UNIVERSIDAD DE CANTABRIA

Una comparación entre C, C++, Java y Ada

Santander, enero de 2009

J. Javier Gutiérrez
gutierjj@unican.es

Una comparación entre C, C++, Java y Ada

UNIVERSIDAD DE CANTABRIA

1. Introducción

UNIVERSIDAD DE CANTABRIA

El lenguaje C es uno de los lenguajes de programación más populares que existen hoy en día

Características más importantes:

- **lenguaje sencillo (aunque más difícil de aplicar)**
- **estructurado**
- **tipos definidos por el usuario**
- **no exige tipificación estricta**
- **permite compilación separada**
- **es de un nivel relativamente bajo**
- **compiladores baratos y eficientes**

Versiones del lenguaje C

UNIVERSIDAD DE CANTABRIA

Existen diversas versiones del lenguaje C:

- **C común: apareció en 1978 y estaba directamente ligado al sistema operativo UNIX**
- **C ANSI o C ISO: estandarización en 1988, y luego en 1995 y 1999; versión no ambigua y más portable, que añade**
  - **asignación de estructuras (registros)**
  - **tipos enumerados**
  - **prototipos de funciones, con especificación de sus argumentos**
  - **librerías estándares (funciones matemáticas, entrada/salida, etc.)**

En este seminario nos referiremos al ISO C 99

El lenguaje C++

UNIVERSIDAD DE CANTABRIA

Es un lenguaje derivado del C, pero mucho más complejo. Incluye:

- **definición de módulos (llamados clases)**
- **programación orientada al objeto**
- **tipificación más estricta**
- **tratamiento de excepciones**
- **plantillas (equivalentes a módulos genéricos)**

El C++ fue estandarizado en 1998.

El lenguaje Java

UNIVERSIDAD DE CANTABRIA

Es un lenguaje con una sintaxis similar a la del C++, pero unos conceptos de diseño totalmente diferentes:

- **portabilidad de clases mediante:**
  - **código intermedio estandarizado, e interpretado por una máquina virtual (portable, pero ineficiente)**
  - **las clases se cargan dinámicamente (y de sitios remotos)**
- **gestión automática de memoria, que aumenta la fiabilidad**
- **programación orientada a objetos**
- **tipificación estricta**
- **tratamiento de excepciones; recientemente, genéricos**

El Java pertenece a la empresa Sun y no está estandarizado

2. Estructura de un programa

UNIVERSIDAD DE CANTABRIA

Estructura de un bloque en Ada y en C/Java:

Ada	C/Java

declare
  <declaraciones>
begin
  <instrucciones>
exception
  <manejadores>
end;
	{
	  <declaraciones>
	
	  <instrucciones>
	}

2. Estructura de un programa (cont.)

UNIVERSIDAD DE CANTABRIA

Estructura de un programa:

Ada	C	Java

with modulo1;
procedure Nombre is
  <declaraciones>
begin
  <instrucciones>
end;
	#include <modulo1.h>
	main()
	{
	  <declaraciones>
	
	  <instrucciones>
	}
		import paquete.*;
		public class Nombre {
		 public static void
		 main (String[] args)
		 {
		  <declaraciones>
		
		  <instrucciones>
		 }
		}

Ejemplos

UNIVERSIDAD DE CANTABRIA

```
#include <stdio.h>
main()
{
   printf("hola\n"); // printf para escribir
}
```

```
with Ada.Text_IO;
procedure Hola is
begin
   Ada.Text_IO.Put_Line("hola"); -- put para escribir
end Hola;
```

Ejemplos (cont.)

UNIVERSIDAD DE CANTABRIA

```
public class Hola {
  public static void main(String[] args)
  {
     // println es una operación del objeto
     // out de la clase System
     System.out.println("hola");
  }
}
```


Algunas observaciones:

UNIVERSIDAD DE CANTABRIA

C/Java	Ada
se distingue entre mayúsculas y minúsculas	no hay distinción
las instrucciones acaban con "`;`", pero los bloques no	instrucciones y bloques acaban con "`;`"
En Java las funciones son bloques que siempre son métodos de clases	Las funciones y procedimientos son bloques que se pueden anidar
comentarios entre `/**/`, o empezando con `//` hasta fin de línea	los comentarios empiezan con "`--`" y continúan hasta el final de la línea
todas las variables deben declararse	idem.
los strings se encierran entre `""`	idem.
en C y Java el retorno de línea se pone en el string `"\n"`	retorno de línea con operaciones específicas (`New_Line`, `Put_Line`)

Importancia de la declaración de variables

UNIVERSIDAD DE CANTABRIA

La siguiente línea de programa en Fortran causó la pérdida de la nave Viking dirigida al planeta Venus:

```
DO 20 I=1.100
```

La línea correcta, que hubiera sido el equivalente de un lazo `for` era:

```
DO 20 I=1,100
```

En su lugar, el compilador interpretó que la instrucción era una asignación a la variable (no declarada) `DO20I`:

```
DO20I = 1.100
```

