

**Catedráticos:** Ing. Edgar Sabán, Ing. Bayron López, Ing. Erick Navarro e Ing. Luis Espino

**Tutores académicos:** Erik Flores, Cristian Alvarado, Ronald Romero, Manuel Miranda, Haroldo Arias

# Compi Pascal

## Contenido

1. Competencias .....	3
1.1. Competencia general .....	3
1.2. Competencia específica .....	3
2. Descripción.....	4
2.1 Descripción General .....	4
2.2 Flujo específico de la aplicación .....	5
2.2.1. Ingreso de código fuente .....	5
2.2.2. Traducción al lenguaje desanidado .....	5
2.2.3. Ejecución .....	6
2.2.3. Generación de reportes .....	6
3. Componentes de la aplicación .....	6
4. Sintaxis de Compi Pascal .....	7
4.1. Generalidades .....	7
4.2. Tipos de dato validos .....	7
4.2.1. String: .....	7
4.2.2. Integer: .....	7
4.2.3. Real: .....	7
4.2.4. Boolean: .....	7
4.2.5. Void: .....	7
4.2.6. Types .....	7
4.2.6.1. Objects .....	7
4.2.6.2. Arrays .....	8
4.3. Declaraciones de variables y constantes .....	9
4.4. Asignación de variables .....	9
4.5. Operaciones aritméticas .....	9

4.6. Operaciones relacionales .....	9
4.7. Operaciones lógicas .....	9
4.8. Estructuras de control.....	10
4.9. Sentencias de transferencia.....	10
4.10. Funciones y procedimientos .....	11
4.10.1. Parámetros de funciones y procedimientos .....	11
4.10.2. Procedimientos anidados .....	12
4.10.3. Funciones anidadas.....	13
4.10.4. Traducción funciones y procedimientos anidados .....	13
4.11. Funciones nativas.....	16
4.11.1. write y writeln .....	16
4.11.2. Exit.....	16
4.11.3. graficar_ts .....	16
5. Reportes generales .....	18
5.12. Tabla de símbolos .....	18
5.13. AST .....	18
5.14. Reporte de errores.....	19
6. Entregables y calificación .....	20
6.1. Entregables .....	20
6.2. Restricciones .....	20
6.3. Consideraciones .....	20
6.4. Calificación .....	21
6.5. Entrega del proyecto.....	21

# 1. Competencias

## 1.1. Competencia general

Que el estudiante aplique la fase de **síntesis del compilador** para realizar un **traductor e intérprete** utilizando herramientas.

## 1.2. Competencia específica

- Que el estudiante **utilice un generador de analizadores léxicos y sintácticos** para construir un **traductor**.
- Que el estudiante **implemente la ejecución de dicha traducción** utilizando **traducción dirigida por la sintaxis** haciendo uso de **atributos heredados** y sintetizados.
- Que el estudiante comprenda los conceptos acerca de traducciones.
- Que el estudiante **maneje la pila o el árbol** que proporciona el analizador sintáctico para simular el paso de atributos heredados.



## 2. Descripción

### 2.1 Descripción General

Para el primer proyecto se deberá desarrollar un traductor y un intérprete, el traductor acepta un lenguaje en el cual es posible definir y utilizar funciones anidadas, este lenguaje luego debe poder ser traducido a una versión donde las funciones han sido desanidadas, este traductor no realiza ejecución de código, su única tarea es la de traducción.



El intérprete ejecutará el código sin funciones anidadas, no es necesario que el código sea traducido primero para poder ejecutarlo, ambas funciones son independientes.

El nombre del lenguaje es llamado "Compi pascal", contará con los componentes descritos en la sección 3 y debe ser desarrollado con el framework .Net utilizando la herramienta irony para implementar un analizador ascendente.

