**Instituto Tecnológico de Costa Rica**

**Ingeniería en Computación**

**Primer proyecto programado**

**Estudiantes:**

**Juliana Campos Parajeles**

**Johanna Ruiz Bran.**

**Sede San Carlos**

**06/05/2017**

**1. Análisis**

Para el desarrollo de esta etapa del proyecto, es necesario implementar un compilador, el cual tendrá como objetivo el análisis contextual del lenguaje desarrollado. Cabe mencionar que el lenguaje a analizar es muy parecido a Python, sin embargo posee distintas características que lo diferencian de este lenguaje, las cuales algunas serán mencionadas a lo largo del escrito. Entre algunas de las solicitudes que se deberán desarrollar para poder cumplir satisfactoriamente los requerimientos del programa se encuentra lo siguiente:

* Los tipos que se permitirán en esta gramática serán integer, string, char y lista (heterogéneas).
* Las variables poseerán el tipo que fue asignado en su primera declaración y esta no podrá cambiar. Así mismo si no fue declarada deberá de no ser reconocida por el sistema.
* El operador suma deberá de poder utilizarse con tipos de datos int y string únicamente y estos no podrán ser combinados con tipos de datos distintos (no se puede hacer suma de integer y string en una expresión).
* Si se utilizan comparaciones con operadores, los tipos de datos permitidos serán char e integer y no podrán ser comparaciones de tipos distintos.
* Cada vez que se produzca un error se deberá de reportar la línea y columna así a como el motivo de falla.
* Todo método invocado deberá de haber sido declarado, así mismo deberá poseer la misma cantidad de parámetros que posee la función y el mismo tipo de estas.
* Se debe de validar el tipo del retorno de una función.
* Cuando se utilice la función len(), ésta solamente podrá recibir string o listas y deberá de retornar un tipo entero (int).
* La función print(), podrá recibir cualquier tipo de valor válido (siguiendo reglas de no concatenación o relación de datos de distinto tipo).

Entre algunos de los puntos a tomar en cuenta para que el programa funcione correctamente se encuentran:

* El programa no aceptará listas dentro de listas.
* En caso de existir un error dentro de un if statement, def statement o un while statement, lo que se encuentra dentro del scope no se ejecuta o verifica (una vez detectado el error, se corta el proceso) y se siguen analizando otras líneas de código no pertenecientes al dichos métodos.
* Las multiplicaciones, divisiones y restas solamente serán válidas con tipos de datos enteros (int).
* La posición sub de un element acces (lista [X]), deberá de ser números enteros (se aceptarán int solamente X= int). Y este (X) no podrá ser una identificación tipo int, sino que propiamente deberá de ser un numero entero.
* No se permitirán funciones dentro de funciones.
* No se podrán enviar elementos con tipo de dato listas por parámetros.
* Entre algunos de los pasos a seguir en una función se encuentra:
  + Se debe de declarar los parámetros antes de crear a la función. El nivel de dichas variables será el mismo de donde fueron declaradas.
  + Una función def solo podrá ser declarada una vez.
  + Cuando se elimina el nivel por ende los identificadores no se eliminan.
  + Si se usan las variables dentro del def deberá de utilizarse el mismo tipo de dato declarado en la global (las variables globales utilizadas no cambiarán de tipo).

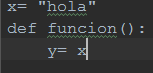
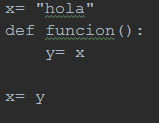
**2. solución o desarrollo**

*Tabla de símbolos:*

Estructura y funcionalidad: Para poder implementar el análisis contextual del lenguaje desarrollado, fue necesaria la creación de una clase, la cual posee la tabla de símbolos. La tabla de símbolos es una lista de objetos de tipo Ident (otra clase que se encuentra ubicada dentro de la clase Tablasimbolos) que contendrá todos los datos como identificaciones de funciones, listas o variables y así mismo dentro de esta almacenar la información necesaria (información de clase Ident que posee atributos: nombre del elemento, tipo, etc). Dentro de esta se encuentran funciones como lo son insertar (inserta dentro de la lista de tabla elementos), eliminar (elimina todos los elementos de un nivel específico enviado) y buscar elementos (buscar la id desde el nivel actual el cual es enviado por parámetro al igual que el texto del id, hasta los niveles más bajos y devuelve el objeto o nulo en caso de no encontrarlo).

