



# Tecnológico de Monterrey

## **Diseño de Compiladores Propuesta de Lenguaje: Aang**

---

**Ana Jimena Lomeli Cantu**  
**A00818665**

---

**Jorge Andrés Sabella Elizondo**  
**A01282292**

Monterrey N.L. March 9, 2020

### **1. Visión**

Nuestra visión es hacer un lenguaje de programación que sea sencillo de usar para todas las personas que quieran aprender a programar. Nuestro lenguaje de programación *Aang* está inspirado en la historia de Avatar: La leyenda de Aang, esto con el objetivo de motivar y atraer a los estudiantes que usarían el lenguaje desarrollado en el curso de Diseño de Compiladores. Esto pensando que no tienen conocimientos previos sobre programación.

## 2. Objetivo

### 2.1. Categoría

El objetivo de este proyecto es desarrollar un compilador de un lenguaje imperativo el cual cuenta con la sintaxis necesaria para enseñar los conceptos básicos de la programación, así como una representación visual del mismo.

## 3. Requerimientos del Lenguaje

### 3.1. Elementos Básicos

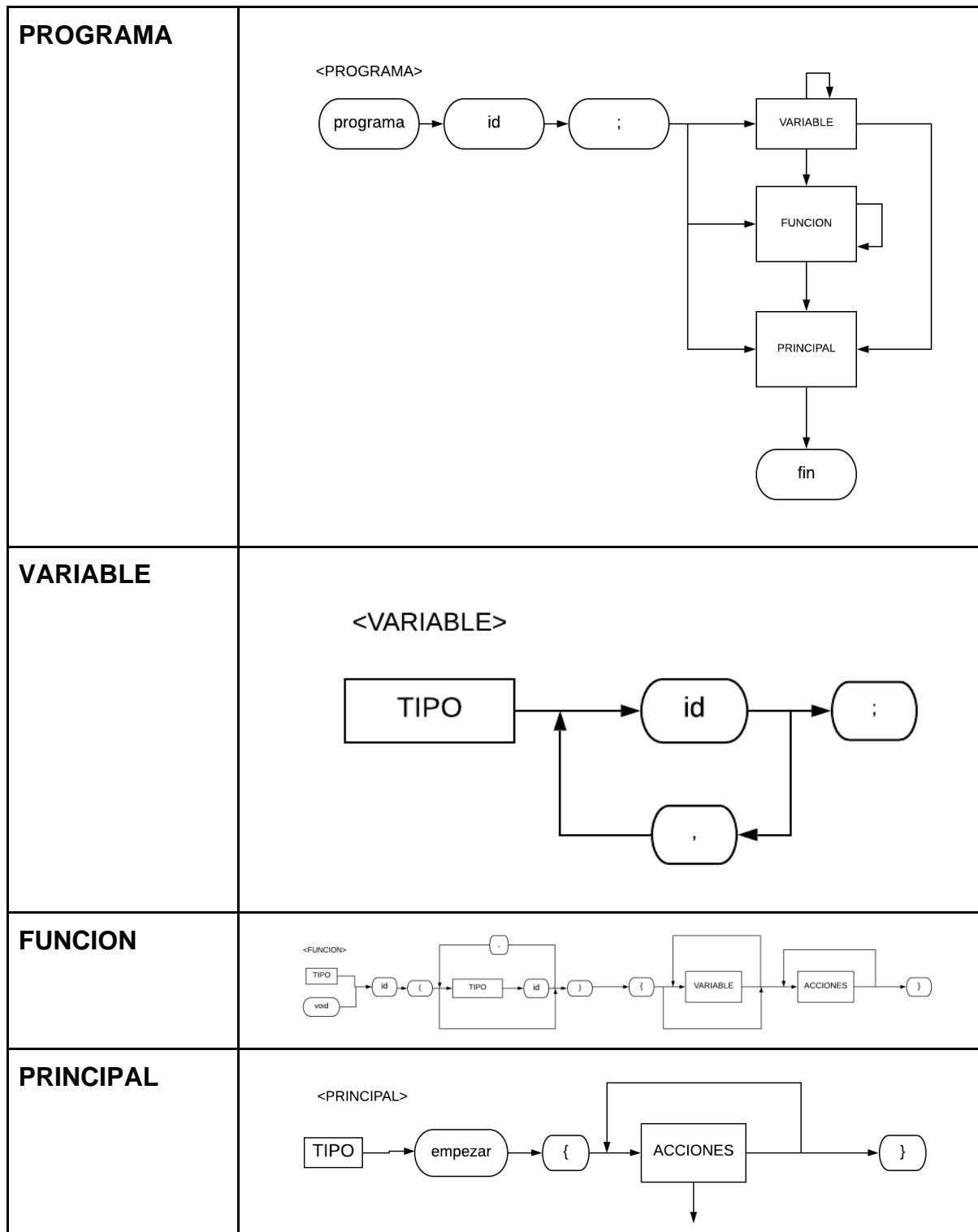
- |      |            |
|------|------------|
| • +  | • char     |
| • -  | • int      |
| • *  | • void     |
| • /  | • si       |
| • =  | • entonces |
| • <  | • sino     |
| • >  | • mientras |
| • == | • imprimir |
| • != | • regresar |
| • (  | • funcion  |
| • )  | • id       |
| • {  | • fin      |
| • }  |            |
| • ;  |            |
| • ,  |            |

