Programación estructurada

1.- INTRODUCCIÓN

El lenguaje C se origina con el lenguaje BCPL, creado por Martin Richards, el lenguaje B implementado por Ken Thompson y por último el lenguaje C , implementado por Dennis Ritchie en 1971.

Lenguaje C es un lenguaje de programación estructurado de propósito general. Se usa a un nivel más bajo, entre lenguaje máquina y el lenguaje de alto nivel.

En lenguaje C la unidad básica de programación es la función y el lenguaje C++ es un lenguaje hibrido que maneja la programación estructurada y la programación orientada a objetos lo que incrementa su productividad.

1.1.- Estructura de un programa en C y C++

Todo programa en C consta de una o más funciones, una de ellas es main. El programa siempre comienza por la ejecución de la función main.

Cada función debe contener:

- Una cabecera de la función, que consta del nombre de la función, después unos paréntesis para una lista opcional de argumentos.
- Después el cuerpo de la función que describe el conjunto de sentencias o conjunto de instrucciones encerradas por un par de llaves. **Cada sentencia de expresión debe terminar con punto y coma (;)**

C++ es muy potente, complejo y extenso y como sus estructuras básicas son las mismas que otros muchos lenguajes resulta de interés trabajar con él

1.2.- Programar bajo entorno Linux

En Linux podemos utilizar editores de texto preinstalados en el sistema como Gedit y compiladores. Tecleamos el programa y lo guardamos con extensión cpp. Por ejemplo guardaremos el programa que vamos a realizar como hola.cpp, abriremos un terminal y teclearemos g++ hola.cpp –o hola (g++ es nombre del compilador, hola.cpp es el nombre del programa, -o hola es lo que indica el nombre que tendrá el ejecutable).
Si no aparece ningún mensaje de error, el último paso es lanzarlo: ./hola

Programar bajo entorno Windows:

En Windows no hay ningún editor avanzado ni compilador así que necesitaremos descargar un entorno de desarrollo que incluya editor y compilador. Vamos a descargar Dev-C++ versión (5.x), "Orwell Dev_C++" (en la versión que incluye el

compilador TDM-GCC), instalarlo y crear un nuevo programa en File>New>Source File. El programa se guardará con una extensión .cpp.
El botón compile permite comprobar si hay errores y generar el ejecutable. Al hacer clic sobre Run se lanzará el programa resultante.

Hola, mundo

Empezamos por escribir algo en la pantalla.

```
//Primer programa de ejemplo en C++ Comentario

#include <iostream>                  Se trata de incluir una librería para que reconozca los procedimientos de entrada y salida con cin y cout que se usan para guardar algo introducido por teclado en una variable, leer teclado o mostrar algo por pantalla, escribir en pantalla.

int main  ()                         Lo que aparece a continuación es el Cuerpo del programa

{
std::cout <<" Hola mundo";           Escribe en pantalla lo que se indica entre comillas dobles.

return 0;                            Indica que el programa ha terminado sin errores


}
```

2.- PROGRAMACIÓN EN C++, CÁLCULOS.

2.1.- Realizar operaciones prefijadas

Para realizar **operaciones matemáticas** indicamos la operación **sin cerrarla entre signo menor doble**:

```
//Suma de dos números
#include <iostream>
int main ()
{
     std::cout <<6+7;                Cuando ejecutamos el programa en pantalla aparece 13

     return 0;
}
```

Los operadores aritméticos son:

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

Hallar el resto de una división también llamado módulo es una importante operación en programación.

```
//módulo
#include <iostream>
int main ()
{
     std::cout <<48%5;
     return 0;
}
```

El resultado es el resto de dividir 48 entre 5; es decir 3.

2.2.- Escribir varios textos

Para escribir varios textos utilizamos una única orden cout pero **cada texto precedido por <<**

```
//Suma y resultado
#include <iostream>
int main ()
{
     std::cout <<"6+7="  << 6+7;
     return 0;
}
```

Cuando el programa se ejecute aparecerá en pantalla 6+7= 13

2.3.- Escribir en varias líneas

Enviar cout con un símbolo especial denominado std::endl (abreviatura de end of line), **salto de línea.**

```
//Suma y resultado en dos líneas
#include <iostream>
```

```
int main ()
{
    std::cout <<6+7=   <<std::endl << 6+7 <<std::endl ;
    return 0;
}
```

Cuando el programa se ejecute aparecerá en pantalla 6+7=13, con salto de línea.

