EL LENGUAJE C++

El lenguaje de programación C++, fue diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue el extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. El nombre "C++", fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

C++ (se lee "ce plus plus"), es un lenguaje de programación en el que el programador escribe lo que el ordenador debe hacer en un "código fuente". Este código es tratado por un programa especial, llamado "compilador", y que lo convierte en "código máquina". El lenguaje C++ es en realidad una ampliación del lenguaje de programación C. De ahí su nombre (C++ es C ampliado). A a su vez, anterior a estos lenguajes, aparecieron otros, como Fortran, Cobol, Basic, y el lenguaje B, que es en el que se basa el lenguaje C.

Posteriormente a C++ aparecen otros lenguajes de programación como Java, o todos los lenguajes relacionados con la creación de sitios webs (JavaScript, Php, entre otros). Algunos de estos lenguajes son específicos de un entorno (por ejemplo, JavaScript se usa exclusivamente para páginas web). Sin embargo hay muchos elementos comunes a la mayoría de lenguajes, con estructuras idénticas o similares (por ejemplo, las repeticiones en bucles, las estructuras condicionales, entre otors). C++ es un lenguaje que puede utilizarse en distintas plataformas como en el sistema operativo Windows, Linux, Androide, entre otros.

Entrada y Salida de Datos

Motivado a que C++, es una ampliación mejorada de C, permite utilizar las mismas sentencias de entrada y salida; sin embargo, por excelencia las empleadas en C++ son:

SENTENCIA DE ENTRADA DE DATOS:

cin >> Entrada de datos

SENTENCIA DE SALIDA DE DATOS:

cout << Salida de datos

Estructura básica de un programa en C++

ÁREA DE MENSAJES

// Comentario de una sola línea
/ Comentario de varias líneas */*

ÁREA DE LIBRERÍAS

 # include <nombre_librería>

ÁREA DE DECLARACIÓN DE FUNCIONES

Se declara dependiendo del tipo

ÁREA DE VARIABLES GLOBALES

Se declara dependiendo del tipo

ÁREA DE LA FUNCIÓN PRINCIPAL

```
int main()
{
        ÁREA DE VARIABLES LOCALES
}
```

ÁREA DE DESARROLLO DE FUNCIONES

Se desarrolla todo el código a ser ejecutado en la función

SINTAXIS DE LA ESTRUCTURA DE UN PROGRAMA EN C++

```
/*  EMITIR UN MENSAJE DE SALUDO AL MUNDO */
# include <iostream>
using namespace std;
int main()
{
        cout  << "Hola Mundo"  <<  endl;
        return 0;
}
```

/* */	Es el espacio reservado donde se escribe un mensaje de varias líneas, que identifican de que se trata el programa que es codificado.
#include <iostream>	Es un componente de la biblioteca estándar (STL) del lenguaje de programación C++ que es utilizado para operaciones de entrada/salida. ... **iostream** define los siguientes objetos: **cin:** Flujo de entrada y **cout:** Flujo de salida (que sale)
using namespace std;	Al ser C++ una ampliación del lenguaje C, es necesario agregar nuevas palabras reservadas. Éstas palabras reservadas están en un "namespace" (espacio de nombres). Para ser más específicos, las palabras reservadas cout y cin están en el namespace std (standard). En caso de que no se declare el uso del namespace std cada vez que se quiera usar cout, se tendría que escribir: **std::cout << "Hola mundo";**
int main() **{** **}**	Esta estructura es la función principal que contiene el código (main significa "principal"). Lo que se escriba dentro de las llaves de esta función es el código fuente en sí, es decir el programa que se ejecutará. Es posible que pueda haber instrucciones fuera de esta función, pero siempre tendrán que estar dirigidas desde esta función.
cout << "Hola Mundo" << endl;	**cout**: de C (lenguaje c) y out (fuera en inglés), cout indica que se mostrará algo en pantalla. **<<**: Estas dos flechas o signos de "mayor que", indican que va a haber una salida de datos. Son un operador de salida y después de las dos flechas se colocará los datos. **"Hola Mundo":** Datos que se observan en pantalla. En este caso es una cadena de texto, para indicar una cadena de texto se escribe la misma entre comillas. **<<**: Otra vez el operador de salida de datos. Indicamos que vamos a sacar más datos en pantalla. **endl**: Esta palabra clave indica que introducimos un salto de línea. De hecho está compuesta por "end" (final) y "l" de línea (line). **;** Toda instrucción debe terminar por un punto y coma.
return 0;	Es una función de tipo entero int main(), es decir debe retornar un valor para indicar que el programa finalizo sin ningún error.

