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 Capricorni[1] (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	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.
	
	El parámetro de configuración **MAX_FILE** define el número máximo de archivos que pueden existir conteniendo todo el código fuente del programa.
	
	El parámetro de configuración **MAX_NAME_FILE** define el número máximo de caracteres que puede tener el nombre o ruta del archivo a incluir.
	
	Estructura:
	incluir <ruta o nombre del archivo a ser incluido>**;**
	
	Ejemplos:
	`incluir "/home/mike/Documentos/proyecto/Principal.acl";`
	`incluir "../Documentos/proyecto/Fuentes.acl";`
	`incluir "Calculos.acl";`
	
	`NOTA: La ruta puede ser absoluta o relativa en Linux exclusivamente.`

Manejo de variables	Estructura:
	<tipo de dato> <variable alfanumérica>;
	
	Ejemplos de sintaxis:
	`entero n1;`
	`byte num, contador, i;`
	`real temp;`

Tipos de datos	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	**si**[<condición>]{
	  <sentencias>;
	}**sino**{
	  <sentencias>;
	}

	si[<condición>, <valor verdadero>, <valor falso>]; **NOTA:** la opción verdadera y falsa no se pueden omitir y deben ser valores.

	probar[<condición>]{
	  caso 1: <sentencia1>;
	  <sentencia2>;
	  romper;
	  caso 2: <sentencia1>;
	  <sentencia2>;
	  romper;
	  defecto: <sentencia1>;
	  <sentencia2>;
	  romper;
	}

Instrucciones Iterativas	**mientras**[<condición>]{ } /* el equivalente a while */
	para[<condición>, <valor inicial>, <incremento>]{ } /* el equivalente a for */
	hacer{ }**mientras**[<condición>]; /* el equivalente a do-while */
	paraCada[<tipo> <variable>, <arreglo/lista>]{ } /* el equivalente a for-each */

Instrucciones de entrada y salida	Sintaxis de entrada:
	Sistema.obtenerEntero[<var>]; //lee un entero
	
	Sistema.obtenerReal[<var>]; //lee un núm. real
	
	Sistema.obtenerCadena[<var>]; //lee una cadena
	
	Sistema.obtenerCaracter[<var>]; //lee un carácter

<table>
<tr><td></td><td>

```
<objeto>.obtenerEntero[<var>];
<objeto>.obtenerReal[<var>];
<objeto>.obtenerCadena[<var>];
<objeto>.obtenerCaracter[<var>];
```

<table>
<tr><td>Sintaxis de salida:
Sistema.mostrar[“hola”];

Sistema.mostrar[“%e + 2”, valor];

<objeto>.mostrar[];</td><td>%e -> entero
%r -> real
%c -> carácter
%s -> cadena</td></tr>
</table>

</td></tr>
<tr><td>Sub-Algoritmos</td><td>

```
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 NO retornan nada.

```
publica <procedimiento>[<parám. entrada>]{

}
```

Opcional

```
privada <procedimiento>[<parám. entrada>]{

}
```

Opcional

NOTA 2: En ambos casos se utilizan sobre objetos.

```
<objeto>.<función>[<parám. entrada>,<tipo salida>];

<objeto>.<procedimiento>[<parám. entrada>];

<objeto>.<procedimiento>[];
```

Ejemplo:

```
Alumno.ObtenerNombre[];
```

</td></tr>
<tr><td>Paso de parámetros por valor y por referencia</td><td>

Por valor

```
<objeto>.<función>[11, 24];
..
<función>[entero n1, entero n2]{

}
```

Por referencia

Si el parámetro es enviado por referencia, se antepone el símbolo ~

</td></tr>
</table>

	<objeto>.<función>[~<parámetro>];
	..
	entero n1 = 11;
	<objeto>.<función>[~n1]
	..
	<función>[entero *n]{
	*n = 24;
	}
	..
	Sistema.mostrar[n1]; //muestra: 24
	
	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).
	
	Si el parámetro es recibido por referencia en la definición de la función se coloca * antes del nombre del parámetro.

Manejo de Arreglos	**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>**];**
Manejo Básico de Archivos (apertura y volcado)	**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*/
Funciones Predefinidas	**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*/

<table>
<tr><td></td><td colspan="2">NOTA: todas están definidas en el objeto Sistema</td></tr>
<tr><td>Operadores Aritméticos y Lógicos</td><td>
Lógicos
<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>
Aritméticos
<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>
</td><td>
Orden de Prioridad
<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>
</td></tr>
<tr><td>Comentarios</td><td colspan="2">

```
// comentario de una sola línea

/*
 Comentario para párrafos.
*/
```

</td></tr>
</table>

B. Palabras reservadas

entero	caso	retornar	menor
byte	defecto	Arreglo	esPar
real	romper	agregar	decimalBin
vacio	mientras	obtener	pontencia
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			

C. Expresión regular para los identificadores

- VARIABLES

```
identificador = (letra + símbolo)(letra + digito + símbolo)*
```

Donde:

```
letra = {a, b, .., ñ, .., z, A, B, .., Ñ, .., Z, á, é, í, ó, ú, Á, É, Í, Ó, Ú}
digito = {0,1,2,3,4,5,6,7,8,9}
símbolo = {“_”}
```

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

```
identificador = letraMa(letra + digito + símbolo)*
```

Donde:

```
letraMa = {A, B, C, .., Z}
letra = {a, b, .., z, A, B, .., Z}
digito = {0,1,2,3,4,5,6,7,8,9}
símbolo = {“_”}
```

D. Expresiones regulares para números enteros y números reales

- NÚMEROS ENTEROS & HEXADECIMALES

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

```
entero = digito(digito)* + #(letra + digito)*
```

Donde:

```
digito = {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)
digito = {0,1,2,3,4,5,6,7,8,9}
```

- NÚMEROS REALES

```
real = entero.entero
```

E. Expresiones regulares para cadenas y caracteres

- CADENA

```
cadena = “(letra + digito + símbolo)*”
```

Donde:

```
letra = {a, b, .., ñ, .., z, A, B, .., Ñ, .., Z, á, é, í, ó, ú, Á, É, Í, Ó, Ú}
digito = {0,1,2,3,4,5,6,7,8,9}
símbolo = {x / x es cualquier carácter diferente de letra, digito o “}
```

- CARÁCTER

```
caracter = ‘(letra + digito + símbolo)’
```

Donde:

```
letra = {a, b, .., ñ, .., z, A, B, .., Ñ, .., Z, á, é, í, ó, ú, Á, É, Í, Ó, Ú}
digito = {0,1,2,3,4,5,6,7,8,9}
símbolo = {x / x es cualquier carácter diferente de letra, digito 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**

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

Ejemplo (hola mundo):

```
clase Hola{

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

Ejemplo:

```
clase 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*/
}
```