En C, Java y Ada todos los objetos deben ser declarados

3. Tipos de datos

UNIVERSIDAD DE CANTABRIA

Tipos predefinidos:

Ada	C	Java
Integer	signed char short int long	byte short int long
Boolean	(se usa int)	boolean
Character (8 bits) Wide_Character (16 bits)	se usa char (8 bits) wchar_t	char (16 bits)
Float	float double	float double
Duration	-	-
String	char[], char*	String

Declaraciones

UNIVERSIDAD DE CANTABRIA

C/Java

```
int lower;
char c,resp;// dos variables de tipo char
int i=0;    // variable inicializada
const float eps=1.0e-5;  // constante C
#define MAX 8            // constante textual (C)
final float eps=1.0e-5;  // constante Java
```

Ada

```
Lower : Integer;
C, Resp : Character; -- dos variables Character
I : Integer := 0;    -- variable inicializada
Eps : constant Float:=1.0e-5; -- Constante
```


Observaciones sobre las declaraciones

UNIVERSIDAD DE CANTABRIA

Puede observarse que, tanto en C y Java como en Ada:

- **las variables se pueden inicializar**
- **se pueden declarar como constantes**
- **se pueden declarar varias variables juntas**

En C y Java se pone primero el tipo, y luego la variable (al revés que en Ada)

Tipos discretos definidos por el usuario

UNIVERSIDAD DE CANTABRIA

Ni en C ni en Java se pueden crear más tipos enteros o reales

Tampoco se pueden definir tipos subrango

En ambos se pueden definir tipos enumerados

Suponer el siguiente tipo enumerado en Ada:

```
type Dimension is (Planox, Planoy, Planoz);
Fuerza, Linea : Dimension;
...
Fuerza:=Planox;
Linea :=Dimension'Succ(Fuerza);
```


Tipos discretos (cont.)

UNIVERSIDAD DE CANTABRIA

Su equivalente en C sería:

```
typedef enum {planox,planoy,planoz} dimension;
dimension fuerza, linea;

linea=planox;
fuerza=linea+1;
```

Observaciones:

- **en C (y en Java también) el operador de asignación es "`=`"**
- **los valores enumerados se tratan como enteros, con valores 0, 1, 2,...**
- **`fuerza` toma el valor `planoy` en el ejemplo**

Tipos discretos (cont.)

UNIVERSIDAD DE CANTABRIA

El equivalente al tipo `Dimension` en Java sería esta clase:

```
public enum Dimension {planox, planoy, planoz};

Dimension fuerza, linea;

linea=Dimension.planox;
```

Caracteres

UNIVERSIDAD DE CANTABRIA

Los caracteres en Ada, C, y Java se representan encerrados entre apóstrofes:

```
char c1,c2;
c1='a';
```

Los caracteres de control tiene una representación especial:

Carácter	C/Java	Ada
new-line	'\n'	ASCII.LF
carácter nulo	'\0'	ASCII.NUL
bell (pitido)	'\a'	ASCII.BEL
apóstrofe	'\''	'''

Arrays

UNIVERSIDAD DE CANTABRIA

Los arrays de C son muy diferentes de los de Ada o Java:

- **no tienen un tamaño especificado**
- **el usuario es responsable de no exceder su tamaño real**
- **se pueden manipular mediante punteros**

Los arrays de Ada y Java son mucho más seguros

Los arrays de Java se crean siempre en memoria dinámica

- **la variable array es una referencia que apunta al array creado dinámicamente**
- **la asignación copia la referencia, no el array**

En Java no hay tipos array, pero sí objetos (o variables) array

Arrays

UNIVERSIDAD DE CANTABRIA

Las siguientes declaraciones de tipos en Ada:

```
Max : constant Integer := 8; -- cte. en Ada
type Vector is array (0..Max-1) of Float;
type Contactos is array (0..Max-1,0..Max-1)
     of Boolean;
```

Tienen el siguiente equivalente en C:

```
#define MAX 8 // constante en C
typedef float vector[MAX]; // de 0 a MAX-1
typedef short int contactos[MAX][MAX];
```

Observar los corchetes (en vez de paréntesis), y que el índice empieza siempre en cero (va de `0` a `MAX-1`)

En Java no hay *tipos* array (pero sí *objetos* array)

Arrays (cont.)

UNIVERSIDAD DE CANTABRIA

Las siguientes declaraciones de datos en Ada:

```
V : Vector:=(Vector'Range => 0.0);
W : Vector:=(0..Max-1 => 0.0);
C1,C2 : Contactos;
A : array (0..3) of Integer;
```

Tienen el siguiente equivalente en C:

```
vector v={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
vector w={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
contactos c1,c2;
int a[4];
```

Literal de array (sólo en inicialización)

Y en Java:

```
float v[]={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
boolean contactos[][]=new boolean[8][8];
int a[]=new int[4];
```


Uso de Arrays

UNIVERSIDAD DE CANTABRIA

Las siguientes instrucciones en Ada:

```
V(3):=10.0;
C1(0,3):=True;
A:=(9,3,0,4);
```

Tienen el siguiente equivalente en C y Java:

```
v[3]=10.0;
c1[0][3]=1; // en C
c1[0][3]=true; // en Java
a[0]=9; a[1]=3; a[2]=0; a[3]=4;
```

No hay asignación de arrays en C

```
c1=c2; // no se puede
```


Strings en C

UNIVERSIDAD DE CANTABRIA

Los strings en C son arrays de caracteres, con un número de elementos variable.