Tipos de datos en C++

Tipo de dato	Descripción
int	Entero
Float	Números con decimales
Double	Números con decimales más largos
Char	Carácter Ejemplo: 'a'
String	Cadena de caracteres Ejemplo: "Nathalia"
Bool	Verdadero o Falso (True – False)

El tipo string

En principio el tipo string requiere que se tenga abierta la librería string, es decir, que al principio del programa, después de la línea de #include<iostream>, se debe colocar otra línea con la librería #include<cstring>, el comienzo del archivo fuente quedará así:

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

Sin embargo en los IDEs modernos, la librería iostream incluye también a la librería cstring, por lo que en realidad la segunda línea es opcional. En todo caso si se observa que al añadir variables de tipo string el programa no compila bien, entonces, se debería añadir esta segunda línea obligatoriamente.

Entrada de string:

Una de las tareas más habituales es pedirle un dato al usuario para guardarlo en una variable, y en muchos casos, esta variable será un string. En principio, lo haremos mediante el uso de cin >>, como para cualquier otro tipo de dato:

```
string nombre;
cin >> nombre;
cout << "Hola " << nombre << endl;
```

Este es un sencillo programa en el que se escribe un nombre y luego lo devuelve en pantalla. Mientras lo que se escribe es una sola palabra el programa funcionará perfectamente, pero si se escribe más de una palabra el string sólo guardará la primera. Por ejemplo, si se escribe como nombre "Nathalia", devolverá "Hola Nathalia", pero si se escribe como nombre "Nathalia Carolina", devolverá también "Hola Nathalia", con lo que no habrá guardado la segunda palabra del nombre.

Esto sucede porque la entrada de datos en pantalla considera que cada dato está separado del siguiente por un espacio en blanco (carácter espacio); con lo cual cada vez que en la entrada de datos hay un espacio se termina de leer el dato, considerando lo que se escribe después del espacio como un nuevo dato. Si se quiere, por tanto poner un string que contenga espacios en blanco, se debe por lo tanto utilizar otro mecanismo que es la función getline.

Esta es una función predefinida que provoca una entrada de datos de tipo string, y que hace que todo lo que se escriba mientras no se pulse el "enter" forme parte del mismo string, aunque haya espacios de por medio.

```
string nombre;
getline(cin, nombre);
cout << "Hola " << nombre << endl;
```

El uso de getline es muy simple, tal como se observa en la segunda línea. En primer lugar se escribe la palabra getline y luego un paréntesis dentro del cual se escribe dos elementos separados por una coma. El primero de ellos es el canal de entrada. Si se quiere recoger el dato en pantalla, se escribe siempre cin, y el segundo será la variable de tipo string en la cual se quiera guardar la cadena.

OPERADORES EN C++

Operadores Aritméticos

Operador	Acción	Ejemplo	Resultado
-	Resta	X = 5 - 3	X vale 2
+	Suma	X = 5 + 3	X vale 8
*	Multiplicación	X = 2 * 3	X vale 6
/	División	X = 6 / 3	X vale 2
%	Módulo	X = 5 % 2	X vale 1
--	Decremento	X = 1; X--	X vale 0
++	Incremento	X = 1; X++	X vale 2

Operadores Relacionales

Operador	Relación	Ejemplo	Resultado
<	Menor	X = 5; Y = 3; if(x < y) x+1;	X vale 5 Y vale 3
>	Mayor	X = 5; Y = 3; if(x > y) x+1;	X vale 6 Y vale 3
<=	Menor o igual	X = 2; Y = 3; if(x <= y) x+1;	X vale 3 Y vale 3
>=	Mayor o igual	X = 5; Y = 3; if(x >= y) x+1;	X vale 6 Y vale 3
==	Igual	X = 5; Y = 5; if(x == y) x+1;	X vale 6 Y vale 5
¡=	Diferente	X = 5; Y = 3; if(x ¡= y) y+1;	X vale 5 Y vale 4

Operadores Lógicos

Operador	Acción	Ejemplo	Resultado
&&	AND Lógico	A && B	Si ambos son verdaderos se obtiene verdadero(true)
||	OR Lógico	A || B	Verdadero si alguno es verdader
!	Negación Lógica	!A	Negación de a

