EJERCICIOS DE LA ASIGNATURA TIEMPO-REAL

GRADO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

EJERCICIOS DE COLECCIONES

CALCULO DE PI/4 SEGÚN LEIBNIZ

Se pide que calcule el valor de Pi/4 según la fórmula de Leibniz[1] y compruebe si coincide con el valor de la constante `M_PI_4` definida en `<cmath>`:

$\sum_{n=0}^{\infty} \frac{(-1)^n}{2 \cdot n + 1} = \frac{\pi}{4}$	`if (pi_4 == M_PI_4) {`
	`  cout << "Hurra, Leibniz estaba en lo cierto!\n";`
	`} else {`
	`  cout << "¿Se equivocó Leibniz? :-( \n";`
	`}`

En caso de que no coincidan los valores de Pi/4, muestre los dos valores calculados y utilice la función `setprecision()`, definida en `<iomanip>`, para incrementar el número de decimales que se muestran por consola:

```
cout << setprecision(20) << pi_4 << " == " << M_PI_4 << '\n';
```

¿Qué está pasando?. ¿Cómo se podría solucionar?. Recuerde que los números reales introducen pequeños errores de redondeo, que a partir de cierta posición decimal no son exactos, y que en este caso la fórmula de Leibniz converge a Pi/4 cuando se hacen infinitas iteraciones. Una pista: hay que utilizar la función `fabs()` definida en `<cmath>`.

SUMA RECURSIVA DE ELEMENTOS DE UN VECTOR

Se pide que haga un programa que genere un vector con los 10 primeros números naturales y posteriormente un segundo vector que contenga las sumas parciales de los elementos del vector original. Por ejemplo:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
Sumas parciales hasta el final
55, 54, 52, 49, 45, 40, 34, 27, 19, 10,
```

Realice dos ampliaciones del programa: una en la que el usuario introduce el tamaño y los valores, y otra en la que el usuario solo indica el tamaño y el programa general aleatoriamente los valores.

CRIBA DE ERASTÓTENES

Es un método para calcular todos los números primos que existen en un rango y que consiste en crear un vector del tamaño del rango. El contenido del vector indica lo que conocemos del número: 'd' (desconocido), 'p' (es primo) y 'c' (es compuesto). Inicialmente todas las posiciones del vector se inicializan a 'd', y se descartan las dos primeras posiciones (asumimos que 0 y 1 son primos). El algoritmo consiste en recorrer el vector desde la posición 2, marcando como primo el primer índice cuyo contenido sea 'd'. A continuación se marcan como compuesto todos sus múltiplos, con cuidado de no sobrepasar el rango inicial. La entrada de la Wikipedia tiene una animación del funcionamiento de la criba. Utilice la clase **`std::vector`** (definida en **`<vector>`**) para almacenar los datos de la criba. Utilice un rango grande

[1] https://es.wikipedia.org/wiki/Serie_de_Leibniz

(por ejemplo, hasta 1000000), ejecute el programa varias veces y mida los tiempos de ejecución con el siguiente código:

```
#include <chrono>
auto tiempo_ini = std::chrono::steady_clock::now();
//AQUÍ AÑADO EL ALGORITMO CUYO TIEMPO DE EJECUCIÓN QUIERO MEDIR
auto tiempo_fin = std::chrono::steady_clock::now();
std::cout << "\n\nTiempo transcurrido: " <<
std::chrono::duration_cast<std::chrono::microseconds>(tiempo_fin-tiempo_ini).count() <<
" us\n";
// duration_cast permite también convertir el tiempo medido en ms, s, etc. Consulte doc
```

Los 8 últimos primos del rango (para comprobar que efectivamente ha resuelto bien el ejercicio) son los siguientes: `999907, 999917, 999931, 999953, 999959, 999961, 999979, 999983`

Amplíe el programa para que el usuario introduzca por consola el valor del rango hasta el que quiere realizar el cálculo.

BÚSQUEDA DE TEXTO EN FICHERO

Se pide que haga un programa que solicite el nombre de un fichero de texto formado por varias líneas (separadas por un retorno de carro, carácter '\n'), una palabra a buscar dentro del texto, e indique cuántas veces aparece dicha palabra en el fichero y en qué líneas (con una sola aparición es suficiente). Almacene en un vector los números de las líneas en que ha sido encontrada la palabra. Utilice la función `find` de `<string>` para buscar la palabra. Dicha función retorna la posición en que ha encontrado la palabra o la constante `string::npos` para indicar que no la ha encontrado. Por tanto, puede utilizar la siguiente estructura de selección:

