Logotipo

Descripción generada automáticamente

Centro de Ciencias Básicas

Ciencias de la Computación.

Autónomas II.

intérprete

**Ingeniería en Computación Inteligente semestre 7°A**

**Profesor: Francisco Javier Ornelas Zapata**

Alumnos:

J.Horacio Cervantes P.

Descripción general de cómo es que funciona, primero lo que se tiene que ejecutar es el Shell, se escribe cualquier cosa y esta se envía a la función run del archivo inter, primero se crea un lexer para realizar la tokenizacion, si no hubieron errores se continua con el parser que realiza un árbol dependiendo de los tokens devueltos por el lexer, si existe un error se retorna el error, si finalmente se crea el contexto global y el intérprete que analizara los árboles y devolverá algún resultado o errores.

El archivo shell, funciona básicamente como nuestra línea de comando, donde podemos sentenciar variables, algunas sentencias como if,for, while,etc, lo que se realiza es que almacena el texto del código y este se envía al otro archivo el cual realiza la tokenizacion, el analizador de los tokens, y finalmente la ejecución de los mismos

|  |  |
| --- | --- |
| Tokens | significado |
| INT | numero entero |
| FLOAT | numero decimal |
| IDENTIFICADOR | variable |
| MAS | símbolo mas |
| MENOS | símbolo menos |
| MUL | símbolo multiplicar |
| DIV | símbolo dividir (/) |
| POT | símbolo potencia ^ |
| LPAREN | paréntesis izquierdo |
| RPAREN | paréntesis derecho |
| LCORCH | corchete izquierdo |
| RCORCH | corchete derecho |
| IG | igual |
| E | equivalente (==) |
| NE | diferente (!=) |
| MEN | menor |
| MAY | mayor |
| MENE | menor igual |
| MEYE | mayor igual |
| COMA | coma (,) |
| FLC | símbolo flecha -> |
| CADENA | una cadena de caracteres |

|  |  |
| --- | --- |
| Palabras clave | significado |
| VAR | instancia para declarar variables |
| and | operación lógica and |
| or | operación lógica or |
| NOT | operación lógica not |
| if | if clásico |
| else if | else if clásico |
| then | establece que es lo que se realizara después de sentenciar algo |
| for | instancia para el for clásico |
| to | indica el valor a alcanzar en el for |
| step | indica de cuanto avanza el for |
| mientras | while clásico, instancia para declarar whiles |
| fun | declarado de funciones |
| else | Instancia clásica else |

Classe Token:

Empecemos con la clase token, lo cual tiene como atributos tipo en donde se establece el tipo de token que es, valor donde se almacena el valor, posición de inicio donde se guarda su posición, posición de fin donde se guarda la posición donde se termina su sentencia, existe un if en el constructor y es por si existe el valor de posición inicio pasado como parámetro, donde realizara una copia la posición inicial pasada como parámetro y donde posición fin será al principio una copia de la posición inicial pero después se realiza un avance y otro if por si existe el valor de posición final el cual será igual a esa posición final pasada como parámetro

Existe una función de representación (en python se define como def \_\_repr\_\_, esto quiere decir que, al momento de imprimir esta clase, se mostrara en la forma que se establezca en la función), en el caso de token, si es que existe un valor almacenado, se imprimirá el tipo de token y de forma continua el valor almacenado, en dado caso que no tenga, solo se mostrara el tipo de token.

La función de partido existe para ver si el valor y el tipo de token son los mismos.

Dentro de esta área de token, existe una serie de constantes las cuales tiene la forma en la que nos referimos a cada tipo de token, por mencionar algunos TT\_INT tiene almacenado el valor INT que es la forma en la cual nos referimos a los números enteros, se tienen todos los símbolos operacionales básicos (+,-,\*,/,^), para la realización de operaciones básicas, también se cuentan con otros símbolos como <,>,<=,>=,=,==, los paréntesis izquierdo y derecho, además cuenta con un token de final de archivo que este se agrega al terminar de leer cada línea donde se establece el final de la línea y finalmente estarían las palabras claves las cuales son una lista de palabras reservadas para ciertas cosas por ejemplo for, then, mientras (esta sería para empezar un ciclo while), operaciones lógicas como and y or, y la palabra VAR para la declaración de variables, al final de la sección se mostrara la lista de palabras reservadas.

Clase Lexer

Tiene como atributos el nombre del archivo, el cual almacenara el nombre del archivo que este leyendo, aunque por el momento no pueda leer archivos, texto el cual almacena una línea del archivo para su análisis, pos el cual guarda un elemento de la clase posición, y carácter actual que almacena el carácter en el cual se está apuntado la posición.

Se cuenta con una función llamada avanzar la cual le permite avanzar de carácter, lo que realiza es que se llama una función del mismo nombre, pero de posición, donde se le pasa el carácter actual, a su vez el carácter actual cambia al siguiente carácter solo si es que existe ese carácter (si es que no se sale de la cantidad de caracteres que tiene la línea).

La función de tokenizar es la que realiza la tokenizacion , cuenta con una lista la cual servirá para almacenar los tokens, contiene un while que revisara cada carácter con la finalidad de determinar su tipo de token, los casos sencillos son los símbolos como +, en esos casos simplemente se crea un token, con el tipo de etiqueta correspondiendo al símbolo, también se le pasa la posición de inicio, aunque existen las excepciones de los símbolos combinados (==,<=,>=) que se hablaran un poco más adelante.

En el caso de los números existe una función para detectar de que tipo de número se trata y que este tenga el token correcto, aquí entra lo que está definido como el campo constante donde nos referimos a la cadena de dígitos la cual es una cadena de números, verificamos que el carácter este en esa cadena, ahí es cuando se llama a la función de hacer números, en esta función contamos con una cadena vacía para almacenar los caracteres, además se cuenta con una bandera para verificar que no ocurran errores en la escritura del número decimal y finalmente guardamos la posición inicial del carácter que inicio el llamado de la función

Entra a un while que realiza la acción mientras que el carácter actual no sea un vacío (None) y este entre en la cadena de dígitos o tenga un punto, si encuentra un punto se hace la pregunta de cuál es valor de la bandera (esto refiriéndose a cuantas veces ha encontrado el punto si lo encontró más de una vez romperá el while provocando que no retorne nada ), si es la primera vez que lo encuentra le sumara uno a la bandera y añadirá el punto a la cadena de almacenamiento, en dado caso que no haya un punto solo se almacenara el carácter en la cadena de almacenamiento, evidentemente después de preguntarse esto se cambia de carácter

Finalmente se pregunta cuanto fue el valor de la bandera si fue cero se retornara un token de tipo int , con el valor convertido a un entero y la posición de inicio del token y la posición de inicio del carácter actual (del último carácter visto), en el caso contrario sería un token de tipo float con su respectivo valor y lo demás mencionado anteriormente.

En el caso que durante el while no se encuentre con ningún símbolo o entre en alguna cadena el leer retornara una lista vacía y un error de carácter ilegal, donde se hace mención del carácter que no está bien establecido.

Al final de este while se agregará un token especial a la lista junto a la posición que determina el final de esa línea.

