Apéndice **W1**

Especificaciones de lenguage algorítmico UPSAM. Versión 2.0

1. Elementos del lenguaje

1.1. Identificadores

Se pueden formar con cualquier carácter alfabético regional (no necesariamente ASCII estándar), dígitos (0-9) y el símbolo de subrayado (_), debiendo empezar siempre por un carácter alfabético. Los nombres de los identificadores son sensibles a mayúsculas y se recomienta que su longitud no sobrepase los 50 caracteres.

1.2. Comentarios

Existen dos tipos de comentarios. Para comentarios de una sóla línea, se utilizará la doble barra inclinada (`//`). Este símbolo servirá para ignorar todo lo que aparezca hasta el final de la línea. Los comentarios también podrán ocupar más de una línea utilizando los caracteres `{` y `}`, que indicarán respectivamente el inicio y el final del comentario. Todos los caracteres incluidos entre estos dos símbolos serán ignorados.

1.3. Tipos de datos estándar

Datos numéricos

- **Enteros**. Se considera entero cualquier valor numérico sin parte decimal, independientemente de su rango. Para la declaración de un tipo de dato entero se utiliza la palabra reservada **`entero`**.
- **Reales**. Se considera real cualquier valor numérico con parte decimal, independiente de su rango o precisión. Para la declaración de un tipo de dato entero se utiliza la palabra reservada **`real`**.

Datos lógicos

Se utiliza la palabra reservada **`lógico`** en su declaración.

Datos de tipo carácter

Se utiliza la palabra reservada **`carácter`** en su declaración.

Datos de tipo cadena

Se utiliza la palabra reservada **`cadena`** en su declaración. A no ser que se indique lo contrario se consideran cadenas de longitud variable. Las cadenas de caracteres se consideran como un tipo de dato estándar pero estructurado (se podrá considerar como un array de caracteres).

1.4. Constantes de tipos de datos estándar

Numéricas enteras

Están compuestas por los dígitos (0..9) y los signos + y − utilizados como prefijos.

Numéricas reales

Los números reales en coma fija, utilizan el punto como separador decimal, además de los dígitos (0..9), y el carácter de signo (+ y −).

Para los reales en coma fija, la mantisa podrá utilizar los dígitos (0..9), el carácter de signo (+ y −) y el punto decimal (**.**). El exponente se separará de la mantisa mediante la letra **E** y la mantisa estará formada por el carácter de signo y los dígitos.

Lógicas

Sólo podrán contener los valores **verdad** y **falso**.

De carácter

Cualquier carácter válido del juego de caracteres utilizados delimitados por los separadores ' o ".

De cadena

Secuencia de caracteres válidos del juego de caracteres utilizados, delimitados por los separadores ' o ".

1.5. Operadores

Operadores aritméticos

Operador	Significado
−	Menos unario
−	Resta
+	Más unario
*	Multiplicación
/	División real
div	División entera
mod	Resto de la división entera
^	Exponenciación

El tipo de dato de una expresión arimética depende del tipo de dato de los operandos y del operador. Con los operadores +, -, * y ^, el resultado es entero si los operandos son enteros. Si alguno de los operandos es real el resultado será de tipo real. La división real (/) devuelve siempre un resultado real. Los operadores **mod** y **div** devuelven siempre un resultado de tipo entero.

Operadores de relación

Operador	Significado
=	Igual a
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que

Operador	Significado
<>	Distinto de

Los operandos deben ser del mismo tipo y el resultado es de tipo lógico.

Operadores lógicos

Operador	Significado
no	Negación lógica
Y	Multiplicación lógica (verdadero si los dos operandos son verdaderos)
o	Suma lógica (verdadero si alguno de los operandos es verdadero)

Los operandos deben ser de tipo lógico y devuelven un operando de tipo lógico.

