Universidad Nacional

Sede Regional Brunca

Paradigmas de la programación

Documentación Proyecto

Nombre del Lenguaje: TryCode

Estudiantes:

Sergio Villanueva Ríos

Ronald Blanco Navarro

Esteban Altamirano Granados

Profesor:

Carlos Carranza Blanco

II Ciclo, 2022

2022
Año de las Universidades Públicas por los Territorios y las Comunidades

Tabla de contenido

Planteamiento del problema

El trabajo consiste en un proyecto universitario donde se desarrollar un lenguaje de programación propio el cual debe contar al menos con los siguientes datos y estructuras.

- Nombre
- Palabras reservadas
- Estructuras de Control: Instrucciones, Condicionales simples y múltiples, Ciclo For, Ciclo While.
- Funciones: con retorno de valores y funciones vacías
- Operaciones:
    - Aritméticas: suma, resta, multiplicación y división.
    - Lógicas: and, or, not.
- Entrada y Salida de datos: Leer y escribir variables
- Semántica.
    - Como se debe construir (Explicación de cómo funciona su lenguaje, si usa una función principal, identación obligatorio, etc.).
- Tipos de datos:
    - 5 simples (enteros, flotantes, caracteres, booleanos y nulos).
    - 2 compuestos (arreglos, listas, matrices, etc.).

Pantalla de programación: El programa debe tener un menú con las opciones (se detallarán más adelante), un output para errores y un segmento donde se podrá programar. Además, deberá tener dos botones uno para compilar y otro para ejecutar.

Menú de opciones: EL aplicativo deberá poder mostrar las estructuras detalladas en el punto “Estructuras del lenguaje” de una forma clara, entendible, explicada y visualmente agradable para el usuario. Las opciones deberán ser: Palabras reservadas, sintaxis (submenús de: control, funciones, operaciones), semántica, tipos de datos. Cada uno de estos segmentos deberá poder generar código automático en el espacio de escritura del código.

Output: Este segmento deberá poder mostrar los errores de sintaxis detectados y si es el caso la salida del aplicativo al ejecutarlo.

Botón de compilar: Es el responsable de la revisión del código escrito de acuerdo con las tablas y el proceso de compilación de un lenguaje.

Botón de ejecutar: Es el responsable de ejecutar el código y darle semántica al mismo.

Solución Adoptada

Con el fin de desarrollar los requerimientos solicitados en la sección anterior, se empleo el lenguaje de programación Python, el cual es un lenguaje de alto nivel, interpretado, de propósito general y de código abierto.

El lenguaje desarrollado tiene como nombre TryCode, de ahora en adelante en el documento se referiría al proyecto con el nombre del lenguaje “TryCode”. TryCode es un lenguaje de programación sencillo y fácil de utilizar con paradina imperativo, que es desarrollado como proyecto universitario con fines académicos el cual tiene como objetivo enfrentar al estudiantado a los retos derivados del desarrollo de un lenguaje de programación y que estos obtengan conocimientos de aspectos relacionados a los paradigmas de programación.

Así pues, para el desarrollo de este proyecto una vez determinado el lenguaje de programación con el cual se va a trabajar, se debe de seleccionar las posibles librerías o paquetes con los cuales codificar el compilador con el cual se ejecutará el código del lenguaje TryCode.

Dado esto, la construcción de la interfaz grafica será realizada para poder insertar el código y poder visualizar el resultado de este en output. Iniciando con la utilización de la biblioteca grafica Tkinter con la cual se implementa la estructura necesaria para la creación de la interfaz construida con múltiples componentes relacionados entre sí. Dando la posibilidad de interactuar mediante la escritura de código en la sección de código o la inserción de este con el uso del menú de opciones y otras funcionalidades referentes a la interfaz gráfica.

Continuando con la elaboración del lenguaje de programación se ha de definir una estructura de análisis y ejecución del código mediante e uso de dos secciones que corresponden a Sintaxis y Semántica. Las cuales se verán ayudadas mediante el uso de la librería disponible para Python llamada SLY con la cual se definirán las posibles estructuras sobre las cuales ir modelando el funcionamiento del lenguaje.

De tal forma que dichas estructuras con múltiples instrucciones se dividirán dentro de la sección de Sintaxis en: Lexer y Parser, más la sección de Semántica con Interpreter.