El string debe terminar siempre con un carácter nulo (código ASCII cero), que sirve para saber donde acaba el string.

Los strings constantes se representan encerrados entre comillas, como en Ada:

```
char linea[80];
char otra[]="hola esto es un string";
char linea2[]="hola esto es una línea\n";
              /* incluye un salto de linea */
```

Los strings constantes ya incluyen el carácter nulo al final.

Strings en Java

UNIVERSIDAD DE CANTABRIA

Los strings en Java son objetos de una clase llamada `String`.

Los strings constantes o literales se representan encerrados entre comillas, como en Ada y C:

```
String linea;
String otra="hola esto es un string";
```

Para los strings existe la operación de concatenación:

```
linea = otra+" largo";
```

Se puede concatenar un string con una variable: ésta se convierte a texto

```
String resultado = "Resultado="+i;
```


Registros o estructuras

UNIVERSIDAD DE CANTABRIA

Los registros se llaman estructuras en C, y cada campo se llama un miembro:

Por ejemplo el siguiente registro Ada:

```
type Fecha is record
     Dia : Integer range 1..31;
     Mes : Integer range 1..12;
     Año : Integer range 1900..2100;
end record;
F1,F2 : Fecha;
...
F1:=(12,4,1996);
F2.Mes:=5;
```


Registros o estructuras (cont.)

UNIVERSIDAD DE CANTABRIA

Tendría el siguiente equivalente en C:

```
struct fecha {
     int dia;
     int mes;
     int agno;
};
struct fecha f1,f2;
...
f1={12,4,1996};
f2.mes=5;
```

Observar que los campos (o miembros) se expresan igual en C y Ada

Registros o estructuras (cont.)

UNIVERSIDAD DE CANTABRIA

También se podía haber escrito:

```
typedef struct {
     int dia;
     int mes;
     int agno;
} fecha;
fecha f1,f2;
```

Observar que la diferencia es que ahora el tipo se llama `fecha` y no `struct fecha`

En Java no hay registros. Se utilizan las clases en su lugar.

Punteros y estructuras de datos dinámicas

UNIVERSIDAD DE CANTABRIA

En Ada se pueden declarar estructuras de datos dinámicas con punteros:

```
type Nudo;
type Punt_Nudo is access Nudo;
type Nudo is record
     Valor : Integer;
     Proximo : Punt_Nudo;
end record;
Primero : Punt_Nudo;
...
Primero:=new Nudo;
Primero.Valor:=3;
```


Punteros (cont.)

UNIVERSIDAD DE CANTABRIA

En C se utiliza el símbolo "`*`" para denotar el objeto al que apunta un puntero; la declaración en C sería:

```
struct nudo {
     int valor;
     struct nudo *proximo;
};
struct nudo *primero;

primero=malloc (sizeof (struct nudo));
primero->valor=3;
primero->proximo=0;
```

Punteros (cont.)

UNIVERSIDAD DE CANTABRIA

Observar que:

- **el símbolo "`->`" se usa para indicar el miembro de una estructura a la que apunta el puntero**
- **el puntero no se inicializa por defecto a 0 (que es el equivalente a `null`)**
- **`malloc`() equivale a `new`, pero es preciso pasarle el número de bytes que se reservan para la nueva variable**
- **el número de bytes que ocupa un tipo de datos se obtiene con `sizeof`()**
- **el operador unario "`&`" se puede usar para obtener un puntero que apunte a una variable (p.e. `&var_i` es un puntero que apunta a `var_i`)**

Punteros en Java

UNIVERSIDAD DE CANTABRIA

En Java todos los objetos (incluidos strings y arrays):

- **se crean dinámicamente**

```
// ejemplos
Display pantalla = new Display("Titulo");
float vector[] = new float[10];
```

- **se destruyen de manera automática**
- **las declaraciones representan referencias o punteros a los objetos**
- **inicialmente valen "`null`"**
- **la asignación copia la referencia; no copia el objeto**

4. Operadores y expresiones

UNIVERSIDAD DE CANTABRIA

Tipo de Operador	C/Java	Ada	Función
Aritméticos	+,- *, /, %	+,- *, /, rem mod abs **	Suma, Resta Multiplicación, División Resto Módulo Valor absoluto Exponenciación
Relacionales	==, != >, >= <, <=	=, /= >, >= <, <=	Igual a, Distinto de Mayor, Mayor o Igual Menor, Menor o Igual
Lógicos	&&,||, !	and, or, not xor	And, Or, Not Xor

Operadores y expresiones (cont.)

