Proyecto LogoTec

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
CE1108 –Compiladores e Intérpretes
II Semestre 2025

Objetivo general

- ➔ Implementar un compilador por medio de la técnica y el arte.

Objetivos específicos

- ➔ Implementación de un compilador completo para un lenguaje de programación usando las herramientas de validación necesarias.
- ➔ Creación del FrontEnd y el BackEnd de un compilador totalmente funcional.
- ➔ Coordinación entre el objeto generado del compilador con un dispositivo de hardware

Datos Generales

- ➔ El valor del Proyecto: 40%
- ➔ Nombre código: LogoTec
- ➔ El proyecto debe ser implementado por grupos de máximo de 4 personas.
- ➔ La fecha de entrega del proyecto es de **20/Noviembre/2025**
- ➔ Cualquier indicio de copia de proyectos será calificado con una nota de 0 y será procesado de acuerdo con el reglamento.

Marco Conceptual

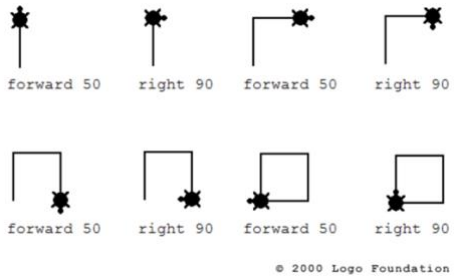
Logo

A finales de los años 60 el matemático Seymour Papert y Wallace Feurzeig se conocieron, el primero era el cofundador del Laboratorio de Inteligencia Artificial del MIT, mientras que el segundo era el director de Bolt, Beranek y Newman, importante empresa que tuvo un papel destacado en la creación de tecnologías como el email, las llamadas VoiceIP y los primeros routers. Los dos, junto a la científica y experta en Lisp, Cynthia Solomon, crearon la primera versión de Logo en 1967.

Seymour Papert creó Logo como lenguaje para los pequeños, con la idea que pudieran **mover el robot tortuga utilizando instrucciones simples**. Una herramienta de aprendizaje basada en la modularidad, la extensión, la interactividad y la flexibilidad. Conceptos importantes en el mundo de la programación y que permitieran a los niños tener una base para después enfrentarse a ideas matemáticas más complejas.

El motivo de la existencia de la tortuga es debido a que necesitaban un apoyo para interpretar los distintos comandos. En aquel momento no había ni siquiera monitores para mostrar la

información, por lo que era difícil relacionar los comandos con figuras geométricas. El **primer uso de Logo en las escuelas se realizó en 1968 en Muzzey Jr High, Lexington MA**, pero no fue hasta el año siguiente cuando se introdujo el robot en forma de tortuga y hasta 1970 no se utilizaron en las escuelas.

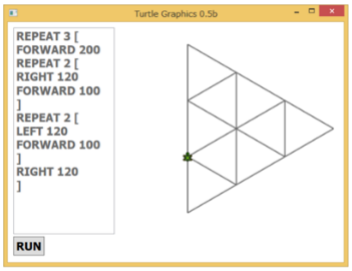


"Logo" no es un acrónimo. Su nombre deriva del griego y significa "pensamiento". La idea proviene de Feurzeig, quien quería distinguir a su lenguaje de programación de otros basadas en números sin gráficos ni lógica.

La expansión de Logo llegó de la mano de la expansión de los ordenadores personales. El grupo del MIT desarrolló para dos máquinas: el **Apple II y el Texas Instruments TI 99/4**. Después llegaron varias versiones comerciales que contribuyeron a acelerar su adopción en las escuelas. En los 80 hubo una prueba piloto patrocinada por el MIT y Texas Instrument donde se distribuyeron hasta cincuenta ordenadores en escuelas de Dallas y Texas. Al mismo tiempo, seis escuelas públicas de New York también recibieron ordenadores para enseñar Logo. En 1981, Papert visitó Australia y recibió una calurosa bienvenida rodeado de robots tortuga, según quedó constancia.

El creciente éxito de Logo llevó a sus creadores en 1980 a formar una nueva compañía, **Logo Computer Systems Inc (LCSI).** La empresa desarrolló Apple Logo, Logo Writer, amplió su disponibilidad comercial y hasta lanzaron un libro, Mindstorms, que fue muy influyente para miles de profesores que decidieron aprovechar sus creativas ideas en clase.

10 Arriba, 20 derecha: así se usaba Logo



Uno de los motivos del éxito de Logo fue que era un lenguaje de programación muy sencillo, tanto a nivel de comandos como en las ideas que transmitía. El objetivo inicial de Logo era controlar listas, pero con la llegada del robot tortuga pronto se pasó a una secuencia de instrucciones para moverse de un lado a otro. La geometría dotó a Logo de un recurso flexible y fácil de entender.

Logo tiene una sintaxis muy sencilla basada en cuatro comandos, que representan las direcciones: **'Forward'**, **'back'**, **'left'**, **'right'**. Estos comandos van acompañados de un valor que no deja de ser el módulo de distancia recorrido. Si por ejemplo se escribe "**FORWARD 100**" significa que la tortuga irá hacia adelante cien unidades. A esta base se le deben añadir comandos como **'repeat'** donde se especificaba entre corchetes la cantidad de veces que debía repetirse un movimiento.

Mediante la combinación de estos comandos, con Logo se podían dibujar formas geométricas como un triángulo o hasta espirales. A medida que fue avanzando el lenguaje, las formas fáciles de realizar iban siendo más complejas.

La sencillez fue sin duda el motivo por el que Logo se expandió como lenguaje para niños pero también hubo otro factor que contribuyó a su uso en las escuelas de diversos países. Logo es un lenguaje de alto nivel y muy estructurado pero además era de los pocos con interpretes y compiladores en español, muchos de ellos de libre distribución.

