UT1. INTRODUCCIÓN A LA PROGRAMACIÓN

Patricia Cilleros

ÍNDICE

- El proceso de programación
- Lenguajes de programación.
- Paradigmas de programación.
- Datos, algoritmos y programas.
- Herramientas y entornos para el desarrollo de programas.

ÍNDICE

- Identificación de los elementos de un programa informático:
  - Estructura y bloques fundamentales.
  - Comentarios.
  - Identificadores.
  - Palabras reservadas.
  - Variables. Declaración, inicialización y utilización. Almacenamiento en memoria.
  - Tipos de datos.
  - Literales.
  - Constantes.
  - Operadores y expresiones. Precedencia de operadores
  - Conversiones de tipo.Implícitas y explicitas (casting).

EL PROCESO DE PROGRAMACIÓN

- Un programa se escribe en un lenguaje de programación y las operaciones que conducen a expresar un algoritmo en forma de programa se llaman programación. Así pues, los lenguajes utilizados para escribir programas de computadoras son los lenguajes de programación y programadores son los escritores y diseñadores de programas.
- En la realidad la computadora no entiende directamente los lenguajes de programación sino que se requiere un programa que traduzca el código fuente a otro lenguaje que sí entiende la máquina directamente. Este lenguaje se conoce como lenguaje máquina y el código correspondiente código máquina.
- Cada lenguaje de programación tiene un conjunto o "juego" de instrucciones (acciones u operaciones que debe realizar la máquina) que la computadora podrá entender directamente en su código máquina o bien se traducirán a dicho código máquina.

EL PROCESO DE PROGRAMACIÓN

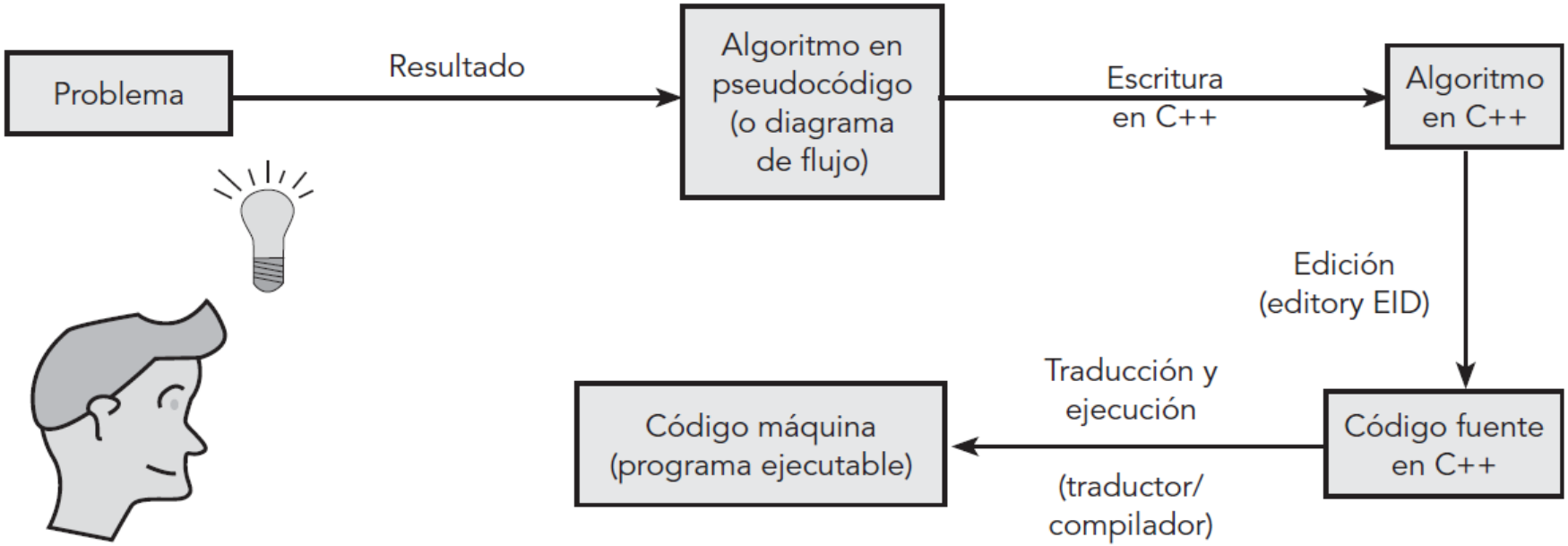


Figura 1.8 Proceso de transformación de un algoritmo en pseudocódigo en un programa ejecutable.