UNIVERSIDAD DE CANTABRIA

Tipo de Operador	C/Java	Ada	Función
Incremento y decremento	++x, --x x++, x--		x=x+(-)1 y valor=x+(-)1 valor=x y x=x+(-)1
Manejo de bits en números enteros	&, |, ^ <<, >>	and, or, xor	And, Or, Or exclusivo Desplazamiento Izq., Der.
Conversión de tipo	(tipo) expr.	tipo(expr.)	Convierte la expresión al tipo indicado

Operadores y expresiones (cont.)

UNIVERSIDAD DE CANTABRIA

Tipo de Operador	C/Java	Ada	Función
Asignación Nota: no hay asignación para arrays ni strings en C	= += -= *= /= %= <<= >>= &= |= ~=	La asignación no es un operador	Asigna el valor y lo retorna izqdo=izqdo+drcho izqdo=izqdo-drcho izqdo=izqdo*drcho izqdo=izqdo/drcho izqdo=izqdo%drcho izqdo=izqdo<<drcho izqdo=izqdo>>drcho izqdo=izqdo&drcho izqdo=izqdo|drcho izqdo=izqdo~drcho

Operadores y expresiones (cont.)

UNIVERSIDAD DE CANTABRIA

Ni en C ni en Java se pueden definir los operadores por el usuario como en Ada.

Sin embargo en C++ si se puede

¡Cuidado con el operador de comparación!

```
i==2 // expresión lógica
i=2  // asignación
```

¡Cuidado con las asignaciones!

```
i:=2;   -- Ada
i=2;    // C
```

5. Entrada/salida simple

UNIVERSIDAD DE CANTABRIA

En Ada:

- **Paquete `Ada.Text_IO`:**

```
Put (Character);     Get (Character);
Put (String);        Get (String);
Put_Line (String);   Get_Line (String, Num);
New_Line;            Skip_Line;
```

- **Paquete Ada.Integer_Text_IO:**

```
Put (Integer);       Get (Integer);
Put (Integer, Ancho);
```

- **Paquete Ada.Float_Text_IO:**

```
Put (Float);         Get (Float);
Put (Float, Antes, Despues, Exp);
```


Entrada/salida simple (cont.)

UNIVERSIDAD DE CANTABRIA

En C, módulo `<stdio.h>`

```
int printf(string_formato[, expr, ...]);
    // escribe y retorna el número de caracteres
    // escritos
int scanf (string_formato,&var[,&var...]);
    // lee y retorna el número de datos leídos
    // correctamente
char *fgets(char *s, int size, FILE* stream);
    // lee una línea y la mete en s, hasta el
    // final de la línea (inclusive) o hasta un máximo
    // de size-1 caracteres; añade un nulo al final
    // stream es un fichero: para el teclado usar stdin
    // retorna NULL si hay error
```


Entrada/salida simple (cont.)

UNIVERSIDAD DE CANTABRIA

Strings de formato más habituales:

```
%d, %i   enteros
%c       char
%s       string (una sola palabra al leer)
%f       leer float; escribir float y double, coma fija
%e       leer float; escribir float y double, exponencial
%lf      leer double en coma fija
%le      leer double en notación exponencial
```

Puede añadirse después del "`%`" la especificación de anchura y precisión (ancho.precisión); p.e.:

```
%12.4f
```

Ejemplo de entrada/salida simple (Ada)

UNIVERSIDAD DE CANTABRIA

```
with Ada.Text_IO, Ada.Integer_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO;
procedure Nota_Media is
     Nota1, Nota2, Nota3, Media : Integer range 0..10;
begin
     Put ("Nota primer trimestre: ");
     Get (Nota1); Skip_Line;
     Put ("Nota segundo trimestre: ");
     Get (Nota2); Skip_Line;
     Put ("Nota segundo trimestre: ");
     Get (Nota3); Skip_Line;
     Media:=(nota1+Nota2+Nota3)/3;
     Put ("La nota media es : ");
     Put (Media); New_Line;
end Nota_Media;
```


Ejemplo de entrada/salida simple (C)

UNIVERSIDAD DE CANTABRIA

```
#include <stdio.h>
int main ()
{
     int nota1, nota2, nota3, media;
     // falta la detección de errores
     printf("Nota primer trimestre: ");
     scanf ("%d", &nota1);
     printf("Nota segundo trimestre: ");
     scanf ("%d",&nota2);
     printf("Nota tercer trimestre: ");
     scanf ("%d",&nota3);
     media=(nota1+nota2+nota3)/3;
     printf ("La nota media es : %d\n",media);
     return 0;
}
```


Entrada/Salida en Java

UNIVERSIDAD DE CANTABRIA

La salida de texto en Java es sencilla a través de la operación `System.out.println()`

La entrada de texto en Java es muy compleja

Lo más cómodo es hacer la entrada/salida con una interfaz gráfica, con ventanas.

