

MANUAL DE USUARIO

Requisitos:

- .Net Framework 4.7.2
- Sistema Operativo: Windows 7 o Windows 10

Descripción de la Aplicación:

La aplicación consiste en un IDE, que actualmente cuenta únicamente con análisis léxico, por lo cual es capaz de colorear ciertas expresiones que son escritas en un cuadro de texto.

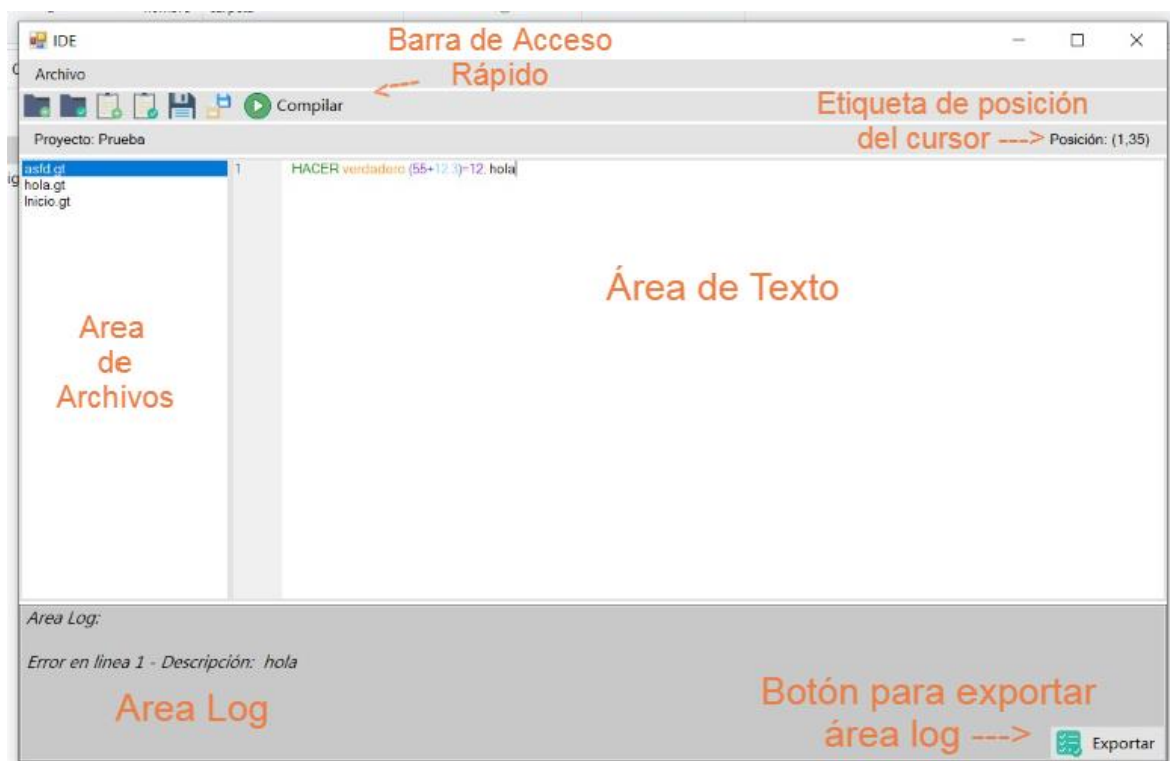
Imagen de Inicio



Botones:

- **Crear Proyecto:** El usuario crea una carpeta con archivos de texto dentro.
- **Abrir Proyecto:** El usuario selecciona un proyecto ya creado.
- **Crear Archivo:** El usuario crea un archivo de texto.
- **Abrir Archivo:** El usuario selecciona un archivo de texto ya creado.

Imagen del IDE:

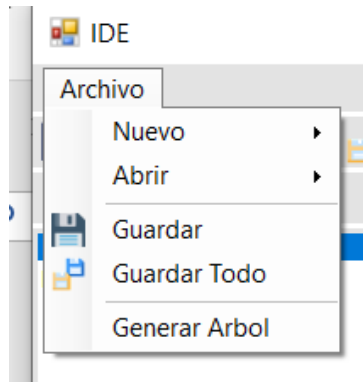


Funcionalidad:

- **Área de Texto:** En ella el usuario puede escribir cualquier cosa (aunque se espera que sea código fuente del lenguaje que interpreta el IDE).
- **Etiqueta de Posición del Cursor:** Indica las coordenadas del Área de Texto en que se encuentra el cursor:
- **Área de Archivos:** En ella se encuentra el nombre de todos los archivos que forman parte del proyecto. Si el usuario hace click en el nombre de alguno de estos archivos, el IDE muestra en el área de texto el código fuente perteneciente al archivo seleccionado.
- **Barra de Acceso Rápido:** En ella se encuentran 7 botones, que, de manera ordenada, hacen lo siguiente (izquierda a derecha):
 1. Crea un nuevo proyecto.
 2. Abre un nuevo proyecto.
 3. Crea un archivo de texto dentro del proyecto.
 4. Abre un archivo de texto externo al proyecto.
 5. Guarda únicamente el archivo que se muestra en el área de texto.
 6. Guarda todos los archivos del proyecto.
 7. Verifica e imprime en el área log los errores léxicos y sintácticos que puede tener el código del área de texto.
- **Área Log:** Área que muestra la fila y columna donde se cometió algún error léxico o sintáctico.

- **Botón Para Exportar Área Log:** Crea un archivo (.gtE) dentro de la carpeta del proyecto. Dicho archivo contiene los errores que el área log ha detectado hasta ese momento.

Botón Archivo: Botón que permite crear nuevos proyectos o archivos, abrir proyectos o archivos, guardar el documento de texto abierto en el área de texto, guardar todos los archivos del proyecto y generar el árbol sintáctico del código fuente (siempre que el mismo no tenga errores sintácticos, ya que también se verifica la estructura).



Lenguaje de Programación:

A continuación, se presentan los tipos de datos que el programa acepta, y también muestra el color del que se debe pintar.

Datos Primitivos:

Tipo	Palabra reservada	Color	Ejemplo
Entero	entero	Morado	15 -90 23
Decimal	decimal	Celeste	-25.2 12.588 5.0
Cadena	cadena	Gris	"es una cadena"
Boolean	booleano	Naranja	verdadero falso
Chart	carácter	Cafe	a A B i

Operadores Aritméticos:

Operador	Color	Ejemplos
+	Azul oscuro	5 + 4, -25.25 + 13, 12 + 4.55, "texto" + 12, unld + 13.25
-	Azul oscuro	5 - 4, -25.25 - 13, 12 - 4.55, unld - 13.25
*	Azul oscuro	5 * 4, -25.25 * 13, 12 * 4.55, unld * 13.25
/	Azul oscuro	5 / 4, -25.25 / 13, 12 / 4.55, unld / 13.25
++	Azul oscuro	5++, 2.5++, unld++
--	Azul oscuro	5--, 2.5--, unld--

Operadores Relacionales:

Operador	Color	Ejemplos
>	Azul oscuro	5 > 4, -25.25 > 13, 12 > 4.55, unld > 13.25
<	Azul oscuro	5 < 4, -25.25 < 13, 12 < 4.55, unld < 13.25
>=	Azul oscuro	5 >= 4, -25.25 >= 13, 12 >= 4.55, unld >= 13.25
<=	Azul oscuro	5 <= 4, -25.25 <= 13, 12 <= 4.55, unld <= 13.25
==	Azul oscuro	5 == 4, -25.25 == 13, 12 == 4.55, unld == 13.25, "texto" == "otro"
!=	Azul oscuro	5 != 4, -25.25 != 13, 12 != 4.55, unld != 13.25, "texto" !=

Operadores Lógicos:

Operador	Color	Ejemplos
	Azul oscuro	5 > 4, -25.25 > 13, 12 > 4.55, unld > 13.25
&&	Azul oscuro	5 < 4, -25.25 < 13, 12 < 4.55, unld < 13.25
!	Azul oscuro	5 >= 4, -25.25 >= 13, 12 >= 4.55, unld >= 13.25

Signos de Agrupación:

Operadores	Color	Ejemplos
()	Azul oscuro	3 - (5 / (25 + 4)) !!(5 > 25) ((!(a > b) && verdadero ((c == 5) && !falso) d <= 4) && 5 >= g

Asignación y Fin de Sentencia:

Símbolo	Color	Ejemplos
=	Rosado	unld = 25 + 25;
;	Rosado	entero mild;

Palabras Reservadas:

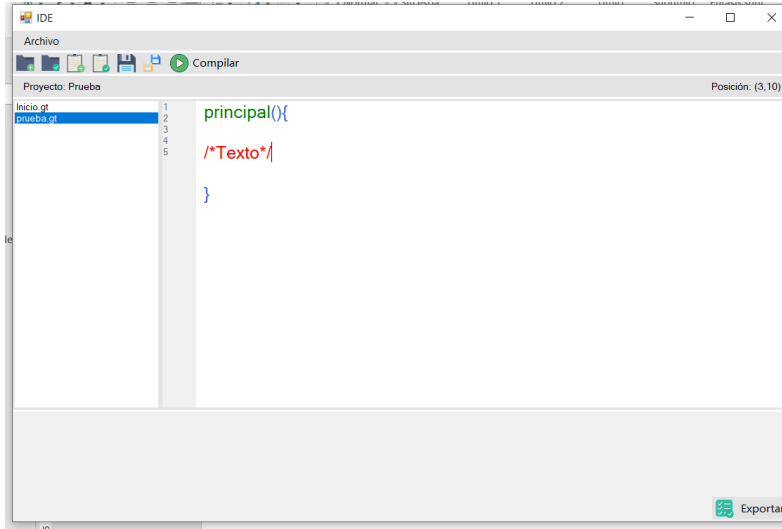
SI, SINO, SINO_SI, MIENTRAS,HACER,DESDE,HASTA, INCREMENTO, entero, decimal, principal...	Verde
--	-------

