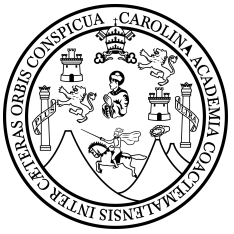
Centro Universitario de Occidente
Universidad de San Carlos de Guatemala
Ingeniería en Ciencias y Sistemas
División de Ciencias de la Ingeniería
Organización de lenguajes y compiladores 2



Proyecto 1
Manual Tecnico

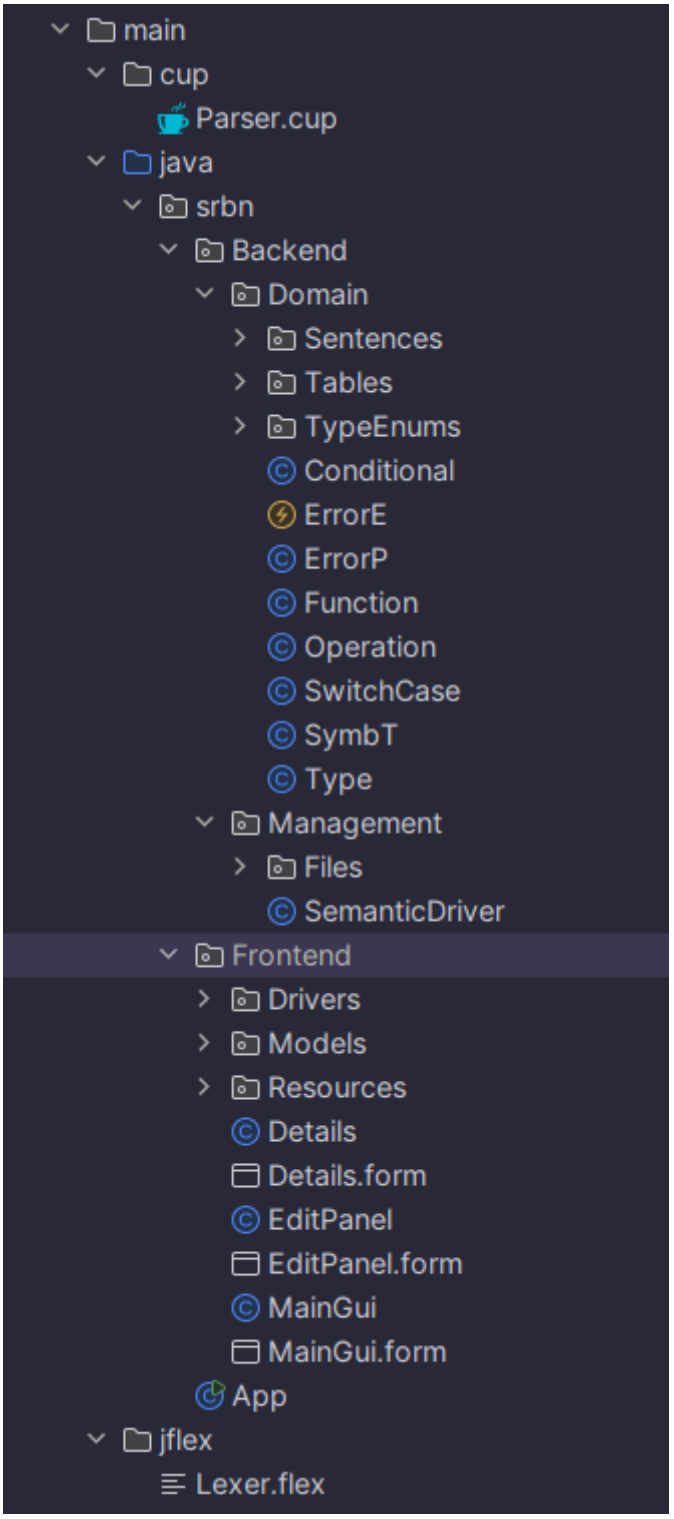
Byron Javier Vasquez Villagran
201931806

Herramientas de desarrollo

Las herramientas (IDEs, lenguaje de programación, librerías, etc) usadas para el desarrollo de la aplicación (pascal semantic analyzer) se listan a continuación