6. Instrucciones de control

UNIVERSIDAD DE CANTABRIA

Instrucción condicional

Ada	C	Java

`if exp_booleana then`
`  instrucciones;`
`end if;`
	`if (exp_entera) {`
	`  instrucciones;`
	`}`
		`if (exp_booleana) {`
		`  instrucciones;`
		`}`

`if exp_booleana then`
`  instrucciones;`
`else`
`  instrucciones;`
`end if;`
	`if (exp_entera) {`
	`  instrucciones;`
	`} else {`
	`  instrucciones;`
	`}`
		`if (exp_booleana) {`
		`  instrucciones;`
		`} else {`
		`  instrucciones;`
		`}`

- En C, un resultado `0` en la `exp_entera` equivale a `False`
- En C, un resultado distinto de `0` equivale a `True`

Un fallo frecuente (C)

UNIVERSIDAD DE CANTABRIA

Estas instrucciones ponen "`valor de i=4`" en pantalla

```
int i=2;

if (i=4) {
   printf("valor de i=%d",i);
} else {
   printf("no es 4");
}
```

En Java y Ada, el compilador hubiera detectado el fallo

Ejemplo de entrada/salida simple (C, versión 2)

UNIVERSIDAD DE CANTABRIA

```
#include <stdio.h>
main ()  {
     int nota1, nota2, nota3, media;

     printf("Nota primer trimestre: ");
     if (scanf ("%d",&nota1)==0 || nota1<0 || nota1>10) {
          printf("Error"); return;
     }
     printf("Nota segundo trimestre: ");
     if (scanf ("%d",&nota2)==0 || nota2<0 || nota2>10) {
          printf("Error"); return;
     }
     printf("Nota tercer trimestre: ");
     if (scanf ("%d",&nota3)==0 || nota3<0 || nota3>10) {
          printf("Error"); return;
     }
     media=(nota1+nota2+nota3)/3;
     printf ("La nota media es : %d\n",media);
}
```

Instrucción condicional múltiple

UNIVERSIDAD DE CANTABRIA

Ada	C/Java

`case exp_discreta is`
`  when valor1 =>`
`  instrucciones;`
`  when valor2 | valor3 =>`
`  instrucciones;`
`  when others =>`
`  instrucciones;`
`end case;`
	`switch (exp_entera) {`
	`  case valor1 :`
	`  instrucciones;`
	`  break;`
	`  case valor2 :`
	`  case valor3 :`
	`  instrucciones;`
	`  break;`
	`  default :`
	`  instrucciones;`
	`  break;`
	`}`

Cuidado: No olvidarse el "`break`"

Lazos

UNIVERSIDAD DE CANTABRIA

Ada	C	Java

`while exp_booleana`
`loop`
`  instrucciones;`
`end loop;`
	`while (exp_entera) {`
	`  instrucciones;`
	`}`
		`while (exp_booleana) {`
		`  instrucciones;`
		`}`

`loop`
`  instrucciones;`
`end loop;`
	`while (1) {`
	`  instrucciones;`
	`}`
		`while (true) {`
		`  instrucciones;`
		`}`

`loop`
`  instrucciones;`
`  exit when`
`  exp_booleana;`
`end loop;`
	`do {`
	`  instrucciones;`
	`} while (exp_entera);`
		`do {`
		`  instrucciones;`
		`} while(exp_booleana);`

Lazos (cont.)

UNIVERSIDAD DE CANTABRIA

Ada	C	Java

`for i in v_ini..v_fin`
`loop`
`  instrucciones;`
`end loop;`
	`for (i=v_ini;`
	`  i<=v_fin; i++)`
	`{`
	`  instrucciones;`
	`}`
		`for (int i=v_ini;`
		`  i<=v_fin; i++)`
		`{`
		`  instrucciones;`
		`}`

Observar que:

- **En C, la variable de control del lazo debe aparecer en las declaraciones; en Ada no; en Java y C99 es opcional**
- **La instrucción C/Java es más flexible: las tres expresiones entre paréntesis son arbitrarias**
- **Puede usarse `break` para salirse de un lazo, de igual modo que la instrucción `exit` incondicional de Ada**

7. Subprogramas y paso de parámetros

UNIVERSIDAD DE CANTABRIA

En C todos los subprogramas se llaman funciones; en Java se llaman métodos, y deben estar incluidos en una clase

Su estructura tiene la forma siguiente:

```
tipo_retornado nombre_funcion (argumentos)
{
     declaraciones locales;
     instrucciones;
}
```

La función retorna un valor al ejecutar la instrucción `return`

Si se desea un procedimiento, se escribe la cabecera como:

```
void nombre_funcion (argumentos)
```


Parámetros

UNIVERSIDAD DE CANTABRIA

Los parámetros C se separan con comas y se especifican así:

```
tipo nombre_parámetro
```

En Ada los parámetros pueden ser de entrada, salida, o entrada/salida

- **el compilador elige si el parámetro se pasa por valor o por referencia, para obtener la máxima eficiencia**

En C los parámetros son sólo de entrada y por valor (se pasa una copia del parámetro a la función)

- **cuando se desea que el parámetro sea de salida o de entrada/salida es necesario pasar un puntero**
- **también se usa un puntero cuando el parámetro es grande**

Parámetros (cont.)