Operadores de cadena

Operador	Significado
+	Concatenación de cadenas
&	Concatenación de cadenas

Trabajan con operandos de tipo cadena o carácter y el resultado siempre será de tipo cadena

Prioridad de operadores

Primarios	( ) [] *Paréntesis en expresiones o en llamadas a procedimientos o funciones. Corchetes en índices de arrays.*
Unarios	-, +, **no**
Multiplicativos	*, /, **div**, **mod**, **y**
Aditivos	+, -, **o**
De cadena	&, +
De relación	=, <, >, <=, >=, <>

2. Estructura de un programa

```
algoritmo <nombre_del_algoritmo>
//Secciones de declaraciones
[const
   //declaraciones de constantes]
[tipos
   //declaraciones de tipos]
[var
   //declaraciones de variables]
//Cuerpo del programa
inicio
   …
fin
```

2.1. Declaración de tipos de datos estructurados

Arrays

```
array[<dimensión>…] de <tipo_de_dato> : <nombre_del_tipo>
```

<dimensión> es un subrango con el índice del límite inferior y el límite superior. Por ejemplo, **array**[5..20] **de entero** declararía un array de 16 elementos enteros. Pueden aparecer varios separados por comas para declarar arrays de más de una dimensión.

<tipo_de_dato> es el identificador de cualquier tipo de dato estándar o definido por el usuario.

<nombre_del_tipo> es un identificador válido que se utilizará para referenciar el tipo de dato.

El acceso a un elemento de un array se realizará indicando su índice entre corchetes. El índice será una expresión entera.

Registros

```
registro : <nombre_del_tipo>
   <tipo_de_dato> : <nombre_del_campo>
   …
fin_registro
```

<tipo_de_dato> es el identificador de cualquier tipo de dato estándar o definido por el usuario.

<nombre_del_tipo> es un identificador válido que se utilizará para referenciar el tipo de dato.

<nombre_del_campo> es un identificador válido que se utilizará para referenciar el campo del registro.

El acceso a un campo de una variable de tipo registro se realizará utilizando el carácter punto (.), por ejemplo MiRegistro.MiCampo.

Archivos secuenciales

```
archivo_s de <tipo_de_dato> : <nombre_del_tipo>
```

<tipo_de_dato> es el identificador de cualquier tipo de dato estándar o definido por el usuario.

<nombre_del_tipo> es un identificador válido que se utilizará para referenciar el tipo de dato.

Archivos directos

```
archivo_d de <tipo_de_dato> : <nombre_del_tipo>
```

<tipo_de_dato> es el identificador de cualquier tipo de dato estándar o definido por el usuario.

<nombre_del_tipo> es un identificador válido que se utilizará para referenciar el tipo de dato.

2.2. Declaración de constantes

Se realiza dentro de la sección de declaraciones de constantes.

```
<nombre_de_constante> = <expresión>
```

<nombre_de_constante> es un identificador válido que se utilizará para referenciar la constante.

<expresión> es una expresión válida. El tipo de la constante será el tipo de dato que devuelva la expresión.

2.3. Declaración de variables

Se realiza dentro de la sección de declaraciones de variables.

```
<tipo_de_dato> : <nombre_de_variable>[= <expresión>]…
```

<tipo_de_dato> es el identificador de cualquier tipo de dato estándar o definido por el usuario.

<nombre_de_variable> es un identificador válido que se utilizará para referenciar la variable. En una declaración es posible declarar varias variables separadas por comas.

Es posible inicializar la variable en la declaración, <expresión> es una expresión válida del tipo de dato de la variable.

2.4. Biblioteca de funciones

Funciones aritméticas

Función	Significado
abs(x)	Devuelve el valor absoluto de la expresión numérica x
aleatorio()	Devuelve un número aleatorio real mayor o igual que 0 y menor que 1
arctan(x)	Devuelve la arco tangente de x
cos(x)	Devuelve el coseno de x
entero(x)	Devuelve el primer valor entero menor que la expresión numérica x
exp(x)	Devuelve el valor e^x
ln(x)	Devuelve el logaritmo neperiano de x.
log10(x)	Devuelve el logaritmo en base 10 de x.
raiz2(x)	Devuelve la raiz cuadrada de x
sen(x)	Devuelve el seno de x
trunc(x)	Trunca (elimina los decimales) de la expresión numérica x.

Funciones de cadena

Función	Significado
longitud(c)	Devuelve el número de caracteres de la cadena c.
posición(c,sc)	Devuelve la posición de la primera aparición de la subcadena sc en la cadena c
subcadena(c,ini[,long])	Devuelve una subcadena de la cadena c formada por todos los caracteres a partir de la posición ini. Si se incluye el argumento long, devuelve sólo los primeros long caracteres a partir de la posición ini.

