

Fundamentos de la Programación. Prácticas de Laboratorio

E.T.S. Ingeniería Informática. Universidad de Málaga.

Contenido

- Práctica de Laboratorio 1. Tema 2. Code::Blocks e Introducción a C++2
- Práctica de Laboratorio 2. Tema 2. Estructuras de Control: Sentencias de Selección 11
- Práctica de Laboratorio 3. Tema 2. Estructuras de Control: Sentencias de Iteración 15
- Práctica de Laboratorio 4. Tema 3. Subprogramas: Procedimientos y Funciones (I)20
- Práctica de Laboratorio 5. Tema 3. Subprogramas: Procedimientos y Funciones (II)23
- Práctica de Laboratorio 6. Tema 4. Tipos Estructurados: Registros y Arrays (I)29
- Práctica de Laboratorio 7. Tema 4. Tipos Estructurados: Registros y Arrays (II)34
- Práctica de Laboratorio 8. Tema 4. Tipos Estructurados: Cadenas de Caracteres (Strings) .. 42
- Práctica de Laboratorio 9. Tema 4. Resolución de Problemas Utilizando Estructuras de Datos47



Esta obra se encuentra bajo una licencia Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional (CC BY-NC-SA 4.0) de Creative Commons.

Fundamentos de la Programación. Práctica de Laboratorio 1

E.T.S. Ingeniería Informática. Universidad de Málaga.

Tema 2: Entorno de Programación Code::Blocks e Introducción a C++

La metodología de trabajo durante la realización de las prácticas en el laboratorio será la siguiente para cada ejercicio:

- En la carpeta **Documentos**, cree una carpeta con el *nombre-del-alumno*, y en esta carpeta, desarrolle los programas con los nombres especificados para cada ejercicio de la práctica.
- Durante la práctica en el laboratorio, el alumno debe dedicar un **máximo de 15 minutos** para **desarrollar** la solución del problema, corregir los errores de compilación, ejecutar el programa y depurar los errores que encuentre.
- Aquellos ejercicios que el alumno no tenga tiempo de terminar durante la práctica en el laboratorio, se deberán terminar en casa, y entregar todos los ejercicios dentro del plazo especificado para la tarea correspondiente.
- Posteriormente, cuando se publique la solución de los ejercicios, el alumno debe **analizar** la solución proporcionada por el profesor en el campus virtual y compararla con su propia solución.
- Los ejercicios serán **corregidos** comparando la salida que produce la ejecución del programa con respecto a la salida especificada en el enunciado de cada ejercicio, por ello, cada programa debe mostrar una interacción con el usuario **exactamente** como se muestra en el enunciado de cada ejercicio.
- El nombre del fichero de cada ejercicio debe ser como se especifica en el enunciado, por ejemplo, el primer ejercicio de la práctica 1 se debe denominar **p1e1.cpp**, y así sucesivamente.

Criterios de codificación:

- El uso de **variables globales** está **prohibido**. Sólo se pueden utilizar variables locales, **declaradas dentro de main**.

Ejercicios de Laboratorio

Ejercicio 1 (p1e1.cpp)

Para que el alumno se familiarice con el entorno de programación (IDE) **Code::Blocks**, el alumno debe seguir las indicaciones que se proporcionan en el documento de la **Guía de uso del entorno de desarrollo Code::Blocks** que se encuentra en el *Campus Virtual*.

En dicho documento se proporciona información sobre como crear, editar y modificar programas fuente en el lenguaje C++, así como su compilación y finalmente la ejecución del programa ejecutable ya compilado.

El programa creado durante la realización de la *Guía de uso del entorno de desarrollo Code::Blocks*, denominado **euros.cpp**, debe renombrarse como **p1e1.cpp**, para que forme parte de los ejercicios de la práctica 1.

En este programa, el programa debe leer de teclado una determinada cantidad de pesetas, y debe calcular la cantidad de euros equivalente, considerando que 1 euro equivale a 166.386 pesetas.

La ejecución del programa desarrollado, cuando se introduce por teclado la cantidad de 500 pesetas, produce el siguiente resultado:

```
Introduzca la cantidad de pesetas: 500
500 pesetas equivalen a 3.00506 euros
```

Ejercicio 2 (p1e2.cpp)

El siguiente programa escrito en C++ calcula la cantidad bruta y neta a pagar por un trabajo realizado en función de las horas y días trabajados. Sin embargo, en el momento en que se intenta compilarlo se producen una serie de errores. El alumno debe localizar dichos errores y corregirlos. Para ello debe examinar los mensajes que proporciona el compilador e interpretarlos convenientemente.

```
#include <iostream>
using namespace std;
const tasa : 25.3;
const PRECIO_HORA = 60.75;
int main()
{
    double horas, dias, total, neto;
    cout << "Introduzca las horas trabajadas: ";
    cin << horas;
    cout << "Introduzca los dias trabajados: ";
    cin >> dias;
    horas*dias*PRECIO_HORA = total;
    neto = total-TASA;
    cout >> "El valor total a pagar es: " >> total >> endl;
    cout << "El valor neto a pagar es: " << NETO << endl;
}
```

Una vez compilado correctamente, la ejecución de este programa para 5 horas trabajadas durante 2 días produce el siguiente resultado:

```
Introduzca las horas trabajadas: 5
Introduzca los dias trabajados: 2
El valor total a pagar es: 607.5
El valor neto a pagar es: 582.2
```

Ejercicio 3 (p1e3.cpp)

Desarrolle un programa que lea dos números de tipo `int` de teclado y posteriormente muestre en pantalla los valores de los dos números leídos.

A continuación, realice **tres ejecuciones** del mismo programa, **sin modificar el programa**, introduciendo los valores que se especifican a continuación para cada una de las ejecuciones:

- Ejecútelos introduciendo dos números de tipo `int` válidos (por ejemplo 1234 y 5678), y deberá producir el siguiente resultado:

```
Introduzca un número entero: 1234
Introduzca otro número entero: 5678
El valor del primer número introducido es: 1234
El valor del segundo número introducido es: 5678
```

- Posteriormente ejecútelos introduciendo por teclado un primer número de tipo `int` (por ejemplo 1234) e introduciendo por teclado un segundo dato que no pertenezca al tipo `int` (por ejemplo hola), y deberá producir el siguiente resultado (en lugar del número cero, podrá mostrar cualquier **valor inespecificado**):

```
Introduzca un número entero: 1234
Introduzca otro número entero: hola
El valor del primer número introducido es: 1234
El valor del segundo número introducido es: 0
```

- Finalmente ejecútelos introduciendo por teclado un primer dato que no pertenezca al tipo `int` (por ejemplo hola), y deberá producir el siguiente resultado (en lugar de los números cero, podrá mostrar cualquier **valor inespecificado**), **nótese que en este caso, el valor del segundo numero no será leído de teclado**:

```
Introduzca un número entero: hola
Introduzca otro número entero: El valor del primer número introducido es: 0
El valor del segundo número introducido es: 0
```

Realice las ejecuciones del programa especificadas anteriormente y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

Finalmente, analice las diferencias entre las tres ejecuciones del mismo programa, escriba la explicación sobre el comportamiento de la ejecución en un comentario en su código, y posteriormente compruebe la solución proporcionada por el profesor cuando sea publicada.

Téngase en cuenta que el comentario de explicación proporcionado por el alumno **no puede ser corregido de forma automática**, por lo que el alumno debe comprobar su propia explicación con respecto a la explicación proporcionada por el profesor, cuando ésta sea publicada.

Ejercicio 4 (p1e4.cpp)

Desarrolle un programa que visualice por pantalla el tamaño en bytes que ocupan todos y cada uno de los tipos básicos vistos en clase: `bool`, `char`, `int`, `unsigned`, `double`, etc. Para ello debe usarse el operador `sizeof(tipo)`. Por ejemplo, para el tipo `int`:

```
cout << "int: " << sizeof(int) << " bytes" << endl;
```

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
bool: 1 bytes
char: 1 bytes
int: 4 bytes
unsigned: 4 bytes
double: 8 bytes
```

Ejercicio 5 (p1e5.cpp)

Desarrolle un programa que lea de teclado una palabra de cuatro letras por teclado (se deben leer cuatro caracteres y almacenarlos en cuatro variables de tipo `char`), y posteriormente escriba dicha palabra de manera que cada letra se encuentre codificada sustituyéndola por aquel carácter que le sigue en la tabla de código ASCII.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca una palabra de 4 letras: SET0
La palabra [SET0] transformada es [TFUP]
```

Otro ejemplo:

```
Introduzca una palabra de 4 letras: hola
La palabra [hola] transformada es [ipmb]
```

Ejercicio 6 (p1e6.cpp)

Desarrolle un programa que lea de teclado un número entero, que representa una cierta cantidad de Bytes, y muestre por pantalla los MiBytes, KiBytes y Bytes que podemos obtener.

Considerando que 1KiByte es equivalente a 1024 Bytes, y 1 MiByte es equivalente a 1024 KiBytes.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo, dado el número 26871979, el resultado sería 25 MiBytes, 642 KiBytes y 171 Bytes, ya que $26871979 \text{ Bytes} = 25 \text{ MiBytes} + 642 \text{ KiBytes} + 171 \text{ Bytes}$.

```
Introduzca una cantidad de Bytes: 26871979
26871979 Bytes corresponden a:
Mibytes = 25
```

```
Kibytes = 642
Bytes   = 171
```

Ejercicio 7 (p1e7.cpp)

El siguiente programa escrito en C++ calcula el área y la longitud de una circunferencia. Al compilarlo se producen una serie de errores que el alumno debe localizar y corregir.

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

const int PI=3.1416

int main()
{
    double longitud, area;
    int radio;

    out << "Hola" ; << endl;
    cout<< "Este programa calcula la longitud y el área de un círculo"

    cin << "Introduce el radio del círculo: " >> radio;

    long = 2*PI*radio;
    area = PI*(radio*radio)
    cout << 'Area = ' << area << endl;
    cout << 'Longitud = ' << area << endl;
}
```

Una vez compilado correctamente, la ejecución de este programa para una circunferencia de radio 30 produce el siguiente resultado:

```
Hola
Este programa calcula la longitud y el área de un círculo
Introduce el radio del círculo: 30
Area = 2827.44
Longitud = 188.496
```

Ejercicio 8 (p1e8.cpp)

Desarrolle un programa que lea de teclado una palabra de cuatro letras minúsculas (supondremos que la entrada de datos es correcta, se deben leer cuatro caracteres y almacenarlos en cuatro variables de tipo `char`), y posteriormente escriba las letras mayúsculas correspondientes a las letras minúsculas leídas previamente.

Tenga presente que no es necesario conocer ni consultar la tabla *ASCII* de codificación de los caracteres. Sólo es necesario conocer que las letras minúsculas (desde la 'a' hasta la 'z') están consecutivas, y que las letras mayúsculas (desde la 'A' hasta la 'Z') están consecutivas.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca una palabra de 4 letras minúsculas: hola
La palabra [hola] transformada es [HOLA]
```

Otro ejemplo:

```
Introduzca una palabra de 4 letras minúsculas: seto
La palabra [seto] transformada es [SET0]
```

Ejercicio 9 (p1e9.cpp)

Desarrolle un programa que lea de teclado una cierta cantidad de segundos y muestre su equivalente en semanas, días, horas, minutos y segundos, según el formato de los siguientes ejemplos. Nótese los

espacios dentro de los corchetes que delimitan el número de semanas, y los ceros al mostrar las horas, minutos y segundos.

Para respetar el formato especificado, se recomienda utilizar los manipuladores de entrada/salida `setw` y `setfill` (puede consultar la explicación de dichos manipuladores en el apartado de *Salida formateada de datos*, en la sección de *Entrada y salida de datos avanzada*, en los apuntes del Tema 2).

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca los segundos: 2178585
2178585 segundos equivalen a [ 3] semanas, 4 dias 05:09:45
```

Otro ejemplo:

```
Introduzca los segundos: 9127145
9127145 segundos equivalen a [ 15] semanas, 0 dias 15:19:05
```

Ejercicio 10 (p1e10.cpp)

Desarrolle un programa que calcule la nota final de una asignatura. Para ello deberá leer por teclado la nota de la parte de teoría y la nota de la parte de problemas, y habrá de calcular la nota final considerando que la parte de teoría vale un *70 %* de la nota final y la de práctica un *30 %*.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca la nota de teoria: 10
Introduzca la nota de practicas: 10
La calificacion es: 10
```

Otro ejemplo:

```
Introduzca la nota de teoria: 8
Introduzca la nota de practicas: 5.5
La calificacion es: 7.25
```

Otro ejemplo:

```
Introduzca la nota de teoria: 4
Introduzca la nota de practicas: 6
La calificacion es: 4.6
```

Ejercicio 11 (p1e11.cpp)

Codifique el siguiente código, ejecútelo y descubra qué hace este programa y cómo lo hace. Escriba la explicación en un comentario en su código, y posteriormente compruebe la solución proporcionada por el profesor.

Téngase en cuenta que el comentario de explicación proporcionado por el alumno **no puede ser corregido de forma automática**, por lo que el alumno debe comprobar su propia explicación con respecto a la explicación proporcionada por el profesor, cuando ésta sea publicada.

```
#include <iostream>
using namespace std;
int main()
{
    int a = 6;
    int b = 14;
    int auxiliar;
    cout << "a vale " << a << " y b vale " << b << endl;

    // ¿Qué hacen estas tres sentencias?
    auxiliar = a;
    a = b;
    b = auxiliar;
```

```
    cout << "a vale " << a << " y b vale " << b << endl;
}
```

Problemas Derivados de la Representación Computacional de los Números

Ejercicio 12 (p1e12.cpp)

Codifique el siguiente programa, ejecútelo varias veces introduciendo como datos de entrada los números mostrados en la siguiente tabla, analice los resultados obtenidos, escriba la explicación sobre los comportamientos de las ejecuciones en un comentario en su código, y posteriormente compruebe la explicación proporcionada por el profesor.