UNIVERSIDAD DE CANTABRIA

En Java los parámetros son también de entrada y por valor, como en C

Puesto que todos los objetos se representan mediante referencias:

- **el paso de objetos es efectivamente un paso por referencia**
- **es posible que la función devuelva valores en el objeto cuya referencia se le pasa**
- **si se declara un argumento como `final`, el método no puede cambiar su valor**

Parámetros (cont.)

UNIVERSIDAD DE CANTABRIA

Por ejemplo la siguiente función es incorrecta en C y Java:

```
void intercambia (int x, int y) // incorrecta
{
     int temp;
     temp=x;
     x = y;
     y = temp;
}
```

La llamada a esta función habría sido:

```
int a,b;
intercambia (a,b);
```

Pero no habría cambiado nada

Parámetros (cont.)

UNIVERSIDAD DE CANTABRIA

La función correcta en C sería:

```
void intercambia (int *x, int *y) // correcta
{
     int temp;
     temp=*x;
     *x = *y;
     *y = temp;
}
```

Y su llamada es:

```
int a,b;
intercambia (&a,&b);
```


Parámetros (cont.)

UNIVERSIDAD DE CANTABRIA

Y la función en Java requiere el uso de objetos (en este caso de una clase `Entero`, que contenga un campo `valor`):

```
void intercambia (Entero x, Entero y)
{
     int temp;
     temp=x.valor;
     x.valor = y.valor;
     y.valor = temp;
}
```

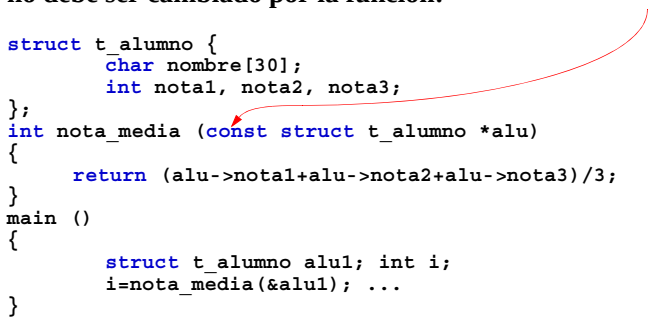
Y su llamada es:

```
Entero a=new Entero(val1),b=new Entero(val2);
intercambia (a,b);
```

Parámetros (cont.)

UNIVERSIDAD DE CANTABRIA

En C, al pasar un parámetro "in" mediante un puntero (para evitar la pérdida de eficiencia de la copia), puede indicarse que el parámetro no debe ser cambiado por la función:



```
struct t_alumno {
        char nombre[30];
        int nota1, nota2, nota3;
};
int nota_media (const struct t_alumno *alu)
{
     return (alu->nota1+alu->nota2+alu->nota3)/3;
}
main ()
{
        struct t_alumno alu1; int i;
        i=nota_media(&alu1); ...
}
```


8. Modularidad y ocultamiento de información

UNIVERSIDAD DE CANTABRIA

Las funciones C pueden compilarse separadamente

Pueden ponerse una o varias funciones (y declaraciones de datos) en un mismo fichero. Se le suele dar la terminación "`.c`"

Puede utilizarse un método similar al de los paquetes en Ada, para crear módulos de programa:

- **la especificación, compuesta por declaraciones y cabeceras de función se pone en un fichero de "cabeceras" (acabado en "`.h`")**
- **el cuerpo se pone en otro fichero acabado en "`.c`"**
- **para usarlo se pone `#include "nombre_fichero.h"`**

Ficheros de cabeceras

UNIVERSIDAD DE CANTABRIA

El método no es tan seguro como los paquetes de Ada, ya que el compilador no comprueba la correspondencia entre la especificación y el cuerpo

En el ejemplo que se muestra a continuación hay 2 módulos de programa:

- **principal: `main.c`**
- **stacks: `stacks.h` y `stacks.c`**

Ejemplo de módulos en C

UNIVERSIDAD DE CANTABRIA

stacks.h

```
typedef struct {
     int elem[30];
     int top;
} stack;
void push(int i,
          stack *s);
int pop (stack *s);...
```

main.c

```
#include "stacks.h"
#include <math.h>
main() {
   stack s;

   push(1,&s);
   ...
}
```

stacks.c

```
#include "stacks.h"
void push(int i,
          stack *s)
{  ...
}
int pop (stack *s)
{  ...
}...
```


Módulos en C++ y Java

UNIVERSIDAD DE CANTABRIA

C++ y Java incluyen un concepto de módulo denominado clase, que permite construir tipos de datos abstractos

La clase permite la comprobación del módulo por parte del compilador, así como el ocultamiento de información, y la abstracción de datos

Permite también conceptos de programación orientada al objeto tales como herencia y polimorfismo

Una clase es funcionalmente equivalente a un tipo etiquetado en Ada 95, junto a sus operaciones primitivas. Un objeto de una clase equivale a una variable del tipo etiquetado

