



Escuela de Ingeniería en Computación

IC-5701 Compiladores e intérpretes

Grupo 01

Profesor Ignacio Trejos Zelaya

Proyecto 01 - Fase 02

Análisis Contextual

Estudiantes

David Castro Holguin - 2018105813

Keila Álvarez Bermúdez - 2018227606

Anner Calvo Granados - 2017107433

Fecha

12/11/2021

Tabla de contenidos

Modificaciones al compilador	3
Triangle	3
Contextual Analyzer	3
Checker	3
IdentificationTable	7
Descripciones	7
Plan de Pruebas Contextuales	11
Discusión y análisis de los resultados obtenidos	19
Una reflexión sobre la experiencia de modificar fragmentos de un compilador	19
Instrucciones de Compilación	20
Instrucciones de ejecución	20

Modificaciones al compilador

Triangle

Contextual Analyzer

Checker

- visitLetCommand

```
public Object visitLetCommand(LetCommand ast, Object o) {
    idTable.openScope();
    ast.D.visit(this, null);
    ast.C.visit(this, null);
    idTable.closeScope();
    return null;
}
```

- visitRepeatUntilDo

```
public Object visitRepeatUntilDo(RepeatUntilDoCommand ast, Object o) {
    // Exp debe ser de tipo Boolean.
    TypeDenoter eType = (TypeDenoter) ast.E.visit(this, null);
    if (!eType.equals(StdEnvironment.booleanType)) {
        reporter.reportError("Boolean expression expected here", "",
            ast.E.position);
    }
    // Com y sus partes deben satisfacer las restricciones contextuales
    ast.C.visit(this, null);
    return null;
}
```

- visitRepeatWhileDo

```
public Object visitRepeatWhileDo(RepeatWhileDoCommand ast, Object o) { //
    // Exp debe ser de tipo Boolean.
    TypeDenoter eType = (TypeDenoter) ast.E.visit(this, null);
    if (!eType.equals(StdEnvironment.booleanType)) {
        reporter.reportError("Boolean expression expected here", "",
            ast.E.position);
    }
    // Com y sus partes deben satisfacer las restricciones contextuales
    ast.C.visit(this, null);
    return null;
}
```

- visitRepeatDoUntilCommand

```

public Object visitRepeatDoUntilCommand(RepeatDoUntilCommand ast, Object o) {
    // Exp debe ser de tipo Boolean.
    TypeDenoter eType = (TypeDenoter) ast.E.visit(this, null);
    if (!eType.equals(StdEnvironment.booleanType)) {
        reporter.reportError("Boolean expression expected here", "",
            ast.E.position);
    }
    // Com y sus partes deben satisfacer las restricciones contextuales
    ast.C.visit(this, null);
    return null;
}

```

- visitRepeatInCommand

```

public Object visitRepeatInCommand(RepeatInCommand ast, Object o) { // Se agre
                                                                    // AST
    // Exp debe ser de tipo array InL of TD.
    // Id es conocida como la variable de iteracion. Id es de tipo TD. Id es
    // declarada en este comando y su alcance es Com; esta declaracion de Id
    // no es conocida por Exp.
    idTable.openScope();
    ast.D.visit(this, null);
    ast.C.visit(this, null);
    idTable.closeScope();
    return null;
}

```

- visitRepeatForRangeDoCommand

```

public Object visitRepeatForRangeDoCommand(RepeatForRangeDoCommand ast, Object o) {
    TypeDenoter eType = (TypeDenoter) ast.E.visit(this, null);
    if (!eType.equals(StdEnvironment.integerType))
        reporter.reportError("integer expression expected in second expression",
            "", ast.E.position);
    else {
        idTable.openScope();
        ast.D.visit(this, null);
        ast.C.visit(this, null);
        idTable.closeScope();
    }
    return null;
}