Téngase en cuenta que el comentario de explicación proporcionado por el alumno **no puede ser corregido de forma automática**, por lo que el alumno debe comprobar su propia explicación con respecto a la explicación proporcionada por el profesor, cuando ésta sea publicada.

Puede consultar una explicación en la sección dedicada a los *problemas derivados de la implementación de los tipos numéricos*, en el apartado de *desbordamiento (overflow)*, en los apuntes del *Tema 2*.

a)	-20	y	30	c)	147483647	y	2000000000	e)	1	y	2147483647
b)	20	y	-30	d)	200000000	y	2000000000	f)	1	y	3000000000

```
#include <iostream>
using namespace std;
int main()
{
    int num1, num2;
    cout << "Introduzca el primer número entero: ";
    cin >> num1;
    cout << "Introduzca el segundo número entero: ";
    cin >> num2;
    int suma = num1 + num2;
    cout << "Primer número: " << num1 << endl;
    cout << "Segundo número: " << num2 << endl;
    cout << "Resultado (num1 + num2): " << suma << endl;
}
```

Ejercicio 13 (p1e13.cpp)

Codifique el siguiente programa, ejecútelo, analice el resultado obtenido, escriba la explicación sobre el comportamiento de la ejecución en un comentario en su código, y posteriormente compruebe la explicación proporcionada por el profesor.

Téngase en cuenta que el comentario de explicación proporcionado por el alumno **no puede ser corregido de forma automática**, por lo que el alumno debe comprobar su propia explicación con respecto a la explicación proporcionada por el profesor, cuando ésta sea publicada.

Puede consultar una explicación en la sección dedicada a los *problemas derivados de la implementación de los tipos numéricos*, en el apartado de la *perdida de precisión* en las operaciones con números reales, en los apuntes del *Tema 2*.

```
#include <iostream>
using namespace std;
int main()
{
    bool ok = (3.0 * (0.1 / 3.0)) == ((3.0 * 0.1) / 3.0);
    cout << "Resultado de (3.0 * (0.1 / 3.0)) == ((3.0 * 0.1) / 3.0): "
        << boolalpha << ok << " -> ERROR" << endl;
}
```

Ejercicio 14 (p1e14.cpp)

Codifique el siguiente programa, ejecútelo, analice el resultado obtenido, escriba la explicación sobre el comportamiento de la ejecución en un comentario en su código, y posteriormente compruebe la explicación proporcionada por el profesor.

Téngase en cuenta que el comentario de explicación proporcionado por el alumno **no puede ser corregido de forma automática**, por lo que el alumno debe comprobar su propia explicación con respecto a la explicación proporcionada por el profesor, cuando ésta sea publicada.

```
#include <iostream>
using namespace std;
int main()
{
    int num11 = -7;
    int num12 = 4;
    double num13 = num11 + num12;
    cout << "Valor de número11 (int):" << num11 << endl;
    cout << "Valor de número12 (int):" << num12 << endl;
    cout << "Valor de número13 (double) (num11 + num12): " << num13 << " CORRECTO" << endl;
    //-----
    int num21 = -7;
    unsigned num22 = 4;
    double num23 = num21 + num22;
    cout << "Valor de número21 (int):" << num21 << endl;
    cout << "Valor de número22 (unsigned):" << num22 << endl;
    cout << "Valor de número23 (double) (num21 + num22): " << num23 << " ERROR" << endl;
}
```

Ejercicio 15 (p1e15.cpp)

Codifique el siguiente programa, ejecútelo, analice el resultado obtenido, escriba la explicación sobre el comportamiento de la ejecución en un comentario en su código, y posteriormente compruebe la explicación proporcionada por el profesor.

Téngase en cuenta que el comentario de explicación proporcionado por el alumno **no puede ser corregido de forma automática**, por lo que el alumno debe comprobar su propia explicación con respecto a la explicación proporcionada por el profesor, cuando ésta sea publicada.

Nótese que es necesario **desactivar** la opción de compilación **-Werror** para que este programa pueda compilar correctamente, ya que el compilador *avisará* del *riesgo* de comparar un valor **int** con un valor **unsigned**, y en caso de que la opción **-Werror** estuviese activada, no superaría el proceso de compilación, ya que sería considerado como un error de compilación.

No olvide volver a **activar** la opción de compilación **-Werror** cuando haya terminado de realizar este ejercicio.

```
#include <iostream>
using namespace std;
int main()
{
    int num11 = -7;
    int num12 = 4;
    cout << "Valor de número11 (int):" << num11 << endl;
    cout << "Valor de número12 (int):" << num12 << endl;
    if (num11 < num12) {
        cout << "El valor " << num11 << " es menor que el valor " << num12
            << " CORRECTO" << endl;
    } else {
        cout << "El valor " << num11 << " NO es menor que el valor " << num12
            << " ERROR" << endl;
    }
    //-----
    int num21 = -7;
    unsigned num22 = 4;
    cout << "Valor de número21 (int):" << num21 << endl;
    cout << "Valor de número22 (unsigned):" << num22 << endl;
    if (num21 < num22) {
```



```

        cout << "El valor " << num21 << " es menor que el valor " << num22
            << " CORRECTO" << endl;
    } else {
        cout << "El valor " << num21 << " NO es menor que el valor " << num22
            << " ERROR" << endl;
    }
}

```

Ejercicio 16 (p1e16.cpp)

Codifique el siguiente programa, ejecútelo, analice el resultado obtenido, escriba la explicación sobre el comportamiento de la ejecución en un comentario en su código, y posteriormente compruebe la explicación proporcionada por el profesor.

Téngase en cuenta que el comentario de explicación proporcionado por el alumno **no puede ser corregido de forma automática**, por lo que el alumno debe comprobar su propia explicación con respecto a la explicación proporcionada por el profesor, cuando ésta sea publicada.

```

#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    //-----
    cout << setprecision(9);
    //-----
    int num11 = 9527;
    int num12 = 15937;
    double res1 = num11 * num12;
    cout << "Valor de número11 (int):" << num11 << endl;
    cout << "Valor de número12 (int):" << num12 << endl;
    cout << "Resultado (double) (num11 * num12):" << res1 << " CORRECTO" << endl;
    //-----
    int num21 = 9527;
    int num22 = 15937;
    float res2 = num21 * num22;
    cout << "Valor de número21 (int):" << num21 << endl;
    cout << "Valor de número22 (int):" << num22 << endl;
    cout << "Resultado (float) (num21 * num22):" << res2 << " ERROR" << endl;
    //-----
}

```

Ejercicio 17 (p1e17.cpp)

Codifique el siguiente programa, ejecútelo, analice el resultado obtenido, escriba la explicación sobre el comportamiento de la ejecución en un comentario en su código, y posteriormente compruebe la explicación proporcionada por el profesor.

Téngase en cuenta que el comentario de explicación proporcionado por el alumno **no puede ser corregido de forma automática**, por lo que el alumno debe comprobar su propia explicación con respecto a la explicación proporcionada por el profesor, cuando ésta sea publicada.

```

#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    //-----
    cout << setprecision(9);
    //-----
    int num11 = 9527;
    int num12 = 15937;
    double num13 = 1.0;
    int res1i = num11 * num12 * num13;
    double res1d = num11 * num12 * num13;
    cout << "Valor de número11 (int):" << num11 << endl;
    cout << "Valor de número12 (int):" << num12 << endl;
    cout << "Valor de número13 (double):" << num13 << endl;
    cout << "Resultado (int) (num11 * num12 * num13):" << res1i << " CORRECTO" << endl;
}

```

```

cout << "Resultado (double) (num11 * num12 * num13): " << res1d << " CORRECTO" << endl;
//-----
int num21 = 9527;
int num22 = 15937;
float num23 = 1.0;
int res2i = num21 * num22 * num23;
double res2d = num21 * num22 * num23;
cout << "Valor de número21 (int):" << num21 << endl;
cout << "Valor de número22 (int):" << num22 << endl;
cout << "Valor de número23 (float):" << num23 << endl;
cout << "Resultado (int) (num21 * num22 * num23):" << res2i << " ERROR" << endl;
cout << "Resultado (double) (num21 * num22 * num23): " << res2d << " ERROR" << endl;
//-----
}

```

Fundamentos de la Programación. Práctica de Laboratorio 2

E.T.S. Ingeniería Informática. Universidad de Málaga.

Tema 2: Estructuras de Control: Sentencias de Selección

La metodología de trabajo durante la realización de las prácticas en el laboratorio será la siguiente para cada ejercicio:

- En la carpeta **Documentos**, cree una carpeta con el *nombre-del-alumno*, y en esta carpeta, desarrolle los programas con los nombres especificados para cada ejercicio de la práctica.
- Durante la práctica en el laboratorio, el alumno debe dedicar un **máximo de 15 minutos** para **desarrollar** la solución del problema, corregir los errores de compilación, ejecutar el programa y depurar los errores que encuentre.
- Aquellos ejercicios que el alumno no tenga tiempo de terminar durante la práctica en el laboratorio, se deberán terminar en casa, y entregar todos los ejercicios dentro del plazo especificado para la tarea correspondiente.
- Posteriormente, cuando se publique la solución de los ejercicios, el alumno debe **analizar** la solución proporcionada por el profesor en el campus virtual y compararla con su propia solución.
- Los ejercicios serán **corregidos** comparando la salida que produce la ejecución del programa con respecto a la salida especificada en el enunciado de cada ejercicio, por ello, cada programa debe mostrar una interacción con el usuario **exactamente** como se muestra en el enunciado de cada ejercicio.
- El nombre del fichero de cada ejercicio debe ser como se especifica en el enunciado, por ejemplo, el primer ejercicio de la práctica 2 se debe denominar **p2e1.cpp**, y así sucesivamente.

Criterios de codificación:

- El uso de **variables globales** está **prohibido**. Sólo se pueden utilizar variables locales, **declaradas dentro de main**.

Ejercicios de Laboratorio

Ejercicio 1 (p2e1.cpp)

Desarrolla un programa que lea un número entero desde teclado y muestre en pantalla un mensaje indicando si el número leído es negativo o no.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. por ejemplo:

```
Introduzca un número entero: 123
El número 123 no es negativo
```

Otro ejemplo:

```
Introduzca un número entero: -56
El número -56 sí es negativo
```

Ejercicio 2 (p2e2.cpp)

Desarrolle un programa que lea tres números enteros y muestre en pantalla cuál de ellos es el **mayor estricto** (único), o una indicación de que no existe.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca tres números enteros: 34 56 25
El numero mayor es: 56
```

Otro ejemplo:

```
Introduzca tres números enteros: 45 23 23
El numero mayor es: 45
```

Otro ejemplo:

```
Introduzca tres números enteros: 78 43 78
No existe un único número mayor
```

Ejercicio 3 (p2e3.cpp)

Desarrolle un programa que lea un carácter del teclado y compruebe si el carácter es una letra (mayúscula o minúscula), en cuyo caso la salida debe ser “Es letra”, o si el carácter es un punto (‘.’), en cuyo caso la salida debe ser “Es punto”. Si el carácter no es ni una letra ni un punto la salida debe ser “Error”.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca un carácter: m
Es letra
```

Otro ejemplo:

```
Introduzca un carácter: X
Es letra
```

Otro ejemplo:

```
Introduzca un carácter: .
Es punto
```

Otro ejemplo:

```
Introduzca un carácter: 2
Error
```

Ejercicio 4 (p2e4.cpp)

Desarrolle un programa que acepte fechas escritas en el formato numérico (día, mes y año) y muestre la misma fecha pero con el mes correspondiente indicado en letras. Si el número de mes es erróneo, entonces mostrará “Error” como nombre del mes. Utiliza la estructura de selección **switch**.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el dia: 15
Introduzca el mes: 2
Introduzca el año: 1978
Dia: 15
Mes: Febrero
Año: 1978
```

Otro ejemplo:

```
Introduzca el dia: 17
Introduzca el mes: 20
Introduzca el año: 1982
Dia: 17
Mes: Error
Año: 1982
```

Ejercicio 5 (p2e5.cpp)

Desarrolle un programa que permita emitir la factura correspondiente a una compra de un artículo determinado del que se adquieren una o varias unidades. El número de unidades y el precio por unidad se introducen por teclado. El IVA a aplicar es del 12 %. Además si el precio total (precio de las unidades + IVA) es mayor de 300 €, se aplicará un descuento del 5 %. El programa mostrará por pantalla el precio total final. En el caso de que se aplique el descuento, deberemos indicarlo también por pantalla .

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca la cantidad de unidades adquiridas: 10
Introduzca el precio de una unidad: 20.5
El precio total a pagar es: 229.6 €
```

Otro ejemplo:

```
Introduzca la cantidad de unidades adquiridas: 10
Introduzca el precio de una unidad: 27.0
Se aplica descuento del 5%
El precio total a pagar es: 287.28 €
```

Ejercicio 6 (p2e6.cpp)

El recibo de la electricidad se elabora de la siguiente forma para una determinada cantidad de Kwh consumidos:

- 1€ de gastos fijos.
- 0.50€/Kwh para los primeros 100 Kwh consumidos.
- 0.35€/Kwh para los siguientes 150 Kwh consumidos.
- 0.25€/Kwh para el resto de Kwh consumidos.

Desarrolle un programa que lea de teclado un número que representa el consumo en Kwh, calcule y muestre en pantalla el importe total a pagar en función del consumo realizado.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el consumo del contador: 325
Consumo: 325 Kwh. Importe: 122.25 €
```

Pruebe la ejecución del programa con los valores de la siguiente tabla:

Consumo (Kwh)	Importe (€)
78	40
132	62.2
234	97.9
273	109.25
325	122.25

Ejercicio 7 (p2e7.cpp)

Diseña un programa que lea el número referente a un mes (desde 1 hasta 12), y calcule el número de días que tiene dicho mes para un año no bisiesto, sabiendo que: enero, marzo, mayo, julio, agosto, octubre y diciembre tienen 31 días, febrero 28 y el resto de los meses 30. Si el número de mes es erróneo, entonces mostrará un mensaje de error. Utiliza la estructura de selección **switch**.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca número de mes (de 1 hasta 12): 6
```

Ese mes tiene 30 días

Otro ejemplo:

```
Introduzca número de mes (de 1 hasta 12): 20
Mes incorrecto
```

Pruebe la ejecución del programa para todos los meses, desde el 1 hasta el 12, y compruebe si produce los resultados esperados.

