UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
LENGUAJES FORMALES Y DE PROGRAMACIÓN
SEGUNDO SEMESTRE 2023

Sección	Catedrático	Tuto académico
A-	Inga. Vivian Damaris Campos González	Enrique Alejandro Pinula Quiñonez
A+	Ing. Otto Amílcar Rodríguez Acosta	Erick Daniel Antillón Chinchilla
B-	Inga. Zulma Karina Aguirre Ordoñez	Elder Anibal Pum Rojas
B+	Ing. David Estuardo Morales Ajcot	Francisco Magdiel Asicona Mateo

Proyecto #2: BizData

Objetivos

- Que el estudiante implemente una solución de software implementando los conceptos vistos en clase y laboratorio.
- Que el estudiante implemente un analizador sintáctico utilizando los conceptos de gramáticas independientes de contexto y árboles de derivación.
- Introducir al estudiante a la ejecución de instrucciones en un lenguaje de programación.

Descripción

BizData (Business Data Analysis) es una plataforma diseñada para que las pequeñas empresas puedan tomar decisiones fundamentadas y estratégicas basadas en el análisis profundo de sus datos comerciales. La tarea del estudiante de Lenguajes Formales y de Programación es crear un analizador léxico y sintáctico, utilizando Python, que permita a las empresas cargar y analizar datos estructurados en un formato especializado con extensión “.bizdata”.

DESCRIPCIÓN DEL LENGUAJE

Importación de Datos:

La importación de datos, los cuáles se utilizarán más adelante en los reportes viene declarada en dos partes:

Sección de Claves: En esta sección se declaran los claves o campos por los que están construidos los registros, su estructura está formada por la palabra reservada Claves, seguido de signo igual, corchete de apertura, lista de claves y corchete de cierre.

Lista de claves está formada por cadenas de caracteres encerradas entre comillas y separadas por coma.

```
Claves = [
    "clave_1", "clave_2", "clave_3", "clave_4"
]
```

```
Claves = [
    "codigo", "producto", "precio_compra",
    "precio_venta", "stock"
]
```

Sección de Registros: En esta sección se detallan los registros que se quieren analizar y sigue la estructura dada por palabra reservada Registros, signo igual, corchete de apertura, lista de registros y corchete de cierre.

Lista de registros: Cada registro está encerrado entre llave de apertura y llave de cierre y sus valores están separados por comas, estos valores pueden ser cadenas de texto, enteros o decimales.

```
Registros = [
    {valor1, valor2, valor3, valor4}
    {valor1, valor2, valor3, valor4}
    {valor1, valor2, valor3, valor4}
    {valor1, valor2, valor3, valor4}
]
```

```
Registros = [
    {1, "Barbacoa", 10.50, 20.00, 6}
    {2, "Salsa", 13.00, 16.00, 7}
    {3, "Mayonesa", 15.00, 18.00, 8}
    {4, "Mostaza", 14.00, 16.00, 4}
]
```

Comentarios:

Comentarios de una línea: Se representan con un numeral y finalizan con un salto de línea.

```
# Comentarios
```

Comentarios multilínea: Inicia con tres comillas simples y finaliza con tres comillas simples.

```
'''
comentario multilinea
'''
```

Instrucciones de Reportería:

- imprimir(cadena): Imprime por consola el valor dado por la cadena.

```
imprimir("Reporte de ");
imprimir("Abarrotería");
>>> Reporte de Abarrotería
```

- imprimirln(cadena):

```
imprimirln("Reporte de ");
imprimirln("Abarrotería");
>>> Reporte de Abarrotería
>>> Abarrotería
```

- conteo(): Imprime por consola la cantidad de registros en el arreglo de registros.

```
conteo();
>>> 46
```

- promedio(“campo”): Imprime por consola el promedio del campo dado.

```
promedio("stock");
>>> 6.25
```