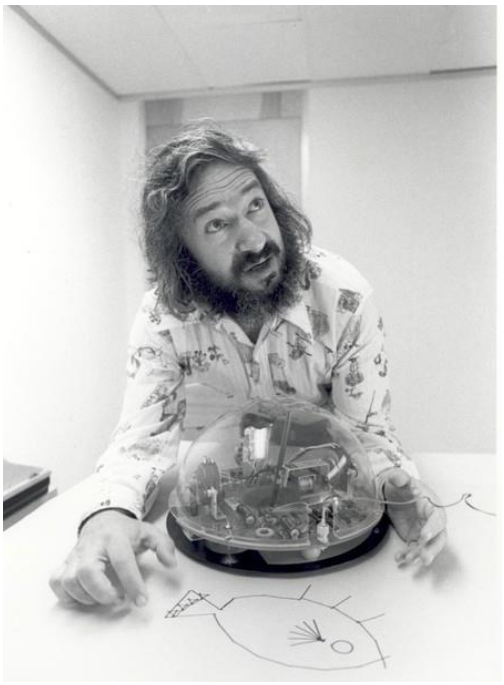
LEGO, nuevas versiones y el éxito posterior

Durante los últimos años de la década de los 80, Logo fue expandiéndose por escuelas de Costa Rica, Latinoamérica, Europa y Japón gracias en parte a la simplificada interfaz de LogoWriter. Pero en 1993, LCSI creó **MicroWorlds Logo**. Fue una implementación muy importante a nivel comercial ya que permitía controlar diversas tortugas a la vez, tenía un aspecto más moderno e incluso se llegó a utilizar para desarrollar juegos, simulaciones y proyectos multimedia más avanzados. Una nueva ola que sentó las bases de todas las versiones más modernas que han llegado hasta nuestros días.

La influencia de Logo también llegó a una reconocida marca de nombre parecido. **LEGO lanzó sus bricks programables**, un proyecto en que los pequeños podían juntar piezas de LEGO y mediante los comandos de Logo y un ordenador conectado, hacerlas mover de un lado a otro. LEGO TC (Technic Control) Logo llegó en 1988 y posteriormente se acabaría convirtiendo en lo que hoy en día son los LEGO Mindstorms.

Logo contribuyó a que varias generaciones de niños descubriesen conceptos básicos de programación. Un conocimiento que tanto pequeños como mayores pueden aprender y donde afortunadamente cada vez hay más recursos disponibles a nuestro alcance.

Tomado de:
Pérez, Enriquez. "La tortuga que nos enseñó a programar: la historia de Logo, el primer lenguaje de programación diseñado para niños"
https://www.xataka.com/historia-tecnologica/tortuga-que-nos-enseno-a-programar-historia-logo-primer-lenguaje-programacion-disenado-para-ninos



https://youtu.be/I79dcvgoUXc

Descripción del problema

La característica más conocida de Logo es la **tortuga**, un cursor gráfico usado para crear dibujos. Aún los niños pequeños rápidamente aprenden a mover y girar la tortuga usando comandos intuitivos y fáciles de recordar. Por ejemplo, al escribir **avanza 50** la tortuga se mueve hacia adelante 50 posiciones. Al escribir **giraderecha 90** la tortuga gira (en el sentido de las agujas del reloj) 90 grados. Mediante la combinación de estos comandos es fácil dibujar un cuadrado.

```
avanza 50 (También se puede abreviar avanza como "av" )
giraderecha 90 (Se puede abreviar giraderecha como "gd" )
avanza 50
giraderecha 90
avanza 50
giraderecha 90
avanza 50
giraderecha 90
```

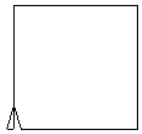
Cuando varios comandos forman un patrón estos pueden ser combinados usando **repite** . Aquí está el mismo cuadrado dibujado usando una sola línea de instrucciones:

```
repite 4 [avanza 50 giraderecha 90]
```

Ya que Logo es un lenguaje extensible, se pueden añadir nuevos comandos mediante la creación de pequeños programas o conjuntos de instrucciones llamados **procedimientos** . Por ejemplo, aquí está un procedimiento para dibujar el cuadrado:

```
para cuadrado
repite 4 [avanza 50 giraderecha 90]
fin

cuadrado
```



Ahora, para dibujar un cuadrado, simplemente se escribe **cuadrado** . Se podrá usar la palabra **cuadrado** como se utiliza cualquier otro comando de Logo, aún para incluirla en otros procedimientos. Por ejemplo, para dibujar una bandera:

```
avanza 60 cuadrado retrocede 60
```

Se podría combinar un cuadrado y un triángulo para construir una casa (y escribir **casa** para dibujarla):

```
para casa
cuadrado
avanza 50
giraderecha 90
triangulo
fin

para triangulo
repite 3 [avanza 50 giraizquierda 120]
fin

casa
```

Es posible utilizar un nombre para representar el tamaño de un cuadrado:

```
para cuad :tamaño
repite 4 [avanza :tamaño giraderecha 90]
fin
```

Ahora se puede dibujar cuadrados de diferentes tamaños escribiendo:

```
cuad 10 cuad 20 cuad 30 ...
```

Como se puede ver, con el solo uso de los gráficos de tortuga, se puede progresar del dibujo de figuras simples con comandos de fácil aprendizaje, a crear figuras complejas usando técnicas de programación bastante sofisticadas. (Tomado de http://neoparaiso.com/logo/)

Para efectos del proyecto de "Compiladores e Intérpretes" del TEC, se desea desarrollen desde 0 (*desde el inicio*) el lenguaje "LogoTec", el cual tendrá las siguientes características:

- Para efectos de la validación del FrontEnd pueden utilizar herramientas adecuadas para tales efectos como Lex y Yacc, o Antlr, pero la Gramatica Libre de Contexto y su implementación debe ser realizada exclusivamente por el equipo de trabajo**.** Esto quiere decir que el compilador debe "reaccionar" a posibles errores.
- Se debe realizar la implementación del BackEnd utilizando las herramientas propuestas por el profesor y vistas en clase. El backend debe generar un **archivo objeto** producto de la compilación del código fuente sin error. Este archivo objeto podrá ejecutarse cuantas veces se desee.
- Se debe implementar un **dispositivo de hardware** que "simule" a la tortuga y que debe tener como mínimo las siguientes características:
    - Debe comunicarse vía inalámbrica con el equipo en donde reside el compilador
    - Debe contener algún tipo de elemento para dibujar sobre alguna superficie, cuando el dispositivo se mueve.
    - Debe tener movimientos de acuerdo con el enunciado del lenguaje de programación (adelante, atrás, etc.)
    - Debe realizar los movimientos programados en el archivo objeto producto de la compilación.
    - Se activa al ejecutar el archivo objeto producto de la compilación, y realizará las funciones programadas cada vez que se ejecute el archivo objeto. Se validará esto de forma estricta (más de una vez la ejecución).
- Por tanto se espera que el proyecto sea **completamente funcional**, y que permita apreciar "graficación" un tanto compleja.
- Se debe presentar por parte del grupo, un **programa con una imagen compleja** (desde el punto de vista de arte), la cual será evaluada en la revisión.
- Se adjunta a la documentación una **serie de videos** que ayudarán al estudiante a "sumergirse" en el lenguaje original, y que le dará los insumos necesarios para poner en práctica el nuevo compilador y la funcionalidad del gráfico.
- Todo el proyecto debe ser implementado por el grupo de estudiantes, en caso de querer la utilización de bibliotecas, módulos u otras facilidades (no mencionadas anteriormente) y que disminuiría la carga de trabajo, se debe consultar previamente con el profesor y obtener su aprobación en dicho uso, en caso contrario se tomará como trabajo inconcluso.

- Se debe generar una interfaz en la computadora **agradable al usuario**, y mínimo similar al juego de Logo, en cuanto a línea de procesamiento de instrucciones y carga y edición de programas (procedimientos, etc).

Compilador de LogoTec

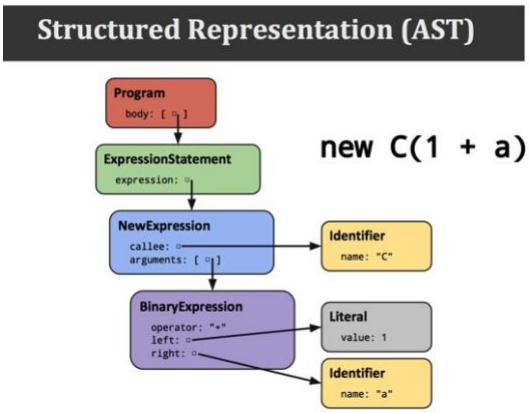
Se debe crear un compilador que permita la ejecución de las instrucciones necesarias para llevar a cabo cada uno de los gráficos.

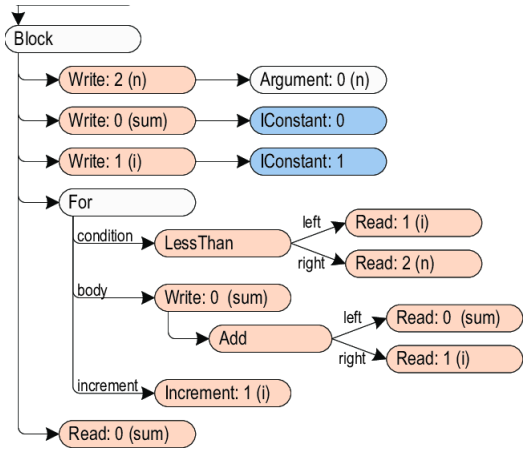
El compilador por implementar debe llevar a cabo todas las etapas de un compilador, y mostrar los correspondientes errores léxicos, sintácticos y semánticos.

Para hacer lo anterior, deberán construir una gramática que soporte la funcionalidad que más adelante se indique. El lenguaje construido debe ser flexible desde el punto de vista, que pueda permitir crear distintos programas que parametrizará gráficos.

En lo que respecta a la implementación del Compilador:

- Se solicitará en el momento de la revisión, una explicación detallada de la Gramática generada. Se debe demostrar un dominio suficiente sobre la "lógica" colocada en la gramática.
- Dentro de la interfaz gráfica solicitada como resultado del proyecto, se requiere tener una facilidad que muestre gráficamente el **Abstract Syntaxis Tree** (AST) resultante del código fuente usado como insumo del compilador, esto es el IR del compilador que es insumo para la etapa del backend. Parte importante de la evaluación del proyecto, será la visualización correcta del AST sobre el programa dado.



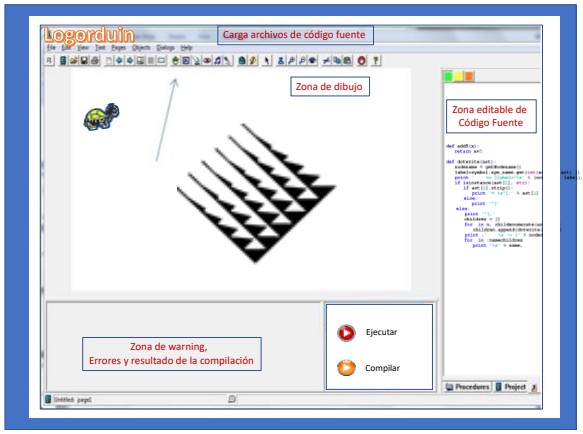


IDE

Se debe crear una interfaz gráfica la cual será el **único medio de revisión del proyecto**. En esta interfaz se deben ver claramente los errores y warnings del compilador. Esta interfaz debe contener los siguientes elementos:

- Poder ser capaces de "cargar" programas predefinidos para modificarlos, compilarlos y ejecutarlos.
- Debe existir un botón de "compilación" el cual no ejecute el código sino simplemente realiza "pasadas" de validación según cada etapa de los compiladores y genere el archivo objeto.
- La ejecución del archivo objeto se deberá hacer fuera del IDE, localizando la carpeta y realizando la ejecución directamente sobre el archivo.
- Debe haber una ventana en donde se pueda cargar o editar los programas y sobre los cuales se ejecutará la compilación.
- Debe haber una ventana en donde aparecerán de forma "decente" los errores que la "compilación" generará. No se permitirá el uso del IDE del lenguaje de programación usado para construir el Compilador ni ningún otro medio. Solamente se hará uso de la interfaz solicitada.
- Debe existir un botón para "imprimir" en la pantalla el AST generado de la validación del código fuente, de acuerdo con lo indicado anteriormente.
- Debe existir una ventana con ciertas dimensiones sobre la cual aparecerá el avatar (tortuga) y representará el "lienzo" sobre el cual se ejecutarán las instrucciones del código fuente, representando gráficamente lo que la ejecución del archivo objeto generará con el dispositivo de hardware. Esto servirá principalmente para la revisión de la entrega 2.