Ejercicio 8 (p2e8.cpp)

Una empresa maneja códigos numéricos con las siguientes características:

- Cada código consta de cuatro dígitos.
- El primer dígito representa a una provincia.
- Los dos siguientes dígitos indican el número de la operación.
- El último dígito es un dígito de control.

Se desea desarrollar un programa que lea de teclado un número entero de cuatro dígitos (el código de provincia es distinto de cero), lo almacene en una variable de tipo entero (`int`), y realice las siguientes acciones:

- Si el código numérico no tiene 4 dígitos, entonces escribe un mensaje de error.
- Si el código numérico tiene 4 dígitos, entonces:
 - Muestra en pantalla la información (provincia, operación y dígito de control) desglosada.
 - Calcula si el código es correcto comprobando si el valor del dígito de control es igual al **resto de dividir** entre 10 el resultado de multiplicar el número de operación por el código de la provincia, y mostrará un mensaje adecuado.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

- Por ejemplo, para el número 32:

```
Introduzca el código numérico de 4 dígitos: 32
Código erróneo
```

- Por ejemplo, para el número 7362:

```
Introduzca el código numérico de 4 dígitos: 7362
Provincia: 7
Número de operación: 36
Dígito de control: 2
Comprobación: correcto
```

- Por ejemplo, para el número 6257:

```
Introduzca el código numérico de 4 dígitos: 6257
Provincia: 6
Número de operación: 25
Dígito de control: 7
Comprobación: error
```

Fundamentos de la Programación. Práctica de Laboratorio 3

E.T.S. Ingeniería Informática. Universidad de Málaga.

Tema 2: Estructuras de Control: Sentencias de Iteración

La metodología de trabajo durante la realización de las prácticas en el laboratorio será la siguiente para cada ejercicio:

- En la carpeta **Documentos**, cree una carpeta con el *nombre-del-alumno*, y en esta carpeta, desarrolle los programas con los nombres especificados para cada ejercicio de la práctica.
- Durante la práctica en el laboratorio, el alumno debe dedicar un **máximo de 20 minutos** para **desarrollar** la solución del problema, corregir los errores de compilación, ejecutar el programa y depurar los errores que encuentre.
- Aquellos ejercicios que el alumno no tenga tiempo de terminar durante la práctica en el laboratorio, se deberán terminar en casa, y entregar todos los ejercicios dentro del plazo especificado para la tarea correspondiente.
- Posteriormente, cuando se publique la solución de los ejercicios, el alumno debe **analizar** la solución proporcionada por el profesor en el campus virtual y compararla con su propia solución.
- Los ejercicios serán **corregidos** comparando la salida que produce la ejecución del programa con respecto a la salida especificada en el enunciado de cada ejercicio, por ello, cada programa debe mostrar una interacción con el usuario **exactamente** como se muestra en el enunciado de cada ejercicio.
- El nombre del fichero de cada ejercicio debe ser como se especifica en el enunciado, por ejemplo, el primer ejercicio de la práctica 3 tiene tres versiones diferentes, y se deben denominar **p3e1a.cpp**, **p3e1b.cpp** y **p3e1c.cpp** respectivamente. El segundo ejercicio de la práctica 3 se debe denominar **p3e2.cpp**, y así sucesivamente.

Criterios de codificación:

- El uso de **variables globales** está **prohibido**. Sólo se pueden utilizar variables locales, **declaradas dentro de main**.
- No se puede **modificar la variable de control** de un bucle **for** dentro de su cuerpo. El número máximo de iteraciones que realiza el bucle for debe quedar claramente especificado en la cabecera del mismo.
- Sólo están permitidas las **sentencias** y estructuras de selección e iteración que aparecen en los **apuntes** y que serán vistas en clase (**if**, **switch**, **while**, **do-while**, **for**, etc.). No se podrán usar otras como **goto**, **continue**, etc. La sentencia **break** sólo se podrá utilizar dentro de la estructura **switch**.

Ejercicios de Laboratorio

Ejercicio 1 (p3e1a.cpp, p3e1b.cpp, p3e1c.cpp)

Desarrolle un programa que lea de teclado el valor de un número N, y si el número leído de teclado es menor que cero, entonces mostrará el mensaje de error ("**Error**") y finalizará el programa adecuadamente, sin abortar ni lanzar excepciones.

En otro caso, el programa debe calcular la **suma** de los N primeros números enteros positivos, y mostrar en pantalla el resultado de dicha suma (en caso de que el valor del número N sea cero, se mostrará cero como resultado de la suma).

Aunque es más adecuado utilizar la estructura iterativa **for** para realizar este proceso iterativo de sumar de forma acumulativa los N primeros números enteros positivos, con el propósito del aprendizaje

de las estructuras iterativas, en este ejercicio se deben desarrollar tres versiones diferentes de este programa:

1. En la primera versión (denominada **p3e1a.cpp**), el proceso iterativo de la suma de los N primeros números enteros positivos se debe realizar con la sentencia de iteración **for**.
2. En la segunda versión (denominada **p3e1b.cpp**), el proceso iterativo de la suma de los N primeros números enteros positivos se debe realizar con la sentencia de iteración **while**.
3. En la tercera versión (denominada **p3e1c.cpp**), el proceso iterativo de la suma de los N primeros números enteros positivos se debe realizar con la sentencia de iteración **do-while**.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo, para el número 10, mostrará un resultado de 55, ya que $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55$.

- Ejemplo para N igual a 10:

```
Introduzca un número: 10
La suma es: 55
```

- Ejemplo para N igual a cero:

```
Introduzca un número: 0
La suma es: 0
```

- Ejemplo para N negativo:

```
Introduzca un número: -5
Error.
```

Ejercicio 2 (p3e2.cpp)

En una fábrica de coches se desea calcular el precio medio de un número de modelos de coche. Para ello, se debe leer de teclado la cantidad de precios de modelos que serán introducidos desde el teclado, y posteriormente, para cada modelo se deberá introducir el precio (en euros) de cada modelo de coche (para esto usaremos una estructura iterativa) y posteriormente calcular el precio medio de los modelos (se recuerda que el precio medio de los modelos debe ser un número real).

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Ejemplo del programa en pantalla:

```
Introduzca número de modelos de coche: 4
Precio modelo 1: 13
Precio modelo 2: 22
Precio modelo 3: 18
Precio modelo 4: 28
El valor medio de los 4 modelos de coche asciende a: 20.25 €
```

Ejercicio 3 (p3e3.cpp)

Desarrolla un programa en C++ que lea de teclado un número entero, en caso de leer un número menor o igual a cero, entonces se volverá a solicitar la introducción de un nuevo número, iterativamente hasta que el número leído sea correcto. A continuación, deberá mostrar en pantalla una única línea con tantos caracteres 'x' como indique el número leído de teclado, según el modelo del siguiente ejemplo. Nótese que al final de la línea se debe mostrar un **salto de línea**.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

- Por ejemplo para un valor de 4:

```
Introduzca un número: 4
xxxx
```


- Por ejemplo para un valor de 5:

```
Introduzca un número: 5
xxxxx
```

Ejercicio 4 (p3e4.cpp)

Desarrolla un programa en C++ que lea de teclado un número entero, en caso de leer un número menor o igual a cero, entonces se volverá a solicitar la introducción de un nuevo número, iterativamente hasta que el número leído sea correcto. A continuación, deberá mostrar en pantalla un bloque con el carácter 'x' con tantas líneas y columnas como indique el número leído de teclado, según el modelo del siguiente ejemplo.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

- Por ejemplo para un valor de 4:

```
Introduzca un número: 4
xxxx
xxxx
xxxx
xxxx
```

- Por ejemplo para un valor de 5:

```
Introduzca un número: 5
xxxxx
xxxxx
xxxxx
xxxxx
xxxxx
```

Ejercicio 5 (p3e5.cpp)

Desarrolla un programa en C++ que lea de teclado un número entero, en caso de leer un número menor o igual a cero, entonces se volverá a solicitar la introducción de un nuevo número, iterativamente hasta que el número leído sea correcto. A continuación, deberá mostrar en pantalla un bloque con los caracteres 'x' y 'o' alternativos, según el modelo del siguiente ejemplo, con tantas líneas y columnas como indique el número leído de teclado.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

- Por ejemplo para un valor de 4:

```
Introduzca un número: 4
xoxo
oxox
xoxo
oxox
```

- Por ejemplo para un valor de 5:

```
Introduzca un número: 5
xoxox
oxoxo
xoxox
oxoxo
xoxox
```

Ejercicio 6 (p3e6.cpp)

Desarrolla un programa que lea una secuencia de caracteres terminada en un punto ('.'), donde el punto no forma parte de la secuencia de caracteres a procesar, y muestre en pantalla la posición en la tabla ASCII asociada a cada uno de los caracteres leídos, incluyendo los espacios en blanco y saltos de línea, pero sin considerar al carácter punto ('.'). Posteriormente y antes de finalizar mostraremos por pantalla el número total de caracteres procesados.

Nótese que para poder leer los caracteres adecuadamente, incluyendo los espacios, se debe utilizar la sentencia `cin.get(c)`, y realizar el procesamiento secuencial con un *bucle de lectura adelantada*.

```
int main()
{
    char c;
    cout << "Introduzca el texto terminado en un punto:" << endl;
    cin.get(c);           // lectura del primer carácter
    while (c != '.') {
        // ...
        // procesar el carácter leído y almacenado en la variable c
        // ...
        cin.get(c);       // lectura del carácter a procesar en la siguiente iteración
    }
    // ...
}
```

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el texto terminado en un punto:
hola adios.
104 111 108 97 32 97 100 105 111 115
Número de caracteres leídos: 10
```

Ejercicio 7 (p3e7.cpp)

La constante matemática π puede ser calculada con la siguiente fórmula:

$$\pi = 2 \times \frac{2}{1} \times \frac{2}{3} \times \frac{4}{3} \times \frac{4}{5} \times \frac{6}{5} \times \frac{6}{7} \times \frac{8}{7} \times \frac{8}{9} \times \dots$$

Esta fórmula fue descubierta en el siglo XVII por un matemático inglés llamado J. Wallis. Desarrolla un programa que lea un valor entero, **n**, de forma iterativa hasta que **n** sea un número mayor que cero, y a continuación calcule π a partir de la fórmula anterior multiplicando las primeras **n** fracciones de la fórmula. Por ejemplo, si **n** tiene el valor **3**, entonces calculará:

$$\pi = 2 \times \frac{2}{1} \times \frac{2}{3} \times \frac{4}{3} = 3.55556$$

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el número de fracciones: 100
El valor de PI con 100 fracciones es: 3.12608
```

Otro ejemplo:

```
Introduzca el número de fracciones: -25
Error. Introduzca el número de fracciones: 1000
El valor de PI con 1000 fracciones es: 3.14002
```

Ejercicio 8 (p3e8.cpp)

Codifique un programa que se comporte como una calculadora iterativa. Para ello deberá tener las siguientes características:

- El calculador realizará operaciones aritméticas de forma iterativa hasta que se introduzca como código de operación el símbolo &.
- Solo efectuará operaciones con dos operandos (binarias).
- Operaciones permitidas: (+,-,*,/).
- Se trabajará con operandos enteros.
- Leerá en primer lugar la operación a realizar, y a continuación los dos operandos numéricos. Si el operador no se corresponde con alguno de los indicados se emitirá un mensaje de error, en otro caso mostrará el resultado de la operación, debiendo tener cuidado con el caso de división por cero.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Operación (+ - * / &): *
Operando 1: 13
Operando 2: 10
Resultado: 130
Operación (+ - * / &): u
ERROR: Operación no válida
Operación (+ - * / &): +
Operando 1: 12
Operando 2: 3
Resultado: 15
Operación (+ - * / &): &
FIN DEL PROGRAMA
```

Ejercicio 9 (p3e9.cpp)

Modifique el programa anterior para que en el caso de que el usuario introduzca una operación equivocada el sistema **aborte la ejecución elevando una excepción** “ERROR: Operación no válida”.

```
throw "ERROR: Operación no válida";
```

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Operación (+ - * / &): *
Operando 1: 13
Operando 2: 10
Resultado: 130
Operación (+ - * / &): u
terminate called after throwing an instance of 'char const*'
```

Fundamentos de la Programación. Práctica de Laboratorio 4

E.T.S. Ingeniería Informática. Universidad de Málaga.

Tema 3: Subprogramas: Procedimientos y Funciones (I)

La metodología de trabajo durante la realización de las prácticas en el laboratorio será la siguiente para cada ejercicio:

- En la carpeta **Documentos**, cree una carpeta con el *nombre-del-alumno*, y en esta carpeta, desarrolle los programas con los nombres especificados para cada ejercicio de la práctica.
- Durante la práctica en el laboratorio, el alumno debe dedicar un **máximo de 25 minutos** para **desarrollar** la solución del problema, corregir los errores de compilación, ejecutar el programa y depurar los errores que encuentre.
- Aquellos ejercicios que el alumno no tenga tiempo de terminar durante la práctica en el laboratorio, se deberán terminar en casa, y entregar todos los ejercicios dentro del plazo especificado para la tarea correspondiente.
- Posteriormente, cuando se publique la solución de los ejercicios, el alumno debe **analizar** la solución proporcionada por el profesor en el campus virtual y compararla con su propia solución.
- Los ejercicios serán **corregidos** comparando la salida que produce la ejecución del programa con respecto a la salida especificada en el enunciado de cada ejercicio, por ello, cada programa debe mostrar una interacción con el usuario **exactamente** como se muestra en el enunciado de cada ejercicio.
- El nombre del fichero de cada ejercicio debe ser como se especifica en el enunciado, por ejemplo, el primer ejercicio de la práctica 4 se debe denominar **p4e1.cpp**, y así sucesivamente.