LENGUAJES DE PROGRAMACIÓN

Los principales tipos de lenguajes de programación son:

- Lenguajes máquina.
- Lenguajes de bajo nivel (ensambladores).
- Lenguajes de alto nivel.

LENGUAJES DE PROGRAMACIÓN

- **Lenguajes máquina:**
- En la década de 1940 cuando nacían las primeras computadoras digitales el lenguaje que se utilizaba para programar era el lenguaje máquina que traducía directamente el código máquina (código binario) comprensible para las computadoras.
- Los investigadores de la época simplificaron el proceso de programación inventando sistemas de notación en los cuales las instrucciones se representaban en formatos nemónicos (nemotécnicos) en vez de en formatos numéricos que eran más difíciles de recordar.

```
Mover el contenido del registro 4 al registro 8
```

se podía expresar en lenguaje máquina como

```
4048          o bien          0010 0000 0010 1000
```

en código nemotécnico podía aparecer como

```
MOV           R5,             R6
```

LENGUAJES DE PROGRAMACIÓN

- **Lenguajes de bajo nivel (ensambladores):**
- Para convertir los programas escritos en código nemotécnico a lenguaje máquina, se crearon programas ensambladores (assemblers). Es decir, los ensambladores son programas que traducen otros programas escritos en código nemotécnico en instrucciones numéricas en lenguaje máquina que son compatibles y legibles por la máquina.
- A estos lenguajes se les denominó de **segunda generación**, reservando el nombre de **primera generación** para los **lenguajes de máquina**.

LENGUAJES DE PROGRAMACIÓN

- **Lenguajes de alto nivel:**
- En las décadas de 1950 y 1960 comenzaron a desarrollarse lenguajes de programación de **tercera generación** que diferían de las generaciones anteriores en que sus instrucciones o primitivas eran de alto nivel (comprensibles por el programador, como si fueran lenguajes naturales) e independientes de la máquina.
- Estos lenguajes se llamaron **lenguajes de alto nivel**. Los ejemplos más conocidos son FORTRAN (FORmula TRANslator) que fue desarrollado para aplicaciones científicas y de ingeniería, y COBOL (COmmon Business Oriented Language), que fue inventado por la U.S. Navy de Estados Unidos, para aplicaciones de gestión o administración.
- Con el paso de los años aparecieron nuevos lenguajes como Pascal, BASIC, C, C++, Ada, Java, C#, HTML, XML…

PARADIGMAS DE PROGRAMACIÓN

- La evolución de los lenguajes de programación ha ido paralela a la idea de paradigma de programación.
- Un paradigma de programación representa fundamentalmente enfoques diferentes para la construcción de soluciones a problemas y por consiguiente afectan al proceso completo de desarrollo de software.
- Los paradigmas de programación clásicos son: **procedimental (o imperativo), funcional, declarativo y orientado a objetos.**

PARADIGMAS DE PROGRAMACIÓN

- **Lenguajes imperativos (procedimentales)**
- El paradigma imperativo o procedimental representa el enfoque o método tradicional de programación.
- Un lenguaje imperativo es un conjunto de **instrucciones que se ejecutan una por una**, de principio a fin, de modo secuencial excepto cuando intervienen instrucciones de salto de secuencia o control. Este paradigma define el proceso de programación como el desarrollo de una secuencia de órdenes (comandos) que manipulan los datos para producir los resultados deseados.

