Algoritmos y Estructuras de Datos
2º de Ingeniería Informática, Curso 2008/2009

SEMINARIO "C para programadores java"
(este seminario forma parte de las actividades del *proyecto piloto*)

Sesión 1

Contenidos:

1. Características generales de C
2. Edición y compilación de un programa C
3. Variables y tipos de datos básicos
4. Operadores
5. Sentencias de control de flujo

Ejercicios

---

1. Características generales de C

- C es un lenguaje:
  - **IMPERATIVO** (vs. orientado a objetos).
  - estructurado de nivel medio-alto:
    - no es bajo nivel, como ensamblador
    - pero es débilmente tipado, hace uso intensivo de punteros, tiene goto…
  - de tamaño reducido
  - pero con una gran potencia y expresividad
  - se asume que "el programador sabe lo que hace" ⇒ errores, más difíciles de detectar; por ejemplo, no hay excepciones[1]

- Pero entonces, **¿por qué aprender C cuando ya sé java?** ¿Porque me obligan?
  - C es al mismo tiempo de nivel alto y bajo, polivalente.
  - Mejor control de los mecanismos de bajo nivel.
  - Mejor rendimiento que java (interpretado vs. compilado)
  - Java oculta muchos detalles necesarios para escribir código del S.O.

[1] ¿Conocéis las excepciones?

- **Tipos de datos:**
  - **elementales**:
    - enteros (`int`, `long`, `short`)
    - caracteres (`char`)
    - reales (`float`, `double`)
    - enumerados (`enum`)
    - no hay booleanos, pero sí en C++
  - **compuestos (ojo**, no son clases**)**:
    - arrays
    - registros (`struct`) y uniones (`union`).
    - No hay colecciones, clases, string ni envolventes[2], pero sí en C++

- **Funciones y procedimientos:**
  - No hay distinción entre unas y otros.
  - Tienen la forma:

```
<tipo_devuelto> <nombre_operacion> (<parámetros>)
{
      <codigo>
}
```

  - Si no devuelve nada o no hay parámetros, se pone `void`.
  - No se pueden anidar.

- **Sentencias de control de flujo**:
  - Iteración (`for`, `while`, `do-while`)
  - Condicionales (`if`, `if-else`)
  - Selección de casos (`switch`)

- **Expresiones y sentencias:**
  - Cualquier expresión en C puede ser una sentencia.
  - La asignación es una expresión, se expresa mediante el símbolo "=" (para la comparación usar "=="). Por ejemplo "`a= 4+(b=0);`".
  - Las sentencias acaban en "`;`".
  - Sentencia vacía, ";", es una sentencia válida.
  - Bloque:
    - secuencia de sentencias (0 o más), entre llaves.
    - es una sentencia.

- **Declaración de variables**:
  - Obligatoria.
  - Forma: `<tipo_var> <nombre_var> [, <nombre_var>];`
  - Las variables pueden ser **globales** (declaradas fuera de una función) o **locales** (declaradas al principio del código de un procedimiento, nunca intercaladas con el código, aunque en C++ sí es posible).
  - Ejemplos:

```
int a,b,c;
int a=0,b,c=1;          (inicialización en la declaración)
```

  - **OJO**: no hay inicialización a 0 automática de variables (en java sí).

REGLA DE ORO: siempre inicializar "todo" antes de usarlo.

[2] ¿Sabéis lo que son?

Escribir por pantalla: **`printf()`**

- No forma parte del lenguaje, hay que "incluirlo": #include <stdio.h>

```
printf ("Hola!");                    /* Esto es un comentario */
printf("%d", expresión_entero);     // Otro comentario
printf("%c", expresión_caracter);
printf("%f", expresión_real);
printf("Un entero: %d y un real: %f", entero, real);
printf("Dos caracteres: %c%c, con fin de línea.\n", car1, car2);
```

A modo de resumen:

	JAVA	**C**	**C++**
paradigma	OO	imperativo	OO + imperativo
nivel	alto	medio-alto	medio-alto
eficiencia	media	alta	alta
interpretado	sí	no	no
excepciones	sí	no	sí
clases, string…	sí	no	sí
union	no	sí	sí
booleano	sí	no	sí
declaración variables entre líneas de código	sí	no	sí
variables autoinicializadas	sí	no	no

2. Edición y compilación de un programa C

- Un programa C está compuesto por un conjunto de funciones.
- Debe existir una función llamada **`main`** ("principal"), es la que será invocada cuando se ejecute el programa.
- El programa puede estar en uno o varios ficheros, con extensión: **.c**

- Ejemplos de programas C:

nada.c

```
main () {}   /* Programa que no hace absolutamente nada */
```

hola.c

```
#include <stdio.h>
/* Este hace un poco más... */
void main (void)
{
    printf("Hola!");
}
```

argum.c

```
#include <stdio.h>  /* Puede ser necesario */
int main(int num_argumentos) /* Cuenta el num. de argumentos */
{
  printf("%d argumentos\n", num_argumentos);
  return 0;
}
```

- **Compara con java:**
  - Aplicación:

```
class <Nombre>
{
  public static void main(String[] args)
  {
     instrucciones;
  }
}
```

  - Applet:

```
class <Nombre> extends Applet
{
  public void init()
  {
     instrucciones;
  }
}
```

- **NOTA IMPORTANTE**: en C y C++, el fichero ejecutable generado corre directamente sobre la máquina hardware, no sobre la máquina virtual java, de modo que el ejecutable a generar es distinto según la máquina y sistema operativo.

- Entrar en la máquina:
  - Lilo: linux
  - login/password: mismos que suma, platón o correo ALU

- Crear y editar el fichero **.c**. Usar "**joe**", "**jed**", "**rhide**", "**vi**"...

- Compilar con **gcc**. Ejemplos de compilación:

```
>> gcc nada.c                  → El ejecutable es "a.out"
>> gcc hola.c -o hola          → El ejecutable es "hola"
>> gcc -o argum.exe argum.c    → El ejecutable es "argum.exe"
```

- Ejecutar los programas compilados:

```
>> hola        → Si no funciona es que el directorio actual no está en el PATH
>> ./hola      → para evitar ese problema
>> ./argum.exe
>> ./argum.exe uno dos tres
```

NOTA: en linux un fichero es ejecutable porque tiene permisos de ejecución, no por tener extensión .exe

3. Variables y tipos de datos básicos.

Tipo	Nombre	Bits[3]	Descripción	Ejemplo

Carácter	**char**	8	Representa un carácter ASCII, aunque se puede usar como un número entre -128 y +127.	`char c1, c2;`
				`c1= 'a';`
				`c2= 'A' + 5;`
				`printf("%c ", c2);`
				`printf("%d ", c1);`

Entero	**int**	32	Entero de 32 bits, con signo.	`int i1= 0;`
				`i1= 288;`
				`i1= 032; /*Octal*/`
				`i1= 0xf0;/*Hexad*/`
				`printf("%d", i1);`

Real	**float**	32	Número real en punto flotante.	`float f1= 0.0;`
				`f1= 3.142;`
				`f1= 3e-12;`
				`printf("%f", f1);`
				`printf("%g", f1);`
				`printf("%5f", f1);`

Real doble	**double**	64	Número real en punto flotante, con doble precisión.	`double d1;`
				`d1= 4.3e-3;`
				`printf("%f", d1);`
				`printf("%4.3f",d1);`

- Variantes: `signed/unsigned`, `short/long`
  - De **`char`**:
    - `unsigned char` (número de 0 a 255)
    - `signed char` (por defecto, equivale a `char`).
  - De **`int`**:
    - `short int` (entero corto, se puede poner sólo `short`)
    - `long int` (entero largo, se puede poner sólo `long`)
    - `long long` (entero muy largo)
    - `unsigned` (equivalente a `unsigned int`), `unsigned long`…
  - De **`double`**: `long double`.
- No existe el tipo **booleano** (en C++ sí)**:**
  - en su lugar se usan los enteros, caracteres o punteros: si vale 0 entonces significa **false**, cualquier otro valor es **true**.
  - un true "generado" es un uno cuando usado como número.
    - `int bol= (63>7);`
    - `char bol2= (5==2+3) && bol;`