- Lenguaje
  - java
    - version 17.0.8
- Analizador lexico:
  - JFlex
    - version 1.9.1
- Analizador Sintactico
  - Cup runtime
    - version 11b
- Entorno de desarrollo (IDE)
  - IntelliJ IDEA
    - version 2024.2 (ultimate edition)
- Sistema operativo de desarrollo
  - Windows 11 - home
    - version 10.0.22621

Organización del código fuente

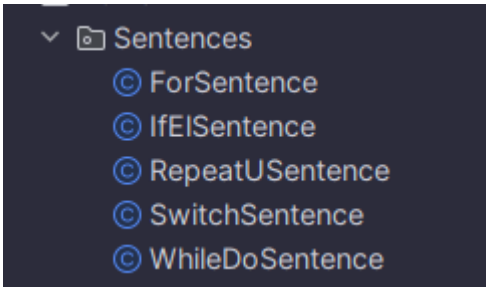


- **Paquete cup:** Contiene el archivo Parser.cup, el cual es responsable del análisis sintáctico. Se utiliza con CUP (Construction of Useful Parsers), una herramienta que genera analizadores sintácticos a partir de una gramática definida, permitiendo descomponer las estructuras del código Pascal en elementos reconocibles.
- **Paquete java.srbn.Backend:** Este paquete contiene la lógica y las estructuras del lado del backend para el procesamiento del lenguaje Pascal.

    - **Subpaquete Domain**: Define las estructuras fundamentales para representar el dominio del compilador, como sentencias y tablas de símbolos.
        - **Sentences**: Contiene clases que representan las distintas sentencias del lenguaje Pascal.
        - **Tables:** Representa las tablas necesarias, como la tabla de símbolos, para manejar las variables, funciones y otros elementos del lenguaje.
        - **TypeEnums:** Define enumeraciones que categorizan diferentes tipos de datos y operaciones en el lenguaje.
        - **Conditional**: maneja las estructuras condicionales, como sentencias if, else, y case.
        - **ErrorE y ErrorP:** Gestionan el manejo de errores tanto a nivel sintáctico como semántico.
        - **Function:** Maneja las funciones definidas en el lenguaje Pascal.
        - **Operation:** Contiene clases que representan operaciones aritméticas, lógicas, etc.
        - **SwitchCase:** Implementa la lógica de la estructura de control case.
        - **SymbT:** Tabla de símbolos, usada para almacenar y gestionar información sobre identificadores (variables, funciones) en el código.
        - **Type:** Representa los tipos de datos (integers, booleans, etc.) que se pueden manejar en Pascal.
    - **Subpaquete Management:** Maneja el control general de archivos y el flujo semántico.
        - **Files:** Gestión de la carga y manipulación de archivos fuente de Pascal.
        - **SemanticDriver:** Realiza las comprobaciones semánticas del código Pascal, asegurando que las operaciones y usos sean válidos según las reglas del lenguaje.
- **Paquete java.srbn.Frontend:** Aquí se encuentran las clases relacionadas con la interfaz gráfica de usuario y la interacción con el usuario.
    - **Subpaquete Drivers:** Podría contener controladores que gestionan la interacción entre el frontend y el backend, facilitando la comunicación entre ambas capas.
    - **Subpaquete Models:** Aquí se define la lógica relacionada con los datos del frontend, como la representación visual de estructuras del código Pascal.
    - Subpaquete Resources: Incluye recursos visuales y archivos relacionados con la interfaz.
    - **Details:** Posiblemente muestra detalles del código, errores o estructura del archivo Pascal.
    - **EditPanel y EditPanel.form:** Componente o panel para editar el código fuente.
    - **MainGui y MainGui.form:** Definen la interfaz principal de la aplicación, proporcionando una ventana de trabajo tipo IDE.
    - **Paquete jflex:** Contiene el archivo Lexer.flex, que define el análisis léxico usando JFlex, una herramienta de generación de analizadores léxicos. Este archivo se encarga de dividir el código Pascal en tokens (palabras clave,

identificadores, operadores, etc.), que luego serán procesados por el parser y el resto del backend.

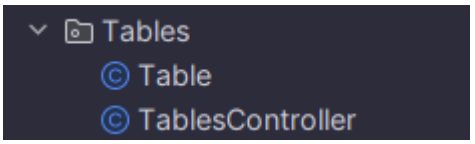
Paquete **Sentences:**



Este paquete contiene clases que representan las distintas sentencias que pueden aparecer en el código Pascal. Cada clase modela un tipo de estructura sintáctica específica.

