Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Organización de Lenguajes y Compiladores 1

Vacaciones de diciembre de 2019

Catedrático: Ing. Mario Bautista

Auxiliar: Nery Galvez

NMScript

Tabla de contenido

OBJETIVOS **3**

OBJETIVO GENERAL 3
OBJETIVOS ESPECÍFICOS 3

DESCRIPCIÓN DE LA SOLUCIÓN **3**

FLUJO DE LA APLICACIÓN **3**

INTERFAZ GRÁFICA **4**

CARACTERÍSTICAS DEL EDITOR 4
MENÚ DE ARCHIVOS 4
MENÚ DE EJECUCIÓN 5
MENÚ DE REPORTES 5
ÁREA DE SALIDA 5

DEFINICIÓN DEL LENGUAJE **5**

ARCHIVOS NMSCRIPT 5
COMENTARIOS 5
Comentario de una línea *5*
Comentarios de varias líneas *6*
TIPOS DE DATOS 6
Tipos de datos primitivos *6*
Tipos de datos por referencia *6*
CARACTERES DE ESCAPE 6
EXPRESIONES 6
Signos de agrupación *6*
Literales *7*

Incremento *7*
Decremento *7*
Expresiones aritméticas *7*
Suma 7
Resta 8
Multiplicación 8
División 8
Potencia 9
Modulo 9
Expresiones relacionales *9*
Mayor que, Menor que, Mayor o igual que, Menor o igual que 9
Igual que, diferente que 10
Expresiones lógicas *10*
Or 10
And 11
Xor 11
Not 11
DECLARACIÓN DE VARIABLES 11
ASIGNACIÓN DE VARIABLES 11
SENTENCIAS DE CONTROL 12
Sentencia IF *12*
Sentencia SWITCH *12*
Sentencia FOR *13*
Sentencia WHILE *13*
Sentencia DO-WHILE *13*
Sentencia BREAK *14*
Sentencia CONTINUE *14*
SENTENCIA IMPORTAR 15
MÉTODOS Y FUNCIONES 15
Método *15*
Función *15*
Parámetros de un método o función *15*
Llamada a métodos y funciones *16*
Sentencia return *16*
Constructores *16*
Método main *17*
Sobre carga de métodos, funciones y constructores *17*
FUNCIONES NATIVAS 18
Println *18*
Print *18*
Graficar_dot *18*
Graficar_entornos *18*
ARREGLOS 19
Declaración de arreglo *19*
Acceso a un arreglo *19*
Asignación a un arreglo *19*
Atributo size en un arreglo *19*
CLASES 20
Declaración de una clase *20*
Instancia de una clase (Objeto) *20*

Acceso a los elementos de un objeto *21*
Asignación a los elementos de un objeto *21*
this *21*
PALABRAS RESERVADAS 22
PRECEDENCIA Y ASOCIATIVIDAD DE OPERADORES 22
RESTRICCIONES DEL PROYECTO 22
FORMA DE ENTREGA DEL PROYECTO 23
FASE 1 DEL PROYECTO 23
Funcionalidades a entregar *23*
Entregables *23*
Fecha de entrega *23*
FASE 2 DEL PROYECTO 23
Funcionalidades a entregar *23*
Entregables *23*
Fecha de entrega *23*

Objetivos

Objetivo general

Implementar una aplicación donde se apliquen los conceptos adquiridos en el curso de Organización de Lenguajes y Compiladores 1.