*Chequeo de alcances:*

Control de Niveles de los identificadores: Para poder mantener un control sobre el nivel de cada uno de los elementos, fue necesaria la creación de una variable global, en la cual se estipula que esta empieza en el nivel cero (Globales) y en funciones como el if statement, while statement o el def statement (los cuales se encuentran identados y su contenido pasa a formar parte de un nivel más alto), el contador global es aumentado cuando se detecta que entró a visitar a alguno de las sentencias anteriormente descritas. Al insertar un nuevo elemento, se le envía a la función de buscar dentro de la tabla la variable global del nivel que ha sido aumentada anteriormente (se envía por parámetro tanto el texto o nombre del identificador, como el nivel en que se encuentra y el puntero). En caso de existir previamente en el mismo nivel o en niveles anteriores; Se evalúa que lo que se le desea asignar es del mismo tipo que el tipo que este poseía anteriormente, caso contrario reportará un error. Si la variable o identificación no existía en el nivel actual o anterior, se insertará dentro de la tabla de elementos la nueva identificación con los datos y el valor global actual en el que se encuentre. Una vez finalizada la sentencia, la variable global es disminuida y se elimina de la lista o tabla de elementos a todos aquellos que posean un nivel superior al inicial.



Y se encuentra en un nivel más alto por lo tanto al terminar de analizarse la función es eliminado y no es reconocido por las variables de niveles anteriores

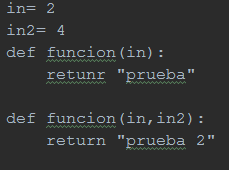
X está declarado en un nivel más bajo (X= “hola”), por lo tanto y puede acceder al valor de x y tener su tipo (2 o string).

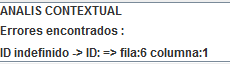
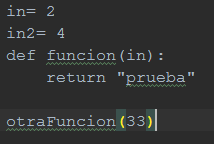
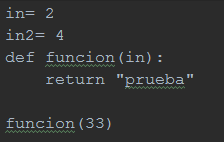
C:\Users\Joha\Desktop\imagescompi\3.PNG

C:\Users\Joha\Desktop\imagescompi\5.PNGC:\Users\Joha\Desktop\imagescompi\Captura.PNGC:\Users\Joha\Desktop\imagescompi\4.PNGControl de Re-declaraciones de identificadores: Al re-declararse un elemento, en caso de ser una lista, se copian los tipos de datos de la lista nueva que poseerá y se suplantan por la que se tenía (la lista vieja poseerá los tipos de datos de los elementos de la nueva lista). Caso contrario solamente se verifica que su tipo sea compatible (del mismo tipo a la declaración anterior), ya que no puede ser distinto al de su origen. Una función def no podrá ser re-declarada.

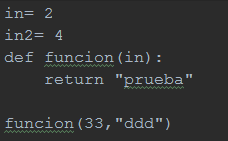
X era de tipo string al pasar a int determina un error de tipo

Como son del mismo tipo su re-asignación está correcta

C:\Users\Joha\Desktop\imagescompi\3.PNG

Control de existencia de métodos: Para comprobar si un método ha sido declarado, al recibir la identificación durante la revisión, esta es enviada a la función buscar de la tabla de elementos la cual devolverá al objeto encontrado en caso de existir. Si este elemento es distinto de nulo, eso significaba que el método existe. Por medio del objeto que es enviado gracias al buscar, se podrá tener conocimiento de toda la información del método existente y se podrá saber sus parámetros y su tipo.

Control de cantidad de parámetros: Es necesario mencionar que al insertar un método, se revisa en la lista de argumentos (argList), que estos sean de los tipos permitidos y que hayan sido declarados anteriormente. Así mismo son introducidos sus tipos en una lista que en caso de no existir error, será enviada como la lista de parámetros que poseerá función nueva.