```
if (una_linea.find(texto_buscado) != std::string::npos) { ... }
```

Por ejemplo, en el *capítulo 1 del Quijote* que se encuentra en la sección de **Prácticas** del Aul@ Virtual, la palabra "*juicio*" aparece en las líneas 3, 7 y 11.

CREACIÓN ALEATORIA DE GRUPOS DE PRÁCTICAS

Se pide aplicar el mismo algoritmo que el de reparto de las cartas de la baraja, pero ahora para repartir los alumnos matriculados en la asignatura en dos grupos de prácticas del mismo tamaño. El fichero "*reyesVisigodos.txt*" del Aul@Virtual contiene la lista de los 50 reyes visigodos que debe repartir aleatoriamente en dos grupos, que serán almacenados en dos fichero de texto, "*grupo1.txt*" y "*grupo2.txt*". Asuma que la longitud máxima de un nombre es 50 caracteres y que pueden tener espacios en blanco (por lo que debe utilizar la función **`std::getline`**, definida en **`<string>`**). La librería `<fstream>` define las clases para crear, leer y escribir en ficheros de texto: **`std::ifstream`** representa un fichero de entrada (lectura de datos) y **`std::ofstream`** un fichero de salida (escritura de datos).

LECTURA DE FICHERO EN FORMATO CSV

El fichero "*flores.csv*" del Aul@ Virtual contiene el conjunto de datos de la flor de Iris[2] en formato CSV (*comma separated values*). Cada línea contiene 4 datos numéricos (longitud y anchura del sépalo, longitud

[2] https://es.wikipedia.org/wiki/Iris_flor_conjunto_de_datos

y anchura de los pétalos) y el nombre de la especie de Iris a que corresponden los valores, separados por comas. Se pide que haga un programa que lea los datos; imprima por consola cuántas muestras hay de cada especie; imprima los valores medios de longitud y anchura del sépalo, longitud y anchura de los pétalos; y finalmente almacene en ficheros distintos los valores de cada especie. Por ejemplo, la salida sería:

```
50 muestras de 'setosa'
        MEDIAS. Sepal_l: 5.006, sepal_w: 3.418, petal_l: 1.464, petal_w: 0.244
50 muestras de 'versicolor'
        MEDIAS. Sepal_l: 5.936, sepal_w: 2.77, petal_l: 4.26, petal_w: 1.326
50 muestras de 'virginica'
        MEDIAS. Sepal_l: 6.588, sepal_w: 2.974, petal_l: 5.552, petal_w: 2.026
```

Declare una estructura para almacenar los valores de las distintas medidas y poder, de esta forma, declarar un vector, como por ejemplo:

```
struct medidas_flor_t {
   double sepal_l, sepal_w, petal_l, petal_w;
};

struct flor_t{
   medidas_flor_t medidas;
   std::string especie;
};
```

Puede declarar variables de este tipo y acceder a cada uno de sus campos utilizando el operador punto (.), como se muestra a continuación:

```
      flor_t ej {1, 2, 3, 4, "ej"};
      ej.medidas.petal_w = 9;
      ej.especie = "especie";
std::vector<flor_t> flores;
```

Recuerde que puede utilizar el tercer parámetro de la función `getline()` para establecer el carácter hasta el que debe leer la entrada; existe una función `std::stod`, definida en `<string>`, que genera un número de tipo **`double`** a partir de su representación como `string`; que puede utilizar un `mapa` para calcular las medias, ya que los mapas no permiten que se repitan sus claves.

DERIVACIÓN DE POLINOMIOS

Suponga que cada posición de un array recoge el coeficiente de un polinomio. La posición primera (array[0]) el término independiente; la segunda posición (array[1]) el coeficiente de la x. La tercera posición (array[2]) el del x^2; y en general, la de la posición i-ésima (array[i]) el coeficiente para x^i. Así, por ejemplo, un array de 10 elementos podría expresar un polinomio de un grado igual o menor que 9: $p(x) = p_0 + p_1 \cdot x + p_2 \cdot x^2 + p_3 \cdot x^3 + \cdots + p_8 \cdot x^8 + p_9 \cdot x^9$.

A partir de lo anterior, se plantea el ejercicio de hacer una función **`derivar_polinomio`** a la que se le pasa un polinomio (representado en una primera versión mediante el tipo std::array , y en una segunda versión del ejercicio con std::vector; no utilice el array de C, int v[]. C++ nos da mejores alternativas), y

que retorna como resultado el polinomio derivado. Haga una función main en la que probar las dos versiones.

En la versión con std::array tendrá que fijar el tamaño del polinomio, mientras que en la de std::vector puede preguntarle al usuario el tamaño en tiempo de ejecución, e ir añadiendo todos los coeficientes que quiera (haga un menú para gestionar este comportamiento). Dicho menú podría incluso incluir una opción "cambiar el tamaño del polinomio". Pero la forma de derivar debe ser básicamente la misma en ambos casos.

MULTIPLICACIÓN DE POLINOMIOS

Diseñe una función que multiplique polinomios, representados tal y como se ha descrito en el ejercicio anterior, y haciendo dos versiones: una con std::array y otra con std::vector. Esa función devuelve el polinomio resultado de realizar la multiplicación. Ejemplo:

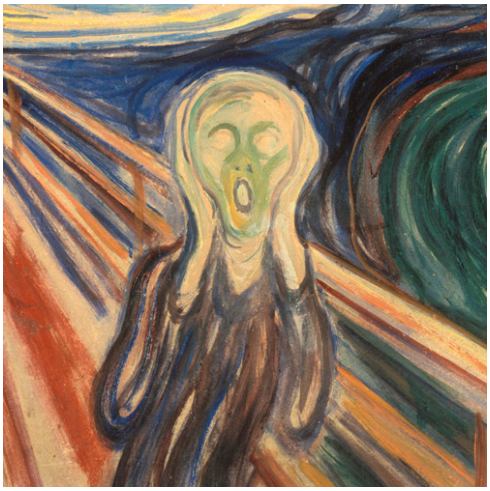
Si polA es $x^3 + 2x^2 + 3x - 6$ y polB es $-5x^4 + 4x^3 - 3x^2 + 2x - 1$,

polA · polB es $-5x^7 - 6x^6 - 10x^5 + 38x^4 - 30x^3 + 22x^2 - 15x + 6$

Haga una función main en la que probar las dos versiones.

Las soluciones vienen a continuación.

Consultar solo si ya está un poco desesperad@, o para comparar soluciones



SOLUCIONES A LOS EJERCICIOS

CALCULO DE PI/4 SEGÚN LEIBNIZ

```
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

int main () {
   const int iteraciones = 5e7;
   double pi_4 = 1, num=-1, den=3;
   for (int i=0; i<iteraciones; ++i) {
      pi_4 += num/den;
      num = -num;
      den += 2;
   }
   if (pi_4 == M_PI_4) {
      cout << "Hurra, Leibniz estaba en lo cierto!\n";
   } else {
      cout << "¿Se equivocó Leibniz? :-(" << setprecision(20) << pi_4 << " == "
         << M_PI_4 << '\n';
   }

