

Programación de Computadores 2023-2

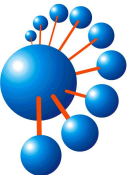
Tema 7: Entradas y salidas



Universidad
de Concepción

José Fuentes - jfuentess@inf.udec.cl

Departamento de
Ingeniería Informática y
Ciencias de la Computación



Manejo de entradas y salidas

Todo lo asociado a
entradas y salidas está
implementado en la
biblioteca `stdio.h`

- Las entradas y salidas C se manejan por medio de streams: una fuente o destino de datos que está asociado a la memoria secundaria o algún otro periférico
- En pasos simples, para utilizar streams se realiza lo siguiente:
 - a. El stream es asociado/conectado a un archivo o periférico por medio de un proceso de apertura. En el caso particular de archivos, la apertura retorna un objeto tipo **FILE**
 - b. Mientras el stream esté abierto, se puede interactuar con él
 - c. Finalmente, cerramos/desconectamos el stream
- Cuando un programa comienza a ejecutarse, tres streams se abren automáticamente (sin intervención del programador/a)
 - a. **stdin**: Entrada estándar (asociada al teclado)
 - b. **stdout**: Salida estándar (asociada a la pantalla)
 - c. **stderr**: Salida estándar para errores (asociada a la pantalla)
- En el caso de archivos, los streams deben ser abiertos explícitamente por los/las programadores

Entradas y salidas estándar (stdin/stdout/stderr)

— — —

Escribir en la salida estándar

```
int printf(char *formato, arg1, arg2, ...);
```

o

```
int fprintf(stdout, char *formato, arg1, arg2, ...);
```

Leer desde la entrada estándar

```
int scanf(char *formato, arg1, arg2, ...);
```

Escribir en la salida estándar de error

```
int fprintf(stderr, char *formato, arg1, arg2, ...);
```

Formatos de conversión (printf y scanf)

— — —

| Operador (precedidos por %) | Función |
|--------------------------------|--|
| c | Convierte un <code>int</code> a un <code>unsigned char</code> e imprimir el caracter |
| i, d | Imprime un <code>int</code> como un <code>signed int</code> (decimal) |
| u | Imprime un <code>unsigned int</code> como un <code>unsigned int</code> (decimal) |
| o | Imprime un <code>unsigned int</code> como un número octal |
| x, X | Imprime un <code>unsigned int</code> como un número hexadecimal |
| e, E | Imprime un <code>double</code> en formato exponencial <code>[-]d.ddde[+-]dd</code> |
| f | Imprime un <code>double</code> en formato exponencial <code>[-]ddd.ddd</code> |
| s | Imprime un string |
| (★) p | Imprime un puntero en formato hexadecimal |

Nota 1: Algunos operadores de conversión pueden ser precedidos por

- **h:** Asociado a `d`, `i`, `o`, `u`, `x` o `X` e indica `short` o `unsigned short`
- **l:** Asociado a `d`, `i`, `o`, `u`, `x`, `X` y `f` e indica `long` o `unsigned long`

Nota 2: Los operadores marcados con (★) no son válidos para `scanf()`

Manejo de archivos: Funciones principales

— — —

```
FILE *fp;
```

fp es un puntero de tipo **FILE**. Corresponde a la referencia a nuestro stream

```
FILE *fopen(char *name, char *mode);
```

Función para abrir el stream. La función abre un stream asociado a un archivo cuyo nombre está almacenado en **name** bajo el modo **mode**. Los modos disponibles son:

- **r**: Sólo lectura
- **w**: Crear archivo para escritura
- **a**: Escribir al final del archivo
- **r+**: Lectura y escritura desde el inicio del archivo
- **w+**: Crea un archivo en modo escritura y lectura
- **a+**: Lectura y escritura al final del archivo
- **b**: A todos los modos anteriores se le puede agregar **b** para entrar en modo binario.

Nota: En caso de error en la apertura del archivo, **fopen** retorna **NULL**

Manejo de archivos: Funciones principales

— — —

```
int getc(FILE *fp);  
int fgetc(FILE *fp);
```

Lee el siguiente caracter en el stream. Retorna un **unsigned char** representado como **int**. Retorna **EOF** si el archivo se acabó

```
int getchar();
```

Equivalente a `getc(stdin)`

```
char *fgets(char *s, int n, FILE *fp);
```

Lee los siguientes `n-1` caracteres del stream y los copia en el arreglo `s`, deteniéndose si encuentra un salto de línea. El string leído es terminado con `'\0'`

```
size_t fread(void *ptr, size_t size, size_t n, FILE *fp);
```

Lee desde el stream `n` elementos de tamaño `size` bytes y los copia en el arreglo apuntado por `ptr`.

Manejo de archivos: Funciones principales

— — —

```
int putc(int c, FILE *fp);  
int fputc(int c, FILE *fp);
```

Escribe el caracter `c` en el stream. Retorna el caracter escrito o **EOF** en caso de error

```
int putchar(int c);
```

Equivalente a `putc(c, stdout)`

```
int fputs(const char *s, FILE *fp);
```

Escribe el string `s` en el stream. Retorna un valor no-negativo o **EOF** en caso de error

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp);
```

Escribe desde el arreglo apuntado por **ptr** `n` elementos de tamaño **size** bytes hacia el stream

Manejo de archivos: Funciones principales

— — —

```
int fseek(FILE *fp, long offset, int origin);
```

Mueve la posición de acceso al stream a una nueva posición

```
long ftell(FILE *fp);
```

Retorna la posición actual del stream o -1 en caso de error

```
void rewind(FILE *fp);
```

Mueve la posición del stream al inicio. Equivalente a `fseek(fp, 0L, SEEK_SET)`

```
int fgetpos(FILE *fp, fpos_t *ptr);
```

Almacena la posición actual del stream en `ptr`

```
int fsetpos(FILE *fp, const fpos_t *ptr);
```

Mueve la posición del stream a la posición apuntada por `ptr`

```
int feof(FILE *fp);
```

Retorna un valor distinto de 0 si el stream está al final

Revisemos algunos ejemplos

— — —

Ejemplo 1:
`leer_archivo.c`

Ejemplo 2:
`concatenar_archivos.c`

Ejemplo 3:
`leer_CSV.c`

Ejemplo 4:
`serializar_arreglo.c`

Ejemplo 5:
`leer_magic_number.c`