



Armus

Lenguaje de Programación
Orientado a Objetos

Manual de ayuda técnica con la descripción del lenguaje, palabras reservadas, expresiones regulares, lista de operadores, caracteres especiales, construcción de comentarios, ejemplos del lenguaje, y otros detalles técnicos.

Materia: Compiladores

Catedrático: Lic. Enmanuel Amaya

Desarrollado por grupo ACL:

- » Néstor Santiago Aldana Rodríguez
- » Bárbara Stefany Aparicio Bermúdez
- » Katherine Estefani Cabrera Blanco
- » Diana Marcela López Rosales
- » Carlos Miguel López Loarca



Universidad Centroamericana
"José Simeón Cañas"
UCA | Septiembre 2016

A. Descripción del diseño del lenguaje y de sus posibles aplicaciones.

El Lenguaje de programación se basa en la ejecución de algoritmos en pseudo-código en el lenguaje español, lo cual fortalece el práctico entendimiento en las personas que se inician en el ámbito de la programación, y además brinda una de una herramienta más clara para el aprendizaje del paradigma de programación orientada a objetos.

Origen del lenguaje Armus

El nombre del lenguaje que se encuentra en la etapa de diseño e identificación lexicográfica ha sido nombrado como **Armus**, dicho nombre tiene origen en el nombre de una estrella llamada Eta Capricorni¹ (también conocida como Armus) es una estrella binaria en la constelación de Capricornio.

Estructura y sintaxis

A continuación, se define toda la estructura y sintaxis para la ejecución de algoritmos.

- Paradigma del lenguaje: Programación Orientada a Objetos (POO)
- Lenguaje sensitivo a mayúsculas y minúsculas (case sensitive).

Manejo de múltiples archivos de código fuente	<p>El lenguaje soporta la inclusión de múltiples archivos en un o más archivos que conforman el código fuente completo del programa escrito en el lenguaje Armus.</p> <p>El parámetro de configuración MAX_FILE define el número máximo de archivos que pueden existir conteniendo todo el código fuente del programa.</p> <p>El parámetro de configuración MAX_NAME_FILE define el número máximo de caracteres que puede tener el nombre o ruta del archivo a incluir.</p> <p><u>Estructura:</u> incluir <ruta o nombre del archivo a ser incluido>;</p> <p><u>Ejemplos:</u> incluir "/home/mike/Documentos/proyecto/Principal.acl"; incluir "../Documentos/proyecto/Fuentes.acl"; incluir "Calculos.acl";</p> <p>NOTA: La ruta puede ser absoluta o relativa en Linux exclusivamente.</p>
Manejo de variables	<p><u>Estructura:</u> <tipo de dato> <variable alfanumérica>;</p> <p><u>Ejemplos de sintaxis:</u> entero n1; byte num, contador, i; real temp;</p>
Tipos de datos	<ol style="list-style-type: none">1. vacio (equivalente a void)2. booleano (0 ó 1, Verdadero o Falso)

¹ https://es.wikipedia.org/wiki/Eta_Capricorni

	3. cadena (cadena de caracteres) 4. caracter (equivalente a char), 2 Bytes 5. byte (-127 a 128), 1 Byte 6. entero (equivalente a int), 4 Bytes 7. real (equivalente float), 4 Bytes 8. objeto (equivalente a Object) 9. Archivo (equivalente a File)
Instrucciones de Selección	<pre> si[<condición>]{ <sentencias>; }sino{ <sentencias>; } </pre>
	<pre> si[<condición>, <valor verdadero>, <valor falso>]; </pre> <p>NOTA: la opción verdadera y falsa no se pueden omitir y deben ser valores.</p>
	<pre> probar[<condición>]{ caso 1: <sentencia1>; <sentencia2>; romper; caso 2: <sentencia1>; <sentencia2>; romper; defecto: <sentencia1>; <sentencia2>; romper; } </pre>
Instrucciones Iterativas	<pre> mientras[<condición>]{ } /* el equivalente a while */ </pre>
	<pre> para[<condición>, <valor inicial>, <incremento>]{ } /* el equivalente a for */ </pre>
	<pre> hacer{ } mientras[<condición>]; /* el equivalente a do-while */ </pre>
	<pre> paraCada[<tipo> <variable>, <arreglo/lista>]{ } /* el equivalente a for-each */ </pre>
Instrucciones de entrada y salida	<p><u>Sintaxis de entrada:</u></p> <p>Sistema.obtenerEntero[<var>]; //lee un entero</p> <p>Sistema.obtenerReal[<var>]; //lee un núm. real</p> <p>Sistema.obtenerCadena[<var>]; //lee una cadena</p> <p>Sistema.obtenerCaracter[<var>]; //lee un carácter</p>