La función hacer identificadores realiza lo siguiente, cuenta con dos variables una para almacenar la cadena y otra para establecer la posición inicio de la secuencia, la cual será una copia de la posición, mientras el carácter actual no sea un vacío y este esté dentro de la constante letras\_digitos junto a \_ se almacena cada carácter en la variable id y se avanza en la posición, después se será una variable para almacenar el tipo de token que tendrá la cadena, si la cadena id entra dentro de las palabras clave, se le asignara el token de palabaraclave, si ese no es el caso se le asignara el token de identificador (prácticamente una variable) retorna un token con el tipo de token, la cadena id, la posición de inicio y una copia de la posición del último token.

La función hacer no equivalente (¡=) como sabemos que recibimos ¡ podemos avanzar en el siguiente carácter, a la par guardamos la posición de inicio del token siguiente , entonces si el siguiente token es un igual, podemos retornar un token con la etiqueta de no equivalentes , junto a las posiciones y sin un mensaje de error, pero si no nos encontramos con ese igual enviamos nada de resultado y un error el cual es de carácter esperado con las determinadas posiciones junto al detalle que se ocupaba poner un igual después de ¡

La función hace equivalente (==) primero asignamos el valor de tipo de token a solamente el token de igual (no confundirlo con el de equivalente podríamos afirmar que este token es = mientras que el de equivalente es ==) guardamos la posición del primer token, después avanzamos en los tokens, si el siguiente carácter es un = se cambia el valor de este tipo de token al token equivalente, finalmente se retorna el token con el dicho tipo de token, con las posición de inicio siendo la copia del primer token, la posición del último carácter visto.

La función hacer menor (< o <=) estas son prácticamente idéntica a la función hace equivalente pero aquí primero se iguala al token menor y si se encuentra el carácter =, se cambia al token mene (menor o igual)

La función hacer mayor (> o >=) es lo mismo solo que el token inicial es el token may y después si se encuentra con el carácter = cambia al token maye.

La función hacer flecha o menos nos ayuda a saber si se creó una función, primero se establece que el tipo de token es menos (refiriéndonos a una resta), se guarda la posición actual del token menos, se avanza de carácter si el siguiente carácter es una > se cambia el tipo de token a FLC que representa la flecha, se avanza de carácter y finalmente se retorna un tipo token con tipo token, la posición copiada y la posición del último carácter como parámetros.

La función de realizar las cadenas, se tiene una variable para almacenar toda la cadena introducida, se guarda en una variable la posición inicio de la cadena (el primer símbolo “), después se crea una bandera para saber si se encontró con algun carácter de salto por ejemplo \n aunque solo se tiene el de salto de línea y tabulador, también se avanza de carácter

Existe una tupla que tiene estos elementos, aquí se establece que tipo de carácter será el que remplazara a estos caracteres con función especial, mientras no se encuentre con un tipo none o las otras comillas se seguirá añadiendo a la cadena, sin importar que se encuentre con un \n o tabulación , si la bandera es verdadera remplazamos estos caracteres \n con unos propios del lenguaje, solo si estos están definidos en la tupla y se añaden a la cadena, la bandera solo se prendera si es que encuentra algún \ si no encuentra simplemente sigue añadiendo caracteres a la cadena

Al finalizar el ciclo se avanza de carácter y se retorna el token siendo su tipo cadena, su posición inicial y la posición final del token actual

Area Posición

Clase posición

Tiene como atributos el index el cual será una forma de saber dónde está, línea lo cual permite saber en qué línea se encuentra, col lo cual permite saber en qué columna se encuentra, nombre del archivo lo que almacena es el nombre del archivo y el texto del archivo (su contenido)

Su función de avanzar lo que realiza es que aumenta el index en uno y también la columna, como tiene el parámetro de carácter actual si este es un salto de línea se aumenta la posición de la línea y columna se vuelve cero, pero siempre se vuelve a retornar a sí mismo.

Su función de copiar realiza una copia de este y esta es retornada

Área Error

Clase error

Esta funciona como la clase principal y las demás son subclases referentes a esta, como parámetros tiene el nombre del error, los detalles referentes a los mismos y de igual forma tienen la posición inicial y la posición final del objeto del error.

Tiene una función llamada as\_string básicamente realiza la misma acción que la función de representación, solo se establece la forma en la cual el error será mostrado al ser impreso en la pantalla.

Clase error carácter ilegal cuenta con tres atributos heredados siendo estos las posiciones y los detalles, después se reúsa la clase de error, pero aquí se le envía el nombre del error de carácter ilegal.

Clase error carácter esperado es prácticamente igual que los otros errores solo que ahora el nombre será error de carácter esperado solo se da en el lexer (el tokenizador)

Clase error de sintaxis cuenta con los mismos atributos heredados y también se reúsa el constructor de error, el nombre será sintaxis invalida y solo ocurrirá en el proceso del parser.

Clase error de ejecución cuenta con los mismos atributos heredados y también se reúsa el constructor de error, el nombre será error en ejecución, además también recibe un contexto y solo ocurrirá en el proceso del intreprete.

Se sobre escribe la función de as\_string dado que ahora tenemos el contexto, es similar al anterior solo que ahora llamaos al rastreo del error (la función de trace back).

La función de generación del trace back, empieza con una variable almacenado la posición de inicio del mismo, además de otra que almacena su contexto, mientras este contexto exista, se le agregara el nombre del archivo, la línea y el nombre del contexto, se moverá la posición a la posición del padre y el contexto ahora será el contexto padre

Al finalizar se retornará la cadena generada

NODOS

Clase nodo numero

La clase tiene s atributo donde se almacenará el número ya sea de tipo decimal o entero, igual cuneta con las posiciones, donde su posición inicial será igual a la posición de inicio del token también cuenta posición final que almacena la posición final del token (el valor), de igual forma cuenta con una función de representación lo que se mostraría al llamar a esta clase seria solamente el token que contenga.

La clase nodo cadena básicamente es igual que al de numero solo que servirá para cadenas.

Clase nodo operacional (se llama asi porque este será la representación de por ejemplo 3 + 2, asi dicho este formará las operaciones).

Contiene los siguientes atributos un nodo izquierdo, donde se guardara el nodo del lado izquierdo, un token operacional el cual guardara el token de algún símbolo de operación y el nodo derecho que almacenara el nodo del lado derecho, de igual forma también se cuenta posición inicio y posición fin, posición inicio tiene la posición de inicio del nodo izquierdo mientras que posición fin tiene la posición fin del nodo derecho, al mismo tiempo también cuenta con una función de representación donde lo que imprimirá será el contenido del nodo izquierdo, el token de operación y el contenido del token derecho.

Clase nodo p (se llama asi porque sería la representación de lo que este fuera de un paréntesi 3 +) tiene los atributos de token operación que guardara la operación y nodo que guardara el nodo número alado de esta operación, también cuenta con posición inicio y posición fin, la posición inicio guarda la posición de inicio del token operacional, mientras que posición fin guarda la posición final del nodo.

Cuenta con una función de representación para imprimir el valor del token operacional y el valor del nodo número.

Clase NodoADVar tiene tres atributos el nombre de la variable del token, la posición inicio y la posición fin, la posición inicio es la posición inicio del token pasado, la posición fin es la posición fin del mismo token, esta clase nos permitirá acceder a la variable

Clase NodoDVar tiene cuatro atributos las mismas posiciones, donde su posición fin será la posición fin del valor del nodo, su posición inicio será la misma posición inicio del nombre de la variable del token, otra será el nombre de la variable del token y el valor del nodo, esta clase nos permitirá cambiar el valor.