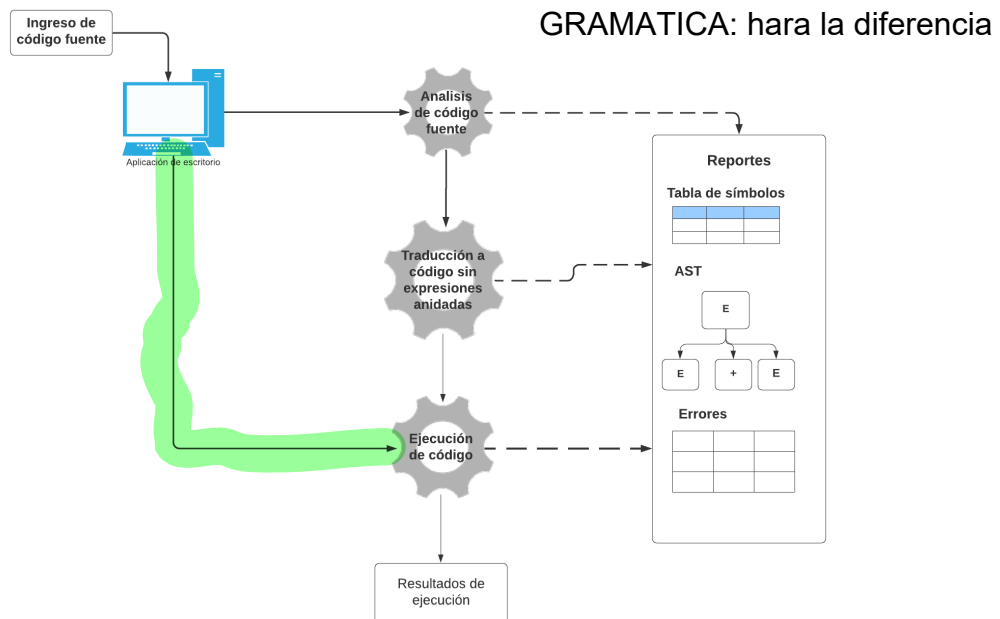
Compi pascal es una variación del lenguaje de programación Pascal. En este lenguaje será posible definir y utilizar funciones anidadas para luego, mediante una traducción, generar código que es aceptado por el lenguaje base pascal, es decir, sin funciones anidadas.

Para la traducción de funciones anidadas a desanidadas del lenguaje compi pascal es necesario la utilización de atributos heredados, esta implementación quedará a discreción de cada estudiante.

El proceso de traducción e interpretación es el siguiente:

- **Traducción del lenguaje compi pascal a pascal puro:** Al ingresar el código de entrada se tiene la opción de traducir dicho código, si este mismo cuenta con funciones anidadas se deberá desanidar por medio de una traducción, el proceso será descrito en la sección 4.
- **Ejecución del código:** El intérprete ejecutará el programa sin funciones desanidadas, si alguna aún se encuentra presente dentro del programa, se deberá reportar como un error.
- **Generación de reportes:** Luego de su traducción y/o ejecución el programa debe poder generar:
  - Tabla de símbolos
  - Listado de errores
  - Diagrama AST

## 2.2 Flujo específico de la aplicación



### 2.2.1. Ingreso de código fuente

El lenguaje está basado en Pascal con **instrucciones limitadas** además de poder definir **funciones dentro de funciones**, **no hay límite para el nivel de anidamiento**, se explica más a detalle en la sección 4.11 para las funciones anidadadas y en la sección 4 se especifican las limitaciones del lenguaje.

### 2.2.2. Traducción al lenguaje desanidado

La **traducción** se realizará sobre la **entrada**, se realiza el análisis léxico, sintáctico y semántico sobre la entrada y generará el mismo lenguaje basado en Pascal, luego al final de la traducción posiblemente mostrará la **lista de errores** que se encontraron en el análisis, mostrará el **reporte de la tabla de símbolos** para asegurar un correcto análisis de la cadena de entrada, luego generará el **lenguaje de salida** que es **con la misma sintaxis** que el lenguaje de entrada, pero con las **funciones desanidadadas** y con nombres diferentes para **evitar conflictos** en el programa.

