Algoritmos y Estructuras de Datos
Ingeniería en Informática, Curso 2º

SEMINARIO DE C
Sesión 3

Contenidos:

1. Funciones de entrada/salida
2. Manejo de ficheros
3. Gestión dinámica de memoria
4. Otras funciones de interés

Ejercicios

1. Funciones de entrada/salida

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

```
printf ( <cadena> );
printf ( <cadena con formato>, <expresión 1>, <expresión 2>, ...);

printf("¡Buenos días!");
printf("Un entero: %d y un real: %f\n", 27*12, 3.1416);
```

- Especificación de **formato** en `printf`:

```
% [flags] [width] [.prec] [F|N|h|l|L] caracter_tipo
```

- Caracteres de **tipo**:
d, i (decimal), o (octal), u (unsigned), x, X (hexadecimal), f (float con la forma [-]dddd.dddd), e, E, (float con la forma [-]d.dddd e[+/-]ddd), g, G (float "inteligente"), c (char), s (string), p (puntero).

- **`sprintf`**. Igual que `printf`, pero en lugar de escribir en pantalla escribe el resultado en una cadena (el primer parámetro).

```
char resultado[100];
sprintf(resultado, "Probando sprintf con el número %u.\n", -1);
printf(resultado);
```

- **Leer de teclado: `scanf`**
Formato parecido a la función `printf`.

```
int scanf ( <cadena con huecos>, &<variable1>, &<variable2>, ...);

int i, j, n;
float r;
char cadena[20];

n= scanf("%d", &i);          → Lee de teclado un entero y lo guarda en i
n= scanf("%d  %f", &j, &r);  → Lee un entero (en j) y un real (en r)
n= scanf("%s", cadena);      → Lee una cadena (hasta el primer espacio)
```

El valor devuelto por `scanf` indica el **número de huecos leídos**.
Los especificadores de formato son los mismos que para `printf`.

- **`sscanf`**. El equivalente a `scanf`, pero en lugar de leer de teclado lee los caracteres de una cadena (el primer parámetro).

```
char entrada_simulada[100]= " 26  88.3  92";
int a, b;
float g;
sscanf(entrada_simulada, "%d %f %d", &a, &g, &b);
```

- **`putchar(char c)`**. Escribe un carácter por pantalla. Equivale a:
`printf("%c", c)`
- **`char getchar(void)`**. Lee un carácter de teclado. Equivale a:

```
(scanf("%c", &c), c)
```

- **Ejemplo.**

```
#include <stdio.h>

main() {
   int i, n;
   float f;
   char cadena[30];

   printf("Introduce un entero, un float y una cadena:");
            /* Probar distintas posibilidades en la siguiente línea */
   n= scanf("%d %f %s", &i, &f, cadena);
            /* Adaptando también este printf, claro */
   printf("Huecos leídos: %d\nValores: %d, %f, %s \n", n, i, f, cadena);

}
```

- **Ojo:** normalmente la salida estándar es la pantalla y la entrada estándar el teclado. Pero se puede utilizar redirección de entrada y salida.

```
>> gcc ejer1.c –o ejer1.out      → Compilamos el programa de ejemplo
>> ./ejer1.out                  → Ejecutamos. E/S: teclado/pantalla
>> ./ejer1.out > res.txt        → Ahora la salida va al fichero res.txt
>> ./ejer1.out < in.txt         → Aquí la entrada está en el fichero in.txt
>> ./ejer1.out >res.txt <in.txt
```

La entrada estándar, salida estándar y salida de error estándar también se pueden manejar como ficheros (con los nombres **`stdin`**, **`stdout`** y **`stderr`**) con las funciones que veremos a continuación.

2. Manejo de ficheros

- Los tipos y funciones necesarios para el manejo de ficheros están definidos en la librería **`stdio.h`**.

