TRABAJO PRÁCTICO COMPILADOR

CONSIDERACIONES GENERALES

Es necesario cumplir con las siguientes consideraciones para evaluar el TP.

1. Cada grupo deberá desarrollar el compilador teniendo en cuenta:
    - Todos los temas comunes (Ver ANEXO TEMAS)
    - El tema especial asignado al grupo.
2. Se fijarán puntos de control con fechas y consignas determinadas

PRIMERA ENTREGA

OBJETIVO: Realizar un analizador lexicográfico y un analizador sintáctico utilizando las herramientas propuestas por la cátedra las cuales deberán operar en conjunto. La aplicación realizada debe mostrar una interfaz gráfica que pueda utilizarse como interfaz del compilador, en la cual se debe poder ingresar código escrito en el lenguaje a compilar, cargar un archivo de código y editarlo, compilar el contenido del programa ingresado y mostrar por pantalla un texto aclaratorio identificando las reglas sintácticas que va analizando el parser. Las impresiones deben ser claras.

El material a entregar será:

- El proyecto utilizando las herramientas seleccionadas para la implementación del analizador léxico y para el analizador sintáctico.
- Un archivo de pruebas generales que se llamará **prueba.txt** que incluya pruebas exhaustivas de los temas comunes y los temas especiales asignados a cada grupo.
- El proyecto deberá generar un archivo con la tabla de símbolos **ts.txt** donde figuren los nombres de todos los identificadores detectados junto con los tipos correspondientes.
- Un archivo ejecutable que se deberá llamar **Compilador** (por ejemplo Compilador.jar que permita ejecutar la aplicación mediante java runtime enviroment, o el archivo .py de punto de entrada que permita ejecutarlo mediante un intérprete de Python).

La entrega deberá subirse a las tareas que los docentes habiliten para tal fin en la plataforma ED.

Fecha de entrega: 22/04/2025

SEGUNDA ENTREGA

OBJETIVO: Realizar un generador de código intermedio que deberá procesar un archivo de entrada (prueba.txt) y devolver el árbol sintáctico del mismo, junto con la tabla de símbolos. Generar además código IR de LLVM a partir del árbol sintáctico mencionado previamente. Mediante una interfaz se deberá poder cargar el archivo de entrada (prueba.txt), modificarlo si resultara necesario, compilarlo y ejecutarlo mostrando por pantalla el resultado de la ejecución. Se debe incorporar un último ítem al menú que muestre el código IR generado por LLVM. Se deberá además realizar los chequeos de tipos y conversiones implícitas correspondientes.

El material a entregar será:

- El proyecto utilizando las herramientas seleccionadas para la implementación del analizador léxico y para el analizador sintáctico.
- Un archivo de pruebas generales que se llamará **prueba.txt** que incluya pruebas exhaustivas de los temas comunes y los temas especiales asignados a cada grupo.

- El proyecto deberá generar un archivo con la tabla de símbolos **ts.txt** donde figuren los nombres de todos los identificadores detectados junto con los tipos correspondientes.
- El compilador deberá realizar el chequeo de tipos y conversiones implícitas correspondientes.
- El compilador deberá generar un archivo PNG con la estructura del árbol sintáctico y un archivo .DOT con la definición en lenguaje dot de graphviz del árbol generado, ambos generados generado a partir del código en el archivo **prueba.txt**
- El compilador deberá generar un archivo LL con el código fuente legible en el lenguaje IR de LLVM y un archivo ejecutable correspondiente al código fuente anterior.
- Un archivo ejecutable que se deberá llamar **Compilador** (por ejemplo Compilador.jar que permita ejecutar la aplicación mediante java runtime enviroment, o el archivo .py de punto de entrada que permita ejecutarlo mediante un intérprete de Python).

La entrega deberá subirse a las tareas que los docentes habiliten para tal fin en la plataforma ED.