- Las **restricciones sobre las funciones** se describirán en la sección 4
- El nombre de las funciones desanidadadas queda a discreción del estudiante, se recomienda **nombrarlas** a manera de que se pueda identificar cuáles son sus funciones padres.
- Se calificará el **correcto funcionamiento de cada función anidada**.
- Si existen errores en el lenguaje fuente, **deberá recuperarse de estos errores y seguir traduciendo**, el archivo de salida tendrá que contar con los mismos errores, la recuperación de errores se especifica en la sección.
- El **reporte de tabla de símbolos** será necesario para comprobar que se realizó el análisis léxico, sintáctico y semántico.
- El **lenguaje de salida** debe ser funcional al probarlo con el intérprete que el estudiante debe realizar, así se verificará que la traducción fue exitosa.



### 2.2.3. Ejecución

Ejecución sobre el código de entrada, si el código que se ejecuta cuenta con funciones anidadas, entonces se marcará como un error y se reportará, de lo contrario generará la salida en consola.



### 2.2.3. Generación de reportes

Cuando el programa traduce el código fuente o ejecuta el código resultado, es posible generar los reportes de tabla de símbolos, AST y errores para la verificación de como el estudiante usa las estructuras internas para la interpretación del lenguaje.

## 3. Componentes de la aplicación

La aplicación deberá contar con los siguientes elementos para su correcto funcionamiento:

- **Botón de traducción:** Este botón realizará la acción traducir el código de entrada con funciones anidadas a un código sin funciones anidadas, además si este código contiene errores debe reportarlos, el código traducido puede generarse en una nueva ventana o reemplazar el código antiguo, queda a discreción del estudiante este paso.
- **Botón de ejecución:** Al presionarlo ejecutará el código de entrada y si existen errores en su análisis y ejecución los deberá reportar.
- **Botón de reportes:** Mostrará al presionarlo el reporte de errores, AST y tabla de símbolos.
- **Consola de salida:** Esta muestra el resultado de la ejecución del código.



(\*Codigo entrada\*)  
program ejemplo;  
begin  
    writeln('hola a todos');  
end.

Compi Pascal

Consola

TraducirEjecutarReportes

Ejemplo de interfaz gráfica

## 4. Sintaxis de Compi Pascal

Compi Pascal provee la posibilidad de **tipado estático**, funciones anidadas y que su traducción sea exactamente la misma a excepción de las funciones, ya que las funciones estarán desanidadas luego de la traducción inicial.

La sintaxis de Compi Pascal es similar a Pascal, pero con la salvedad que no se utilizarán todas las funcionalidades que este lenguaje posee, a continuación, se describen las instrucciones válidas.

En el siguiente enlace se encuentra de forma más detallada la sintaxis y ejemplos: <https://www.tutorialspoint.com/pascal/index.htm>

### 4.1. Generalidades

- El lenguaje **no es case sensitive**, por lo tanto, un **identificador** declarado como **PRUEBA** es igual a uno declarado como **prueba**, esto también aplica para las **palabras reservadas**.
- Existen 3 tipos de **comentarios**:
  - o Comentarios de una línea **//**
  - o Comentarios de múltiples líneas **(\* ... \*)**
  - o Comentarios de múltiples líneas **{ ... }**

### 4.2. Tipos de dato validos

#### 4.2.1. String:

Se deben utilizar **comillas simples**, las comillas dobles no están permitidas para una cadena.

#### 4.2.2. Integer:

Son números **enteros positivos o negativos**.

#### 4.2.3. Real:

Son números **decimales positivos o negativos**.

#### 4.2.4. Boolean:

Valores **true y false**.

#### 4.2.5. Void:

Para definir los **procedimientos**.

#### 4.2.6. Types

En pascal podemos definir diversos tipos de datos, para este proyecto limitaremos solo a los siguientes:

##### 4.2.6.1. Objects

Estos pueden **contener cualquier tipo de dato o arreglo** en su interior, incluyendo **otros objects o arreglos de objects**.