```

- visitRepeatForRangeUntilCommand

```
public Object visitRepeatForRangeUntilCommand(RepeatForRangeUntilCommand ast, Object o) { //
//
    TypeDenoter e1Type = (TypeDenoter) ast.E1.visit(this, null);
    if (!e1Type.equals(StdEnvironment.integerType))
        reporter.reportError("Integer expression expected here", "", ast.E1.position);
    else {
        idTable.openScope();
        ast.D.visit(this, null);
        TypeDenoter e2Type = (TypeDenoter) ast.E2.visit(this, null);
        if (!e2Type.equals(StdEnvironment.booleanType))
            reporter.reportError("Boolean expression expected here", "", ast.E2.position);

        ast.C.visit(this, null);
        idTable.closeScope();
    }
    return null;
}
```

- visitRecursiveDeclaration

```
public Object visitRecursiveDeclaration(RecursiveDeclaration ast, Object o) {
    this.saveId = true;
    this.recOn = true;
    ast.D1.visit(this, null);
    this.saveId = false;
    ast.D1.visit(this, null);
    this.recOn = false;
    return null;
}
```

- visitProcFuncDeclaration

```
public Object visitProcFuncSDeclaration(ProcFuncSDeclaration ast, Object o) {
    ast.D1.visit(this, null);
    ast.D2.visit(this, null);
    return null;
}
```

- visitFuncDeclaration

```
public Object visitFuncDeclaration(FuncDeclaration ast, Object o) { // HUBO UN CAMBIO(
    if (this.saveId) {
        this.visitFuncDeclarationID(ast, o);
    } else if (this.saveId == false) {
        this.visitFuncDeclarationBody(ast, o);
    } else if (this.recOn == false) {
        this.visitFuncDeclarationID(ast, o);
        this.visitFuncDeclarationBody(ast, o);
    }
    return null;
}
```

- visitProcDeclaration

```
public Object visitProcDeclaration(ProcDeclaration ast, Object o) { // HUBO UN CAMBIO0000000000000000
    if (this.saveId == true & this.recOn == true) {
        this.visitProcDeclarationID(ast, o);
    } else if (this.saveId == false & this.recOn == true) {
        this.visitProcDeclarationBody(ast, o);
    } else if (this.recOn == false) {
        this.visitProcDeclarationID(ast, o);
        this.visitProcDeclarationBody(ast, o);
    }
    return null;
}
```

- visitFuncDeclarationID

```
public Object visitFuncDeclarationID(FuncDeclaration ast, Object o) {
    ast.T = (TypeDenoter) ast.T.visit(this, null);
    idTable.enter(ast.I.spelling, ast); // permits recursion
    if (ast.duplicated)
        reporter.reportError("identifier \"%s\" already declared", ast.I.spelling, ast.position);
    idTable.openScope();
    ast.FPS.visit(this, null);
    idTable.closeScope();
    return null;
}
```

- visitFuncDeclarationBody

```
public Object visitFuncDeclarationBody(FuncDeclaration ast, Object o) {
    idTable.openScope();
    ast.FPS.visit(this, null);
    TypeDenoter eType = (TypeDenoter) ast.E.visit(this, null);
    idTable.closeScope();
    if (!ast.T.equals(eType))
        reporter.reportError("body of function \"%s\" has wrong type", ast.I.spelling, ast.E.position);
    return null;
}
```

- visitProcDeclarationID

```
public Object visitProcDeclarationID(ProcDeclaration ast, Object o) {
    idTable.enter(ast.I.spelling, ast); // permits recursion
    if (ast.duplicated)
        reporter.reportError("identifier \"%s\" already declared", ast.I.spelling, ast.position);
    idTable.openScope();
    ast.FPS.visit(this, null);
    idTable.closeScope();
    return null;
}
```

- visitProcDeclarationBody

```
public Object visitProcDeclarationBody(ProcDeclaration ast, Object o) {
    idTable.openScope();
    ast.FPS.visit(this, null);
    ast.C.visit(this, null);
    idTable.closeScope();
    return null;
}
```

