# Objetivos generales

- Que el estudiante implemente las fases de un compilador en el análisis léxico, análisis sintáctico y semántico.
- Aplicar los conocimientos a la traducción dirigida por la sintaxis, así como el manejo de errores para crear herramientas útiles.
- Elaborar la lógica para la solución del problema planteado.

# Objectivos específicos

- Construir y utilizar árboles de sintaxis abstracta.
- Combinar la funcionalidad de JFlex y Cup en aplicaciones reales.
- Desarrollar aplicaciones de escritorio utilizando lenguaje Java
- Realizar análisis semántico utilizando árboles de sintaxis abstracta.
- Manejar errores léxicos, sintácticos y semánticos.

# Descripción de la actividad

La empresa de desarrollo Code'n Bug ha decidido migrar un antiguo sistema escrito en lenguaje Pascal a un programa en java. Sin embargo, el sistema tiene muchos archivos y la empresa no ha encontrado un IDE de calidad para evaluar si dichos archivos están escritos respetando las reglas del lenguaje Pascal, esto es necesario ya que se deben descartar los archivos que no cumplan con el lenguaje.

Por lo tanto, usted debe realizar un programa de escritorio similar a un IDE que permita cargar y/o trabajar archivos escritos en Pascal y realizar todas las comprobaciones léxicas, sintácticas y semánticas.

Otro requerimiento que se desea es poder visualizar la tabla de símbolos y tipos de cada archivo, así como el árbol de activaciones de subprogramas.

# Funcionalidad del IDE

- El IDE debe permitir abrir varios archivos en pestañas diferentes para trabajarlos de forma independiente.
- El IDE debe permitir crear nuevos archivos.
- El IDE debe permitir guardar cambios en archivos.
- El IDE debe tener la opción "Guardar como".
- El IDE debe permitir visualizar los errores léxicos, sintácticos y semánticos después de analizar un archivo.
- El IDE debe permitir visualizar la tabla o tablas de símbolos del archivo analizado si no se encontraron errores.
- El IDE debe permitir visualizar el árbol de activaciones del archivo analizado si no se encontraron errores.
- El IDE debe tener una opción para analizar todos los archivos abiertos.

# Definición del lenguaje Pascal

Por razones de tiempo se han contemplado solo las siguientes estructuras y características del lenguaje:

## Estructura general

```
program {name of the program}
type {global types declaration block}
const {global constant declaration block}
var {global variable declaration block}

function {function declarations, if any}
{ local variables }
begin
...
end;

procedure { procedure declarations, if any}
{ local variables }
begin
...
end;

begin { main program block starts}
...
end. { the end of main program block }
```

## Sintaxis básica

### Comentarios

Comentario de línea usando llaves

```
{ This is a single line comment in pascal }
```

Comentario multilínea usando (* *)

```
(* This is a multi-line comments
   and it will span multiple lines. *)
```

## Case non-sensitive

Pascal no diferencia entre minusculas y mayusculas. Esto afecta al nombre de variables, subprogramas y palabras reservadas.

## Statements

Un programa en Pascal está hecho de statements. Estos statements pueden ser declaraciones, asignaciones, lectura de data, escritura de datos, decisiones lógicas, estructuras de control, etc.

```
readln (a, b, c);
s := (a + b + c)/2.0;
area := sqrt(s * (s - a)*(s-b)*(s-c));
writeln(area);
```

## Statement begin end

Begin end es un statement especial que permite agrupar más statements. Se usa dentro de estructuras de control como en subprogramas y en el programa principal.