Funciones de conversión de número a cadena

Función	Significado
código(car)	Devuelve el código ASCII del carácter car.
carácter(x)	Devuelve el carácter correspondiente al código ASCII x
valor(c)	Convierte la cadena c a un valor número. Si el contenido de la cadena c no puede converitirse a un valor numérico (contiene caracteres alfabéticos, signos de puntuación inválidos, etc), devuelve 0
cadena(x)	Convierte a cadena el valor numérico x

Funciones de información

Función	Significado
tamaño_de(<variable>)	Devuelve el tamaño en bytes de la variable

2.5. Procedimientos de entrada/salida

leer(<lista_de_variables>), lee una o más variables desde la consola del sistema.

escribir(<lista_de_expresiones>), escribe una o más expresiones en la consola del sistema

2.6. Instrucción de asignación

<variable> ă <expresión>

Primero evalúa el valor de la expresión y lo asigna a la variable. La variable y la expresión deben ser del mismo tipo de dato.

3. Estructuras de control

3.1. Estructuras selectivas

Estructura selectiva simple y doble

```
si <expresión_lógica> entonces
   <acciones>
[si_no
   <acciones>
fin_si
```

Estructura selectiva múltiple

```
según_sea <expresión> hacer
   <lista_de_valores> : <acciones>
   …
[si_no
   <acciones>]
fin_según
```

<expresión> puede ser cualquier expresión validad.

<lista_de_valores> será uno o más valores separados por comas de mismo tipo que <expresión>. La estructura verifica si el valor de la expresión concide con alguno de los valores de la primera lista de valores>; si esto ocurre realiza las acciones correspondientes y el flujo de control sale de la estructura, en caso contrario evalua la siguiente lista. Las acciones de la claúsula **si_no** se ejecutará si ningún valor coincide con la <expresión>.

3.2. Estructuras repetitivas

Estructura mientras

```
mientras <expresión_lógica> hacer
   <acciones>
fin_mientras
```

Estructura repetir

```
repetir
   <acciones>
hasta_que <expresión_lógica>
```

Estructura desde

```
desde <variable> ă <valor_inicial> hasta <valor_final>
          [incremento | decremento <valor_incremento>] hacer
   <acciones>
fin_desde
```

<variable>, puede ser cualquier variable en la que se pueda incrementar o decrementar su valor, es decir todas las numéricas, las de tipo carácter y las lógicas.

<valor_inicial>, es una expresión con el primer valor que toma la variable del bucle. Debe ser del mismo tipo que la variable del bucle.

<valor_final>, es una expresión con el último valor que toma la variable del bucle. Debe ser del mismo tipo que la variable del bucle. El bucle finaliza cuando la variable toma un valor mayor que este valor inicial.

<valor_incremento> es una expresión con el valor en el que se incrementará o decrementará la variable del bucle al final de cada iteración.

4. Programación modular

4.1. Cuestiones generales

El ámbito de las variables declaradas dentro de un módulo (procedimiento o función) es local, y el tiempo de vida de dicha variable será el tiempo de ejecución del módulo.

4.2. Procedimientos

Declaración

```
procedimiento <nombre_procedimiento>([<lista_parámetros_formales>])
[declaraciones locales]
inicio
   …
fin_procedimiento
```

<nombre_procedimiento> debe ser un identificador válido.

<lista_parámetros_formales> son uno o más grupos de parámetros separados por punto y coma. Cada grupo de argumentos se define de la siguiente forma:

{**E** | **E/S**} <tipo_de_dato> **:** <lista_de_parámetros>

E indica que el paso de parámetros se realiza por valor.

E/S indica que el paso de parámetros se realiza por referencia.

<tipo_de_dato> es un tipo de dato estándar o definido previamente por el usuario.

<lista_de_parámetros> es uno o más identificadores válidos separados por comas.

Llamada a procedimientos

[**llamar_a**] <nombre_procedimiento>([<lista_parámetros_actuales>])

La lista de parámetros actuales es una o varias variables o expresiones separadas por comas que deben coincidir en número, orden y tipo con la lista de parámetros formales de la declaración.

4.3. Funciones

Declaración

```
<tipo_de_dato> :función <nombre_función>([<lista_parámetros_formales>])
[declaraciones locales]
inicio
   …
   devolver(<expresión>)
fin_función
```

<tipo_de_dato> es un tipo de dato estándar o definido previamente por el usuario. Se trata del tipo del dato que devuelve la función.

<nombre_función> debe ser un identificador válido.