Para la **definición** de objects, Compi Pascal **se limita** a lo que tiene el lenguaje nativo de Pascal, ya que Compi Pascal utiliza los objects como la definición de objects de C, por lo cual la estructura de estos será la siguiente:

```

{
    Definición de un Type en Compi Pascal
}

program Hello;

type
    Rectangle = object
    var
        length, width: integer;
        area, volumen: real;
    end;

var
    r1: Rectangle;

begin
    r1.length := 10;
    r1.volumen := 50.0;
    writeln (r1.length);
    writeln (r1.volumen);
end.

```

#### Consideraciones:

- No es necesario inicializar los valores de los atributos del object.
- Cuando se declara un type, sus variables internas se inicializarán con los valores por defecto de Pascal.
- No se permiten la declaración de funciones o procedimientos dentro del object.

#### 4.2.6.2. Arrays

Los arreglos pueden ser de cualquier tipo de dato válido (Incluso arreglos o types) y son dinámicos, además poseen la siguiente estructura:

```

type
    array-identifier = array[index-type] of element-type;

```

#### Consideraciones:

No se utilizarán Enumerated Types y Subrange Types.



### 4.3. Declaraciones de variables y constantes

La sintaxis de la declaración debe ser igual a la sintaxis de pascal, en donde se pueden definir listas de variables de un tipo en específico y para el caso de constantes únicamente una constante por definición.

#### Consideraciones:

- Las constantes es obligatorio que se declaren con un valor.
- Si se definen variables con asignación de un valor, únicamente se permite agregar una variable en la definición como lo delimita el lenguaje pascal.
- No puede declararse una variable o constante con un identificador que tengan el mismo nombre.
- Si una variable no se inicializa, tomará los valores por defecto del lenguaje Pascal.
- Las variables solo aceptan un tipo de dato, el cual no puede cambiarse en tiempo de ejecución.
- Se debe validar que el tipo de dato y el valor sean compatibles, según las definiciones del lenguaje Pascal.

### 4.4. Asignación de variables

Las variables no pueden cambiar de tipo de dato, se deben mantener con el tipo declarado inicialmente.

#### Consideraciones

- No es posible cambiar el tipo de dato de una variable, a menos que el tipo de dato no se haya especificado al declarar la variable.
- Una constante no puede ser asignada.
- Se debe validar que el tipo de la variable y el valor sean compatibles.

### 4.5. Operaciones aritméticas

Entre las operaciones disponibles se encuentran las siguientes

- Suma (+)
- Resta (-)
- Multiplicación (\*)
- División (/)
- Modulo (%)

### 4.6. Operaciones relacionales

- Mayor que (>)
- Menor que (<)
- Mayor o igual que (>=)
- Menor o igual que (<=)
- Igualdad (=)
- Diferenciación (<>)

### 4.7. Operaciones lógicas

- AND
- OR
- NOT

#### 4.8. Estructuras de control

Las estructuras de control permiten regular el flujo de la ejecución del programa. Este flujo de ejecución se puede controlar mediante sentencias condicionales que realicen ramificaciones e iteraciones. El lenguaje soporta las siguientes estructuras:

- If-then
- If-then-else
- Case
- Case - else
- While-do
- Repeat-until
- For-do



#### 4.9. Sentencias de transferencia

Las sentencias de transferencia permiten manipular el comportamiento normal de los bucles, ya sea para detenerlo o para saltarse algunas iteraciones. El lenguaje soporta las siguientes sentencias:

- Break
- Continue

##### Consideraciones

- Se debe verificar que la sentencia break aparezca dentro de un ciclo o dentro de una sentencia Case.
- Se debe verificar que la sentencia continue aparezca dentro de un ciclo.