Criterios de codificación (además de los indicados en las prácticas anteriores):

- El uso de **variables globales** está **prohibido**. Sólo se pueden utilizar parámetros y variables locales, **declaradas dentro de los subprogramas**.
- En una función, sólo se utilizará **una única sentencia return**, y será la última sentencia del cuerpo de dicha función. En un procedimiento, no se utilizará ninguna sentencia **return**.
- Todos los parámetros de entrada de tipos simples (**bool**, **char**, **int**, **double**, etc.) se realizarán mediante el **paso por valor**.
- Todos los parámetros de salida o entrada/salida se realizarán mediante el **paso por referencia**.

Ejercicios de Laboratorio

Ejercicio 1 (p4e1.cpp)

Desarrolle un programa que imprima una pirámide de dígitos como la de la figura, leyendo desde el teclado el número de filas de la misma (debe ser mayor o igual a cero y menor de 10). Por ejemplo, para 5 filas:

```
Introduzca el numero de filas (menor de 10): -4
Error. Introduzca el numero de filas (menor de 10): 20
Error. Introduzca el numero de filas (menor de 10): 5
-----
  1
 121
12321
1234321
123454321
-----
```

Nótese que tanto al principio como al final aparece una línea con tantos guiones (-) como la anchura de la base de la pirámide, y terminada con un *salto de línea* (`endl`).

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

Ejercicio 2 (p4e2.cpp)

Diseña un programa que calcule e imprima en pantalla los N primeros números primos, siendo N un número que se introduce por teclado.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca un número: 8
2, 3, 5, 7, 11, 13, 17, 19
```

Ejercicio 3 (p4e3.cpp)

Dos números a y b se dice que son amigos si son distintos, y la suma de los divisores de a (salvo él mismo) es igual a b y viceversa (la suma de los divisores de b (salvo él mismo) es igual a a). Por ejemplo los números 220 y 284 son amigos.

Desarrolla un programa que tenga como entrada de teclado dos números enteros a y b y que muestre en la pantalla un mensaje indicando si son amigos o no.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca dos numeros: 220 284
220 y 284 son amigos
```

Otro ejemplo:

```
Introduzca dos numeros: 100 150
100 y 150 no son amigos
```

Ejercicio 4 (p4e4.cpp)

Dos números a y b se dice que son amigos si son distintos, y la suma de los divisores de a (salvo él mismo) es igual a b y viceversa (la suma de los divisores de b (salvo él mismo) es igual a a). Por ejemplo los números 220 y 284 son amigos.

Desarrolla un programa que tenga como entrada de teclado dos números enteros n y m , mayores que cero, ($n < m$), y que muestre en la pantalla todas las parejas de números amigos que existan en el intervalo determinado por n y m . Las parejas de números amigos se mostrarán sin repetición, primero el número menor y después el mayor. En caso de intervalo incorrecto, el programa mostrará el mensaje de error ("Error") y finalizará el programa adecuadamente, sin abortar ni lanzar excepciones.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo en el intervalo desde 1 hasta 2000 sólo existen dos parejas de amigos: el 220 es amigo del 284, y el 1184 es amigo del 1210.

```
Introduzca un intervalo (dos números): 1 2000
Amigos: 220, 284
Amigos: 1184, 1210
```

Ejercicio 5 (p4e5.cpp)

Diseñe un programa que lea un número entero de teclado y escriba un rombo (relleno) con asteriscos (*), según el siguiente ejemplo para un número leído con valor 4:

```

Introduzca un número: 4
-----
      *
     * *
    * * *
   * * * *
  * * * *
 * * *
* *
*
-----

```

Nótese que tanto al principio como al final aparece una línea con tantos guiones (-) como la anchura del rombo, y terminada con un *salto de línea* (`endl`).

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

Ejercicio 6 (p4e6.cpp)

Diseña un programa que lea de teclado un número entero n mayor que cero y muestre las n primeras filas del siguiente triángulo. Por ejemplo:

```

Introduzca numero de filas: 11
-----
      1
     232
    34543
   4567654
  567898765
 67890109876
7890123210987
890123454321098
90123456765432109
0123456789876543210
123456789010987654321
-----

```

Nótese que tanto al principio como al final aparece una línea con tantos guiones (-) como la anchura de la base de la pirámide, y terminada con un *salto de línea* (`endl`).

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

Fundamentos de la Programación. Práctica de Laboratorio 5

E.T.S. Ingeniería Informática. Universidad de Málaga.

Tema 3: Subprogramas: Procedimientos y Funciones (II)

La metodología de trabajo durante la realización de las prácticas en el laboratorio será la siguiente para cada ejercicio:

- En la carpeta **Documentos**, cree una carpeta con el *nombre-del-alumno*, y en esta carpeta, desarrolle los programas con los nombres especificados para cada ejercicio de la práctica.
- Durante la práctica en el laboratorio, el alumno debe dedicar un **máximo de 25 minutos** para **desarrollar** la solución del problema, corregir los errores de compilación, ejecutar el programa y depurar los errores que encuentre.
- Aquellos ejercicios que el alumno no tenga tiempo de terminar durante la práctica en el laboratorio, se deberán terminar en casa, y entregar todos los ejercicios dentro del plazo especificado para la tarea correspondiente.
- Posteriormente, cuando se publique la solución de los ejercicios, el alumno debe **analizar** la solución proporcionada por el profesor en el campus virtual y compararla con su propia solución.
- Los ejercicios serán **corregidos** comparando la salida que produce la ejecución del programa con respecto a la salida especificada en el enunciado de cada ejercicio, por ello, cada programa debe mostrar una interacción con el usuario **exactamente** como se muestra en el enunciado de cada ejercicio.
- El nombre del fichero de cada ejercicio debe ser como se especifica en el enunciado, por ejemplo, el primer ejercicio de la práctica 5 se debe denominar **p5e1.cpp**, y así sucesivamente.

Criterios de codificación (además de los indicados en las prácticas anteriores):

- El uso de **variables globales** está **prohibido**. Sólo se pueden utilizar parámetros y variables locales, **declaradas dentro de los subprogramas**.
- En una función, sólo se utilizará **una única sentencia return**, y será la última sentencia del cuerpo de dicha función. En un procedimiento, no se utilizará ninguna sentencia **return**.
- Todos los parámetros de entrada de tipos simples (**bool**, **char**, **int**, **double**, etc.) se realizarán mediante el **paso por valor**.
- Todos los parámetros de salida o entrada/salida se realizarán mediante el **paso por referencia**.

Ejercicios de Laboratorio

Ejercicio 1 (p5e1.cpp)

Desarrolle un programa que calcule el valor de S para un número real X ($0 \leq X \leq 1$) dado por teclado, utilizando la serie de Taylor para calcular el valor de e^x :

$$S = 1 + X + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^4}{4!} + \dots$$

Nota: No se añadirán más sumandos cuando se añada uno con valor menor que 0.0001. No se puede utilizar la función *pow* ni las demás funciones de la biblioteca `<cmath>`.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el valor de X [0..1]: 0.3
Serie: 1.34986
```

Otro ejemplo:

```
Introduzca el valor de X [0..1]: 0.5
Serie: 1.64872
```

Otro ejemplo:

```
Introduzca el valor de X [0..1]: 0.9
Serie: 2.45959
```

Ejercicio 2 (p5e2.cpp)

Desarrolle un programa que lea un número **N** por teclado mayor o igual a cero y calcule e imprima el n -ésimo número de la serie de Fibonacci, comenzando la cuenta en cero. Los dos primeros números de esta serie son el cero y el uno, y a partir de éstos, cada número de la secuencia se calcula realizando la suma de los dos anteriores, es decir: $f_n = f_{n-1} + f_{n-2}$. Ej: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca un número: 8
fibonacci(8): 21
```

Otro ejemplo:

```
Introduzca un número: -5
Error. Introduzca un número: 20
fibonacci(20): 6765
```

Ejercicio 3 (p5e3.cpp)

El *máximo común divisor* de dos números enteros es el mayor número entero que es divisor de ambos números. Se puede calcular el *máximo común divisor* de dos números enteros positivos mediante el *algoritmo de Euclides*:

Dados dos números enteros positivos, si son distintos, entonces se realiza un proceso iterativo hasta que los dos números sean iguales, en el cual, en cada iteración, al número mayor se le restará el número menor. Finalmente, el *máximo común divisor* será igual al valor de ambos números (*que ahora son iguales*).

	M	N	
	24	18	
24-18 \Rightarrow	6	18	
	6	12	$\Leftarrow 18-6$
	6	6	$\Leftarrow 12-6$

Adicionalmente, dos números enteros son **coprimos** si su *máximo común divisor* es igual a **1**.

Desarrolla un programa que tenga como entrada de teclado dos números enteros n y m , mayores que cero, ($n < m$), y que muestre en la pantalla todas las parejas de números coprimos que existan en el intervalo determinado por n y m . Las parejas de números coprimos se mostrarán sin repetición, primero el número menor y después el mayor. En caso de intervalo incorrecto, el programa mostrará el mensaje de error ("**Error**") y finalizará el programa adecuadamente, sin abortar ni lanzar excepciones.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

```
Introduzca un intervalo (dos números): 10 15
Coprimos: 10, 11
Coprimos: 10, 13
Coprimos: 11, 12
Coprimos: 11, 13
```


Coprimos: 11, 14
Coprimos: 11, 15
Coprimos: 12, 13
Coprimos: 13, 14
Coprimos: 13, 15
Coprimos: 14, 15

Ejercicio 4 (p5e4.cpp)

Una secuencia de números enteros en zigzag es aquella en la que todos sus elementos son o bien estrictamente mayores o bien estrictamente menores que sus vecinos. Por ejemplo, la secuencia 4 2 3 1 5 3 es en zigzag, pero las secuencias 7 3 5 5 2 (el tercero es mayor que el segundo pero no es mayor que el cuarto) y 3 8 6 4 5 (el tercero es menor que el segundo pero es mayor que el cuarto). Cualquier secuencia de longitud 1 siempre es en zigzag, pero una secuencia de longitud cero no es en zigzag.

Desarrolle un programa que lea por teclado una secuencia de longitud indefinida de números, que acabará con un cero (considerando que el cero no pertenece a la secuencia) y muestre un mensaje diciendo si se trata de una secuencia en zigzag o no.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

- Por ejemplo:

```
Introduzca una secuencia de numeros terminada en cero:
4 2 3 1 5 3 0
La secuencia introducida SI es en zigzag
```

- Por ejemplo:

```
Introduzca una secuencia de numeros terminada en cero:
7 3 5 5 2 0
La secuencia introducida NO es en zigzag
```

- Por ejemplo:

```
Introduzca una secuencia de numeros terminada en cero:
3 8 6 4 5 0
La secuencia introducida NO es en zigzag
```

- Por ejemplo:

```
Introduzca una secuencia de numeros terminada en cero:
0
La secuencia introducida NO es en zigzag
```

- Por ejemplo:

```
Introduzca una secuencia de numeros terminada en cero:
2 0
La secuencia introducida SI es en zigzag
```

- Por ejemplo:

```
Introduzca una secuencia de numeros terminada en cero:
1 2 0
La secuencia introducida SI es en zigzag
```

- Por ejemplo:

```
Introduzca una secuencia de numeros terminada en cero:
2 1 0
La secuencia introducida SI es en zigzag
```

- Por ejemplo:

```

Introduzca una secuencia de numeros terminada en cero:
2 2 0
La secuencia introducida NO es en zigzag

```

Ejercicio 5 (p5e5.cpp)

Desarrolle un programa que lea de teclado el número de filas (mayor que cero y menor que diez, leerá de forma iterativa mientras el número leído sea incorrecto), y muestre en pantalla una figura como el siguiente ejemplo para un valor de 4.

```

Introduce el número de filas: -5
Error. Introduce el número de filas: 20
Error. Introduce el número de filas: 4
0 1 2 3
1 2 3 0
2 3 0 1
3 0 1 2

```

El programa deberá definir y utilizar el siguiente subprograma:

```

■ void incremento_circular(int& x, int max);

```

que recibe un primer parámetro de entrada/salida con un valor entero y lo modifica incrementándolo en uno. Sin embargo, si el resultado de incrementar el valor del primer parámetro es mayor o igual al valor del segundo parámetro, entonces el valor del primer parámetro se inicializa a cero.