En resumen, la interfaz gráfica debe tener ser similar a lo siguiente:



Sintaxis

- Se debe respetar la sintaxis que más adelante se indica, pero el grupo puede "enriquecer" dicha gramática añadiendo las sentencias o comportamiento que consideren necesario, pero siempre deben respetar lo colocado en el presente documento. Además se deben tomar en cuenta las siguientes instrucciones:
    - Las variables usadas en el programa fuente deben tener un máximo de 10 posiciones. Debe iniciar siempre con una letra minúscula, las demás letras pueden ser minúsculas o mayúsculas, y solamente deberá permitir letras, números, y los símbolos especiales "_", "&" y "@". Todo programa debe contener al menos una variable. De lo contrario a todo lo anterior, se debe **generar un error**.
    - Los comentarios en el código fuente es por línea, y siempre debe iniciar con // y se comenta toda la línea, ejemplo:

      // Esto es un comentario

      Todo código fuente debe tener al menos un comentario indicando el nombre y la funcionalidad del código. De lo contrario **debe generar un error.** Los comentarios pueden estar presentes en cualquier lugar dentro del código del programa, pero siempre debe existir al menos uno en la primera línea de este.
    - En una misma línea se pueden colocar más de una instrucción
    - Se deben realizar todas las validaciones pertinentes como a nivel léxico, sintáctico, semántico, etc.
    - El cuerpo básico de un procedimiento debe cumplir con el siguiente formato:

      ```
      Para nombre_del_procedimiento [lista de parámetros]

          ... // Instrucciones.
      fin
      ```
    - Un procedimiento puede llevar parámetros de entrada.
    - Un procedimiento difiere de otro procedimiento por el nombre del mismo y los parámetros que contenga. Por tanto, cabe mencionar, que la firma de un procedimiento es el nombre y sus parámetros, tomando en cuenta cantidad de parámetros. No puede existir más de un procedimiento con la misma firma, pues **generaría un error**.
    - Adjunto una serie de sentencias mínimas que el Lenguaje debe contener, deben considerar las "expresiones" pues podrían ser colocadas en muchos lugares. Pueden mejorar sustancialmente la sintaxis recomendada, pero se debe respetar lo indicado a continuación.
- También como en otros entornos de programación un procedimiento puede llamar a otro procedimiento, lo que hará esto es realizar las instrucciones que tiene definidas el procedimiento al que se llama.

```
PARA nombre_del_procedimiento1 [lista de parámetros]
  Lista de instrucciones
  .
  .

FIN

PARA nombre_del_procedimiento2 [lista de parámetros]
  Lista de instrucciones
  .
  .
  nombre_del_procedimiento1

FIN
```

Sentencia en Lenguaje	Acción de la sentencia
`Haz nombre_variable valor`	A una variable se la puede asignar un valor, este valor puede ir cambiando a lo largo de la ejecución de un programa. El valor depende del tipo de dato que se quiera guardar en la variable. Ésta puede guardar los siguientes tipos de datos, son: - Entero (números sin decimales) - Lógico (True o False) - Cadena de texto **Haz** variable1 5 **Haz** variable2 “Hola” **Haz** variable3 TRUE El tipo de datos de la variable será asignado con el valor inicial usado en la creación de la variable. Si posteriormente se le asigna otro valor distinto al tipo de datos original, se deberá **generar un error** “semántico” tal y como se vio en clase.
`inic identifica = n;`	Se altera el valor actual de la variable por el dato colocado en la instrucción. n es el valor para usar. Ej. **INIC mivar =** 100 **INIC mivar =** 100 + 45 El valor usado en la inicialización de la variable puede ser el resultado de una operación.

`inc [N1]`
 	
`inc [N1 N2]`
	Incremente el valor de N1. Se tiene dos sintaxis:
	1. Si solo viene un identificador, el incremento será N1 + 1
	2. En caso de que se tengan una variable y un valor. Se incrementa N1 en N2 veces.
	
	Importante, N1 siempre debe ser una variable numérica pues es la que sufre el incremento. En caso de no ser una variable, debe **generar un error.**
	
	Ej.
	`INC [var]`
	
	`INC [var 5]`
	
	`INC [var var3]`
	
	Se puede utilizar un numero o una variable como segundo parámetro.

`avanza n`
 	
`av n`
	Mueva el avatar n cantidad de unidades hacia delante.
	Ej.
	`AVANZA 10`
	`AV 5*3`
	
	El valor por moverse puede ser producto de una operación, incluso usando variables.

`retrocede n`
 	
`re n`
	Mueva n cantidad de posiciones hacia atrás
	
	Ej.
	`Retrocede 10`
	`RE 5*3`
	
	El valor por moverse puede ser producto de una operación, incluso usando variables.

`giraderecha n`
 	
`gd n`
	Gira a la derecha el ángulo especificado en grados.
	Ej.
	`GiraDerecha 90`
	`GD 45`

	Observar que valor esta dado en grados, y especifica la forma de girar.

`giraizquierda n`
 	
`gi n`
	Gira a la izquierda el ángulo especificado en grados.
	Ej.
	`GiraIzquierda 90`
	`GI 45`
	
	Observar que valor esta dado en grados, y especifica la forma de girar.

