Las funciones son una parte fundamental de la programación en C++. Son bloques de código que realizan tareas específicas y se pueden llamar desde otros lugares en el programa. El uso de funciones facilita la modularización del código y lo hace más legible y mantenible. Hasta el momento hemos trabajado exclusivamente en una función llamada main y hemos resuelto toda la lógica de nuestro algoritmo en dicha función.
Sin embargo, cambiar a un esquema de trabajo en el que haremos uso de funciones para resolver nuestras soluciones de código nos trae una serie de beneficios pero también la necesidad de realizar una análisis inicial más sofisticado.
Bajo este nuevo paradigma de resolución de problemas en el que no resolveremos íntegramente nuestro programa en main sino que **dividiremos parte de los problemas en funciones**, es necesario determinar qué acciones se realizarán dentro de la función main y qué acciones realizarán las funciones.
De esta manera main se encargará de organizar el flujo principal del programa y delegará en llamados a funciones parte de la resolución de la lógica del mismo.

Antes de analizar de manera técnica y ver en profundidad la sintaxis necesaria para trabajar con funciones veamos teóricamente algunos conceptos:

¿Qué son las Funciones?
Una función en C++ y, prácticamente en todos los lenguajes de programación, es un bloque de código que realiza una tarea específica. Cada función tiene un nombre único y se compone de una serie de declaraciones que se ejecutan cuando la función es llamada. Las funciones pueden aceptar datos de entrada, realizar operaciones y devolver resultados. Los datos que recibe una función suelen conocerse como **parámetros**.

¿Para qué sirven las Funciones?

Las funciones tienen varios propósitos en la programación en C++:

- **Reutilización de Código**: Las funciones permiten escribir una vez una pieza de código y reutilizarla en diferentes partes del programa, evitando la duplicación innecesaria de código. Esto facilita el mantenimiento y la actualización del software.

- **Abstracción**: Las funciones ocultan los detalles de implementación, lo que facilita la comprensión del programa y permite a los programadores trabajar en niveles más altos de abstracción. Esto significa que puede pensar en una función como una **caja negra** que realiza una tarea específica sin necesidad de conocer los detalles internos.

- **División del Trabajo**: Un programa complejo se puede dividir en funciones más pequeñas y manejables. Cada función se encarga de una tarea específica, lo que facilita el desarrollo y la depuración. Esto también permite que varios programadores trabajen en diferentes partes del programa simultáneamente.

Declaración de una función

En C++, la declaración de una función se realiza indicando cual es el **tipo de dato que devolverá** la función, el **nombre** de la función y los **parámetros** que recibirá la función.
Es importante destacar que una función podría no recibir parámetros si no es necesario y también podría no devolver un valor si tampoco es requerido.

Veamos algunos ejemplos de declaración de funciones:

```
bool esNumeroPar(int numero);
```

```
void intercambiar(int &numero1, int &numero2);
```

```
string obtenerNombreDia(int numeroDia);
```

```
void mostrarSaludo(string nombre, int horaDelDia);
```

```
int lanzarDadoDeSeisCaras(); // Función que no recibe parámetros
```

Analicemos, por ejemplo, la primera de las declaraciones de funciones.

```
bool esNumeroPar(int numero);
```

Podemos observar que el valor devuelto de la función es un valor booleano. Es decir, true o false. Todavía no sabemos bien qué hará la función pero sabemos que luego de utilizarla la misma nos devolverá un valor bool para continuar con nuestro trabajo. Por ejemplo, si desde main hacemos uso de la función *esNumeroPar* entonces al utilizarla debemos suministrarle un valor entero como parámetro y sabremos que devolverá un valor bool como valor de retorno.

Luego, vemos que la función se llama *esNumeroPar*. Los nombres de las funciones siguen las mismas reglas que las de declaración de variables. Se recomienda establecer un nombre descriptivo para que al leerlo podamos inferir qué hará la función. En este caso, el nombre de la función nos da a entender que determinará si un número es par o no. Y teniendo en cuenta el valor devuelto nos indicará que es par con un true y de lo contrario nos devolverá un false. Esto empieza a tener sentido. Lo único que necesitaría la función es, de alguna manera, contar con un número para poder analizar su paridad.