Ejemplo de un Programa simple en C++

/*Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.*/

```
#include<iostream>
using namespace std;
int main()
{
	int com, sb, tot_vta, tpag, v1, v2, v3;
	cout << "Ingrese el sueldo basico: " << endl;
	cin >> sb;
	cout << "Ingrese la Venta 1: " << endl;
	cin >> v1;
	cout << "Ingrese la Venta 2: " << endl;
	cin >> v2;
	cout << "Ingrese la Venta 3: " << endl;
	cin >> v3;
	tot_vta = v1+v2+v3;
	com = tot_vta*0.10;
	tpag = sb+com;
	cin.get(); // espera un Enter, no cualquier tecla
	cout << "El total mensual para el empleado:" << tpag << endl;
	return 0;
}
```

ESTRUCTURAS DE CONTROL

Una estructura cambia el flujo normal de ejecución del programa, de manera que éste no se realice de una manera secuencial. Distinguimos entre las estructuras de control y otro tipo de estructuras como pueden ser las funciones o las clases. Hay dos tipos de estructuras de control, que son las condicionales y los bucles.

Estructuras Condicionales

La estructura if

Dentro de las estructuras de control condicionales, la más importante es la estructura if. Esta consiste en que el programador indica una condición. Esta condición es evaluada como un dato booleano. Si el valor del dato es verdadero la condición se cumple, y por lo tanto, la sentencia, o el bloque que va asociado a la condición, se

ejecuta. Si el resultado de la condición es falso, la sentencia o bloque, no se ejecuta. Es decir, si la condición que se establece se cumple, el programa realiza la acción, y si no es así no la realiza. La sintaxis que se usa para esta estructura es la siguiente:

```
if (condición)
 {
    <bloque de sentencias>
}
```

Cada uno de los componentes de esta estructura son los siguientes:

- **if:** Es la palabra clave que indica el tipo de estructura. Esta palabra se pone siempre al principio.
- **(condicion):** Entre paréntesis se escribe una condición. Esta tiene que dar un resultado booleano de "true" o "false", por lo que en la mayoría de las veces suele ser una operación lógica o condicional, por ejemplo: ('a' > 'b').
- **{<bloque de sentencias>}:** Después de la condición se escribe el código al que se quiera que afecte la estructura. Éste, si tiene más de una sentencia, debe ir en un bloque, (escrito entre llaves), ya que de otra manera sólo afectaría a la primera sentencia.

Ejemplo:

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

int main()
{
        int a;
        cout << "Escribe un número mayor que 10 : ";
        cin >> a;

        if  (a > 10)
        {
                cout << "Bien Hecho !! " << endl;
                cout << "El número " << a << " es mayor que 10." << endl;
         }

        return 0;
}
```

La estructura if – else

Si lo que se quiere es que la condición se cumpla y se ejecute un código, y cuando no se cumpla se ejecute otro distinto, lo que se debe hacer es añadir después dela estructura if la palabra clave else, seguida del bloque de sentencias que debe ejecutarse cuando la condición no se cumple. La estructura completa es:

```
if (<condicion>)
{
        <sentencias cuando SI se cumple>;
  }
else
{
         <sentencias cuando NO se cumple>;
}
```

Es decir, el programa evalúa la condición escrita después del if, y si el resultado es verdadero se ejecuta el primer bloque de sentencias, pero si es falso se ejecutará el bloque que hay detrás del else.

Ejemplo:

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

int main()
{
        int a;
        cout << "Escribe un número mayor que 10 : ";
        cin >> a;

        if (a > 10)
         {
                cout << "Bien Hecho !! " << endl;
                cout << "El número " << a << " es mayor que 10." << endl;
        }
        else
        {
                cout << "Numero equivocado!! " << endl;
                cout << "El número " << a << " NO es mayor que 10." << endl;
        }

        return 0;
```

```
}
```

Condicional switch – case (Múltiple)

Posteriormente de conocer la estructura if y else, para elegir lo que el programa debe hacer. Cada una de las opciones viene controlada por una condición, si se cumple la condición se ejecuta la opción, y si no se pasa a comprobar la siguiente.