`ocultatortuga`
 	
`ot`
	Coloca a la tortuga (dispositivo) en una posición inicial (parte superior izquierda).
	
	Ej.
	`OT`
	`OcultaTortuga`

`ponpos [X Y]`
 	
`ponxy x y`
	Coloca el avatar (tortuga) en la posición de las coordenadas X, Y.
	Ej.
	
	`PonPOS [ 100 0 ]`
	
	`PonXY 0 0`
	
	El avatar debe moverse a la posición indicada, pero no debe realizar ningún dibujo en el proceso de ubicación hasta el punto destino.

`ponrumbo n`	Coloca al avatar (tortuga) en la dirección del ángulo n. Ej. `PonRumbo 90`
`rumbo`	Indica la orientación que tiene actualmente el avatar (tortuga). Ej. `Muestra RUMBO`

`ponx n`	Coloca al avatar (tortuga) en la coordenada X especificada. Ej. `PonX 100`
`pony n`	Coloca al avatar (tortuga) en la coordenada Y especificada. Ej. `PonY 100`

`bajalapiz`
 	
`bl`
	El lápiz se coloca en posición para dibujar e inicia a dibujar cuando se mueve.
	
	Ej.
	`BajaLapiz`

`subelapiz`
 	
`sb`
	El lápiz deja de dibujar cuando se mueve.
	
	Ej.
	`SubeLapiz`

`poncolorlapiz c`
 	
`poncl color`
	Establece el color con que se pinta. La tortuga (dispositivo) tendrá la posibilidad de tener tres colores, esto es, tres tipos de lápices, pero uno es el principal.
	
	Al seleccionar un color, significa que el lapiz principal es el ubicado en este color.
	
	Los tres colores son: Negro, Azul y Rojo.
	
	Ej.
	`PonCL azul`
	`PonColoLapiz azul`

`centro`	Coloca la tortuga al centro de la zona de dibujo sin borrar nada.
`espera n`	El avatar (tortuga) se detendrá por n/60 segundos.

	Ej. `Espera 10`
`Ejecuta [ordenes]`	Permite ejecutar las “Ordenes” Ej. `Ejecuta [AV 10 GD 90]`
`repite n [ordenes]`	Ejecuta todas las instrucciones que se encuentran en el cuerpo (Ordenes) por n cantidad de veces. Ej. `REPITE 4 [Avanza 10 GiraDerecha 90]`
`SI (condición) [instrucciones]`	Realiza las instrucciones indicadas SI se cumple con la condición colocada. Ej. `SI (10>2) [RE 10 GD 20]` En la “condición” puede haber variables o valores, pero siempre debe retornar un valor booleano, de lo contrario debe **generar un error.**

`SI (condición) [instrucciones1] [instrucciones2]`	Realiza las primeras instrucciones si la condición expresada se cumple, y realiza el otro grupo de instrucciones sino se cumple la condición.
	`SI (var1=1)`
	`[RE 10 GD 20]`
	`[RE 25 GD 10]`

`HAZ.HASTA [instrucciones] (condicion)`	Repite la lista de instrucciones tantas veces hasta que se cumpla la condición. Primero se asegura que las instrucciones se ejecutan al menos una vez antes de comprobar la condición.
	`HAZ.HASTA [`
	`SI [numero=3] [salida 1]`
	`SI [var1=5] [salida 0]`
	`]`
	`(bucle=1)`

	En la “condición” puede haber variables o valores, pero siempre debe retornar un valor booleano, de lo contrario debe **generar un error.**

`HASTA (condicion) [instrucciones]`	Repite la lista de instrucciones tanta veces hasta que se cumpla la condición. Si la condición expresada no se cumple no se ejecutan las instrucciones ni una sola vez.
	
	`HAZ.HASTA (bucle=1)`
	`[`
	`SI [numero=3] [salida 1]`
	`SI [var1=5] [salida 0]`
	`]`

`HAZ.MIENTRAS [instrucciones] [condicion]`	Repite la lista de instrucciones tantas veces como se de la condición expresada. Primero lee el conjunto de instrucciones, de esta forma se asegura que las instrucciones se ejecutan al menos una vez antes de comprobar la condición.
	
	`HAZ.MIENTRAS [`
	`SI [numero=3] [salida 1]`
	`SI [var1=5] [salida 0]`
	`]`
	`(bucle=1)`
	
	En la “condición” puede haber variables o valores, pero siempre debe retornar un valor booleano, de lo contrario debe **generar un error.**

`MIENTRAS (condicion) [instrucciones]`	Repite la lista de instrucciones tanta veces como se de la condición. Si la condición expresada no se cumple no se ejecutan las instrucciones ni una sola vez.
	
	`MIENTRAS (bucle=1)`
	`[`
	`SI [numero=3] [salida 1]`
	`SI [var1=5] [salida 0]`
	`]`

`iguales? N1 N2`	Devuelve TRUE si N1 y N2 son iguales, de lo contario devuelve FALSO. Ej. `Iguales? 10 2*5`
`Y N1 N2`	Devuelve TRUE si tanto la condición N1 como N2 son ciertos. Ej. `Y (10>2) (2>5)`
`O N1 N2`	Devuelve TRUE si al menos una de las condiciones es cierta. Ej. `O (10>2) (2>5)`
`mayorque? N1 N2`	Devuelve TRUE si N1 es mayor a N2 Ej. `MayorQue? 10 2*5`
`menorque? N1 N2`	Devuelve TRUE si N1 es menor a N2. Ej. `MenorQue? 10 2*5`
`Diferencia N1 N2 N3 …`	Restar números (n1 – n2 – n3 …) Ej. `GiraIzquierda Diferencia 100 45`
`azar n`	Generar un número aleatorio comprendido entre 0 y n. Ej. `GiraIzquierda Azar 360`

`producto N1 N2 N3 …`	Multiplicar números (N1 * N2 * N3 …) Ej. **Producto** 1 2 3 4 5
`potencia N1 N2`	Calcular una potencia. Ej. **Potencia** 2 3
`división N1 N2`	Dividir dos números (N1 / N2) Ej. **división** 12 4 Solo retorna la posición entera.
`suma N1 N2 …`	Suma números. Ej. **Suma** 1 2 3 4 5

Para la solución del problema del proyecto presentado se espera del estudiante:

- Fuerte investigación sobre las posibles soluciones en las distintas etapas del problema
- Búsqueda de distintas fuentes que servirán de insumo técnico-práctico para las soluciones así como la ayuda de otras áreas para cumplir con el objetivo del proyecto.
- Selección de criterios acordes al nivel de nuestro ambiente Tec en donde se seleccionen las mejores soluciones correspondientes.
- Una solución solvente de acuerdo con lo esperado.

