Proyecto de clase

El objetivo del proyecto es implementar un compilador (analizador sintáctico, léxico y semántico) para un lenguaje de programación orientado a objetos, débilmente tipado y con ligadura de tipo estática y explícita, con alcance estático que soporta anidación de bloques pero no de subrutinas y que no permite el manejo de variables globales. La especificación se describe más adelante.

El programa permitirá seleccionar el archivo de código fuente a compilar. Al iniciar la compilación, se muestra el nombre y ruta del archivo que se están compilando y luego la lista de errores indicando un mensaje claro para el usuario y la línea donde está el error.

Estructura general de un programa

El lenguaje es orientado a objetos, por lo tanto deberá existir una clase con un método llamado principal que será la entrada del programa. El método principal siempre tendrá la siguiente sintaxis:

```
publico estatico vacio principal(cadena [] argumentos)
{
 .....
}
```

La clase que contiene el método principal deberá ser declarada como estática.

Justo antes de la definición de la primera clase, se pueden activar o desactivar variables de entorno (palabras en mayúscula):

- MOSTRARERRORES: Lista todos los errores de compilación
- ACTIVARCONSOLA: Activa la entrada y salida por consola
- VERIFICARENTRADA: Muestra el valor anterior de una variable ingresada por teclado

Un archivo puede tener la definición de varias clases. Los nombres de las clases siempre inician con mayúscula (obligatoriamente).

Los comentarios se encierran entre los caracteres ///

Los métodos pueden recibir una lista de parámetros y es posible darles valores por defecto. Para definir un parámetro de un método se indica el tipo, el nombre y opcionalmente el valor por defecto: tipo nombreparametro[=valor]

Si hay más de un parámetro se separan por comas.

Ejemplo de un programa:

```
///  Estructura general de un programa ///

usar nombrepaquete.*;
usar nombrepaquete.Nombreclase;

activar  NOMBREVARIABLEENTORNO;
desactivar NOMBREVARIABLEENTORNO;

visibilidad estatico clase Nombreclase
{

        publico estatico vacio principal(cadena [] argumentos)
        {
                /// Lógica del método principal ///
        }

        visibilidad estatico tiporetorno nombremetodo(listaparametros)
        {
                ///  lógica del método ///
                retornar elemento;   ///Elemento: valor, constante, variable, colección ///
        }

}
```

Visibilidad

La visibilidad puede ser:

- publica
- privada
- protegida

Tipos de datos

Nombre	**Tipo**	**Tamaño (bits)**	**Rango**	**Operaciones**
entero	Número entero	32	-2,147,483,648 a 2,147,483,649	Aritméticas, relacionales, asignación
enterolargo	Número entero	64	-9*10^18 a 9*10^18	Aritméticas, relacionales, asignación
real	Número real	32	-3,4*10^38 a 3,4*10^38	Aritméticas (excepto %), relacionales, asignación
complejo	Número complejo (a + bi)	64	a y b son reales	Aritméticas (excepto %, ** y //), asignación
caracter	Carácter unicode	16	\u0000 a \uFFFF	Aritméticas (excepto %, / y //), relacionales, asignación
cadena	Cadena de caracteres	N.A.	Hasta 1000 caracteres	Aritméticas (excepto %, / y //), relacionales, asignación
logico	Valor de verdad	1	verdadero, falso (1 o 0)	Lógicas, asignación

Declaración de variables

Se inicia con el tipo de dato seguido del nombre de la variable.
Ejemplo: entero edad = 0;
Si define un solo carácter, enciérrelo entre comillas simples.
Si define una cadena de caracteres, enciérrela entre comillas dobles (tipo cadena).

Operadores aritméticos

Operador	**Descripción**
+	Suma
-	Resta
*	Multiplicación
/	División
//	División entera
%	Resto de división o módulo (sólo para enteros)
**	Potenciación

Operadores relacionales

Operador	**Descripción**
==	Es igual
!=	Es diferente
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual

Operadores lógicos

Operador	**Descripción**
&&	Y (and)
||	O (or)
!	No (not)

Operador de asignación

Los operadores de asignación permiten asignar un valor a una variable. El operador de asignación es el operador igual (=).

Operaciones abreviadas:

Operación	Operador	Utilización	Operación equivalente
Suma	+=	A =+ B	A = A + B
Resta	-=	A =- B	A = A - B
Multiplicación	*=	A =* B	A = A * B
División	/=	A =/ B	A = A / B
Resto de división	%=	A =% B	A = A % B

Entrada y salida por teclado

- **Salida:**
  escribir("Mensaje a mostrar");

- **Entrada:**
  nombrevariable = leer();
  nombrevariable = leer("Mensaje a mostrar");

Caracteres especiales en las cadenas

Mediante la contra barra (\) se puede introducir en una cadena un carácter especial. Algunos tienen una funcionalidad específica:

\n → Salto de línea

\t → Tabulación horizontal

\r → Retornar cursor

Pero en otros casos, puede usarse para que en la cadena se tenga en cuenta el carácter especial literalmente:

\\ → Contra barra

\' → Comilla simple

\" → Comilla doble

Ejemplo: escribir("\"Mensaje\n a\n\tmostrar\"");

La anterior instrucción es correcta y su salida sería:
"Mensaje
a
mostrar"

Estructuras condicionales

- **Condicional simple:**

  Ejemplo:

```
si (a > b) entonces
{
    x=1;
}
```

- **Condicional doble:**

  Ejemplo:

```
si (a > b) entonces
{
   x=1;
}
sino
{
  x=2;
}
```

- **Condicional múltiple:**

  Ejemplo:

```
segun(n)
{
        caso 1:  A;
                 terminar;
        caso 2:  B;
                 terminar;
        caso 3:  C;
                 terminar;
        sino: E;
                 terminar;
}
```

Estructuras repetitivas

- **Bucle mientras**

  Ejemplo:

```
entero s = 0;
entero i = 1;

mientras(i < 10) hacer
{
        s = s + i;
        i++;
}
```

- **Bucle hacer-mientras**

  Ejemplo:

```
entero x = 0;
entero y = 1;
hacer
{
        x = x + y;
        y= 2 * y;
        z= y / 2;
}mientras (z<2);
```

- **Bucle para**

  Ejemplo:

```
entero f = 1;
para(entero i = 1; i < 10; i++)
{
f = f * i;
}
```