Dibujo con letras blancas

Descripción generada automáticamente

**Unidad Desconcentrada de la Escuela de Ingeniería en Computación**

**Campus Tecnológico Local San Carlos**

**Compiladores e interpretes**

**Proyecto II**

**Estudiantes:**

Julio Cesar Castro

Josué Orozco Ávila

**Profesor:**

Ing. Oscar Viquez

**Mayo, 2023**

**Soluciones e implementación**

Para el desarrollo de este proyecto se llevó a cabo la implementación el patrón de diseño Visitor para poder implementar validaciones dentro de las reglas contextuales creadas en el AlphaParser.

Para llevar a cabo el desarrollo del proyecto se diseñó lo siguiente:   
  
**Tabla símbolos**:

Cuando en el codigo se crea una nueva declaración ya sea de una variable, clase, método, arreglo, o un dato compuesto, este se agrega a la lista de tabla símbolos, por medio de un token, se guarda en el nivel para definir su alcance además su tipo y tipo de dato.

En caso de que una variable no sea global y solo sea asignada dentro de un método, clase o un operador, esta se elimina de la tabla de símbolos una vez el usuario sale del nivel del método, clase u operador.

**Chequeo y alcances:**

Se realizo un control del alcance de los identificadores según su nivel. Para explicar esto de una mejor manera es mejor ejemplificar de la siguiente manera:

Si tenemos una variable declarada dentro de una función, esta tendrá el nivel de la función, por lo que, si yo quiero utilizarla fuera de esta, generará un reporte de error alertando que la variable no ha sido creada. De esta misma manera se valida si una variable ya fue declarada a un mismo nivel dos veces, es decir, la variable “x” se está declarando 2 veces en un mismo método. También en la verificación de alcance, se verifica el hecho de que una variable que es global se utilice en cualquier parte del codigo, o que una variable de un tipo de clase se pueda usar en un método solo si esta se instancia primero. Lo que permite que dentro de un método exista dos variables “x”, sin que haya choque, debido a que una variable “x” será de tipo clase, y la otra será propia del método, por lo que no generará ningún choque.

La verificación de métodos es basada en la misma lógica que le verificación de variables, dos métodos no se pueden crear en un mismo nivel si estos tienen un mismo nombre.

El control de la cantidad de parámetros es que a la hora de declarar parámetros dentro de una función esta sea capaz de reconocer el tipo y la cantidad de parámetros que son.

Por ultimo el control de nombres de campos de clase se basa en la misma estructura de verificación de las anteriores, no se permite crear dos clases de igual nombre.

El manejo de errores en el chequeo de alcance se basa en identificar y alertar al usuario si está declarando dos clases, métodos o variables en un mismo nivel.

**Chequeo de tipos:**

Para el chequeo de tipos se utilizó una estrategia para manejar los tipos. Se creó un TypesManager que me permite establecer tipos específicos para un arreglo, tipo básico, tipo clase y un tipo compuesto, por lo que para realizar una asignación deben de ser únicamente del tipo permitido en estas clases.

El chequeo de tipo en expresiones lógicas me permite verificar dentro de una verificación si el tipo de las variables que se están comparando son iguales, es decir si comparo “a” con “b”, donde “a” es un char y “b” es un int, esto generará un conflicto de tipos, por lo que se manejará un control de error alertando al usuario que el tipo es incorrecto.

Para una expresión compleja se verifica que todos sean de un mismo tipo, por ejemplo, si a “x” se le asignará el valor de (m-b+z/l\*i) todos deben de ser de tipo int para poder realizar la operación matemática.

El chequeo de tipos en argumentos de métodos verifica que el tipo del parámetro y la cantidad que sean los correctos a la hora de utilizar el método.

Para el chequeo de tipos en campos de clases, lo que se verifica es cuando esta es llamada en otro nivel, para que funcione primero se tiene que instanciar. Esto para poder verificar primeramente si existe la clase para asi asignar o identificar el tipo a la variable que se va a utilizar o agregar un valor.

El chequeo de tipos en los retornos de funciones se basa en verificar que el tipo de dato que ser retorna en la función, sea el mismo tipo de cuando se creó la función. Por ejemplo, una función de tipo int, no puede retornar un dato char.

El manejo de errores se hace en cada verificación de tipado de las asignaciones, expresiones lógicas, retornos, expresiones complejas y argumentos de métodos. Si el tipado no coincide con los tipos asignados en el TypesManager o no coincide a la hora de comparar uno con otro, se le alertará al usuario que tiene un error en el tipado, en la visita que se generó el error y cual fue el tipo de dato que se ingresó.

**Observaciones:**

Por recomendación del profesor no se llevó a cabo las verificaciones ni la documentación de la rubrica “Chequeo de tipos en métodos preestablecidos (ord, chr,len)” debido a que esto ya se realiza dentro de la regla Cast

|  |  |
| --- | --- |
| Rubro | % |
| **Tabla de Símbolos** | **10** |
| Estructura y Funcionalidad | 10 |
| **Chequeo de Alcances** | **30** |
| Control de Niveles de los identificadores | 5 |
| Control de Redeclaraciones de identificadores | 5 |
| Control de existencia de métodos | 5 |
| Control de cantidad de parámetros | 6 |
| Control de nombres de campos en clases | 6 |
| Reporte de Errores (errores significativos) | 3 |
| **Chequeo de tipos** | **50** |
| Chequeo de tipos asignaciones | 4 |
| Chequeo de tipo en expresiones lógicas | 7 |
| Chequeo de tipos en métodos preestablecidos (ord, chr,  len) | 7 |
| Chequeo de tipos en Expresiones complejas | 7 |
| Chequeo de tipos en campos de clases | 7 |
| Chequeo de tipos en argumentos de métodos | 7 |
| Chequeo de tipos en los retornos de funciones | 5 |
| Reporte de Errores (errores significativos) | 6 |
| **Documentación** | **10** |
| Formato y Contenido | 5 |
| Ortografía y Gramática | 5 |
|  | **100** |

Bibliografía:

ANTLR. (s.f.). ANTLR - Parser generator. Recuperado el 15 de mayo de 2023, de <https://www.antlr.org/>

Microsoft. (s.f.). C# (Visual C#). Recuperado el 15 de mayo de 2023, de <https://learn.microsoft.com/es-es/dotnet/csharp/>

**VarDeclAST**

Estructura en el parser:

*varDecl : type ident (COMMA ident)\* SEMICOLON #VarDeclAST;*

Estructura en el AContextual:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Explicación:

Para la declaración de una variable se debe verificar el tipo de esta, por lo que debido que tenemos declaración de variables y listas, primeramente se hace la verificación de esto. Por lo que se hace un Substring para dividir el tipo de la variable en 2, si los últimos 2 caracteres son iguales a “[]” quiere decir que el tipo de dato es una lista, si este no tiene los corchetes cuadrados quiere decir que el dato es un tipo de dato básico.

Si el tipo de variable es básica, se realiza un visit child para ver todas las variables declaradas con un tipo, por ejemplo: int x,y,z.

Para la declaración de arreglos se verifica que el tipo de arreglo declarado exista, si este existe se realiza un visit child para verificar todos los arrays que hayan de este tipo.

Por ultimo está el tipo compuesto, este dato es de una instancia de una clase, por lo que se verifica primero que la clase exista, si este existe de igual manera se realiza un visit child para guardar todos sus instancias