La estructura switch tiene también la misma función, pero aquí, todas las opciones dependen del valor de una variable. Se tiene una variable, y cada opción consiste en comparar esa variable con otro elemento, de manera que si el valor de la variable y el del elemento comparado son iguales, el resultado es verdadero, la condición se cumple, y se ejecuta el código asociado. Si no es así se pasa a la siguiente opción donde se vuelve a repetir el proceso. Si no se ejecuta ninguna opción puede ponerse una opción por defecto al final. El código de esta estructura es el siguiente:

```
switch (x)
{
        case 1:
                <sentencias para x == a>;
                break;
        case 2:
                <sentencias para x == b>;
                break;
        .
        .
        .
        case n:
                <sentencias para x == n>;
                break;
        default:
                <sentencias para x == n>;
}
```

El código está estructurado de la siguiente manera:

- Empezamos poniendo la palabra clave switch, y después, entre paréntesis la variable con la que compararemos las demás (x).
- El resto del código forma un bloque, y va escrito entre llaves. Ester código se compone de varias opciones.

- Cada una de las opciones (excepto la última) empieza por la palabra clave case seguida del elemento con la que se hace una comparación de igualdad y después dos puntos case a:
- Cada una de las opciones (excepto la última) acaba con la palabra clave break;
- En cada una de las opciones anteriores, entre el código del principio (case a:) y el del final (break;), se escriben las sentencias que deben ejecutarse cuando la condición se cumpla.
- La opción final indica lo que debe ejecutarse en el caso en que ninguna de las anteriores se cumpla, y empieza por la palabra clave default: (acabada con dos puntos), seguida de las sentencias que deben ejecutarse cuando no se cumpla ninguna opción anterior.

Como se observa se pueden colocar tantas opciones case.... break; como se quiera o se necesite. El case a: indica que se inicia una nueva opción, la cual ejecutará el código que le sigue en el caso de que la variable principal (x) sea igual a la indicada a.

El break; hace que el flujo interrumpa, y se salga del bloque. De no ponerlo el programa no sale del bloque y el resto del código del bloque, perteneciente a otras opciones, también se ejecutaría.

La última opción default: indica lo que se debe hacer cuando no hay ningún elemento anterior que coincida. Aunque no es obligatorio ponerla, es conveniente si se quiere que el programa haga algo cuando no se cumplen las opciones anteriores. Al ser la última no es necesario poner el break; al final, ya que es ahí donde termina el bloque.

Ejemplo:

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

int main()
{
        cout << "OPERACIONES CON DOS NUMEROS" << endl;

        //Decalramos las variables

        int a, b;
        char op;

        cout << "Escribe un numero : ";cin >> a; //pedimos el primer número
        cout << "Escribe otro número : "; cin >> b; //pedimos el segundo número
        cout << "Tus numeros son " << a << " y " << b << endl; //confirmamos números
        cout << "Elige una operacion, para ello escribe su letra: " << endl; //explicación al usuario
        cout << "s = Suma; r = Resta; m = Multiplicacion; d = Division." << endl;
        cout << "Tu operacion : ";
```

```
        cin >> op;              //pedimos operación
        switch (op)                     //inicio estructura.
        {
                case 's':   //opción 1: suma
                            cout << "Operacion : Suma " << endl;
                             cout << a << " + " << b << " = " << a+b << endl;
                            break;
                case 'r':    //opcion 2: resta
                             cout << "Operacion : Resta " << endl;
                            cout << a << " - " << b << " = " << a-b << endl;
                            break;
                case 'm':   //opcion 3: multiplicacion
                            cout << "Operacion : Multiplicacion " << endl;
                            cout << a << " * " << b << " = " << a*b << endl;
                            break;
                case 'd':   //opcion 4 : división
                            cout << "Operacion : Division " << endl;
                            cout << a << " / " << b << " = " << a/b << endl;
                            break;
                default:    //Respuesta para opción equivocada.
                            cout << "La letra escrita no se corresponde con ninguna operacion." << endl;
        }
        return 0;
}
```

Estructuras Repetitivas

Ciclo While

Un ciclo o bucle, en programación, es una sentencia o bloque de sentencias que se repiten de forma continua hasta que una condición, que va asociada al bucle, deje de cumplirse. Se utiliza normalmente para realizar acciones repetidas, sin tener que escribir repetidamente el código. Esto ahorra tiempo y líneas de código.
En un ciclo se tienen los siguientes elementos:

- **El nombre, o palabra clave**; con el que identificamos al bucle.
- **La sentencia o bloque de sentencias** ; que deben repetirse
- **La condición**; La cual mientras se cumpla seguirá repitiéndose el bloque.
- **La variación de la condición**; La condición inicial debe variar, normalmente en cada repetición, de manera que llegue un momento que no se cumpla, para poder salir del bucle.

El tipo de bucle más sencillo es el bucle while. Para crear un bucle while, simplemente hace falta poner la palabra clave while seguida de la condición entre paréntesis, y después el bloque de sentencias, su sintaxis es como se muestra a continuación:

```
while (<condicion>)
{
        <Sentencias que se repiten>
}
```

Ejemplo:

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

int main()
{
        cout << "Tabla del 7" << endl << endl;
        int i= 0;

        while (i <=10)
        {
                cout << 7 << " x " << i << " = " << 7*i << endl;
                i = i+1;
        }

        return 0;
}
```

Ciclo do – while

El bucle do while es muy parecido al bucle while. La gran diferencia es que aquí, aunque la condición no se cumpla, el código se ejecutará siempre al menos una vez. Esto es debido a que primero se ejecuta el código, y después se comprueba la condición. Si ésta es verdadera, el bucle se repite, si no es así se sale de él y continúa con el resto del programa.

Sintaxis de do while

La sintaxis es ligeramente diferente del bucle while, ya que aquí la condición se india al final de la estructura:

```
do
{
        <Sentencias que se repiten>;
} while (<condicion>);
```

Como con cualquier otro bucle, hay que tener en cuenta que tras un número de repeticiones tiene que llegar un momento en que la condición no se cumpla, para poder salir del bucle, y no caer en el error de hacer un bucle infinito.

Ejemplo:

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

int main()
{
        int i=0, r;
        cout << "Escribe un numero para repetir la frase.";
        cin >> r;
        cout << "La frase se escribirá al menos una vez aunque el número sea cero o negativo." << endl;

        do
        {
                i++;
                cout << i << ".- Esta es una frase repetida" << endl;
        }
        while(i < r);

        return 0;
}
```

Ciclo For

En los bucles for tanto la inicialización como la actualización forman parte de la estructura, de manera que la variable de control es controlada en la misma estructura.

Sintaxis del bucle for: Su estructura es la siguiente:

for (<inicializacion>;<condicion>;<actualización>) { <sentencias que se repiten>; }

El código de esta estructura:

- **for** : Empezamos el bucle escribiendo la palabra clave for
- **<inicialización>**: Entre paréntesis y separados por punto y coma se escribe tres sentencias, la primera de ellas es la "inicialización", donde se da un valor inicial a la variable de control. Si ésta no estaba declarada anteriormente, la podemos declarar aquí también ej.: int i=0;.
- **<condición>**: La segunda sentencia del paréntesis es la "condición", en donde se indica la condición que tiene que ser verdadera para que el bucle se repita. Evidentemente en la condición tiene que aparecer la variable de control. por ejemplo: i <= 10;.
- **<actualización>** : En esta tercera sentencia de dentro del paréntesis indicamos qué es lo que le debe ocurrir a la variable de control tras cada repetición, para que ésta cambie y en un momento dado se pueda salir del bucle, en el ejemplo que estamos siguiendo, sería el incremento en una unidad: i++;.
- **{<sentencias que se repiten>;}:** Dentro de las llaves, escribimos el bloque de sentencias que se repetirán en cada vuelta.

Siguiendo con los ejemplos que hemos puesto antes podemos escribir un bucle que escriba una tabla de multiplicar, por ejemplo la del 5:

```
for (int 1 = 0 ; i <= 10 ; i++)
{
        cout << "5 x " << i << " = " << 5*i << endl;
}
```

Ejemplo:

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

int main()
{
        cout << "TABLAS DE MULTIPLICAR" << endl;
        int t;
```

```
        cout << "Que tabla quieres? (del 1 al 10) :";
        cin >> t;
        if (t >=1 and t<=10)
        {
                for (int i=1; i<=10 ; i++)
                {
                        cout << t <<  " x " << i << " = " << t*i << endl;
                }
        }
        else
        {
                cout << "Error: tu número no está entre el 1 y el 10." << endl;
        }

        return 0;
}
```

FUNCIONES EN C++

Una función es un bloque de código que realiza alguna operación. Una función puede definir opcionalmente parámetros de entrada que permiten a los llamadores pasar argumentos a la función. Una función también puede devolver un valor como salida. Las funciones son útiles para encapsular las operaciones comunes en un solo bloque reutilizable, idealmente con un nombre que describa claramente lo que hace la función.