2.4.- Pedir datos al usuario

Para leer datos se emplea la orden std::cin, y estos datos se guardaran en una variable.**Al principio del programa se deben declarar las variables**, detallando el tipo de datos que se van a guardar, por ejemplo, **int si es un número entero y el nombre de esa variable**.

Vamos a pedir un número al usuario y cuando se introduzca ese número, se calculará su cuadrado.

```
//Cuadrado de un número
#include <iostream>

int main ()
{
    int numero;
    std::cout<<"Dime un numero:";
    std::cin>>numero;
    std:: cout <<"El cuadrado de tu numero es"<<numero*numero;
    return 0;
}
```

numero guarda un número entero
Pedimos el número al usuario
Leemos un dato que guardamos en la variable numero.

Calculamos y mostramos el resultado.

Problema 1: Calcula la resta de dos números que introduzca el usuario. Utiliza tres variables.

2.5.- Evitar escribir std::

En programas largos puede resultar tedioso escribir std:: delante de cada cout y de cada cin. Añadiendo **using namespace std; al principio del programa** se evitará esto último.

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

int main ()
{
      int n1, n2, suma;
      cout<<"Dime un numero:";
      cin>>n1;
      cout<<"Dime otro numero:";
      cin>>n2;
      suma = n1+n2;
      cout <<"Su suma es"  <<suma <<endl;

      return 0;
}
```

2.6.- Números con decimales

Para poder ver los decimales de los números resultado de una operación, deberemos utilizar una variable de **tipo float**, que tiene una **precisión de siete cifras**:

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

int main ()
{
      float n1, n2, cociente;
      cout<<"Dime un numero:";
      cin>>n1;
      cout<<"Dime otro numero:";
      cin>>n2;
      cociente = n1/n2;
      cout <<"La división es"<<cociente <<endl;

      return 0;
}
```

Si queremos una precisión mayor utilizaremos una variable de **tipo double que tiene 15 cifras de precisión** y permite disminuir errores de redondeo aunque ocupa más espacio.

2.7.- Funciones matemáticas

Para utilizar funciones matemáticas que tiene incorporadas C++, incluiremos cmath al principio del código (**#include<cmath>**).En funciones trigonométricas, el ángulo se indica en radianes, no en grados.

- **Raíz cuadrada: sqrt(x)**
- **x elevado a y: pow(x,y)**
- **coseno: cos(x)**
- **seno: sin(x)**
- **Tangente: tan(x)**
- **Exponencial de x (e elevado a x): exp(x)**
- **Logaritmo natural o neperiano en base e: log(x)**
- **Logaritmo en base 10: log10(x)**

```
//seno
#include <iostream>
#include<cmath>
using namespace std;

int main ()
{
      int numero=9;
      double raiz = sqrt(numero)";
      cout<<"La raiz de:" <<numero<<"es"<<raiz << endl;

      float angulogrados=45;
      float PI=3.14159265;
      double anguloradianes=angulogrados*PI/180;
      cout <<"El seno de"     <<angulogrados << "es"<<sin(anguloradianes) << endl;

      return 0;
}
```

Problema 2: Crea un programa que calcule el coseno de ángulos con los siguientes grados: 30, 45,60 y 90

Problema 3: Realiza un programa que pida cuatro notas de examen a un alumno calcule su media.

Problema 4: Crea un programa que pida al usuario una cantidad de pulgadas y calcule su equivalencia en metros (1 pulgada = 0,0254 m). ¿qué tipo de variable necesitas usar?

3.- CONDICIONES

3.1.- if

Para comprobar si se cumple una condición se emplea if (condición) sentencia;

```
//comprobar si un número es positivo
#include <iostream>
using namespace std;

int main ()
{
	int numero;
	cout<<"Escribe un número:";
	cin>> numero;
	if (numero>0)
		cout <<"El número es positivo.";

	return 0;
}
```

La orden if permite ejecutar una única sentencia en caso de que se cumpla la condición, para poder realizar varias sentencias las **agruparemos entre llaves** {} formando una sentencia compuesta.

```
// sentencia compuesta
#include <iostream>
using namespace std;

int main ()
{
	int numero;
	cout<<"Escribe un número:";
	cin>> numero;
	if (numero>0)
	{
		cout <<"El número introducido es positivo.";
		cout <<" También puedes escribir números negativos.";
	}

	return 0;
}
```