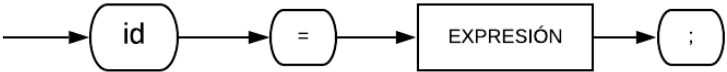
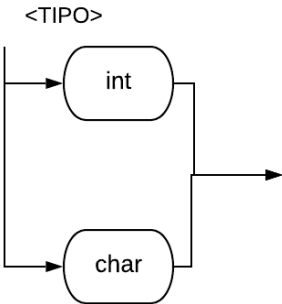
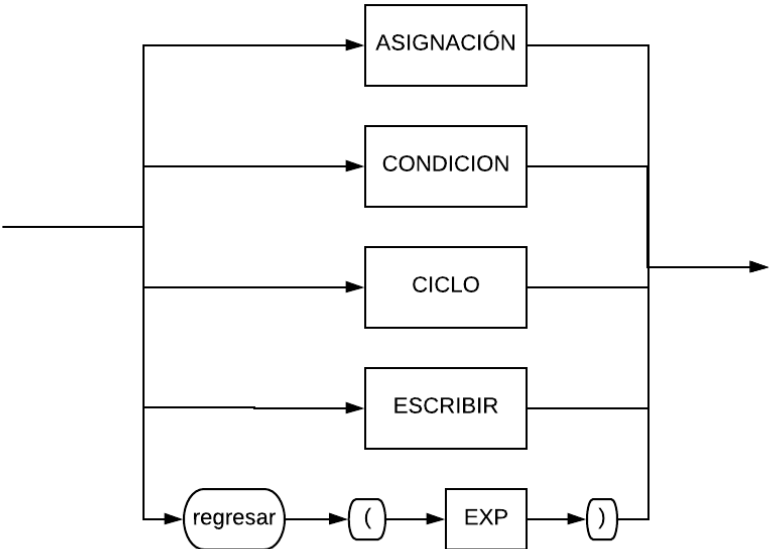
### 3.2. Estructuras

- |             |            |
|-------------|------------|
| • PROGRAMA  | • ESCRIBIR |
| • FUNCION   | • FACTOR   |
| • VARIABLE  | • TERMINO  |
| • PRINCIPAL | • FUNCION  |
| • TIPO      | • ACCION   |
| • BLOQUE    | • VAR_CTE  |
| • CONDICION | • CICLO    |
| • EXPRESION |            |