Comentarios:

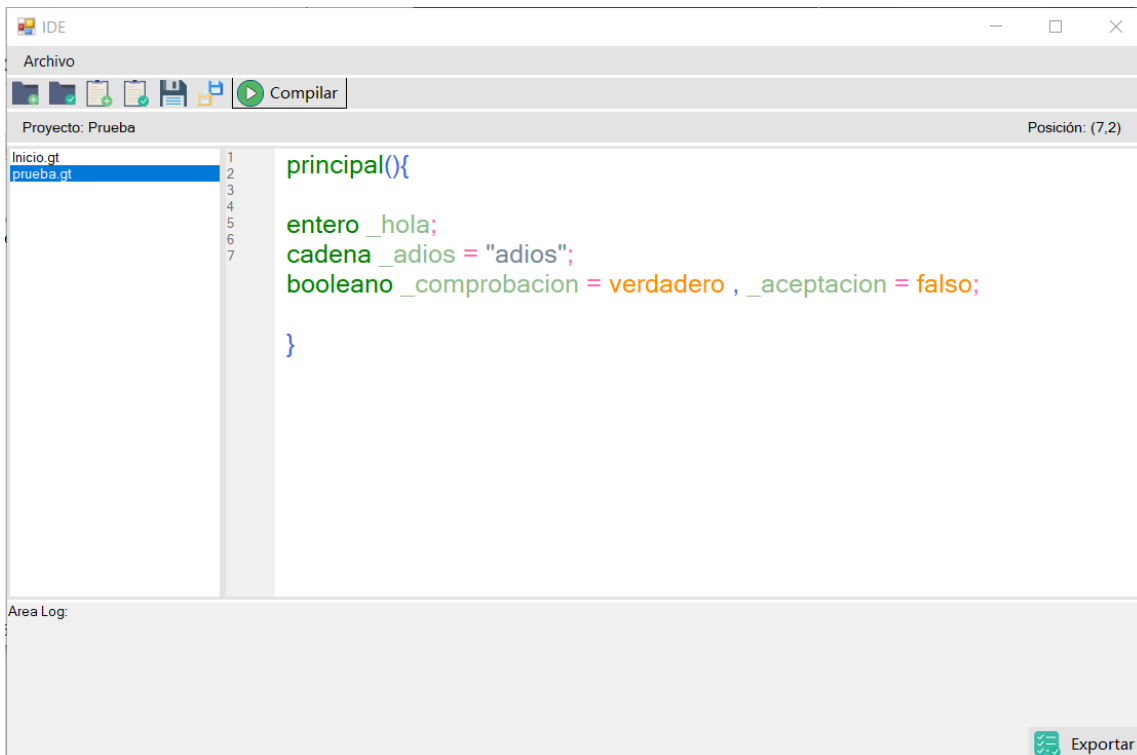
/* Esto también 5 es un comentario */	Rojo
---------------------------------------	------