Por ejemplo:

<i>max</i>	<i>x</i>		<i>nuevo valor de x</i>
4	0	→	1
4	1	→	2
4	2	→	3
4	3	→	0

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

Ejercicio 6 (p5e6.cpp)

Desarrolle un programa que calcule el valor de S para un número real X ($0 \leq X \leq 1$) dado por teclado, utilizando la serie de Taylor para calcular el valor de $\arcsin(x)$:

$$S = X + \frac{1}{2} \frac{X^3}{3} + \frac{1 \cdot 3}{2 \cdot 4} \frac{X^5}{5} + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6} \frac{X^7}{7} + \dots$$

Nota: No se añadirán más de 7 sumandos. No se puede utilizar la función *pow* ni las demás funciones de la biblioteca `<cmath>`.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```

Introduzca el valor de X [0..1]: 0.3
Serie: 0.304693

```

Otro ejemplo:

```

Introduzca el valor de X [0..1]: 0.5
Serie: 0.523598

```

Otro ejemplo:

Introduzca el valor de X [0..1]: 0.9
Serie: 1.11034

Ejercicio 7 (p5e7.cpp)

Dada una sucesión, de longitud indeterminada, de caracteres ceros y unos, construir un programa que permita calcular el tamaño de la mayor **subsucesión ordenada** de menor a mayor. La sucesión se lee desde el teclado, y el final viene dado por el carácter punto ('.'). Considere el concepto de *búffer de entrada*.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

- Por ejemplo:

Introduzca sucesión de ceros y unos hasta punto: 001001101.
Mayor subsucesión ordenada: 4

- Por ejemplo:

Introduzca sucesión de ceros y unos hasta punto: 0100101111.
Mayor subsucesión ordenada: 5

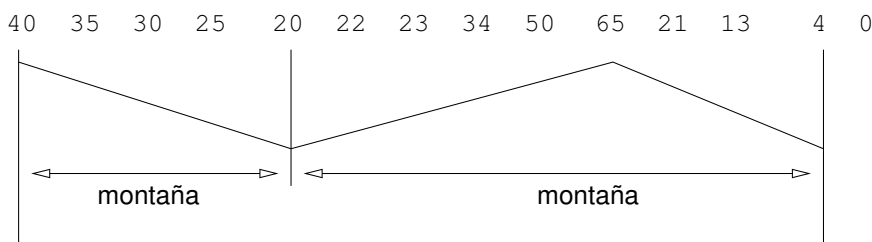
Ejercicio 8 (p5e8.cpp)

Decimos que una sucesión a_1, a_2, \dots, a_n de enteros forma una montaña, si existe un h tal que : $1 \leq h \leq n$ y además

$$a_1 < \dots a_{h-1} < a_h > a_{h+1} > \dots a_n$$

de tal forma que a_h es la cima de la montaña. Definimos la anchura de una montaña como la cuenta de números de enteros que la forman. Por ejemplo la sucesión $-7, -1, 6, 21, 15$ es una montaña de anchura 5.

Dada una secuencia de números enteros, separados, indistintamente, por espacios en blanco y/o saltos de línea, y terminada en cero (0), considerando que el cero (0) no forma parte de la secuencia. La secuencia de números contiene como mínimo una montaña (suponemos que la secuencia de enteros de entrada es una secuencia correcta de montañas), diseña un programa que calcule la anchura de la montaña más ancha.



Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

Introduzca sucesión de enteros hasta cero: 40 35 30 25 20 22 23 34 50 65 21 13 4 0
Mayor Montaña: 9

Ejercicio 9 (p5e9.cpp)

Decimos que una sucesión a_1, a_2, \dots, a_n de enteros forma una montaña, si existe un h tal que : $1 \leq h \leq n$ y además

$$a_1 < \dots a_{h-1} < a_h > a_{h+1} > \dots a_n$$

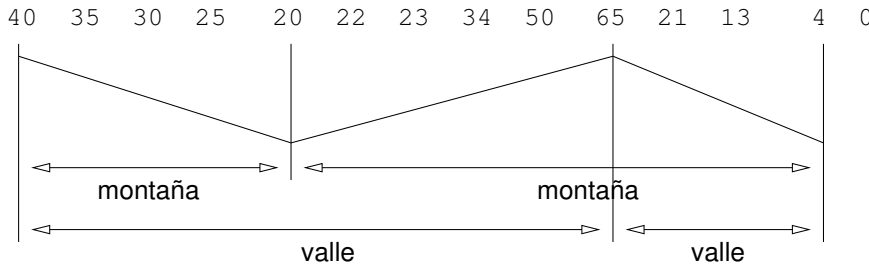
de tal forma que a_h es la cima de la montaña. Definimos la anchura de una montaña como la cuenta de números de enteros que la forman. Por ejemplo la sucesión $-7, -1, 6, 21, 15$ es una montaña de anchura 5.

Definimos un valle de la misma forma que una montaña pero cambiando el signo de las desigualdades de la definición anterior:

$$a_1 > \dots a_{h-1} > a_h < a_{h+1} < \dots a_n$$

de tal forma que a_h es el cauce del valle. Definimos la anchura de un valle como la cuenta de números de enteros que lo forman. Por ejemplo la sucesión $24, 13, 6, 15, 50$ es un valle de anchura 5.

Dada una secuencia de números enteros, separados, indistintamente, por espacios en blanco y/o saltos de línea, y terminada en cero (0), considerando que el cero (0) no forma parte de la secuencia. La secuencia de números contiene como mínimo una montaña y un valle (suponemos que la secuencia de enteros de entrada es una secuencia correcta de montañas y valles), diseña un programa que calcule la anchura de la montaña más ancha y la anchura del valle más ancho.



Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca sucesión de enteros hasta cero: 40 35 30 25 20 22 23 34 50 65 21 13 4 0
Mayor Montaña: 9
Mayor Valle: 10
```

Fundamentos de la Programación. Práctica de Laboratorio 6

E.T.S. Ingeniería Informática. Universidad de Málaga.

Tema 4: Tipos Estructurados: Registros y Arrays (I)

La metodología de trabajo durante la realización de las prácticas en el laboratorio será la siguiente para cada ejercicio:

- En la carpeta **Documentos**, cree una carpeta con el *nombre-del-alumno*, y en esta carpeta, desarrolle los programas con los nombres especificados para cada ejercicio de la práctica.
- Durante la práctica en el laboratorio, el alumno debe dedicar un **máximo de 35 minutos** para **desarrollar** la solución del problema, corregir los errores de compilación, ejecutar el programa y depurar los errores que encuentre.
- Aquellos ejercicios que el alumno no tenga tiempo de terminar durante la práctica en el laboratorio, se deberán terminar en casa, y entregar todos los ejercicios dentro del plazo especificado para la tarea correspondiente.
- Posteriormente, cuando se publique la solución de los ejercicios, el alumno debe **analizar** la solución proporcionada por el profesor en el campus virtual y compararla con su propia solución.
- Los ejercicios serán **corregidos** comparando la salida que produce la ejecución del programa con respecto a la salida especificada en el enunciado de cada ejercicio, por ello, cada programa debe mostrar una interacción con el usuario **exactamente** como se muestra en el enunciado de cada ejercicio.
- El nombre del fichero de cada ejercicio debe ser como se especifica en el enunciado, por ejemplo, el primer ejercicio de la práctica 6 se debe denominar **p6e1.cpp**, y así sucesivamente.

Criterios de codificación (además de los indicados en las prácticas anteriores):

- **No** está permitido utilizar los **arrays predefinidos**, se deben utilizar los **arrays** proporcionados por la **biblioteca <array> estándar de C++**.
- Todos los parámetros de entrada de tipos compuestos (**struct**, **array**, **string**) se realizarán mediante el **paso por referencia constante**.
- Todos los parámetros de salida o entrada/salida se realizarán mediante el **paso por referencia**.

Ejercicios de Laboratorio

Ejercicio 1 (p6e1.cpp)

Defina el tipo **Vector** como un array de 5 números **reales**. Diseñe una función para buscar el valor del mayor elemento de un array de números reales (de tamaño 5, aunque el programa deba funcionar adecuadamente para cualquier otro valor). Además, diseñe el programa y los subprogramas necesarios para probar adecuadamente su funcionamiento.

```
double buscar_mayor(const Vector& v);
```

Téngase en cuenta que la función debe funcionar adecuadamente incluso aunque todos los elementos sean negativos.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca 5 números: 5.7 2.0 7.5 3.8 4.1
El mayor elemento de la lista es 7.5
Lista: 5.7 2 7.5 3.8 4.1
```

Compruebe, así mismo, que los siguientes datos también producen los resultados esperados:

Datos					Mayor
5.7	2.0	7.5	3.8	4.1	7.5
7.5	2.0	5.7	3.8	4.1	7.5
5.7	2.0	4.1	3.8	7.5	7.5
-7	-4	-2	-5	-3	-2

Ejercicio 2 (p6e2.cpp)

Diseña una función **media** para calcular la media de las estaturas de una clase. La función recibe como parámetro de entrada una estructura de tipo **TEstaturas**, con las estaturas (números reales) de los alumnos de una determinada clase. Define dicho tipo de forma que pueda almacenar hasta un máximo de **MAX = 20** estaturas. El tipo debe definirse de forma que se pueda controlar en todo momento la cantidad de estaturas almacenadas, que podrá ser menor que el máximo especificado anteriormente. Crea después otras dos funciones para determinar cuántos alumnos son más altos y cuántos más bajos que la media. El nombre de las funciones queda a criterio del alumno.

Crea también una función **main** para comprobar que las tres funciones se han codificado correctamente.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Cuántas estaturas va a introducir (maximo 20): 5
Introduzca las 5 estaturas: 1.90 1.80 1.75 1.95 1.70
La media es: 1.82
Numero de alumnos más altos que la media: 2
Numero de alumnos más bajos que la media: 3
```

Ejercicio 3 (p6e3.cpp)

Desarrolle un programa que lea de teclado una secuencia de números entre 0 y 9 y cuente el número de veces que se repite cada dígito. La secuencia de números de entrada se da por finalizada al leer un número negativo.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca una secuencia de números (hasta negativo): 1 8 7 3 4 8 5 9 5 0 0 4 8 4 5 3 2 8 -1
La frecuencia de cada dígito es:
0: 2
1: 1
2: 1
3: 2
4: 3
5: 3
6: 0
7: 1
8: 4
9: 1
```

Ejercicio 4 (p6e4.cpp)

Un histograma es una gráfica que muestra la frecuencia con que aparecen en una lista dada los distintos valores que la pudieran formar. Por ejemplo, si los valores de una lista pueden estar comprendidos entre 0 y 9, y la lista está formada por:

```
6 4 4 1 9 7 5 6 4 2 3 9 5 6 4
```

entonces su histograma vertical será:

```

-----
      *
      *  *
      * * *
    * * * * * *
-----
0 1 2 3 4 5 6 7 8 9

```

Esto indica que el 0 y el 8 no aparecen ninguna vez, el 1, 2, 3 y 7 aparecen una vez, el 5 y 9 dos veces, etc.

Nótese que tanto al principio como al final aparece una línea con tantos guiones (-) como la anchura de la base del histograma, y terminada con un *salto de línea* (`endl`). Des mismo modo, la lista de números también está terminada con un *salto de línea* (`endl`).

Desarrolle un programa que lea una lista de números comprendidos entre 0 y 9 (la lista acabará cuando se lea un número negativo, y a priori no se puede determinar cuantos números contiene) e imprima por pantalla un histograma vertical como el anterior.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```

Introduzca una secuencia de números (hasta negativo): 6 4 4 1 9 7 5 6 4 2 3 9 5 6 4 -1
-----
      *
      *  *
      * * *
    * * * * * *
-----
0 1 2 3 4 5 6 7 8 9

```

Ejercicio 5 (p6e5.cpp)

Eratóstenes desarrolló un algoritmo para encontrar números primos, al que se conoce como “*La criba de Eratóstenes*”. Según este algoritmo, para encontrar todos los números primos menores que un número positivo **K** dado:

1. Se generan todos los números desde el 0 hasta el número **K-1**, y se almacenan en una lista.
2. Se *tachan* de esa lista los números cero (0) y uno (1).
3. Para cada número $i < \sqrt{K}$ (ó $i^2 < K$) que no haya sido *tachado* de la lista, se *tachan* de la lista todos aquellos números **j** que sean múltiplos de **i**. Por ejemplo:
 - a) Se *tachan* de la lista todos los números múltiplos de **2** a partir de él.
 - b) Se *tachan* de los que queden en la lista todos números los múltiplos de **3** a partir de él.
 - c) Así sucesivamente con **5**, **7** (y sucesivos primos que no han sido tachados) hasta que ya no se puedan *tachar* más números.
4. El proceso termina cuando $i \geq \sqrt{K}$ (ó $i^2 \geq K$).
5. Los números que queden en la lista sin *tachar* son los números primos menores que el **K** dado.

```

0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

```

Se debe crear un procedimiento denominado **eratostenes** que, mediante la criba descrita anteriormente y haciendo uso de arrays, debe tomar como parámetro un número **K** (menor o igual que una constante dada `MAXIMO = 100`) e imprimir por pantalla todos los números primos menores que **K**.

Diseñe un programa que lea por teclado un número **K** (menor o igual que un valor constante **MAXIMO** = 100), invoque al procedimiento anterior para mostrar todos los números primos menores que **K** mediante “la criba de Eratóstenes”.

Nota: para calcular la raíz cuadrada de un número se puede utilizar la función `double sqrt(double)` de la biblioteca `<cmath>`.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el valor de K (<= 100): 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Ejercicio 6 (p6e6.cpp)

Para realizar operaciones con números complejos, podemos definir el siguiente tipo:

```
struct Complejo {
    double real;
    double img;
};
```

Desarrolle subprogramas que realicen las operaciones de leer, mostrar, suma, resta, multiplicación y división de números complejos definidos con el tipo anterior, así como el programa principal para probar adecuadamente su funcionamiento.

```
void leer(Complejo& c);
void escribir(const Complejo& c);
void sumar(Complejo& res, const Complejo& c1, const Complejo& c2);
void restar(Complejo& res, const Complejo& c1, const Complejo& c2);
void multiplicar(Complejo& res, const Complejo& c1, const Complejo& c2);
void dividir(Complejo& res, const Complejo& c1, const Complejo& c2);
```

Nótese que, salvo leer y escribir, el resto de operaciones (sumar, restar, multiplicar y dividir) no muestran nada en pantalla, ni tampoco leen nada de teclado.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca un número complejo (real, img): 9 12
Introduzca un número complejo (real, img): 3 3
( 9, 12 ) + ( 3, 3 ) = ( 12, 15 )
( 9, 12 ) - ( 3, 3 ) = ( 6, 9 )
( 9, 12 ) * ( 3, 3 ) = ( -9, 63 )
( 9, 12 ) / ( 3, 3 ) = ( 3.5, 0.5 )
```

Ejercicio 7 (p6e7.cpp)

Vamos a trabajar con listas de números enteros de hasta un tamaño máximo de **MAX** = 10. Define el tipo de datos **TLista** para ello y diseña un procedimiento **criba** que recibe como parámetros de entrada una lista de números enteros **lista1** de tipo **TLista** y un número **x**. El procedimiento devolverá como parámetro de salida otra lista **lista2** de tipo **TLista** que contendrá sólo aquellos números de **lista1** que están repetidos **x** veces. En la lista **lista2** no habrá elementos repetidos.

Además, diseñe el programa principal y los subprogramas necesarios para probar adecuadamente su funcionamiento.

Para leer la lista de números **lista1**, se le pedirá primero al usuario el número de valores que va a introducir, controlando que este valor sea mayor que 0 y menor o igual que **MAX**.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:


```

Cuántos números desea introducir (máximo 10): 9
Introduzca 9 números: 1 3 4 3 1 3 0 -6 4
Introduzca el numero de repeticiones para realizar la criba: 2
La lista cribada es: 1 4

```

Ejercicio 8 (p6e8.cpp)

Consideremos un vector \mathbf{V} que contiene \mathbf{N} valores enteros (array de enteros de tamaño \mathbf{N} , siendo \mathbf{N} una constante con valor 5). Definimos el *centro* \mathbf{c} del vector \mathbf{V} como el índice entre $\mathbf{1}$ y $\mathbf{N-2}$ que verifica la siguiente propiedad:

$$\sum_{i=0}^{c-1} (c-i) \cdot V[i] = \sum_{j=c+1}^{n-1} (j-c) \cdot V[j]$$

Esta propiedad no siempre se verifica; en ese caso, decimos que el vector no tiene centro.

Diseña un procedimiento `centroVector` que reciba como parámetro un vector \mathbf{V} y nos devuelva dos valores como parámetros: el primero indica si existe o no el centro del vector, y el segundo, indica el índice en el que se encuentra el centro en caso de existir.

Además, diseñe el programa principal y los subprogramas necesarios para probar adecuadamente su funcionamiento.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

Por ejemplo, para el vector de tamaño 5 y elementos { 6, 2, 3, 0, 1 }, el centro de este vector es el índice 1 (casilla donde está el elemento 2), ya que al calcular los sumatorios:

- Sumatorio izquierda: $(1-0) \cdot V[0] = 1 \cdot 6 = 6$
- Sumatorio derecha: $(2-1) \cdot V[2] + (3-1) \cdot V[3] + (4-1) \cdot V[4] = 1 \cdot 3 + 2 \cdot 0 + 3 \cdot 1 = 6$

```

Introduzca 5 numeros enteros:
6 2 3 0 1
El centro de este vector es el indice 1

```

Por ejemplo, para el vector de tamaño 5 y elementos { 1, 2, 1, 1, 0 }, este vector no tiene centro.

