Este archivo se dedica a explicar el concepto de las funciones como una herramienta fundamental en la programación estructurada para modularizar y organizar el código.

Aquí tienes un resumen de los puntos clave de este archivo, con referencias a las fuentes:

1. **¿Qué es una Función?**
    - Una función es un **subprograma** que lleva a cabo una **tarea específica y bien definida**.
    - Normalmente, a una función **se le pasan datos** (argumentos o parámetros) y **nos devuelve datos** (un valor de retorno).
    - La función se **ejecuta cuando es llamada** desde otra parte del programa.
    - En lenguajes como C, **no se distingue formalmente entre funciones y procedimientos**, aunque existen funciones que no devuelven valor (tipo void), que se comportan como procedimientos.
2. **Ventajas de Utilizar Funciones**
    - Hacen que el programa sea **más simple de comprender**, ya que cada módulo (función) realiza una tarea particular.
    - La **depuración** se acota a cada módulo específico.
    - Las **modificaciones** al programa se reducen a alterar solo los módulos necesarios.
    - Una vez que un módulo está **bien probado**, puede **reutilizarse** tantas veces como se requiera sin necesidad de revisarlo nuevamente.
    - Se logra **independencia del código** en cada módulo.
    - Permiten **dividir un problema complejo** en partes más pequeñas, facilitando su diseño, desarrollo, prueba y modificación de forma independiente. Estas partes se llaman habitualmente **módulos**.
3. **Conceptos Clave en el Diseño con Funciones (Modularización)**
    - **Cohesión:** Se refiere a que **cada módulo se enfoque en un único proceso o entidad**. A mayor cohesión, mejor será el módulo en términos de diseño, programación, prueba y mantenimiento. En el diseño estructurado, alta cohesión se logra cuando un módulo realiza una sola tarea sobre una sola estructura de datos. Una prueba de cohesión es si el módulo puede describirse con una oración simple con un solo verbo activo.
    - **Acoplamiento:** Mide el **grado de interrelación de un módulo con otros**. A menor acoplamiento, mejor, ya que el módulo será más fácil de manejar. Se logra bajo acoplamiento reduciendo las interacciones, la cantidad y complejidad de los parámetros, y minimizando los parámetros por referencia y los efectos colaterales.
4. **Estructura de un Programa con Funciones**
    - Uno de los módulos (funciones) debe llamarse **main**.
    - La **ejecución del programa siempre inicia en main**.
    - main puede **llamar a otras funciones**, y estas a su vez pueden llamar a otras funciones, creando una jerarquía de llamadas.
    - Las **definiciones de las funciones pueden aparecer en cualquier orden**, pero deben ser **independientes**; la definición de una función no puede estar dentro de otra.
    - Cuando una función es "llamada", sus instrucciones se ejecutan. Una función puede ser llamada desde múltiples puntos del programa.
    - Una vez terminada la ejecución de una función, el **control regresa al punto desde donde fue llamada**.
5. **Paso de Información y Retorno**

- La información se pasa a una función a través de **argumentos o parámetros**.
- Una función suele **devolver un solo valor** mediante la instrucción **return**.
- Sin embargo, hay funciones que **aceptan información pero no devuelven nada** (como printf) o que **devuelven varios valores** (como scanf).

6. **Tipos de Funciones**
   - **Funciones Predefinidas:** Ya están definidas en el lenguaje (o sus bibliotecas) y pueden usarse directamente, como POW o strlen.
   - **Funciones Definidas por el Usuario:** Creadas por el programador cuando las funciones predefinidas no cumplen una tarea específica.
7. **La Sentencia return**
   - Fuerza la **salida inmediata** del cuerpo de la función, retornando el control al punto de llamada.
   - Su forma general es return [expresión];.
   - Si una función es de un tipo distinto de void, debe tener al menos una sentencia return con un valor.
   - Si el tipo de la expresión devuelta no coincide con el tipo de la función, se realiza una **conversión automática de tipo**.
   - Una función de tipo void no necesita tener return, y si lo usa, no debe ir seguido de un valor.
8. **Ámbito de las Variables (Variables Locales y Globales)**
   - **Variables Locales:** Son las definidas dentro de las llaves de una función (incluida main). Tienen validez y existen **solo dentro de esa función**.
   - **Variables Globales:** Pueden ser usadas **desde cualquier función** y durante toda la ejecución del programa. Se definen **fuera de cualquier función**, usualmente al inicio del código, después de los include. El ejemplo de void imprimeValor() demuestra cómo una variable local contador en main es diferente de otra variable local contador en imprimeValor().
9. **Paso de Parámetros**
   - **Paso por Valor:** Se le pasa a la función una **copia del valor** de la variable. Los cambios hechos a los parámetros dentro de la función **no afectan a la variable original** fuera de ella. El ejemplo de int suma(int a, int b) ilustra esto.
   - **Paso por Referencia:** Se le pasa a la función la **dirección de memoria** de la variable. La variable que recibe este parámetro **debe ser un puntero**. Esto permite que la función **modifique el valor de la variable original** fuera de la función. El ejemplo de int suma(int *a, int *b) demuestra esto.
   - En la **llamada a la función**, los argumentos (valores o direcciones) deben pasarse **en el mismo orden** que los parámetros están definidos en el prototipo.
   - Si una función no recibe parámetros, se usan paréntesis vacíos () en la declaración y la llamada. Olvidar los paréntesis causa un error de compilación.
   - El compilador verifica que los tipos sean compatibles, aunque forzar cambios de tipo incompatibles puede no dar error pero sí resultados inesperados.
   - Los identificadores de los argumentos en la llamada no necesitan coincidir con los nombres de los parámetros formales dentro de la función.
10. **Sintaxis (Declaración, Definición, Llamada)**

- **Declaración (o Prototipo):** Indica el tipo de dato que devuelve la función, su nombre y los tipos de datos de sus parámetros. Se hace antes de main o de la primera llamada. Tipo_devuelto Nombre_de_funcion (tipo variable_1, tipo variable_2 , ...);.
- **Definición (o Desarrollo):** Es el bloque de código que implementa la lógica de la función. Tipo_devuelto Nombre_de_funcion (tipo variable_1, tipo variable_2 , ...) { ... cuerpo de la función ... }. No lleva punto y coma al final de la línea de la signatura. Los nombres de las variables en la definición son los parámetros formales, usados internamente por la función.
- **Llamada (o Invocación):** Es el punto en el código donde se usa la función. Consiste en el nombre de la función seguido de los argumentos entre paréntesis. z = suma(x, y);.

En resumen, las funciones son elementos esenciales para estructurar programas, promoviendo la organización, la reusabilidad y facilitando el manejo de la complejidad a través de la modularización, siguiendo principios de alta cohesión y bajo acoplamiento.