3.2.- El caso contrario: else

Qué hacemos si no se cumple una condición.

```
// if y else
#include <iostream>
using namespace std;

int main ()
{
	int numero;
	cout<<"Escribe un número:";
	cin>> numero;
	if (numero>=0)
		cout <<"El número introducido es positivo.";

	else
		cout <<"El número introducido es negativo.";

	return 0;
}
```

3.3.- Operadores relacionales: <, <=, >, >=, ==, !=

La mayor parte de los operadores de comparación están claros pero hay algunos un poco menos evidentes:

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

```
// Comprobar el valor de una variable
#include <iostream>
using namespace std;
```

```
int main ()
{
      int numero;
      cout<<”Escribe un número:”;
      cin>> numero;
      if (numero==13)
            cout <<”Has escrito trece.”;

      return 0;
}
```

Problema 5
Crea un programa que pida un número entero al usuario e indique si es múltiplo de 5. Para ello, deberá comprobar si el resto que obtiene al dividir dicho número entre cinco es cero.

Problema 6
Realiza un programa que pida dos números e indique si el primero es múltiplo del segundo.

Problema 7
Crea un programa que pida al usuario dos números enteros y diga cuál es menor.

Problema 8
Realiza un programa que pida dos número al usuario. Si el segundo no es cero, mostrará la división de ambos; de lo contrario, aparecerá un mensaje de error.

3.4.- Condiciones con operadores lógicos: &&, ||, !

Las condiciones se pueden encadenar con operadores lógicos “y”, “o”, “no” como se muestra en la tabla adjunta.

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

```
// operadores lógicos
#include <iostream>
using namespace std;

int main ()
{
      int numero;
      cout<<"Escribe un número:";
      cin>> numero;
if (!(numero==0))
      cout <<"No es cero.";
if ((numero==2)||(numero==3))
      cout <<" Es dos o tres.";
if ((numero>=2)&&(numero<=7))
      cout <<" Esta entre 2 y 7 (incluidos).";

      return 0;
}
```

3.5.- Operador condicional: ?

Para asignar un valor a una variable en función de si se cumple o no una condición utilizaremos el operador condicional u operador ternario **?**.

Variable = condición ? valor si se cumple: valor si no se cumple

Si se cumple la condición, toma el primer valor y si no se cumple es segundo.

Veremos un ejemplo en comparación con el if equivalente:

```
// Operador condicional
#include <iostream>
using namespace std;

int main ()
{
      int a=3, b=5;
      int mayor= a>b ? a:b
      cout<<"El mayor es:" << mayor;

      return 0;
}


// If alternativo al operador condicional
```

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

int main ()
{
      int a=3, b=5, mayor;
if (a>b)
      mayor = a;
else
      mayor = b;
      cout <<" el mayor es:"<< mayor;

      return 0;
}
```

3.6.- switch

Para analizar varios valores posibles de una misma variable, se puede emplear la orden switch. La variable entre paréntesis y cada posible valor se indica tras la palabra **case:**. Las ordenes a ejecutar deben terminar con **break** y usando **default** se indica qué hacer si no se da ninguno de los casos preestablecidos.

```
// Condiciones con switch
#include <iostream>
using namespace std;

int main ()
{
      int numero;
      cout <<" Introduce un número del 1 al 5";
      cin>> numero;
      switch (numero)
      {
            case 1: cout<<"Uno";
                  break;
            case 2: cout<<"Dos";
                  break;
            case 3: cout<<"Tres";
                  break;
            case 4: cout<<"Cuatro";
                  break;
            case 5: cout<<"Cinco";
                  break;
            default: cout<<"Valor incorrecto!";
      }
```

```
    return 0;
}
```

Cuando varios casos tengan que mostrar un mismo resultado, se puede dejar vacío el primero de ellos. Por ejemplo como 9 y 10 son ambos un sobresaliente,

```
case 9:
case 10: cout<<"Sobresaliente";
    break;
```

Problema 9
Crea un programa que pida al usuario tres números reales e indique el valor numérico del mayor de ellos. Tener en cuenta que el usuario puede meter algún valor repetido.

Problema 10
Elabora un programa que pida al usuario dos números enteros y diga:" Uno de los números es negativo", "los dos números son negativos" o bien " Ninguno de los números es negativo", según corresponda.

Problema 11
Diseña, usando if, un programa que pida al usuario un número del 1 al 10 y diga si es par o no.

Problema 12
Elabora, usando switch, un programa que pida al usuario un número del 1 al 10 y escriba el nombre de la nota correspondiente.

Problema 13
Crea un programa, usando switch, que pida a un usuario un número del 1 al 10 y diga si es múltiplo de 3 o no.

4.- BUCLES

Muchas veces tendremos que verificar las condiciones más de una vez y para ello utilizaremos bucles.

4.1.- while

Para repetir una condición utilizaremos while (mientras) en lugar de if.