Objetivos específicos

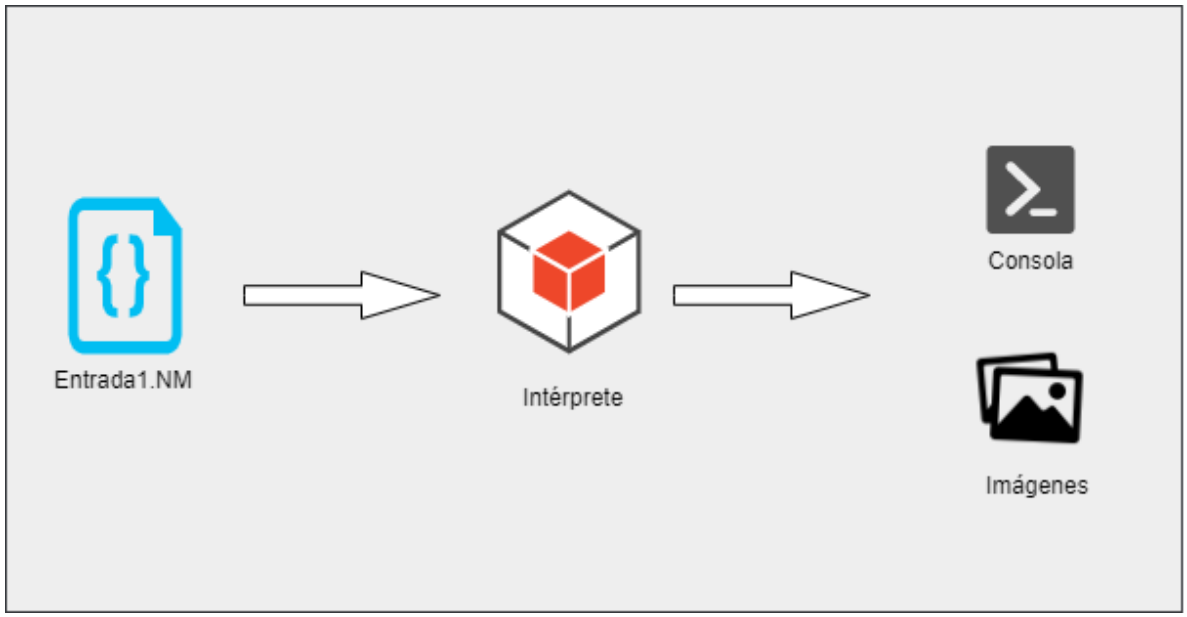
- Aplicar la fase de análisis léxico, sintáctico y semántico para la implementación de un intérprete.
- Construir un árbol de sintaxis abstracta (AST) para la ejecución de código de alto nivel.
- Detectar y reportar errores léxicos, sintácticos, semánticos y de ejecución.

Descripción de la solución

El estudiante tiene que implementar un intérprete que ejecute instrucciones de alto nivel definidas en el lenguaje **NMScript**, el cual se define más adelante. Para la implementación se tiene que crear un AST, el cual servirá para la ejecución de las instrucciones del lenguaje.

Flujo de la aplicación

- Entrada: la aplicación tendrá como entrada el contenido de un archivo con código **NMScript**.
- Procesamiento: el intérprete **NMScript** tomará la cadena de entrada, realizará análisis léxico, sintáctico, creará un AST para posteriormente ejecutar cada una de las instrucciones definidas.
- Salida: la aplicación puede generar una salida en consola, una salida por medio de la generación de imágenes o ambas.



Interfaz gráfica

La aplicación deberá de contar con una interfaz gráfica amigable, la cual permitirá interactuar con el intérprete, manipular archivos de entrada y generar reportes, cada uno de sus elementos se describe a continuación:

Características del editor

- Tendrá múltiples pestañas.
- Tendrá autocompletado de palabras.
- Debe de mostrar la fila y columna actual del cursor dentro de una pestaña.
- Se debe de pintar los tokens de la pestaña actual, el color de cada token se define en la siguiente tabla:

Token	Color
Palabras reservadas	Azul
Cadenas, caracteres	Naranja
Números	Morado
Comentarios	Gris
Otro	Negro

Menú de archivos

- Abrir: permitirá cargar el contenido de un archivo a una pestaña dentro del editor.
- Guardar: guardará en un archivo la cadena de texto que se encuentre en la pestaña actual.
- Guardar como: guardará en un nuevo archivo la cadena de texto que se encuentre en la pestaña actual.
- Crear: creará una nueva pestaña dentro del editor, la pestaña no tendrá contenido.

Menú de ejecución

- Ejecutar: se tomará la cadena que se encuentre en la pestaña actual y se realizará el proceso de interpretación.

Menú de reportes

- Reporte del AST: en este reporte se mostrará una imagen con el AST que se genera al analizar la cadena de entrada de la pestaña actual.
- Reporte de errores: en este reporte se mostrarán todos los errores que ocurran durante la interpretación. El reporte debe de ser generado en un archivo HTML en forma de tabla, el contenido mínimo del reporte es el siguiente:

No.	Tipo	Descripción	Fila	Columna
1	Léxico	Carácter '^' incorrecto	5	10
2	Sintáctico	No se esparaba el componente **contador**	15	1
3	Semántico	Operación incorrecta: Cadena / Booleano	20	5
4	De ejecución	División por cero	21	2

Área de salida

El editor debe de contar con una consola en la cual se mostrará la salida de la función Imprimir.