```

Introduzca 5 numeros enteros:
1 2 1 1 0
Este vector no tiene centro

```

Fundamentos de la Programación. Práctica de Laboratorio 7

E.T.S. Ingeniería Informática. Universidad de Málaga.

Tema 4: Tipos Estructurados: Registros y Arrays (II)

La metodología de trabajo durante la realización de las prácticas en el laboratorio será la siguiente para cada ejercicio:

- En la carpeta **Documentos**, cree una carpeta con el *nombre-del-alumno*, y en esta carpeta, desarrolle los programas con los nombres especificados para cada ejercicio de la práctica.
- Durante la práctica en el laboratorio, el alumno debe dedicar un **máximo de 35 minutos** para **desarrollar** la solución del problema, corregir los errores de compilación, ejecutar el programa y depurar los errores que encuentre.
- Aquellos ejercicios que el alumno no tenga tiempo de terminar durante la práctica en el laboratorio, se deberán terminar en casa, y entregar todos los ejercicios dentro del plazo especificado para la tarea correspondiente.
- Posteriormente, cuando se publique la solución de los ejercicios, el alumno debe **analizar** la solución proporcionada por el profesor en el campus virtual y compararla con su propia solución.
- Los ejercicios serán **corregidos** comparando la salida que produce la ejecución del programa con respecto a la salida especificada en el enunciado de cada ejercicio, por ello, cada programa debe mostrar una interacción con el usuario **exactamente** como se muestra en el enunciado de cada ejercicio.
- El nombre del fichero de cada ejercicio debe ser como se especifica en el enunciado, por ejemplo, el primer ejercicio de la práctica 7 se debe denominar **p7e1.cpp**, y así sucesivamente.

Criterios de codificación (además de los indicados en las prácticas anteriores):

- No está permitido utilizar los **arrays predefinidos**, se deben utilizar los **arrays** proporcionados por la **biblioteca <array> estándar de C++**.
- Todos los parámetros de entrada de tipos compuestos (**struct**, **array**, **string**) se realizarán mediante el **paso por referencia constante**.
- Todos los parámetros de salida o entrada/salida se realizarán mediante el **paso por referencia**.

Ejercicios de Laboratorio

Ejercicio 1 (p7e1.cpp)

Desarrolle un subprograma que encuentre el elemento mayor de un array de dos dimensiones (de 5 filas y 7 columnas) de números enteros recibido como parámetro. Este subprograma devolverá como parámetros de salida tanto el valor mayor, como la fila y columna donde se encuentra. Si el elemento mayor aparece varias veces, se deja a criterio del alumno cuál seleccionar.

Téngase en cuenta que el subprograma debe funcionar adecuadamente incluso aunque todos los elementos sean negativos.

Además, desarrolle programa principal y los subprogramas necesarios para poder leer y mostrar los valores del array de dos dimensiones.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca 5 filas de 7 números
15  8  1 24 17 12 21
16 14  7  5 23 18 24
```

```

22 20 13 6 4 2 11
3 21 19 12 10 15 6
9 2 25 18 11 22 17

```

El número 25 es el mayor elemento de la matriz
Se encuentra en [4][2]

```

15 8 1 24 17 12 21
16 14 7 5 23 18 24
22 20 13 6 4 2 11
3 21 19 12 10 15 6
9 2 25 18 11 22 17

```

Compruebe, así mismo, que los siguientes datos también producen los resultados esperados:

Introduzca 5 filas de 7 números

```

-15 -18 -11 -24 -17 -12 -21
-16 -14 -17 -15 -23 -18 -24
-22 -20 -13 -16 -14 -12 -11
-13 -21 -19 -12 -10 -15 -16
-19 -12 -25 -18 -11 -22 -17

```

El número -10 es el mayor elemento de la matriz
Se encuentra en [3][4]

```

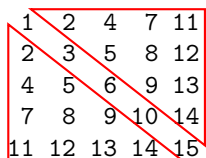
-15 -18 -11 -24 -17 -12 -21
-16 -14 -17 -15 -23 -18 -24
-22 -20 -13 -16 -14 -12 -11
-13 -21 -19 -12 -10 -15 -16
-19 -12 -25 -18 -11 -22 -17

```

Ejercicio 2 (p7e2.cpp)

Un array bidimensional **a** de M filas y M columnas, es simétrico si sus elementos satisfacen la condición $a[i][j]$ es igual a $a[j][i]$ para todo i, j . Desarrolle una función que determine si un array de ese tipo es simétrico, para un tamaño de M igual a 5. Además, diseñe el programa y los subprogramas necesarios para probar adecuadamente su funcionamiento.

Téngase en cuenta que, una vez que sepamos que una determinada matriz **no** es simétrica, los bucles deben terminar. Además, se debe evitar realizar las comprobaciones dobles, por ejemplo, si se ha comprobado la igualdad de los elementos $a[3][4]$ con $a[4][3]$, entonces no se debe volver a comprobar, de forma redundante, la igualdad de los elementos $a[4][3]$ con $a[3][4]$.



```

1 2 4 7 11
2 3 5 8 12
4 5 6 9 13
7 8 9 10 14
11 12 13 14 15

```

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

Introduzca 5 filas de 5 números

```

1 2 4 7 11
2 3 5 8 12
4 5 6 9 13
7 8 9 10 14
11 12 13 14 15

```

La matriz

```

1 2 4 7 11
2 3 5 8 12
4 5 6 9 13
7 8 9 10 14

```

```
11 12 13 14 15
SI es simétrica
```

Sin embargo:

```
Introduzca 5 filas de 5 números
1 2 4 7 20
2 3 5 8 12
4 5 6 9 13
7 8 9 10 14
11 12 13 14 15
```

```
La matriz
1 2 4 7 20
2 3 5 8 12
4 5 6 9 13
7 8 9 10 14
11 12 13 14 15
NO es simétrica
```

Ejercicio 3 (p7e3.cpp)

Un *cuadrado mágico* de orden n es una ordenación de los números desde el **1** hasta el n^2 en una tabla cuadrada ($n \times n$) de manera que la suma de cada fila y de cada columna y de las dos diagonales principales es la misma.

Por ejemplo, la figura muestra un cuadrado mágico de orden **5** en el que todas las filas, columnas y las dos diagonales suman **65**.

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

Defina el tipo **Cuadrado** como un array de dos dimensiones de 5 filas y 5 columnas de números enteros. Desarrolle la siguiente función, que devuelve **true** si el parámetro es un *cuadrado mágico*, y **false** en caso contrario.

```
bool esMagico(const Cuadrado& A);
```

Nótese que, además de comprobar que la suma de cada fila y de cada columna y de las dos diagonales principales es la misma, también debe comprobar que todos los números se encuentran en el rango de **1** hasta n^2 sin repetición.

Téngase en cuenta que, una vez que sepamos que una determinada matriz **no** es un cuadrado mágico, los bucles deben terminar.

Además, desarrolle el programa principal y los subprogramas necesarios para probar su funcionamiento. Así, el programa debe leer los valores de la matriz, y mostrarlos a continuación. Finalmente mostrará un mensaje indicando si el cuadrado es mágico o no lo es.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

- Por ejemplo:

```
Introduzca 5 filas de 5 numeros:
15 8 1 24 17
16 14 7 5 23
```

```

22  20  13   6   4
 3  21  19  12  10
 9   2  25  18  11

```

El cuadrado:

```

15   8   1  24  17
16  14   7   5  23
22  20  13   6   4
 3  21  19  12  10
 9   2  25  18  11

```

sí es mágico

- Por ejemplo:

Introduzca 5 filas de 5 numeros:

```

15   8   1  24  17
16  14   7   5  23
22  20  13   6   4
10  21  19  12   3
 9   2  25  18  11

```

El cuadrado:

```

15   8   1  24  17
16  14   7   5  23
22  20  13   6   4
10  21  19  12   3
 9   2  25  18  11

```

no es mágico

- Por ejemplo:

Introduzca 5 filas de 5 numeros:

```

15   8   1  24  17
 3  14   7   5  23
22  20  13   6   4
16  21  19  12  10
 9   2  25  18  11

```

El cuadrado:

```

15   8   1  24  17
 3  14   7   5  23
22  20  13   6   4
16  21  19  12  10
 9   2  25  18  11

```

no es mágico

- Por ejemplo:

Introduzca 5 filas de 5 numeros:

```

17   8   1  24  15
16  14   7   5  23
22  20  13   6   4
 3  21  19  12  10
11   2  25  18   9

```

El cuadrado:

```

17   8   1  24  15
16  14   7   5  23
22  20  13   6   4
 3  21  19  12  10

```

11 2 25 18 9

no es mágico

Ejercicio 4 (p7e4.cpp)

Desarrolle un programa que lea una sucesión de 10 números enteros, encuentre el valor máximo y lo imprima junto con el número de veces que aparece, y las posiciones en que este ocurre. El proceso se repite con el resto de la sucesión hasta que no quede ningún elemento por tratar.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca 10 números: 7 10 143 10 52 143 72 10 143 7
143 aparece 3 veces, en posiciones 3 6 9
72 aparece 1 vez, en posición 7
52 aparece 1 vez, en posición 5
10 aparece 3 veces, en posiciones 2 4 8
7 aparece 2 veces, en posiciones 1 10
```

Ejercicio 5 (p7e5.cpp)

Un *cuadrado mágico* de orden n es una ordenación de los números desde el 1 hasta el n^2 en una tabla cuadrada ($n \times n$) de manera que la suma de cada fila y de cada columna y de las dos diagonales principales es la misma.

La regla para generar el cuadrado mágico (si n es impar) del ejercicio anterior es la siguiente:

Se empieza poniendo un 1 en la casilla situada en la mitad de la primera fila. A partir de ese momento se inicia un proceso que visitará las diferentes casillas subiendo a la izquierda diagonalmente para ir poniendo en ellas los siguientes números de forma consecutiva hasta poner el número final N^2 . Si en este proceso se pasa el borde superior o izquierdo del cuadrado, se continúa por el extremo opuesto (ejemplos: en la figura los pasos marcados desde 1 a 2 y desde 3 a 4). Por otro lado, si en este proceso se alcanza una casilla ya rellena, se baja una posición desde el último valor depositado (ejemplo: en la figura el paso desde 5 a 6). Si en este proceso se pasa el borde inferior del cuadrado entonces se continúa por la primera fila del cuadrado.

Diseñe un subprograma que construya un cuadrado mágico para una constante N impar con valor 5. Construir un programa que muestre por pantalla el cuadrado mágico creado.