- **Lenguajes declarativos**
- En contraste con el paradigma imperativo el paradigma declarativo solicita al programador que describa el problema en lugar de encontrar una solución algorítmica al problema. Se basa en la lógica formal y en el cálculo de predicados de primer orden.
- El lenguaje declarativo por excelencia es Prolog.

PARADIGMAS DE PROGRAMACIÓN

- **Lenguajes orientados a objetos**
- El paradigma orientado a objetos se asocia con el proceso de programación llamado programación orientada a objetos (POO) consistente en un enfoque totalmente distinto al proceso procedimental. El enfoque orientado a objetos guarda **analogía con la vida real**. **El desarrollo de software OO se basa en el diseño y construcción de objetos que se componen a su vez de datos y operaciones que manipulan esos datos.** El programador define en primer lugar los objetos del problema y a continuación los datos y operaciones que actuarán sobre esos datos.

PARADIGMAS DE PROGRAMACIÓN

- **Lenguajes orientados a objetos**
- Las **ventajas** de la programación orientada a objetos se derivan esencialmente de la estructura modular existente en la vida real y el modo de respuesta de estos módulos u objetos a mensajes o eventos que se producen en cualquier instante.
- C++, lenguaje orientado a objetos, por excelencia, es una extensión del lenguaje C y contiene las tres **propiedades más importantes: encapsulamiento, herencia y polimorfismo**.
- Hoy día Java y C# son herederos directos de C++ y C, y constituyen los lenguajes orientados a objetos más utilizados en la industria del software del siglo XXI.

EJERCICIOS

- Clasifica los siguientes lenguajes de programación en:
  - Funcional
  - Orientado a objetos
  - Imperativa
  - Declarativa

Lenguajes:
COBOL, LISP, BASIC, LM, ADA, SCHEME, PASCAL, SMALLTALK,VISUAL BASIC, APL, C++, C#, FORTRAN, ALGOL, C, GPSS, JAVA, PROLOG.

HERRAMIENTAS Y ENTORNOS PARA EL DESARROLLO DE PROGRAMAS.

- El IDE que utilizaremos para programar será eclipse. Es gratuito y podéis descargarlo de su web aquí.

- Para ver la descargar e instalación puedes consultar este tutorial.

ESTRUCTURA Y BLOQUES FUNDAMENTALES

- **Clase**: La unidad básica de un programa en Java.
- **Método**: Un bloque de código reutilizable que realiza una tarea (en este caso main y otroMetodo).
- **Bloques de código**: Definidos por llaves {} y ejecutan un conjunto de instrucciones.
- **La aplicación se ejecuta desde el método principal main()** situado en una clase. Dicha clase es la clase principal (por ella se **empieza a ejecutar el programa**).

```
// Clase principal
public class EjemploEstructura {

// Método principal: punto de entrada del programa
   public static void main(String[] args) {
// Bloque de código que se ejecuta al iniciar el programa
      System.out.println("Hola, mundo!");
   }

   // Otro método de la clase
   public static void otroMetodo() {
      // Bloque de código independiente
      int a = 5;
      System.out.println("El valor de a es: " + a);
   }
}
```

COMENTARIOS

- Las primeras líneas que forman parte de un programa suelen estar compuestas de comentarios acerca del nombre
- del programa, el programador, la fecha de creación, observaciones…
- Aparecen con // si lo que viene a continuación es texto en una sola línea o con /* ….. */ cuando se trata de más de una
- línea.
- Hay en Java otro tipo de comentario /** comentario de documentación, de una o más líneas */ -> **Javadoc**
- **Es recomendable poner comentarios en el código para facilitar su mantenimiento y entendimiento por otros programadores.**

PALABRAS RESERVADAS

- **Palabras reservadas**: Son términos predefinidos que tienen un significado especial en Java y no se pueden usar como identificadores (ej. class, if, else, public, etc.).

abstract	continue	for	new	switch
boolean	default	goto	null	synchronized
break	do	if	package	this
byte	double	implements	private	threadsafe
byvalue	else	import	protected	throw
case	extends	instanceof	public	transient
catch	false	int	return	true
char	final	interface	short	try
class	finally	long	static	void
const	float	native	super	while