Para efectos de la revisión del proyecto, uno de los rubros de evaluación será presentar el código suficiente para poder recrear lo más parecido posible a por lo menos una de las siguientes imágenes:

Documentación y aspectos operativos

- ➔ Se deberá entregar un documento que contenga:
  - ✓ Diagrama de arquitectura de la solución que refleje un nivel de detalle suficiente para entender a términos generales el funcionamiento del proyecto. Deben investigar cómo se hace un diagrama de arquitectura general a nivel profesional.
  - ✓ Problemas conocidos: En esta sección se detalla cualquier problema que no se ha podido solucionar en el trabajo.
  - ✓ Problemas encontrados: descripción detallada, intentos de solución sin éxito, soluciones encontradas con su descripción detallada, recomendaciones, conclusiones y bibliografía consultada para este problema específico.
  - ✓ Conclusiones y Recomendaciones del proyecto. Conclusiones y recomendaciones con sentido y que confirmen una experiencia profunda en el proyecto.
  - ✓ Bibliografía consultada en todo el proyecto

Etapas del Proyecto

1. Definición del Lenguaje de Alto Nivel

- Sintaxis: Diseñar la sintaxis del lenguaje incluyendo variables, control de flujo, etc.
- Semántica: Definir las reglas semánticas para el uso correcto del lenguaje.

2. Análisis Léxico

- Especificación de Tokens: Definir los tokens del lenguaje (palabras clave, identificadores, operadores, etc.).
- Implementación del Analizador Léxico: Utilizar herramientas como Lex, Antlr para implementar el analizador léxico.

3. Análisis Sintáctico

- Gramática: Definir la gramática del lenguaje utilizando **BNF** o EBNF.
- Implementación del Analizador Sintáctico: Utilizar herramientas como Yacc o **ANTLR** para implementar el analizador sintáctico que genere el AST.

4. Análisis Semántico

- Verificación de Tipos: Implementar la verificación de tipos y la comprobación de reglas semánticas.
- Manejo de Errores: Desarrollar un sistema de manejo de errores semánticos.

---

6. Generación de Código

- Código Intermedio: Diseñar un formato de código intermedio.
- Generación de Código de Máquina: Traducir el código intermedio optimizado a código de máquina específico de la plataforma destino. Pueden hacer uso de herramientas que toman un IR para la generación de “código intermedio” y “código final” tales como **LLVM o GCC**.

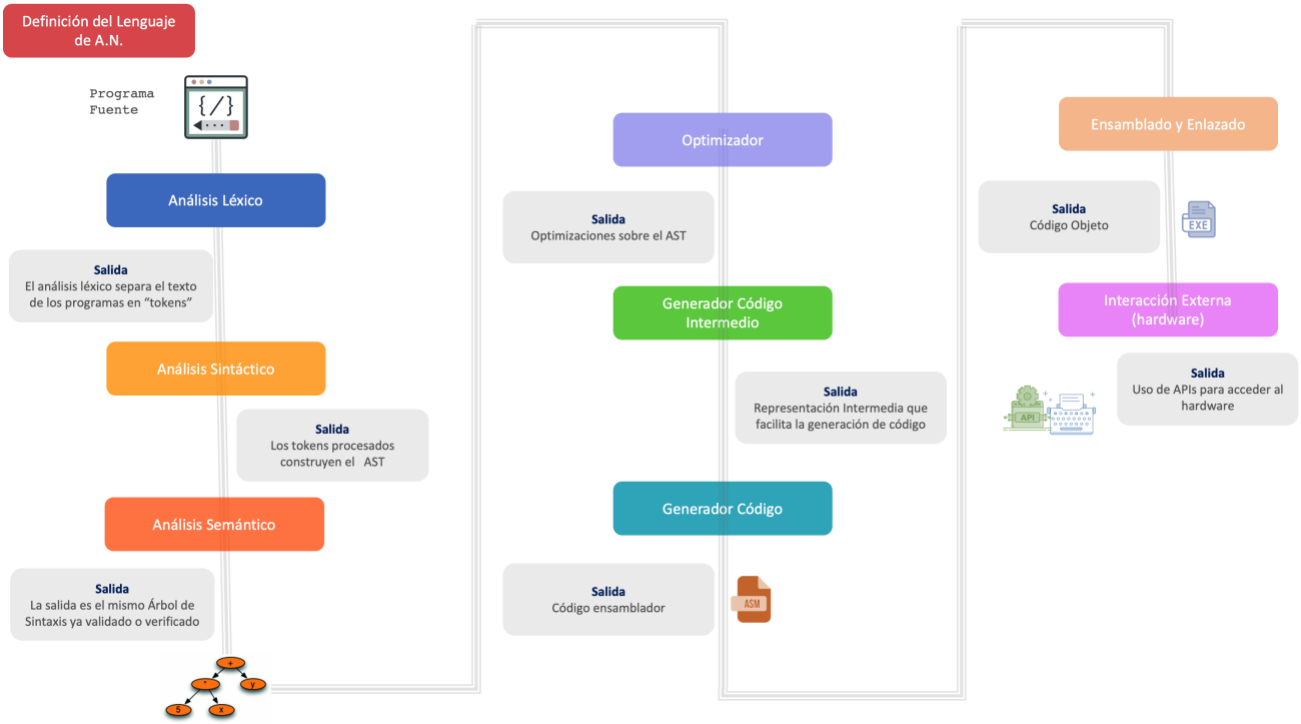
---

7. Ensamblado y Enlazado

- Ensamblador: Implementar o utilizar un ensamblador para convertir el código de máquina a un ejecutable.
- Enlazador: Enlazar las bibliotecas y dependencias necesarias para crear el ejecutable final.