<lista_parámetros_formales> son uno o más grupos de parámetros separados por punto y coma. Cada grupo de argumentos se define de la siguiente forma:

{**E** | **E/S**} <tipo_de_dato> **:** <lista_de_parámetros>

E indica que el paso de parámetros se realiza por valor.

E/S indica que el paso de parámetros se realiza por referencia.

<tipo_de_dato> es un tipo de dato estándar o definido previamente por el usuario.

<lista_de_parámetros> es uno o más identificadores válidos separados por comas.

<expresión> es el valor de retorno de la función. Debe coincidir con el tipo de dato de la declaración.

Llamada a funciones

<nombre_función>([<lista_parámetros_actuales>])

La lista de parámetros actuales es una o varias variables o expresiones separadas por comas que deben coincidir en número, orden y tipo con la lista de parámetros formales de la declaración. Al devolver un valor y no existir funciones que no devuelven valores (funciones `void` de C o Java), la llamada debe hacerse siempre dentro de una expresión.

5. Archivos

5.1. Archivos secuenciales

Apertura del archivo

abrir(<variable_tipo_archivo,<modo_apertura>,<nombre_archivo>)

<var_tipo_archivo> es una variable de tipo archivo secuencial.

<modo_apertura> indica el tipo de operación que se realizará con el archivo. En el caso de archivos secuenciales será:

lectura, coloca el puntero al siguiente registro al comienzo del archivo y sólo realiza operaciones de lectura. El archivo debe existir previamente.

escritura, coloca el puntero al siguiente registro al comienzo del archivo y sólo realiza operaciones de escritura. Si el archivo no exisite, primero crea un archivo vacío. Si el archivo existe, sobrescribe los datos que tenga.

añadir, coloca el puntero al siguiente registro en la marca de final de archivo y sólo realiza operaciones de escritura.

<nombre_archivo>, es una expresión de cadena con el nómbre que el sistema dará al archivo.

Cierre del archivo

cerrar(<lista_variables_tipo_archivo>)

Cierra el archivo o archivos abiertos previamente.

Entrada/salida

leer(<variable_tipo_archivo>, <variable>)

Leer del archivo abierto para lectura representado por <variable_tipo_archivo> el siguiente registro. El tipo de la variable debe coincidir con el tipo base del archivo definido en la declaración del tipo de dato.

escribir(<variable_tipo_archivo>, <expresión>)

Escribe en el archivo abierto para escritura y representado por la variable de tipo archivo el valor de la expresión. El tipo de la expresión debe coincidir con el tipo base del archivo definido en la declaración del tipo de dato.

5.2. Archivos de texto

Se considera el archivo de texto como un tipo especial de archivo compuesto de caracteres o cadenas. La declaración de un tipo de dato de tipo archivo de texto sería por tanto:

archivo_s de carácter : <nombre_tipo>

archivo_s de cadena : <nombre_tipo>

La lectura de un carácter único en un archivo de texto se haría de la forma **leer**(<variable_tipo_carácter>) que leería el siguiente carácter del archivo. La lectura de una variable de tipo cadena (**leer**(<variable_tipo_cadena>)), leería todos los caracteres hasta el final de línea.

La escritura de datos en un archivo de texto también se podrá hacer carácter a carácter (**leer**(<variable_tipo_carácter>)) o línea a línea (**leer**(<variable_tipo_cadena>))

La detección del final de línea en un archivo de texto cuando se lee carácter a carácter se realizaría con la función **fdl**:

fdl(<variable_tipo_archivo>)

La función **fdl** devuelve el valor lógico **verdad**, si el último carácter leído es el carácter de fin de línea.

5.3. Archivos directos

Apertura del archivo

abrir(<variable_tipo_archivo,<modo_apertura>,<nombre_archivo>)

<var_tipo_archivo> es una variable de tipo archivo directo.

<modo_apertura> indica el tipo de operación que se realizará con el archivo. En el caso de archivos directos será:

lectura, coloca el puntero al siguiente registro al comienzo del archivo y sólo realiza operaciones de lectura. El archivo debe existir previamente.

escritura, coloca el puntero al siguiente registro al comienzo del archivo y sólo realiza operaciones de escritura. Si el archivo no exisite, primero crea un archivo vacío. Si el archivo existe, sobrescribe los datos que tenga.

lectura/escritura, coloca el puntero al comienzo del archivo y permite operaciones tanto de lectura como de escritura.

<nombre_archivo>, es una expresión de cadena con el nómbre que el sistema dará al archivo.