Sintaxis del Lenguaje

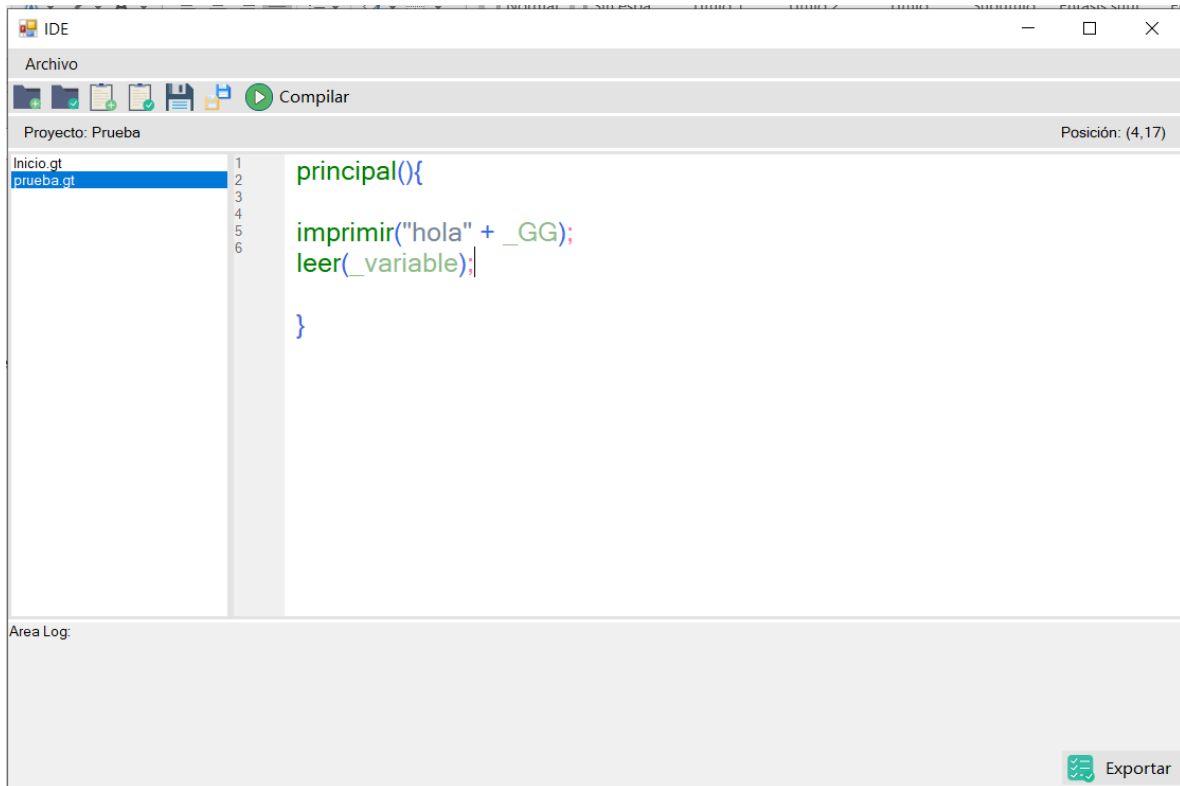
Estructura del Código: El código siempre deberá empezar con la palabra: “principal”, la cual deberá estar acompañada de dos paréntesis “()”, seguido de dos llaves, dentro de las cuales deberá ir el texto.



Declaración de Variables: para declarar una variable, primero se debe poner el identificador de la variable (entero, cadena, etc), después el nombre de la variable (el cual debe iniciar siempre con un guion bajo), si se desea asignar un valor en la declaración se debe seguir con un signo igual seguido del valor a asignar. Se pueden declarar en una sola línea la cantidad de variables que sea, siempre que las mismas estén separadas por comas, se culmina con “;”.



Imprimir y Leer: para imprimir, primero se debe escribir la palabra “imprimir”, seguido de dos paréntesis, dentro de los paréntesis se debe escribir lo que se imprimirá, esto pueden ser variables o cadenas (se puede escribir cualquier cantidad de cadenas o variables, siempre que las mismas estén separadas por “+”), se finaliza con “;”. Para leer, primero se debe escribir la palabra “leer”, seguida de dos paréntesis, dentro de los paréntesis debe ir una variable (la cual será la variable a leer), se finaliza con “;”.