- EXP

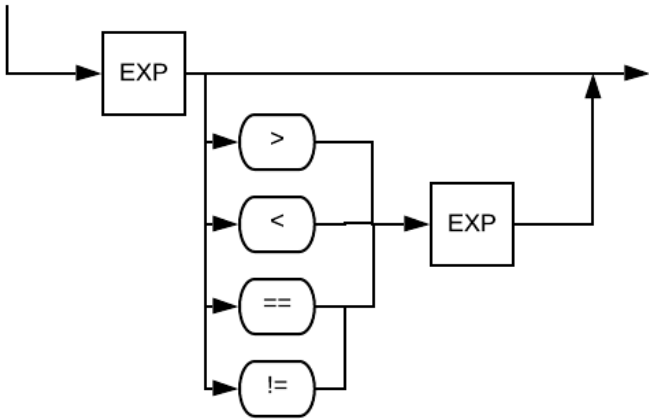
### 3.3. Diagramas de Sintaxis



<b>ASIGNACION</b>	<p>&lt;ASIGNACIÓN&gt;</p> 
<b>TIPO</b>	<p>&lt;TIPO&gt;</p> 
<b>ACCIONES</b>	<p>&lt;ACCIONES&gt;</p> 

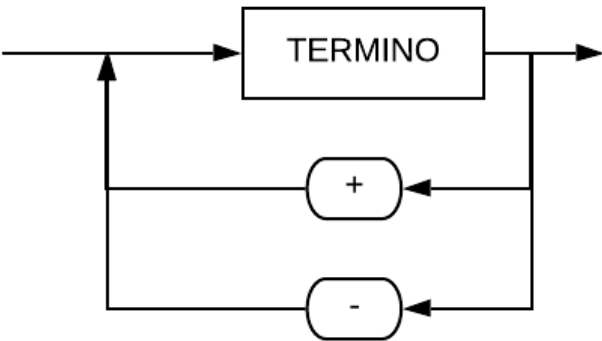
**EXPRESION**

<EXPRESIÓN>



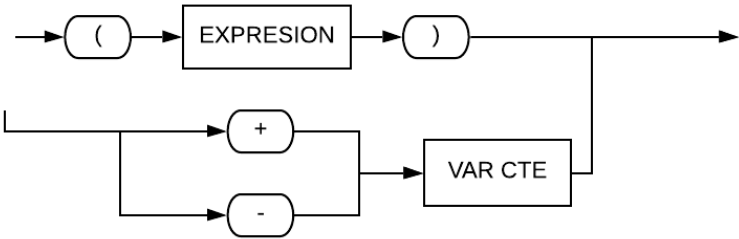
**EXP**

<EXP>



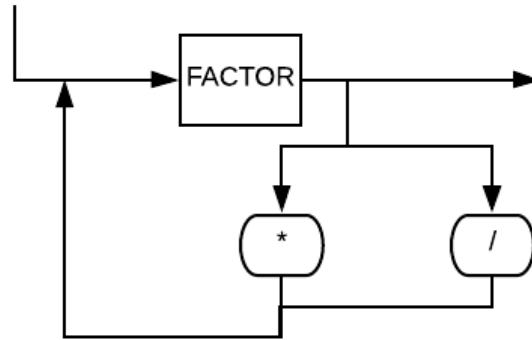
**FACTOR**

<FACTOR>



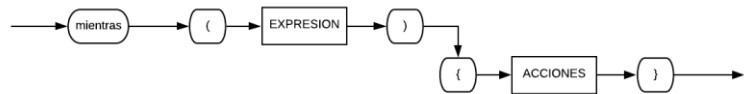
**TERMINO**

<TERMINO>



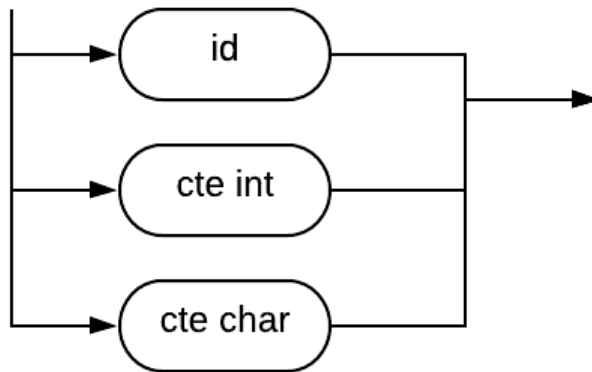
**CICLO**

<CICLO>



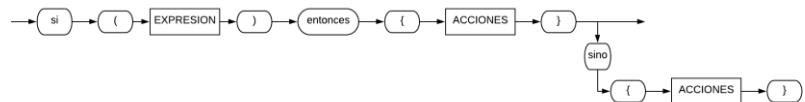
**VAR\_CTE**

<VAR CTE>

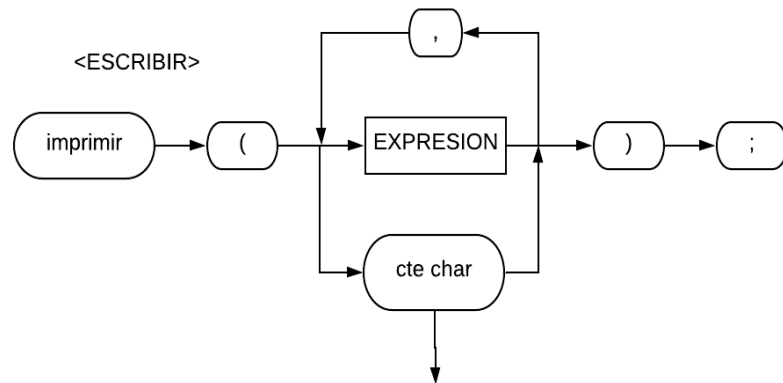


**CONDICION**

<CONDICION>



## ESCRIBIR



### 3.4. Características Semánticas

- Las expresiones aritméticas pueden ser utilizadas únicamente con tipos de dato *int*.
- Se debe definir el nombre del programa al inicio.
- Se utiliza { } para definir una función o un bloque.
- Ejemplos Aritméticos

```
programa Ejemplo1;  
    int empezar {  
        regresar(2+2);  
    }  
fin
```

```
programa Ejemplo2;  
    int a,b;  
    int empezar {  
        a=1;  
        b=1;  