- visitLocalDeclaration

```
public Object visitLocalDeclaration(LocalDeclaration ast, Object o) {
    idTable.openScope();
    ast.D1.visit(this, null);
    idTable.openScope();
    ast.D2.visit(this, null);
    idTable.to_assign();
    return null;
}
```

IdentificationTable

- To_assing()

Se crea un método to_assign que es el encargado que llevar a cabo la funcionalidad del local.

```
public void to_assign() {
    IdEntry entryDeclaration = this.latest, localDeclaration = entryDeclaration, localEntry;

    while (entryDeclaration.level == this.level) {
        localDeclaration = entryDeclaration;
        localDeclaration.level = localDeclaration.level - 2;
        entryDeclaration = localDeclaration.previous;
    }

    while (entryDeclaration.level == this.level - 1) {
        localEntry = entryDeclaration;
        entryDeclaration = localEntry.previous;
    }
    localDeclaration.previous = entryDeclaration;

    this.level = level - 2;
}
```

Descripciones

- Repeat ... end

repeat while Exp do Com end

repeat until Exp do Com end

Los comandos repeat while do y repeat until do, ambos están conformados por una expresión y un comando, la parte contextual de dicho comando se ejecuta por medio del método visitRepeatWhileDo y visitRepeatUntilDo respectivamente.

Primeramente se crea una variable de tipo TypeDenoter en la cual se guardará el tipo de la expresión que se encuentra en su respectivo ast el cual ingresa por parámetro. Después se pregunta si el tipo de esta expresión es booleano por medio de un if, en caso de que la expresión no sea booleana entrará al if en el cual se enviará un reportError con el mensaje "Boolean expression expected here".

Seguidamente por medio del ast que ingresa como parámetro al método y el comando de dicho ast se ejecuta el método visit en el cuál se comprueba si el comando satisface las restricciones contextuales. Por último, el método retorna null.

repeat do Com while Exp end**repeat do Com until Exp end**

Los comandos repeat do while y repeat do until, ambos están conformados por un comando y una expresión, la parte contextual de dicho comando se ejecuta por medio del método visitRepeatDoWhileCommand y visitRepeatDoUntilCommand respectivamente.

Primeramente se crea una variable de tipo TypeDenoter en la cual se guardará el tipo de la expresión que se encuentra en su respectivo ast, el cual ingresa por parámetro. Después se pregunta si el tipo de esta expresión es booleano por medio de un if, en caso de que la expresión no sea booleana entrará al if en el cual se enviará un reportError con el mensaje "Boolean expression expected here".

Seguidamente por medio del ast que ingresa por parámetro al método y el comando de dicho ast se ejecuta el método visit en el cuál se comprueba si el comando satisface las restricciones contextuales. Por último, el método retorna null.

- Repeat for .. end

repeat for Id := range Exp1 .. Exp2 do Com end

El comando repeat for range do está conformado por una declaración la cual contiene un identificador y una expresión, también lo conforma una expresión y un comando, la parte contextual de dicho comando se ejecuta por medio del método visitRepeatForRangeDoCommand.

Primeramente se crea una variable de tipo TypeDenoter en la cual se guardará el tipo de la segunda expresión que se encuentra en su respectivo ast, el cual ingresa por parámetro. Después se pregunta si el tipo de esta expresión es entero por medio de un if, en caso de que la expresión no sea un número entero entonces entrará al if en el cual se enviará un reportError con el mensaje "integer expression expected in second expression".