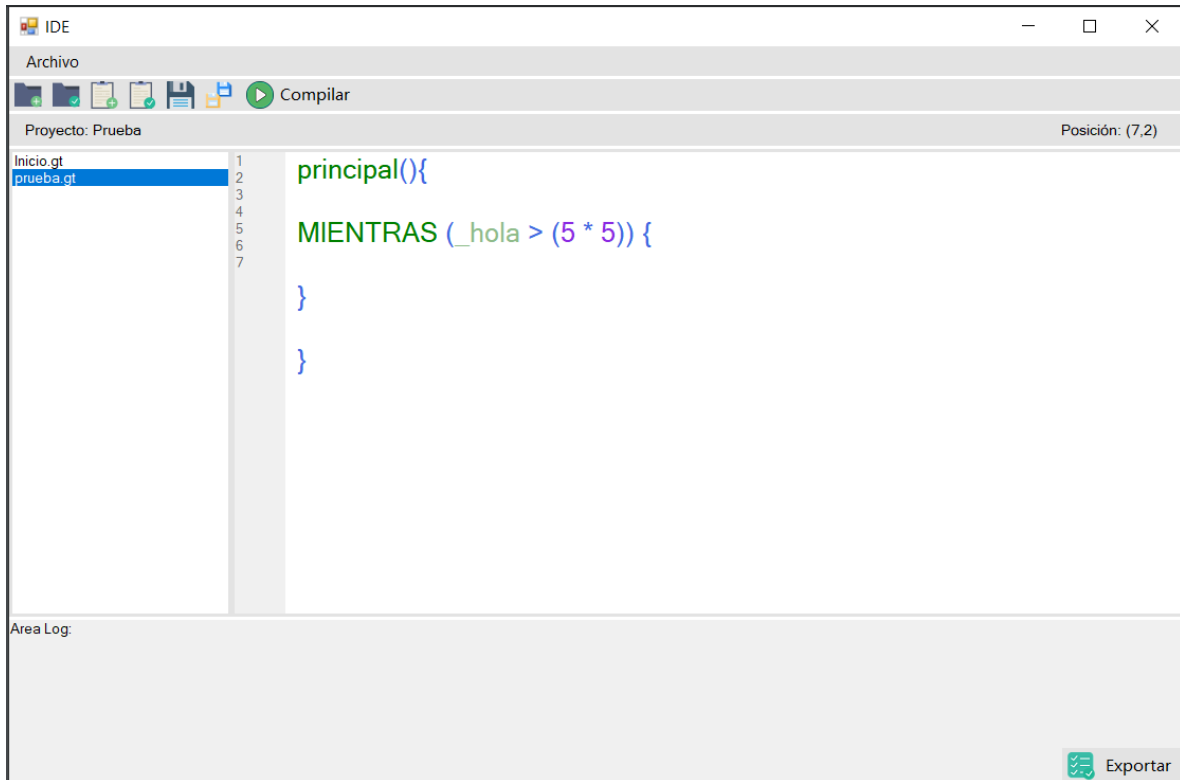


```
1 principal(){
2
3
4
5 imprimir("hola" + _GG);
6 leer(_variable);
7
8 }
```

Estructura SI, SINO_SI, SINO: Siempre se deberá empezar con la palabra “SI”, seguida de dos paréntesis dentro de los cuales debe ir una operación lógica (por lo mismo la operación debe arrojar un resultado booleano), después se continua con dos llaves, dentro de las cuáles puede ir cualquier tipo de código. La estructura SINO_SI sirve para evaluar una nueva condición, siempre que la condición anterior no se haya cumplido (por lo cual se puede poner una seguida de otra las veces que sea); por lo mismo, “SINO_SI” no debe ir escrita después de una estructura “SINO”. La estructura “SINO_SI” tiene la misma forma que la estructura “SI”, la única diferencia se encuentra en la palabra reservada que se escribirá. La estructura “SINO” se coloca cuando las condiciones anteriores no se cumplen y se necesita que el programa ejecute alguna acción aunque ninguna condición se haya cumplido; por lo tanto, la estructura “SINO” solo puede ser escrita, si anteriormente se finalizó una serie de condiciones. La estructura “SINO” empieza con la palabra “SINO” y le siguen dos llaves entre las cuales se escribirá cualquier código.

```
SI (_a > _b) {  
  _T = 500;  
}  
  
SI (_a > _b){  
  _a = 770;  
}  
SINO {  
  _b = 50;  
}  
  
SI (!falso){  
  _a = 770;  
}  
SINO_SI (_c > _b && verdadero){  
  _b = 50;  
}  
SINO_SI (verdadero){  
  //sentencias  
}  
SINO {  
  //sentencias  
}
```


Estructura MIENTRAS: La estructura inicia con la palabra “MIENTRAS”, seguida de dos paréntesis dentro de los cuales deberá ir una operación lógica, la cual debe tener un resultado booleano, después se escriben dos llaves dentro de las cuales debe ir texto. Esta estructura sirve para que se repita la lo que esté dentro de las llaves, siempre que se cumpla la condición establecida dentro de los paréntesis.