Clase Nodoif tiene como atributos la tupla de casos, el caso de else y sus respectivas posiciones, la posición inicio se iguala al primer caso de la tupla y su respectiva posición de inicio, la posición fin si existe algún caos en caso else, se le asigna esa posición, pero en dado caso que no pues se asigna el caso de la tupla que sea la última, de cualquiera de los dos objetos sera su respectiva posición final.

Clase Nodofor tiene los siguientes parámetros y atributos un nombre de la variable, la que estará dentro de la sentencia del for, el valor de inicio de esa variable, el valor final que es le valor que se trata de alcanzar, el valor del paso que sería el posible valor que se establezca para que se incremente la variable y final mente el cuerpo, que seria la expresión que vendría después de then, también cuenta con las posiciones, su posición inicio será igual a la posición inicio del token pasado como nombre de la variable y su posición fin seria la posición fin del cuerpo.

Clase Nodowhile tiene los siguientes atributos la condición que será alguna expresión, el cuerpo que también será alguna expresión, de igual forma cuenta con las posiciones, su posición inicio será la posición inicio de la condición, su posición fin será la posición fin del cuerpo.

Clase Nodofun tiene los siguientes atributos nombre de la función, que será el token que contenderá el nombre, nombre de los argumentos, serán los nombres de los tokens que servirán como parámetros de la función, cuerpo que contendrá las sentencia de lo que realizará la función, la posición inicio del nodo será el token nombre si es que este existe, sino será el primer paramento de la función si ninguna de las cosas anteriores existe será el cuerpo de la función.

Clase Nodollamada servirá para poder llamar a la función, tiene los siguientes parámetros nodo con la función a llamar, los posibles nodos de los argumentos (parámetros) y las posiciones la posición de inicio será la posición inicio de la función a llamar si existen nodos de los parámetros la posición final será la posición final del último argumento si no existen argumentos será la posición final del nodo con la función a llamar.

Clase NodoLista servirá para crear listas tiene como parámetros los elementos de la lista, la posición de inicio y la posición fin vienen como parámetros a diferencia de los otros nodos, esto es porque puede que no tenga nada la lista asi que esto nos servirá para saber en donde se sentenció la lista

AREA DE PARSER

Gramática

La gramática

3 + 2 \* 3 todo esto es una expresión o se podría considerar como toda la línea

Las partes separadas por la suma es un (3) + (2\*3) es un término y los números en sí mismos son un factor. Con esto en mente se podrá entender un poco mejor las reglas gramaticales

Entonces una expresión consta de la palabra clave VAR después un identificador un símbolo de igualdad (=) y una expresión como si no se cuenta con eso se procede a buscar los siguientes elementos:

Busca una expresión de comparación, debe tener ya sea ninguna o más estructuras de las siguiente (las palabras clave or o and y otra expresiones de comparación)

Expresión se comparación consta de una la palabra clave NOT y de una expresión comparativa si no encuentra esto busca lo siguiente:

expresión aritmética y ya sea ninguna o más estructuras (de cualquier token compa racional (<,>,!=,etc) y otra expresión aritmética)

Expresión aritmética consta de un término y pude tener o no, ya sea una un token más o menos y otro termino

Un término es factor, que puede además tener ya sea una vez o más un símbolo de división o multiplicación además de otro factor.

Un factor tendrá un símbolo de menos o mas además de otro factor, en dado caso que esto no suceda se llama a la regla de power

Power consta de dos cosas una llamada y puede que tenga un símbolo de potencia y otro factor o no tenerlo

Llamada consta de un átomo y opcionalmente puede tener paréntesis izquierdo puede que dentro de los paréntesis puede que tenga una expresión, más de una coma y después otra expresión (coma expresión coma expresión algo asi) y finalmente un paréntesis derecho

Un átomo puede ser un numero entero, flotante, una variable o una cadena, en dado caso que no tenga esos elementos

Se busca que tenga un paréntesis izquierdo una expresión y un paréntesis derecho.

Si no se llama a la definición de lista.

Si no se llama la expresión if.

Si no se llama a la expresión for

Si no se llama a la expresión while.

Si no se llama a la definición de función

La definición de lista tiene un corchete izquierdo. puede que tenga una expresión, más de una coma y después otra expresión (coma expresión coma expresión algo asi) y termina con un corchete derecho.

Expresión if consiste en una palabra clave if después por una expresión después por then y una expresión además pueden tener ninguna o más estructuras else if (su palabra clave else if su condición su palabra clave then su expresión) y posiblemte una palabra clave else y su expresión.

Expresión for consiste en la palabra clave for, después un identificador un símbolo de igual una expresión la palabra clave to después una expresión, posiblemente la palabra step junto a una expresión, la palabra clave then y finalmente una expresión.

Expresión while consiste en la palabra clave mientras, una expresión después la palabra clave then y finalmente una expresión.

Definición de función consiste en la palabra clave fun un identificador opcional (esto representa el nombre de la función), un paréntesis izquierdo atributos opcionales que para estos será un identificador cero o más de las siguientes cosas (coma y otro identificador) un paréntesis derecho después una flecha y finalmente una expresión que sería lo que se va realizar al llamar a la función

expr : Palabra clave:VAR Identificador IG expr

:comp-expr ((palabraclave:and|palabra calve:or) comp-expr)\*

comp-expr : NOT comp-expr

: arit-expr ((E||GT|LTE|GTE) arith-expr)\*

arith-expr :term ((mas|menos) term)\*

term : factor ((Mul|DIV) factor)\*

factor : (mas|menos) factor

: power

power : call (Potencia factor)\*

llamada : atom (LPAREN (expr (COMA expr)\*)? RPAREN)?

átomo : INT|FLOAT|CADENA|IDENTIFICADOR

: LPAREN expr RPAREN

: list-expr

: if-expr

: for-expr

: while-expr

: func-def

list-expr : LCORH (expr (COMA expr)\*)? RCORH

if-expr : Palabra calve:if expr Palabra clave:then expr

(PALABRA CLAVE:else if expr CPALABRA CLAVE:then expr)\*

(PALABRA CLAVE:else expr)?

for-expr : PALABRACLAVE:for IDENTIFICADOR IG expr PALABRACLAVE:to expr

(PALABRACLAVE:step expr)? Palabracalve:then expr

while-expr : PALABRACLAVE:mientras expr PALABRACLAVE:then expr

func-def : PALABRACLAVE:fun IDENTIFICADOR?

LPAREN (IDENTIFICADOR (COMA IDENTIFICADOR) \*)? RPAREN

FLECHA expr

La clase parser

Tiene los siguientes atributos, tiene una lista de tokens para almacenar los tokens(se llama tokens) recibidos del lexer, a su vez tiene una variable llamada tok\_idx el cual almacenera la posición en la lista de tokens y token actual el cual almacena el token que se revisa en ese momento.

también se cuenta con una función de avanzar donde se incrementa el índice de tok\_idx y se pregunta si es que el índice está dentro del tamaño de la lista, en dado caso que, si token actual toma el valor de dicha posición de la lista, siempre se retorna el token actual.

La función parse genera una variable res la cual es un objeto parser realizando la función expr después, si es que no hay un error y aun no se alcanzado el final de la línea significa que hay tokens que no pasaron por el parecer por lo cual eso significa que ocurrió un error de sintaxis asi que si esto pasa se retorna un error de sintaxis donde se menciona que falto un operando

La funcion op bin tiene como parámetros una función a, los operandos que serían los tokens de operación y una función b, la cual será nada si es que no la pasan, si esta no es nada será la misma función a.

