidades distintas, importar estilos a través de indicar la ruta de la hoja de estilos a utilizar e importar componentes existentes en otros archivos.

Import CSS

Para poder importar una lista de estilos contenidos en un archivo de estilo gráfico CSS se seguirá la siguiente sintaxis:

```
import “RutaHaciaArchivo.css” ;
```

Cuadro 19 Sintaxis importación de archivo CSS

En caso de realizar varias importaciones CSS, se aplicarán únicamente los estilos de la última importación.

Import Component

Para poder importar componentes de diferentes archivos se utilizará la siguiente sintaxis:

```
Import id from “rutaHAciaArchivo.ufe” ;
```

Cuadro 20 Sintaxis importación de componente

Donde

- Id: será el identificador del componente a agregar, este hace referencia al nombre del componente declarado en el archivo indicado. Recordar que pueden haber uno o más componentes dentro de un archivo UFE.

Insertar un elemento importado

La utilidad de importar un elemento es el poder insertar ese elemento las veces que sea necesario. Para su inserción dentro de un componente bastará con la siguiente sintaxis:

```
</idComponente>
```

Cuadro 21 Sintaxis para la inserción de un componente

```
import "./estilos1.css";
/* Importación desde elementosForm1 */
import Boton1 from "./elementosFomr1.ufe";
import Sppiner1 from "./elementosFomr1.ufe";
import tituloF from "./elementosFomr1.ufe";
/* Importación desde otro archivo */
import Boton2 from "./componentes/utilidades.ufe";
import BotonAlerta from "./componentes/utilidades.ufe";
component App()
{
     Var contador = 100;
     return (
          <Panel id= primary x = 0  y = 0 height = 1000 width = 1000 color = "red" border = 2>
               </boton1>
               </Sppiner1>
               </tituloF>
               <text x = 300 y = 300 className = "titulo segundo"> Segundo parte de
elementos</text>
               </boton2>
               </BotonAlerta>
          </Panel>
     );
 }
```

Cuadro 22 Ejemplo de utilización de elementos importados

Instrucción SI, SINO, SINO-SI

Esta instrucción recibe una condición booleana y si esta se cumple, se ejecutará la lista de instrucciones que traiga consigo. Así mismo pueden venir más condiciones Sino si con sus respectivas condiciones. Si ninguna condición se cumple puede o no que venga la instrucción sino, la cual no tiene condición que evaluar. La sintaxis es la siguiente:

```
import "./estilos1.css";
component App()
{
       /* Lista de sentencias iniciales */
       Si (condición) {
              Lista Instrucciones …
       } Sino Si (6 > 8) {
              Lista Instrucciones…
       } Sino Si (var1 == true) {
              Lista Instrucciones…
       } sino {
              Lista Instrucciones …
       }
       /* Otra forma de utilizar la instrucción Si */
       SI (condición) {
              Lista Instrucciones …
       }

       /* Otra forma de utilizar la instrucción Si */
       Si (condición){
              Lista Instrucciones …
       } sino {
              Lista Instrucciones …
       }
   return (
         …
    );
 }
```

Cuadro 23 Sintaxis de un if

Ciclo Repetir

Este ciclo ejecutará el número de veces que se indique, según el valor entero que se le envié.

```
Repetir (<NUMERO DE VECES>) {
       Instrucciones …
}
```

Cuadro 24 Sintaxis de un Repetir

Ejemplo:

```
component App()
{
       Repetir ((5 - var1) * var2) {
              Imprimir (“Iteracion #” + contador) ;
              Contador = Contador + 1 ;
       }
       return (
          …
       );
 }
```

Cuadro 25 Ejemplo de un Repetir

Ciclo Mientras

Este ciclo ejecutará su contenido siempre que se cumpla la condición que se le dé por lo que podría no ejecutarse si la condición es falsa desde el inicio. La estructura de un ciclo “mientras” es la siguiente:

```
Mientras (<CONDICION>) {
   LINSTRUCCIONES … ;
 }
```

Cuadro 28 Sintaxis de un Mientras

Ejemplo:

```
component App()
{
       Var condición = true;
       Mientras (condición) {
              Imprimir(“ciclo…”) ;
       }
}
```