Cierre del archivo

cerrar(<lista_variables_tipo_archivo>)

Cierra el archivo o archivos abiertos previamente.

Acceso secuencial

leer(<variable_tipo_archivo>, <variable>)

Leer del archivo abierto para lectura representado por <variable_tipo_archivo> el siguiente registro. El tipo de la variable debe coincidir con el tipo base del archivo definido en la declaración del tipo de dato.

escribir(<variable_tipo_archivo>, <expresión>)

Escribe en el archivo abierto para escritura y representado por la variable de tipo archivo el valor de la expresión. El tipo de la expresión debe coincidir con el tipo base del archivo definido en la declaración del tipo de dato.

Acceso directo

leer(<variable_tipo_archivo>, <posición>, <variable>)

Leer el registro situado en la posición relativa <posición> y guarda su contenido en la variable.

escribir(<variable_tipo_archivo>, <posición> <variable>)

Escribe el contenido de la variable en la posición relativa <posición>.

5.4. Consideraciones adicionales

Detección del final del archivo

Al cerrar un archivo abierto para escritura se coloca después del último registro la marca de fin de archivo. La función **fda** permite detectar si se ha llegado a dicha marca.

fda(<variable_tipo_archivo>)

Devuelve el valor lógico **verdad**, si se ha intentado hacer una lectura secuencial después del último registro.

Determinar el tamaño del archivo

La función **lda** devuelve el número de bytes del archivo.

lda(<nombre_archivo>)

<nombre_archivo> es el nombre del archivo físico.

Para determinar el número de registros de un archivo se puede utilizar la expresión:

lda(<nombre_archivo>) / **tamaño_de**(<tipo_base_archivo>)

Otros procedimientos

borrar(<nombre_archivo>)

Elimina del disco el archivo representado por la expresión de cadena <nombre_archivo>. El archivo debe estar cerrado.

renombrar(<nombre_archivo>,<nuevo_nombre>)

Cambia el nombre al archivo <nombre_archivo> por el de <nuevo_nombre>. El archivo debe estar cerrado.

6. Variables dinámicas

Declaración de tipos de datos dinámicos

puntero_a <tipo_de_dato> **:** <nombre_del_tipo>

Declara el tipo de dato <nombre_del_tipo> como un puntero a variables de tipo <tipo_de_dato>.

El valor constante **nulo**, indica una referencia a un puntero nulo.

Referencia al contenido de una variable dinámica

<variable_dinamica>⇒

Asignación y liberación de memoria con variables dinámicas

reservar(<variable_dinámica>)

Reserva espacio en memoria para una variable del tipo de dato del puntero y hace que la variable dinámica apunte a dicha zona.

liberar(<variable_dinamica>)

Libera el espacio de memoria apuntado por la variable dinámica. Dicha variable queda con un valor indeterminado.

7. Programación orientada a objetos

7.1. Clases y objetos

Declaración de clases.

```
clase <nombre_de_clase>
   //Declaración de atributos
   //Declaración de constructores y métodos
fin_clase
```

<nombre_de_clase> es un identificador válido.

Declaración de tipos de referencias

<nombre_de_clase> : <nombre_de_referecia>

<nombre_de_clase> es el nombre de una clase previamente declarada.

<nombre_de_referencia> es un identificador válido que se utilizará para referenciar a un objeto de dicha clase.

La declaración de una referencia a una clase se hará en la sección de declaraciones de variables o tipos de datos de un algoritmo, o dentro de la sección de variables de otra clase.

Instanciación de clases

nuevo <nombre_de_constructor>([<argumentos_constructor>])

La declaración **nuevo** reserva espacio para un nuevo objeto de la clase a la que pertenece el constructor y devuelve una referencia a un objeto de dicha clase. <nombre_de_constructor> tendrá el mismo nombre de la clase a la que pertenece. La llamada al constructor puede llevar argumentos para la inicialización de atributos (véase más adelante en el apartado de constructores).