- **ForSentence**: Representa la estructura de la sentencia for, que es utilizada para realizar bucles con una cantidad definida de iteraciones.
- **IfElSentence**: Modela la sentencia condicional if-else, que se utiliza para tomar decisiones en el flujo del programa.
- **RepeatUSentence**: Representa la sentencia repeat-until, que permite la ejecución repetida de un bloque de código hasta que se cumpla una condición.
- **SwitchSentence**: Modela la sentencia case (o switch), utilizada para evaluar una expresión y ejecutar el bloque de código correspondiente según el valor resultante.
- **WhileDoSentence**: Representa la estructura de control while-do, que ejecuta un bloque de código mientras se cumpla una condición booleana.

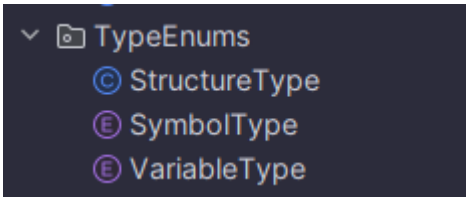
Paquete **Tables:**



Este paquete gestiona las tablas necesarias para el funcionamiento del compilador, como la tabla de símbolos y otras estructuras relacionadas con la semántica del lenguaje.

- **Table**: Clase principal que representa una tabla de símbolos u otra estructura de datos utilizada para almacenar información durante la compilación.
- **TablesController**: Controlador que gestiona y coordina las diferentes tablas, permitiendo su actualización y consulta durante el análisis del código Pascal.

Paquete **TypeEnums**:

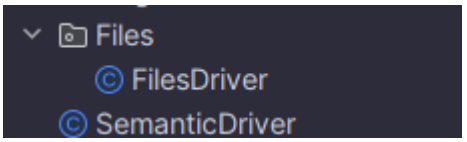


Este paquete define diferentes enumeraciones que categorizan los tipos de datos, estructuras y símbolos presentes en el lenguaje Pascal, y se utilizan a lo largo del proceso de compilación para garantizar la correcta tipificación y operación de los elementos.

- **StructureType**: Define los tipos de estructuras disponibles en el lenguaje, como arrays, registros, etc.

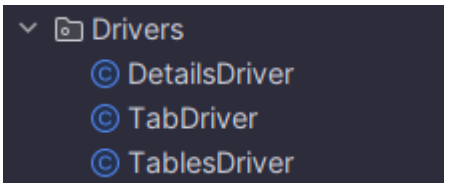
- **SymbolType**: Define los tipos de símbolos (variables, funciones, procedimientos, etc.) que pueden ser manejados en el lenguaje.
- **VariableType**: Enumera los tipos de variables soportadas, como integer, boolean, real, entre otros.

Paquete **Files:**
Aquí se gestionan los archivos que maneja el compilador, desde la carga de archivos fuente hasta su procesamiento en las diferentes etapas.



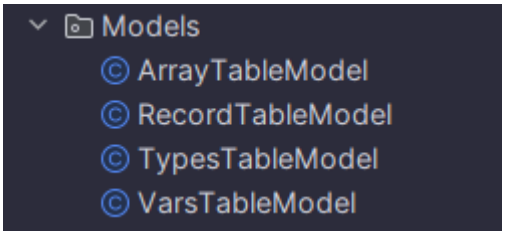
- **FilesDriver**: Responsable de controlar la gestión de los archivos, incluyendo la apertura, guardado, y manipulación de los archivos de código fuente en Pascal.
- **SemanticDriver**: Maneja el control y la verificación de las reglas semánticas del lenguaje, asegurando que las operaciones y el uso de variables sean coherentes con el tipo de datos y otros aspectos del lenguaje.

Paquete **Frontend.Drivers:**
Los controladores en este paquete permiten la interacción entre la interfaz gráfica del usuario y el backend del programa.



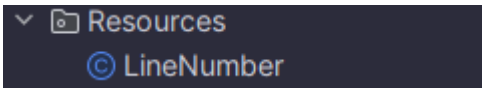
- **DetailsDriver**: Controlador que gestiona los detalles de visualización, como la presentación de información relevante sobre el código, errores, o resultados del análisis.
- **TabDriver**: Controlador que gestiona las pestañas o paneles en los que el usuario puede abrir y trabajar con múltiples archivos a la vez.
- **TablesDriver**: Gestiona la visualización y manipulación de las tablas de símbolos o estructuras relacionadas con el backend dentro del entorno visual.