Fecha de entrega: 28/05/2025

ANEXO TEMAS

TEMAS COMUNES

OPERADORES ARITMÉTICOS

Se deberán admitir las operaciones de suma (a + b), resta (a - b), multiplicación (a * b) y división (a / b) junto con la de valor opuesto (-a). Se podrán agregar otras operaciones adicionales que se consideren apropiadas. Se cumplirá con la precedencia y asociatividad usual de las operaciones matemáticas, pudiendo igualmente usarse los paréntesis para modificarla.

OPERADORES DE COMPARACIÓN

Se podrán comparar dos expresiones por igualdad (a == b), desigualdad (a != b), mayor (a > b), mayor o igual (a >= b), menor (a < b) o menor o igual (a <= b). Todos los operadores de comparación poseen la misma precedencia y no son asociativos.

OPERADORES LÓGICOS

Se deberán admitir las operaciones de conjunción (a and b), disyunción (a or b) y negación (not a). Se cumplirá con la precedencia y asociatividad usual de dichos operadores (negación, conjunción, disyunción), pudiendo usarse los paréntesis para modificarla.

EXPRESIONES

Una expresión puede combinar operaciones aritméticas entre constantes y variables numéricas o uso a funciones que retornan un valor numérico compatible.

Una expresión también puede incluir operadores lógicos y relacionales aplicados a constantes, variables o usos de funciones que retornan un valor apropiado para ese contexto.

La precedencia y asociatividad de los operadores se indica en la siguiente tabla:

Operador	Asociatividad
or	Izquierda
and	Izquierda
not	Izquierda
== != > >= < <=	No es posible asociar
+ -	Izquierda
*** /**	Izquierda
- (unario)	Izquierda

SENTENCIAS DE ITERACIÓN

El lenguaje soportará dos tipos de ciclos, LOOP WHEN y BACKWARD_LOOP WHEN. Estos estarán compuestos por las palabras reservadas LOOP WHEN o BACKWARD_LOOP WHEN según corresponda, seguidas por una expresión lógica, la palabra reservada THEN, un conjunto de instrucciones y finalizando con la palabra reservada END_LOOP. La diferencia entre ambos ciclos radica en que, el primero evalúa la condición y de cumplirse ejecuta el conjunto de instrucciones de inicio a fin para luego evaluar la condición

nuevamente. Mientras que el segundo ejecuta las instrucciones de la última a la primera para luego evaluar la condición y verificar si repite el ciclo.

Los ciclos deberán soportar las instrucciones especiales BREAK y CONTINUE, para salir del ciclo y continuar con la siguiente iteración respectivamente. Dichas instrucciones NO deberán ser soportadas fuera del contexto de una sentencia de iteración y deberán arrojar un error de compilación.

SENTENCIAS DE SELECCIÓN

Las instrucciones de selección estarán compuestas por la palabras reservadas CONDITION o BACKWARD_CONDITION seguidas de una expresión lógica, luego de la palabra reservada THEN a continuación un conjunto de sentencias y finalmente la palabra reservada END. Opcionalmente podrán tener la palabra reservada ELSE o ELSE_BACKWARD (en cuyo caso se omite el END mencionado previamente), seguidas de otro conjunto de sentencias y finalmente la palabra reservada END.

TIPO DE DATOS

Se soportan los tipos booleano (boolean), número entero (integer), número representado en punto flotante (float) y arreglos de flotantes (float_array) .

Para las constantes booleanas literales se utilizan los valores **true** y **false** para representar verdadero y falso respectivamente.

Las constantes flotantes literales pueden definirse con o sin parte decimal. Si tienen parte decimal, se asumen de punto flotante. En caso contrario, se asumen enteras. Para las constantes de valores reales deberá utilizarse obligatoriamente el punto (.) como separador decimal. Algunos ejemplos de constantes de valores en punto flotante:

99999.99 **99.** **.9999**

Para el caso de los arreglos, tendrán N elementos flotantes, separados por coma, entre corchetes, algunos ejemplos de esto son los siguientes:

[-8., 7.3, -10.] **[9.5,-7.4]** **[0., 9., 15., 0.15]**

El tamaño del arreglo se especificará al momento de la declaración. Se inicializará con todos los valores en 0 por defecto. A un arreglo podrá asignarse otra variable de tipo arreglo o un arreglo literal (en ambos casos siempre que sean del mismo tamaño), o también asignarse una expresión a una posición específica del mismo por ejemplo (el indice debe ser una expresión entera):

mi_arreglo[7] = 15 **mi_arreglo[5+x] = 22 + y**

Por otro lado los arrays deberán soportar las operaciones **all()** y **any()**. Ambas recibirán una condición, una expresión numérica y un array y deberán chequear si todos o alguno de los elementos del array respectivamente cumplen con la condición y retornar true o false según corresponda.