La variable left tendrá el objeto de parserresultado haciendo registro del resultado de la función a que pueden ser otras funciones de parser si existe un error en res este se retorna, después entra a un ciclo while se revisa que el token actual entre en alguno de los tokens de operación mandados o que el valor y el tipo si entren en estos, si ese es el caso la variable token op guardara el token actual, se realizara un avance en res (llamando su función res.registro\_avance) se llama a la función avanzar de parser (dado que trabajamos con un self), rigth guardara al objeto res haciendo registro del valor retornado de la función b si existe un error en res, se retorna res, left será ahora una nodo op donde se almacenara su valor anterior, el token op y la variable right

Al final se retornara el resultado de res.exito con parámetro left, básicamente funciona para buscar algún elemento gramatical izquierdo sus símbolos tokens de unión y un elemento gramatical derecho.

La función de expresión, tiene la variable res que es un objeto de tipo parser resultado primero se busca saber si el token actual es del tipo palabra clave y si es que es igual a VAR, si es el caso realiza la función de registro avanzar de res y se avanza en nodo, después si el token actual no es igual a un identificador se envía un error de sintaxis estableciendo que se buscaba un identificador, si no es el caso el token actual (nombre de la variable) se guarda en una variable, se vuelve a llamar registro avance y también se avanza de token, se verifica que el token actual sea un = si no es el coso regresa un error de sintaxis con los parámetros de la posición inicio del token actual y la posición final del token actual, además se menciona que faltaba un = , sino se vuelve a avanzar en res y en el token actual y una variable vuelve a llamar a la función de expresión dentro del registro de res, si res tiene un error se retornar res, sino pues se retorna el valor de la función éxito con parámetros el nodoDvar con parámetros nombre var y la variable que llamao otra vez a expresión (expr)

Si nuca se encuentra con var lo que realiza es que crea una variable llamada nodo la cual realiza un registro de res realizando una operación binaria con los operandos lógicos.

Si existe un error en res se retorna un error de sintaxis, mencionado que le falta algo, si no es asi retorna el resultado de res éxito con nodo como parámetro

La función de termino hace referencia a la estructura gramatical del mismo nombre donde se retorna el resultado de la función de op bin, teniendo como parámetros la función de un factor y los tokens de tipo MUL y DIV haciendo referencia a las operaciones de multiplicar y dividir

La función factor hace referencia a la regla que se describió anterior mente, tiene dos variables, tok actual que almacena el token del nodo y res que es un objeto parser resultado, si el token actual es un símbolo de suma o de resta, se llama la función de res registro avance y después se avanza en la lista de los tokens, una variable llamada factor se iguala al resultado de la función registro de res, con la función de expresión como parámetro, si existe un error en res, se retorna res, si no pues e retorna el resultado de la función existo de res con un nodo p con token actual y el factor

La función power retorna el resultado de la función op bin con los parámetros siguientes parámetros la función átomo, el símbolo de potencia y la función factor

La función de llamada contiene su res, después crea una variable átomo que guarda el registro de llamar a la función átomo, si res tiene un error se retorna res, después se verifica que el siguiente token sea un paréntesis izquierdo, si no tiene el paréntesis izquierdo retornara la variable átomo.

En el caso de que si lo tenga, se realiza un avance en res y en los tokens, además se crea una variable para almacenar los argumentos llamada nodos args, si el siguiente token es un paréntesis derecho solo se avanza en res y de token, en el caso contrario se añade el registro de llamar a la función expresión, si hay un error en res se retorna un error de sintaxis, se abre un ciclo while mientras que el token actual sea un token coma, se avanza en res y en los tokens, después se añade a nodos args el registro de llamar a la función expresión, si existe un error en res se retorna error, una vez terminado el ciclo, se verifica que el siguiente token sea un parentecis derecho si no es asi se retorna un error de sintaxis, se vuelve a avanzar en res y en token, finalmente se retorna el registro del nodo llamada con la variable atomo y los nodos args.

La función átomo tiene una variable para almacenar el token actual, y otro que es res siendo un objeto de parserresultado, dependiendo de este token actual podrá retornar un tipo de nodo o llamar a otra función del parser ya sea expresión, expresión if, expresión for, expresión while ,etc.

En cada algunos de los casos se verifican que no exista un error en res, si es que existe este error se retorna solamente res, pero si no es asi se retorna el éxito de llamar a cada función que está llamando.

Si no sucede ningún caso se retornará un error de sintaxis haciendo referencia que faltaba alguna símbolo

La función de expresión comparativa tiene un objeto de tipo parser resultado llamado res, primero busca que el token actual tenga un partido con el token de palabra clave y que este sea NOT, si es el caso se guarda su operación, se hace un avance en res y después en los tokens se crea una variable que se llama nodo y esta llama a otra vez a la función de expresión comparativa si ocurre un error en res se retorna res, si no es asi se crea un nodop con el token operacional y la variable nodo.

Si nunca encuentra NOT se crea una variable nodo que llama al res hacer registro de la función operación binaria con la función de expresión aritmética, los operandos comparativos como parámetros, si ocurre un error en error de sintaxis con el detalle que falto algún componente al finalizar se retorna el éxito de res con la variable nodo como parámetro.

La función de expresión aritmética retorna el valor de realizar la función operación binaria con la función termino y los tokens más o menos como parámetro.

Expresión if, crea su respectivo objeto parser resultado, tiene una lista para los casos y una variable que maneje el posible caso else primeramente si no se encuentra con la palabra clave if, retornara un error de sintaxis, donde se menciona que este elemento no se encuentra, en el caso contrario se avanza en res y también en los tokens, después se crea una variable condición que almacena el registro del resultado de llamar la función expresión , si existe el error en res se retorna res, si no existe el símbolo then se crea un error de sintaxis que establece que esta palabra clave falto, se vuelve hacer un avance en ambas cosas, se crea una variable de expresión la cual guarda el registro de llamar a expresión, si existe un error en res se retorna res, después se almacena a la lista para los casos, la variable de condición con la variable de expresión (siendo asi el primer caso).

Se abre un ciclo while que establece que mientras el token actual tenga un partido con las palabras clave y else if, pues repite el avence de res y de token, se almacena otras vez una condición en la variable condición, si tiene un error se retorna res, si el token no es then retorna un error de sintaxis indicando que falto esta palabra, después e avanza en res y el token, nuevamente la variable expresión almacena el registro del resultado de llamar la función expresión y se vuelven almacenar la condición y la expresión en casos.

Fuera del while se pregunta si es que el token actual tiene un partido con la palabra clave else, si es el caso se avanza en el registro y el token, para después almacenar en la variable caso else el registro de llamar la función de expresión para finalmente retornar el valor de la función éxito de res con un nodo if con la lista de los casos y el caso else como parámetros.