   const double epsilon = 1e-7;
   if (fabs(pi_4-M_PI_4)<epsilon) {
      cout << "Hurra, Leibniz estaba en lo cierto!\n";
   } else {
      cout << "¿Se equivocó Leibniz? :-(" << setprecision(20) << pi_4 << " == "
         << M_PI_4 << '\n';
   }
}
```

En relación a la pregunta sobre la precisión del cálculo de PI Leibniz, el ejercicio es una demostración de dos problemas habituales en programación:

1. La representación de números reales no es exacta en el computador. Siempre se comete un error de redondeo, que se amplía conforme se hacen operaciones con los números reales. Por ejemplo, fallo de los misiles Patriot desplegados en la guerra de Irak en 1994 (https://www.xataka.com/historia-tecnologica/dia-que-misil-mato-a-28-soldados-porque-sistema-defensa-antimisiles-ignoro-error-0-000000095-segundos ; también está en el Aul@ Virtual).
2. No todos los algoritmos son igual de eficientes para resolver un determinado problema, y hay que buscar el mejor, aunque eso requiera un poco de investigación previa. Esto es lo que dice la Wikipedia sobre la fórmula de Leibniz:

   En la práctica, la fórmula de Leibniz es muy poco eficiente para el cálculo de π, pues requiere un número enorme de pasos para obtener cierta precisión. Para calcular π con 10 decimales correctos hacen falta más de cinco mil millones de operaciones matemáticas, que los ordenadores tardarán más en realizar que en calcular π con millones de decimales correctos mediante fórmulas más eficientes.

Con números reales es básicamente imposible aplicar el operador igualdad justamente por el problema de redondeo, por lo que habitualmente se define un error mínimo que queremos cometer (el famoso 'epsilon') y se comprueba entonces si el error es menor que ese Epsilon de la siguiente forma general:

```
const double epsilon = 1e-7;  // por ejemplo … el valor "bueno" depende de cada
                              // caso (no es lo mismo un programa que opera con
                              // distancias kilométricas que con atómicas
if (fabs(valor1 – valor2) < epsilon) {
        // suponemos que los valores están "lo suficientemente cerca"
        // como para considerar que son iguales
} else {
         // los valores no están "lo suficientemente cerca"
}
```

Más información sobre números reales y errores de redondeo en https://aulavirtual.upct.es/mod/resource/view.php?id=50144

SUMA RECURSIVA DE ELEMENTOS DE UN VECTOR

```
#include <iostream>
#include <vector>

using namespace std;

int main () {
   vector<int> vs(10), vs_suma;
   for (int i=0; i<vs.size(); ++i)
      vs[i] = i+1;
   for (const auto &v : vs)
      cout << v << ", ";
   cout << "\n Sumas parciales hasta el final\n";
   vs_suma.reserve(vs.size());
   for (int suma, i=0; i < vs.size(); ++i) {
      suma=0;
      for (int j=i; j<vs.size(); ++j) {
         suma+=vs[j];
      }
      vs_suma.push_back(suma);
   }
   for (const auto &v : vs_suma)
      cout << v << ", ";
}
```

AMPLIACIÓN:

```
int f_menu() {
   int menu;
   do {
      cout << "1.- Diez primeros numeros\n";
      cout << "2.- 'n' números aleatorios\n";
```

```
      cout << "3.- 'n' numeros introducidos por el usuario\n";
      leer_descartando_entrada_incorrecta(cin, menu);
   } while(menu<1 or menu>3);
   return menu;
}

vector<int> generar_vector (const int opcion) {
   vector<int> vs;
   int tam, val;
   if (opcion == 1) {
      tam = 10;
   }else {
      cout << "Tam vector? ";
      leer_descartando_entrada_incorrecta(cin, tam);
   }
   vs.reserve(tam);
   random_device rd;
   default_random_engine gen(rd());
   uniform_int_distribution<> vals (1, 100);
   switch (opcion) {
      case 1:
         for (int i=0; i<tam; ++i)
            vs.push_back(i+1);
         break;
      case 2: // aleatorios
         for (int i=0; i<tam; ++i)
            vs.push_back(vals(gen));
         break;
      case 3:
         for (int i=0; i<tam; ++i) {
            leer_descartando_entrada_incorrecta(cin, val);
            vs.push_back(val);
         }
         break;
   }
   return vs;
}