## Palabras reservadas

| | | | | | | |
|---|---|---|---|---|---|---|
| and | array | begin | case | const | div | do |
| downto | else | end | file | for | function | goto |
| if | in | label | mod | nil | not | of |
| or | packed | procedure | program | record | repeat | set |
| then | to | type | until | var | while | with |

## Caracteres permitidos por Pascal

Letras en mayúscula [A - Z]

Letras en minúscula [a - z]

Dígitos 0 - 9

Símbolos especiales - + * / := , . ;. () [] = {} ` espacio en blanco

## Declaración de tipos

Pascal tiene tipos escalares predefinidos los cuales son los siguientes:

- integer: numérico entero
- real: numérico con decimales
- boolean: numérico entero
- char: numérico entero
- string: cadena de caracteres

También se tienen tipos de datos estructurados como:

- Arreglos
- Subrangos
- Records

Sin embargo, el programador puede definir tipos nuevos a partir de los tipos predefinidos y estructurados.

Para declarar un tipo personalizado se debe hacer en el bloque de tipos iniciado por la palabra type, siguiendo la sintaxis siguiente:

```
type
id-de-tipo1, id-tipo-2 = tipo
```

Por ejemplo:

```
type
days, age = integer;
yes, true = boolean;
name, city = string;
fees, expenses = real;
```

### Definiendo un tipo basado en subrangos

```
type
subrango-id = lower-limit ... upper-limit;
```

lower-limit y upper-limit deben ser literales enteras o constantes enteras.

### Definiendo un tipo basado en arreglo

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

Donde index-type indica el rango de datos definido por valores enteros.

por ejemplo

```
type
   vector = array [ 1..25] of real;
```

### Definiendo un tipo basado en record

Pascal permite definir tipos de datos que almacenen múltiples elementos de diferente tipo.

```
type
Books = record
   title: packed array [1..50] of char;
   author: packed array [1..50] of char;
   subject: packed array [1..100] of char;
   book_id: integer;
end;
```

## Declaración de constantes

Las constantes son elementos que no cambian durante la ejecución del programa. Pascal permite definir constantes globales de los siguientes tipos:

- Escalares: integer, real, boolean y char
- cadenas de caracteres

Ejemplo

```
const
P = 18;
Q = 90;
PI = 3.141516;
HOLA_MUNDO = 'hola mundo';
```

## Declaración de variables

Las variable se declaran en un bloque que inicia con la palabra var seguido de los nombres:

```
var
A_Variable, B_Variable ... : Variable_Type;
```

Las variables se declaran fuera del cuerpo de un subprograma o programa principal.

Ejemplo

```
var
age, weekdays : integer;
taxrate, net_income: real;
choice, isready: boolean;
initials, grade: char;
name, surname : string;
```

## Definiendo una variable de tipo subrango

```
var
subrange-name : lowerlim ... upperlim;
```

Donde lowerlim y upperlim son valores enteros, literales o constantes.

Ejemplo

```
var
marks: 1 ... 100;
```

## Definiendo una variable de tipo arreglo

```
var
```

```
   array-name: array [index-type] of element-type;
```

Donde index-type indica el rango de datos definido por valores enteros.

Ejemplo

```
var
   n: array [1..10] of integer;
```

### Definiendo una variable de tipo record

```
var
record-name = record
   field-1: field-type1;
   field-2: field-type2;
   ...
   field-n: field-typen;
end;
```

Ejemplo

```
var
Books : record
   title: packed array [1..50] of char;
   author: packed array [1..50] of char;
   subject: packed array [1..100] of char;
   book_id: integer;
end;
```

## Operadores

### Operadores aritméticos

| Operador | Descripción |
|---|---|
| + | suma dos operandos |
| - | resta dos operandos |
| * | multiplica dos operandos |

| | |
|---|---|
| div | divide numerador entre denominador |
| mod | operación módulo que retorna el sobrante de una división |

Precedencia mayor de * div mod seguido de + -

## Operadores relacionales

| Operador | Descripción |
|---|---|
| = | Compara si dos operandos son iguales o no |
| <> | Compara si dos operandos son diferentes o no |
| > | Compara si primer operando es mayor al segundo |
| < | Compara si primer operando es menor al segundo |
| <= | Compara si primer operando es menor o igual al segundo |
| >= | Compara si primer operando es mayor o igual al segundo |

todos tienen la misma precedencia

## Operadores booleanos

| Operador | Descripción |
|---|---|
| and | si dos operandos son verdaderos, la condición es verdadera |
| and then | igual al and pero se respeta el orden de evaluación y el segundo solo se evalúa si es necesario |
| or | si cualquier operandos es verdadero, la condición es verdadera |
| or else | igual al or pero se respeta el orden de evaluación y el segundo solo se evalúa si es necesario |
| not | invierte el valor del operando |