Sintaxis:

La sintaxis busca describir el conjunto de cadenas de caracteres validas presente en un lenguaje de programación especificando su estructura gramatical. Para esto se divide esta sección en el Lexer (analizador Léxico) y Parser (Analizador Sintáctico), los cuales se encuentran estrechamente relacionados en el análisis del código TryCode.

El código TryCode se encuentra compuesto de expresiones finalizadas en el carácter ‘;’, sin tomar en cuenta espacios e identación. Cada una de estas expresiones deben de pasar dentro del Lexer y el Parser, para ser parte de la estructura de expresiones.

Lexer:

Una vez definida una expresión al encontrar un ‘;’ se procede a ingresar a la sección del TryCodeLexer donde se empieza el análisis de esta.

Para esto el analizador léxico ayudado por una clase heredada de Sly llamada Lexer procede a separar la expresión en múltiples secciones mediante el uso de tokens previamente definidos dentro de la clase en un diccionario. Dichas palabras serán consideradas como Palabras Reservadas.

Ejemplos de estas Palabras reservadas serian: NAME, NUMBER, STRING, FLOAT, IF, THEN, ELSE, FOR, WHILE, FUN, TO, ARROW, EQEQ , NOEQ, LTEQ, GTEQ, LT, GT, TRUE , FALSE , AND, OR, NOT. Las cuales poseen significado: AND:’AND’, EQQE: ’==’, GT: ‘>’…..

Aunado a estos existen los literales, los cuales consisten en caracteres con significado especial al momento de la creación de expresiones, estos serían: =, +, - , /, *, (,) y ;.

Sin embargo, un lenguaje de programación no se encuentra solamente definido con palabras reservadas y literales. Para esto se debe de definir reglas sobre el análisis de conjunto de caracteres diferentes que pueden llegar a convertirse en múltiples valores, variables, saltos de línea, etc.

Por tanto, para su detección se da uso de expresiones regulares de Python, conocidas como RegEx, con los cuales se realiza una acción especifica en la construcción de la expresión. Ejemplo del análisis de una expresión seria: a=2;

Token(type='NAME', value='a', lineno=1, index=0, end=1)

Token(type='=', value='=', lineno=1, index=1, end=2)

Token(type='NUMBER', value=2, lineno=1, index=2, end=3)

Parser:

Seguidamente aparece la clase TryCodeParser la cual obtiene los tokens de la clase TryCodeLexer y con estos crea diferentes estructuras que van desde expresiones, condicionales, definición de variables, tipos de datos y más.

Para esto se inicia captando los tokens y seleccionando un posible caso de uso que coincida con su estructura, ayudado por la clase heredada de SLY conocida como Parser. El objetivo de esto es la creación de un árbol de expresiones interrelacionadas jerárquicamente que puede o no aceptar las expresiones entrantes.

De tal forma que la expresión es captada por declaraciones que ejercen función como nodos de un árbol con su declaración en el nodo[0], más su contenido en los hijos del en el nodo[1]:expresión y nodo[2]:expresión.

De dicha estructura en forma de árbol compuesto de nodos se pueden emplear múltiples subestructuras con funcionalidades variadas. Puesto que las expresiones se relacionan entre sí mediante el uso de decoradores de Python: @ y la recursividad.

Un ejemplo de esto es: a=2, expresión que consiste en una asignación de variable (cabe destacar que TryCode es de tipado inferido y dinámico) que coincidiría con: @_“var_assing”. Esta declaración se retorna a si misma, relaciona con otra declaración.

La declaración relacionada seria @_('NAME "=" expr') donde se retorna ("var_assign", p.NAME, p.expr), el cual a su vez al dividirse en tres secciones mantienendo la primera intacta, la segunda se relaciona con @_("NAME") que a su vez retornan su valor como una cadena de caracteres que funciona como nombre de variable, y la tercer sección @_("NUMBER") que especifica el tipo de dato junto con su valor.

El resultado final de efectuar el análisis del Parser es un árbol que debe de ser interpretado y ejecutado por la siguiente sección la cual seria el Interpreter mediante el uso de la clase TryCodeExecute.

Ejemplo básico del árbol resultante de a=2:

```
('var_assign', 'a', ('num', 2))
```

Semántica:

Interpreter:

El lenguaje TryCode es un lenguaje altamente inferido y dinámico, donde las construcciones gramaticales explicadas en la sección de sintaxis pasan a la clase interprete, que es donde se realiza el análisis del conjunto de reglas que dan el significado de dichas construcciones.