La función expresión for, tiene un objeto parser resultado llamado res, verificamos que el token actual sea la palabra clave for, si no se tiene se retorna un error de sintaxis diciendo que le faltaba la palabra for, en caso contrario avanza en res y de token, después si se encuentra con un token diferente a un identificador pues retorna un error mencionando que falta eso, después se crea una variable llamada var name que almacene el token actual (este sería la variable), se avanza en registro y en token, si el token no es un token de igualdad se retorna un error, mencionado que hace falta un =, se avanza en el registro y el avance, el valor de inicio es una variable que se Iguala a un registro de la función expresión, se verifica que no ocurra un error en res, después si no existe el token to pues se manda un error mencionando esto, se avanza en el registro y en el token, valor final guarda el registro de res con la función expresión, si hay un error en res este se retorna.

Verificados si es que existe un token con step, si es asi se avanza en el registro y el token, la variable caso step guardara el registro de la función de expresión si hay un error en res este se retorna, pero si no se encuentra con el token step, pues caso step vale None, después si no existe el token then pues se retorna un error de sintaxis el cual mención que esta palabra falta, se avanza en el registro y el token, después una variable cuerpo guarda el registro de la función expresión, si hay un error en res se retorna y finalmente se retorna res éxito con un nodofor con las variables mencionadas como parámetros.

La función expresión while tiene res, primero se verifica que se tenga el token mientras si no se tiene retorna un error de sintaxis el cual se dice que falta mientras, después continua avanzando en res, y el token, después se crea una variable condición que almacena el registro de llamar a la función expresión, si existe error se retorna res, después verificamos que el token actual sea un then, si no es el caso se retorna, se avanza en res y token después, se crea una variable llamada cuerpo para que guarde el resultado del registro al realizar la función de expresión, si existe un error en res, se retorna res, finalmente se retorna un nodowhile con las variables mencionadas como parámetros.

La función definición de función, tiene res (paser resultado), verificamos que el token actual sea fun si no es asi se retorna un error de sintaxis diciendo que falta fun, se avanza en res y en los tokens, si es que el token es un identificador, este se almacena en la variable nombre, después se vuelve a avanzar en res y en tokens, si no hay un token paréntesis izquierdo se retorna un error de sintaxis mencionando que eso falta, en dado caso que no exista un identificador se le asigna none y se vuelve a realizar lo anterior mente mencionado, se vuelve a avanzar en tokens y res, se crea una lista que almacenara los tokens de argumento , esta se llama tokens\_args, si se encuentra un identificador después del paréntesis izquierdo, se asume que se están poniendo parámetros en la función asi que se guarda el token en token args, se vuelve a avanzar en res y posición, después se declara un ciclo while el cual tiene como condición que el token actual sea una coma, asi que se avanza en res y token si el siguiente token después de una coma no es un identificador se retorna un error de sintaxis que mencionado que falta este identificador, en caso contrario se almacena en tokens args, al finalizar el ciclo, se verifica que exista el paréntesis derecho sino se retorna un error de sintaxis mencionando que falto el paréntesis o el identificador.

En el caso que no exista un token identificador después del token paréntesis izquierdo, solo verifica que exista el token paréntesis derecho, si no se encuentra retorna un error de sintaxis mencionando esto, se vuelve a avanzar en res y tokens, si no se encuentra el token de flecha marca un error de sintaxis indicando esto, se vuelve a avanzar en res y token, después se crea una variable llamada retorno (representaría lo que retornaría la función), en esta se guarda el registro de res llamando a la función expresión , si existe un error en res se retorna res, finalmente si se retorna un nodo función teniendo como parámetros las variables de nombre, tokens args y retorno.

La función de definición de lista tiene su respectivo res, se crea una lista para los elementos de la lista que será creada, esta se llama elementos, posición inicio es una copia de la posición del token actual, después si el token actual no es un corchete izquierdo se retorna un error de sintaxis indicando esto , después se avanza en el registro y en el token, después se pregunta si el siguiente token es un corchete derecho si es asi solo se avanza en res y token, esto indicaría que está vacía la lista, en caso siguiente se añade a elementos el registro de llamar a la función expresión, si ocurren un error, se retorna este error, se abre un ciclo el cual continua hasta que el tipo actual de token no sea una coma lo que realiza dentro del ciclo es que avanza en res y de token, después vuelve a añadir a elementos el registro de llamar a la función de expresión al terminar el ciclo si no sigue un token de corchete derecho se retorna un error de sintaxis mencionando esto, finalmente se retorna un nodo lista con las variables de elementos, la copia de la posición del primer token y la posición final del token actual(este debería de ser el corchete derecho).

Clase parser resultado (su finalidad es verificar que no existan errores)

Tiene tres parámetros error que es donde se almacenará el error, nodo donde se almacena el nodo (en dado caso que sea un éxito) y el contador de avance el cual básicamente sirve como un contador de pasos.

La función de éxito tiene como parámetro un nodo y lo que realiza es que almacena ese nodo en su atributo de nodo.

La función de fallo tiene como parámetro un error y lo que realiza es que almacena ese error solamente si no existe un error en su atributo error y si no ha avanzado el contador de avance

La función de registro tiene como parámetro res, evaluamos si es que no tiene un error si lo tiene se retorna el objeto res, al finalizar retorna los nodos de res, de igual forma se avanza el contador de avance

La función de registro avance solamente hace que avance el contador de avance.

AREA DE ALMACEN.

Esta área tiene la finalidad de almacenar el valor de los nodos y realizar operaciones con estos mismos.

Clase valor almacenara las posiciones iniciales y finales de dichos nodos, también tiene el contexto.

La función de poner posición, como su nombre menciona establece los parámetros introducidos como sus posición inicial y posición final respectivamente.

La función de poner contexto establece el parámetro recibido como su contexto.

Cuenta con todas las funciones de las clases (las que realizan operaciones y copiado) que se describirán más adelante solo que aquí siempre se denotara un error de ejecución el cual dice que es una operación invalida., en el caso de la función copiar y de la funcion is true, en la primera se marca que no está definida y en la segunda siempre se retornara falso.

La función operación invalida, tiene como parámetro otro, si otro no existe este será igual a self y se retorna un error de ejecución estableciendo que se trata de una operación ilegal.

Nótese que las siguientes clases serán subclases de valor, casi todas las funciones se llaman igual, pero están sobre escritas (realizan otra cosa)

Clase Numero: tiene como parámetros el valor, que almacena el valor del nodo, tiene posición de inicio y posición final las cuales son heredas de Valor

La función de sumar a, tiene como parámetro otro objeto de la clase número, primero se verifica que el objeto enviado sea igual que un objeto número, si ese es caso sus valores son sumados y se retorna un objeto número con el resultado, además del dicho contexto, si se pregunta por el valor None esto representa el error en la operación dado que es una suma no debería de ocurrir (por ejemplo, donde se pudiera ocurrir este error seria en una división entre cero)

Si no se trata de un número se establecerá un error de operación invalida

La función de restar a es prácticamente los mismo que la anterior función solo que se restan los valores

La función de multiplicar a es los mismo solo que se multiplican los valores

La función de dividir a realiza lo mismo que las anteriores solo que aquí divide y además verifica que el parámetro de otro no tenga el valor de cero, si esto ocurre se retorna un error de ejecución donde se establece que el nombre del error es división entre cero, además de las posiciones inicio del objeto otro y la posición fin del objeto otro, además el contexto.

La función de potenciar a realiza lo mismo que las anteriores solo que aquí se eleva el número izquierdo ala número derecha.