```
// Pedir un número positivo con while
#include <iostream>
```

```
using namespace std;

int main ()
{
	int numero;
	cout <<" Escribe un numero positivo";
	cin>> numero;

	while (numero<=0)
	{
	cout <<" Ese numero no es positivo"<<endl;
	cout <<" Escribe otro numero positivo";
	cin>> numero;
	}

	return 0;
}
```

Si el usuario escribe un número negativo, le dirá que no es negativo y que escriba uno positivo tantas veces como el usuario meta números negativos. Una vez que meta un número positivo, se saldrá del while y terminará el programa porque ya no se cumple la condición del while.

4.2.- do-while

El do-while, haz mientras se cumpla la condición establecida.

```
// Pedir un número positivo con do-while
#include <iostream>
using namespace std;

int main ()
{
	int numero;
	do
	{
	cout <<" Escribe un numero positivo";
	cin>> numero;
	if(numero<=0)
		cout <<" Ese numero no es positivo"<<endl;
	}
	while (numero<=0);
	return 0;
}
```

Con while, si introducimos un número positivo la primera vez, la condición del while no se cumple y por tanto no se entrará en el bucle while y el programa terminará sin

embargo con do-while, siempre se realizará una vez las órdenes que se indiquen ya que la condición se comprobará al final.

4.3.- Contadores

En programación se suelen usar mucho contadores para lo que usaremos while, aumentando el valor de una variable.

```
// Contador con while
#include <iostream>
using namespace std;

int main ()
{
	int numero=1;
	while (numero<=10)
	{
	cout<<numero<< "  ";
	numero=numero+1;
	}
	return 0;
}
```

El contador mostrará los números de 1 al 10 separados por tantos espacios como se meta entre las comillas.

4.4- for

Para crear contadores se puede usar una orden, for (para) de modo que podamos agrupar los tres primeros pasos, valor inicial, incremento y comprobación.

```
// Contador con for
#include <iostream>
using namespace std;

int main ()
{
	int numero;
	for(numero=1; numero<=10; numero=numero+1)
		cout<<numero<< "  ";
	return 0;
}
```

Podemos salir de un bucle antes de tiempo utilizando la orden break.

```
// Interrumpir un bucle con break
#include <iostream>

using namespace std;

int main ()
{
	int numero;
	for (numero=1; numero<=10; numero++)
	{
		if (numero == 5)
		break;
		cout << numero <<" ";
	}
return 0;
}
```

A veces programamos de forma que un bucle no tenga salida de forma intencionada o por error.

```
// Bucles sin fin
#include <iostream>
using namespace std;

int main ()
{
	while(3>2) cout>>"Hola";
	for (int numero=0; numero<=10;)
	cout << " Hola ";
	for (;;)
	cout << " Hola ";
	return 0;
}
```

Podemos anidar bules para por ejemplo escribir tablas de multiplicar del 1 al 5.

```
// Bucles anidados
#include <iostream>
using namespace std;

int main ()
{
	int tabla, numero;
	for (tabla=1; tabla<=5; tabla++)
	{
```

```
            for(numero = 1; numero<=10; numero++)
            cout<<tabla<<"por"<<numero <<"es"<<tabla*numero<<endl;
            break;
            cout << endl;
      }
      return 0;
}
```

4.5- Incremento

Incrementar variables es una operación tan común que algunos lenguajes incorporan una forma abreviada **numero++**.

```
// Contador con for
#include <iostream>
using namespace std;

int main ()
{
      int numero;
      for(numero=1; numero<=10; numero++)
            cout<<numero<< " ";
      return 0;
}
```

4.6 Otras operaciones aritméticas abreviadas

Para aumentar o disminuir el valor de una variable en varias unidades utilizamos una notación abreviada en vez de **valor = valor + 3** utilizamos **valor += 3**

```
// Contador con for decrementando de 2 en 2
#include <iostream>
using namespace std;

int main ()
{
      int numero;
      for(numero=10; numero>=0; numero-=2)
            cout<<numero<< " ";
      return 0;
}
```

4.7- Declarar una variable dentro de for

Podemos declarar variables en otro sitio que no sea al principio del programa. Es muy normal que declaremos variables al principio de un for de modo que la variable desaparece al terminar el for.