Módulos en C++

UNIVERSIDAD DE CANTABRIA

Especificación de una cola de números enteros

```
class cola_t {
     // parte privada
     int contenido[30];
     int principio,fin;
public: // a partir de aquí la parte visible
     cola_t();     // constructor
     ~cola_t();    // destructor
     void inserta(int e);
     int extrae();
};
```

Módulos en Java

UNIVERSIDAD DE CANTABRIA

La principal utilidad para especificar módulos en Java es la clase, muy similar a la de C++

Las clases Java no tienen una especificación aparte. En su lugar, se utilizan herramientas automáticas que extraen del código la documentación, incluyendo las interfaces

Una clase Java se guarda en un fichero y puede contener dentro otras clases internas

Se pueden agrupar varias clases en un paquete; se representa mediante un directorio

Existe también el concepto de una interfaz: debe ser cumplida por las clases que la implementen.

Módulos en Java (cont.)

UNIVERSIDAD DE CANTABRIA

```
public class Notas
{
    private int nota1, nota2, nota3;

    /** Pone los valores de las tres notas */
    public void ponNotas (int n1, int n2, int n3) {
        nota1=n1;
        nota2=n2;
        nota3=n3;
    }

    /** Calcula la media entera */
    public int media() {
        return (nota1+nota2+nota3)/3;
    }
}
```


9. Excepciones

UNIVERSIDAD DE CANTABRIA

El C++ y el Java presentan un mecanismo de excepciones similar al de Ada aunque más potente

Las excepciones son clases (Java: derivadas de `Exception`) que se pueden elevar o "lanzar" ("`throw`")

Las ventajas sobre el mecanismo de Ada son:

- **cada operación debe declarar las excepciones que puede lanzar**
- **al lanzar una excepción se le pueden pasar parámetros**
- **en Java, el compilador comprueba que no se olvidan manejadores de excepción**

La desventaja es una mayor complejidad

Ejemplo de excepciones (C++)

UNIVERSIDAD DE CANTABRIA

```
// clase que se usará para manejar excepciones
class error_de_rango {
   public:
      float valor_erroneo;
      error_de_rango(float v) { // constructor
         valor_erroneo=v;
      }
}
```


Ejemplo de excepciones C++ (cont.)

UNIVERSIDAD DE CANTABRIA

```
// clase con operaciones que lanzan excepciones
class termometro {
      float temperatura;
   public:
      termometro (float inicial)
         throw error_de_rango;
      void pon_valor (float valor)
         throw error_de_rango;
      float lee_valor();
};
```


Ejemplo de excepciones C++ (cont.)

UNIVERSIDAD DE CANTABRIA

```
// operación que arroja una excepción
termometro::termometro(float inicial)
{
     if (inicial>100 || inicial<0) {
          throw error_de_rango(inicial);
     } else {
          temperatura=inicial;
     }
};
```

Ejemplo de excepciones C++ (cont.)

UNIVERSIDAD DE CANTABRIA

```
// manejo de excepciones
...
try {
     termometro.pon_valor(t);
     ...
} catch (error_de_rango error) {
     printf("Valor incorrecto: %f",
               error.valor_erroneo);
     <otras instrucciones de manejo>;
} catch (...) {
     <instrucciones de manejo de otras excep.>;
}
```


10. Abstracción de tipos y reusabilidad

UNIVERSIDAD DE CANTABRIA

En C++ existe el concepto de plantilla ("template"), equivalente a los módulos genéricos en Ada

```
template <class T> class Vector { //vector de Ts
     T* v;
     int max
public:
     Vector(int size) // constructor
     {
          v=new T[max=size];     //crea array de Ts
     }
     int size() {return max;} //retorna tamaño
     Vector operator+(Vector V1, Vector V2)
     {...;}
};
```


11. Llamada a funciones C desde Ada

UNIVERSIDAD DE CANTABRIA

Es posible invocar funciones C desde programas Ada, y viceversa. Para llamar a una función C:

- **Se debe usar el `pragma Convention` para los tipos de datos que se vayan a pasar como parámetros. El pragma se pone después de la declaración del tipo:**

  ```
  pragma Convention (C,tipo-de-datos);
  ```

- **Se debe usar el `pragma Import` para las funciones C que se deseen usar. El pragma se pone después de la declaración de la cabecera del subprograma:**

  ```
  pragma Import(C,nombre-subprograma, string);
  ```

- **El paquete `Interfaces.C` contiene tipos C y funciones útiles**

Llamada a funciones C (cont.)

UNIVERSIDAD DE CANTABRIA

Ejemplo: uso de la función C `system()`, que permite ejecutar una orden del sistema operativo.

El prototipo de la función C es:

```
int system (const char s*)
```

Esta función pasa el string indicado por `s` al sistema operativo, como una orden a ejecutar.