	<pre> <objeto>.obtenerEntero[<var>]; <objeto>.obtenerReal[<var>]; <objeto>.obtenerCadena[<var>]; <objeto>.obtenerCaracter[<var>]; </pre>
	<div> <div> <p><u>Sintaxis de salida:</u></p> <p>Sistema.mostrar["hola"];</p> <p>Sistema.mostrar["%e + 2", valor];</p> <p><objeto>.mostrar[];</p> </div> <div> <p>%e -> entero %r -> real %c -> carácter %s -> cadena</p> </div> </div>
Sub-Algoritmos	<pre> publica <función>[<parám. entrada>,<tipo salida>]{ /* puede no tener parám. Entrada (vacío) o el tipo de salida puede ser vacío */ retornar <valor de retorno>; } privada <función>[<parám. entrada>,<tipo salida>]{ /* puede no tener parám. Entrada (vacío) o el tipo de salida puede ser vacío */ retornar <valor de retorno>; } NOTA: Los procedimientos pueden o no recibir parámetros de entrada (opcional), pero <u>NO retornan nada</u>. publica <procedimiento>[<parám. entrada>]{ } privada <procedimiento>[<parám. entrada>]{ } NOTA 2: En ambos casos se utilizan sobre objetos. <objeto>.<función>[<parám. entrada>,<tipo salida>]; <objeto>.<procedimiento>[<parám. entrada>]; <objeto>.<procedimiento>[]; <u>Ejemplo:</u> Alumno.ObtenerNombre[]; </pre>
Paso de parámetros por valor y por referencia	<p><u>Por valor</u></p> <pre> <objeto>.<función>[11, 24]; .. <función>[entero n1, entero n2]{ } </pre> <p><u>Por referencia</u></p> <div>Si el parámetro es enviado por referencia, se antepone el símbolo ~</div>

	<pre> <objeto>.<función>[~<parámetro>]; .. entero n1 = 11; <objeto>.<función>[~n1] .. <función>[entero *n]{ *n = 24; } .. Sistema.mostrar[n1]; //muestra: 24 </pre> <p>Si el parámetro es recibido por referencia en la definición de la función se coloca * antes del nombre del parámetro.</p> <p>NOTA: Se ha pasado por la referencia el parámetro enviado a la función, es decir, que todo lo que se cambie al parámetro dentro de la función también se cambiará en el parámetro original (no se hace una copia del parámetro).</p>
Manejo de Arreglos	<pre> Arreglo<tipo> <objeto arreglo>; <objeto arreglo>.agregar[<elemento1>]; <objeto arreglo>.agregar[<elemento2>]; <objeto arreglo>.obtener[<elemento1>]; <objeto arreglo>.obtener[<elemento2>]; <objeto arreglo>.cuantos[]; <objeto arreglo>.quitar[<elemento>]; </pre>
Manejo Básico de Archivos (apertura y volcado)	<pre> Archivo <objeto>; <objeto>.abrir[<ruta o nombre del archivo>]; <objeto>.leerLinea[]; /*lee solo una línea del archivo*/ <objeto>.volcado[]; /*obtiene todo el contenido del archivo*/ <objeto>.cerrar[]; /*cierra el archivo*/ </pre>
Funciones Predefinidas	<pre> concatenar[<cadena1>, <cadena2>]; parteEntera[<número real>]; /*extrae la parte entera de un número real*/ comparar[<cadena1>, <cadena2>]; /*devuelve 1 si son iguales, 0 diferentes*/ mayor[<num1>, <num2>]; menor[<num1>, <num2>]; esPar[<numero>]; /*devuelve 1 si es par, 0 si no lo es*/ decimalBin[<número entero>]; /*transforma de decimal a binario*/ potencia[<número>, <exponente>]; absoluto[<numero>]; /*obtiene el valor absoluto de un numero*/ modulo[<entero1>,<entero2>]; /*obtiene el residuo de una división*/ longitudCadena[<cadena>]; /*devuelve la longitud de una cadena*/ </pre>