## 4.10. Funciones y procedimientos

Compi Pascal permite el uso de funciones y procedimientos, donde las funciones realizan una acción de retorno y los procedimientos no.

```
program functionsandprocs;
var a:integer = 50;
var min:integer;
// factorial usando la función Exit para asignar el retorno.
function factorial( num : integer) : integer;
begin
  if num = 1 then
    Exit(1)
  else
    Exit(num * factorial( num-1 ));
end;
// factorial usando el nombre de la función para asignar el retorno
function factorial2( num : integer) : integer;
begin
  if num = 1 then
    factorial2 := 1
  else
    factorial2 := num * factorial2( num-1 )
end;
// ejemplo de procedimiento
procedure findMin(x, y, z: integer; var m: integer); {m es por referencia}
(* Finds the minimum of the 3 values *)
begin
  if x < y then
    m:= x
  else
    m:= y;

  if z < m then
    m:= z;
end;
begin
  writeln('el resultado es: ', factorial(5));
  writeln('el resultado es: ', factorial2(5));
  findMin(1, 2, 3, min); (* Procedure call *)
  writeln(' Minimum: ', min);
end.
```

### 4.10.1. Parámetros de funciones y procedimientos

Todos los parámetros son por valor, a menos que se le anteponga la palabra reservada **var** al nombre del parámetro, lo cual lo convierte a una referencia. Tal como aparece en el ejemplo del inciso 4.10.



#### 4.10.2. Procedimientos anidados

Una de las características del lenguaje es el uso **procedimientos anidados**, lo que significa que se **pueden definir procedimientos** o **funciones dentro** de los **mismos procedimientos** en el lenguaje para luego al momento de la **traducción** estos **procedimientos** sean **desanidados** por **completo**.

```
(*Ejemplo entrada de un procedimiento adentro de un procedimiento*)
```

```
program Ejemplo_anidadas;
```

```
(* procedimiento padre *)
```

```
procedure saludo;
```

```
    (*Procedimiento hijo de saludo*)
```

```
    procedure despedida;
```

```
    begin
```

```
        writeln('adios compañero');
```

```
    end;
```

```
begin
```

```
    writeln('hola compañero');
```

```
    despedida();
```

```
end;
```

```
(* Main *)
```

```
begin
```

```
    saludo();
```

```
end.
```

### 4.10.3. Funciones anidadas

Al igual que los procedimientos anidados, las funciones pueden llevar funciones o procedimientos dentro de la definición de sus elementos, pero a diferencia de los procedimientos todas las funciones deben retornar un valor.

```
(* Ejemplo entrada de un procedimiento adentro de un procedimiento *)
program Ejemplo_anidadas_2;
(* procedimiento padre *)
function calificacion:integer;
    (*Procedimiento hijo de saludo*)
    function nota:integer;
    begin
        writeln('Generando nota');
        nota :=10;
    end;

begin
    writeln('Creando nota');
    calificacion:=nota()*10;
end;

(* Main *)
begin
    writeln(calificacion());
end.
```



### 4.10.4. Traducción funciones y procedimientos anidados

Pascal ya cuenta con el soporte de funciones anidadas, por lo cual, se debe realizar una traducción sobre la llamada a la función anidada, y desanidar la función escribiendo una nueva, eliminando la función original y escribiendo las funciones nuevas ya desanidadas.

```
(*Ejemplo entrada de un procedimiento adentro de un procedimiento*)
program Ejemplo_anidadas_4;
(* Procedimiento padre *)
function calificacion():integer;
    (* Procedimiento hijo *)
    function nota:integer;
    begin
        writeln('Generando nota');
        nota :=10;
    end;
begin
    writeln('Creando nota');
    calificacion:=nota()*10;
end;
```

```
(* Main *)  
begin  
  writeln(calificacion()); //llamada a función calificación  
end.
```

*Archivo de entrada sin traducir.*

```
(*Ejemplo entrada de un procedimiento adentro de un procedimiento*)  
program Ejemplo_anidadas_5;  
(*Procedimiento hijo desanidado*)  
function calificacion_nota:integer;   
begin  
  writeln('Generando nota');  
  calificacion_nota :=10;   
end;  
(* Procedimiento padre *)  
function calificacion():integer;  
begin  
  writeln('Creando nota');  
  calificacion:=calificacion_nota()*10;  
end;  
(* Main *)  
begin  
  writeln(calificacion()); //llamada a función calificación  
end.
```

Ejemplo de traducción del archivo de entrada.

