UNIVERSIDAD SAN SEBASTIAN

VOCACIÓN POR LA EXCELENCIA

# Paradigmas de Programación

UNIVERSIDAD SAN SEBASTIAN

VOCACIÓN POR LA EXCELENCIA

# Entrada y Salida

# Entrada y Salida

- Los programas requieren comunicarse con su ambiente para realizar trabajo efectivo.

- Problema:
  - Un programador usa nombres (abstracciones) definidos al interior de un programa para representar la interacción con el ambiente.
  - Los datos se almacenan en forma externa al programa y los administra el sistema operativo.

- Se requiere asociar las abstracciones al interior de los programas con los elementos concretos externos.

# Entrada y Salida

- Un lenguaje proporciona capas de abstracción que simplifican el acceso para el programador.
  - Presentes en bibliotecas y que presentan una forma *amable* de trabajar con archivos.

- Las **primitivas** del sistema (`open()`, `read()`, `write()`, `close()`) gestionan la comunicación real con los dispositivos.
  - Estas instrucciones las interpreta el sistema operativo (no son instrucciones de hardware).
  - La interacción con el sistema se realiza a través de descriptores de archivo proporcionados por el sistema operativo.

# Entrada y Salida

- Clasificación de E/S:
  - flujos, corrientes (`stream I/O`):
    - Consiste en leer ítemes desde una fuente, uno a la vez.
    - Igualmente, se envían ítemes uno a uno hacia una salida.
  - amortiguada (`Buffered I/O`):
    - Se leen bloques de datos, los que se almacenan temporalmente en memoria.
    - Además de mejorar la eficiencia, hay dispositivos que funcionan nativamente de esta manera (disco, por ejemplo).

# Entrada y Salida

- El esquema usual de uso de entradas y salidas es:
  - `open` : establece la conexión entre el programa y su ambiente, un open por cada archivo antes de su primer uso.
  - múltiples operaciones de entrada o salida, según corresponda,
  - `close`: avisa al sistema operativo que el archivo correspondiente ya no será más utilizado. Si el programa termina sin cerrar sus archivos, los cierra al sistema.

- Como excepción a esta regla se encuentran la entrada, salida y error estándar, que se abren automáticamente al iniciar el programa.

# Eventos especiales de E/S

- Los eventos especiales de I/O son situaciones que interrumpen el flujo normal de entrada o salida:
  - **Fin de archivo** (*End of File,* `EOF`): Se alcanza el final del archivo o entrada. Esta condición es frecuente, es necesario dominarla.
  - **Archivo no encontrado** (*File not found)*: Archivo no existe o la ruta es incorrecta.
  - **Registro inexistente** (*Record not found)*: Registro que no se encuentra donde se espera (más típico en bases de datos, pero simulable en archivos).
  - **Error de E/S** (*I/O error*): Fallo al leer o escribir datos (problemas físicos, permisos, … ).

# Entrada y Salida

- Abstracción común: en la partida existen 3 corrientes (`streams`) a disposición del programador:
  - `stdin` (entrada estándar), asociada al teclado del terminal del usuario
  - `stdout` (salida estándar), asociada a la pantalla del terminal del usuario
  - `stderr` (error estándar), asociada a la pantalla del terminal del usuario
- Esta es herencia del sistema operativo Unix.

# Entrada y salida en C

- Modo `Stream`
  - Su biblioteca se encuentra en stdio.h
  - Múltiples funciones, tanto de entrada como de salida:
    - `fopen`, `fclose`, `scanf`, `printf`, `getchar`, `putchar` y otras.
  - En https://es.wikipedia.org/wiki/Stdio.h se encuentra una lista de estas funciones.
- Varias de estas funciones pueden operar sobre archivos (se antepone f al nombre de la función) o sobre memoria (se antepone s al nombre de la función).

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

# Entrada y salida en C

- Funciones de bajo nivel (*low level functions*)
  - `fd = open(name, rwmode)`: abre un archivo de nombre `name`, `wrmode` es 0, 1 o 2 para entrada, salida y entrada/salida, respectivamente. `fd` es un **descriptor de archivo** (`int`)
  - `fd = creat(name, pmode)`: crea un archivo con nombre `name`. Si el archivo existe, lo trunca a 0 caracteres, si no existe lo crea con el modo de protección que indica `pmode`.
  - `int close(fd)`: libera los recursos asociados al descriptor `fd`
  - `int unlink(const char *path)`: desconecta un archivo del sistema de archivos.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

# Ejemplo: cp simplificado

```c
#define NULL 0
#define BUFSIZE 512
#define PMODE 0644

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>

/* cp:  copy f1 to f2 */
int main(int argc, char **argv)
{
   int f1, f2, n;
   char buf[BUFSIZE];
   void error(char *, char *);
```

```c
   if (argc != 3)
      error("Usage: cp from to", NULL);
   if ((f1 = open(argv[1], 0)) == -1)
      error("cp: can't open %s", argv[1]);
   if ((f2 = creat(argv[2], PMODE)) == -1)
      error("cp: can't create %s ", argv[2]);
   while ((n = read(f1, buf, BUFSIZE)) > 0)
      if (write(f2, buf, n) != n)
         error("cp: write error", NULL);
   return 0;
}

void error (char *s1, char *s2) {
    printf(s1, s2);
    printf("\n");
    exit (1);
}
```

UNIVERSIDAD
SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

# FIN