```
// Declarar una variable dentro de un for
#include <iostream>
using namespace std;

int main ()
{
	for(int numero=10; numero>=0; numero-=2)
		cout<<numero<< " ";
	return 0;
}
```

Problema 14
Diseña un programa, usando while, que escriba en pantalla, de mayor a menor, los números impares pares del 1 al 30.

Problema 15
Elabora un programa que dé al usuario la oportunidad de adivinar un número del 1 al 100, en un máximo de seis intentos. En cada intento, el programa deberá avisar al usuario de si se ha pasado o se ha quedado corto.

Problema 16
Crea un programa que pida números positivos al usuario y calcule la suma de todos ellos (saldremos del bucle con un número negativo o con el cero).

Problema 17
Elabora un programa que pida al usuario su código de usuario y su contraseña, y que no le permita finalizar hasta que introduzca el código de usuario 2017 y la contraseña 7890.

5.- ESTRUCTURAS BÁSICAS DE DATOS

5.1.- Contacto con los arrays

Un array es un conjunto de elementos del mismo tipo. Por ejemplo, un array "datos" que tenga capacidad para 7 elementos (int datos[7]) contendrá desde el primer elemento datos[0] hasta el último datos[6]. En los arrays se suele utilizar un for.

```
// Arrays1
#include <iostream>
using namespace std;
```

```
int main ()
{
      int datos[7] = (300,200,100,0,-100,250,175);
      int suma;
      int i;
      for (i=0; i<=6; i++)
      suma +=datos[i];
      cout<<”Su suma es”<<suma;
      return 0;
}
```

5.1.- Un array para almacenar datos

Se suelen utilizar mucho los arrays para guardar datos que mete el usuario y luego deberán ser manipulados, por ejemplo pedir al usuario que meta cinco datos y que el programa los muestre en el orden inverso al que fueron introducidos.

```
// Arrays2
#include <iostream>
using namespace std;

int main ()
{     int datos[5];
      int i;
      for (i=0; i<=4; i++)
      {
      cout<<”Dime el dato”<<i+1<<”:”;
      cin>>datos[i];
      }
      for (i=4; i>=0; i--)
      cout<<”El dato”<<i+1<<”es”<<datos[i]<<endl;
      return 0;
}
```

Problema 18

Crea un programa que pida cuatro números al usuario, los memorice utilizando un array, calcule su media aritmética y , finalmente, muestre en pantalla tanto la media como los datos tecleados.

5.2.- Arrays bidimensionales

Para declarar arrays de dos o más dimensiones por ejemplo si deseamos guardar datos de dos o más grupos de medidas en un experimento, existen dos opciones:

- Usar double medidas[20] y recordar que las diez primeras medidas pertenece un primer grupo y el resto al segundo grupo.

- Usar double medidas [2][10] de modo que agrupamos en 2 grupos de 10 datos cada grupo las medidas siendo [0][i]los datos del primer grupo de medidas y [1][i] los datos del segundo grupo de medidas y de forma que el primer dato es el 0 y el último es 9 de cada grupo.

```
// Arrays bidimensionales
#include <iostream>
using namespace std;

int main ()
{     int medidas[2][10];
            {
            {2.25, 2.27, 2.30, 2.32, 2.33, 2.35, 2.36, 2.37, 2.38, 2.39}
            {2.23, 2.26, 2.27, 2.29, 2.31, 2.32, 2.33, 2.35, 2.38, 2.40}
            };
      cout<<"La tercera medida de la serie 2 es "<<tiempos[1][2];
      return 0;
}
```

5.3.- Valores iniciales de arrays

Los datos de un array no tienen por qué tener como valor inicial 0, ya que pueden contener "basura", es decir, lo que en la posición de memoria que se está usando quedé guardado de una operación anterior al igual que ocurre en los espacios reservados para variables simples de modo que:

```
//Contenido de datos sin inicializar
#include <iostream>
using namespace std;

int main ()
{     float valor;
      int datos[3];
      cout<<"valor vale"<<valor<endl;
      for (int i=0; i<=2; i++)
      cout<<"El dato"<<i+1<<"es"<<datos[i]<<endl;
      return 0;
}
```

5.4.- Arrays sobredimensionados

Cuando no se sabe la cantidad de datos que se van a almacenar empleamos arrays de gran tamaño y llevamos un contador de la cantidad de datos que realmente contiene por ejemplo:

```
//Array sobredimensionado
#include <iostream>
using namespace std;

int main ()
{     int datos[1000];
      int contador=0;
      int actual;
      do
            {
            cout<<"dime un dato (5555 para salir):";
            cin>>actual;
            if ((actual != 55555) && (contador<=999))
                  {
                  datos[contador] = actual;
                  contador++;
                  }
            }
      while ((actual != 55555) && (contador<=999));
      cout<<"Los"<<contador<<"datos son: ";
      for (int i=0; i<contador; i++)
      cout<<datos [i]<<" ";
      return 0;
}
```

Problema 19
Crea un programa que pida 11 números al usuario, los memorice (utilizando una tabla no variables), calcule su media aritmética y finalmente muestre en pantalla los datos tecleados y la media.

Problema 20
Elabora un programa que almacene en una tabla el número de días que tienen cada mes (suponiendo que se trata de un año no bisiesto), pida al usuario que indique un mes determinado (considerando 1=enero u 12= diciembre) y muestra en pantalla el número de días que tiene el mes indicado.

Problema 21
Crea un programa que pida al usuario diez números enteros e indique cuál es el menor.

Problema 22
Busca información sobre la "ordenación de burbuja" y crea un programa que pida diez datos numéricos al usuario y los muestre ordenados de menor a mayor.

6.- FICHEROS

Para no perder la información que maneja un determinado programa, volcaremos los datos antes de salir del programa a un fichero.
Para manejar ficheros, hay que primero abrir el fichero, luego leer sus datos o escribirlos en el fichero y por último cerrar el fichero.
Hay que recordar que si queremos abrir un fichero, éste tiene que existir y para escribir en un fichero éste no puede ser de sólo lectura.

6.1.- Escritura en un fichero de texto

Para volcar los datos se debe crear un **ofstream** (output file stream) y luego cerrarlo al final.

```
// Escritura en un fichero de texto
#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
ofstream fichero("prueba.txt");
fichero <<"Esto es una linea" <<endl;
fichero <<"Esto es otra linea" <<;
fichero <<" y esto es la continuación de la linea anterior" <<endl;
fichero.close();
cout<<"Fichero creado"<<endl;
return 0;
}
```

6.2.- Lectura de un fichero de texto

Para leer un fichero, el fichero será un **ifstream** (input file stream) y la lectura se realiza con **>>**.

```
// Lectura de una palabra de un fichero de texto
#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
ifstream fichero("prueba.txt");
string palabra;
fichero>>palabra
cout<<"Se ha leido:"<<endl;
```

```
cout<<palabra << endl;
fichero.close();
return 0;
}
```

6.3.- Leer toda una línea, incluyendo espacios

Cin se detiene cuando encuentra el primer espacio pero si queremos leer toda la línea, incluyendo espacios, se puede usar **getline(fichero, texto);**

```
// Lectura de una frase de un fichero de texto
#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
ifstream fichero("prueba.txt");
string linea;
getline(fichero, linea);
cout<<"Se ha leido la linea"<<endl;
cout<<linea << endl;
fichero.close();
return 0;
}
```

6.4.- Lectura hasta el final del fichero

Pero normalmente se leerá todo el contenido del fichero para lo cual se comprobará si se ha alcanzado su **.eof()** (end of file), que será "verdadero" si el fichero ha terminado.

```
// Lectura hasta el final de un fichero de texto
#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
string linea;
ifstream fichero("prueba.txt");
while(!fichero.eof())
      {
      getlinte(fichero,linea);
      if (!fichero.eof())
```

```
        cout<<linea<<endl;
        }
fichero.close();
return 0;
}
```

Esta será la estructura básica de casi cualquier programa que deba leer un fichero completo, de principio a fin: abrir, procesar todos los datos con **while(!fichero.eof())** y cerrar. Por ejemplo, se podrían mostrar y numerar así las líneas de un fichero.

```
int contador=1;
ifstream fichero("prueba.txt");
while(!fichero.eof())
        {
        getlinte(fichero,linea);
        if (!fichero.eof())
        cout<<contador<<":"<<linea<<endl;
        contador++;
        }
fichero.close();
```

6.5.- Pedir el nombre al usuario

El nombre del fichero no tiene porqué estar prefijado en el programa. Normalmente se pide al usuario.