Precedencia: not > and > or > or else, and then

# Cadenas

Pascal tiene un tipo de dato para manejar cadenas de caracteres llamado ‘string’, sin embargo también se puede definir una cadena de caracteres como un arreglo del tipo char. Los caracteres que pueden ser

incluidos en las cadenas pueden ser números, letras, espacios en blanco, caracteres especiales de pascal o una combinación de todos.

Ejemplo del uso de cadenas

```
program exString;
var
   greetings: string;
   name: array [1..10] of char;
   organization: string;
   message: array [1..9] of char;

begin
   greetings := 'Hello ';
   message := 'Good Day!';

   writeln('Please Enter your Name');
   readln(name);

   writeln('Please Enter the name of your Organization');
   readln(organization);

   writeln(greetings, name, ' from ', organization);
   writeln(message);
end.
```

## Booleans

Pascal tiene un tipo de dato para manejar valores booleanos, sin embargo el tipo Boolean es básicamente un tipo integer.

Un valor 0 equivale a falso, y cualquier valor diferente de cero es considerado verdadero.

Ejemplo

```
program exBoolean;
var
exit: boolean;
```

```
choice: char;
   begin
   writeln('Do you want to continue? ');
   writeln('Enter Y/y for yes, and N/n for no');
   readln(choice);

if(choice = 'n') then
   exit := 1
else
   exit := 0;

if (exit) then
   writeln(' Good Bye!')
else
   writeln('Please Continue');

readln;
end.
```

## Arreglos

Pascal tiene un tipo de dato estructurado para almacenar varios valores del mismo tipo con un tamaño estático. La particularidad en Pascal es que los índices de inicio y fin pueden ser definidos por el programador.

Para acceder a valores dentro de un arreglo se usan índices que deben ser valores enteros.

Ejemplo

```
program exArrays;
var
   n: array [1..10] of integer;   (* n is an array of 10 integers *)
   i, j: integer;

begin
   (* initialize elements of array n to 0 *)
   for i := 1 to 10 do
       n[ i ] := i + 100;   (* set element at location i to i + 100 *)
```

```
    (* output each array element's value *)

   for j:= 1 to 10 do
      writeln('Element[', j, '] = ', n[j] );
end.
```

## Records

Pascal tiene un tipo de dato estructurado para almacenar varios valores de diferente tipo y agruparlos en una entidad específica.

Para acceder a un campo de un record se usa el operador punto .

```
program exRecords;
type
Books = record
   title: packed array [1..50] of char;
   author: packed array [1..50] of char;
   subject: packed array [1..100] of char;
   book_id: longint;
end;

var
   Book1: Books; (* Declare Book1 and Book2 of type Books *)

begin
   (* book 1 specification *)
   Book1.title  := 'C Programming';
   Book1.author := 'Nuha Ali ';
   Book1.subject := 'C Programming Tutorial';
   Book1.book_id := 6495407;

   (* print Book1 info *)
   writeln ('Book 1 title : ', Book1.title);
   writeln('Book 1 author : ', Book1.author);
   writeln( 'Book 1 subject : ', Book1.subject);
   writeln( 'Book 1 book_id : ', Book1.book_id);
```

```
   writeln;
end.
```

## Condiciones

### if-then

Sintaxis: if condition then S

Donde condition es cualquier expresión booleana, y S es un statements. De ser necesarios varios statements entonces se deben agrupar usando el statement begin end.

Ejemplos

```
if (a <= 20) then
   c:= c+1;

if (a <= 20 and x) then
begin
   c:= c+1;
   { more statements }
end;
```

### if-then-else

Sintaxis: if condition then S1 else S2;

Donde condition es cualquier expresión booleana, S1 y S2 son statements. Nótese que después de S1 no se coloca el punto y coma.