En caso contrario entrará al else, donde se abre el openScope haciendo uso del idTable, para la declaración y el comando. Seguidamente por medio del ast que ingresa como parámetro al método y la declaración y el comando de dicho ast se ejecuta el método visit en el cuál se comprueba si estos satisfacen las restricciones contextuales. Por último, se cierra el scope por medio del closeScope, igualmente haciendo uso del idTable y se retorna null.

repeat for Id := range Exp1 .. Exp2 while Exp3 do Com end**repeat for Id := range Exp1 .. Exp2 until Exp3 do Com end**

El comando repeat for range while do y el comando repeat for range until están conformados por una declaración la cual contiene un identificador y una expresión, también los conforman dos expresiones y un comando, la parte contextual de dicho comando se ejecuta por medio de los métodos visitRepeatForRangeWhileCommand y visitRepeatForRangeUntilCommand respectivamente.

Primeramente se crea una variable de tipo TypeDenoter en la cual se guardará el tipo de la segunda expresión que se encuentra en su respectivo ast, el cual ingresa por parámetro.

Después se pregunta si el tipo de esta expresión es entero por medio de un if, en caso de que la expresión no sea un número entero entonces entrará al if en el cual se enviará un reportError con el mensaje "integer expression expected here".

En caso contrario ingresa al else, donde se abre el openScope haciendo uso del idTable, esto para la declaración y el comando. Seguidamente por medio del ast que ingresa como parámetro al método y la declaración de dicho ast se ejecuta el método visit en el cuál se comprueba si el esta satisface las restricciones contextuales.

Después, se crea otra variable de tipo TypeDenoter donde al igual que en la anterior se guardará el tipo de la expresión, en este caso el de la tercera expresión que compone el comando. Nuevamente se hace la verificación del tipo por medio de un if donde se pregunta si esta expresión no es de tipo booleano, en caso de no serlo entrará if en el cual se enviará un reportError con el mensaje "Boolean expression expected here". Por último, se cierra el scope por medio del closeScope, igualmente haciendo uso del idTable y se retorna null.

repeat for Id in Exp do Com end

El comando repeat for in está conformado por una declaración la cual contiene un identificador y una expresión, también está conformado por un comando, la parte contextual de dicho comando se ejecuta por medio del método visitRepeatInCommand.

Primeramente se abre el openScope haciendo uso del idTable, para la declaración y el comando. Seguidamente por medio del ast que ingresa como parámetro al método y la declaración de dicho ast se ejecuta el método visit en el cual se comprueba si esta satisface las restricciones contextuales. Por último, se cierra el scope por medio del closeScope, igualmente haciendo uso del idTable y se retorna null.

- **Let Dec in Com end**

El comando let in está conformado por una declaración y un comando, la parte contextual de dicho comando se ejecuta por medio del método visitLetCommand.

Primeramente se abre el openScope haciendo uso del idTable, para la declaración y el comando. Seguidamente por medio del ast que ingresa como parámetro al método y la declaración y comando que lo conforman se ejecuta el método visit en el cual se comprueba si estos satisfacen las restricciones contextuales. Por último, se cierra el scope por medio del closeScope, igualmente haciendo uso del idTable y se retorna null.

- **var Id := Exp (Declaración de variable inicializada)**

Para crear una variable en el checker y que podamos usarla posteriormente, se hace uso de la funcionalidad TypeDenoter, la cuál deriva del AST padre, y tiene como trabajo verificar los tipos de variables, por lo que al crear una variable derivada de un ast, se hace un "casteo" del tipo que trae el ast(Posteriormente se explica de mejor manera).

- **Validación de unicidad**

Para obtener un valor de unicidad en las variables se hace uso del ast Declaration, el cual tiene como una de sus funcionalidades el verificar si existe un duplicado o no, al hacer el análisis se lleva a cabo de la siguiente manera, al ingresar a la sección de los identificadores(se explica su funcionamiento más abajo), se asigna un ast que tendrá el valor a evaluar, posteriormente este ast se llama con la funcionalidad duplicated el cuál trabaja con un valor booleano, por lo que, si el resultado es verdadero significa que hay complicaciones con el nombre, es decir, está duplicado y devuelve un reporte de error informando la situación, de no ser así se continúa con el flujo normal de la ejecución.