Las constantes de cadenas de caracteres se representan como un texto delimitado por comillas dobles (“"”), de la forma "xxx xxx xxxx". Se acepta que dentro de la cadena puedan escribirse comillas dobles siempre que se las anteponga con una barra invertida (“\"”). De forma similar, la combinación “\n” se interpretará como un carácter de nueva línea y la combinación “\t”, como carácter de tabulación. La barra invertida, por usarse como carácter de escape dentro de la cadena, deberá siempre ser escrita ante poniéndole otra barra invertida (quedando finalmente como “\\”) para que sea interpretada como un único carácter “\”. Algunos ejemplos de cadenas literales:

"Esta es una cadena de caracteres."

"Primera línea\nSegunda \"línea\"\nLa \\última/ línea"

Las constantes deben ser reconocidas y validadas en el analizador léxico, de acuerdo a su tipo.

VARIABLES

Los nombres de las variables definidas en este lenguaje deberán cumplir la restricción de comenzar con una letra, siguiendo con cero o más letras, dígitos o guiones bajos ('_'). Las letras soportadas serán aquellas que el estándar Unicode considera como tal, pudiendo ser tanto mayúsculas como minúsculas. Se distinguen las diferencias entre mayúsculas y minúsculas. Ejemplos:

i monto_total aux_1 año función

Las variables no guardan su valor en la tabla de símbolos.

Las asignaciones deben ser permitidas, solo en los casos en los que los tipos son compatibles, caso contrario deberá desplegarse un error. Las variables y constantes integer y float deben ser interoperables entre sí y el resultado de la operación siempre deberá ser un float, mientras que las variables bool serán incompatibles con el resto.

A una variable de tipo arreglo, puede asignarse una constante o variable de tipo arreglo siempre y cuando tenga las mismas dimensiones. Los arreglos no pueden utilizarse en operaciones aritmetico lógicas, salvo en el caso de acceso a componentes individuales del arreglo o en la utilización de operaciones all y any. Ejemplos:

x = a + c[5] a > c and any(5, >, b)

ASIGNACIONES

Las asignaciones en este lenguaje serán simples, de la forma A:=B, siendo A una variable previamente declarada y B una expresión aritmético-lógica del mismo tipo o de un tipo compatible. Para determinar qué asignaciones deberán permitirse en el lenguaje, según los tipos de A y B, deberá tenerse en cuenta la siguiente tabla:

A ↓ B →	**integer**	**float**	**bool**	**arreglo**
integer	permitido	no permitido	no permitido	no permitido
float	permitido	permitido	no permitido	no permitido
bool	no permitido	no permitido	permitido	no permitido
arreglo	permitido (*)	permitido (*)	no permitido	permitido (**)

(*) Permitido sólo para los casos en los que se asigna una posición específica del arreglo.

(**) Siempre que sean del mismo tamaño

COMENTARIOS

Los comentarios multilínea deberán permitir múltiples niveles de anidamiento, asegurando que cada apertura se corresponda con su cierre correspondiente. El primer nivel deberá ser **(*** y ***)**, el segundo nivel **[*** y ***]**, y el tercer nivel **{*** y ***}**. Luego del tercer nivel vuelve a iniciarse desde el primer nivel.