Por último, podemos observar que la función recibe un solo parámetro. Un número entero llamado vagamente *numero*. Esto significa que quien use la función sólo podrá hacerlo si proporciona un número entero como parámetro. Deducimos entonces, aún sin ver el desarrollo de la función, que la función recibirá el número, determinará si es par y devolverá true o false según sea la situación.

Ahora veamos otra de las declaraciones de funciones:

```
void mostrarSaludo(string nombre, int horaDelDia);
```

En este caso, vemos una función cuyo valor devuelto es **void.** Este tipo de dato nos indica que la función no devolverá nada. Por lo que al llamarla, no debemos esperar un valor devuelto porque no lo recibiremos.
También podemos observar que recibe dos parámetros. Un string llamado *nombre* y un int llamado *horaDelDia*.
Por último, el nombre de la función *mostrarSaludo* nos da una idea de lo que la función podría llegar a hacer. Mostrará por pantalla un saludo personalizado dependiendo del

nombre de la persona y la hora del día. Por ejemplo, dirá "Buenos días, Angel"; "Buenas tardes, Laura", etc.

Tipos de Parámetros en Funciones

Las funciones pueden tener parámetros, que son valores que se pasan a la función cuando se la llama. Los parámetros permiten que las funciones trabajen con datos variables. No todos los parámetros se pasan de la misma manera. Existirán situaciones en donde queremos que la función reciba un **valor**, desde main por ejemplo, pero no tenga la capacidad de cambiarlo en main. También existirán situaciones donde vamos a querer que la función reciba una **referencia**, también por ejemplo desde main, y si cambiamos el contenido de este parámetro dentro de la función también cambie en la función que la llamó (o sea, main).

En C++, hay tres tipos principales de parámetros:

- **Parámetros por valor**: En este caso, son los valores los que se pasan a la función, pero la función trabaja con copias de esos valores. Los cambios en los parámetros no afectan a las variables originales fuera de la función.

- **Parámetros por referencia**: En este caso, se pasan a la función referencias directas de las variables con las que debe trabajar. Los cambios en los parámetros afectan a las variables originales fuera de la función. Se caracterizan por llevar un **&** antepuesto al nombre de la variable.

- **Parámetros por dirección (se ve en detalle en Programación II)**: En este caso, la función recibe las direcciones de memoria de las variables con las que deberá trabajar. Al recibir la dirección de memoria de las variables la función tiene la capacidad de acceder al contenido de la variable y también modificarlo directamente en el área de memoria. Los vectores y matrices siempre se pasan por dirección.

La palabra clave return

Hemos mencionado que una función puede devolver un valor. Por ejemplo, en la primera función que analizamos: `bool esNumeroPar(int numero);` hemos llegado a la

conclusión que la misma debe de alguna manera poder indicarnos que el número es par devolviendo el valor true o que no lo es devolviendo el valor false.
La palabra clave **return** permite realizar esa tarea. En primer lugar, finaliza inmediatamente la ejecución de la función y devuelve el flujo de ejecución a quien la haya llamado. Además, si especificamos que la función debe devolver algún valor, entonces la instrucción *return* deberá ir acompañada del valor que deberá devolver para esa ejecución de función.

Por otro lado, si la función no debe devolver nada (los casos de funciones que tenían *void* como tipo devuelto), entonces, en ese caso, la instrucción *return* simplemente indica que la función ha llegado a su fin. En estos casos, la incorporación o no de dicha instrucción *return* es opcional.