	NOTA: todas están definidas en el objeto Sistema																															
Operadores Aritméticos y Lógicos	<u>Lógicos</u>	<u>Orden de Prioridad</u>																														
	<table><tr><th>Operador</th><th>Descripción</th></tr><tr><td>==</td><td>Igual</td></tr><tr><td><</td><td>menor</td></tr><tr><td><=</td><td>Menor o igual</td></tr><tr><td>></td><td>Mayor</td></tr><tr><td>>=</td><td>Mayor o igual</td></tr><tr><td><></td><td>!= Distinto</td></tr><tr><td>&&</td><td>Operador Y</td></tr><tr><td> </td><td>Operador O</td></tr><tr><td>!</td><td>Negación</td></tr></table>	Operador	Descripción	==	Igual	<	menor	<=	Menor o igual	>	Mayor	>=	Mayor o igual	<>	!= Distinto	&&	Operador Y		Operador O	!	Negación	<table><tr><th>Operador</th><th>Descripción</th></tr><tr><td>()</td><td>Paréntesis</td></tr><tr><td>* , /</td><td>Multiplicación, División</td></tr><tr><td>+ , -</td><td>Suma, Resta</td></tr><tr><td>==, <>, >, <, >=, <=</td><td>Igual, Distinto, Mayor, Menor, Mayor o igual, Menor o igual</td></tr></table>	Operador	Descripción	()	Paréntesis	* , /	Multiplicación, División	+ , -	Suma, Resta	==, <>, >, <, >=, <=	Igual, Distinto, Mayor, Menor, Mayor o igual, Menor o igual
	Operador	Descripción																														
==	Igual																															
<	menor																															
<=	Menor o igual																															
>	Mayor																															
>=	Mayor o igual																															
<>	!= Distinto																															
&&	Operador Y																															
	Operador O																															
!	Negación																															
Operador	Descripción																															
()	Paréntesis																															
* , /	Multiplicación, División																															
+ , -	Suma, Resta																															
==, <>, >, <, >=, <=	Igual, Distinto, Mayor, Menor, Mayor o igual, Menor o igual																															
	<u>Aritméticos</u>																															
	<table><tr><th>Operador</th><th>Descripción</th></tr><tr><td>+</td><td>Suma</td></tr><tr><td>-</td><td>Resta</td></tr><tr><td>*</td><td>Multiplicación</td></tr><tr><td>/</td><td>División</td></tr></table>	Operador	Descripción	+	Suma	-	Resta	*	Multiplicación	/	División																					
Operador	Descripción																															
+	Suma																															
-	Resta																															
*	Multiplicación																															
/	División																															
Comentarios	// comentario de una sola línea /* Comentario para párrafos. */																															

B. Palabras reservadas

entero	caso	retornar	menor
byte	defecto	Arreglo	esPar
real	romper	agregar	decimalBin
vacio	mientras	obtener	potencia
booleano	para	cuantos	absoluto
cadena	hacer	quitar	modulo
caracter	paraCada	abrir	longitudCadena
Objeto	Sistema	leerLinea	clase
Archivo	obtenerEntero	volcado	incluir
si	obtenerReal	cerrar	verdadero
sino	obtenerCadena	concatenar	false
probar	obtenerCaracter	parteEntera	obtenerBooleano
publica	mostrar	comparar	mayor
privada	mi		