#### 4.10.5. Uso de parámetros y variables locales en funciones y procedimientos anidados



En el lenguaje pascal las funciones hijas pueden usar las variables locales y parámetros de las funciones padres, abuelas, etc.

Al momento de traducir una función o un procedimiento, se debe traducir a una nueva función con los parámetros de los padres **SI ESTOS SON USADOS DENTRO DE LA FUNCION HIJA.**



Si una variable declarada en el padre es utilizada por el hijo, esta debe ser agregada como parámetro a la función hija al momento de desanidarla, además enviarla al escribir de nuevo la llamada a la función o procedimiento desanidado.

```
(*Ejemplo entrada de un procedimiento adentro de un procedimiento*)
program Ejemplo_anidadas_6;

(*Variables del programa*)
var unidad : integer;

(* procedimiento padre *)
function calificacion2(a:integer; var b:integer):integer;
  Var q : integer = 100;

  (*Funcion anidada 1*)
  function nota2(c:integer):integer;
  begin
    writeln('Generando nota2');
    nota2 :=b*400*c*q;
  end;
begin
  calificacion2:=nota2(50);
end;

(* Main *)
begin
  unidad := 50;
  writeln(calificacion2(10, unidad));
end.
```

Archivo de entrada de funciones anidadas con parámetros

```

(*Ejemplo entrada de un procedimiento adentro de un procedimiento*)
program Ejemplo_anidadas_6;

(*Variables del programa*)
var unidad : integer;

(*Función desanidada 1*)
//Se agregó el parámetro "b" del padre porque lo utiliza el hijo
//Se agregó la variable del padre "q" como parámetro "calificacion2_q"
function calificaion2_nota2(var b:integer;c:integer;calificacion2_q:integer):integer;
begin
    writeln('Generando nota2');
    calificaion2_nota2 :=b * 400 * c * calificacion2_q;
end;

(* procedimiento padre *)
function calificacion2(a:integer; var b:integer):integer;
Var q : integer = 100;
begin
    //se modifica la llamada a la funcion con el parametro adicional
    // se agrega "q" como parámetro porque el hijo lo utiliza
    calificacion2:=calificaion2_nota2(b,50,q);
end;

(* Main *)
begin
    unidad := 50;
    writeln(calificacion2(10, unidad));
end.

```

Ejemplo de archivo con código traducido.

#### 4.11. Funciones nativas

##### 4.11.1. write y writeln

Esta función nos permite imprimir cualquier expresión que le mandemos como parámetro, y en un formato comprensible, tomar las consideraciones y variaciones que posee el lenguaje pascal.

##### 4.11.2. Exit

A diferencia de la asignación del valor de retorno de una función en base a su nombre, esta función también detiene la ejecución en el lugar exacto en donde es invocada.

##### 4.11.3. graficar\_ts

Esta función nos va a permitir graficar la tabla de símbolos en cualquier parte del programa donde se encuentre esta instrucción, esto servirá para validar el correcto manejo de los ámbitos.



### Consideraciones:

- **Es obligatorio que la gramática de las funciones este expresada para un análisis descendente, por lo tanto, es obligatorio el uso de atributos heredados o en su defecto utilizar la pila o el árbol del analizador sintáctico para obtener los valores anteriores.**
- **La traducción para funciones debe de eliminar las funciones anidadas y crear nuevas en otra parte del código.**
- Las funciones y procedimientos no soportan sobrecarga.
- Las funciones y procedimientos en Compilador Pascal se pueden anidar cualquier cantidad de veces
- Las funciones anidadas pueden invocar otras funciones GLOBALES y locales siempre y cuando estén en el mismo ámbito.
- No es posible declarar variables dentro del cuerpo del procedimiento o función.
- Los parámetros deben tener distinto nombre.
- Las funciones pueden retornar cualquier tipo de dato y este debe estar especificado.
- Únicamente las funciones pueden retornar un valor.
- La asignación de nombres a las nuevas funciones y/o parámetros al momento de la traducción queda a discreción del estudiante.
- En el cuerpo de la función debe haber una asignación de la forma **name := expresión;** que asigna un valor al nombre de la función. Este valor es retornado cuando la función es ejecutada.