También es posible definir un comentario con "$", indicando que el comentario termina al final de la línea. Ejemplos:

..... $ .....$.....$..... (válido)

..... (* ..[*..*].....[*.....*].....*) ..... (válido)

..... (* . [*..{*....(*..[*...*]...*)....*}....*]. *) ..... (válido)

..... (* .....*).....*) ..... (inválido, no están balanceados)

ENTRADA Y SALIDA

Las salidas se implementarán mediante la instrucción DISPLAY. Esta deberá permitir imprimir expresiones (enteras, booleanas, float y array) y constantes string. Como se muestra en los siguientes ejemplos:

DISPLAY("ewr") (* donde "ewr" debe ser una cte literal string *)

DISPLAY(5.5 + aux)

DISPLAY(7 <= mi_var)

Por otro lado, la lectura de valores por consola se realizará mediante las instrucciones **INPUT_INT() INPUT_FLOAT() INPUT_BOOL() e INPUT_ARRAY()**.

DECLARACIONES

Todas las variables deberán ser declaradas dentro de un bloque especial para ese fin, delimitado por las palabras reservadas DECLARE.SECTION y ENDDECLARE.SECTION, siguiendo el siguiente formato:

```
DECLARE.SECTION
        Línea_de_Declaración_de_Tipos
ENDDECLARE.SECTION
```

Cada Línea_de_Declaración_de_Tipos tendrá la forma: Tipo : Lista de Variables

La Lista de Variables debe ser una lista de variables separadas por comas. Pueden existir varias líneas de declaración de tipos.

Ejemplos de formato:

```
DECLARE.SECTION
        INT : a, b , c
        FLOAT : d, e
        ARRAY[5]: f
ENDDECLARE.SECTION
```

PROGRAMA

Todas las sentencias del programa deberán ser declaradas dentro de un bloque especial para ese fin, delimitado por las palabras reservadas PROGRAM.SECTION y ENDPROGRAM.SECTION, siguiendo el siguiente formato:

```
PROGRAM.SECTION
        Lista_de_Sentencias
ENDPROGRAM.SECTION
```

Nota: la zona de declaración de variables deberá ser previa a la sección del programa.

TABLA DE SÍMBOLOS

La tabla de símbolos tiene la capacidad de guardar las variables y constantes con sus atributos. Los atributos portan información necesaria para operar con constantes, variables.

Ejemplo 1ra ENTREGA (agregando tipos de datos)

NOMBRE	TOKEN	TIPO	VALOR	LONG
a1	ID	Float	_	_
b1	ID	Integer	_	_
	CTE_STR	_	hola	4
	CTE_STR	_	mundo	5

TEMAS ESPECIALES

ECONTIGUOUS

El comando cuenta con lista de variables y busca dos o más elementos de la lista que sean contiguos e iguales. Retorna la posición del primer elemento que es parte de alguna ocurrencia. Retorna -1 si la lista no cumple con la condición.
Por ejemplo
var a1=10, a2=20, a3=30, a4=40, a5=50, a6=60 //definición y asignación de
// variables (necesarias para el ejemplo)
b= ectg(a1,a2,a3,a3,a5) // b=3 posición de primer elem. de la ocurrencia 30
b= ectg (a1,a2,a3,a4,a5,a6) // b=-1 no existe ocurrencia
b= ectg (a1,a1,a1,a1) // b=1 posición de primer elem. de la ocurrencia 10
b= ectg (a1,a2,a2,a3,a3,a2) // b=2 posición de primer elem. de la ocurrencia 20
b= ectg (a1) // b=-1 no existe ocurrencia, lista de un solo elemento
b= ectg (a1,a2,a3) // b=-1 no existe ocurrencia

MEDIANA

La sentencia recibe una lista de expresiones y retorna su mediana. Si la lista está vacía, se mostrará un mensaje "La lista está vacía". El valor retornado debe ser un flotante. La cantidad de elementos de la lista deberá ser impar, en caso de que sea par se deberá imprimir el mensaje Ejemplos:
med = mediana([1, 2, 3, 4, 5]) # retorna 3
med = mediana([4, 7, 1, 3, 2, 5, 6]) # retorna 4
med = mediana([]) # La lista está vacía, retorna 0
med = mediana([1,2,3,4]) # imprime la cantidad de elementos debe ser impar, retorna 0.