```
// Pedir el nombre de un fichero de texto al usuario
#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
string nombre;
string linea;
cout<<"Dime el nombre del fichero:";
getline(cin, nombre);
ifstreamfichero(nombre.c_str());
while(!fichero.eof())
        {
        getlinte(fichero,linea);
        if (!fichero.eof())
        cout<<linea<<endl;
```

```
    }
fichero.close();
return 0;
}
```

6.6.- Errores en el acceso a ficheros

Con .fail(), podremos comprobar si ha habido algún error por ejemplo si se ha intentado abrir un fichero que no existe. Esto será verdadero en caso de que exista algún problema.

```
// Comprobación de errores al acceder a un fichero
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
using namespace std;

int main ()
{
string nombre;
string linea;
cout<<”Dime el nombre del fichero:”;
getline(cin, nombre);
ifstreamfichero(nombre.c_str());
if(fichero.fail()
{
cout<<”¡No existe el fichero!”<<endl;
exit(1);
}
while(!fichero.eof())
{
getlinte(fichero,linea);
if (!fichero.eof())
cout<<linea<<endl;
}
fichero.close();
return 0;
}
```

Problema 23

Crea un programa que pida números y los guarde en un fichero numeros.txt, separados por espacios en blanco. Finalizará cuando se introduzca el número 0.

Problema 24

Elabora un programa que pida un nombre de fichero y muestre su contenido, haciendo una pausa cada 7 líneas (lee línea a linea con getline y comprueba si (numerolinea%7==6)

Problema 25
Diseña un programa que muestra el resultado de la suma de los números almacenados en numeros.txt

7.- FUNCIONES

7.1.- Problemas de un código repetitivo

Aparecen las funciones como herramienta para no repetir un mismo código varias veces en un programa ya que repetir código puede causar varios problemas como:

- Llevará más tiempo escribir el programa
- El programa resulta menos legible
- Pueden cometerse más errores al escribir muchas líneas de código.

Un ejemplo de programa repetitivo puede ser cuando queremos que aparezca en pantalla varios textos subrayados.

```
#include <iostream>
using namespace std;
int main ()
{
	int i;
	cout<<"primer ejercicio"<<endl;
	for(i=0; i<20; i++)
		cout<< " - ";
	cout<<endl;

	cout<<"segundo ejercicio"<<endl;
	for(i=0; i<20; i++)
		cout<< " - ";
	cout<<endl;

	cout<<"tercer ejercicio"<<endl;
	for(i=0; i<20; i++)
		cout<< " - ";
	cout<<endl;

	return 0;
}
```

Este código hace aparecer en pantalla Primer ejercicio subrayado, luego segundo ejercicio subrayado y así sucesivamente.

Crearemos una función subrayar que realice el código repetitivo y la llamaremos cuando queramos que aparezca.

```
    cout<<"primer ejercicio"<<endl;
    subrayar();

    cout<<"segundo ejercicio"<<endl;
    subrayar();

    cout<<"tercer ejercicio"<<endl;
    subrayar();

    return 0;
}
```

7.2.- Primera función

Para hacer funciones haremos preceder el nombre de la función de **void** y pondremos entre llaves los pasos que debe dar dicha función.

```
//primera función
#include <iostream>
using namespace std;

void subrayar()
{
int i;
for(i=0; i<20; i++)
cout<< " - ";
cout<<endl;
}
int main ()
{
    cout<<"primer ejercicio"<<endl;
    subrayar();

    cout<<"segundo ejercicio"<<endl;
    subrayar();

    cout<<"tercer ejercicio"<<endl;
    subrayar();
 return 0;
}
```

Este programa tiene dos bloque uno de ellos es **main**, el cuerpo del código, el otro bloque, **subrayar** que se encarga de realizar la tarea repetitiva.

El término void indica que no se trata de una función matemática que devuelva un resultado numérico, sino de un procedimiento o subrutina que realiza algo y no devuelve ningún resultado.

Además la variable i es una variable local ya que solo es accesible dentro del bloque en el que está declarada luego su valor no se podrá consultar ni variar desde main. A diferencia de las variables globales que son las que se meten fuera de main y de subrayar de modo que sea accesible desde ambas funciones como se ve en el siguiente ejemplo.

```
//Ejemplo de variable global
#include <iostream>
using namespace std;
int n; //varible global
void duplicarN()
{
n=n*2;
}
int main ()
{
  n=5;
  duplicarN();
  cout<<n<<endl;
  return 0;
}
```

7.3.- Parámetros de una función

Es muy habitual indicar a la función los datos con los que deseamos que trabaje.
Por ejemplo en el caso de antes para indicar la cantidad de guiones que queremos que ponga en cada caso.