La instanciación se se puede realizar en una sentencia de asignación, o en la propia declaración de la referencia, dentro de la sección de declaraciones del algoritmo.

```
MiObjeto ă nuevo MiClase(arg1,arg2,arg3) //Dentro del código ejecutable
MiClase : MiObjeto = nuevo MiClase(arg1,arg2,arg3) //En la sección var
```

Referencias a miembros de una clase

```
NombreReferencia.nombreDeMiembro //Para atributos
NombreReferencia.nombreDeMiembro([listaParamActuales]) //Para métodos
```

Constructores

```
constructor <nombre_de_clase>[<lista_parametros_formales>])
//Declaración de variables locales
inicio
   //Código del constructor
fin_constructor
```

Existe un constructor por omisión sin argumentos al que se le llama mediante <nombre_de_clase()>. Al igual que con los métodos se admite la sobrecarga dentro de constructores, distinguiéndose los distintos constructores por el orden, número y/o tipo de sus argumentos.

Puesto que la misión de un constructor es inicializar una instancia de una clase, la <lista_de_parámetros_formales> sólo incluyen argumentos de entrada, por lo que se puede omitir la forma en la que se pasan los argumentos.

Destructores

No se considera la existencia de destructores. Las instancias se consideran destruidas cuando se ha perdido una referencia a ellas (recolector de basura).

Visibilidad de las clases

Se consideran todas las clases como públicas, es decir, es posible acceder a los miembros de cualquier clase declarada en cualquier momento.

Referencia a la instancia desde dentro de la declaración de una clase

Es posible hacer referencia a una instancia desde dentro de una clase con la palabra reservada **instancia** que devuelve una a la instancia de la clase que ha realizado la llamada al método. De esta forma **instancia.**UnAtributo, haría referencia al valor del atributo UnAtributo dentro de la instancia actual.

7.2. Atributos

Declaración de atributos

La declaración de los atributos de una clase se realizará dentro de la sección de declaraciones **var** de dicha clase.

```
var
   [const][privado|publico|protegido][estático]
          <tipo_de_dato> : <nombre_atributo> [ = <valor_inicial>]
   …
```

<nombre_atributo> puede ser cualquier identificador válido.

<tipo_de_dato> puede ser cualquier tipo de dato estándar, definido por el usuario o otra clase declarada con anterioridad.

Es posible dar un valor inicicia al atributo mediante una expresión de inicialización que deberá ser del mismo tipo de dato que el atributo.

Visibilidad de los atributos

Por omisión se considera a los atributos privados, es decir, sólo son accesibles por los miembros de la clase. Para que pueda ser utilizado por los miembros de otras clases de utilizará el modificador **público**. El modificador **protegido** se utiliza para que sólo pueda ser utilizado por los miembros de su clase y por los de sus clases hijas.

Atributos de clase (estáticos)

Un atributo que tenga el modificador **estático** no pertenece a ninguna instancia de la clase, sino que será común a todas ellas. Para hacer referencia a un atributo de una clase se utilizará el nombre de la clase seguido del nombre del atributo. (MiClase.MiAtributoEstático)

Atributos constantes

El modificador **const** permite crear atributos constantes que no se modificarán durante el tiempo de vida de la instancia.

7.3. Métodos

Declaración de métodos

La declaración de métodos se realizará dentro de la clase después de la declaración de atributos sin indicar ninguna sección especial.

```
[estático][abstracto][público|privado|protegido] <tipo_de_retorno>
   método <nombre_del_método>(<lista_de_parámetros_formales>)
//declaración de variables
inicio
  //Código
  [devolver(<expresión>)]
fin_método
```

<nombre_del_método> es un identificador válido.

<tipo_de_retorno> es cualquier tipo de dato estándar, estructurado o una referencia a un objeto. La declaración **devolver** se utiliza para indicar el dato de retorno que devuelve la función que debe coincidir con el tipo de retorno que aparece en la declaración. Si el método no devuelve valores se utilizará la palabra reservada **nada** y no aparecerá la palabra **devolver**.

La lista de parámetros formales se declararía igual que en los procedimientos y funciones. El paso de argumentos se realizará como en los procedimiento y funciones normales.

Las variables locales se declararán en la sección **var** entre la cabecera del método y su cuerpo.

Visibilidad de los métodos

Por omisión, se consideran los métodos como públicos, es decir, es posible acceder a ellos desde cualquier lugar del algoritmo. Para que pueda ser utilizado sólo por miembros de su clase de utilizará el modificador **privado**,en el caso de los procedimientos, o **privada**, en el caso de las funciones. El modificador **protegido** o **protegida** se utiliza para que sólo pueda ser utilizado por los miembros de su clase y por los de sus clases hijas.