8. Interacción con Dispositivos Externos

- API para Dispositivos Externos: Diseñar e implementar una API que permita al código compilado interactuar con dispositivos externos.



Entrega 1	Entrega 2	**Entrega Final**
1 Oct 2025	20 Oct 2025	**20 Nov 2025**
15%	15%	**10%**

Lo esperado

Para efectos del proyecto de "Compiladores e Intérpretes" del TEC, se desea desarrollen desde 0 (*desde el inicio*) el lenguaje propuesto, el cual tendrá las siguientes características:

- Para efectos de la validación léxica y sintáctica, pueden utilizar herramientas como **Lex y Yacc, ANTLR, etc..** El compilador debe "reaccionar" a posibles errores y evitar su ejecución.
- El lenguaje de programación usado para programar se deja a criterio de su propio equipo de trabajo.
- Decisiones técnicas como el tipo de lenguaje ensamblador, y demás decisiones se dejan a criterio del grupo, pero estas deben ser ampliamente justificadas en la documentación final.

- Se espera que el proyecto sea **completamente funcional**, y que permita la impresión de todo y cualquier imagen o elemento escrito en el código del lenguaje de programación creado.
- **Se debe crear un compilador con absolutamente todas las etapas de un Compilador** (pueden hacer uso de herramientas para tal caso) y se deben generar las señales necesarias para que cuando se ejecute el archivo objeto generado se pueda enviar dichas señales a un dispositivo externo (no al mismo usado en el compilador) para presentar las imágenes creadas.
- Se debe presentar por parte del grupo, un **programa con una imagen compleja** (desde el punto de vista de arte), la cual será evaluada en la revisión.
- Se debe generar una imagen de las solicitadas.
- Todo el proyecto debe ser implementado por el grupo de estudiantes, en caso de querer la utilización de bibliotecas, módulos u otras facilidades (no mencionadas anteriormente) y que disminuiría la carga de trabajo, se debe consultar previamente con el profesor y obtener su aprobación en dicho uso, en caso contrario se tomará como trabajo inconcluso.
- Se debe generar una interfaz en la computadora **agradable al usuario tipo IDE**, donde se pueda escribir y editar de manera amigable los programas de acuerdo con el lenguaje de programación creado.
- La Documentación a entregar es sobre el entregable (código fuente, ejemplos, etc.)

Evaluación

1. No debe imprimirse la documentación, y esta debe encontrarse en formato PDF.
2. Se debe hacer la entrega de lo solicitado en la fecha indicada antes de medianoche y solamente se aceptará la entrega en el **TecDigital**, no por otros medios.
3. Las citas de revisión serán determinadas por el profesor durante las lecciones o mediante algún medio electrónico.
4. El profesor llevara un listado de todos y cada uno de los requerimientos mencionados en el presente enunciado del proyecto, de tal manera que se pueda evaluar cada uno de ellos y dar el puntaje determinado.
5. Dentro de la revisión del proyecto se deberá presentar la gramática creada y dar una explicación detallada de la manera en que se implementó el compilador mostrando todos los elementos necesarios para entender su funcionalidad.
6. Todos los integrantes del grupo deberán estar presente en la defensa del proyecto, y todos deben estar preparados para contestar las preguntas que se les puede realizar en base a la implementación realizada. Solamente en casos justificados y vistos previamente con por lo menos un día de anticipación se podrá ausentar algún miembro del equipo.
7. La revisión de la documentación será realizada por parte del profesor, no durante la defensa del proyecto.
8. No se revisarán funcionalidades incompletas o no integradas.

Atributos

Se debe **entregar un documento aparte** que contenga una portada y el siguiente detalle.

Debe comentar de qué manera se aplicaron los atributos que posee el curso y su impacto, esto por medio de una tabla en donde se detalle para cada atributo lo siguiente:

- Aplicación del atributo en la solución del proyecto
- Impacto del proyecto en la sociedad
- Retroalimentación obtenida gracias al proyecto

Favor colocar la información antes solicitada para cada uno de los atributos siguientes:

Conocimiento de ingeniería
Capacidad para aplicar los conocimientos a nivel universitario de matemáticas, ciencias naturales, fundamentos de la Ingeniería y conocimientos especializados de ingeniería para la solución de problemas complejos de Ingeniería.

Persona ingeniera y el mundo (PM)
Analiza y evalúa el impacto ambiental y desarrollo sostenible en: la sociedad, la economía, la sostenibilidad, la salud y la seguridad, los marcos legales y el medio ambiente, al resolver problemas complejos de ingeniería.