Cuadro 29 Ejemplo de un Mientras

Función Nativa Imprimir

Esta función será llamada a través de la palabra reservada "imprimir" y contará con un único parámetro que puede ser de cualquier tipo y cualquier tipo de operación. Se agregará el texto resultante a la consola de salida. Tendrá la siguiente sintaxis:

```
Imprimir (<OPERACION>);
```

Cuadro 30 Sintaxis de un Imprimir

Ejemplo:

```
import "./estilos1.css";
component App()
{
       Imprimir ("Debo apresurarme" + " sino perderé ") ;
       return (
          …
       );
 }
```

Cuadro 31 Ejemplo de un Imprimir

Arreglos

Un arreglo es una colección de datos que se encuentran ubicados por medio de un índice dentro de una misma variable.

UFE soporta únicamente arreglos de una sola dimensión y dada la naturaleza del lenguaje de programación UFE, los arreglos pueden o no ser homogéneos, es decir que un arreglo puede almacenar varios tipos de datos.

UFE soporta dos tipos de declaración de arreglos, una en la cual se le indica la longitud del arreglo y otra en el cual la longitud del arreglo viene dado por el valor asignado.

Acceso a elemento de arreglo

Para obtener el valor de un arreglo bastará con indicar el índice que representa la posición del elemento dentro del arreglo. Cabe resaltar que en UFE los arreglos inician en la posición 0.

```
ACCESO → identificador [Expresion] ;
```

Cuadro 13 Sintaxis acceso a elemento de un arreglo

Primera forma de declaración de arreglo

Esta forma de declaración, instanciará un arreglo con tamaño del número indicado y todos sus valores serán inicializados con un valor numérico cero (0) .

```
DECARR → 'var' identificador [ Expresion ] ;
```

Cuadro 14 Sintaxis declaración de arreglo

Segunda forma de declaración de arreglo

Esta forma de declaración, instanciará un arreglo con tamaño del número indicado y todos sus valores serán inicializados con un valor numérico cero (0) .

```
DECARR → 'var' identificador { LISTAEXPRESIONES } ;
```

Cuadro 15 Sintaxis declaración de arreglo

Donde listaExpresiones representa a una lista de expresiones separadas por coma.

Asignación de valores a elementos del arreglo

La sintaxis para la asignación de un valor a un elemento del arreglo será la siguiente:

```
ASIGITEM → identificador [ Expresion ] = Expresion ;
```

Cuadro 16 Sintaxis asignación de elemento de arreglo

```
import "./estilos1.css";
component App()
{
     var lista [10];
     lista[0] = 10;
     lista[2] = lista[10] + " = diez";

     var lista2 = {1,1.1, 20*20/20, lista[0]*2 , "hola mundo"};
     imprimir(lista[0] + lista2[4]);
}
```

Cuadro 17 Ejemplo de uso de arreglos

Requisitos mínimos

- Interfaz gráfica
    - Abrir
    - Guardar
    - Guarda como
    - Edición de múltiples pestañas
    - Mostrar reportes
    - Consola
    - Tabla de símbolos
- HTML reducido
    - Todos los tags
- CSS
    - Estilos completos
    - Sub estilos
- UFE
    - Declaración de variables
    - Asignación de variables
    - Ciclos
    - Arreglos
    - Sentencia if
    - Componentes
        - Panel
        - Texto
        - Botones
        - Lista
        - Spinner
- Manuales técnicos y de usuario

Consideraciones

- La aplicación debe ser desarrollada en el lenguaje de programación Java.
- El IDE queda a discreción del estudiante.
- Los analizadores deberán ser implementados utilizando las herramientas JFlex y CUP.
- Se calificará a través de la ejecución de los archivos .jar enviados y es responsabilidad del estudiante haberlos generado.
- El nombre del entregable será [OLC1]Proyecto1_#carnet.zip
- Copias tendrán un punteo de 0 y serán reportados a las autoridades correspondientes.
- No se aceptarán entregas tardes.

Fecha límite de entrega 18 de septiembre de 2019 a las 23:59.