- La librería **`stdio.h`** define el tipo **`FILE`** (ojo, en mayúsculas). Para usar ficheros debemos usar punteros al tipo `FILE`.

```
#include <sdtio.h>
FILE *f, *g;
```

- **Operaciones con ficheros:**
  - Abrir un fichero: `fopen`
  - Leer/escribir en un fichero: `fprintf, fscanf, getc, putc, fgets, fputs, fread, fwrite`
  - Moverse a un punto: `fseek`
  - Comprobar si se ha llegado al final: `feof`
  - Cerrar un fichero: `fclose`

- **Abrir un fichero**: **`fopen`**
  `FILE * fopen ( char *nombre , char *modo )`

  - **Valor devuelto:** devuelve un puntero al fichero abierto. Si no se puede abrir devuelve `NULL`.
  - **Modo de acceso:** de tipo cadena

	Fichero de texto	Fichero binario
Abrir sólo lectura	`"r"`	`"rb"`
Crear y abrir para escritura	`"w"`	`"wb"`
Abrir para añadir al final	`"a"`	`"ab"`
Abrir para lectura/escritura	`"r+"`	`"r+b"`
Crear y abrir para lectura/escritura	`"w+"`	`"w+b"`
Abrir o crear para añadir	`"a+"`	`"a+b"`

- **Cerrar un fichero**: **`fclose`**
  `int fclose ( FILE *fichero )` . Devuelve **`EOF`** si hubo algún error.

- **Leer/escribir en un fichero.**
- **`fprintf`**, **`fscanf`**. Igual que `printf` y `scanf` pero en un fichero, que se pasa como primer parámetro.

```
FILE *f;
int n;
f= fopen("prueba", "r+");
if (!f) {printf("Error. No puedo..."); return;}
fprintf(f, "Hola número %d", n);
fscanf(f, "%d", &n);
```

- **`fgetc(FILE *f)`**, **`fputc(int c, FILE *f)`**. Leer o escribir un carácter en un fichero. Devuelven la constante **`EOF`** si hay un error.

```
char c;
while ((c= fgetc(f))!= EOF)
```

```
printf("Leído: %c\n", c);
```

- **fgets**, **fputs**. Leer o escribir una línea de texto de un fichero. Recordar que `scanf("%s", ...)` lee una cadena pero acabando en el primer espacio en blanco (tabulador, intro o espacio).

`int fputs (char *cadena, FILE *fichero)`

Escribe la cadena `cadena` hasta el final de la misma (carácter `'\0'`). Devuelve `EOF` si ha ocurrido un error.

`char *fgets (char *cadena, int max, FILE *fichero)`

Lee la siguiente línea del fichero en `cadena`, acabando en un final de línea (carácter `'\n'`). Devuelve `NULL` si ha ocurrido un error.

- **fread**, **fwrite**. Leer o escribir en un fichero en **modo binario**.

`unsigned fread(void *ptr, unsigned size, unsigned n, FILE *fichero)`

Lee del fichero un bloque de `size*n` bytes y los almacena a partir de la posición apuntada por `ptr`. Se supone que `size` es el tamaño del tipo de datos a leer y `n` el número de datos. Devuelve el número de datos leídos (si no hay error, debería ser igual a `n`).

`unsigned fwrite(void *ptr, unsigned size, unsigned n, FILE *fichero)`

Escribe en el fichero un bloque de `size*n` bytes, que se encuentran a partir de la posición apuntada por `ptr`. Se supone que `size` es el tamaño del tipo de datos a escribir y `n` el número de datos. Devuelve el número de datos escritos (si no hay error, debería ser igual a `n`).

- **fseek**. Moverse a un punto concreto de un fichero.

`int fseek(FILE *fichero, long offset, int modo)`

Permite moverse a una posición cualquiera dentro de un fichero. La posición nueva está dada por `offset` (desplazamiento, en bytes) a partir de la posición dada por `modo`: `SEEK_SET` (=0) desde el comienzo del fichero; `SEEK_CUR` (=1) desde la posición actual; `SEEK_END` (=2) desde el final del fichero.
Función de consulta: `long int` **`ftell`** `(FILE *fichero)`.