C. Expresión regular para los identificadores

- VARIABLES

identificador = (letra + símbolo)(letra + dígito + símbolo)*

Donde:

letra = {a, b, ..., ñ, ..., z, A, B, ..., Ñ, ..., Z, á, é, í, ó, ú, Á, É, Í, Ó, Ú}

dígito = {0,1,2,3,4,5,6,7,8,9}

símbolo = {"_"}

- CLASES (*Sub-conjunto de identificador de variable*)

identificador = letraMa(letra + dígito + símbolo)*

Donde:

letraMa = {A, B, C, ..., Z}

letra = {a, b, ..., z, A, B, ..., Z}

dígito = {0,1,2,3,4,5,6,7,8,9}

símbolo = {"_"}

D. Expresiones regulares para números enteros y números reales

- NÚMEROS ENTEROS & HEXADECIMALES

entero = dígito(dígito)* + #(letra + dígito)*

Donde:

dígito = {0,1,2,3,4,5,6,7,8,9}

letra = {A, B, C, D, E, F} (números Hexadecimal 10, 11, 12, 13, 14, 15)

dígito = {0,1,2,3,4,5,6,7,8,9}

Un número entero puede ser definido como un número Hexadecimal, Ejemplo: #A32FF (anteponiendo el símbolo #)

- NÚMEROS REALES

real = entero.entero

E. Expresiones regulares para cadenas y caracteres

- CADENA

cadena = "(letra + dígito + símbolo)*"

Donde:

letra = {a, b, ..., ñ, ..., z, A, B, ..., Ñ, ..., Z, á, é, í, ó, ú, Á, É, Í, Ó, Ú}

dígito = {0,1,2,3,4,5,6,7,8,9}

símbolo = {x / x es cualquier carácter diferente de letra, dígito o "("}

- CARÁCTER

caracter = '(letra + dígito + símbolo)'

Donde:

letra = {a, b, ..., ñ, ..., z, A, B, ..., Ñ, ..., Z, á, é, í, ó, ú, Á, É, Í, Ó, Ú}

dígito = {0,1,2,3,4,5,6,7,8,9}

símbolo = {x / x es cualquier carácter diferente de letra, dígito o "("}

F. Lista de operadores y caracteres especiales

Lógicos

Operador	Descripción
==	Igual
<	Menor
<=	Menor o igual
>	Mayor
>=	Mayor o igual
<> !=	Distinto
&&	Operador Y
	Operador O
!	Negación

Aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División

Caracteres Especiales

Carácter	Descripción
~	Referencia
.	Punto
#	Inicio de Número Hexadecimal
;	Punto y coma
,	Coma
=	Asignación
{	Llave de apertura
}	Llave de cierre
[Corchete de apertura
]	Corchete de cierre
(Paréntesis de apertura
)	Paréntesis de cierre

NOTA: En la composición de cadenas se agregan además todos los caracteres de la tabla ASCII

G. Forma de construcción de comentarios en el lenguaje

Para la utilización de los comentarios se hará de igual forma que en lenguaje C y Java, para los comentarios de una sola línea se utiliza // , para los comentarios de párrafos /* */

Ejemplos:

// comentario de una sola línea

/*

Comentario para párrafos.

*/

H. Un ejemplo de programa para escribir “Hola mundo” en el lenguaje

Manejo de ámbito de variables y métodos: **pública y privada**

```
class <nombre_clase>{
    privada entero variable;
    privada metodo[] {
        entero variable;
        mi.variable = variable;
    }
}
```

Ejemplo (hola mundo):

