Otras funciones útiles:

“getpid()”

Uso: obtener PID de proceso actual.

“getppid()”

Uso: obtener PID del proceso padre.

“sleep()”

sleep(int num_segundos);

Uso: num_segundos son los segundos que el programa quedará bloqueado.

“fprintf()”

*int fprintf(FILE *stream, const char *format, ...);*

Uso: se utiliza para imprimir texto a un flujo de salida específico, . Similar a printf(), pero en lugar de escribir siempre a stdout, permite especificar el FILE * de destino.

Parámetros:

- *FILE stream: Puntero a la estructura de archivo donde se escribirá la salida (por ejemplo, un archivo abierto con fopen).
- *const char format: Cadena de formato que especifica cómo se deben imprimir los datos siguientes.
- ...: Argumentos adicionales (variables) que se formatearán según la cadena format.

Devuelve: Número de caracteres impresos (excluyendo el '\0' final) en caso de éxito. O -1 si error.

Ejemplo:

```
fprintf(fp, "Hola, tengo %d años.\n", edad);
```

“syserr()”

*void syserr(const char *mensaje);*

Uso: se utiliza para imprimir un mensaje de error personalizado junto con la descripción del último error del sistema (almacenado en errno).

Parámetros:

- **const char mensaje*: Mensaje o contexto adicional que describe la situación en la que ocurrió el error.

Ejemplo:

```
if (!fp){
        syserr("No se pudo abrir el archivo");  // Manejo adicional de error
 }
```

“gets()”

*char *gets(char *s);*

Uso: lee una línea completa desde la entrada estándar (stdin) y la almacena en la cadena apuntada por s. Deja de leer cuando encuentra un carácter de nueva línea (\n) o el fin de archivo.

Parámetros:

- **char s*: Puntero a un búfer (un array de caracteres) donde se almacenará la línea leída. Este búfer debe ser lo suficientemente grande.

Devuelve: La línea en s.

Ejemplo:

```
int main(void) {
        char buffer[100];
        printf("Ingresa un texto: ");
        if (gets(buffer) != NULL){
                printf("Leíste: %s\n", buffer);
        }else{
                printf("Fin de archivo\n");
        }
        return 0;
}
```

“strtok()”

*char *strtok(char *str, const char *delim);*

Uso: divide una cadena en subcadenas separadas por un delimitador o conjunto de delimitadores. Subsecuentes llamadas a strtok(NULL, delim) continúan analizando la misma cadena por donde lo había dejado.

Parámetros:

- *char str: La cadena que se desea fragmentar en la primera llamada. En llamadas subsecuentes, se pasa NULL para continuar con el análisis de la misma cadena.
- *const char delim: Cadena de caracteres que se utilizan como delimitadores (por ejemplo, ",", " ", etc.).

Devuelve:

- Retorna la cadena
- NULL cuando no hay más tokens.

Ejemplo:

```
int main(void) {
        char str[] = "uno,dos,tres";
        char *token = strtok(str, ",");

        while (token != NULL) {
                printf("Token: %s\n", token);
                token = strtok(NULL, ",");
         }

         return 0;
}
```

“strcmp()”

*int strcmp(const char *str1, const char *str2);*

Uso: compara dos cadenas.

Parámetros: Las dos cadenas.

Devuelve:

- Valor negativo si str1 es lexicográficamente menor que str2.
- 0 si ambas cadenas son iguales.
- Valor positivo si str1 es lexicográficamente mayor que str2.

Ejemplo: int resultado = strcmp(cadena1, cadena2);

“getchar()”

int getchar(void);

Uso: lee el siguiente carácter disponible en la entrada estándar (stdin).

Devuelve:

- El ASCII del carácter como entero.
- EOF si se llega al fin del archivo.

Ejemplo:

```
int c = getchar();
printf("Leíste el carácter: %c\n", c);
```

“sizeof()”

sizeof(<tipo o expresión>);

Parámetros: Cualquier tipo de dato.

Devuelve: cantidad de bytes que ocupa en memoria un tipo de dato.

Ejemplo:

```
printf("sizeof(int) = %zu bytes\n", sizeof(int));
```