- contarsi(“campo”, valor): Imprime por consola la cantidad de registros en la que el campo dado sea igual al valor dado.

```
contarsi("stock", 0);
>>> 0

contarsi("stock", 1);
>>> 18

contarsi("stock", 2);
>>> 7
```

- datos(): Imprime por consola los registros leídos.

```
datos();
>>> codigo  producto  precio_compra precio_venta stock
>>> 1       Barbacoa  10.50         20.00        6
>>> 2       Salsa     13.00         16.00        7
>>> 3       Mayonesa  15.00         18.00        8
>>> 4       Mostaza   14.00         16.00        4
```

- sumar(“campo”): Imprime en consola la suma todos los valores del campo dado.

```
sumar("stock");
>>> 25
```

- max(“campo”): Encuentra el valor máximo del campo dado.

```
max("precio_venta");
>>> 20.00
```

- min(“campo”): Encuentra el valor mínimo del campo dado.

```
min("precio_compra");
>>> 10.50
```

- exportarReporte(“titulo”): Genera un archivo html con una tabla en donde se encuentren los registros leídos y con el título como parámetro.

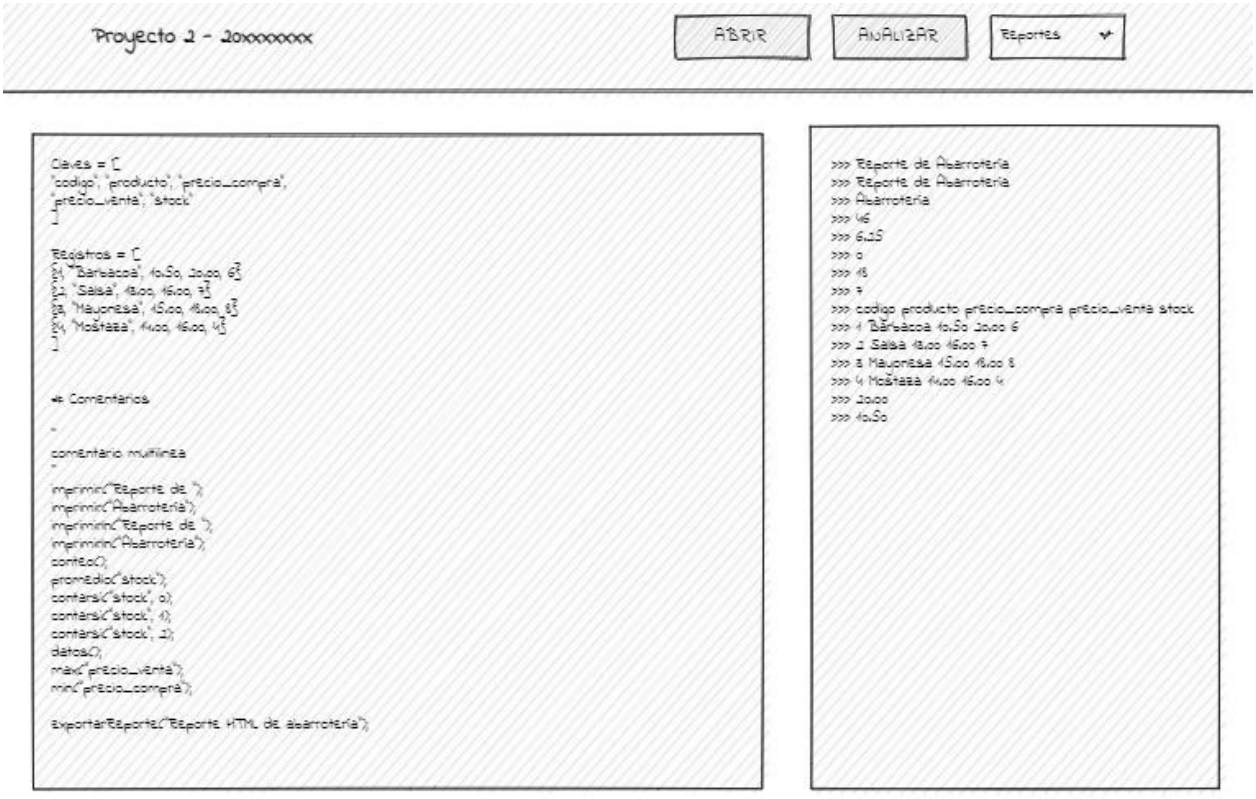
```
exportarReporte("Reporte HTML de abarrotería");
```