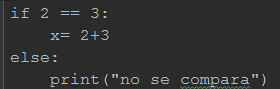
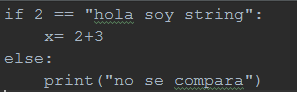
Se analiza que tanto la cantidad de parámetros coincida al igual que su tipo

C:\Users\Joha\Desktop\imagescompi\www.PNG

*Chequeo de tipos:*

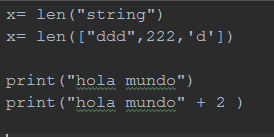
Inferencia y chequeo de tipos en asignaciones: Para poder realizar la inferencia y comparación de tipos, primeramente se busca al id dentro de la lista de la tabla de elementos. Una vez que se ha comprobado que el objeto retornado es igual a nulo (no existe), se visita a la expresión o elemento con el cual se desea asignar. Una vez analizada la expresión, esta devuelve el tipo de dato de dicha expresión, es en este punto donde se evalúa si esta expresión es del mismo tipo que el id encontrado. En caso de no ser así, se ingresa el error dentro del área de texto. Caso contrario, se inserta dicho id dentro de la tabla de elementos con sus distintos atributos correspondientes. Si se detecta que el id ya se encontraba dentro de la tabla (valor retornado diferente de nulo), solamente se analiza que el valor retornado por la expresión sea del mismo tipo que la que posee el objeto retornado (id buscado y encontrado).

Chequeo de tipo en expresiones lógicas: En las expresiones de comparación o lógicas, se visita hasta encontrar el tipo el elemento a comparar tanto de la derecha como de la izquierda de la comparación (“additionExpression”= izquierda y “comparison”= derecha). Una vez que se obtiene el tipo de dato que poseen los elementos de la expresión, se analiza si estos son nulos, si el valor de tipo del lado izquierdo es nulo, se envía un error a la interfaz del usuario. Si el de la derecha el nulo, se sigue evaluando a la expresión. Caso contrario (ambos valores son distintos de nulo), se verifica si estos son iguales (poseen el mismo tipo), si no poseen el mismo tipo, se almacena dentro del área de texto el error (datos incompatibles).

C:\Users\Joha\Desktop\imagescompi\11111.PNG

Uno es de tipo int y el otro de string por lo tanto al compararse no son iguales y se reporta el error en la interfaz del usuario

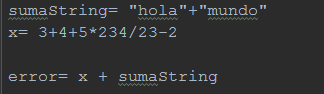
Chequeo de tipos en métodos preestablecidos (print, len): En estos métodos, en el caso del print, se analiza que la expresión contenida dentro de esta esté correcta, una vez analizada esta, se retorna el tipo de dicha expresión. Si esta expresión es igual a nulo (existió un error) o no pertenece a los tipos de datos aceptados dentro del programa, se almacena dentro del área de texto el error. En el len, se analiza que dentro de la expresión que este contiene no se encuentra ningún error (en caso de encontrar alguno se almacena dentro del área de texto y se corta el proceso de esa expresión). Además se analiza que lo que se introduzca dentro de este (la expresión) sea igual a una lista o string, caso contrario se envía un error.

C:\Users\Joha\Desktop\imagescompi\rff.PNG

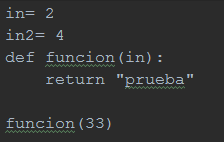
No se puede sumar string con int

X toma valor int porque len retorna int

Chequeo de tipos en Expresiones complejas: Para poder chequear el tipo de dato de las expresiones en suma, resta, multiplicación o división, se deberá de recorrer o visitar toda la estructura hasta llegar al dato primitivo y determinar o retornar el tipo de dato correspondiente al elemento. Una vez recuperado el tipo de dato de la izquierda de la expresión, se evalúa al otro elemento para determinar su tipo. (“multiplicationExpression”= izquierda y “additionFactor”= derecha en caso de suma y resta, en caso de división y multiplicación “elementExpression”=izquierda y “multiplicationFactor”= derecha) En caso de que sean diversos elementos (entro a la derecha de las funciones), se toma que el valor de tipo será el primer elemento que se encuentre. De esta forma si existen distintos elementos, todos deberán de poseer el mismo tipo del primer elemento encontrado (si el primero es int todos el resto debe ser tipo int por ejemplo). Una vez retornado el valor de cada uno de las partes de la expresión, se evalúa si estas son del mismo tipo. Caso contrario se almacena el error. En el caso de sumas se evalúa que los elementos retornados sean de tipo entero o string y sean iguales y en el caso de resta, multiplicación y división, se evalúa que el tipo sea únicamente enteros.