TIPOS DE DATOS

- **Primitivos**: Tipos básicos como int, double, boolean, char.
- **No primitivos**: Tipos de objetos o referencias como String, ArrayList, Object.

```
public static void main(String[] args) {
        // Tipos primitivos
        int entero = 10;
        double decimal = 3.14;
        boolean esVerdad = true;
        char letra = 'A';

        // Tipos no primitivos (objetos)
        String texto = "Hola";

        System.out.println("Entero: " + entero);
        System.out.println("Decimal: " + decimal);
        System.out.println("Booleano: " + esVerdad);
        System.out.println("Caracter: " + letra);
        System.out.println("Texto: " + texto);
    }
```

VARIABLES: DECLARACIÓN, INICIALIZACIÓN Y UTILIZACIÓN. ALMACENAMIENTO EN MEMORIA

- **Declaración**: Informar al compilador sobre el tipo de una variable (int edad). Se distinguen las mayúsculas de las minúsculas y no hay longitud máxima. Por convenio empiezan por minúscula y si contienen varias palabras, se unen y todas las palabras excepto la primera empiezan por mayúscula.
- **Inicialización**: Asignar un valor a la variable por primera vez (edad = 25).
- **Memoria**: Las variables se almacenan en la memoria.

```
public static void main(String[] args) {
        int edad;        // Declaración
        edad = 25;       // Inicialización
        String nombre = "Juan"; // Declaración e inicialización
simultánea
        System.out.println(nombre + " tiene " + edad + " años.");
    }
```

VARIABLES: DECLARACIÓN, INICIALIZACIÓN Y UTILIZACIÓN. ALMACENAMIENTO EN MEMORIA

- La **visibilidad** o ámbito de una variable es la parte del código de una aplicación donde la variable es accesible y puede ser utilizada. Las variables se declaran dentro de un bloque (por bloque se entiende el contenido entre las llaves {}) y son accesibles sólo dentro de ese bloque.
- Las variables declaradas en el bloque de la clase como n1 se consideran miembros de la clase (**variable global**), mientras que las variables n2 y suma pertenecen al método *main* y sólo pueden ser utilizados en el mismo. Las variables declaradas en el bloque de código de un método son variables que se crean cuando el bloque se declara, y se destruyen cuando finaliza la ejecución de dicho bloque. -> **Variable Local.**
- Las **variables globales** se inicializan **por defecto** (las numéricas con 0 y los caracteres con '\0', mientras que las variables locales no se inicializan por defecto.

VARIABLES: DECLARACIÓN, INICIALIZACIÓN Y UTILIZACIÓN. ALMACENAMIENTO EN MEMORIA

```
public class Ejemplo {
//variable miembro de la clase
static int n1=50;
    public static void main(String[] args) {
        int n2=30, suma=0; //variables locales
        suma=n1+n2;
        System.out.println("La suma es:"+suma);
    }
}
```

CONSTANTES

- **Constantes**: Son variables cuyo valor no cambia una vez que se asignan. Se declaran usando **final**. Por convenio, se declaran en **mayúsculas**.

```
public class EjemploConstantes {
    // Declaración de una constante
    public static final double PI = 3.1416;  // Constante (final no permite cambios)

    public static void main(String[] args) {
        double radio = 5.0;
        double circunferencia = 2 * PI * radio; // Uso de la constante
        System.out.println("La circunferencia es: " + circunferencia);
    }
}
```

IDENTIFICADORES

- Los **identificadores** son los nombres que les damos a variables, métodos, clases, etc. Deben comenzar con una letra, $ o _ y no pueden contener espacios. Después pueden combinar letras y números.

```
public class EjemploIdentificadores {
    // Identificadores válidos
    int edad;       // Variable
    String nombre;  // Variable
    final int MAX_EDAD = 100; // Constante (usualmente con mayúsculas)

    public void imprimirDatos() {  // Método con un identificador
        System.out.println(nombre + " tiene " + edad + " años.");
    }
}
```

