Algoritmos y Estructuras de Datos
Ingeniería en Informática, Curso 2º, Año 2004/2005

SEMINARIO DE C
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 estructurado de nivel alto, medio-bajo (débilmente tipado, con punteros, goto), de tamaño reducido pero con una gran potencia y expresividad.

- **Tipos de datos elementales**: enteros (`int`, `long`, `short`), caracteres (`char`), reales (`float`, `double`), enumerados (`enum`) y **compuestos**: arrays, registros (`struct`) y uniones (`union`).

- No hay distinción entre **funciones y procedimientos**. 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 "`;`".

- **Declaración de variables**:

```
<tipo_variable> <nombre_variable> [, <nombre_variable>];
```

  Las variables pueden ser **globales** (declaradas fuera de un procedimiento o función) o **locales** (declaradas al principio del código de un procedimiento).

- Se **distingue entre mayúsculas y minúsculas** en el nombre de variables, funciones, procedimientos y palabras clave. Todas las palabras clave van en minúsculas.

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

```
printf ("Hola mundo");        /* Esto es un comentario */
printf("%d", expresión_entero);
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);
```

2. Edición y compilación de un programa C

- Un programa C está compuesto por un conjunto de funciones. Debe existir una función llamada **`main`**, que 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

```
/* Este hace un poco más... */
#include <stdio.h>  /* Librería que contiene printf */

main (void)
{
    printf("Hola mundo");
}
```

argum.c

```
#include <stdio.h> /* A veces se puede omitir... */

int main(int num_argumentos) /* Cuenta el num. de argumentos */
{
  printf("%d argumentos\n", num_argumentos);
  return 0;
}
```

- Entrar en la máquina:
  - Lilo: linux
  - login: aed*XX*
  - password:

- Cambiar el password de la cuenta: **yppasswd**

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

- Compilar con **gcc**. Ejemplos:

```
>> 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
>> ./argum.exe
>> ./argum.exe uno dos tres
```

3. Variables y tipos de datos básicos.

Tipo	Nombre	Bits*	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("%c", 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`, etc.
  - De **`double`**: `long double`.

- No existe el tipo **booleano**. En su lugar se usan los enteros, caracteres o punteros: si vale 0 entonces significa **false**, cualquier otro valor es **true**.

```
int bol= (63>7);
char bol2= (5==2+3) && bol;
```

- Enumerados: **`enum`**

```
enum dias {lunes, martes, miercoles};  /* Definición del tipo */
enum dias d1= lunes, d2= miercoles;   /* Declaración de variables */

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 imprimen 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 long` o un `long double`? Ver **`man`**.

*Puede variar según la máquina y el compilador usados.

4. Operadores

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	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	`sizeof(`*tipo*`)`	Devuelve el tamaño (en bytes) de un tipo de datos.	`sizeof(char);` `sizeof(double);`
Concatenación de expresiones	*expr1* , *expr2*	Sirve para unir varias expresiones en una misma línea de código. El resultado final es el de la derecha.	`int i;` `i= (i= 2, i++, i*= 2);`

Condicional	*exp1* ? *exp2*:*exp3*	Expresión condicional. Si `exp1` es **true** el resultado es `exp2`, si no `exp3`.	`int a, b, max, min;`
			`max= (a>b ? a : b);`
			`min= (a<b ? a : b);`

- **Precedencia** (de más a menos): 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+= i++, i+= ++i);
d= 3.0; d*= 2.0; i= (d==6.0? 2 : 8);
```

5. Sentencias de control de flujo

- **Goto**: ¡¡¡Prohibido usarlo!!! (Recordar los espaguetis.)

- **Condicional**: **`if`** y **`if-else`**. La condición va siempre entre paréntesis.

```
if (condicion)
   sentencia;
```

```
if (condicion) {
   bloque sentencias
}
```

```
if (condición)
   sentencia o bloque
else
   sentencia o bloque
```

```
if (condición)
   sentencia o bloque
else if (condicion)
   sentencia o bloque
else
   sentencia o bloque
```

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 (`cte`*i*). 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 (condicion)
  sentencia;
```

```
while (condicion) {
  bloque sentencias
}
```

```
do
  sentencia o bloque
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;`
		`}`

La ejecución de la iteración se puede detener a medio con **`break`** (continuar con la siguiente sentencia después del bucle) o con **`continue`** (comprobar la condición y seguir con la ejecución del bucle si es 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? Nota: usar el operador `sizeof`.

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.