Reporte HTML de abarrotería				
código	producto	precio_compra	precio_venta	stock
1	Barbacoa	10.50	20.00	6
2	Salsa	13.00	16.00	7
3	Mayonesa	15.00	18.00	8
4	Mostaza	14.00	16.00	4

Componentes de interfaz gráfica.

La aplicación cuenta con una interfaz gráfica que posee las siguientes características:

- Cargar archivo: Un botón que al presionarlo permita cargar el archivo con extensión “bizdata”.
- Área de texto: Debe tener un área donde se pueda visualizar y modificar el código “bizdata”.
- Analizar archivo: Un botón que analice el código “bizdata”.
- Consola: Un área de texto que no se pueda editar, solamente visualizar texto generado por las instrucciones dadas por el lenguaje.
- Menú Reportes: Un menú que pueda generar los siguientes reportes:
    - Reporte de Tokens
    - Reporte de Errores
    - Árbol de derivación



Reportes

Se deben generar en formato “html” los siguientes reportes.

1. Reporte de errores: Se debe generar una tabla con todos los errores léxicos y sintácticos que se encontraron, indicando el caracter o token leído, fila y columna.
2. Reporte de tokens: Se debe generar una tabla con todos los tokens analizados indicando el tipo de token, lexema, fila y columna del token leído.
3. Árbol de derivación generado en la lectura del código fuente utilizando Graphviz.

Entregables

- Manual de usuario.
- Manual técnico: Debe incluir expresiones regulares, método del árbol y AFD para el analizador léxico, así como gramática independiente del contexto utilizada en el analizador sintáctico.
- Código fuente.

Consideraciones importantes

- La práctica se debe de desarrollar individualmente.
- Utilizar el lenguaje de programación Python y las librerías Tkinter para la parte visual y gráficos con Graphviz. NO se permiten el uso de otras librerías.
- El analizador léxico y el analizador sintáctico lo debe realizar el estudiante, debe coincidir con la documentación del AFD y de la gramática tipo 2, respectivamente.
- La entrega se realizará en la plataforma UEDI. Se debe de entregar en un archivo “txt” el enlace de su repositorio que contenga los entregables solicitados anteriormente.
- Agregar al auxiliar correspondiente a su repositorio, si no se tendrá una penalización.
- La calificación se realizará en línea y se grabará, esto para que quede constancia de la forma en que se calificó y como soporte en la toma de decisiones en reclamos por parte del alumno si se presenta el caso y se calificará desde lo entregado; asegurándose de la fecha de finalización esté dentro de la fecha límite de entrega.
- La calificación es personal con una duración máxima de 30 minutos, en el horario posteriormente convenido.
- El estudiante es responsable del horario que elija para calificarse, en caso de no poder presentarse deberá notificar al auxiliar con suficiente anticipación (2 días antes) para ceder su lugar a otro estudiante, en caso contrario el estudiante solo obtendrá el 80% de su nota obtenida.
- COPIA PARCIAL O TOTAL DEL PROYECTO TENDRÁ UNA NOTA DE CERO PUNTOS, Y SE NOTIFICARÁ AL CATEDRÁTICO DEL CURSO Y POSTERIORMENTE SI SE REQUIERE A LA ESCUELA DE SISTEMAS PARA QUE SE APLIQUEN LAS SANCIONES CORRESPONDIENTES.

Fecha de entrega del proyecto: 26 de octubre de 2023 antes de las 23:59 horas