Las funciones de obtener los aspectos de equivalencia (==,!=,<,>,<=,>=) y las que realizan las operaciones lógicas or y and son prácticamente iguales a las que realizan operaciones normales, solo que aquí pues se realizan esas dichas operaciones lógicas y retorna el valor como un int (0 falso, 1 verdadero ) y el noad cambia dicho valor lógico dependiendo del valor del pasado en el parámetro.

También tiene una función de copiado que copea todos los elementos del número enviado como parámetro.

La función is true, retorna el valor de evaluar el su propio valor con la operación ¡=0 si su valor es 0 pues retornara falso.

Tiene una función de representación la cual solo muestra el valor del objeto número, además se cuentan con tres constantes null que es cero, falso que también es cero y verdadero que es uno.

La calase cadena alacena los valores de tipo cadena o string, cuenta con un atributo valor a parte de los heredados de valor, cuenta con las funciones de copiar y representación, al imprimirse solo devuelve el valor, en el caso de esta clase su función is true devuelve verdadero si es que la cadena tiene algo dentro.

La función sobre escrita sumar a, sirve para concatenar dos cadenas, primeramente, se verifica que ambos parámetros sean objetos cadena, si es asi se realiza la operación con su dicho contexto, si no es asi retorna una operación ilegal.

La función multiplicar por es proactivamente lo mismo que la anterior solo que aquí se duplica la cadena por la cantidad establecida en otro y otro debe ser un número, también tiene una función solo para imprimir lo que esta dentro de la misma (\_\_srt\_\_)

La clase lista sirve para establecer una lista, como atributo propio tiene elementos, estos representaran los elementos de una lista, en este caso la función sumar añade un elemento a la lista, retornado esta nueva lista, la función restar a elimina el elemento que este en el índice que se pasa , primero se verifica que se haya pasado un numero para poder eliminar ese elemento en ese espacio, después se intenta realizar esto si no se puede se dice que el índice esta fuera de los parámetros de la lista (del tamaño), si índice lo que se pasó por los parámetros no es un número se retorna un error de ejecucucion estableciendo que es un error invalido.

La función multiplicar a concatena las cadenas pasadas, primero se verifica que el parámetro otro sea una lista si es asi se concatenan las cadenas y se retorna la cadena sin ningún error, si el parámetro no es una cadena se retorna un error invalido, la función de dividir a retorna el valor que se encuentra en el índice dado, primero se verifica que el elemento enviado en el parámetro sea un número, en dado caso que no sea asi se retorna una operación invalida después se intenta retornar el valor que este en ese lugar, en dado caso que no se pueda se retorna un error de ejecución que establece que dicho índice estaba fuera del rango de la lista, además también cuenta con su propia versión de copiar que retorna una copia, además tiene una forma de representación en la cual se imprime todos los elementos de la lista separados por comas, aunque también se puede usar str la cual solo imprimirá los valores sin [].

La clase función base es subclase de Valor y es la clase madre de las demás clases funciones, tiene como parámetros el nombre de la función, en dado caso que no exista un nombre pues se llamara anónima.

Tiene una función para establecer el nuevo contexto, se llama contexto nuevo, se establece el contexto de la función donde el nombre del contexto será el nombre de función, el padre del nuevo contexto será el contexto anterior (el contexto que tiene el objeto función el cual debería de ser el programa general en dado caso que no se esté llamando dentro de otra función) la tabla de símbolos se actualizara a la del padre lo cual debería permitir que se usen los parámetros, dentro de la función y se retorna este nuevo contexto.

La función de comprobación de argumentos tiene la finalidad de comprobar que la cantidad de argumentos pasados son los correctos, tiene un objeto res, se comprueba el tamaño de los argumentos obtenidos como parámetro, con el nombre de los argumentos ya establecidos en la función, si los argumentos recibidos en los parámetros son menos o más que los nombres de los argumentos se devuelve un error de ejecución denotando esto, en caso contrario se retorna un éxito vacío.

La función calcular argumentos tiene como función que cada argumento que se pasó se le cambiara el contexto y pasara a estar dentro de la tabla de símbolos del nuevo contexto con su valor correspondiente.

La función calcular y comprobar llama a la función de comprobación de argumentos y la de calcular argumentos.

La clase función, tiene el cuerpo, que seria las cosas que se establecieron para realizar, nodos argumento, que son los parámetros de la función.

La función ejecutar como su nombre menciona esta ejecuta la función, se genera un nuevo interprete, un nuevo contexto a través de la función de nuevo contexto la cual hereda de función base y un objeto resultado de ejecución llamado res, se hace un registro de realizar la función de comprobación y calculaminento de argumentos, si ocurre un error en res se retorna res , se crea una variable valor para llamar la ejecución de la función, que debería de sean ser la misma que fue definida, si ocurre un error en res, este se retornara si no es el caso se retornara res éxito con valor como parámetro.

También se cuenta con una función para copear el contenido de una clase función, además se cuenta con una forma de representación la cual simplemente imprimirá <función y el nombre de esta>

La clase de funciones internas establece las funciones prestablecidas por el lenguaje, la función ejecutar en esta clase tiene un rol parecido a la función de visitar en la clase de interprete, lo que se realiza es que se crea un objeto resultado de ejecución, y también un nuevo contexto (el contexto de ejecución), después se busca el nombre de la función pero se le agrega la palabra ejecutar\_ antes para poder llamar a su respectiva función de ejecución, si el nombre de la función no corresponde con algún método definido se ira a no ejecutar el cual levanta un error mencionando que este método de ejecución no está definido, después se hace un registro de el comprobante de argumentos y su calculador, si ocurre un error se retorna res, si no se crea una variable de valor a retornar la cual guarda el registro del método con el contexto de ejecución (aquí se debería de ejecutar la función) si ocurre un error se retorna el error, en dado caso que todo haya salido bien se retornara un registro exitoso con valor a retornar como parámetro. De igual forma tiene su función de copeado y su función de representación.

Después vienen todas las ejecuciones de las funciones prestablecidas, lo que se realiza dentro de ellas son las ejecuciones de las funciones no se explicaran todas por que es seria extenderlo demasiado, asi que primeramente se explicara lo que se realiza en cada una, la función print imprime lo que se escriba dentro de ella no importa que no tenga comillas, la función print\_ret retorna una cadena (retorna lo que se escribió dentro de ella), input retorna lo que el usuario escriba, input int retorna un numero entero escrito por el usuario, limpiar realiza la instrucción de limpiado del sistema operativo cls si es windows y clear si es Linux o derivados, las funciones es número, lista, función. cadena retornan verdadero si es que el parámetro coincide con lo que se busca (si se usa es\_numero retornara verdadero si el parámetro fue un numero), agregar agrega un elemento a la lista, pop elimina un cierto índice de la lista, conca concatena dos listas, sqrt retorna la raíz de un número, aleatorio entero retorna un numero aleatorio entre los dos números de parámetros incluidos los valores de los parámetros

Contexto

Clase contexto tendrá el contexto del programa, se refiriéndose si estamos dentro de una función o si se está dentro de todo el programa

Tiene como atributos mostrar nombre el cual, almacenara el nombre del contexto, padre será el padre del contexto por ejemplo si se origina un error en alguna función, el contexto seria ese error y el padre de ese contexto seria la función mientras que el padre del contexto de la función sería el programa en general, la posición del padre seria la posición en la cual el contexto cambia (se podría interpretar como la parte en código en la cual se generó la función), y la tabla de símbolos que almacena las variables de cada contexto

Tabla de símbolos

Servirá para rastrear los nombres de las variables y sus valores.