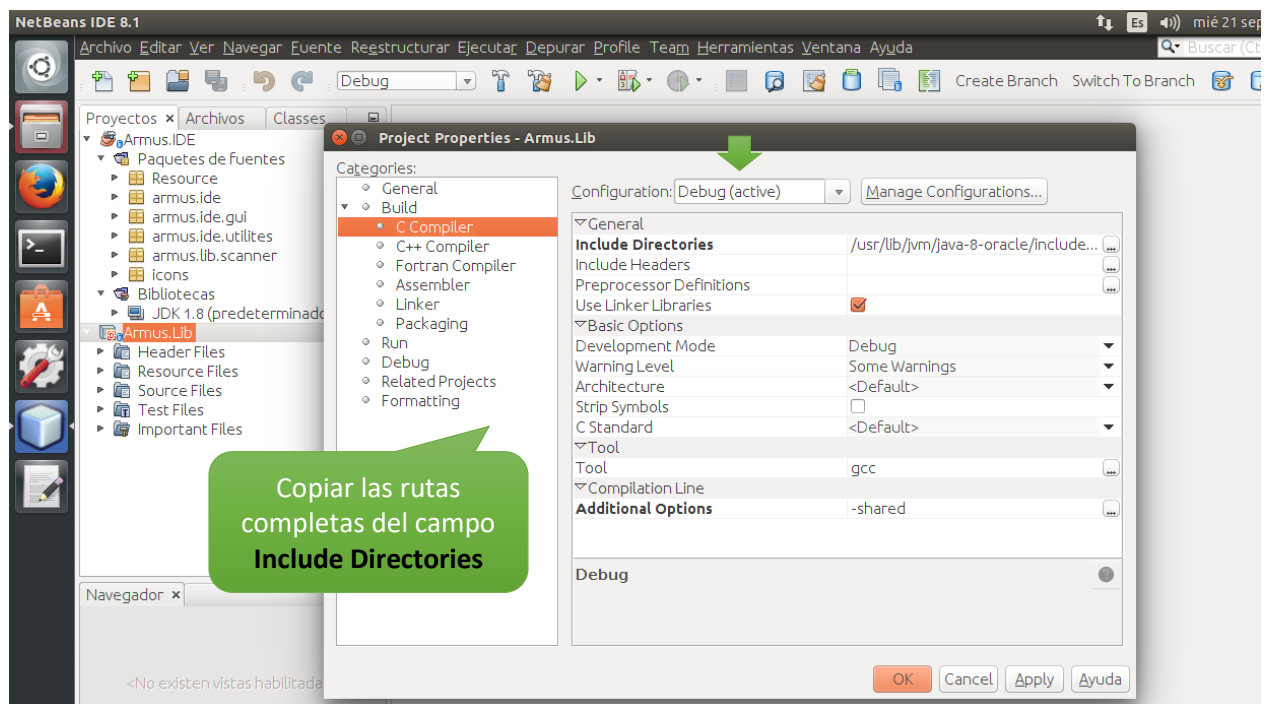
```
class Hola{
    publica principal[vacio,vacio]{
        Sistema.mostrar[“Hola mundo\n”];
    }
    /*comentario*/
}
```


Ejemplo:

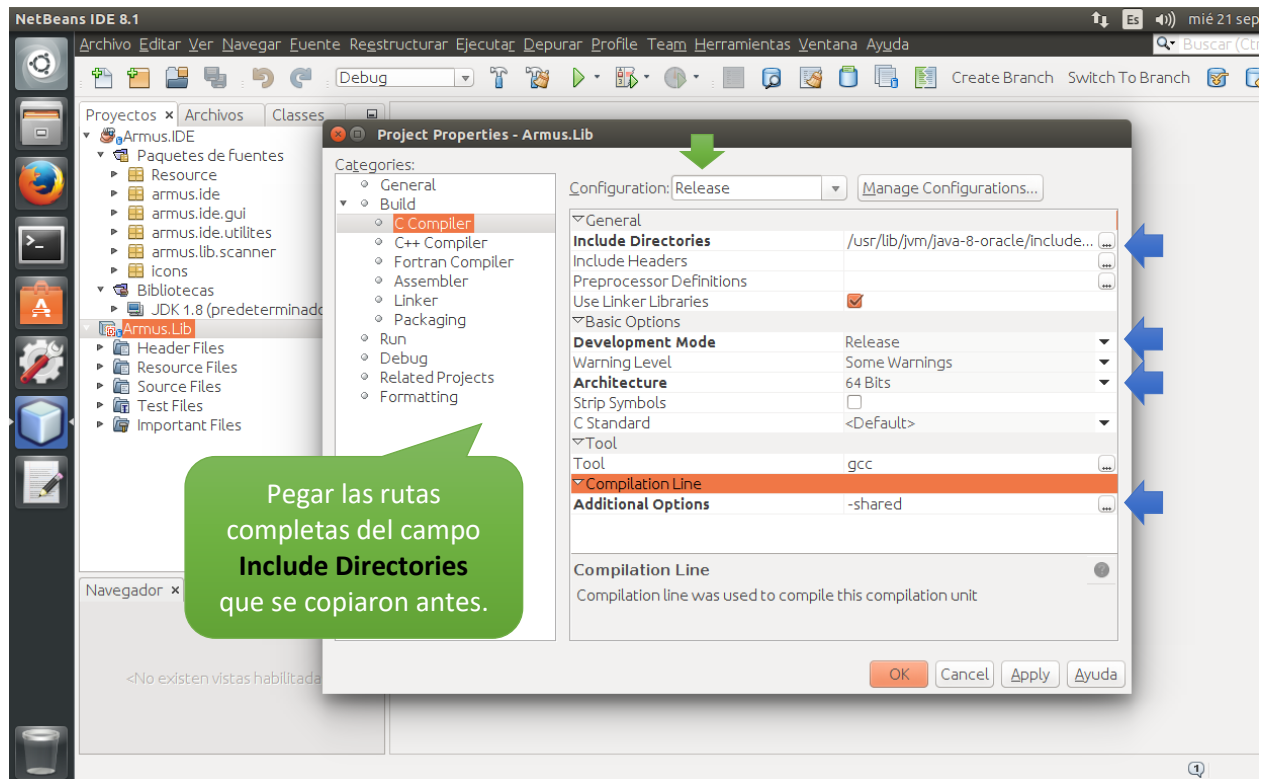
```
class Prueba {
    publica principal[vacio,vacio]{
        cadena cad;
        entero n;
        Sistema.mostrar["Ingrese su nombre\n"];
        Sistema.obtenerCadena[cad];
        si[Sistema.comparar[cad,"carlos"]]{
            Sistema.mostrar["son iguales\n"];
        }sino{
            probar[cad]{
                caso 1: Sistema.mostrar["son diferentes\n"];
                romper;
                caso 2: cad = "nestor";
                romper;
            }
            n = Sistema.calculo3x5[vacio,entero];
        }
        publica calculo3x5[vacio,entero]{
            entero m = 0;
            m = 3 * 5;
            retornar m;
        }
        /*comentario*/
    }
}
```

I. Configuración para la generación de la librería de JNI para el IDE y Analizador Lexicográfico en Netbeans sobre Linux Ubuntu 14.04