## 5. Reportes generales

### 5.12. Tabla de símbolos



Este reporte mostrará la **tabla de símbolos** después de la **ejecución del archivo**. Se deberán de mostrar **todas** las **variables, funciones y procedimientos** que **fueron declarados**, así como su **tipo** y toda la información que el estudiante **considere necesaria**.

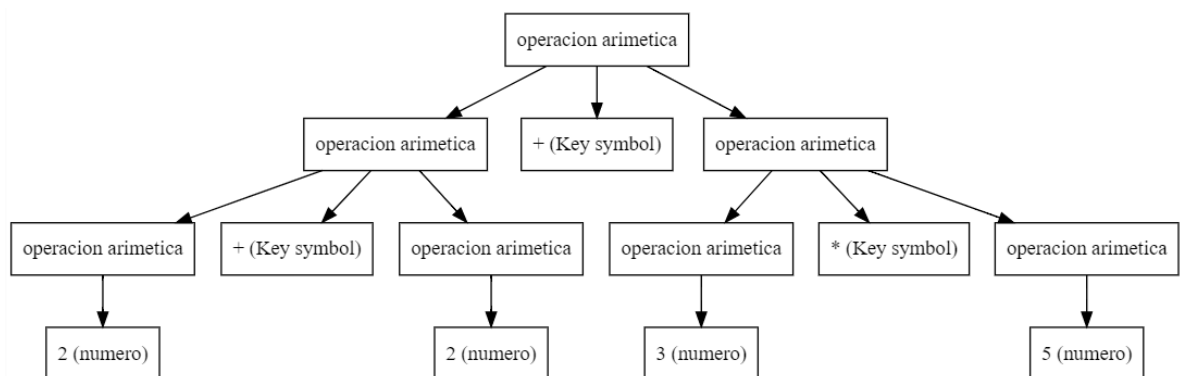
Nombre	Tipo	Ámbito	Fila	Columna
Temp	string	global	5	8
A	number	funcionA	10	7
B	number	funcionB	12	5
C	boolean	global	13	10

*Ejemplo de posible reporte de tabla de símbolos*

### 5.13. AST



Este reporte mostrara el **árbol sintáctico** que se **produjo** al **analizar el archivo de entrada**. Este debe de representarse como un grafo, se recomienda se utilizar Graphviz. El Estudiante deberá mostrar los nodos que considere necesarios y se realizarán preguntas al momento de la calificación para que explique su funcionamiento.



*Ejemplo de posible salida del reporte AST*



## 5.14. Reporte de errores

El traductor e intérprete deberá ser capaz de detectar todos los errores que se encuentren durante el proceso de traducción. Todos los errores se deberán de recolectar y se mostrará un reporte de errores en el que, como mínimo, debe mostrarse el tipo de error, su ubicación y una breve descripción de por qué se produjo.

Los tipos de errores se deberán de manejar son los siguientes:

- Errores léxicos.
- Errores sintácticos.
- Errores semánticos

La tabla de errores debe contener la siguiente información:

- **Línea:** Número de línea donde se encuentra el error.
- **Columna:** Número de columna donde se encuentra el error.
- **Tipo de error:** Identifica el tipo de error encontrado. Este puede ser léxico, sintáctico o semántico.
- **Descripción del error:** Dar una explicación concisa de por qué se generó el error.
- **Ámbito:** Si fue en una función o procedimiento (decir en cuál fue) o en el ámbito global.

Tipo	Descripción	Línea	Columna
Sintáctico	Se esperaba "=", en lugar se encontró "=="	112	15
Semántico	El tipo string no puede multiplicarse con un real	80	10
sintáctico	Se esperaba la palabra reservada "if" en su lugar se encontró "else"	1000	5

*Ejemplo de posible reporte de errores*

### Consideraciones

Se deben reportar errores tanto en tiempo de traducción como tiempo de ejecución. Queda a discreción del estudiante si presentarlos en reportes separados o en un mismo reporte especificando en qué fase ocurrió el mismo.