- Local

Inicialmente el local abre el openScope desde el IdentificationTable haciendo uso del idTable, para la declaración número 1, lo que nos permite cambiar el nivel máximo, de igual manera pasa con la declaración número 2.

Se abre el openScope de IdentificationTable, una vez hecho esto es momento de asignar la exportación, por lo que para tomar la información de la primera declaración se debe bajar el nivel del scope la misma cantidad de declaraciones que entraron, 2 en este caso.

El siguiente paso es tomar las entradas de la segunda declaración dado que son las que se exportan, haciendo posible que la segunda declaración pueda ver las entradas de la primera declaración, pero sin que estas segundas se exporten, una vez culminada esta funcionalidad se cierra el scope.

- Recursive

Para el análisis del recursive se lleva a cabo la estrategia de minipasadas. Se hace uso de dos variables para hacer el recorrido en el análisis, las cuales funcionan como banderas para guardar movimientos y saber que la recursión está en progreso.

El análisis del mismo se hace por partes, inicialmente se hace un llamado a los visit correspondientes, donde, en el caso de FuncDeclaration ingresa llamando a la parte de la identificación, en esta parte se revisa usando un TypeDenoter la correcta estructura de la entrada, la cual es agregada al idTable haciendo a través del metodo enter. Una vez hecho esto se tiene una verificación que revisa la no duplicación, de ser este el caso emite un reporte con la eventualidad.

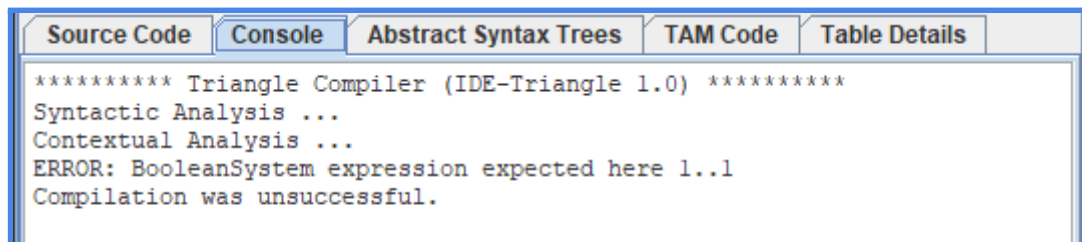
Cuando culmina esta parte se hace un openScope de identificationTable para aumentar el nivel y se visita el ast del FormalParameterSequence, el cuál posteriormente se cierra.

Para la segunda parte en la evaluación de FuncDeclaration se analiza el “body”, el cuál sigue la siguiente estructura: se incrementa el nivel usando el openScope de identificationTable y se visita el ast del FormalParameterSequence, posteriormente se analiza el tipo de la expresión usando un TypeDenoter, el cuál se revisa teniendo que ser igual que el tipo de la variable inicializada al inicio del recorrido en la parte de la identificación.

De igual manera funciona con el ProcDeclaration el cuál sigue la misma estructura que el FuncDeclaration en su identificador mencionado anteriormente, sin embargo, en la sección del “body” este inicia aumentando el nivel haciendo uso del openScope para posteriormente visitar el ast del FormalParameterSequence y el ast del Comando.

Nuevos errores contextuales

Al realizar modificaciones dentro del lenguaje de triángulo, aparecieron nuevos errores contextuales dentro del sistema. Por ejemplo al añadir las nuevas condiciones contextuales de tipos, surgen errores como el de la imagen en el que se genera un error al no cumplir con el tipo de expresión requerida. Ocurre del mismo modo con todas las nuevas reglas contextuales añadidas.



