PRÁCTICO 1
INF329 SA COMPILADORES. GESTIÓN 1-2023

1. *(Teoría)* ¿Qué es un error Léxico? ***Ejemplifique***

2. *(Teoría)* En el siguiente fragmento de código JAVA, se genera un error de "Type mismatch" (tipos no coinciden)

```
String s = "1";
int x = 1 + s;        //Type mismatch
```

¿Qué **fase** del compilador fue el que arrojó este error?

3. (*Esquema de traducción)* Escriba un esquema de traducción para la siguiente construcción ficticia:

```
DO
  Sentencia;
UNTIL ExprBoole1 TOGGLE ExprBoole2;
```

Esta construcción ejecuta la Sentencia en un ciclo. Para salir de este ciclo, al llegar al **UNTIL** :

La primera vez consulta por la ExprBoole1, si es true se sale del ciclo; la segunda vez consulta por la ExprBoole2, si es false se sale del ciclo; la tercera vez consulta por la ExprBoole1, si es true se sale del ciclo; la cuarta vez consulta por la ExprBoole2, si es false sale del ciclo; …

Note que en cada ciclo se consulta por la ExprBoole1 (true para salir) y en el siguiente ciclo por la ExprBoole2 (false para salir), pero jamás se consulta por ambas simultáneamente.

Recuerde que Sentencia, ExprBoole1 y ExprBoole2 generan su propio C3

4. *(Esquema de traducción)* Escriba un esquema de traducción C3 para la construcción ficticia:

```
until (ExprBoole){
      Sentencia;
}norZero (ID);
```

// **ID** = una variable (por ejemplo: **Z**)

Esta construcción hace lo siguiente:

a) Consulta el valor de verdad de ExprBoole. Si es falsa continúa en (b); si es verdadera todo finaliza.
b) Ejecuta la Sentencia.
c) Decrementa el valor del **ID**. Si **ID** es mayor que cero, continúa en (a); caso contrario todo finaliza

Recuerde que cada ExprBoole y Sentencia generan su propio C3.

5. *(Esquema de traducción)* Escriba un esquema de traducción C3 para la construcción ficticia:

```
whileCase (Expr != Expr3){
  case Expr1 : Sentencia1;
  case Expr2 : Sentencia2;
}
```

Esta construcción hace lo siguiente:

a) Verifica que **Expr** ==Expr3. Si son iguales, todo termina; caso contrario, continúa en (b)
b) Verifica que **Expr** == Expr1. Si son iguales, ejecuta la Sentencia1 y luego continúa en (a); caso contrario, continúa en (c).
c) Verifica que **Expr** == Expr2. Si son iguales, ejecuta la Sentencia2 y luego continúa en (a); caso contrario, continúa en (a).

*Recuerde que **Expr**, Expr1, Expr2, Expr3, Sentencia1 y Sentencia2 generan su propio C3.*

6. (*Esquema de traducción)* Escriba un esquema de traducción C3 para la construcción ficticia:

```
forX ID=NUM1 to NUM2 STEP NUM3{
  par
       Sentencia1;
  impar
       Sentencia2
}until (ExprBoole);
```

// **NUM1** y **NUM2** = 2 números enteros (por ejemplo: 3 y 8)
// **ID** = una variable (por ejemplo: **Z**)

Esta construcción **empieza** asignando **ID** = **NUM1**. Luego entra a un ciclo, el cual hace lo siguiente:

a) Si el **ID** es par ejecuta la Sentencia1; caso contrario, ejecuta la Sentencia2.
b) Hace **ID** = **ID** + **NUM3**
c) Si **ID** > **NUM2** O la *ExprBoole* es true, todo finaliza. Caso contrario, continúa en (a).

Rcuerde que ExprBoole, Sentencia1 y Sentencia2, generan su propio C3.

7. *(Código-3)* Escriba un esquema de traducción para la construcción:

```
iftwice (ExprBoole){
  Sentencia1;
}
else {
  Sentencia2;
}
```

Si la ExprBoole es true, Sentencia1 se ejecutará **dos veces** (twice); si es false, se ejecuta la Sentencia2 (una sola vez).

Tome en cuenta que ExprBoole, Sentencia1 y Sentencia2 generan su propio código-3.

8. Convierta el siguiente fragmento JAVA a Código-3

```
while (true){
  read(Dato);
   if (Dato==0)
      break;

  if (Dato%2==0)
      S = S + Dato;
   else
     S = S - Dato;
}

print(S);
```

9. Convierta el siguiente fragmento JAVA a Código-3

```
do{
  read(Dato);
  if (Dato != 100)
       S = S + Dato;
  else
       S = S - Dato;
}while (Dato != 0 && Dato !=100)

print(S);
```

*Use con total libertad las variables **enteras** Dato y S.*

10. Convierta el siguiente fragmento JAVA a Código-3

```
i=0;
c=10;
while (true){ //Loop-forever
    if (i % 2 != 0){
        System.out.print("impar");
        c--;
    }
    else{
       System.out.print("par");
    }

   if (c == 0){
       break;    //Salir del while
   }

   i++;
} //End while

System.out.print("Fin");
```

*Use con total libertad las variables **enteras** i y c.*

DIAGRAMA DE TRANSICIONES

1. *(Diagrama de Transiciones)* Dibujar un dt que, trabajando con Letras y Dígitos, reconozca los siguientes tokens:

NUM → Números enteros (solamente Dígitos).
HEX → Núm. hexadecimales (deben tener al menos una letra hexa y pueden terminar en "h"). Por ejemplo: A12h, 123h, B1, …

ID → Combinación de letras y dígitos que no son NUM o HEX

2. Un lenguaje usa Tokens que son formados con **solamente** Dígitos y Letras. Estos son:

NUM → números enteros
LETNUM → Empieza con Letra(s) y termina con Digito(s).
NUMLET → Empieza con Digito(s) y termina con Letra(s).
ID → (ninguno de los anteriores).