Clase tablasim tiene como atributos una lista llamada símbolos, que almacena las variable, un padre que servirá para almacenar las variables del programa general (por ejemplo, si creamos una función que pide dos variables)

La función get, tiene como parámetro un nombre de variable, primero se le asigna a valor el resultado de llamar la función de Python get usando al atributo padre y como parámetro, nombre lo que hace es que retorna cualquier valor que tenga asignado con esa llave (el nombre, es similar a buscar un índice), también se le asigna el valor determinado None, si es que no encuentra la llave, entonces si el valor y el atributo padre son None se retorna el valor determinado None

Si se encuentra se retorna el valor encontrado.

La función set tiene como parámetros un nombre y un valor, estos se ingresan en la tabla siendo el índice o llave el nombre y el valor el valor obtenido.

La función remover borra una cierta variable si es que esa se llama

AREA INTREPRETE

Clase intreprete: la clase tiene dos parámetros siendo nodo, el cual recibirá los nodos del parecer y contexto

La función de visitar tiene como parámetros el contexto y un nodo, esta ayuda a determinar que función de visitar llamar dependiendo del nodo al cual se refiera, asi que method\_name tiene visitar\_ y se le añade el nombre del nodo que se recibe, asi que se crea una variable que guarda lo que realice la función que tenga method\_name y en dado caso que no exista esta función se realiza la función no\_visitar al finalizar se retorna la variable, pero en forma de lista con nodo y contexto

La función de no visitar tiene como parámetros el nodo y contexto, pero como solo esta sucede cuando no se encuentra el método que tenga el nombre de un dicho nodo, solo crea un error donde se menciona que no se visitó tal tipo de nodo

La función de v NodoOP tiene como parámetros un nodo y contexto, se crea una variable res la cual es un objeto de resultado de ejecución (ResultadoEJ), se crea una variable que contiene el resultado del registro de res evaluando al nodo izquierdo del nodo mandado, junto al contexto, si existe un error en res, este se retornara, vuelve a realizar el mismo proceso con el nodo derecho.

Después de esto lo que pasa es que se busca el valor del token operacional del nodo, dependiendo de este token se realiza dicha operación, guardando el resultado y un posible error, la operación se realiza de izquierda a derecha (el número que existen en izquierda se opera con el número que existe en la derecha)

Si llegara a existir un error, se retornaría el resultado de la función de fallo de res(teniendo como parámetro el error encontrado), en dado caso contrario se retornara el resultado de la función de éxito de res, teniendo como parámetro el resultado de la función de poner posición (el resultado debería ser un numero), con los parámetros de la posición inicial del nodo y la posición final del nodo (recuérdese que aquí solo se entrara a los nodos de tipo operacional), también realizando la función de poner contexto con el parámetro del contexto actual (el que fue recibido en los parámetros )

Nótese que la v es visitar\_

La función de vNodoNumero tienen como parámetros un nodo y el contexto, solo retorna el resultado de la clase RestultadoEj al realizar su función de éxito con un objeto número teniendo el valor del token de dicho nodo, también se le ponen contexto y se le pone la posición con la posición inicial y posición final de dicho nodo, la razón de que se ejecute la función de éxito de ResultadoEj es dado que no existen tokens operacionales en un número.

La funcion vNodoCadena es básicamente lo mismo que la funcion vNodoNumero solo que aquí se retorna un valor tipo cadena

La función de vNodoP tiene como parámetros un nodo y el contexto, se crea res siendo un objeto ResultadoEj y se crea una variable que guardara el resultado del registro de res al visitar el nodo tomado como parámetro y con su dicho contexto, si existe un error, se retornara res, continua poniendo un error con None, se busca que el tipo del token operacional del nodo, si este tiene la etiqueta de menos se realizara una multiplicación por el numero negativo, si el token operacional tiene un partido con la palabra clave NOT realiza una negación, cada realización de operación retorna un posible error y un resultado.

Si existe ese error se retorna el resultado de la función de fallo de res teniendo como parámetro dicho error, en dado caso contrario se retorna el resultado de la función de éxito de res teniendo como parámetro el resultado de la función de poner posición (el resultado debería ser un numero), con los parámetros de la posición inicial del nodo y la posición final del nodo, también junto a la función de poner contexto con el parámetro del contexto recibido.

La funcion vNodoADVar tiene como parámetros un nodo y un contexto, crea una variable res que es un objeto ResultadoEJ, se crea una variable para almacenar el valor del token que está en el nodo, dicho nombre se busca en la tabla de símbolos del contexto mandado, si no se encuentra este nombre se retorna un error de ejecución en el cual se menciona que la variable no está definida (además de pasar los parámetros de posición y contexto a este error).

Se crea una copia del valor, pero cambia los valores de posición inicio y fin a los del token VAR y no a los del valor asignado.

Finalmente retorna res éxito con el valor (la copia) obtenido.

La función vNodoDVar es muy similar a la función anterior solo que valor se iguala al registro de res visitando al nodo mandado, con su dicho contexto, si existiera un error en res este seria retornado, en dado caso que no, se cambiaria el valor de la variable y se retornara el resultado de éxito de res con el valor como parámetro.

La función vNodoif recibe las mismos parámetros que las anteriores clases de visitar, también tiene su respectivo objeto resultado de ejecución, entonces por cada condición y expresión en los casos del nodo, creara una variable condición de ejecución que será el valor del registro tomando como parámetros la visita de la condición y su contexto, si hay un error en res, se retorna res, si la función is true da un verdadero, se crea una variable llamada valor de la expresión la cual almacena el registro de llamar la función de visitar la expresión, si ocurre un error en res se retorna, si no ocurre un error pues se retorna res con el registro de la función éxito de res con valor de expresión como parámetro, al salir de este for, se pregunta si es que existe en el nodo el caso de else, si existe se crea una variable valor else el cual amacena el registro de visitar el coso else del nodo, y su contexto, si ocurre un error en res pues se retorna res, en caso contrario se retorna éxito con valor else como parámetro, si no ocurre ninguno de estos casos se retorna res éxito pero con none como parámetro.

La funcion vNodofor, tiene su respectivo res, primero se crea una variable llamada valor inicial la cual hace registro de la función visitar con el valor inicial del nodo, su dicho contexto, si existe un error en res se retorna res, después está la de valor final la cual también hace un registro de la función visitar, pero ahora es el valor final del nodo, igual si existe un error en res, se retorna res.

Se verifica si es que existe algún valor en caso paso del nodo, si es asi se crea una variable caso paso la cual realiza lo mismo que las anteriores solo que con la visita del valor de caso paso del nodo, en caso de que no se retorna un objeto número con el valor 1.

Una variable i guarda el valor de la variable valor inicio dependiendo del valor de paso se va saber si el valor de inicio es mayor que el valor deseado, o al revés, dependido de esto será la condición que se tomara, mientras exista esta condición lo que se realizara se añadirá el nombre de la variable a la tabla de contexto con su valor inicial, después el valor de i se acumulara sumando el valor de caso paso , después se registra las demás visitas del cuerpo del nood, junto a su contexto, al terminar esta condición se retorna una lista con los elementos visitados

La función vNodowhile, tiene su respectivo res, con un while que se realizara infinitamente, una variable condición guarda el registro la visita de la condición del nodo si ocurre un error en res se retorna res si la condición no es cierta se rompe el ciclo while, si no res hace registro de la visitada del cuerpo del nodo con su dicho contexto, si ocurre un error en res, este se retorna, finalmente se hace registro del éxito retornando una lista con los elementos visitados.