void f_vector_suma2() {
   vector<int> vs, vs_suma;
   int menu;
   menu = f_menu();
   vs = generar_vector(menu);
   for (const auto &v : vs)
      cout << v << ", ";
   cout << "\n Sumas parciales hasta el final\n";
   vs_suma.reserve(vs.size());
   for (int suma, i=0; i < vs.size(); ++i) {
      suma=0;
      for (int j=i; j<vs.size(); ++j) {
         suma+=vs[j];
      }
      vs_suma.push_back(suma);
   }
   for (const auto &v : vs_suma)
      cout << v << ", ";
}
```

CRIBA DE ERASTÓTENES

```
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;

void f_criba() {
   const int tam = 1e6;
   vector<char> criba (tam, 'd');
   criba[1]='p';
   for (int i=2; i<tam; ++i) {
      if (criba[i]=='d') {
         criba[i]='p';
         for (int j=i*2; j<tam; j+=i) {
            criba[j]='c';
         }
      }
   }
   for(int i=tam-1e2; i<tam; ++i)
      if(criba[i]=='p')
         cout << i << ' ';
   cout << '\n';
}


int main() {
   auto tiempo_ini = std::chrono::steady_clock::now();
   f_criba();
   auto tiempo_fin = std::chrono::steady_clock::now();
   std::cout << "\n\nTiempo transcurrido: "
         << std::chrono::duration_cast<std::chrono::microseconds>
         (tiempo_fin-tiempo_ini).count() << " us\n";
   return 0;
}
```

BÚSQUEDA DE TEXTO EN FICHERO

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>

using namespace std;

int main() {
   ifstream quijote;
   string fichero, texto_buscado;
   cout << "Nombre fichero? ";
   cin >> fichero;
   quijote.open(fichero);
   if (quijote) {
      vector<string> texto;
      vector<int> lineas_con_texto;
      string aux;
```

```
    texto.reserve(50);
    lineas_con_texto.reserve(50);
    texto.emplace_back(""); // para empezar a contar en la linea 1
    while (!quijote.eof()) {
      getline(quijote, aux);
      texto.emplace_back(aux);
    }
    cout << "Palabra buscada? ";
    cin >> texto_buscado;
    for (int i=1; i<texto.size(); ++i) {
      if (texto[i].find(texto_buscado) != std::string::npos) {
        lineas_con_texto.push_back(i);
      }
    }
    cout << "'" << texto_buscado << "' encontrado " << lineas_con_texto.size()
       << " veces\nLíneas: ";
    for (const auto &l : lineas_con_texto)
      cout << l << ", ";
  } else {
    cout << "[ERROR] '" << fichero << "' no encontrado\n";
  }
}
```

CREACIÓN ALEATORIA DE GRUPOS DE PRÁCTICAS

```
#include <iostream>
#include <fstream>
#include <vector>
#include <random>
#include <string>

using namespace std;

int main() {
    const std::string nombre_fichero {"reyesVisigodos.txt"};
    std::ifstream fichero (nombre_fichero);
    if (fichero) {
        std::vector<std::string> v_reyes, v_grupo1, v_grupo2;
        while (!fichero.eof()) {
            std::string tmp;
            std::getline(fichero, tmp);
            if (!tmp.empty()) {
                v_reyes.push_back(tmp);
            }
        }
        std::random_device rd;
        std::mt19937 generador(rd());
        const int tam = v_reyes.size();
        v_grupo1.reserve(tam/2+1);
        v_grupo2.reserve(tam/2+1);
        for (int i=tam-1; i>=tam/2; --i) {
            std::uniform_int_distribution<> distribucion(0, i);
            int v = distribucion(generador);
            v_grupo1.push_back(std::move(v_reyes[v]));
            v_reyes.erase(begin(v_reyes)+v);
        }
```

```
        v_grupo2 = v_reyes;
        {
            std::ofstream grupo ("grupo1.txt");
            for (const auto &r : v_grupo1)
                grupo << r << std::endl;
        }
        {
            std::ofstream grupo ("grupo2.txt");
            for (const auto &r : v_grupo2)
                grupo << r << std::endl;
        }

    } else {
        std::cout << "Error abriendo fichero " << nombre_fichero << std::endl;
    }

    return 0;

}
```