Para dar una mejor explicación del análisis semántico que se realiza en la clase “Interpreter” se mostrará con un ejemplo sencillo, donde se asigna el valor a una variable, para este ejemplo se omitirán la explicación del “parser” y “lexer”, el ejemplo se enfoca únicamente en el “interpreter”:

a=2;

Se partirá de los valores que recibe la clase “interpreter”, definidos en la clase “parser” al momento de querer asignar el valor a la variable “a”. Dichos valores corresponden a un árbol de expresiones el cual será recorrido para compilar el código TryCode.

El resultado de dicha acción se cargará en la variable llamada “result” que es la que devolverá el resultado en el output, se encargará del recorrido de la función “walkTree”, esta función es recursiva, que es donde se realizara la lectura de los nodos y con base a su contenido realiza la interpretación para posteriormente realizar la asignación.

El recorrido inicia haciendo la comparativa del contenido del nodo raíz, buscando que token o literal posee, para el caso de la asignación de la variable “a”, el nodo raíz(node[0]) contiene “var_asign” que corresponde a la acción que se realizara con la variable, una vez detectada la asignación, la función se vuelve a llamar pero ahora realizando la búsqueda el tipo de valor que se va a asignar a la variable, para se hace la comparativa del nodo con el valor “num”.

Finalizando con el cargado de la variable a que posee el valor 2, dentro de la memoria de la computadora mediante el uso del diccionario de entorno env.

Ejemplos Sintaxis:

Comentarios:

- #a=5;

Asignación de variables

- a=2;
- b =” Hola Mundo”;
- c=TRUE;

Mostar contenido variables:

- a;
- PRINT(a);

Mostrar Cadena de Caracteres:

- PRINT(“hola”);

Operaciones Aritméticas:

- 2+2;
- 3-3;
- 4/4;
- 5*5;

Comparadores:

- a==3;
- b!=4;
- c<=3;
- v>=4;
- k<9;
- k>6;

Comparadores lógicos:

- a==3 AND b==4;
- a==3 OR b==4;
- NOT a==3 AND b==4;

Sintaxis IF:

Ejemplo 1

```
a=1;
IF a==1 THEN
a
ELSE
a=a+2;
```

Ejemplo 2

```
a=2;
b=2;
IF a==2 AND b==2 THEN
3
ELSE
4;
```

Sintaxis FOR:

```
FOR a=0 TO 5 THEN a;
```

Sintaxis WHILE:

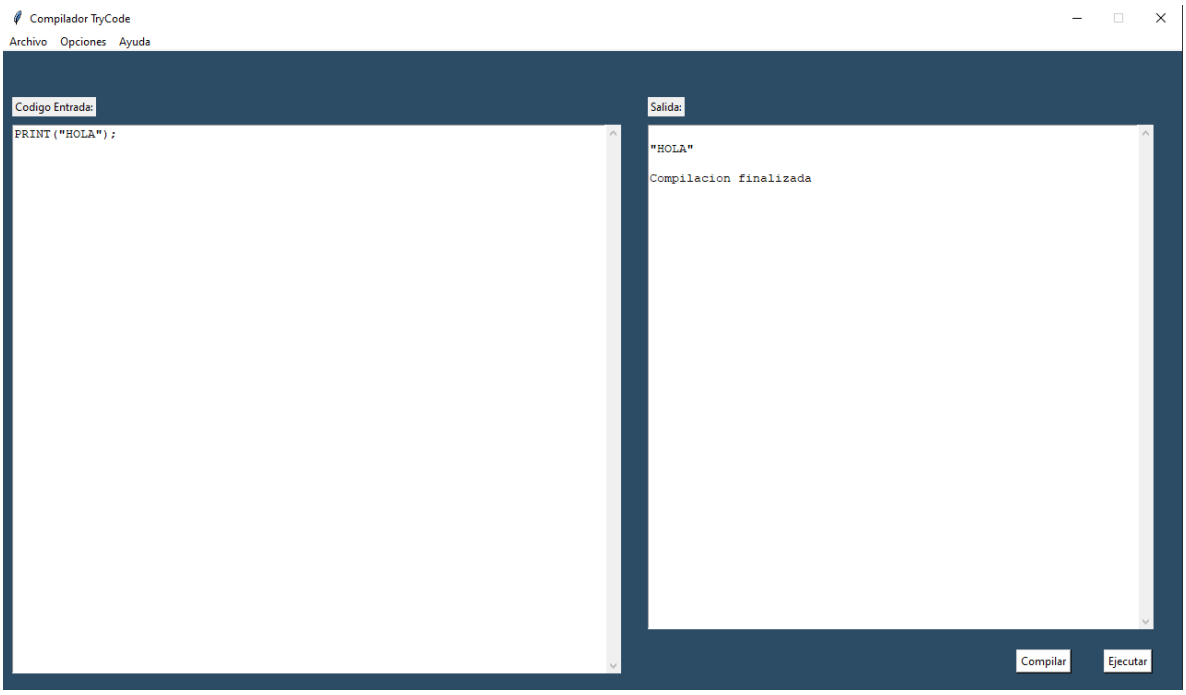
```
a=1;
WHILE a<3 THEN
a
a=a+1;
```

Sintaxis Función:

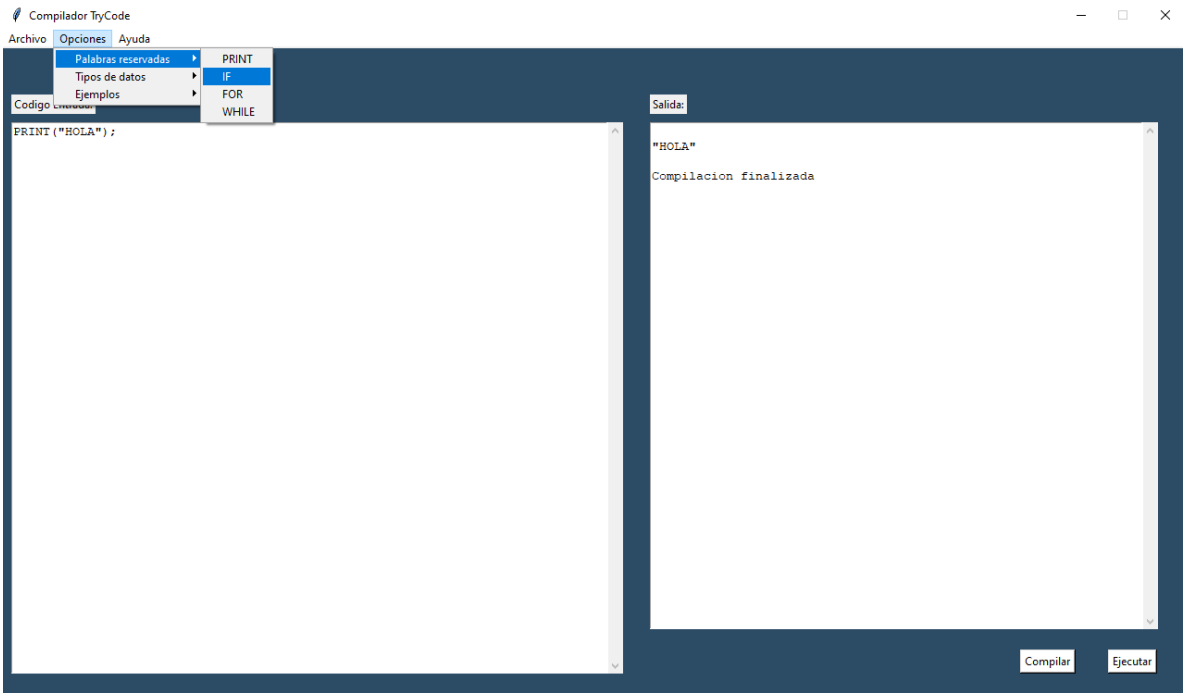
- FUN prueba()-> PRINT("HOLA");
- prueba();

Pantalla de Programación

En la siguiente imagen se muestra la pantalla principal donde se encuentra la sección para ingresar le código, y la sección donde se mostrará la salida al compilar o ejecutar.



En la siguiente imagen se muestra el menú de opciones desplegables



Referencias Bibliograficas:

SLY (Sly Lex Yacc) — sly 0.0 documentation. (s. f.). https://sly.readthedocs.io/en/latest/sly.html