- **feof**. Comprobar si se ha llegado al final de un fichero.

`int feof(FILE *fichero)`

Devuelve **true** (valor distinto de 0) si hemos llegado al final del fichero o **false** (valor 0) en caso contrario.

- **Ejemplo.**

```
#include <stdio.h>
#include <stdlib.h>    /* Función random() */
#include <time.h>      /* Función time() */

int main() {
   FILE *e, *s;
   char *nombre= "ejemplo.txt";
   int n, m, total= 0;
   if (!(e= fopen(nombre, "w"))) {
      printf("Error. No se puede crear %s\n", nombre);
      return 1;
   }
   srandom(time(NULL));  /* Inicializar generador de random */
   fprintf(e, "%d %d\n", random()%10, random()%4);
   if (fclose(e)==EOF) {
      printf("Error. No se puede cerrar %s\n", nombre);
      return 2;
   }
   if (!(s= fopen("ejemplo.txt", "r"))) {
      printf("Error. No se puede abrir %s\n", nombre);
      return 3;
   }
   while (!feof(s)) {
      total++;
      if (fscanf(s, "%d %d\n", &n, &m)==2)
         printf("Real Murcia: %d - Real Madrid: %d\n", n, m);
      else
         printf("El fichero no tiene el formato esperado\n");
   }
   printf("Realizados %d pronósticos.\n", total);
   if (fclose(s)==EOF) {
      printf("Error. No se cerrar el fichero %s\n", nombre);
      return 4;
   }
   return 0;
}
```

- Probar distintas posibilidades en la línea señalada por: ➡ , en lugar de `"w"` poner `"a"`, `"r"`, `"r+"`, `"w+"`, ... ¿Cuál es el resultado?

3. Gestión dinámica de memoria

- Se puede **reservar memoria en tiempo de ejecución**. Las funciones para reserva de memoria están en: `stdlib.h`

- Toda la memoria reservada **debe ser liberada** antes de acabar. Es decir, no hay liberación automática de memoria.

- El manejo de memoria dinámica se hace usando punteros.
```
int *arrayDimanico;
float **matrizDinamica= NULL;
```

- **Reserva de memoria: `malloc`**
```
void * malloc (unsigned tamano)
```
  - Reserva `tamano` bytes de memoria y devuelve un puntero a la zona reservada.
```
arrayDinamico = (int *) malloc (100*sizeof(int));
```
  - Devuelve `NULL` si no se ha podido reservar esa cantidad.
  - No se inicializa la memoria.
  - ¿Cómo reservar memoria para `matrizDinamica`?

- **Reserva e inicialización de memoria: `calloc`**
```
void * calloc (unsigned ndatos, unsigned size)
```
  - Reserva `ndatos*size` bytes de memoria y devuelve un puntero a la zona reservada. Se supone que `size` es el tamaño del tipo de datos a escribir y `ndatos` el número de datos.
```
arrayDinamico = (int *) calloc (100, sizeof(int));
```
  - Devuelve `NULL` si no se ha podido reservar esa cantidad.
  - Se inicializa la memoria reservada con 0.

- **Relocalizar de memoria: `realloc`**
```
void * realloc (void *ptr, unsigned tamano)
```
  - Redimensiona la cantidad de memoria reservada previamente a la nueva cantidad de bytes dada en `tamano`.

- **Liberar memoria: `free`**
```
void free(void *ptr)
```
  - Libera una zona de memoria reservada previamente. No usar si `ptr == NULL`.

- **Ejemplo**.

```
#include <stdio.h>
#include <stdlib.h>

char *memoria= NULL;

int main() {
  char c;
  long i, tam= 0;
  do {
    tam+= 1000000;
    memoria= (char *) malloc (tam);
    if (!memoria) {
      printf("Error. Imposible reservar más memoria.\n");
      return 0;
    }
    printf("Reservados %ld bytes...\n", tam);
    for (i= 0; i<tam; i++)
      memoria[i]= (char) i;
    printf("Pulse s para seguir saturando la máquina.\n");
    while ((c= getchar())=='\n');
    free(memoria);
  } while (c=='s' || c=='S');
  return 0;
}
```

- **Otro ejemplo:** páginas 310 y 311 del texto guía.