LITERALES

- **Literales**: Son valores fijos que asignamos a las variables (ej. 42, 3.1416, "Hola!", false).

```
public static void main(String[] args) {
        // Literales
        int numero = 42;         // Literal entero
        double pi = 3.1416;      // Literal decimal
        char letra = 'A';        // Literal carácter
        String saludo = "Hola!";  // Literal cadena
        boolean esCierto = false; // Literal booleano
        System.out.println("Número: " + numero);
        System.out.println("Pi: " + pi);
        System.out.println("Letra: " + letra);
        System.out.println("Saludo: " + saludo);
        System.out.println("Es cierto: " + esCierto);
    }
```

EJERCICIOS

- Realiza los ejercicios del Boletín 1.

OPERADORES Y EXPRESIONES. PRECEDENCIA DE OPERADORES

- Los **operadores** son símbolos que indican cómo se van a manipular los datos (operandos).
- Se clasifican en:
  - Operadores aritméticos
  - Operadores relacionales
  - Operadores lógicos
  - Operadores de incremento y decremento
  - Operadores de asignación

OPERADORES Y EXPRESIONES. PRECEDENCIA DE OPERADORES

- **Operadores aritméticos**

Suma: +
Resta: -
Producto: *
División: /
Módulo o resto de una división de números enteros (no para float); %

- **Operadores relacionales**

Mayor: >	Mayor o igual: > =
Menor: <	Menor o igual: < =
Igual: = =	Distinto: ! =

OPERADORES Y EXPRESIONES. PRECEDENCIA DE OPERADORES

- **Operadores lógicos**: son necesarios para describir condiciones en las que aparecen conjunciones, disyunciones, exclusivas y negaciones:
  - Conjunciones **Y (&&)**
  - Disyunciones **O (||)**
  - O, **exclusiva ^** significa deriva en verdadero si ambos operandos son distintos: verdadero-falso o falso-verdadero, y produce falso sólo si ambos operandos son iguales: verdadero-verdadero o falso-falso.

Operador	Operación lógica	Ejemplo
Negación (!)	No lógica	`! (x >= y)`
O, exclusiva(^)	operando_1 ^ operando_2	`x < n ^ n > 9`
Y, lógica (&&)	operando_1 && operando_2	`m < n && i > j`
O, lógica ||	operando_1 || operando_2	`m = 5 || n != 10`

OPERADORES Y EXPRESIONES. PRECEDENCIA DE OPERADORES

- **Operadores lógicos**
- En Java, **los operandos que se encuentran a la izquierda de && y || se evalúan siempre en primer lugar**; si el valor de tales operandos determina de forma inequívoca el valor de la expresión, el de la derecha no se evalúa; por tanto, si el operando a la izquierda de && es falso o el de || es verdadero, el de la derecha no se evalúa; esta propiedad se denomina **evaluación en cortocircuito.**

- **Operadores | y &** : Se permite utilizar operadores | y & con operandos tipo boolean, los cuales corresponden con el operador or (||) y and (&&), respectivamente, con una diferencia importante: no evalúan en cortocircuito sino que **ambos operandos siempre se evalúan**.

OPERADORES Y EXPRESIONES. PRECEDENCIA DE OPERADORES

- **Operadores lógicos**
- **! not** produce falso si su operando es verdadero y viceversa:

Operando (a)	NOT a
Verdadero	Falso
Falso	Verdadero

- **Operadores de incremento y decremento**: son dos operaciones típicas para **aumentar o disminuir en una unidad el valor de una variable contador**. Son ++ y --.

OPERADORES Y EXPRESIONES. PRECEDENCIA DE OPERADORES

- **Prioridad y asociatividad**
- Si dos operadores se aplican en la misma expresión, el de mayor prioridad se aplica primero.
- Los del mismo grupo tienen igual prioridad y asociatividad.
- La **asociatividad izquierda-derecha** implica la aplicación del operador más a la izquierda primero; en la asociatividad **derecha-izquierda** se emplea primero el operador que se encuentra más a la derecha.
- Los **paréntesis tienen máxima prioridad**.