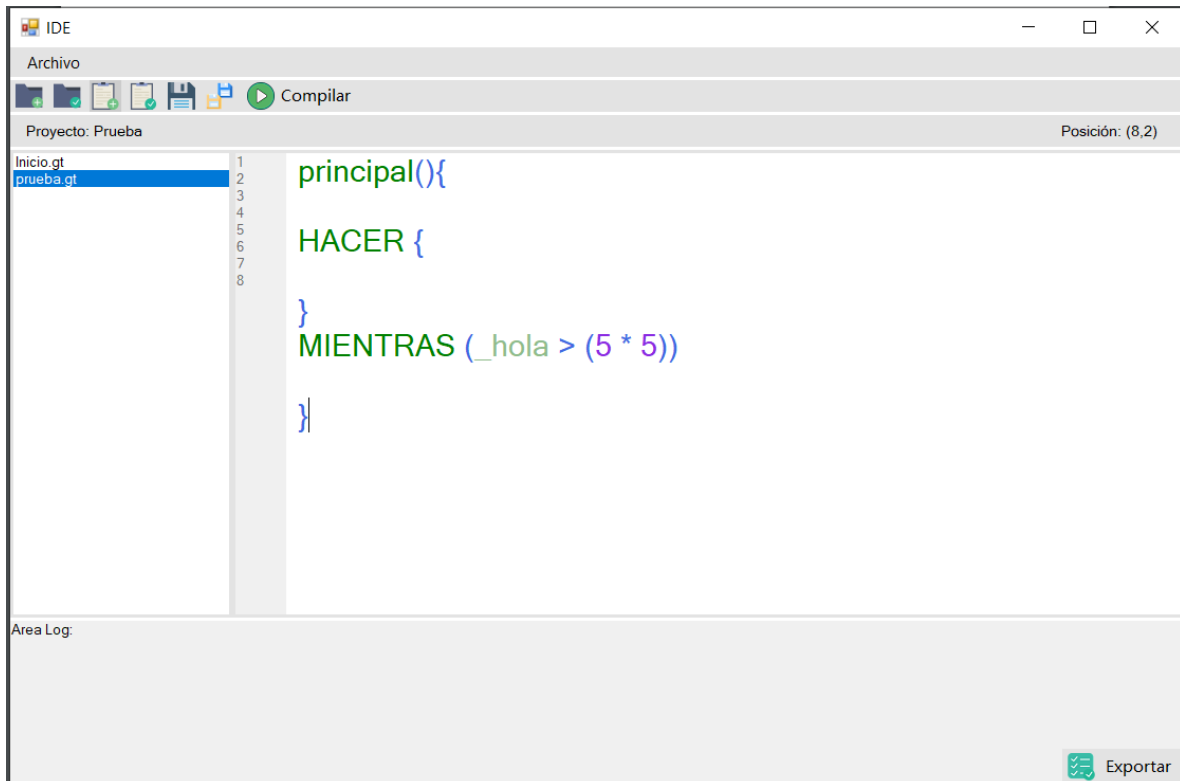


The screenshot shows an IDE window titled "IDE" with standard window controls. Below the title bar is a menu bar with "Archivo" and a toolbar with icons for file operations and a "Compilar" button. The main area is divided into a project explorer on the left showing "Inicio.gt" and "prueba.gt", and a code editor on the right. The code editor displays the following code:

```
1 principal(){
2
3
4
5 MIENTRAS (_hola > (5 * 5)) {
6
7 }
```

Below the code editor is an "Area Log:" section. In the bottom right corner, there is an "Exportar" button with a checklist icon.

Estructura HACER MIENTRAS: Esta estructura inicia con la palabra “HACER” seguida de dos llaves dentro de las cuales debe ir texto. La estructura finaliza con la palabra “MIENTRAS” seguida de dos paréntesis dentro de los cuales debe ir una operación lógica, la cual debe tener un resultado booleano. Esta estructura no finaliza con “;”. Esta estructura sirve para que el contenido que esté dentro de las llaves se repita siempre que la condición establecida sea verdadera, aunque se diferencia de la estructura HACER, ya que HACER MIENTRAS evalúa la condición al encontrar la llave de cierre, por lo que siempre se ejecutará al menos una vez.