Ejemplo: La siguiente función acepta dos enteros de un autor de llamada y devuelve su suma; a y b son parámetros de tipo: int

```
int sum(int a, int b)

{

   return a + b;

}
```

La función puede ser invocada, o llamada, desde cualquier lugar del programa. Los valores que se pasan a la función son los argumentos, cuyos tipos deben ser compatibles con los tipos de los parámetros en la definición de la función.

```
int main()
{
   int i = sum(10, 32);
   int j = sum(i, 66);
   cout << "The value of j is" << j << endl; // 108
}
```

No hay ningún límite práctico para la longitud de la función, pero el buen diseño tiene como objetivo las funciones que realizan una sola tarea bien definida. Los algoritmos complejos deben dividirse en funciones más sencillas y fáciles de comprender siempre que sea posible.

Las funciones definidas en el ámbito de clase se denominan funciones miembro. En C++, a diferencia de otros lenguajes, una función también pueden definirse en el ámbito de espacio de nombres (incluido el espacio de nombres global implícito). Estas funciones se denominan funciones libres o funciones no miembro; se usan ampliamente en la biblioteca estándar.

Las funciones pueden ser sobrecargadas, lo que significa que diferentes versiones de una función pueden compartir el mismo nombre si difieren por el número y/o tipo de parámetros formales.

ELEMENTOS DE UNA DECLARACIÓN DE FUNCIÓN

Una declaración de función mínima consta del tipo de valor devuelto, el nombre de la función y la lista de parámetros (que pueden estar vacíos), junto con palabras clave opcionales que proporcionan más instrucciones al compilador.

El siguiente ejemplo es una declaración de función:

```
int sum(int a, int b);
```

Una definición de función consiste en una declaración, más el cuerpo, que es todo el código entre las llaves:

```
int sum(int a, int b)
{
   return a + b;
}
```

Los elementos necesarios de una declaración de función son los siguientes:

El tipo de retorno, que especifica el tipo del valor que devuelve la función, o void si no se devuelve ningún valor. En C++11, auto es un tipo de retorno válido que indica al compilador que infiera el tipo de la sentencia de retorno. En C++14, decltype(auto) también está permitido. Para obtener más información, consulte más adelante Deducción de tipos en tipos de valor devueltos.

El nombre de la función, que debe comenzar con una letra o un carácter de subrayado y no puede contener espacios. En general, los caracteres de subrayado iniciales en los nombres de función de la biblioteca estándar indican funciones miembro privadas o funciones que no son miembro que no están diseñadas para su uso por parte del código.

La lista de parámetros, que es un conjunto delimitado por llaves y separado por comas de cero o más parámetros que especifican el tipo y, opcionalmente, un nombre local mediante el cual se puede acceder a los valores de dentro del cuerpo de la función.

Los elementos opcionales de una declaración de función son los siguientes:

1. constexpr, que indica que el valor devuelto de la función es un valor constante que se puede calcular en tiempo de compilación.

```
constexpr float exp(float x, int n)
{
   return n == 0 ? 1 :
      n % 2 == 0 ? exp(x * x, n / 2) :
      exp(x * x, (n - 1) / 2) * x;
};
```

2. Su especificación de vinculación, extern o static.

```
//Declare printf with C linkage.
extern "C" int printf( const char *fmt, ... );
```

3. inline, que indica al compilador que reemplace todas las llamadas a la función con el propio código de la función. La inserción en línea puede mejorar el rendimiento en escenarios donde una función se ejecuta rápidamente y se invoca varias veces en una sección del código crítica para el rendimiento.

```
inline double Account::GetBalance()
{
   return balance;
}
```

4. Una noexcept expresión que especifica si la función puede o no lanzar una excepción. En el ejemplo siguiente, la función no produce una excepción si la is_pod expresión se evalúa como true.

```
#include <type_traits>
template <typename T>
T copy_object(T& obj) noexcept(std::is_pod<T>) {...}
```

5. (solo funciones miembro) Los calificadores cv, que especifican si la función es const o volatile.
6. (solo funciones de los miembros) virtual, override, o final. virtual especifica que una función se puede reemplazar en una clase derivada. override significa que una función de una clase derivada reemplaza una función virtual. final significa que una función no se puede invalidar en ninguna clase derivada adicional. Para más información, consulte Funciones virtuales.
7. (solo funciones miembro) static aplicado a una función miembro significa que la función no está asociada a ninguna instancia de objeto de la clase.
8. (solo funciones miembro no estáticas) El calificador de referencia, que especifica al compilador qué sobrecarga de una función, debe elegir cuando el parámetro implícito del objeto (*this) es una referencia rvalue frente a una referencia lvalue.