```
void construirMagico(Cuadrado& A);
```

Por ejemplo, la figura muestra un cuadrado mágico de orden 5 en el que todas las filas, columnas y las dos diagonales suman 65.

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

El cuadrado magico para $N = 5$ es:

```

15  8  1 24 17
16 14  7  5 23
22 20 13  6  4
 3 21 19 12 10
 9  2 25 18 11

```

Ejercicio 6 (p7e6.cpp)

Se desea crear una aplicación que permita asignar cargos electos a diversos partidos en unas elecciones, considerando el sistema conocido como ley de D'Hondt. El sistema de D'Hondt es una fórmula electoral, creada por Victor d'Hondt, que permite obtener el número de cargos electos asignados a las candidaturas, en proporción a los votos conseguidos.

El procedimiento consiste en lo siguiente: tras escutar todos los votos, calcular una serie de divisores para cada partido político. La fórmula de los divisores es V/N , donde V representa el número total de votos recibidos por el partido, y N representa cada uno de los números enteros de 1 hasta el número de cargos electos de la circunscripción objeto de escrutinio. Una vez realizadas las divisiones de los votos de cada candidatura por cada uno de los divisores desde 1 hasta N , la asignación de cargos electos se hace ordenando los cocientes de las divisiones de mayor a menor y asignando a cada uno un escaño hasta que éstos se agoten. Por ejemplo, para una distribución de votos como la siguiente:

	Partido A	Partido B	Partido C	Partido D	Partido E
Votos	340000	280000	160000	60000	15000

Y un número de cargos electos a distribuir de 7, los divisores para cada partido político serían los siguientes:

	1	2	3	4	5	6	7
A	$V_A/1=340000$	$V_A/2=170000$	$V_A/3=113333$	$V_A/4=85000$	$V_A/5=68000$	$V_A/6=56667$	$V_A/7=48571$
B	$V_B/1=280000$	$V_B/2=140000$	$V_B/3=93333$	$V_B/4=70000$	$V_B/5=56000$	$V_B/6=46667$	$V_B/7=40000$
C	$V_C/1=160000$	$V_C/2=80000$	$V_C/3=53333$	$V_C/4=40000$	$V_C/5=32000$	$V_C/6=26667$	$V_C/7=22857$
D	$V_D/1=60000$	$V_D/2=30000$	$V_D/3=20000$	$V_D/4=15000$	$V_D/5=12000$	$V_D/6=10000$	$V_D/7=8571$
E	$V_E/1=15000$	$V_E/2=7500$	$V_E/3=5000$	$V_E/4=3750$	$V_E/5=3000$	$V_E/6=2500$	$V_E/7=2143$

Las celdas marcadas se corresponden con los 7 cocientes más grandes, y por lo tanto el resultado sería el siguiente:

	Partido A	Partido B	Partido C
Cargos	3	3	1

No olvides aplicar Diseño Descendente a la hora de diseñar el programa. Podemos suponer que el máximo número de Cargos será 15 y el de Partidos 10. Podemos, así mismo, suponer que el nombre de los partidos será de una única letra, y así se podrá almacenar en un tipo `char` (en caso de que el nombre tuviese varias letras, se debería almacenar en un tipo `string`).

Implementa un programa que tenga una entrada y salida de datos como el siguiente ejemplo:

```

Introduzca el Numero de Cargos (>= 1 y <= 15): 7
Introduzca el Numero de Partidos (>= 1 y <= 10): 5
Introduzca el Nombre y Numero de Votos por Partido:
Partido 1: A 340000
Partido 2: B 280000
Partido 3: C 160000
Partido 4: D 60000
Partido 5: E 15000

```

```
Los Cargos Electos son:
A 3
B 3
C 1
```

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

Ejercicio 7 (p7e7.cpp)

El denominado juego de la vida fue ideado por J.H. Conway en 1969. Aunque parezca un juego, realmente es un modelo de una población de seres vivos idealizados que interactúan con su entorno. Los seres vivos se encuentran distribuidos en una matriz cuadrada de **TAM** (una constante, en nuestro caso con valor 5) filas y columnas que representa su mundo. En un instante dado, cada casilla de la matriz puede estar ocupada por un único ser vivo o vacía. La simulación comienza con una generación inicial. Cada nueva generación se obtiene a partir de la generación anterior atendiendo a las siguientes reglas:

- Si en una generación una determinada casilla está vacía, en la siguiente generación nacerá un ser vivo en la casilla correspondiente si el número de seres vivos vecinos (arriba, abajo, derecha, izquierda y diagonales) es igual a 3.
- Si en una generación una determinada casilla está ocupada por un ser vivo, en la siguiente generación ese ser vivo permanecerá en la casilla correspondiente si el número de seres vivos vecinos es igual a 2 o 3. En otro caso, ese ser vivo morirá, por lo que la casilla correspondiente en la siguiente generación permanecerá vacía.

Diseña un programa que realice las siguientes acciones:

- Primero debe leer de teclado el número de generaciones que se desean mostrar (incluida la inicial), considerando que si se introduce un valor menor o igual a cero, entonces mostrará el mensaje de error ("**Error**") y finalizará el programa adecuadamente, sin abortar ni lanzar excepciones.
- Después debe leer de teclado los datos para completar una matriz de tamaño *TAM x TAM* que constituye la generación inicial. El carácter 'o' representará una casilla vacía y el carácter 'x' representará un ser vivo.

Posteriormente mostrará por pantalla la generación inicial introducida e irá obteniendo y mostrando las sucesivas generaciones hasta completar el número de generaciones solicitadas.

Ayuda: utiliza dos matrices para almacenar una generación y la siguiente.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo, considerando **TAM** con valor 5:

```
Introduzca numero de generaciones: 3
Introduzca generacion inicial:
ooxoo
oxoox
xooxx
ooxoo
xoooo

Generacion 1 (inicial):
ooxoo
oxoox
xooxx
ooxoo
xoooo

Generacion 2:
ooooo
oxxox
oxxxx
oxoxo
ooooo

Generacion 3:
```


ooooo
oxoox
xooox
oxoxx
ooooo

Fundamentos de la Programación. Práctica de Laboratorio 8

E.T.S. Ingeniería Informática. Universidad de Málaga.

Tema 4: Tipos Estructurados: Cadenas de Caracteres (Strings)

La metodología de trabajo durante la realización de las prácticas en el laboratorio será la siguiente para cada ejercicio:

- En la carpeta **Documentos**, cree una carpeta con el *nombre-del-alumno*, y en esta carpeta, desarrolle los programas con los nombres especificados para cada ejercicio de la práctica.
- Durante la práctica en el laboratorio, el alumno debe dedicar un **máximo de 35 minutos** para **desarrollar** la solución del problema, corregir los errores de compilación, ejecutar el programa y depurar los errores que encuentre.
- Aquellos ejercicios que el alumno no tenga tiempo de terminar durante la práctica en el laboratorio, se deberán terminar en casa, y entregar todos los ejercicios dentro del plazo especificado para la tarea correspondiente.
- Posteriormente, cuando se publique la solución de los ejercicios, el alumno debe **analizar** la solución proporcionada por el profesor en el campus virtual y compararla con su propia solución.
- Los ejercicios serán **corregidos** comparando la salida que produce la ejecución del programa con respecto a la salida especificada en el enunciado de cada ejercicio, por ello, cada programa debe mostrar una interacción con el usuario **exactamente** como se muestra en el enunciado de cada ejercicio.
- El nombre del fichero de cada ejercicio debe ser como se especifica en el enunciado, por ejemplo, el primer ejercicio de la práctica 8 se debe denominar **p8e1.cpp**, y así sucesivamente.

Criterios de codificación (además de los indicados en las prácticas anteriores):

- **No** está permitido utilizar los **arrays predefinidos**, se deben utilizar los **arrays** proporcionados por la **biblioteca <array> estándar de C++**.
- Todos los parámetros de entrada de tipos compuestos (**struct**, **array**, **string**) se realizarán mediante el **paso por referencia constante**.
- Todos los parámetros de salida o entrada/salida se realizarán mediante el **paso por referencia**.
- **No** está permitido utilizar las funciones de conversión (**stoi**, **stoul**, **stod**, **to_string**, etc) que proporciona la biblioteca de C++.

Ejercicios de Laboratorio

Ejercicio 1 (p8e1.cpp)

Diseñe un procedimiento que recibe como parámetro de entrada/salida una cadena de caracteres, y elimina de dicha cadena de caracteres todas las vocales, devolviendo finalmente, por el mismo parámetro, la cadena de caracteres original con las vocales eliminadas. Además, diseñe el programa y los subprogramas necesarios para probar adecuadamente su funcionamiento.

```
void eliminar_vocales(string& cadena);
```

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca una cadena: hola y adios en el muelle
Cadena original: hola y adios en el muelle
Cadena resultado: hl y ds n l mll
```

Otro ejemplo:

```
Introduzca una cadena: hoolaa y aadios een eel muueellee
Cadena original: hoolaa y aadios een eel muueellee
Cadena resultado: hl y ds n l mll
```

Ejercicio 2 (p8e2.cpp)

Desarrolle un programa que lea de teclado una cadena de caracteres (**string**) que representa un valor numérico en base decimal (base 10), calcule y almacene su valor numérico en una variable de tipo **int**, y posteriormente muestre en pantalla el valor numérico de tipo **int**, así como el resultado de multiplicar por 2 dicho valor.

Nota: desarrolle el programa sin utilizar las funciones de conversión (**stoi**, **stoul**, **stod**, **to_string**, etc) que proporciona la biblioteca de C++.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca un valor numérico: 7805
Entrada: 7805
Valor: 7805
Doble: 15610
```

Ejercicio 3 (p8e3.cpp)

Una palabra w es un anagrama de la palabra v , si podemos obtener w cambiando el orden de las letras de v . Por ejemplo, *vaca* lo es de *cava*. Diseña un programa que lea un texto y determine de cuantas palabras leídas es anagrama la primera palabra dentro de dicho texto. Para el propósito de este ejercicio, consideraremos adicionalmente que una palabra también es anagrama de ella misma.

- El texto contiene un número indefinido de palabras en letras minúsculas y termina con la palabra *fin*.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos minúsculas).
- Las palabras del texto estarán separadas, indistintamente, por espacios en blanco y/o por saltos de línea.

Ayuda: nótese que el **texto completo** no se puede almacenar en una variable de tipo **string**, ya que el número de palabras que contiene está *indefinido*. Así mismo, no es posible almacenar en un array todas las palabras del texto, ya que no conocemos el *límite máximo* de palabras que contiene. Por ello, se debe realizar un **procesamiento secuencial** de las palabras del texto, mediante un bucle de lectura adelantada:

```
string palabra;
cin >> palabra;
while (palabra != "fin") {
    // procesar la palabra
    cin >> palabra;
}
```

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el texto en minúsculas hasta (fin) con el anagrama a comprobar al principio.
vaca la vaca es cava casa vaso avac vbba ccav fin
En este texto hay 3 anagramas como <vaca>.
```

Nótese que los anagramas de *vaca* son *cava*, *avac*, así como también contabilizaremos *vaca*, dando una cuenta total de 3.

Ejercicio 4 (p8e4.cpp)

Supongamos que deseamos evaluar a un determinado número de alumnos siguiendo el criterio de que aprobará una determinada evaluación aquel que supere o iguale la nota media de la clase en dicha evaluación.

Diseña un programa que lea por teclado un número de alumnos, (como máximo podrá tomar el valor de `MAX_ALUMNOS=20`), y las notas de tres evaluaciones para cada alumno (`N_EVALUACIONES=3`), y como resultado emita un informe indicando para cada alumno el resultado de cada evaluación (*Aprobado* o *Suspense*). Considerando que el nombre de un alumno será una única palabra, sin espacios en blanco.

Nota: desarrolle el programa sin utilizar las funciones de conversión (`stoi`, `stoul`, `stod`, `to_string`, etc) que proporciona la biblioteca de C++.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduce el numero de alumnos de la clase (maximo 20): 3
Introduce el nombre del alumno y sus 3 notas: Juan 5.3 3.2 8.5
Introduce el nombre del alumno y sus 3 notas: Luis 2.3 5.2 7.5
Introduce el nombre del alumno y sus 3 notas: Pedro 6.3 5.2 6.5
Alumno    Nota-1    Nota-2    Nota-3
-----
Juan      Aprobado   Suspense  Aprobado
Luis      Suspense   Aprobado  Aprobado
Pedro     Aprobado   Aprobado  Suspense
```

Nótese que la cabecera de la tabla de notas está separada del resto por una línea de 40 guiones (-).

Ejercicio 5 (p8e5.cpp)

Diseña un programa que se comporte como una calculadora que pida repetidamente un operador de conjuntos y dos operandos que sean conjuntos formados por letras minúsculas y que escriba el resultado de la operación. Las operaciones se expresan como caracteres, siendo válidas las siguientes:

- '+' Unión de conjuntos
- '-' Diferencia de conjuntos
- '*' Intersección de conjuntos

El proceso se repetirá hasta que se introduzca como código de operación el carácter '&'. Los operadores y el resultado se expresan como cadenas de caracteres.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca la operacion a realizar (+,-,*) (& para terminar): +
Introduzca op1: azufre
Introduzca op2: zafio
El resultado es: zafioure
Introduzca la operacion a realizar (+,-,*) (& para terminar): *
Introduzca op1: azufre
Introduzca op2: zafio
El resultado es: azf
Introduzca la operacion a realizar (+,-,*) (& para terminar): -
Introduzca op1: abril
Introduzca op2: arco
El resultado es: bil
Introduzca la operacion a realizar (+,-,*) (& para terminar): &
Fin del programa
```

Ejercicio 6 (p8e6.cpp)

Diseñe un programa que lea de teclado un patrón (una cadena de caracteres) y un texto, y produzca como resultado las palabras del texto que contengan a dicho patrón. En la salida no habrá palabras repetidas.

- El texto contiene un número indefinido de palabras en letras minúsculas y termina con la palabra `fin`.
- En el texto aparecerán un número máximo de 10 palabras distintas que cumplan la condición especificada.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos minúsculas).
- Las palabras del texto estarán separadas, indistintamente, por espacios en blanco y/o por saltos de línea.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el patrón en minúsculas: re
Introduzca el texto en minúsculas hasta (fin):
creo que iremos a la direccion que nos dieron aunque pienso que
dicha direccion no es correcta pero pediremos una direccion directa fin
Resultado:
creo iremos direccion correcta pediremos directa
```

Ejercicio 7 (p8e7.cpp)

Diseñe un programa que lea de teclado un texto, y produzca como resultado las palabras del texto junto con las posiciones en las que aparece dicha palabra dentro del texto. En la salida no habrá palabras repetidas.

- El texto contiene un número indefinido de palabras en letras minúsculas y termina con la palabra `fin`.
- En el texto aparecerán un número máximo de 15 palabras distintas.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos minúsculas).
- Las palabras del texto estarán separadas, indistintamente, por espacios en blanco y/o por saltos de línea.
- Cada palabra en el texto aparecerá repetida un número máximo de 15 veces.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el texto en minúsculas hasta (fin):
creo que iremos a mi casa primero y que despues iremos a la
casa que quieras y que al final iremos a tu casa fin
creo 1
que 2 9 15 18
iremos 3 11 21
a 4 12 22
mi 5
casa 6 14 24
primero 7
y 8 17
despues 10
la 13
```

quieras 16
al 19
final 20
tu 23

Fundamentos de la Programación. Práctica de Laboratorio 9

E.T.S. Ingeniería Informática. Universidad de Málaga.