Definición del lenguaje

Archivos NMScript

Los archivos con código NMScript tendrá extensión **[.NM]**, el contenido de cada archivo se define a continuación:

- Una lista de importaciones, las importaciones no necesariamente vienen en la parte de arriba del archivo, pueden venir en cualquier parte e incluso puede venir un archivo sin importaciones.
- Puede venir la definición de una y solo una clase dentro del archivo.

Comentarios

Comentario de una línea

Los comentarios de una línea iniciarán con "//" y terminarán con un salto de línea.

```
// Este es un comentario de una línea

// Este es un comentario de una línea
```

Comentarios de varias líneas

Los comentarios de varias líneas iniciarán con "/*" y terminarán con "*/".

```
/**************************
***** COMPILADORES 1 ********
**************************/
```

Tipos de datos

El lenguaje NMScript es un lenguaje con tipado fuerte, esto quiere decir que se conoce el tipo de dato de una variable desde el momento en que fue declarada. También tiene un tipado estático, lo que indica que una variable o expresión puede tener solo un tipo durante toda la ejecución.

Tipos de datos primitivos

Tipo de dato	Ejemplo
int	100, 024, 1, - 450, -5
double	0.2455, -6.245, -4.0
char	'a', '', ' ', '%'
boolean	true, false
String	"Compi1", "", "20", "- 4.45", "%%%#"
null	null

Tipos de datos por referencia

- Arreglos
- Objetos

Caracteres de escape

Los caracteres de escape son caracteres que se definen de forma especial dentro de una cadena de texto (String). A continuación, se definen los caracteres que se tienen que manejar.

Carácter de escape	Descripción
\"	Comillas dobles
****	Barra invertida
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal

Expresiones

Signos de agrupación

Los signos de agrupación se utilizarán para dar orden y cierta precedencia a las operaciones. Se utilizarán los paréntesis "(4 + 5) * 3".

Literales

Los literales hacen referencia a las expresiones donde su tipo de dato es primitivo.

Literal	Ejemplo
Entero	100, - 42, 6
Decimal	-0.5482, 5.1247, - 0.0
Booleano	true, false
Carácter	‘’, ‘(‘, “$’
Cadena	“Compiladores 1”, “”, “100”, “,#,”

Incremento

Esta expresión devuelve el valor actual de la variable y posteriormente aumenta en uno su valor. Esto es exclusivamente para identificadores.

```
Int a = 100;

a++;

double b = a++ * 4;
```

Decremento

Esta expresión devuelve el valor actual de la variable y posteriormente disminuye en uno su valor. Esto es exclusivamente para identificadores.

```
Int a = 100;

a--;

double b = a-- * 4;
```

Expresiones aritméticas

A continuación, se presenta el sistema de tipos de las operaciones aritméticas. Para los casos donde se opere con un tipo de dato **char** se tomará el valor del código ASCII del carácter.

Suma

El operador será es el signo más [+]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	int
int	double	double
int	char	int
int	String	String
double	int	double
double	double	double
double	char	double
double	String	String

char	int	int
char	double	double
char	char	int
char	String	String
String	int	String
String	double	String
String	char	String
String	String	String
String	Boolean	String

Resta

El operador será es el signo más [-]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	int
int	double	double
int	char	int
double	int	double
double	double	double
double	char	double
char	int	int
char	double	double
char	char	int

Multiplicación

El operador será es el signo más [*]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	int
int	double	double
int	char	int
double	int	double
double	double	double
double	char	double
char	int	int
char	double	double
char	char	int

División

El operador será es el signo más [/]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	int
int	double	double

int	char	int
double	int	double
double	double	double
double	char	double
char	int	int
char	double	double
char	char	int

Potencia

El operador será es el signo más [**]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	double
int	double	double
int	char	double
double	int	double
double	double	double
double	char	double
char	int	double
char	double	double
char	char	double

Modulo

El operador será es el signo más [%]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2	Tipo de dato resultado
int	int	double
int	double	double
int	char	double
double	int	double
double	double	double
double	char	double
char	int	double
char	double	double
char	char	double

Expresiones relacionales

A continuación, se presenta el sistema de tipos de las operaciones relacionales. El resultado de estas operaciones será un valor de tipo booleano (true o false).

