

## Entrada – Salida

### Acceso Archivos

### Salto

### Número Variable de Parámetros

## Entrada - Salida

### • Funciones Básicas:

- Leer un carácter de la entrada estándar, entrega EOF en caso de fin de entrada:

```
c=getchar();
```

- Colocar un carácter en la salida estándar, retorna el carácter escrito o EOF en caso de error:

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

## Entrada - Salida

- La entrada o salida se puede redirigir con “<” o “>”
- Leer desde “datos.txt”:

```
programa < datos.txt
```

- Escribir en archivo “resultado.txt” la ejecución de programa:

```
programa > resultado.txt
```

## Entrada - Salida

- Escribir la salida al final de un archivo:

```
programa >> resultado.txt
```

- Interconexión de salida con entrada:

```
programa | programa2
```

- La salida de “programa” es la entrada de “programa2”

## Salida con Formato

- Función de salida “printf” convierte, da formato e imprime sus argumentos en la salida:

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

- Retorna el número de caracteres impresos
- El formato contiene 2 tipos de objetos: caracteres ordinarios y especificaciones de conversión
- La especificación de conversión comienzan con “%” y termina con un carácter de conversión

## Salida con formato

- Entre el % y el carácter de conversión puede estar:
  - Signo “-”, especifica ajuste a la izquierda
  - Número que indica ancho mínimo del campo
  - Un punto, que separa ancho de campo con precisión
  - Un número (la precisión) que indica el número máximo de una cadena que será impreso o el número de decimales o el mínimo de dígitos para un entero
  - Una “h” para un short o una “l” para un long
- Printf se confundirá y habrán resultados erróneos si no hay suficientes argumentos o son de tipos incorrectos

## Salida con formato

- Error:

```
printf(s) /* Error si s contiene % */
```

```
printf("%s", s) /* Forma correcta*/
```

- Función que escribe en un string:

```
sprintf(char *s2, char *format, arg1, arg2...);
```

- Escribe en s2 en vez de la salida estándar

## Entrada con Formato

- Función de entrada análoga a “printf”:

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

- Lee caracteres de la entrada estándar y los interpreta de acuerdo a “format”
- Almacena los resultados en los argumentos restantes
- Retorna el número de elementos de entrada que coincidieron

## Entrada con formato

- Función que lee desde un string:

```
int sscanf(char *s, char *format, arg1, arg2..);
```

- Lee desde “\*s”, de acuerdo al formato “\*format” y las coincidencias se almacenan en los argumentos
- El formato puede contener:
  - Blancos o tabuladores, los que se ignoran
  - Caracteres ordinarios
  - Caracteres de conversión

## Entrada con formato

- Ejemplo: leer líneas con fechas de la forma:

25 Dic 2003

- Con “scanf” sería:

```
int dia, agno;
char nombres[20];

scanf("%d %s %d", &dia, nombres, &agno);
```

## Acceso a Archivos

- Puntero a un archivo:

```
FILE *fp;
FILE *fopen(char *nombre, char *modo);
```

- “fopen” se encarga de negociar con el sistema operativo para obtener acceso al archivo, retornando un puntero
- El primer argumento de “fopen” es el nombre del archivo

## Acceso a Archivos

- El segundo argumento de “fopen” es el modo en que se utilizará el archivo:
  - lectura “r”, escritura “w”, añadir “a”
- Si un archivo que no existe, se abre para escribir o añadir, es creado si es posible
- Abrir un archivo existente para escribir: borra el contenido actual
- En caso de error, “fopen” retorna NULL: leer archivo inexistente, escribir sin permiso, etc

## Acceso a Archivos

- Leer de un archivo:

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

- Retorna el siguiente carácter del archivo referenciado por "fp", o EOF si hay un error
- Escribir en un archivo:

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

- Escribe "c" en el archivo "fp". Regresa el carácter escrito o EOF en caso de error

## Acceso a Archivos

- Cuando se ejecuta un programa en C, el sistema operativo es responsable de abrir 3 archivos y proporcionar punteros hacia ellos:
  - stdin: normalmente se conecta al teclado
  - stdout y stderr: se conectan a la pantalla
- Se pueden definir "getchar" y "putchar" en términos de "getc" y "putc":

```
#define getchar()    getc(stdin)
#define putchar(c)   putc((c), stdout)
```

## Acceso a Archivos

- Para entrada o salida de archivos con formato se puede usar:

```
int fscanf(FILE *fp, char *format, ...);
int fprintf(FILE *fp, char *format, ...);
```

- Los "archivos": stdin, stdout y stderr son objetos de tipo FILE \*, son constantes, no variables
- Se puede redirigir stdin y stdout hacia un archivo u otro flujo

## Acceso a Archivos

- Muy importante:

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

- Interrumpe la conexión establecida por "fopen", liberando el puntero
- La mayoría de los sistemas operativos tienen límites para el número de archivos abiertos por un programa
- fclose vacía el buffer en el cual putc está colocando la salida

## Acceso a Archivos

- Ejemplo de copiar un archivo:

```
void filecopy(FILE *in, FILE *out){
    int c;

    while((c=getc(in))!=EOF)
        putc(c, out);
}
```

- Útil para escribir el comando “cat” (mycat.c)