The screenshot shows the 'Console' tab of the Triangle Compiler IDE. The text in the console is as follows:

```
***** Triangle Compiler (IDE-Triangle 1.0) *****  
Syntactic Analysis ...  
Contextual Analysis ...  
ERROR: BooleanSystem expression expected here 1..1  
Compilation was unsuccessful.
```

Plan de Pruebas Contextuales

Pruebas If

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
IfErr1	! Error: ! error en condición ! tipos incompatibles	Muestra error y no compila	Muestra error y no compila
IfErr2	! Error:	Muestra error y no compila	Muestra error y no compila
IfErr3	! Error:	Muestra error y no compila	Muestra error y no compila
IfOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
IfOk2	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta.
IfOk3	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
IfOk4	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas Let

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
LetErr1	! Error: ! debe dar un error, a es una constante	Muestra error y no compila	Muestra error y no compila
LetErr2	! Error: ! debe dar un error, a es una constante	Muestra error y no compila	Muestra error y no compila
LetOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
LetOk2	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
LetOk3	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas local

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
LocalErr1	! Error: ! Fuera de local no se ve lo que es privado	Muestra error y no compila	Muestra error y no compila
LocalErr2	! Error: ! a no debe ser visible aquí	Muestra error y no compila	Muestra error y no compila
LocalErr3	! Error: ! b no debe ser visible aquí	Muestra error y no compila	Muestra error y no compila
LocalErr4	! Error: ! a no debe ser visible aquí	Muestra error y no compila	Muestra error y no compila
LocalErr5	! Error: ! b no debe ser visible aquí	Muestra error y no compila	Muestra error y no compila
LocalErr6	! Error: ! c no debe ser visible aquí	Muestra error y no compila	Muestra error y no compila
LocalOk1	! Ok.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
LocalOk2	! Ok.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
LocalOk3	! Ok	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
LocalOk4	! Ok.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

LocalOk5	! Ok.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
-----------------	-------	--	--

Pruebas recursive

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RecursiveErr1	! Error: ! Declaraciones mutuamente recursivas, con errores de tipo	Muestra error y no compila	Muestra error y no compila
RecursiveErr2	! Error: ! Declaraciones mutuamente recursivas, con número de parámetros incorrecto	Muestra error y no compila	Muestra error y no compila
RecursiveErr3	! Error: ! Nombre de función repetido	Muestra error y no compila	Muestra error y no compila
RecursiveErr4	! Error: ! Declaraciones con parámetros repetidos	Muestra error y no compila	Muestra error y no compila
RecursiveOk1	! OK.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RecursiveOk2	! OK.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RecursiveOk3	! OK.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RecursiveOk4	! OK.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RecursiveOk5	! OK.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RecursiveOk6	! OK.	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Prueba RepeatDoUntil

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatDoUntilErr1	! Error: ! condición no booleana	Muestra error y no compila	Muestra error y no compila
RepeatDoUntilOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas RepeatDoWhile

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatDoWhileErr1	! Error: ! condición no booleana	Muestra error y no compila	Muestra error y no compila
RepeatDoWhileOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas RepeatFor

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatForErr1	! Error: ! i debe ser inaccesible aquí	Muestra error y no compila	Muestra error y no compila
RepeatForErr2	! Error: ! debe imprimir 10 unos, si no hubiera otro error...	Muestra error y no compila	Muestra error y no compila
RepeatForErr3	! Error: ! k debería ser entera	Muestra error y no compila	Muestra error y no compila
RepeatForErr4	! Error: ! debe dar error, i es entera, no char	Muestra error y no compila	Muestra error y no compila
RepeatForErr5	! Error: ! debe dar un error, variable protegida no debería poder pasarse por referencia	Muestra error y no compila	Muestra error y no compila
RepeatForErr6	! Error: ! debe dar un error, variable protegida no debe poder ser asignada	Muestra error y no compila	Muestra error y no compila

RepeatForOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RepeatForOk2	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RepeatForOk3	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas RepeatForIn