LECTURA DE FICHERO EN FORMATO CSV

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>

using namespace std;

struct medidas_flor_t {
  double sepal_l , sepal_w , petal_l, petal_w;
};

struct flor_t{
  medidas_flor_t medidas;
  string especie;
};

ostream& operator<< (ostream &os, const flor_t &f) {
  os << '[' << f.especie << "] ";
  os << "sepal_l: " << f.medidas.sepal_l << ", sepal_w: " << f.medidas.sepal_w;
  os << ", petal_l: " << f.medidas.petal_l << ", petal_w: " << f.medidas.petal_w;
  return os;
}

struct medias_flor_t{
  medidas_flor_t valores;
  int num_muestras;
};

vector <flor_t> leer_csv(){
  vector <flor_t> flores;
  flores.reserve(100);
  ifstream csv;
  string fichero;
  cout << "Nombre fichero? ";
  cin >> fichero;
```

```
  csv.open(fichero);
  if (csv) {
    while (!csv.eof()) {
      string aux_s;
      flor_t aux_f;
      getline(csv, aux_s, ',');
      aux_f.medidas.sepal_l = stod (aux_s);
      getline(csv, aux_s, ',');
      aux_f.medidas.sepal_w = stod (aux_s);
      getline(csv, aux_s, ',');
      aux_f.medidas.petal_l = stod (aux_s);
      getline(csv, aux_s, ',');
      aux_f.medidas.petal_w = std::stod (aux_s);
      getline(csv, aux_s);
      aux_f.especie = aux_s;
      //cout << aux_f << '\n';
      flores.push_back(aux_f);
    }
  } else {
    cout << "[ERROR] '" << fichero << "' no encontrado\n";
  }
  return flores;
}

void crear_ficheros_especie(const vector <flor_t> flores){
  map<string, ofstream> ficheros;
  for (const auto &f : flores) {
    if (!ficheros[f.especie]) {
      ficheros[f.especie] = ofstream {f.especie+".txt"};
    }
    ficheros[f.especie] << f.medidas.sepal_l << ", " << f.medidas.sepal_w << ", ";
    ficheros[f.especie] << f.medidas.petal_l << ", " << f.medidas.petal_w << '\n';
  }
}

int main() {
  vector <flor_t> flores = leer_csv();
  map<string, medias_flor_t> muestras;
  for (const auto &f : flores) {
    if (muestras.count(f.especie) == 0) { // si no existe
      muestras[f.especie] = {0.0, 0.0, 0.0, 0.0};
    }
    muestras[f.especie].valores.sepal_l += f.medidas.sepal_l;
    muestras[f.especie].valores.sepal_w += f.medidas.sepal_w;
    muestras[f.especie].valores.petal_l += f.medidas.petal_l;
    muestras[f.especie].valores.petal_w += f.medidas.petal_w;
    ++muestras[f.especie].num_muestras;
  }
  for (const auto &e : muestras) {
    const int n_muestras = e.second.num_muestras;
    cout << n_muestras << " muestras de '" << e.first << "'\n";
    cout << "\t MEDIAS. Sepal_l: " << e.second.valores.sepal_l/n_muestras;
    cout << ", sepal_w: " << e.second.valores.sepal_w/n_muestras;
    cout << ", petal_l: " << e.second.valores.petal_l/n_muestras;
    cout << ", petal_w: " << e.second.valores.petal_w/n_muestras << '\n';
  }
```

```
    crear_ficheros_especie (flores);
}
```