Ejemplos

```
if (a <= 20) then
   c:= c+1
else
begin
   c:= c - 1;
   { more statements }
end;
```

## if-then-else-if-else

Se pueden enlazar condiciones else-if.

```
if(boolean_expression 1)then
   S1 (* Executes when the boolean expression 1 is true *)

else if( boolean_expression 2) then
   S2 (* Executes when the boolean expression 2 is true *)

else if( boolean_expression 3) then
   S3 (* Executes when the boolean expression 3 is true *)

else
   S4; ( * executes when the none of the above condition is true *)
```

Solo la statement S4 finaliza con punto y coma

## case

Sintaxis:

```
case (expression) of
   L1 : S1;
   L2 : S2;
   ...
   ...
   Ln: Sn;
else
   Sm;
end;
```

Donde expression puede ser cualquier valor integer, character or boolean.

L1, L2, etc son etiquetas cuyo tipo es el mismo que expression

Ejemplo:

```
program checkCase;
var
   grade: char;
```

```
begin
   grade := 'F';
   case (grade) of
      'A' : writeln('Excellent!' );
      'B', 'C': writeln('Well done' );
      'D' : writeln('You passed' );

   else
      writeln('You really did not study right!' );
    end;

   writeln('Your grade is  ', grade );
end.
```

## Ciclos

### while-do

Sintaxis:

while (condition) do S;

Donde condition es cualquier expresión booleana.

### for-do

Sintaxis:

```
for variable-name := initial_value to final_value do
   S;
```

Donde variable-name es de tipo numérico.

### repeat-until

Sintaxis:

repeat S until condition;

Donde condition es cualquier expresión booleana.

### statement de control break

Palabra reservada que finaliza el ciclo.

```
break;
```

### statement de control continue

Palabra reservada que finaliza la iteración en ejecución.

```
continue;
```

## Funciones

```
function name(argument(s): type1; argument(s): type2; ...): function_type;
local declarations;

begin
   ...
   < statements >
   ...
   name:= expression;
end;
```

## Procedimientos

```
procedure name(argument(s): type1, argument(s): type 2, ... );
   < local declarations >
begin
   < procedure body >
end;
```

## Paso de parámetros

En Pascal, por defecto, el paso de parámetros a subprogramas es por valor. Sin embargo si para algún parámetro se desea el envío por referencia entonces la definición del parámetro debe incluir al inicio la palabra reservada var.

Ejemplo:

```
procedure swap(var x, y: integer);
var
   temp: integer;

begin
   temp := x;
   x:= y;
   y := temp;
end;
```

## Lectura/escritura

### writeln

Statement similar a un subprograma que permite la escritura de valores en pantalla. Puede recibir uno o varios parámetros.

```
writeln('this is a ', varHello, ' ', varWorld, '!!' );
```

Salida en pantalla:

```
this is a Hello world!!
```

### readln

Statement similar a un subprograma que permite la lectura de uno o más valores. Puede recibir uno o varios parámetros.

```
readln(varHello, varWorld);
```

# Importante

- Uso de herramientas jflex y cup para análisis léxicos y sintácticos.
- Usar lenguaje de programación JAVA
- Práctica obligatoria para tener derecho al siguiente proyecto.
- Es válido utilizar algún IDE o cualquier editor de texto.
- Las copias obtendrán nota de cero y se notificará a coordinación.
- No es válido copy/paste desde internet.

# Entrega

La fecha de entrega es el día 09 de septiembre antes de las 14:00 horas por medio de la plataforma Classroom. Los componentes a entregar en repositorio git son:

- Código fuente
- Código compilado (ejecutable)
- Manual técnico incluyendo gramáticas utilizadas.
- Manual de usuario

# Calificación

La calificación se realizará el día 09 de septiembre a partir de las 14:00. El estudiante debe llevar impresa la hoja de registro de punteo que se publicará en los siguientes días.