Rubrica de Calificación
Entrega 1 (1 Oct 2025) - FrontEnd

Criterio	Muy Adecuado	Adecuado	Poco Adecuado	No hubo participación o Trabajo Deficiente
Gramática **(Se cumple con los ejemplos de sintaxis mostrados)**	Se presenta la gramática libre de contexto y se respeta la sintaxis del enunciado (10 pts)	No se respeta completamente la sintaxis de los ejemplos del enunciado (5 pts)	No se la sintaxis y la gramática tiene problemas (2 pts)	No hay evidencia del cumplimiento (0 pts)
Programa Complejo creado por el Equipo	Se presenta de forma ordenada y completa lo solicitado. (10 pts)	Se presenta de forma desordenado lo solicitado. (2 pts)	Se presenta de forma desordenado e incompleto lo solicitado (1 pts)	Ausencia de lo solicitado (0 pts)
Reconocimiento de Errores	Se encuentran todos los errores del programa prueba que presenta el profesor (25 pts)	Se encuentra el 70% de los errores mostrados (10 pts)	Se encuentra menos de 50% de los errores a detectar (5 pts)	No se encuentran los errores a detectar. (0 pts)
Reconocimiento de la Sintaxis	Se reconoce toda la sintaxis del enunciado colocado en el programa del profesor (25 pts)	Se reconoce al menos el 70% de la sintaxis del enunciado colocado en el programa del profesor (10 pts)	Se reconoce menos del 50% de la sintaxis del enunciado (5 pts)	No se reconoce la sintaxis (0 pts)
Presentación gráfica del AST	Se presenta de forma visual el AST del programa presentado por el profesor (20 pts)	El AST mostrado muestra algunos problemas o no es completo (10 pts)	El AST mostrado muestra bastantes problemas (5 pts)	No muestra el AST esperado (0 pts)
Documentación	Se presenta documentación de acuerdo a lo solicitado en el enunciado (10 pts)	Documentación incompleta (5 pts)	Documentación desordenada o no completa (1 pts)	Ausencia de lo solicitado (0 pts)

Rubrica de Calificación
Entrega 2 (20 Oct 2025) - BackEnd

Criterio	Muy Adecuado	Adecuado	Poco Adecuado	No hubo participación o Trabajo Deficiente
Generación de código final	Se genera exitosamente un archivo objeto producto de la compilación (20 pts)	Se requieren procesos adicionales al compilador para generar archivo objeto (10 pts)	Presenta problemas en la generación del archivo objeto (5 pts)	No hay evidencia del cumplimiento (0 pts)
Documenta y explica el proceso de Optimización de Código generado	Se muestra dominio del tema mostrando cada paso y elemento de la optimización realizada por la herramienta del backend (10 pts)	No se muestran con seguridad la etapa de optimización de código en la revisión del proyecto ni se documenta completamente (5 pts)	Explicación escaza o documentación incompleta (2pts)	No se evidencia la entrega del cumplimiento (0 pts)
Documenta y explica el proceso de Generación de código	Se muestra dominio del tema mostrando cada paso y elemento de la generación de código realizada por la herramienta del backend (10 pts)	No se muestran con seguridad la etapa de generación de código en la revisión del proyecto ni se documenta completamente (5 pts)	Explicación escaza o documentación incompleta (2pts)	No se evidencia la entrega del cumplimiento (0 pts)
Programa Complejo creado por el Equipo	Se ejecuta el programa objeto del código fuente del programa complejo creado por el equipo enviado a la pantalla (20 pts)	Se presentan ejecución parcial o problemática y requiere . (5 pts)	Se presenta de forma desordenado e incompleto lo solicitado (1 pts)	Ausencia de lo solicitado (0 pts)
Implementación de una de las imágenes solicitadas en el enunciado	Se ejecuta el programa objeto del código fuente del programa complejo creado por el equipo enviado a la pantalla (20 pts)	Se presentan ejecución parcial o problemática y requiere . (5 pts)	Se presenta de forma desordenado e incompleto lo solicitado (1 pts)	Ausencia de lo solicitado (0 pts)
Ejecución de la imagen solicitada en la revisión por parte del profesor	Se genera exitosamente la imagen solicitada por el profesor en la revisión (15 pts)	Se presenta ejecución parcial o problemática y requiere . (5 pts)	Se presenta de forma desordenado e incompleto lo solicitado (1 pts)	Ausencia de lo solicitado (0 pts)
Documentación	Se presenta documentación de acuerdo a lo solicitado en el enunciado (5 pts)	Documentación incompleta (2 pts)	Documentación desordenada o no completa (1 pts)	Ausencia de lo solicitado (0 pts)

Rubrica de Calificación
Entrega 3 (20 Nov 2025) - Hardware

Criterio	Muy Adecuado	Adecuado	Poco Adecuado	No hubo participación o Trabajo Deficiente
Dispositivo de hardware (tortuga) sincronizado con un archivo objeto generado del Compilador	Se evidencia comunicación entre el hardware y la ejecución del archivo objeto (20 pts)	Se requiere aspectos adicionales para la ejecución del hardware (15 pts)	Presenta problemas en ejecución del hardware (5 pts)	No hay evidencia del cumplimiento (0 pts)
Dispositivo de hardware (Componentes)	El dispositivo de hardware es completo de acuerdo al enunciado (15 pts)	Dispositivo incompleto en un 70% (10 pts)	Dispositivo incompleto en menos del 50% (2pts)	No se evidencia la entrega del cumplimiento (0 pts)
Ejecución completa del Programa Complejo creado por el Equipo	Se ejecuta el programa objeto del código fuente del programa complejo creado por el equipo enviado a la pantalla (20 pts)	Se presentan ejecución parcial o problemática y requiere . (5 pts)	Se presenta de forma desordenado e incompleto lo solicitado (1 pts)	Ausencia de lo solicitado (0 pts)
Implementación de una de las imágenes solicitadas en el enunciado	Se ejecuta el programa objeto del código fuente del programa complejo creado por el equipo enviado a la pantalla (20 pts)	Se presentan ejecución parcial o problemática y requiere . (5 pts)	Se presenta de forma desordenado e incompleto lo solicitado (1 pts)	Ausencia de lo solicitado (0 pts)
Ejecución de la imagen solicitada en la revisión por parte del profesor	Se genera exitosamente la imagen solicitada por el profesor en la revisión (20 pts)	Se presenta ejecución parcial o problemática y requiere . (5 pts)	Se presenta de forma desordenado e incompleto lo solicitado (1 pts)	Ausencia de lo solicitado (0 pts)
Documentación de ATRIBUTOS	Se presenta documentación de acuerdo a lo solicitado en el enunciado (5 pts)	Documentación incompleta (2 pts)	Documentación desordenada o no completa (1 pts)	Ausencia de lo solicitado (0 pts)