Ejemplo: uso de la función system()

```
with Ada.Text_IO,Interfaces.C;
```

UNIVERSIDAD DE CANTABRIA

```
use Ada.Text_IO;
procedure Test_System is
   Orden : String(1..50);
   N_Orden : Natural;
   Codigo_Error : Integer;

   function System (Str : Interfaces.C.Char_Array)
            return Interfaces.C.Int;
   pragma Import(C,System,"system");
begin
   Put("Introduce orden : ");
   Get_Line(Orden,N_Orden);
   Codigo_Error:=Integer
      (System(Interfaces.C.To_C(Orden(1..N_Orden))));
   New_Line;
   Put_Line("Codigo de error : "&Integer'Image(Codigo_Error));
end Test_System;
```


Llamada a funciones Java desde Ada

UNIVERSIDAD DE CANTABRIA

Hay dos aproximaciones no estándares, que proporcionan una interfaz Ada para clases Java mediante herramientas automáticas:

- **Compiladores de Ada a código intermedio Java.**
  - **por ejemplo: jgnat**
- **Compiladores de Ada a código nativo, con una herramienta que**
  - **permite llamar a clases Java**
  - **estas clases se ejecutan en una máquina virtual**
  - **la cual se ejecuta en concurrencia con la aplicación**

12. Conclusiones: ¿por qué usar Ada o Java?

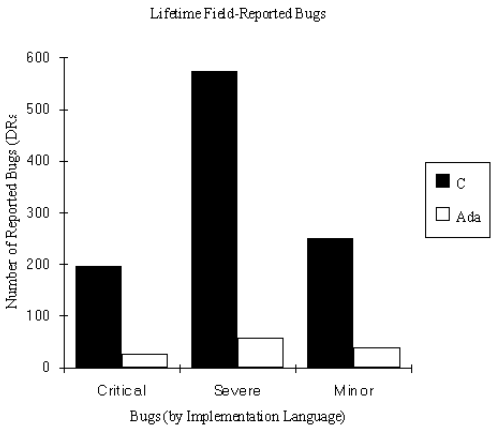
UNIVERSIDAD DE CANTABRIA

Comparación de costes en C y Ada en Verdix:

	C	ADA
Líneas totales	1.925.523	1.883.751
Líneas de código (SLOC)	1.508.695	1.272.771
Correcciones	13890	5841
Correcciones/KSLOC	9.21	4.59
Coste	$15.873M	$8.446M
Coste/SLOC	$10.52	$6.62
Defectos	1020	122
Defectos/KSLOC	0.676	0.096

Comparación de defectos

UNIVERSIDAD DE CANTABRIA



Complejidad del lenguaje C

UNIVERSIDAD DE CANTABRIA

El lenguaje C es potente y sencillo, a costa de complicar la labor de programación.

La siguiente línea compila sin errores en C:

```
z+=!x||(w%=++h==x&&q^~q)*3;for(x=k(i<
p->y);x!=(g)w+(--q);z|=!(x&&w%3)/**/){k;}
```

¡Recordar que un programa se escribe una vez y se lee muchas veces!

El código equivalente en Ada ocuparía varias líneas, pero:

- **Sería mucho más fácil de entender.**
- **Tendría más posibilidades de ser correcto, ya que el compilador detecta muchos errores.**

Conclusión

UNIVERSIDAD DE CANTABRIA

En C hay muchos errores que el compilador no detecta, lo que hace que los programas sean poco fiables

La programación de sistemas grandes es muy difícil, dada la ausencia de módulos en el programa

El lenguaje C++ corrige parte de estos problemas

- **detecta más errores (tipificación más estricta), introduce módulos y es más expresivo (excepciones, plantillas)**

Sin embargo, no soporta concurrencia, ni tiempo real, y los programas escritos en C++ son menos fiables que en Ada o en Java.

Bibliografía

UNIVERSIDAD DE CANTABRIA

[1] **Brian W. Kernighan, and Dennis Ritchie, "El lenguaje de programación C", 2ª edición, Prentice Hall, 1991**

[2] **Bjarne Stroustrup, "The C++ programming language", 2ª edición, Addison Wesley, 1991**

[3] **A. Burns and A.J. Wellings, "Real-time systems and programming languages", 2ª edición, 1996**

[4] **K. Arnold y J. Gosling, "The Java Programming Language, Second Edition", Addison Wesley, 1999,**

Direcciones Interesantes

UNIVERSIDAD DE CANTABRIA

Páginas con recursos Ada:

- `http://info.acm.org/sigada/`
- `http://www.adapower.com/`

GtkAda: (Visual Ada)

- `http://libre.act-europe.fr/GtkAda/`

La asociación Ada-Spain

- `http://www.adaspain.org/`

Ada-Europe Home Page

- `http://www.ada-europe.org/`

Distribución de gnat

- `http://libre.act-europe.fr/GNAT/`

Direcciones Interesantes

UNIVERSIDAD DE CANTABRIA

Distribución de java development kit

- `http://java.sun.com/j2se/`

Entorno de programación Java

- `http://www.bluej.org/`

Entorno de programación visual Java

- `http://www.borland.com/jbuilder/foundation/`

UNIVERSIDAD DE CANTABRIA