Métodos estáticos

Un método que tenga el modificador **estático** o **estática** no pertenece a ninguna instancia de la clase, sino que será común a todas ellas. Para hacer referencia a un método de una clase se utilizará el nombre de la clase seguido del nombre del método. (`MiClase.MiMétodoEstático()`)

Sobrecarga de métodos

Se permite la sobrecarga de métodos, es decir, la declaración de métodos con el mismo nombre pero con funcionalidades distintas. Para que en la llamada se pueda distinguir entre los métodos sobrecagados, número, orden o tipo de sus argumentos deben cambiar.

Ligadura de métodos

La ligadura de la llamada de un método con el método correspondiente se hace **siempre** de forma dinámica, es decir, en tiempo de ejecución, con lo que se permite la existencia de **polimorfismo**.

7.4. Herencia

clase <clase_derivada> **hereda_de** <superclase>

<clase_derivada> es un identificador válido.

<superclase> es una clase declarada anteriormente.

La clase derivada:

- Hereda todos los métodos y atributos de la superclase accesibles (atributos públicos y protegidos y métodos públicos y protegidos) presentes sólo en la superclase.
- Sobreescribre todos los métodos y atributos de la superclase accesibles (atributos públicos y protegidos y métodos públicos y protegidos) presentes en ambas clases
- Añade todos los métodos y atributos presentes sólo en la clase derivada.

Es posible accede acceder a atributos de la superclase o ejecutar sus métodos mediante la palabra reservada **super**.

- Referencia a un miembro de la superclase `super.nombreMiembro()`.
- Referencia al constructor de la superclase: `super()`.

Clases y métodos abstractos

Clases en las que algunos o todos los miembros no tienen implementación, por lo que no pueden instanciarse directamente. Servirán de clase base para clases derivadas.

abstracta clase <clase_base>

Aquellos métodos sin implementación se podrían declarar sin inicio ni fin de método.

```
abstracta TipoDato: función NombreMétodo([paramFormales])
abstracto procedimiento NombreMétodo([paramFormales])
```

En estos casos, la clases hijas deberían implemtar el método.

Herencia múltiple

No se considera la existencia de herencia múltiple.

8. Palabras reservadas

símbolo, palabra	Traducción / Significado
-	Menos unario (negativo)
-	Resta
&	Concatenación
ă	Operador de asignación
⇒	Referencia a una variable apuntada
*	Multiplicación
.	Cualificador de acceso a registros o a miembros de una clase
.	Separador de decimales
/	División real
//	Comentario de una sóla línea. Ignora todo lo que aparezca a continuación de la línea.
[]	Índice de array
^	Exponenciación
{	Inicio de comentario multilínea. Ignora todo lo que aparezca hasta encontrar el carácter de final de comentario (})
}	Fin de comentario multilínea. Ignora todo lo que aparezca desde el carácter de inicio de comentario ({)
'	Comilla simple, delimidador de datos de tipo carácter o cadena.
"	Comilla doble, delimitador de datos de tipo carácter o cadena.
+	Más unario (positivo)
+	Suma
+	Concatenación
<	Menor que
<=	Menor o igual que
<>	Distinto de
=	Igual a
>	Mayor que
>=	Mayor o igual que
abrir	Abre un archivo
abs(x)	Devuelve el valor absoluto de la expresión numérica x
abstracto	Declaración de métodos abstractos (sin implementación)
aleatorio()	Devuelve un número aleatorio real mayor o igual que 0 y menor que 1
algoritmo	`program`, inicio del pseudocódigo.
añadir	Modo de apertura de un archivo
arctan(x)	Devuelve la arco tangente de x
archivo_d	Declaración de archivos directos
archivo_s	Declaración de archivos secuenciales
array	Declaración de arrays
borrar	Borra un archivo del disco
cadena	`string`
cadena(x)	Convierte a cadena el valor numérico x
carácter	`char`
carácter(x)	Devuelve el carácter correspondiente al código ASCII x
cerrar	Cierra un archivo
cerrar	Cierra uno o más archivos abiertos
clase	Inicio de la declaración de una clase
código(car)	Devuelve el código ASCII del carácter car.
const	Inicio de la sección de declaraciónes de constantes
const	Declaración de atributos constantes en la definición de clases
constructor	Inicio de la declaración de un constructor
cos(x)	Devuelve el coseno de x
decremento	Decremento en estructuras repetitivas desde
desde	Inicio de estructura repetitiva desde, `for`
devolver	Indica el valor de retorno de una función