Paquete **Frontend.Models:**
Define las clases relacionadas con la representación de datos y su visualización en la interfaz gráfica.



- **ArrayTableModel**: Modela la representación de arrays (arreglos) dentro de la interfaz.
- **RecordTableModel**: Gestiona la representación de registros, una estructura de datos compuesta por múltiples campos.
- **TypesTableModel**: Modela la representación de los diferentes tipos de datos dentro del entorno gráfico.
- **VarsTableModel**: Gestiona la representación de variables dentro de la tabla de símbolos y su visualización en la interfaz.

Paquete **Frontend.Resources:**
Contiene los recursos visuales y de apoyo que se utilizan en la interfaz gráfica.



- **LineNumber**: Clase que posiblemente gestiona la numeración de líneas dentro del editor de texto.
- **Details**: Gestiona la visualización de detalles adicionales sobre el archivo o el código que el usuario está editando.

Gramática

Asignación de variables

```
smtmtAs::=    ID:id ASSIGN opdata:dp
        |     ID:id arrayIndex:ai ASSIGN opdata:dp
        |     ID:id DOT ID:attrib ASSIGN opdata:dp
        ;

opdata ::=      underOperations:dp1 PLUS opdata:dp2
        |       underOperations:dp1 MINUS opdata:dp2
        |       underOperations:dp1
        ;

underOperations::=  dataPrim:dp1 MULT underOperations:dp2
        |       dataPrim:dp1 MOD underOperations:dp2
        |       dataPrim:dp1 DIV underOperations:dp2
        |       dataPrim:dp1
        ;

dataPrim   ::=    NUMBER:num
        |       DECIMAL:dec
        |       STRING_CONT:str
        |       CHAR_CONT:ch
        |       dataId:id
        ;

dataId      ::=  ID:id
        |     ID: id arrayIndex:ai
        ;
```

1. Regla smtmtAs

La regla smtmtAs define una asignación. Se pueden tener tres tipos de asignaciones:

- **ID:id ASSIGN opdata:dp**: Asigna el valor de una operación (o dato) a una variable identificada por ID.
- **ID:id arrayIndex:ai ASSIGN opdata:dp**: Asigna el valor de una operación a un índice específico de un arreglo.
- **ID:id DOT ID:attrib ASSIGN opdata:dp**: Asigna el valor de una operación a un atributo específico de un objeto o registro, referenciado por ID y DOT ID (similar al acceso de un campo de registro o atributo de un objeto en Pascal).

En todos los casos, el lado derecho de la asignación (después del operador ASSIGN) es evaluado por la regla opdata, que define las operaciones posibles.

2. Regla opdata

La regla opdata define cómo se pueden construir las expresiones aritméticas, las cuales pueden involucrar operadores como PLUS y MINUS. Esta regla puede tomar tres formas:

- **underOperations:dp1 PLUS opdata:dp2**: Suma el resultado de una operación a otra.
- **underOperations:dp1 MINUS opdata:dp2**: Resta el resultado de una operación de otra.
- **underOperations:dp1**: Usa directamente el resultado de una operación sin aplicar PLUS o MINUS.

Esta regla permite manejar tanto operaciones simples (como una variable o un número) como expresiones más complejas que involucren operadores aritméticos.

3. Regla underOperations

Esta regla permite manejar operaciones más básicas, como multiplicación, módulo y división. Las formas que toma son las siguientes:

- **dataPrim:dp1 MULT underOperations:dp2**: Multiplica el valor de una operación por otra.
- **dataPrim:dp1 MOD underOperations:dp2**: Calcula el módulo entre dos valores.
- **dataPrim:dp1 DIV underOperations:dp2**: Divide el valor de una operación por otra.
- **dataPrim:dp1**: Se utiliza el valor primario de la operación (número, variable, etc.) sin aplicar ninguna operación adicional.

La regla underOperations se encarga de manejar las operaciones a un nivel más básico que opdata, limitándose a multiplicaciones, divisiones y módulos.