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatForInErr1	! Error: ! expresión (proveedora de datos) no es un arreglo	Muestra error y no compila	Muestra error y no compila
RepeatForInErr2	! Error: ! x no es conocida en la expresión	Muestra error y no compila	Muestra error y no compila
RepeatForInErr3	! Error: ! x no puede ser modificada	Muestra error y no compila	Muestra error y no compila
RepeatForInErr4	! Error: ! x no es visible aquí	Muestra error y no compila	Muestra error y no compila
RepeatForInErr5	! Error: ! x es un carácter del arreglo. error: no es putint()	Muestra error y no compila	Muestra error y no compila
RepeatForInErr6	! Error: ! la variable protegida x no debe pasarse por variable	Muestra error y no compila	Muestra error y no compila
RepeatForInOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RepeatForInOk2	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RepeatForInOk3	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RepeatForInOk4	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas RepeatForUntil

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatForUntilErr1	! Error: ! i debe ser inaccesible aquí	Muestra error y no compila	Muestra error y no compila
RepeatForUntilErr2	! Error: ! k debe ser entera	Muestra error y no compila	Muestra error y no compila
RepeatForUntilErr3	! Error:	Muestra error y no compila	Muestra error y no compila
RepeatForUntilErr4	! Error: ! debe dar error, i es entera, no char	Muestra error y no compila	Muestra error y no compila
RepeatForUntilErr5	! Error: ! debe dar un error, variable protegida no debería poder pasarse por referencia	Muestra error y no compila	Muestra error y no compila
RepeatForUntilErr6	! Error: ! debe dar un error, variable protegida no debe poder ser asignada	Muestra error y no compila	Muestra error y no compila
RepeatForUntilErr7	! Error: ! condición no booleana	Muestra error y no compila	Muestra error y no compila
RepeatForUntilOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RepeatForUntilOk2	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RepeatForUntilOk3	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas RepeatForWhile

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatForWhileErr1	! Error: ! i debe ser inaccesible aquí	Muestra error y no compila	Muestra error y no compila
RepeatForWhileErr2	! Error: ! k debe ser entera	Muestra error y no compila	Muestra error y no compila
RepeatForWhileErr3	! Error: ! k debería ser entera	Muestra error y no compila	Muestra error y no compila

RepeatForWhileErr4	! Error: ! debe dar error, i es entera, no char	Muestra error y no compila	Muestra error y no compila
RepeatForWhileErr5	! Error: ! debe dar un error, variable protegida no debería poder pasarse por referencia	Muestra error y no compila	Muestra error y no compila
RepeatForWhileErr6	! Error: ! debe dar un error, variable protegida no debe poder ser asignada	Muestra error y no compila	Muestra error y no compila
RepeatForWhileErr7	! Error: ! condición no booleana	Muestra error y no compila	Muestra error y no compila
RepeatForWhileOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RepeatForWhileOk2	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
RepeatForWhileOk3	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas RepeatUntilDo

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatUntilDoErr1	! Error: ! condición no booleana	Muestra error y no compila	Muestra error y no compila
RepeatUntilDoOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas RepeatWhileDo

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatWhileDoErr1	! Error: ! error contextual: a y b no están declaradas	Muestra error y no compila	Muestra error y no compila
RepeatWhileDoErr2	! Error: ! error de tipo	Muestra error y no compila	Muestra error y no compila
RepeatWhileDoOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas Skip

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
SkipOK1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
SkipOK2	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Pruebas VarInit

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
VarInitErr1	! Error: ! put imprime caracteres, no enteros	Muestra error y no compila	Muestra error y no compila
VarInitErr2	! Error:	Muestra error y no compila	Muestra error y no compila
VarInitOk1	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
VarInitOk2	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
VarInitOk3	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta
VarInitOk4	! OK	No muestra error y compila de forma correcta	No muestra error y compila de forma correcta

Discusión y análisis de los resultados obtenidos

Para el plan de prueba se decide hacer uso de todas las herramientas brindadas por el profesor, por ende, se plantean todos los casos de prueba y se presentan en una tabla donde se agrupan por cada una de las funcionalidades disponibles. Todas las pruebas se centran en los siguientes componentes principales, la compilación, verificación de que los elementos del análisis sintáctico y léxico funcionen de manera correcta, y se presentan los errores correspondientes.