símbolo, palabra	Traducción / Significado
div	División entera
e	Exponente
e	Paso de argumentos por valor
e/s	Paso de argumentos por referencia
entero	`integer`, `int`, `long`, `byte`, etc.
entero`(x)`	Devuelve el primer valor entero menor que la expresión numérica x
entonces	`then`
escribir	Escribie una o más expresiones en un dispositivo de salida (consola, archivo, etc.)
escritura	Modo de apertura de un archivo
estático	Declaración de atributos o métodos de clase o estáticos
exp`(x)`	Devuelve el valor e^x
falso	Falso, `false`
fda	Fin de archivo
fdl	Fin de línea
fin	Fin de algoritmo
fin_clase	Final de la declaración de una clase
fin_constructor	Fin de la declaración de un constructor
fin_desde	Fin de estructura repetitiva desde
fin_función	Fin de la declaración de una función
fin_mientras	Fin de estructura repetitiva mientras
fin_procedimiento	Fin de un procedimiento
fin_registro	Fin de la declaración de registro
fin_según	Fin de estructura selectiva múltiple
fin_si	`end if`, fin de estructura selectiva simple
función	Inicio de la declaración de una función
hacer	`do`
hasta	`to`
hasta_que	Fin de estructura repetitiva repetir
hereda_de	Indica que una clase derivada hereda miembros de una superclase
incremento	Incremento en estructuras repetitivas desde
inicio	Inicio del código ejecutable de un algoritmo, módulo, constructor, etc.
instancia	Referencia a la instancia actual de la clase donde aparece
lda	Devuelve la longitud en bytes de un archivo
lectura	Modo de apertura de un archivo
lectura/escritura	Modo de apertura de un archivo
leer	Lee una o más variables desde un dispositivo de entrada (consola, archivo, etc.)
liberar	Libera el espacio asignado a una variable dinámica
ln`(x)`	Devuelve el logaritmo neperiano de x.
log10`(x)`	Devuelve el logaritmo en base 10 de x.
longitud`(c)`	Devuelve el número de caracteres de la cadena c.
llamar_a	Instrucción de llamada a un procedimiento
mientras	`while`, inicio de estructura repetitiva mientras
mod	Módulo de la división entera
nada	Tipo de retorno de métodos que no devuelven valores, `void`
no	`Not`
nuevo	Reserva espacio en memoria para un objeto de una clase y devuelve una referencia a dicho objeto.
nulo	Constante de puntero nulo
o	`Or`
posición`(c,sc)`	Devuelve la posición de la primera aparición de la subcadena sc en la cadena c
privado	Modificador de acceso privado a un atributo o método
procedimiento	Inicio de la declaración de un procedimiento
protegido	Modificador de acceso a un atributo o método que permite el acceso a los miembros de su clase y de las clases hijas
público	Modificador de acceso público a un atributo o método
puntero_a	Declaración de tipos de datos de asignación dinámica
raiz2`(x)`	Devuelve la raiz cuadrada de x

símbolo, palabra	Traducción / Significado
real	`float`, `double`, `single`, `real`, etc.
registro	`record`, inicio de la declaración de registro
renombrar	Cambia el nombre de un archivo
repetir	`repeat`, inicio de estructura repetitiva repetir
reservar	Reserva espacio en memoria para una variable dinámica
según_sea	Inicio de estructura selectiva múltiple, `case`, `select case`, `switch`.
sen`(x)`	Devuelve el seno de x
si	Inicio de estructura selectiva simple / doble, `if`
si_no	`else`
subcadena`(c,ini[,long])`	Devuelve una subcadena de la cadena c formada por todos los caracteres a partir de la posición ini. Si se incluye el argumento long, devuelve sólo los primeros long caracteres a partir de la posición ini.
super	Permite el acceso a miembros de la superclase.
tamaño_de`(x)`	Devuelve el tamaño en bytes de la variable x
tipos	Inicio de la sección de declaraciónes de tipos de datos
trunc`(x)`	Trunca (elimina los decimales) de la expresión numérica x.
valor`(c)`	Convierte la cadena c a un valor número. Si el contenido de la cadena c no puede converitirse a un valor numérico (contiene caracteres alfabéticos, signos de puntuación inválidos, etc), devuelve 0
var	Inicio de la sección de declaraciones de variables, o de la declaración de atributos de una clase
verdad	Verdadero, `true`
y	`and`