## Acceso a Archivos

- Funciones para testear archivos:
  - Ver si hay un error en un archivo:

```
ferror(fp); /* retorna verdadero en caso
             de error */
```

- Ver si está en EOF:

```
feof(fp); /* retorna verdadero si llego a EOF
           del archivo */
```

## Acceso a Archivos

- Cuando ocurre un error o fin de archivo, se activan indicadores de estado
- La expresión “*errno*” puede contener un número de error e información adicional acerca del error más reciente
- Se puede utilizar la función:

```
void perror(char *s)

    – Imprime “s” y un mensaje de error asociado al valor de
      errno
```

## Acceso a Archivos

- “Moverse” en archivos:

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

- Coloca el puntero “fp” a “offset” bytes de la posición “orig”
- “orig” puede ser:
  - SEEK\_SET: inicio de archivo
  - SEEK\_CUR: posición actual
  - SEEK\_END: fin de archivo

## Acceso a Archivos

- Con la función “ftell” se puede saber la posición actual del archivo (-1 en caso de error):

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

- Mover el puntero al inicio del archivo:

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

- Existen alternativas a ftell y fseek: fgetpos y fsetpos

## Ejecución de comandos

- Función:

```
int system (char * comando);
```

- Ejecuta el comando y continúa con la ejecución normal
- Regresa un estado entero propio de la ejecución del comando
- Imprimir fecha y hora actual en la salida estándar:

```
system("date");
```

## Saltos

- Funciones que permiten “salirse” desde el fondo de un conjunto de llamadas a funciones:

```
int setjmp(jmp_buf env);
void longjmp(jmp_buf env, int val);
```

- Setjmp “almacena” el contexto de ejecución (estado) del programa en “env”, para que sea usado por “longjmp”
- Setjmp retorna 0 cuando se llama directamente y no-cero cuando es retornado desde “longjmp”

## Saltos

- Longjmp restaura el contexto guardado previamente con setjmp
- Después de que longjmp haya acabado, el programa continúa normalmente, como si la llamada a setjmp hubiese retornado el valor “val”
- Si se llama a longjmp con un segundo argumento (val) de valor 0, se devuelve 1 en su lugar
- Son útiles para tratar con errores e interrupciones encontrados en una subrutina de bajo nivel de un programa

## Saltos

```
if(setjmp(env)==0){
    /* Aquí se ejecutan llamadas a funciones de cualquier
    nivel de profundidad. En cualquier momento se puede
    ejecutar:
        longjmp(env, val);
    Esto concluye abruptamente la ejecución y se retorna a
    la otra rama de este if */
} else {
    /* Aquí se llega después de que se ejecuta un longjmp
    en la otra rama del if. El segundo parámetro del longjmp
    se recibe como el valor retornado por el setjmp, y debe
    ser distinto de cero */
}
```

## Saltos

- Ejemplo:

```
main(){
    jmp_buf env;
    int i;

    i = setjmp(env);
    printf("Valor de i = %d\n", i);

    if (i != 0) exit(0);

    longjmp(env, 2);
    printf("Se imprime esta linea?\n");
}
```

## Punteros a Funciones

- No existen variables de tipo función, pero es posible tener una variable que es un puntero a una función
- Ejemplo, ordenamiento de números y strings:
  - Utilizar rutina de ordenamiento quicksort
  - Quicksort recibe como parámetro la función con la cual debe realizar las comparaciones
  - El programa principal decide si el ordenamiento es por número o string, y llama a quicksort de la manera adecuada
  - Ejemplo: ordena.c

## Número variable de parámetros

- Declaración de función con parámetros variables:

```
void funcion(val1, val2, ...);
```

- El “...” indica que el número y tipo del resto de argumentos puede variar
- En <stdarg.h> están los macros que definen como avanzar sobre una lista de argumentos
- Se utiliza el tipo “va\_list” para declarar una variable que referirá a cada argumento

## Número variable de parámetros

- La macro “va\_start” inicializa la variable para apuntar al primer argumento sin nombre
- Para recorrer el arreglo de parámetros se utiliza “va\_arg”, que regresa un argumento y avanza el puntero al siguiente
- Al finalizar, se debe llamar a “va\_end”, para realizar labores de limpieza necesarias
- Ejemplo directo es la función printf.c

## Biblioteca Estándar

- Entrada – Salida:

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

- Obliga la escritura de los datos que están en el buffer para el “fp”. Retorna 0 en caso de éxito, EOF en caso de error dando el valor apropiado la variable errno.

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

- Lee los siguientes “n-1” caracteres y los guarda en el buffer apuntado por “s”. La lectura se detiene si encuentra EOF o fin de línea

## Biblioteca Estándar

- Entrada – Salida:

```
char *gets(char *s);
```

- Lee una línea desde la entrada estándar y la almacena en “s”, hasta que se encuentre un salto de línea o EOF.
- Es MUY PELIGROSA, ya que no se hace comprobación sobre “buffer overflow”: podría comprometer la seguridad del sistema
- Se recomienda utilizar “fgets” en su lugar

## Biblioteca Estándar

- Funciones útiles:

```
double atof(char *ptr);
```

```
int atoi(char *ptr);
```

```
long atol(char *ptr);
```

```
int rand(void);
```

- Devuelve un entero pseudoaleatorio en el rango de 0 a RAND\_MAX