        regresar(a+b);  
    }  
fin
```

```
programa Ejemplo3;  
    int a,b,c;  
    int multiplicar (int a, int b) {  
        regresar (a*b);  
    }  
  
    int empezar {  
        a=1;  
        b=1;  
        regresar(multiplicar(a,b));  
    }  
fin
```

- Ejemplos de Ciclos

```

programa Ejemplo4;
  int a,b;

  int empezar {
    a=10;
    b=1;
    mientras(a>b) {
      a = a-1;
    }
    regresar(0);
  }
fin

```

- Ejemplos de Condiciones

```

programa Ejemplo4;
  int a,b;

  int empezar {
    a=10;
    b=1;
    si(a!=b) entonces {
      imprimir(a*b);
      a = a-1;
    } sino {
      regresar(a/b);
    }
    regresar(0);
  }
fin

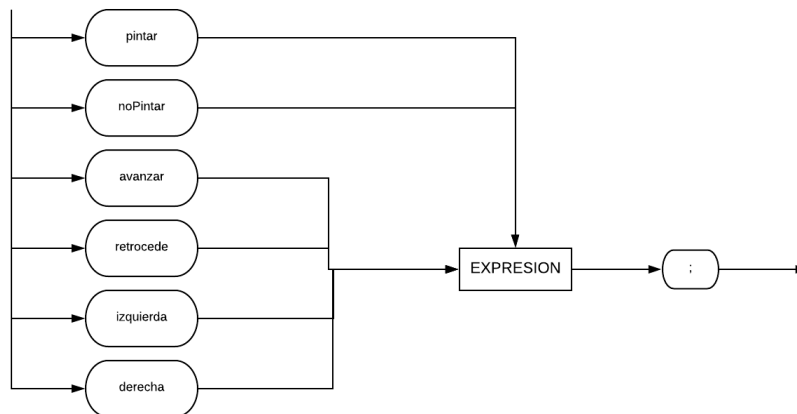
```

### 3.5. Funciones Especiales

- pintar(): se colorea el espacio en el que se encuentra el personaje.
- noPintar(): no se colorea el espacio en el que se encuentra el personaje.
- derecha(): el personaje gira a la derecha.
- izquierda(): el personaje gira a la izquierda.
- avanzar(): el personaje avanza hacia la dirección en donde apunta.
- retrocede(): el personaje retrocede al último espacio donde estuvo colocado.