Prioridad	Operadores	Asociatividad
1	new (creación de objetos)	
2	. [] () ++ -- (prefijo)	I -D D - I
3	++ -- (postfijo)	I - D
4	~ ! - +	D - I
5	(type)	D -I
6	* / %	I - D
7	+ - (binarios)	I - D
8	<< >> >>>	I - D
9	< <= > >= instanceof	I - D
10	== !=	I - D
11	&	I - D
12	^	I - D
13	|	I - D
14	&&	I - D
15	||	I - D
16	?: (*expresión condicional*)	D - I
17	= *= /= %= += -= <<= >>= >>>= &= |= ^=	D - I
18	, (*operador coma*)	I - D

I - D: Izquierda – Derecha
D - I: Derecha – Izquierda

OPERADORES Y EXPRESIONES. PRECEDENCIA DE OPERADORES

- **Operadores de asignación**

=	A = B	Asignación.
*=	A *= B	Multiplicación y asignación. La operación A*=B equivale a A=A*B.
/=	A /= B	División y asignación. La operación A/=B equivale a A=A/B.
+=	A += B	Suma y asignación. La operación A+=B equivale a A=A+B.
-=	A -= B	Resta y asignación. La operación A-=B equivale a A=A-B.
%=	A %= B	Módulo y asignación. La operación A%=B equivale a A=A%B.

EJERCICIOS

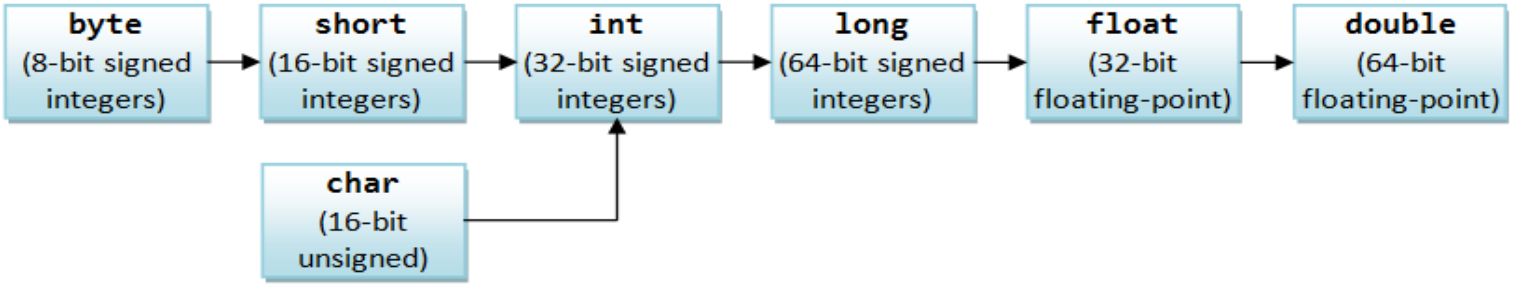
- Realiza los ejercicios del Boletín 2.

CONVERSIONES DE TIPO. IMPLÍCITAS Y EXPLICITAS (CASTING).

- Existen dos tipos de conversiones:
  - **Casting implícito**: se hacen de forma automática al asignar un tipo a otro, sin que hagamos nada.
  - **Casting explícito**: debemos controlar e indicar nosotros la conversión.

CONVERSIONES DE TIPO. IMPLÍCITAS Y EXPLICITAS (CASTING).

- **Casting implícito**: automáticamente podemos asignar un valor literal o una variable de un tipo de dato, a otra variable de un tipo distinto, siempre convierta de un tipo a otro tipo más a la derecha, de los siguientes:



Orders of Implicit Type-Casting for Primitives

- Esto quiere decir que no necesitamos escribir ningún tipo de código para pasar de un tipo de datos a otro.
- **Se da cuando "metemos" un valor pequeño en un contenedor (variable) más grande**.