4. Regla dataPrim

Esta regla define los datos primarios que pueden ser utilizados en una operación. Estos son los valores básicos que se pueden manipular:

- **NUMBER:num**: Un número entero.
- **DECIMAL:dec**: Un número decimal.
- **STRING_CONT:str**: Una cadena de texto.
- **CHAR_CONT:ch**: Un carácter.
- **dataId:id**: Un identificador, que podría ser una variable o un índice de un arreglo (definido por dataId).

La regla dataPrim define los elementos más simples que pueden formar parte de una operación, como números y cadenas.

5. Regla dataId

Esta regla maneja el acceso a variables y arreglos:

- **ID:id**: Un identificador simple (una variable).
- **ID:id arrayIndex:ai**: Un identificador con un índice de arreglo, es decir, el acceso a un elemento específico dentro de un arreglo.

Estructura del programa

```
prgmHeader  ::=     PROGRAM ID
                    TYPE typeDecl
                    CONST consDecl
                    VAR varDecl
                |   PROGRAM ID
                    CONST consDecl
                    VAR varDecl
                |   PROGRAM ID

                    TYPE typeDecl
                    VAR varDecl
                |   PROGRAM ID
                    TYPE typeDecl
                    CONST consDecl
                |   PROGRAM ID
                    VAR varDecl
                |   PROGRAM ID
                    CONST consDecl
                |   PROGRAM ID
                    TYPE typeDecl
                |   PROGRAM ID
                ;
```

Regla prgmHeader

Esta regla define cómo puede comenzar la cabecera de un programa Pascal. Se incluyen varias combinaciones posibles que reflejan las distintas secciones opcionales (tipos, constantes y variables) que puede tener un programa.

- **PROGRAM ID**: La cabecera básica de un programa, simplemente declara el nombre del programa (ID).
- **PROGRAM ID TYPE typeDecl VAR varDecl**: Declara un programa que incluye declaraciones de tipos y variables.
- **PROGRAM ID CONST consDecl VAR varDecl**: Declara un programa con constantes y variables.
- **PROGRAM ID TYPE typeDecl CONST consDecl**: Declara un programa que tiene tanto tipos como constantes.

Esta regla muestra que la declaración de PROGRAM puede combinarse de diferentes formas, permitiendo que un programa incluya solo algunas de las secciones opcionales: tipos, constantes, o variables.

Declaración de tipos

```
typeDeclDef ::=     idDecl:idList EQUAL varDeclType:type SEMICOLON
        |       idDecl:idList EQUAL RECORD recAttrbDecl:rvd END SEMICOLON
        |      error SEMICOLON
        ;

varDeclType ::=     varTypes:vt
        |       arrayDecl:arr
        |       lengthDecl:ld
        ;
arrayDecl   ::=     ARRAY OPENBSQUARE lengthDecl:ld
                    CLOSEBSQUARE OF varTypes:vt
        |       PACKED ARRAY OPENBSQUARE lengthDecl:ld
                    CLOSEBSQUARE OF varTypes:vt
        ;

lengthDecl  ::=     NUMBER:num
        |       NUMBER:num SUSPEND NUMBER:numf
        |       ID:id SUSPEND ID:idf
        |       ID:id SUSPEND NUMBER:numf
        |       NUMBER:num SUSPEND ID:idf
        ;

recAttrbDecl::=     ID:id COLON varTypes:vt SEMICOLON
        |       ID:id COLON varTypes:vt SEMICOLON recAttrbDecl:rvd
        |       ID:id COLON arrayDecl:arr SEMICOLON
        |       ID:id COLON arrayDecl:arr SEMICOLON recAttrbDecl:rvd
        |       error SEMICOLON
        ;
```

Regla typeDeclDef

Define las diferentes formas en que se puede declarar un tipo de dato. Las opciones incluyen:

- **idDecl:idList EQUAL varDeclType:type SEMICOLON**: Declara un tipo con una lista de identificadores (idList) que son asignados a un tipo específico (varDeclType), que puede ser un tipo simple o un tipo compuesto como un array o un registro.
- **idDecl:idList EQUAL RECORD recAttrbDecl:rvd END SEMICOLON**: Define un tipo de registro (estructura), que contiene atributos declarados dentro del bloque recAttrbDecl.
- **error SEMICOLON**: Maneja errores de sintaxis en la declaración de tipos.