Mayor que, Menor que, Mayor o igual que, Menor o igual que

Los operadores de estas operaciones serán los siguiente [>, <, >=, <=]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2
int	int
int	double
int	char
double	int
double	double
double	char
char	int
char	double
char	char

Igual que, diferente que

Los operadores de estas operaciones serán los siguiente [==, !=]. A continuación, se presenta la tabla de tipos de esta operación.

Operando 1	Operando 2
int	int
int	double
int	char
double	int
double	double
double	char
char	int
char	double
char	char
String	String
Boolean	Boolean
Objeto	Objeto
Objeto	null
null	Objeto
null	null

Expresiones lógicas

A continuación, se presenta la tabla de verdad de las operaciones lógicas. El resultado de estas operaciones será un valor de tipo booleano (true o false). Los operadores que reciben estas expresiones tienen que ser de tipo booleano.

Or

El operador será definido por los caracteres [||]. Cabe mencionar que esta operación se realiza en modo de corto circuito, lo que quiere decir que sí el primer operando es verdadero ya no se evalúa el segundo operando porque ya se sabe que la condición es verdadera.

And

El operador será definido por los caracteres [&&]. Cabe mencionar que esta operación se realiza en modo de corto circuito, lo que quiere decir que sí el primer operando es falso ya no se evalúa el segundo operando porque ya se sabe que la condición es falsa.

Xor

El operador será definido por los caracteres [^].

Not

El operador será definido por los caracteres [!].

X	Y	X || Y	X && Y	X ^ Y	!X
false	**false**	false	false	false	true
false	**true**	true	false	true	true
true	**false**	True	false	true	false
true	**true**	true	true	false	false

Declaración de variables

Esta instrucción permitirá guardar en tabla de símbolos una variable con su respectivo valor. A continuación, se presenta un ejemplo de cómo se utiliza esta instrucción.

```
int contador = suma(x , y);

Nodo raíz = new Nodo(100);

String nombre = getNombre() + "_" + contador;
```

Asignación de variables

Esta instrucción permitirá modificar el valor de una variable que fue previamente declarada. A continuación, un ejemplo de su uso.

```
int contador = suma(x , y);

contador = contador--;

Nodo n = new Nodo(100);

n.izquierdo.valor = 10;
```

Sentencias de control

Sentencia IF

Esta instrucción contendrá una lista de condiciones las cuales serán evaluadas para ver qué condición es la que se cumple, de cumplirse con una condición se ejecutará el bloque de instrucciones de la condición. En esta instrucción se puede definir un caso que se ejecute cuando ninguna de las condiciones se cumple, este caso es el **ELSE**. A continuación, un ejemplo de su uso.

```
If( a == 10 && true){
        /* INSTRUCCIONES */
}else if(false){
        /* INSTRUCCIONES */
}else{
        /* INSTRUCCIONES */
}
```

```
If(raiz != null){
        /* INSTRUCCIONES */
}
```

Sentencia SWITCH

Esta instrucción recibirá una expresión de control y una lista de casos (opciones). Esta sentencia comparará el valor de la expresión de control con el valor del caso, si estos valores coinciden se ejecuta el bloque de instrucciones de ese caso. Si dentro del caso que se ejecuta no se encuentra una instrucción break se ejecutarán el resto de casos que se encuentren debajo sin realizar la comparación entre la expresión de control y el valor del caso, sí se definiera un caso **default** y este está debajo del caso que se cumple también se ejecuta.