CONVERSIONES DE TIPO. IMPLÍCITAS Y EXPLICITAS (CASTING).

```
//Ambos literales son literales de int, pero cumplen el rango de cada tipo
byte varPequena = 127;
short varGrande = 12365;
//Convierto de literal int a variable long
long enteroGrande = 24556;
//Convierto de tipo long a tipo float
float decimalSimple = enteroGrande;
//Convierto de float a double
double decimalDoble = decimalSimple;
//Convierto de literal float a double
decimalDouble = 3.14F;
```

CONVERSIONES DE TIPO. IMPLÍCITAS Y EXPLICITAS (CASTING).

- **Casting explícito**: Se dan cuando intento asignar un tipo de datos más grande a una variable más pequeña.
- El formato del casting es: (tipo)valorAConvertir;
- (Donde el tipo es el tipo al que se quiere convertir, y valorAConvertir, el valor desde el que se quiere convertir).

```
//una variable int ocupa 4 bytes, short 2 bytes
int entero = 13;
short enteroPequeno = (short)entero;
long numeroLargo = 13;
entero = (int)numeroLargo;
double decimal = 23423.234234;
entero = (int) decimal;
```

CONVERSIONES DE TIPO. IMPLÍCITAS Y EXPLICITAS (CASTING).

- **Casting caracter-entero:** en algunos lenguajes de programación existe una relación entre los caracteres y los número enteros.
- Los caracteres se codifican atendiendo al estandar ISO/IEC 8859-1. Los 128 primeros caracteres de esa tabla correponden con el estándar ASCII. En dichas tablas existe una correspondencia entre el nº de orden del caracter y la representación literal del carácter. Por lo tanto, hay una serie de **conversiones implícitas y explícitas que se permiten entre los tipos int y char:**

```
//Casting implícito
char letraA = 65; //valor ASCII -> 'A'
int valorA = letraA;
int valorB = 'B'; //entero ASCII -> 66
//Casting explícito
char letraB = (char)valorB; //convierto de int a char
```

CONVERSIONES DE TIPO. IMPLÍCITAS Y EXPLICITAS (CASTING).

- **Aspectos a tener en cuenta:**
- Cualquier operación aritmética entre dos tipos de datos inferiores a int produce como resultado un int:

```
byte enteroByte = 4;
short enteroShort = 6;
short resultado = enteroByte + enteroShort; //Error
//Soluciones
short resultado = (short)(enteroByte + enteroShort);
int resultado = enteroByte + enteroShort;
```

CONVERSIONES DE TIPO. IMPLÍCITAS Y EXPLICITAS (CASTING).

- **Aspectos a tener en cuenta:**
- Cualquier operación aritmética entre dos tipos de datos distintos, produce como **resultado un tipo de datos igual al tipo mayor**:

```
long enteroLargo = 12;
int entero = 234;
entero * enteroLargo; // resultado tipo long
double decimal = 23.234;
entero - decimal; // resultado tipo double
```

CONVERSIONES DE TIPO. IMPLÍCITAS Y EXPLICITAS (CASTING).

Aspectos a tener en cuenta:

Siempre existe riesgo de pérdida de datos al hacer un **casting de un tipo más grande hacia un tipo más pequeño**:

```
int entero = 263;
```

El contenido de la variable entero (4 bytes) es:

00000000-00000000-00000001-00000111

Al hacer un casting a tipo byte:

```
byte numero = (byte)entero;
```

La variable byte solo tiene 8 bits, por lo que solo se está guardando el último byte de la variable entero:

00000111 cuyo valor es 7.

Pérdida de precisión de tipos reales: Si intentamos **asignar un valor entero muy grande a este tipo de datos** (float o double), se realiza un casting implícito, pero hay riesgo de pérdida de datos.

EJERCICIOS

- Realiza los ejercicios del Boletín 3.

Bibliografía

- Luis Joyanes Aguilar, Ignacio Zahonero Martínez, *Programación en C,C++, Java y UML*. McGraw Hill.
- https://programacion.abrilcode.com/doku.php?id=bloque2:casting
- https://docs.oracle.com/javase/specs/jls/se8/html/index.html