Dibuje un Diagrama de Transiciones, para reconocer a (los nombres de) estos Tokens.

3. *(Diagrama de Transiciones)* Un lenguaje usa Tokens que son formados con **solamente** Dígitos :

NUMP → Números enteros cuya cantidad de dígitos significativos es par
NUMI → Números enteros cuya cantidad de dígitos significativos es impar

Dibuje un dt para reconocer a (los nombres de) estos Tokens.

Por "dígito significativo", en este caso, se refiere a que no se tomen en cuenta los ceros a la izquierda del NUM.

4. *(Diagrama de Transiciones)* Un lenguaje usa Tokens que son formados con **solamente** Dígitos y Letras. Estos son:

NUMO → Números octales, los cuales **pueden** terminar con la vocal '**O**'. Recuerde que los octales solo usan los dígitos 0 al 7.
NUM → Números enteros que no son un NUMO. Opcionalmente Pueden terminar en la letra '**d**'.
ID → combinación de letras y/o dígitos que no son NUM ni NUMO (debe tener al menos una letra)

Dibuje un dt para reconocer a (los nombres de) estos Tokens.
Lexemas de ejemplo:

23456	**NUMO**	(porque solo usa dígitos octales)
23456**d**	**NUM**	(porque son solo dígitos y termina en **'d'**)
7**8**69**5**	**NUM**	(son solo dígitos y usa dígitos **no**-octales)
7563**O**	**NUMO**	(solo usa dígitos octales y termina en 'O')
854O	**ID**	(al utilizar el 8 ya deja de ser octal)
o1234	**ID**	(empieza con letra. No es NUMO ni NUM)
345dd	**ID**	
AB2C	**ID**	

5. *(Diagrama de Transiciones)* Un lenguaje usa Tokens que son formados con **solamente** Dígitos, Letras.

NUM → Números enteros
IDNUM → Alternación estricta de Letras y Dígitos (al menos 2 chars)
ID → Ninguno de los anteriores. (Combinación de Letras, Dígitos y "$")

Dibuje un dt para reconocer a (los nombres de) estos Tokens.
Ejemplos:

234	2d4w5	dd333	a3b5	a4	Z
NUM	IDNUM	ID	IDNUM	IDNUM	ID

/* Alternación estricta: Letra Digito Letra Digito … o
Digito Letra Digito Letra …
*/

6. *(Diagrama de Transiciones)* Un lenguaje toma como tokens a toda subsecuencia formada con **solo Letras**. Estos son:

IDBA → Termina con la subsecuencia "BA".
IDA → Es 'A' o, termina en 'A' pero el anterior char no es 'B'.
ID → subsecuencia de Letras que no es IDBA ni IDA.

Dibuje un dt para reconocer a (los nombres de) estos Tokens.

7. *(Diagrama de Transiciones)* Un lenguaje forma sus tokens con **todas** las cadenas formadas con Letra, Digito y '**.**' . Estos son:

NUM → Números enteros
FLOAT → Números reales informales que terminan en 'F'.
DOUBLE → Números reales informales que **NO** terminan en 'F'.
ID → Combinación de Letras y/o Dígitos, que no es NUM, FLOAT ni DOUBLE.

Note que el punto ('.') puede generar un error: si no forma parte de un FLOAT o un DOUBLE; o si el char que le sigue no es Digito.

Un número es considerado **real** si y solo si está presente el **punto** ('**.**') **decimal**. Se le dice informal, porque se le permite al programador (si él quiere), ignorar la parte entera o la parte decimal, pero no ambas.
Ejemplos: 0.47, **.**47 , 37**.**0, 37**.** , 60.00, 223.876

Dibuje un dt, para reconocer a estos Tokens.

8. *(Aplicación de los dt)* En este contexto, llamamos **palabra** a toda subsecuencia formada con **solamente letras**.

Asumiendo que el cabezal de la Cinta de Caracteres está en la primera celda, dibuje un dt que recorra todas las celdas y devuelva la **palabra** de mayor longitud que está presente en la Cinta. Este dt toma como separador de las **palabras** a todo char que no sea Letra.

Si la Cinta no tiene ninguna **palabra**, su dt devolverá " " (cadena vacía).

*Por comodidad, asuma que todas las **palabras** (si existen) tienen diferente longitud.*

9. *(Aplicación de los dt)* Asumiendo que el cabezal de la Cinta de Caracteres está en la primera celda, dibuje un dt que recorra todas las celdas y devuelva **la cantidad de palabras** que aparecen **dentro** de los comentarios de línea. Los comentarios que se usan son los usados por JAVA (//….)

*Se considera **palabra** a toda subsecuencia que no tiene Espacios ni EOF*

10. *(Aplicación de los dt)* Asumiendo que el cabezal de la Cinta de Caracteres está en la primera celda, dibuje un dt, que recorra todas las celdas y devuelva (return en un estado final), la cantidad de palabras que hay en la Cinta. *Se considera palabra a toda subsecuencia que no tiene Espacios ni EOF*	**11.** *(Aplicación de los dt)* Asumiendo que el cabezal de la Cinta de Caracteres está en la primera celda, dibuje un dt, **de no más de tres estados**, que recorra todas las celdas y devuelva (return en un estado final), la cantidad de palabras que hay en la Cinta. *Se considera palabra a toda subsecuencia que no tiene Espacios ni EOF.*