La función vNodoLista tiene su res, se crea una lista para visitar cada elemento existente en la lista, por cada elemento en la lista de elementos del nodo, se añadirá a la lista el registro de visitar cada elemento del nodo, con su dicho contexto, si existe un error retorna res, finalmente retorna el registro exitoso de crear una lista con parámetro la lista que almaceno las visitas, además del contexto y la posición correspondientes

La función vNodofun tiene su respectivo res, primero se obtiene el nombre de la función, si este nombre existe claro, sino se retorna un vacío (None), después se almacena el cuerpo de la función, también pasa con los argumentos del nodo, pero solo el nombre no serían los tokens después se crea un objeto función con el nombre, el cuerpo, y los argumentos como parámetros también se pasa el contexto y la respectiva posición inicio del nodo y la posición fin del nodo

Después si la función tiene un nombre este pasara a la tabla de símbolos, finalmente se retorna el registro con éxito de la función

La funcion vNodollamar tiene su respectivo res y tiene una lista para guardar los argumentos (parametros), se establece una variable de valor a llamar, la cual registra la vistita con su contexto y posición del nodo a llamar el cual debería ser una función ya existente si ocurre un error en el registro se retorna res, se realiza una copia para poder modificar la posición para poder encontrar el error más fácilmente, cada argumento se registra su visita en sus argumentos con su contexto, si ocurre un error en res se retorna res, después finalmente se registra el valor de la función al ejecutarse con los argumentos encontrados, si ocurre un error en el registro se retorna res y finalmente se retorna el registro exitoso del valor al ejecutar al función

Clase ResultadoEj tiene una función muy similar a la de parser resultado, dado que hace un seguimiento del posible error y de los resultados de la clase interprete, cuenta con los atributos de valor y de erro, donde se guardará el valor del resultado de visitar dicho nodo y el posible error de visitar dicho nodo.

La función de registro toma como parámetro res, el cual será un objeto de tipo resultado de ejecución, lo que realiza es que verifica que res no tenga un error, en dado caso que lo tenga este se almacena en error, si no existe el error se retorna el valor que contenga res

La función de éxito al igual que en parser resultado lo que realiza es que actualiza el valor del objeto ReslutadoEj, tiene como parámetro un valor (que debería ser un objeto de tipo número)

La función fallo tiene como parámetro un error (que debería de ser un objeto de la área de error), lo que realiza es actualizar ese error recibido en el atributo de error de la clase.

Área de ejecución (RUN)

Se cera una tabla de símbolos que estarán globalmente, con sus respectivos nombres y valores, cuando se cree el contexto esta tabla será puesta en dicho contexto.

Aquí se recibe el texto que es enviado del archivo de Shell, donde se crea un lexer, y se guardan los tokens que el lexer mediante la función tokenizar devuelve y también el posible error, después se crea un parser el cual crea el árbol de los tokens mandados, retornara el árbol (los nodos) y el posible error, se verifica que no exista un posible error si es asi retorna el error, se crea una variable de contexto la cual es un objeto contexto, indicando que el contexto actual es el programa completo, además se agregan la table de los símbolos globales (de ciertos valores que vienen determinados), continua con una clase de intreprete donde se visitara los nodos que retorna el parser y el contexto, el cual debería de retornarnos un numero asi que se crea una variable que almacene este resultado y posible error.

Al final se le retornara el valor del resultado y el error del resultado

Como declarar variables seria asi

VAR nombre de la variable = valor

Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente



También puede almacenar respuestas de las sentencias if

Texto

Descripción generada automáticamente



Se pueden almacenar funciones ya sea anónimas o ya declaradas

Texto

Descripción generada automáticamente



Sentencia if

If condicione then expresión (suma o algo asi) else if condición then expresión else expresión

Captura de pantalla de computadora

Descripción generada automáticamente



Captura de pantalla de computadora

Descripción generada automáticamente

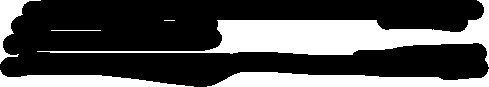


Sentencia for variable = valor inicial to valor final step valor then hacer cosas

Opcional step esto indicaría de cuanto avanza el for

Captura de pantalla de computadora

Descripción generada automáticamente



Captura de pantalla de computadora

Descripción generada automáticamente



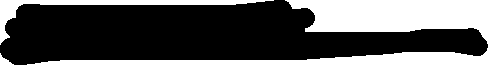
Captura de pantalla de computadora

Descripción generada automáticamente

Sentencia mientras condición then exprecion (un while clásico)

Captura de pantalla de computadora

Descripción generada automáticamente



Funciones

Una captura de pantalla de una computadora

Descripción generada automáticamente



Funcionamiento de las cadenas (string)

Captura de pantalla de computadora

Descripción generada automáticamente

Funcionamiento de las listas:

Captura de pantalla de computadora

Descripción generada automáticamente

Uso de algunas funciones y constantes en el programa, los ceros son porque algunas funciones retornan null(cero).

|  |  |
| --- | --- |
| funciones internas y constantes | significado |
| NULL | retorna cero |
| TRUE | retorna uno |
| FALSE | retorna cero |
| math\_pi | retorna el valor de pi establecido en python |
| print(cosas a escribir) | imprime cosas |
| print\_ret(cosas a escribir) | retorna lo que se escribio en el print |
| inpunt() | el usario puede ingresar cadenas |
| inpunt\_ret() | el usuario puede ingresar números enteros |
| lim() | limpia la pantalla |
| es\_numero(elemento) | regresa TRUE si el elemento es un numero |
| es\_cadena(elemento) | regresa TRUE si el elemento es una cadena |
| es\_lista(elemento) | regresa TRUE si el elemento es una lista |
| es\_funcion(elemento) | regresa TRUE si el elemento es una función |
| agregar (lista, elemento) | agrega un elemento a una lista |
| pop(lista, indice a quitar) | quita el elemento de la lista que este en el índice dado |
| conca (lista, otra lista) | concatena listas |
| sqrt(numero) | retorna el valor de la raíz cuadrada del número dado |
| aleatorio\_ent(número inferior, numero sup) | retorna un numero entero entre el número inferior y el superior incluyéndolos |

Captura de pantalla de computadora

Descripción generada automáticamenteCaptura de pantalla de computadora

Descripción generada automáticamente

Si se pone lim() se limpia la pantalla

Captura de pantalla de computadora

Descripción generada automáticamente

Explicación de calcular el factorial en el lenguaje realizado:

Pues la forma más sencilla de realizar esto es simplemente realizando un for, con el límite siendo el numero que queramos y ya.

Captura de pantalla de computadora

Descripción generada automáticamente

Como podemos ver el ultimo numero en la lista el factorial de 5 que fue el número que el usuario queria saber su factorial, se le sumo uno a ese número por que el for solo establece relaciones < o > (en c sería algo asi for i=1: i<num; i++), es por eso, que se le sumo uno, es recomendable escribir con espacios para que no se generen errores.

Fuentes que ayudaron a desarrollar esto:

<https://ruslanspivak.com/lsbasi-part7/>