<MOVIMIENTOS>



### 3.6. Tipos de Datos

- Int
- Char

## 4. Lenguaje y SO

El lenguaje de programación especificado se desarrollará utilizando como analizador léxico y sintáctico ANTLR con Python y MacOS/Windows para el desarrollo del sistema operativo y las pruebas.

## 5. Bibliografía

- Little Duck 2020

A continuación se Describen las características generales del lenguaje gráfico que se desarrollará durante los próximas semanas.

La estructura general de un programa escrito en Aang es:

```
programa Ejemplo3;  
    <Declaracion de Variables Globles>  
    <Definicion de Funciones>  
  
    ## procedimiento principal... comentario  
    int empezar {  
        <Estatutos>  
    }  
    fin
```

\* Las palabras y simbolos en bold son reservados y el ## indica comentario

\* Las secciones en color verde son opcionales.

#### Declaracion de Variables: (globales y locales)

sintaxis:

```
var ##Palabra reservada  
tipo lista_ids;  
<tipo lista ids;> etc...
```

donde

tipo = **int y char**.

lista\_ids = identificadores separados por comas, tienen maximo una dimension [N] de 0 a N-1.

Ej: **int** id1[cte-entera], id2, id3;

Se definen 3 variables enteras donde la primera tiene una dimension de tamaño N.

#### Declaracion de Funciones: (Se pueden definir 0 o más funciones)

sintaxis:

```
<tipo-retorno> id (<Parametros>)  
{  
    <Variable>  
    <Acciones>  
}
```

Los parámetros siguen la sintaxis de la declaración de variables simples (no dimensionadas) y son únicamente de entrada.

Tipo-retorno puede ser de cualquier tipo soportado o bien void (si no regresa valor)

#### Para los Estatutos/Acciones:

La sintaxis básica de cada uno de los estatutos en el lenguaje Aang es:

##### **Asignación**

Id <dimensión> = Expresión;

A un identificador (que pudiera ser simple o ser una casilla de un elemento dimensionado) se le asigna el valor de una expresión. Cabe aclarar que siempre, a excepción de en la declaración, las Dimensiones son Expresiones aritméticas.

**Id** <dimension> = Nombre\_funcion(<parametros>);

A un identificador, se le asigna el valor que regresa una función.

O bien, pudiera ser algo como: Id<dimension> = Nombre\_funcion(<parametros>) + Id <dimension>-cte

A un identificador se le puede asignar el resultado de una expresión en donde se invoca a una función.

## Llamada a una función Void

**Nombre\_Funcion**(<parámetros>);

Se manda llamar una función que no regresa valor (caso de funciones void).

## Retorno de una función

**regresar** ( exp )

El Estatuto va dentro de las funciones e indica el valor de retorno (si no es void)

## Lectura

Leer(id<dimension>, id<dimension>,...);

Se puede leer uno o mas identificadores (con o sin dimensiones) separados por comas.

## Escritura

imprimir ("letrero" o expresión<"letrero" o expresión...>);

Se pueden escribir letreros y/o resultados de expresiones separadas por comas.

## Estatuto de Decisión (puede o no venir un "sino")

**Si**(<expresión>) **entonces**

{<Acciones>}

**Sino**

{<Acciones>}

## Estatutos de Repetición

### Ciclo

**mientras**(<expresión>)

{<Acciones>}

Repite acciones hasta que la expresión sea verdadera.

### Ciclo-No-Condicional

**desde** Id<dimensiones> = exp **hasta** exp **hacer**

{acciones}

Repite desde N hasta M brincando en 1 en 1.

## Expresiones

Las expresiones en Aang son las tradicionales (como en C y en Java). Existen los operadores aritméticos, lógicos y relacionales: +, -, \*, /, &(and), |(or), <, >, ==, etc. Se manejan prioridades tradicionales, se pueden emplear para alterarla.

En **Aang** existen identificadores, palabras reservadas, constantes enteras y constantes char.