## 6. Entregables y calificación

Para el desarrollo del proyecto se deberá utilizar un repositorio de **github**, este repositorio deberá ser privado.

### 6.1. Entregables

Todo el código fuente se va a manejar en github, por lo tanto, el estudiante es el único responsable de mantener actualizado dicho repositorio hasta la fecha de entrega, se calificará hasta el último commit antes de la fecha y hora límite.

- Código fuente publicado en un repositorio de github.
- Enlace al repositorio y **permisos a los auxiliares** para poder acceder.

### 6.2. Restricciones

- La herramienta para generar los analizadores del proyecto será Irony. La documentación se encuentra en el siguiente enlace <https://archive.codeplex.com/?p=irony>
- Para la generación de graficas se debe de utilizar Graphiz.
- Copias de proyectos tendrán de manera automática una nota de 0 puntos y serán reportados a la Escuela de Ciencias y Sistemas los involucrados.
- ***Para el desarrollo del proyecto se deberá de crear una carpeta en esta ruta C:\compiladores2, y aquí deben de colocar el DLL de Irony (esta se debe de llamar Irony.dll) para la referencia del proyecto de Visual Studio, adicional agregar que su IDE guarde en esta ruta las imágenes que generan las gráficas y archivos de reportes, esto para tener un estándar y evitar problemas en la calificación por rutas quemadas en el código, ya que no se podrá modificar nada el día de la calificación.***
- ***Se debe de utilizar Visual Studio 2019 como IDE de desarrollo y se debe utilizar una aplicación de Windows Forms App con .Net Core con el SDK 3.1***
- El desarrollo y entrega del proyecto es individual.

### 6.3. Consideraciones

- Durante la calificación se realizarán preguntas sobre el código para verificar la autoría de este, de no responder correctamente la mayoría de las preguntas se reportará la copia.
- El repositorio únicamente debe contener el código fuente empleado para el desarrollo, no deben existir archivos pdf o docx.
- El sistema operativo a utilizar es libre.
- El lenguaje está basado en Pascal, por lo que el estudiante es libre de realizar archivos de prueba en estas herramientas, el funcionamiento debería ser el mismo y limitado a lo descrito en este enunciado.
- Pascal en línea [https://www.onlinegdb.com/online\\_pascal\\_compiler](https://www.onlinegdb.com/online_pascal_compiler)
- Se van a publicar archivos de prueba en el siguiente repositorio:  
[https://github.com/CrisAlva25/Compi2\\_1S2021](https://github.com/CrisAlva25/Compi2_1S2021)



## 6.4. Calificación

- La calificación se realizará dentro de la máquina de los auxiliares, ya que es muy importante que tengan la última versión de su proyecto subida a github y las rutas definidas anteriormente.
- Se tendrá un máximo de 30 minutos por estudiante para calificar el proyecto.
- La hoja de calificación describe cada aspecto a calificar, por lo tanto, si la funcionalidad a calificar falla en la sección indicada se tendrá 0 puntos en esa funcionalidad y esa nota no podrá cambiar si dicha funcionalidad funciona en otra sección.
- Si una función del programa ya ha sido calificada, esta no puede ser penalizada si en otra sección la función falla o es errónea.
- Los archivos de entrada podrán ser modificados solamente antes de iniciar la calificación eliminando funcionalidades que el estudiante indique que no desarrolló.
- Los archivos de entrada podrán ser modificados si contienen errores léxicos, sintácticos o semánticos no descritos en el enunciado o provocados para verificar el manejo y recuperación de errores.

## 6.5. Entrega del proyecto

- La entrega será mediante github, y se va a tomar como entrega el código fuente publicado en el repositorio a la fecha y hora establecidos.
- Cualquier commit luego de la fecha y hora establecidas invalidará el proyecto, por lo que se calificará hasta el último commit dentro de la fecha válida.
- **No habrá prorroga**
- Fecha de entrega:

**Domingo 14 de marzo hasta las 23:59 PM**