Dentro de esta sentencia también se puede definir un caso por defecto, el cuál será ejecutado cuando ninguno de los casos coincida con el valor de la expresión de control. Este caso por defecto puede venir en cualquier parte de la instrucción Switch, no necesariamente al final. A continuación, un ejemplo de su uso.

```
switch ( a ){
        case “Mate1”:
            //Instrucciones
        case “Compi1”:
            //Instrucciones
}
```

```
switch (“Hola”){
        Case “hola”
            //Instrucciones

        default:
            //Instrucciones
}
```

Sentencia FOR

Esta sentencia estará conformada por lo siguiente:

- Inicialización: se podrá declarar cualquier tipo de variable o realizar una asignación.
- Condición: será la expresión que se verificará en cada iteración para decidir si se ejecuta el bloque de instrucciones o no.
- Actualización: esta parte permite actualizar la variable de control del ciclo. Se puede definir una asignación de variable, incremento o decremento.

```
for (double i = 0.0 ; i < 10; i = i + 0.5){
        //Instrucciones
}
```

Sentencia WHILE

Esta instrucción ejecutará un conjunto de instrucciones mientras el resultado de su expresión de control sea verdadero.

```
While( true){
        //Instrucciones
}
```

```
While(a != 15 || b < 10){
        //Instrucciones
}
```

Sentencia DO-WHILE

Esta instrucción al igual que el WHILE ejecutará un conjunto de instrucciones mientras el resultado de su expresión de control sea verdadero, la diferencia es que esta instrucción ejecuta su bloque de instrucciones una vez sin evaluar la condición, la condición se empieza a evaluar a partir de la segunda iteración.

```
do{
        //Instrucciones
} while (x != null) ;
```

Sentencia BREAK

Esta instrucción permite alterar el flujo de ejecución del programa, la sentencia BREAK tendrá los siguientes usos:

- Terminar la ejecución del ciclo más cercano.
- Terminar la ejecución de una instrucción SWITCH.

```
while(true){
        if(a == “Compi1”){
                break;
        //Código inaccesible
          }
}
```

Nota: se debe de verificar que esta instrucción se encuentre dentro de un ciclo o dentro de un SWITCH.

Sentencia CONTINUE

Esta instrucción permite alterar el flujo de ejecución del programa, la sentencia CONTINUE tendrá los siguientes usos:

- Pasar a la siguiente iteración del ciclo más cercano.

```
while(true){
        if(a == “Compi1”){
                continue;
        //Código inaccesible
          }
}
```

Nota: se debe de verificar que esta instrucción se encuentre dentro de un ciclo.

Sentencia Importar

Esta instrucción permitirá importar código desde otros archivos para poder utilizar los elementos del archivo importado.

```
Import “Entradas/e1.NM”;

Import “e2.NM”;
```

Métodos y funciones

Método

Se puede definir a un método como un conjunto de instrucciones dentro de un bloque al cual se le define un nombre y el cual puede ser invocado desde otra parte del programa. Los métodos no retornan valores al momento de invocarlos como lo hacen las funciones.

```
Void metodo1(int a, Nodo [][][] arreglo, Arbol ab){
        // Instrucciones
}
Void metodo2(){
        //Instrucciones
}
```

Función

Se puede definir a una función como un conjunto de instrucciones dentro de un bloque al cual se le define un nombre y la cual puede ser invocada desde otra parte del programa. A diferencia de los métodos las funciones siempre devuelven un valor al momento de invocarlas.

```
Nodo funcion1(int a, Nodo [][][] arreglo, Arbol ab){
        // Instrucciones
}
Nodo[][][] funcion2(){
        //Instrucciones
}
```

Parámetros de un método o función

Los parámetros son variables que se definen para un método o función, al realizar una llamada se deben de enviar valores a los parámetros definidos. Cabe resaltar que una función puede o no tener parámetros.

Llamada a métodos y funciones

Una llamada es una invocación a ejecutar las instrucciones definidas dentro de un método o función.

```
metodo1();

nodo.insertar(10);

int contador = getContador();

Nodo raiz = arbol.getRaiz();
```

Sentencia return

Esta instrucción permite alterar el flujo de ejecución del programa, existen dos formas de utilizar esta instrucción:

- **return**: para terminar de ejecutar un método. Se debe de validar el posible error semántico donde la instrucción venga dentro de una función y no dentro de un método.
- **return EXPRESION**: para terminar de ejecutar una función. Se debe de validar el posible error semántico donde la instrucción venga dentro de un método y no dentro de una función.

```
return ;

return new Nodo (19,” Compi1”);

return null;

return “Hola” + “mundo” + “;”;
```

Constructores

Los constructores son funciones que se ejecutan al momento de instanciar un objeto y tienen las siguientes características:

- Pueden o no tener parámetros.
- Se encuentran dentro de una clase:
- Se debe validar que el nombre del constructor sea igual que el nombre de la clase en la que se encuentra.
- Una clase puede tener cero o varios constructores.

```
class Nodo {

        //Constructores

        Nodo(){  }

        Nodo(int valor, String nombre){  }


}
```

Método main

El método main será el encargado de iniciar la ejecución del programa, este tiene las siguientes características:

- La definición es la de un método normal, la palabra **main** NO es palabra reservada.
- Este método no recibe parámetros.
- Este método se buscará en la clase definida en el archivo principal a la hora de iniciar la ejecución.

```
Class Main{
        /* Aquí inicia la ejecución del programa */
        void main(){
        }
        //Este NO es el método donde inicia la ejecución
        void main(int a){
        }
}
```

Sobre carga de métodos, funciones y constructores

La sobre carga de métodos permite definir métodos con el mismo nombre, pero que tiene diferente cantidad de parámetros o que tiene la misma cantidad, pero el tipo de los parámetros es diferente.

```
Void metodo1(int a, Nodo [][][] arreglo, Arbol ab){
        // Instrucciones
}
Void metodo2(){
        //Instrucciones
}
```

Funciones nativas

Estas son funciones propias del lenguaje NMScript, su funcionamiento se define más adelante.

Println

Esta función recibe de parámetro una expresión e imprime en la consola de la aplicación el resultado, al final del valor de la expresión se agrega un salto de línea.

```
Println(“Hola” + “mundo” + 100);

Println(arreglo[3][2][1][4] + “ – “ + “!!”);
```

Print

Esta función recibe de parámetro una expresión e imprime en la consola de la aplicación el resultado, a diferencia de **Println** esta NO agrega un salto de línea al final del valor de la expresión.

```
Print(“Hola” + “mundo” + 100);

Print(arreglo[3][2][1][4] + “ – “ + “!!”);
```

Graficar_dot

Esta función genera una imagen utilizando el programa Graphviz, los parámetros que recibe son los siguientes:

- Nombre de la imagen: expresión de tipo cadena con el nombre con el que se guardará la imagen a generar.
- Contenido dot: expresión de tipo cadena con el contenido dot con el cual se va a generar la imagen.

```
String ruta = “imagen1.jpg”;

String contenido = "digraph D { A -> {B, C, D} -> {F} }";

Graficar_dot(ruta, contenido);
```

Graficar_entornos

Esta función generará una imagen de Graphviz donde se vean todos los entornos que son accesibles hasta el momento.

```
Graficar_entornos();

While(true){

        Graficar_entornos();

}
```

Ejemplo de la información que debe tener un entorno:

Tipo	Nombre	Valor
Int	Contador	15
Persona	Arreglo_personas	Persona[]

Arreglos

Dentro del lenguaje se podrán manejar arreglos, estos arreglos pueden tener una profundidad de N. A continuación, se muestra cómo es el uso de arreglo.

Declaración de arreglo

```
Nodo [] [] [] arreglo1 = new Nodo [2][2][2];

int [] arreglo2 = {10, 5, -240, -54789};

String [] [] = getCadenas();
```

Acceso a un arreglo

```
Println( arreglo[0][1][4]);

int valor = Nodos[0].izquierdo.valor;
```

Asignación a un arreglo

```
Nodo [] [] [] arreglo1 = new Nodo [2][2][2];

int [] arreglo2 = {10, 5, -240, -54789};

arreglo1[0][0] = new Nodo[2]; //Cambiando valor

arreglo2[2] = 8173; //Cambiando valor
```

Atributo size en un arreglo

Una vez definido un arreglo se puede obtener su tamaño, esto se hará accediendo al atributo size del arreglo, este es un atributo propio de todos los arreglos.

```
Nodo [] [] [] arreglo1 = new Nodo [2][3][4];

int [] arreglo2 = {10, 5, -240, -54789, 1, 3, 4};

Println(arreglo1.size); // Tamaño 2

Println(arreglo1[0].size); //Tamaño 3

Println(arreglo2.size); //Tamaño 7
```

Clases

Declaración de una clase

Las clases servirán para definir los atributos y el comportamiento que tendrán los objetos dentro del programa, una clase está compuesta por lo siguiente:

- Conjunto de variables.
- Conjunto de métodos.
- Conjunto de funciones.
- Conjunto de constructores.

```
class Nodo {
        //Variables, Métodos, Funciones, Constructores
}
```

Instancia de una clase (Objeto)

Una vez definida una clase se pueden realizar instancias de esta, a esto se le conocerá como objeto. Para instanciar un nuevo objeto se utilizará la palabra reservada **new** y se indicará con qué constructor se quiere construir la instancia.

```
/* entrada1.NM*/
Import "entrada2.NM";
class Main{
        Void main(){
        Persona p1 = new Persona ();
        Persona p2 = new Persona (12);
        }
}
```

```
/* entrada2.NM */
class Persona {
        int edad;
        Persona () {
        }
        Persona (int edad){
                this.edad = edad;
        }
}
```

Acceso a los elementos de un objeto

Se podrá acceder a los atributos, métodos o funciones del objeto, a continuación, un ejemplo de cómo se haría.

```
/* Archivo 1*/
Class Main{
    Void main(){
    Persona p = new Persona (12);
    Println("Edad" + p.getEdad());
    Println(p.hijo.hijo.edad);
    }
}
```

Asignación a los elementos de un objeto

Se podrá modificar el valor de los atributos del objeto, a continuación, un ejemplo de cómo se haría.

```
/* Archivo 1*/
Class Main{
    Void main(){
    Persona p = new Persona (12);
    p.edad = 50;
    p.hijo.hijo.edad = 5;
    }
}
```

this

La palabra reservada **this** dentro de una clase servirá para hacer referencia a ella misma, a través de utilizar **this** se podrá acceder a los atributos, métodos o funciones que tenga la clase.

```
/* entrada2.NM */
class Persona {
    int edad;
    Persona (int edad){
        this.edad = edad;
    }
}
```

Palabras reservadas

int	double	char	boolean	String
Println	Print	Graficar_dot	Graficar_entornos	if
else	switch	case	default	for
while	do	break	continue	Import
void	new	class		

Precedencia y asociatividad de operadores

La precedencia dentro de la tabla está definida de menor a mayor.

Nivel de precedencia	Operador	Asociatividad
1	||	Izquierda
2	&&	Izquierda
3	^	Izquierda
4	>, <, >=, <=, ==, !=	No asociativo
5	+, -	Izquierda
6	*, /, %	Izquierda
7	!	Derecha

Nota: de tener algún problema con la precedencia definida se adjunta un enlace donde pueden consultar otra tabla de precedencia y asociatividad: https://introcs.cs.princeton.edu/java/11precedence/

Restricciones del proyecto

- La aplicación deberá de desarrollarse utilizando el lenguaje Java.
- Las herramientas para realizar los analizadores serán JFlex y Cup.
- El proyecto se realizará de manera individual si se detecta algún tipo de copia el laboratorio quedará anulado.
- La calificación será sobre el archivo ejecutable de su aplicación, de no cumplir con este requerimiento NO SE CALIFICARÁ.

Forma de entrega del proyecto

El proyecto estará dividido en dos fases, a continuación, se define cómo se van a realizar las entregas.

Fase 1 del proyecto

Funcionalidades a entregar

- Interfaz Gráfica
- Comentarios
- Tipos de datos
- Caracteres de escape
- Expresiones
- Declaraciones de variables
- Asignación de variables
- Sentencias de control
- Sentencia Importar
- Funciones nativas

Entregables

- Todas las funcionalidades antes descritas.
- Código fuente de la aplicación.
- Ejecutable de la aplicación, **LA CALIFICACIÓN SERÁ DESDE ESTE EJECUTABLE**, de no funcionar no se calificará.

Fecha de entrega

Domingo 15 de diciembre de 2019, hasta las 23:59

Fase 2 del proyecto

Funcionalidades a entregar

- Métodos y Funciones
- Arreglos
- Clases

Entregables

- Todas las funcionalidades antes descritas.
- Código fuente de la aplicación.
- Ejecutable de la aplicación, **LA CALIFICACIÓN SERÁ DESDE ESTE EJECUTABLE**, de no funcionar no se calificará.

Fecha de entrega

Domingo 5 de enero de 2020, hasta las 23:59