Definición de una función
La declaración de una función nos da una idea general de lo que la función devolverá, cómo se llamará y qué parámetros recibirá. Más importante aún, le indicará al compilador que habrá un elemento de nuestro programa con esas características y nos lo permitirá usar sin haberlo definido aún. Pero hasta ahora, no hemos escrito código de ninguna función. Y ese paso se realiza mediante la **definición** de una función.
En la definición de una función indicaremos exactamente qué es lo que deberá hacer la función. Línea por línea. Es decir, realizaremos el algoritmo de nuestra función dentro de ella.

```
bool esNumeroPar(int numero){
  bool esPar;
  if (numero % 2 == 0){
    esPar = true;
  }
  else{
    esPar = false;
  }
  return esPar;
}
```

Podemos observar aquí que la definición de la función incorpora un par de llaves. Dentro podremos declarar las variables que sean necesarias para el desarrollo de nuestro algoritmo. En este caso, utilizamos una variable booleana para guardarnos el valor true o false según corresponda al análisis del parámetro *numero*. Luego, como hemos aprendido, la instrucción *return* devuelve a quien haya llamado a *esNumeroPar* el valor concordante con la paridad del número suministrado.

Detengámonos un segundo para analizar la definición de la función *esNumeroPar*. Tenemos un valor devuelto, en este caso bool. Un nombre que la identifica y una serie de parámetros o un par de paréntesis vacíos si no recibe parámetros. Luego, un par de llaves y una serie de instrucciones, una debajo de la otra, en la que utilizamos variables, decisiones, ciclos, etc. para poder resolver nuestro algoritmo.
Sí, así es. Desde el primer día estuvimos haciendo definiciones de funciones. Sólo que de una en particular. Siempre estuvimos trabajando con la definición de la función main.

Veamos otro ejemplo:

```
void mostrarSaludo(string nombre, int horaDelDia){
  string saludo = "Buenas noches, ";
  if (horaDelDia ⩾ 6 && horaDelDia ⩽12){
    saludo = "Buen día, ";
  }
  else if(horaDelDia ⩾ 13 && horaDelDia ⩽ 19){
   saludo = "Buenas tardes, ";
  }
  saludo = saludo + nombre;
  cout << saludo;
}
```

Como habíamos anticipado, la función mostrarSaludo muestra un saludo personalizado dependiendo del nombre de la persona y la hora del día.

Llamado de una función

Falta el paso fundamental. Hemos declarado y definido funciones pero nos falta hacer uso de ellas. Básicamente, una función se llama desde otra función. Al principio, haremos funciones que se llamarán desde main. Pero luego, haremos funciones que se llamarán desde otras funciones también de nuestra autoría.
Por ahora, hagamos una función main que haga uso de las dos funciones que hemos desarrollado:

```
#include <iostream>
using namespace std;

void mostrarSaludo(string nombre, int horaDelDia);
bool esNumeroPar(int numero);

int main(){
  string nombreDelUsuario;
  int horaDelDia;
  int n;
```

```
  bool valorDevuelto;

  cout << "Ingresa tu nombre: ";
  cin >> nombreDelUsuario;
  cout << "Ingresa un numero para determinar su paridad: ";
  cin >> n;
  cout << "Ingresa la hora del dia en que estas usando el programa: ";
  cin >> horaDelDia;

  mostrarSaludo(nombreDelUsuario, horaDelDia);
  valorDevuelto = esNumeroPar(n);

  if (valorDevuelto == true){
    cout << "El numero ingresado es par";
  }
  else{
    cout << "El numero ingresado no es par";
  }

  return 0;
}

void mostrarSaludo(string nombre, int horaDelDia){
  string saludo = "Buenas noches, ";
  if (horaDelDia ⩾ 6 && horaDelDia ⩽12){
    saludo = "Buen día, ";
  }
  else if(horaDelDia ⩾ 13 && horaDelDia ⩽ 19){
   saludo = "Buenas tardes, ";
  }
  saludo = saludo + nombre;
  cout << saludo;
}

bool esNumeroPar(int numero){
  bool esPar;
  if (numero % 2 == 0){
    esPar = true;
  }
  else{
    esPar = false;
  }
  return esPar;
}
```