Regla varDeclType

Esta regla define los posibles tipos que puede tener una variable. Las opciones incluyen:

- **varTypes:vt**: Un tipo básico (por ejemplo, integer, real).
- **arrayDecl:arr**: Un array, que se define mediante la regla arrayDecl.
- **lengthDecl:ld**: Una declaración relacionada con el tamaño o longitud de un tipo de dato, como los índices de un array.

Regla arrayDecl

Define la estructura de un array en Pascal, que puede ser declarado de dos formas:

- **ARRAY OPENBSQUARE lengthDecl CLOSEBSQUARE OF varTypes**: Un array simple, donde se especifica la longitud (lengthDecl) entre corchetes y el tipo de dato (varTypes).
- **PACKED ARRAY OPENBSQUARE lengthDecl CLOSEBSQUARE OF varTypes**: Un array "packed", que en Pascal indica que el array debe ser almacenado en el menor espacio posible en memoria.

Regla lengthDecl

Describe cómo se define la longitud o los índices de un array o una estructura de tipo. Las formas que puede tomar incluyen:

- **NUMBER:num**: Un solo número que representa la longitud o el índice.
- **NUMBER:num SUSPEND NUMBER:numf**: Un rango numérico (por ejemplo, 1..10).
- **ID:id SUSPEND ID:idf**: Un rango de identificadores que se utilizan para definir los límites del array o tipo.
- **ID:id SUSPEND NUMBER:numf**: Un rango donde el primer valor es un identificador y el segundo es un número.
- **NUMBER:num SUSPEND ID:idf**: Un rango donde el primer valor es un número y el segundo es un identificador.

Regla recAttrbDecl

Esta regla define los atributos de un registro (RECORD) en Pascal. Las opciones son:

- **ID:id COLON varTypes:vt SEMICOLON**: Define un atributo del registro con un tipo básico (por ejemplo, integer, char).
- **ID:id COLON varTypes:vt SEMICOLON recAttrbDecl:rvd**: Define múltiples atributos, donde después del primer atributo se añade otro declarado por recAttrbDecl.
- **ID:id COLON arrayDecl:arr SEMICOLON**: Define un atributo que es un array.
- **ID:id COLON arrayDecl:arr SEMICOLON recAttrbDecl:rvd**: Define múltiples atributos, donde el primero es un array y se añaden más atributos después.

- **error SEMICOLON**: Maneja errores de sintaxis en la declaración de atributos del registro.

Definicion de constantes

```
consDef    ::=    ID:id EQUAL NUMBER:num SEMICOLON
        |     ID:id EQUAL DECIMAL:dec SEMICOLON
        |     ID:id EQUAL STRING_CONT:str SEMICOLON
        ;
```

Regla consDef

Esta regla especifica cómo se deben declarar las **constantes** en el programa. Las constantes son valores fijos que no cambian durante la ejecución del programa. La sintaxis sigue el formato típico de Pascal, donde el identificador de la constante se asigna a un valor específico (número, decimal, cadena) y se termina con un punto y coma.

Las formas posibles son:

ID:id EQUAL NUMBER:num SEMICOLON: Declara una constante que es un número entero.
ID:id EQUAL DECIMAL:dec SEMICOLON: Declara una constante que es un número decimal.
ID:id EQUAL STRING_CONT:str SEMICOLON: Declara una constante que es una cadena de texto

En cada caso, el identificador (ID) es el nombre de la constante, y el valor se asigna usando el símbolo EQUAL (=).

Acceso a arreglos

```
arrayIndex  ::=    OPENBSQUARE NUMBER:num CLOSEBSQUARE
        |     OPENBSQUARE ID:id CLOSEBSQUARE
        ;
```

R**egla arrayIndex**

Esta regla define cómo se puede acceder a un **índice de un arreglo**. Los arreglos en Pascal se acceden especificando un índice dentro de corchetes ([]). La regla cubre dos formas posibles de especificar el índice:

OPENBSQUARE NUMBER:num CLOSEBSQUARE: El índice es un número fijo. Esto accede a un elemento específico del arreglo usando un número.
OPENBSQUARE ID:id CLOSEBSQUARE: El índice es una variable, lo que permite acceder a un elemento del arreglo usando el valor almacenado en una variable.

Esto permite acceder a elementos específicos de un arreglo, ya sea utilizando un número fijo o una variable que contiene el índice.