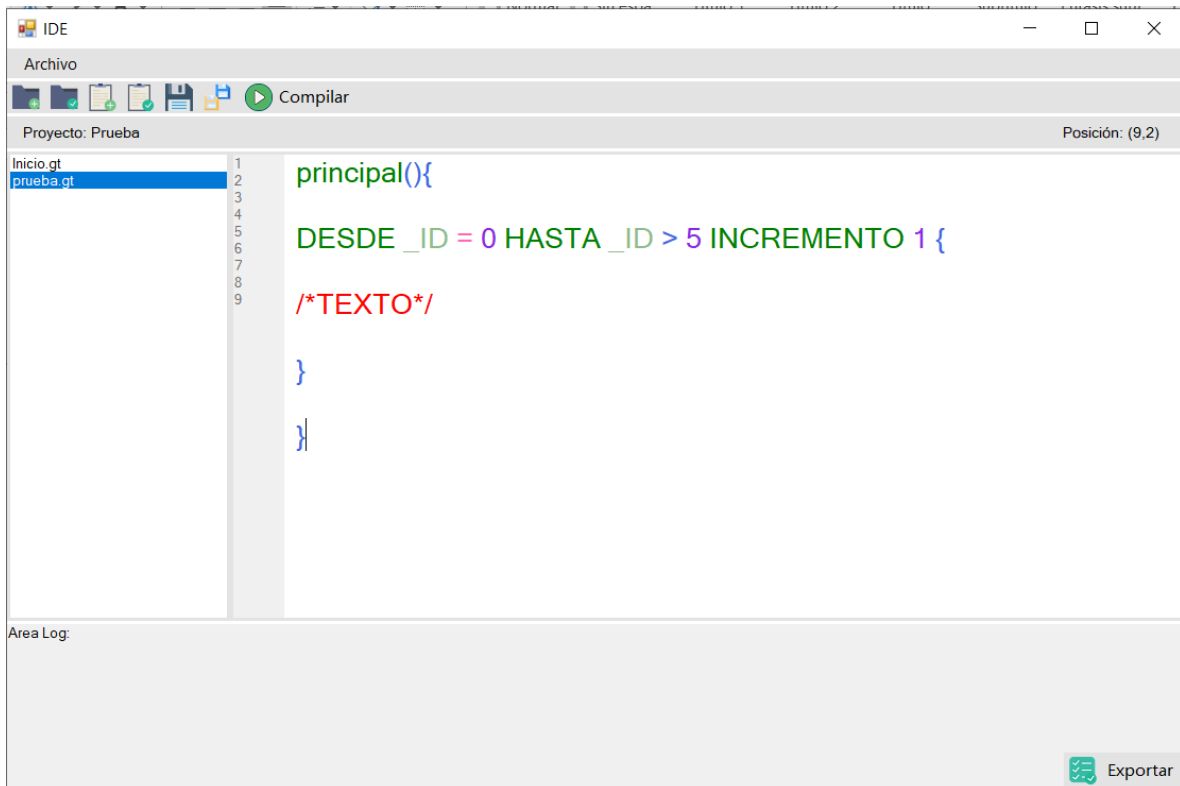


The screenshot shows an IDE window titled "IDE" with standard window controls. Below the title bar is a menu bar with "Archivo" and a toolbar with icons for file operations and a "Compilar" button. The status bar indicates "Proyecto: Prueba" and "Posición: (8,2)". The main editor area displays the following code:

```
1 principal(){
2
3
4
5
6 HACER {
7
8 }
MIENTRAS (_hola > (5 * 5))
}
```

At the bottom of the window is an "Area Log:" section and an "Exportar" button with a checklist icon.

ESTRUCTURA DESDE HASTA INCREMENTO: Esta estructura inicia con la palabra “INCREMENTO”, seguida de una asignación de valor a una variable entera, esto seguido de la palabra “HASTA” la cual está seguida de una operación lógica (la cuál involucra a la variable, un operador lógico y un número entero), todo esto seguido de la palabra “INCREMENTO” la cual debe ir seguida por un número entero, para finalizar con dos llaves dentro de las cuales debe ir texto. Esta estructura asigna un valor a una variable, fija un límite a la variable y le indica el valor que debe incrementarse la variable con cada repetición. La finalidad de esta estructura es que el texto que se encuentra dentro de las llaves se repita la cantidad de veces que la estructura lo permita.