4. Otras funciones de interés

- **Con cadenas**: `string.h` `stdlib.h`

Nombre	Sintaxis	Explicación
strlen	`unsigned strlen(char *s)`	Devuelve la longitud de la cadena.
strcpy	`char *strcpy(char *des, char *src)`	Copia la cadena `src` en `des` y devuelve `des`.
strcat	`char *strcat(char *des, char *src)`	Concatena a `des` la cadena `src` y devuelve `des`.
strcmp	`int strcmp(char *s1, char *s2)`	Compara las dos cadenas. El resultado es: 0 si son iguales; <0 si `s1<s2` ; y >0 si `s1>s2`.
atoi	`int atoi(char *s)`	Convierte una cadena a un entero. Otras funciones relacionadas: `atol, atof, itoa, itol, ...`

- **Con memoria**: `string.h`

Nombre	Sintaxis	Explicación
memcpy	`void *memcpy(void *dest, void *src , unsigned n)`	Copia `n` bytes desde la posición apuntada por `src` hasta la apuntada por `dest`. Ver también `memmove`.
memcmp	`void *memcmp(void *s1, void *s2 , unsigned n)`	Compara `n` bytes de `s1` y de `s2`. El resultado es como en `strcmp`.
memset	`void *memset(void *s, int c, unsigned n)`	Escribe en los `n` bytes de `s` el valor (byte) `c`.

- **Con caracteres**: `ctype.h`

Nombre	Sintaxis	Explicación
isalpha	`int isalpha(int c)`	Devuelve **true** si el carácter `c` es una letra.
isupper	`int isupper(int c)`	Devuelve **true** si el carácter `c` es una letra mayúscula.
islower	`int isulower(int c)`	Devuelve **true** si el carácter `c` es una letra minúscula.
isspace	`int isspace(int c)`	Devuelve **true** si el `c` es un espacio, tabulador o '\n'.
toupper	`int toupper(int c)`	Devuelve el carácter `c` en mayúsculas.
tolower	`int tolower(int c)`	Devuelve el carácter `c` en minúsculas.

- **Otras**: `stdlib.h`

Nombre	Sintaxis	Explicación
system	`int system(char *command)`	Ejecuta un comando del sistema operativo. Devuelve 0 si no hay error.
exit	`void exit(int status)`	Cierra todos los ficheros y termina la ejecución del programa, devolviendo `status`.

Ejercicios

1. Busca en la ayuda (`man`) el significado y la sintaxis de las siguientes operaciones, y las librerías donde se encuentran. Escribe algún programa sencillo que las use.

a) perror
b) opendir
c) readdir
d) gettimeofday
e) excel
f) qsort
g) getline

2. Escribe un procedimiento que reserve memoria para una matriz de reales de tamaño **a×b**, y otro que multiplique dos matrices de dimensiones compatibles.

3. Usando el procedimiento anterior, escribe un programa que multiplique dos matrices **A** y **B** leídas de ficheros, la primera de tamaño **n×m** y la segunda de tamaño **m×p**. Las matrices deben leerse cada una de un fichero distinto y el resultado debe estar en otro fichero. El formato del fichero es el siguiente: en la primera línea habrán 2 enteros, indicando el tamaño de la matriz en cada dimensión. A continuación vienen las filas de la matriz, cada una en una misma línea. Los valores son números reales. El resultado debe estar en otro fichero con el mismo formato.

   Por ejemplo, un fichero de definición de una matriz podría ser el siguiente:

```
3 4
1    0.1  0    1.2
0    1.2  3.4  0.3
2.8  2.4  5.3  5.2
```

   Los nombres de los ficheros serán los argumentos del programa o (si no existen) se solicitarán a través del teclado.