- Enumerados: **`enum;`** basados en enteros; ojo, son tipo básico, *referencia directa*[4]

```
enum dias {lunes, martes, miercoles};  /* Definicion tipo   */
enum dias d1= lunes, d2= miércoles;    /* Declaracion vars  */
enum sexo {hombre=1, mujer=2, nsnc=3}; /* Asignación manual */
enum sexo s1= hombre;
enum sexo s2= 3;           /* Válido, aunque no recomendado */
printf("%d ", s1);     /* Se deben imprimir como enteros... */
```

- Probar el tamaño (en bytes) de los distintos tipos: **`sizeof(`**`tipo`**`)`**

```
printf("Tamaño de long long: %d bytes\n", sizeof(long long));
```

- ¿Cómo escribir un `unsigned`, un `long`…? → **`man -a`**

[3] Puede variar según la máquina y el compilador usados

[4] ¿conocéis el concepto de referencia directa vs. indirecta?

4. Operadores: (casi) como en java

Tipo	Operadores	Descripción	Ejemplo

Aritméticos	Suma: +
	Resta: -
	Multiplicación: *
	División: /
	Módulo: %
		Operadores aritméticos elementales. Más operaciones en las librerías matemáticas.	`char c= 'A'-'a';`
			`int i= c%4;`
			`long l= 0xff*((double) 1);`
			`double d= (2/3)*3;`

Relacionales	Igual: ==
	Distinto: !=
	Menor: <
	Menos o igual: <=
	Mayor: >
	Mayor o igual: >=
		Operadores de comparación entre enteros, caracteres y reales.	`int cond1= (32>=4);`
			`int cond2= (cond1<6);`
			`cond1==cond2;`

Booleanos	No lógico: ! Y lógico: && O lógico: ||	Operaciones lógicas, aplicadas sobre enteros, siendo 0 → **false** y otro valor **true**.	`int a, b, c;` `(!(a && (b || c)))`

Operaciones con bits	No nivel bits: ~
	Y nivel bits: &
	O nivel bits: |
	Xor nivel bits: ^
	Despl. izq.: <<
	Despl. Der.: >>
		Operaciones lógicas aplicadas a nivel de bits. En los desplazamientos: `a << c`, significa desplazar `c` bits el valor `a`.	`int a= 0xff, b= 0xf0;`
			`(a & b);`
			`(a | b);`
			`a << 2;`

Asignación[5]	Asignación: =
	Suma con asig.: +=
	Resta con asig.: -=
	*=, /=, …
		Operaciones de asignación a una variable. Las de forma: `a+= b`, son equivalentes a: `a= a + b`	`int a, b, c;`
			`a= b= c= 0;`
			`a+= 1;`
			`b+= c+= a;`

Incrementales	Preincremento: ++c
	Postincremento: c++
	Predecremento: --c
	Postdecremento: c--
		Preincremento: devuelve `c` y hace `c= c+1`
		Postincremento: hace `c= c+1` y devuelve `c`
			`char c, d= 0;`
			`c= d++;`
			`d= c++;`

Size of[6]	`sizeof(`*tipo*`)`	Devuelve el tamaño (en bytes) de un tipo de datos.	`sizeof(char);` `sizeof(double);`
Concatenación de expresiones	*expr1* , *expr2*	Une varias expresiones en una misma línea de código. Resultado = expr2.	`int i;` `i= (i= 2, i++, i*= 2);`
Condicional	*exp1* ? *exp2*:*exp3*	Si exp1, exp2, si no exp3.	`int a, b, max, min;` `min= (a<b ? a : b);`

- **Precedencia** (de + a -): aritméticos, relacionales, booleanos, asignación y coma. Ver tablas.
- **Conversión de tipos**. Puede ser de dos tipos:
  - **Implícita.** Se convierte la expresión al tipo de mayor rango:
    long double > double > float > unsigned long > long > unsigned int > int > char
  - **Explícita.** Haciendo un *casting*: `(tipo) expresión`
- ¿Cuál es el resultado de las siguientes expresiones?

```
double d; int i;     d= 1/2;     i= 1/2;     d= 1./2;    i= 0.9/2;
d= (double) 1/2;     d= ((double) (i=1))/2; (i= 1, i+= 2, i= 2*i++);
d= 3.0; d*= 2.0;     i= (d==6.0? 2 : 8);
```

[5] OJO: errores al confundir = vs == no necesariamente indicados por compilador.