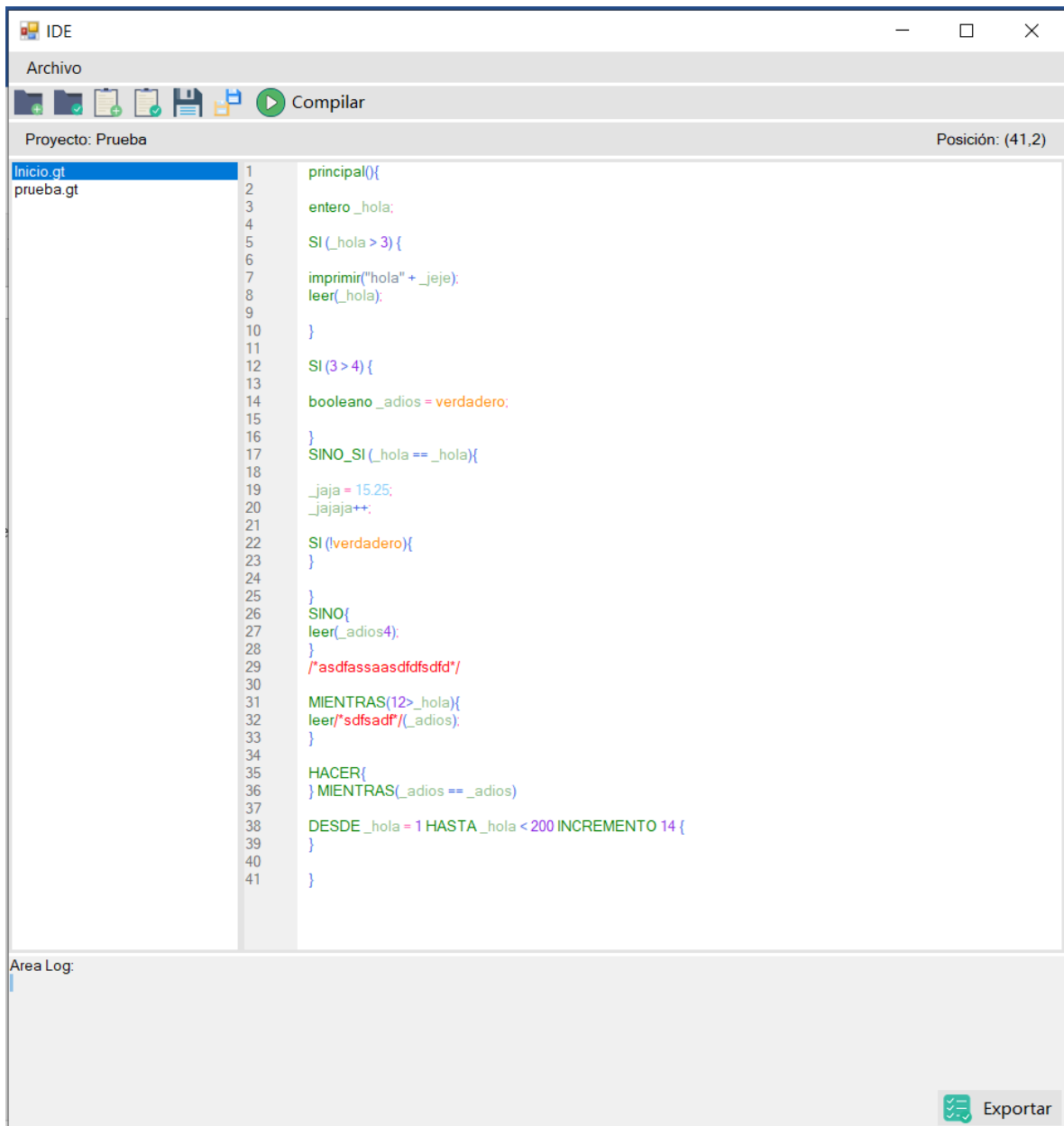



The screenshot shows an IDE window titled "IDE" with a menu bar containing "Archivo" and a toolbar with icons for file operations and a "Compilar" button. Below the toolbar, it says "Proyecto: Prueba" and "Posición: (9,2)". The main editor area displays the following code:

```
1 principal(){
2
3
4
5 DESDE _ID = 0 HASTA _ID > 5 INCREMENTO 1 {
6
7
8 /*TEXTO*/
9
10 }
11 }
```

At the bottom of the window, there is an "Area Log:" section and an "Exportar" button with a document icon.

Imagen del IDE





A screenshot of the 'Archivo' (File) menu in the IDE. The menu is open, showing options: 'Nuevo' (New), 'Abrir' (Open), 'Guardar' (Save), 'Guardar Todo' (Save All), and 'Generar Arbol' (Generate Tree). The 'Nuevo' and 'Abrir' options have right-pointing arrows next to them. The 'Guardar' option is highlighted with a blue background.

```

graph TD
    Codigo((Codigo)) --> Texto1((Texto))
    Codigo --> SimbAlf((Simbólico/Alf))
    Codigo --> FPPrincipal((FPPrincipal))
    Texto1 --> Texto2((Texto))
    Texto1 --> CicloF((CicloF))
    SimbAlf --> 1_1((1))
    SimbAlf --> 2_1((2))
    SimbAlf --> 3_1((3))
    CicloF --> ChargedDescCond((ChargedDesc/Cond))
    CicloF --> ExtractCicloF((ExtractCicloF))
    ChargedDescCond --> ExtractDescuento((ExtractDescuento))
    ChargedDescCond --> ExtractBasta1((ExtractBasta))
    ExtractCicloF --> ExtractBasta1
    ExtractCicloF --> ExtractDesde((ExtractDesde))
    ExtractDesde --> desde((desde))
    ExtractDesde --> PREDEFIN((PREDEFIN))
    desde --> Num1((Num))
    desde --> plus((+))
    desde --> Toldd1((Toldd))
    desde --> hasta((hasta))
    desde --> PREASTA((PREASTA))
    hasta --> Num2((Num))
    hasta --> Opdd((Opdd))
    hasta --> Toldd2((Toldd))
    hasta --> Num3((Num))
    hasta --> PREC/CREM/ENTO((PREC/CREM/ENTO))
    hasta --> Texto3((Texto))
    PREC/CREM/ENTO --> 1_2((1))
    PREC/CREM/ENTO --> 2_2((2))

```