```
//función con parámetro
#include <iostream>
using namespace std;

void subrayar(int cantidad)
{
int i;
for(i=0; i<cantidad; i++)
cout<< " - ";
cout<<endl;
```

```
}
int main ()
{
      cout<<”primer ejercicio”<<endl;
      subrayar(15);

      cout<<”segundo ejercicio”<<endl;
      subrayar(18);

      cout<<”tercer ejercicio”<<endl;
      subrayar(21);
  return 0;
}
```

Veamos el ejemplo de una función con dos parámetros:

```
 //función con dos parámetros
#include <iostream>
using namespace std;

void mostrarsuma(int n1, int n2)
{
cout<< n1+n2<<endl;
}
int main ()
{
int n=5;
mostrarsuma(12,n);
return 0;
}
```

Ejercicio 26
Crea una función escribir tabla que muestre en pantalla la tabla de multiplicar del número que el usuario introduzca por teclado.

7.4.- Valor devuelto por una función

Hay veces que necesitamos que una función realice una serie de cálculos y devuelva el resultado para usarlo en otra parte del programa por ejemplo creamos una función para elevar un número al cubo:

```
//función que devuelve un valor
#include <iostream>
using namespace std;
```

```
int cubo(int n)
{
int resultado=n*n*n;
return resultado;
}
int main ()
{
int numero;
cout<<”Dime un numero: ”;
cin>>numero;
cout<<”Su cubo es ” <<cubo(numero)<<endl;
return 0;
}
```

Como podemos observar una función que devuelve un valor no será **void** sino que será **int** o de algún otro tipo y deberá terminar por la orden **return.**

Si una función puede devolver un valor de entre varios distintos, puede contener varias órdenes **return** distintas. Por ejemplo, creamos una función que diga cuál es el mayor de dos números reales:

```
//función con varios return
#include <iostream>
using namespace std;
float mayor(float n1, float n2)
{
if (n1>n2)
return n1;
else
return n2;
}
int main ()
{
float numero1, numero2;
cout<<”Dime un numero: ”;
cin>>numero1;
cout<<”Dime otro numero: ”;
cin>>numero2;
cout<<”El mayor es ” <<mayor(numero1, numero2)<<endl;
return 0;
}
```

Ejercicio 27

Crea una función suma que devuelva la suma de tres número que se indiquen como parámetros.

Ejercicio 28
Crea una función esprimo que reciba un número y devuelva el valor 1 si es primo y el valor 0 si no lo es.

7.5.- Modificar el valor de un parámetro

Cuando queremos modificar el valor de un dato que una función reciba como parámetro, los cambios no se conservan cuando se sale de dicha función como podemos ver en el ejemplo siguiente:

```
//Intentar modificar parámetro
#include <iostream>
using namespace std;
void duplicar(int n)
{
n=n*2;
}
int main ()
{
int numero=5;
cout<<"El numero vale"<<numero<<endl;
duplicar(numero);
cout<<"Tras duplicar se convierte en " <<numero<<endl;
return 0;
}
```

Al realizar este ejemplo observamos que el resultado de la función no se guarda en numero y por lo tanto sale que el duplicado sigue siendo el número sin duplicar.
Para solucionar este problema basta con indicar un tipo de datos distinto de void y usar una orden return como estudiamos en el apartado anterior.

Habrá ocasiones en que nos interese modificar el valor de un parámetro, por ejemplo en las ocasiones en que haya que devolver más de un resultado. Para conseguirlo, debemos preceder el nombre del parámetro por el símbolo **&** (denominado "ampersand" en inglés y "et", en francés) con el fin de indicar que los cambios que se realicen en ese parámetro deben conservarse:

```
//Modificar un parámetro
#include <iostream>
using namespace std;
void duplicar(int &n)
{
n=n*2;
```

```
}
int main ()
{
int numero=5;
cout<<”El numero vale”<<numero<<endl;
duplicar(numero);
cout<<”Tras duplicar se convierte en “ <<numero<<endl;
return 0;
}
```

A esta forma de pasar parámetros modificables se la conoce como “paso por referencia” frente a la forma habitual denominada “paso por valor”.

Ejercicio 29
Crea una función intercambia que reciba dos números enteros e intercambie sus valores por medio de la utilización de dos parámetros por referencia**: intercambiar(int &a, int &b).**