[6] sizeof() es el único operador no presente en java y sí en C y C++; el motivo de que no esté en java: su objetivo es la portabilidad, pero en java ésta queda garantizada por el uso de la máquina virtual

5. Sentencias de control de flujo: (casi) como en java

- **Goto**: (existe, pero no usar, *lógica espagueti*)
- **Condicional**: **`if`** e **`if-else`**. La condición va siempre entre paréntesis.

`if (condición)`
`  sentencia`
	`if (condición)`
	`  sentencia`
	`else`
	`  sentencia`
		`if (condición)`
		`  sentencia`
		`else if (condición)`
		`  sentencia`
		`else`
		`  sentencia`

  - Ambigüedad y resolución: el **`else`** va siempre con el **`if`** más próximo.

- **Selección múltiple**: **`switch`**.
  - Según una expresión (entre paréntesis) salta a un sitio dentro de una lista de sentencias.
  - Los casos deben ser constantes (`ctei`).
  - Puede existir o no un caso por defecto.
  - A partir de un punto, la ejecución es secuencial (continúa con las sentencias siguientes).
  - Parar con **`break`**.

`switch (expresión) {`
`  case cte1: sentencias`
`  case cte2: sentencias`
`  ...`
`  default: sentencias`
`}`
	`switch (expresión) {`
	`  case cte1: sentencias`
	`  break;`
	`  case cte2: sentencias`
	`  break;`
	`  ...`
	`  default: sentencias`
	`}`

- **Iteración**:
  - **`while`** y **`do-while`**. Repetir mientras las condición sea cierta.

`while (condición)`
`  sentencia`
	`do`
	`  sentencia`
	`while (condición);`

  - **`for`**. Iteración con inicialización, condición e incremento.

Uso	Significado	Equivalente a

`for (exp1; exp2; exp3)`
`  sentencia o bloque`
	Ejecuta la expresión de inicialización `exp1`. Mientras la condición `exp2` sea **true** ejecuta `sentencia o bloque` y después `exp3` (incremento).	`exp1;`
		`while (exp2) {`
		`  sentencia o bloque`
		`  exp3;`
		`}`

OJO: no hay for para colecciones, al contrario que en java;

La ejecución de la iteración se puede detener a medio con:
  - **`break:`** continuar con la siguiente sentencia después del bucle
  - **`continue:`** comprobar condición y seguir ejecución del bucle si cierta.

- Ejemplo:

```
int i;
for (i= 1; i<=10; i++)
   printf("Paso %d\n", i);
do {
   if (i%3==0 && i%4==0)
      printf("%d es divisible por %d\n", i, 3*4);
} while (i++<100);
```

Ejercicios

1. Editar, compilar y ejecutar el siguiente programa C. Predecir el resultado antes de ejecutar.

```
#include <stdio.h>

void main (void)
{
   int i, j;
   int cond;
   for (i= 1; i<=100; i++) {
      cond= 0;
      for (j= 2; j<=i/2 && !cond; j++)
         cond= i%j == 0;
      if (!cond)
         printf("Encontrado: %d\n", i);
   }
}
```

2. ¿Cuántos bytes ocupa una variable de tipo enumerado con tres posibles valores? ¿Qué tipo de los estudiados ocupa más bytes? ¿Cuál menos?

3. Escribe un programa que calcule cuántos números enteros, entre 0 y 10000, tienen en su representación binaria 3 unos. Por ejemplo, el $111_b = 7$ sería el primero de estos números.