Tema 4: Resolución de Problemas Utilizando Estructuras de Datos

La metodología de trabajo durante la realización de las prácticas en el laboratorio será la siguiente para cada ejercicio:

- En la carpeta **Documentos**, cree una carpeta con el *nombre-del-alumno*, y en esta carpeta, desarrolle los programas con los nombres especificados para cada ejercicio de la práctica.
- Durante la práctica en el laboratorio, el alumno debe dedicar un **máximo de 50 minutos** para **desarrollar** la solución del problema, corregir los errores de compilación, ejecutar el programa y depurar los errores que encuentre.
- Aquellos ejercicios que el alumno no tenga tiempo de terminar durante la práctica en el laboratorio, se deberán terminar en casa, y entregar todos los ejercicios dentro del plazo especificado para la tarea correspondiente.
- Posteriormente, cuando se publique la solución de los ejercicios, el alumno debe **analizar** la solución proporcionada por el profesor en el campus virtual y compararla con su propia solución.
- Los ejercicios serán **corregidos** comparando la salida que produce la ejecución del programa con respecto a la salida especificada en el enunciado de cada ejercicio, por ello, cada programa debe mostrar una interacción con el usuario **exactamente** como se muestra en el enunciado de cada ejercicio.
- El nombre del fichero de cada ejercicio debe ser como se especifica en el enunciado, por ejemplo, el primer ejercicio de la práctica 9 se debe denominar **p9e1.cpp**, y así sucesivamente.

Criterios de codificación (además de los indicados en las prácticas anteriores):

- **No** está permitido utilizar los **arrays predefinidos**, se deben utilizar los **arrays** proporcionados por la **biblioteca <array> estándar de C++**.
- Todos los parámetros de entrada de tipos compuestos (**struct**, **array**, **string**) se realizarán mediante el **paso por referencia constante**.
- Todos los parámetros de salida o entrada/salida se realizarán mediante el **paso por referencia**.

Ejercicios de Laboratorio

Ejercicio 1 (p9e1.cpp)

Diseña un programa que lea de teclado un texto y muestre por pantalla un listado de todas las longitudes distintas de las palabras del texto, indicando para cada longitud cuantas veces aparece, considerando que **en la salida no debe haber longitudes repetidas**.

- El texto contiene un número indefinido de palabras en letras minúsculas y termina con la palabra **"fin"**, considerando que la palabra **"fin"** no forma parte del texto, y no será incluida en las estadísticas.
- En el texto aparecerán un número máximo de 10 palabras de longitudes distintas.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos minúsculas).
- Las palabras del texto estarán separadas, indistintamente, por espacios en blanco y/o por saltos de línea.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los

errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca un texto (fin para terminar):
creo que iremos a mi casa primero y que despues iremos a la
casa que quieras y que al final iremos a tu casa fin
Longitudes Repeticiones
4           4
3           4
6           3
1           5
2           4
7           3
5           1
```

Ejercicio 2 (p9e2.cpp)

Se desea informatizar la gestión de la venta de entradas para una sala de cine, que consta de cinco (5) filas, cada una de ellas con siete (7) asientos, de tal forma que cada asiento puede estar en estado **reservado** o **libre**, representados por las letras ('x') y ('o') respectivamente en la siguiente figura:

	0	1	2	3	4	5	6
0	x	x	o	o	x	o	o
1	o	o	o	o	o	o	o
2	o	x	o	o	o	x	o
3	x	x	x	x	o	o	o
4	x	o	o	o	o	o	x

Defina el tipo `SalaCine` como un array de dos dimensiones de 5 filas de 7 elementos de tipo `char`, que represente el estado de las reservas de los asientos de la sala de cine, según lo representado en la figura anterior. Además, defina los siguientes subprogramas:

- `void inicializar(SalaCine& sc)`

crea e inicializa la sala de cine `sc` recibida como parámetro con todos los asientos en estado libre.

- `void mostrar(const SalaCine& sc)`

muestra en pantalla el estado de los asientos de la sala de cine `sc` recibida como parametro, según el formato del siguiente ejemplo (la línea horizontal está formada por 16 guiones -):

```
    0 1 2 3 4 5 6
-----
0: x x o o x o o
1: o o o o o o o
2: o x o o o x o
3: x x x x o o o
4: x o o o o o x
```

- `void comprar_ticket_consecutivo(SalaCine& sc, int fila_1, int fila_2, int n, bool& ok, int& fil_sel, int& col_sel)`

dada la sala de cine `sc`, un rango de filas (desde `fila_1` hasta `fila_2`, ambas inclusive), y un número (`n`) de asientos solicitados recibidos como parámetros, si los parámetros son correctos (`fila_1` menor o igual a `fila_2`, ambos dentro del rango correcto, y `n` mayor que cero), entonces busca si hay algún bloque de (`n`) asientos libres **consecutivos** en la **misma fila** en dicho rango de filas,

- Si lo encuentra, entonces seleccionará el primer bloque (menor fila y menor columna) de `n` asientos libres **consecutivos** en la **misma fila** que se encuentre dentro del rango especificado, marcará todos los asientos seleccionados al estado reservado, el parámetro de salida `ok` tomará el valor `true`, y devolverá en los parámetros de salida `fil_sel` y `col_sel` el valor de la fila y columna respectivamente donde se encuentra el primer asiento del bloque seleccionado.

- Si los parámetros son erróneos, o no encuentra el bloque libre consecutivo dentro del rango especificado, entonces el parámetro de salida `ok` tomará el valor `false`.

Por ejemplo, para la figura del subprograma `mostrar`, para el rango de filas `{ 2, 3 }` y una solicitud de (3) asientos consecutivos, el parámetro `ok` tomará el valor `true`, el parámetro `fil_sel` tomará el valor 2 y `col_sel` tomará el valor 2, y la sala de cine quedará en el siguiente estado de reservas (enmarcadas las nuevas reservas):

```

      0 1 2 3 4 5 6
-----
0: x x o o x o o
1: o o o o o o o
2: o x x x x x o
3: x x x x o o o
4: x o o o o o x

```

■ `void cancelar_ticket(SalaCine& sc, int fila, int columna, bool& ok)`

dada la sala de cine `sc`, y un número de fila y columna que especifica un determinado asiento en la sala de cine `sc`, recibidos como parámetros, si los parámetros son correctos (`fila` y `columna` dentro del rango correcto), y el asiento especificado está reservado en la sala `sc`, entonces **cancela** la reserva de dicho asiento en la sala `sc`, que **pasará** a estar libre, y el parámetro de salida `ok` tomará el valor `true`. En otro caso, el parámetro de salida `ok` tomará el valor `false`.

Diseña un programa que permita comprobar el comportamiento de los subprogramas especificados anteriormente.

Ejercicio 3 (p9e3.cpp)

Una matriz de dos dimensiones guarda una imagen de 7×12 caracteres que tiene dibujada una circunferencia (que no tiene por qué estar centrada). Diseña **una función `diametro`**, que recibiendo como parámetro de entrada una variable conteniendo una imagen, devuelva un número entero que indique el diámetro de la circunferencia dibujada en ella. Se puede suponer que el color de fondo tiene el carácter `'.'` y el de la línea de la circunferencia el carácter `'*'`, que siempre existe una única circunferencia y no hay ningún tipo de error.

Por ejemplo, las siguientes circunferencias tienen un diámetro de 7, 6, 3 y 1 respectivamente.

```

....*.....      .....*.....      .....*.....      .....*.....
...*.*.....      ..**.....      .....*.....      .....*.....
..*...*.....      .*.*.....      .....*.....      .....*.....
.*....*.....      *...*.....      .....*.....      .....*.....
..*...*.....      *...*.....      .....*.....      .....*.....
...*.*.....      .*.*.....      .....*.*.....      .....*.....
....*.....      ..**.....      .....*.....      .....*.....

```

Diseña un programa que lea de teclado una imagen que contenga una circunferencia, calcule su diámetro utilizando la función especificada anteriormente, y finalmente lo muestre por pantalla.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

Introduce la imagen de 7 x 12 caracteres:

```

....*.....
...*.*.....
..*...*.....
.*....*.....
..*...*.....
...*.*.....
....*.....

```

Resultado: 7

Ejercicio 4 (p9e4.cpp)

Diseña un programa que lea de teclado un texto y muestre por pantalla un listado de todas las palabras del texto indicando para cada una su primera y última posición en el texto, considerando que **en la salida no debe haber palabras repetidas**.

- El texto contiene un número indefinido de palabras en letras minúsculas y termina con la palabra "fin", considerando que la palabra "fin" no forma parte del texto, y no será incluida en las estadísticas.
- En el texto aparecerán un número máximo de 15 palabras distintas.
- Cada palabra tiene un número indefinido pero limitado de caracteres (todos alfabéticos minúsculas).
- Las palabras del texto estarán separadas, indistintamente, por espacios en blanco y/o por saltos de línea.

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el texto en minúsculas hasta (fin):
creo que iremos a mi casa primero y que despues iremos a la
casa que quieras y que al final iremos a tu casa fin
creo 1 1
que 2 18
iremos 3 21
a 4 22
mi 5 5
casa 6 24
primero 7 7
y 8 17
despues 10 10
la 13 13
quieras 16 16
al 19 19
final 20 20
tu 23 23
```

Ejercicio 5 (p9e5.cpp)

Un antiguo método de cifrado consta de una clave alfanumérica, compuesta por **todas** las letras minúsculas (sin incluir la letra ñe) y **todos** los dígitos, dispuesta en una tabla bidimensional de 6×6 elementos como en el siguiente ejemplo:

	0	1	2	3	4	5
0	p	k	a	f	5	v
1	e	9	o	t	y	0
2	s	3	z	7	d	j
3	r	n	b	u	m	1
4	2	w	4	h	8	g
5	c	x	6	q	i	l

Se deberán diseñar las constantes, tipos y subprogramas necesarios, así como el siguiente subprograma:

- El subprograma `cifrar` tiene tres parámetros, de tal forma que recibe como primer parámetro de entrada una clave almacenada en una tabla de cifrado como la especificada anteriormente. El segundo parámetro también es de entrada, es una cadena de caracteres que contiene el texto de entrada que se debe cifrar. El tercer parámetro es de salida, y contendrá una cadena de caracteres con el texto cifrado, resultado de cifrar el texto de entrada del segundo parámetro según el siguiente esquema:

- Para cada **par** de caracteres del texto de entrada (texto a cifrar), se producen también **dos** caracteres en el texto cifrado, según el siguiente proceso:
 - Se buscan ambos caracteres en la tabla de la clave (*se puede suponer que ambos caracteres estan en la tabla*), proporcionando los índices de la fila y columna donde está almacenado cada carácter ($f1, c1$) y ($f2, c2$). Por ejemplo, para el par de caracteres **ho**, se obtienen las coordenadas ($4, 3$) y ($1, 2$) de la tabla.
 - Se obtienen los caracteres que se encuentran en la tabla en las posiciones ($f1, f2$) y ($c1, c2$). Por ejemplo, las anteriores coordenadas producen las coordenadas ($4, 1$) y ($3, 2$), que corresponden con los caracteres **wb** según la tabla anterior.
 - Estos nuevos caracteres se **añaden** al final del texto cifrado.
- Se repite el proceso anterior por cada par de caracteres del texto de entrada. En caso de que el texto de entrada tenga un número de caracteres impar, se desecha el último carácter del mismo, y no será tenido en cuenta.
- Se puede suponer que en el texto de entrada sólo aparecerán letras minúsculas y dígitos. También se puede suponer que la tabla clave también es correcta (aparecen todas las letras minúsculas y todos los dígitos). Por ejemplo, para el texto de entrada **holayadios**, produce el texto cifrado: **wbc6e4j8os**.

Desarrolle, además, el programa principal que lea de teclado una cadena de caracteres (sin espacios en blanco), y la filtre para eliminar todos aquellos caracteres que no sean letras minúsculas ni dígitos. Posteriormente cifrará (utilizando el subprograma **cifrar**) el texto de entrada filtrado anteriormente, produciendo el texto cifrado. A continuación procederá a volver a cifrar el texto cifrado para obtener el texto descifrado. Finalmente mostrará en texto de entrada filtrado, el texto cifrado y el texto descifrado. Este programa utilizará la siguiente clave de cifrado:

```
const Matriz CLAVE = {{
    {'p','k','a','f','5','v'}},
    {'e','9','o','t','y','0'}},
    {'s','3','z','7','d','j'}},
    {'r','n','b','u','m','l'}},
    {'2','w','4','h','8','g'}},
    {'c','x','6','q','i','l'}};
};
```

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación. Por ejemplo:

```
Introduzca el texto (sin espacios): hola95,72;adios
Entrada: [hola9572adios]
Cifrado: [wbc6eydradx4]
Descifrado: [hola9572adio]
```

Ejercicio 6 (p9e6.cpp)

Desarrolla un programa que lea de teclado un número entero **K** y una colección de números enteros para completar una matriz 2D de tamaño 3×4 .

A partir de los datos leídos, el programa calculará y mostrará por pantalla los **K** números de la matriz que más veces se repiten.

Notas:

- Si **K** es mayor que el número de valores distintos de la matriz, entonces se mostrarán todos esos valores distintos (ver ejemplo 2).
- Si hay valores de la matriz que se repiten igual número de veces y alguno de ellos no se puede mostrar porque de hacerlo se excedería el valor de k, da igual los valores que queden sin mostrarse (ver ejemplo 3).

Ejecute el programa y compruebe si produce los resultados esperados. En caso contrario, corrija los errores y repita el proceso de comprobación.

- Ejemplo 1 (el orden de los elementos en la salida no es importante):

```
Introduzca k: 4
Introduzca la matriz 3x4 (por filas):
45  -17  867  45
 2  867  -17   3
 1   -2   45   3
Los valores que mas se repiten son: 45 -17 867 3
```

- Ejemplo 2 (el orden de los elementos en la salida no es importante):

```
Introduzca k: 8
Introduzca la matriz 3x4 (por filas):
45  -17  867  45
 2  867  -17   3
 1   -2   45   3
Los valores que mas se repiten son: 45 -17 867 3 2 1 -2
```

- Ejemplo 3 (el orden de los elementos en la salida no es importante):

```
Introduzca k: 3
Introduzca la matriz 3x4 (por filas):
45  -17  867  45
 2  867  -17   3
 1   -2   45   3
Los valores que mas se repiten son: 45 -17 867
```

En este caso, son también válidas las siguientes salidas: { 45, -17, 867 }, { 45, -17, 3 }, y { 45, 867, 3 }