MÚLTIPLOS

La sentencia permite encontrar los múltiplos de una variable o constante pivot entera dentro de una lista de expresiones enteras. El elemento pivot deberá ser mayor a uno y será un valor constante o una variable siempre entera, si no cumpliese con la validación deberá mostrar un mensaje “El valor debe ser >1”.
Si no se encuentran múltiplos del pivot en la lista, se emitirá el mensaje “No se encontraron múltiplos”.La lista de expresiones podría ser vacía en cuyo caso se emitirá un mensaje “La lista está vacía”.En los casos en los que no pueda efectuarse la operación el valor retornado por la función será una lista vacía.
Por ejemplo:
resul=multiplos(3;[10,20,30,40,5,4]) Los múltiplos de 3 son: [30], la función retorna un 1.
resul=multiplos(5;[2,2,2,4]) No se encontraron múltiplos, la función retorna un 0.
resul=multiplos(1;[2,1,1,4]) “El valor debe ser >1”, la función retorna un 0.
resul=multiplos(2;[10,20,30,40,50,40]) Los múltiplos de 2 son: [10, 20, 30, 40, 50], la función retorna un 6.
resul=multiplos(4;[]) La lista está vacía, la función retorna un 0.

PROMEDIO_PONDERADO

La sentencia calcula el promedio ponderado de una lista de expresiones enteras o flotantes con sus respectivos pesos. Si la lista de valores o pesos está vacía, se muestra el mensaje "La lista está vacía". Si las listas no tienen la misma cantidad de elementos, se muestra el mensaje "Las listas deben tener la misma longitud".
Ejemplos:
prom = promedio_ponderado([10, 20, 30], [0.2, 0.3, 0.5]) # retorna 23.0
prom = promedio_ponderado([], []) # "La lista está vacía", retorna 0.0
prom = promedio_ponderado([10, 20], [0.3]) # "Las listas deben tener la misma longitud", retorna 0.0

SUMA ACUMULATIVA

La sentencia recibe una expresión entera y una lista de expresiones enteras o flotantes y retorna la suma acumulada de los elementos hasta la posición indicada. Si la lista está vacía, se muestra el mensaje "La lista está vacía".
Ejemplos:
suma = suma_cumulativa(2, [1, 2, 3, 4]) # retorna 6
suma = suma_cumulativa(3, []) # "La lista está vacía", retorna 0
suma = suma_cumulativa(2, [5, -2, 4]) # retorna 7

MODA

La sentencia calcula la moda de una lista de números, es decir, el valor que más se repite. Si hay múltiples valores con la misma frecuencia máxima, devuelve el primero de ellos. Si la lista está vacía, se muestra el mensaje "La lista está vacía".
Ejemplos:
m = moda([1, 2, 2, 3, 3, 3, 4]) # retorna 3
m = moda([10, 20, 10, 20]) # retorna 10
m = moda([]) # "La lista está vacía", retorna -1

RANGO_NUMERICO

La sentencia determina el rango de una lista de valores numéricos, es decir, la diferencia entre el valor máximo y el valor mínimo. Si la lista está vacía, se muestra el mensaje "La lista está vacía".
Ejemplos:
r = rango_numerico([5, 2, 8, 10, 3]) # retorna 8 (10 - 2)
r = rango_numerico([1]) # retorna 0
r = rango_numerico([]) # "La lista está vacía", retorna 0

VALOR_MAS_CERCANO

La sentencia recibe una lista de valores numéricos y un valor de referencia, y retorna el número de la lista que sea más cercano al valor dado. Si hay varios valores a la misma distancia, se retorna el primero de ellos. Si la lista está vacía, se muestra el mensaje "La lista está vacía".
Ejemplos:
v = valor_mas_cercano(5, [1, 2, 8, 6, 4]) # retorna 6
v = valor_mas_cercano(10, [15, 12, 8, 9]) # retorna 9
v = valor_mas_cercano(3, []) # "La lista está vacía", retorna 0