C:\Users\Joha\Desktop\imagescompi\in.PNG

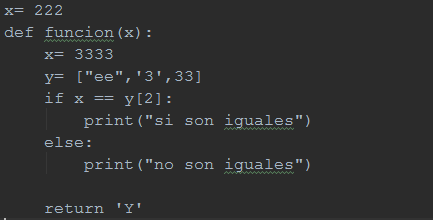
Inferencia de tipos en argumentos de métodos: Para poder conocer el tipo de dato de los distintos parámetros o argumentos que se encuentran un método, se debe de obtener la identificación de cada uno de sus elementos, esto por medio del recorrido o visitas al argList dentro de esta visita se debe de buscar los identificadores. Cuando se encuentren al objeto, se analizará su tipo de dato y este será introducido dentro de la lista provisional de argumentos del método. En caso de que no se encuentre a la identificación en el nivel anterior (la identificación debe de estar declarada anteriormente), se reportará un error.



Debe de estar declarada anteriormente. En este caso in ya fue declarada con valor 2 (int). “in= 2”

Cabe mencionar que a una función solo le pueden llegar identificaciones debido a que así se planteó su estructura en el scanner.

Chequeo de tipos en los retornos de funciones: Para poder determinar de qué tipo es una función gracias a su retorno, es necesario que al realizar una variable global en la cual se almacenará el tipo de retorno que se efectuará en dentro del def. Al realizarse el recorrido para insertar una función o método, al llegar al return (“returnstatement”), en esta visita se evalúa la expresión que se posee dentro. Si la expresión está correcta y es distinta de nulo o igual a los distintos tipos que se poseen dentro del sistema, la variable global se le asigna el valor del tipo retornado en la expresión. Este valor al terminar las visitas correspondientes será enviado como parámetro al insertarse a la función y será su tipo.



Si se encuentra un return se actualiza el valor que tendrá la función como retorno

*Errores*

Reporte de Errores (errores significativos): Para poder reportar estos y más errores, en el instante que se ha detectado un error (no coinciden datos, id no está declarado, etc.), este es introducido al textview para que el usuario pueda observar dichos errores. Algunos errores tuvieron que ser almacenados (la línea y columna), debido a que se reportaron dentro de una función que no poseía el control sobre el elemento a evaluar (el elemento era evaluado más adentro y según lo que tipo que daba en otras funciones, es incorrecta en la que se evalúa actualmente).

**3. Resultados obtenidos**

|  |  |  |
| --- | --- | --- |
| Rubro | | % |
| Contenido | **Tareas** | |
| Tabla de símbolos | Estructura y funcionalidad | 100% |
| Chequeo de alcances | Control de Niveles de los identificadores | 100% |
| Control de Re-declaraciones de identificadores | 100% |
| Control de existencia de métodos | 100% |
| Control de cantidad de parámetros | 100% |
| Reporte de Errores (errores significativos) | 100% |
| Chequeo de tipos | Inferencia y chequeo de tipos en asignaciones | 100% |
| Chequeo de tipo en expresiones lógicas | 100% |
| Chequeo de tipos en métodos preestablecidos (print, len) | 100% |
| Chequeo de tipos en Expresiones complejas | 100% |
| Inferencia de tipos en argumentos de métodos | 100% |
| Chequeo de tipos en los retornos de funciones | 100% |
| Reporte de Errores (errores significativos) | 100% |
| Documentación | Formato y Contenido | 100% |
| Ortografía y Gramática | 100% |

**4. Conclusiones**

Para concluir se puede citar: (diferente no está dentro de la gramática)

* Es necesario seguir los pasos de la estructura gramatical para de esta forma evitar posibles errores durante el recorrido del programa.
* La lista de tabla de elementos del lenguaje es de gran importancia debido a que con esta se puede mantener un control y se puede almacenar datos para posteriormente analizar dicha información.
* No se podrán utilizar elemento que no se hayan declarado anteriormente dentro del scanner del proyecto. Por ejemplo el Scanner no contempla el operador “!=“ por lo tanto este no será reconocido.
* El recorrido en árbol a las diversas funciones del lenguaje fue primordial para poder analizar su contexto.