Cuando se inician las pruebas y documentación de las mismas, se toma la decisión de comenzar con los errores, dado que son los más sencillos de identificar, dado que al compilar el archivo .tri el error se muestra inmediatamente y este mismo error es comparado con la información presentada por el profesor con respecto al motivo de los errores a fin de corroborar que el proceso tanto correcto como incorrecto fuese hecho de manera correcta. De igual forma se llevó a cabo el proceso con el plan de pruebas positivas, el objetivo en este caso es que todas las pruebas muestran que la compilación fue correcta, esto partiendo con los nuevos elementos del analizador contextual, además de visualizar el HTML creado en cada caso así como su respectivo XML.

Todas las pruebas presentaron el comportamiento esperado, tanto las pruebas que deberían dar error, como las pruebas que son exitosas en la compilación. Se concluye, una vez realizadas las pruebas, que los resultados obtenidos estaban dentro de las expectativas que se tienen en cada uno de los procesos. En general, el grupo se concuerda con que se logró hacer un buen trabajo, y que se logró comprender todo el proceso para un análisis sintáctico y léxico de un compilador.

Una reflexión sobre la experiencia de modificar fragmentos de un compilador

Modificar un analizador contextual creado por alguien más fue muy gratificante y enriquecedor ya que se comprende cómo crearlos de una forma inteligente y sencilla, además de al ir recorriendo el código se aprenden buenas técnicas de programación que siempre son bien recibidas. Además, nos da la posibilidad de poner en práctica lo aprendido en el curso e ir despejando dudas que surgen conforme el programador se adentra más en el desarrollo del código. También es de ayuda para comprender cómo el funcionamiento sintáctico y léxico afecta directamente el correcto funcionamiento de un programa con este.

Instrucciones de Compilación

Para compilar el programa es necesario abrir el proyecto preferiblemente en NetBeans, esto dado que el desarrollo del proyecto ya fue implementado y la valor ahora es mejorarlo, y según instrucciones la mejor opción el este entorno de desarrollo. Posteriormente se abre un nuevo proyecto y se abre la carpeta llamada ide-triangle-v1.1.src, esta carpeta contiene toda la parte relevante del proyecto, todas la carpetas que son necesarias para la implementación del código.

Una vez dentro de esta carpeta, hay que dirigirse a la carpeta llamada scr y posteriormente a la carpeta llamada GUI, donde se encuentra un archivo llamado main.java que será el encargado de llevar a cabo la ejecución del proyecto. Posteriormente a encontrar este archivo main.java lo que resta es la ejecución del mismo, la cuál, si se está en netbeans, será necesario únicamente acceder en las opciones de la parte superior del entorno a la pestaña llamada RUN y ahí seleccionar la primera opción, o en su defecto presionar F6, que es el atajo del teclado predeterminado para correr un programa en netbeans.

Instrucciones de ejecución

- Abrir la interfaz del programa, para abrirlo únicamente hay que dar click al archivo .jar adjunto.
- Una vez abierto el programa, se debe acceder al explorador de archivos
- Elegir en la carpeta de pruebas uno de los archivos .tri disponibles para hacer la prueba correspondiente.
- Cuando se haya cargado el archivo .tri, es momento de hacer la compilación, esta se lleva a cabo dando click a las llaves que aparecen en el header de la interfaz del programa.

Descripción de las tareas realizadas

Keila y David

- Implementación del Var
- Implementar el IF y sus modificaciones
- Implementar el Let y sus modificaciones
- Implementar los Repeat y sus variantes
- Implementar la documentación de las partes que modificó

Anner

- Implementación de la funcionalidad Local
- Implementación de la funcionalidad Recursive
- Implementar la documentación de las partes que modificó