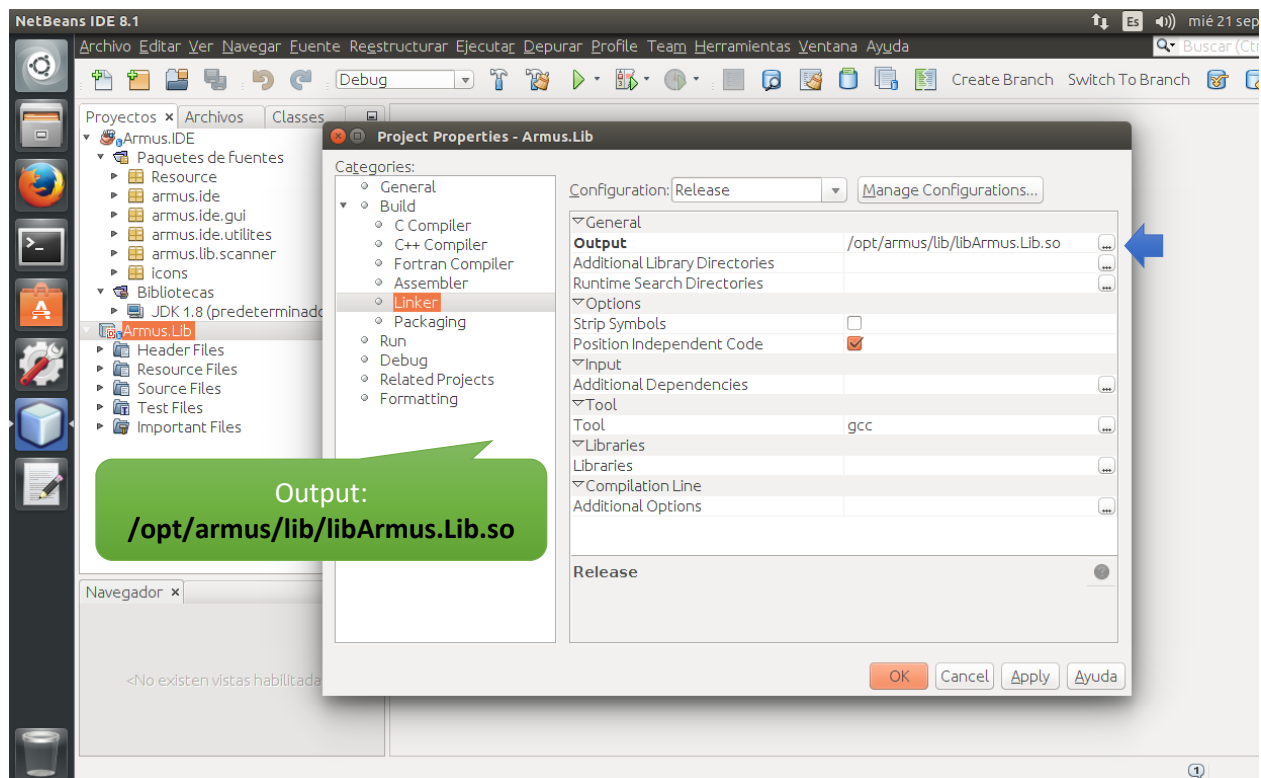
Abrimos **Netbeans** y hacemos clic derecho sobre el proyecto **Armus.Lib**, seleccionamos **Propiedades** y en la opción de **C Compiler** cambiamos en la parte de arriba Debug (active) por **Release**.



Una vez realizado ese cambio colocamos la siguiente configuración en los parámetros mostrados.



Ahora hacemos clic en la opción de **Linker** y añadimos la ruta en el campo **Output**.



Dicho directorio debe ser creado previamente con los permisos del usuario que ejecutará y generará la librería en Netbeans, para ello, abrimos una terminal (o consola) en Linux y escribimos los siguientes comandos:

sudo su	Accedemos como usuario root
cd opt/	Entramos al directorio opt
mkdir armus/	Creamos la carpeta armus
mkdir lib/	Creamos la carpeta lib

El archivo **libArmus.Lib.so** siempre debe estar en esta ruta para que el programa IDE y Analizador Léxico funcionen correctamente.

```
root@vm-ubuntu: /opt/armus/lib
root@vm-ubuntu: /opt/armus/lib#
```

Ahora creamos los permisos para el usuario **mike** en dicha carpeta:

chown mike -R opt/	Otorga permisos al usuario Mike recursivamente en el directorio opt
--------------------	--

```
root@vm-ubuntu: /
root@vm-ubuntu: /opt/armus/lib# cd /
root@vm-ubuntu: /# chown mike -R opt/
```

Ahora sí, vamos nuevamente a Netbeans y en la parte superior cambiamos de Debug a Release, ejecutamos el proyecto y automáticamente la librería se creará con la configuración y ruta establecidas.

