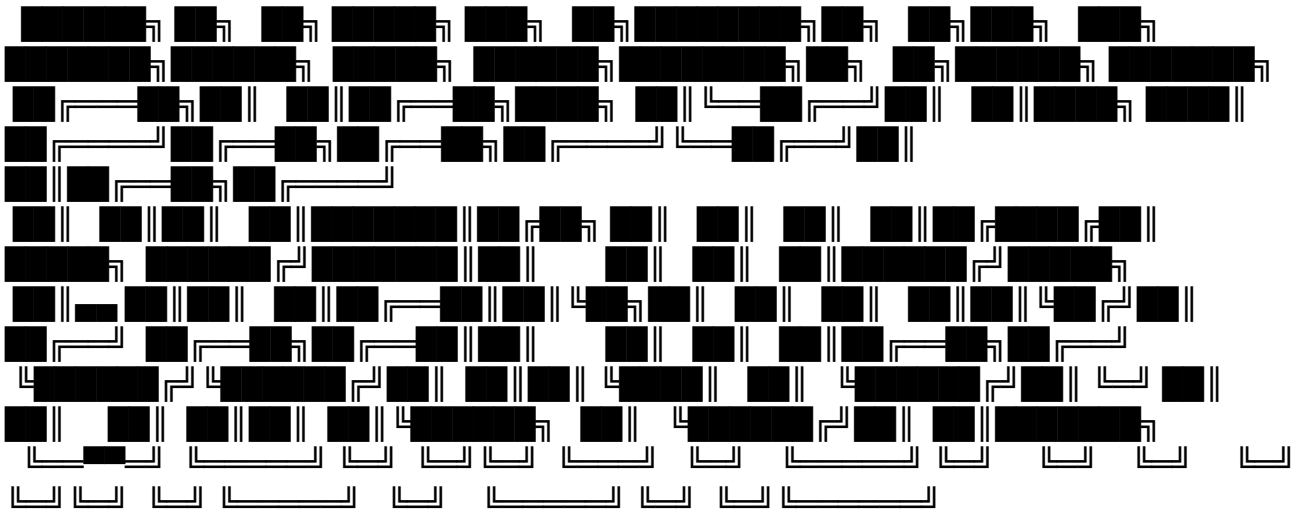
/*



PRESENTA

_____

_____

```
 _
' )    |V| _´_|_ _  _| _    _| _   |V| _  _ _|_ _  _ _  _ | _
 /_ .  |  |(/_ | (_)(_|(_)  (_|(/_  |  |(_)| | | (/_(_(_||  |(_)
```

Qué pasa chavales, muy buenas a todos, aquí Crespo comentando (xD en serio, willy pls). Con este código, escrito en C++, vamos a hacer que
este ordenador calcule pi utilizando el método que se usó para hacer ciertos cálculos en el desarrollo del proyecto Manhattan: el método
de Montecarlo.
Si quieres saber las ideas básicas de este algoritmo o no sabes como coj***s has llegado aquí y qué es este código
hablandome así wtf, te recomiendo pasarte por YouTube y verte este vídeo
--> https://youtu.be/DQ5qqHukkAc <--.

Si no ves un montón de símbolos raros, te recomiendo que estires el tamaño de tu ventana, te llevarás una sorpresa --->

Muy bien, ahora que estamos en la misma onda, sabemos que vamos a calcular pi a partir de la proporción entre puntos aleatorios dentro
de un círculo y los totales (ya que todos caen dentro del cuadrado). Puesto en ecuaciones:

$$pi = \frac{4 \ (\text{nº de puntos dentro del circulo})}{(\text{nº de puntos "lanzados"})}$$

Atentos: en vez de medir las proporciones en un circulo entero vamos a calcularlo para un CUADRANTE de este circulo; un cuarto de él,
como una porción de una pizza partida en cuatro.
Con más exactitud: tenemos un cuadrado de lado r. Dentro está inscrita tal porción del circulo, cuyo centro estaría ubicado en uno de
los vertices del cuadrado, que designamos con coordenadas (x=0 , y=0). Vamos a dividir el numero de puntos que caigan dentro de
esa porción (es decir que verifican que x^2 + y^2 < r^2) entre el numero de puntos totales lanzados.

Además, no solo vamos a hacer esto una vez: se realizaran varias tandas de tiradas, cada una con el mismo número de puntos aleatorios
a lanzar. Dado que los resultados de Montecarlo no son los mismos en cada ejecución del programa (dada la naturaleza aleatoria
del cálculo), haremos que el código repita el método un cierto número de veces para hacer estadística con los varios pi's que nos
saque. Obtendremos media y desviación estándar, con lo que tendremos una estimación del error del método. En pocas palabras: tendremos
un pi fiable, no tan dependiente de la ejecución y con un error que nos permita acotar el valor exacto.

Así he nombrado a cada variable:

* r : el radio del circulo.
* N : el número total de puntos a lanzar. Es un input.
* x : posición del punto aleatorio en el eje X.
* y : posición del punto aleatorio en el eje Y.
* cota : es el número de veces que queremos que se repita internamente Montecarlo para tener una estimación del error.
* c : número de puntos que han caido dentro de la porción.
* pi_ar : array (como un vector) de valores de pi recogidos en cada tanda de tiradas. Sus dimensiones son "cota".
* pi : valor de pi obtenido como la media de los valores de "pi_ar".
* err : error en el valor de pi obtenido como la desviación estándar de los valores de "pi_ar".

*/

#include <iostream> // Cargo librerías (colecciones de funciones ya hechas que hacen cositas): "iostream" me permite sacar texto
#include <cmath> // y números por el terminal, para que puedas ver el valor de pi, y "cmath" es una colección de funciones
#include <fstream> // matemáticas que necesito, como elevar al cuadrado y hacer la raiz cuadrada.

```
using namespace std;  // A efectos práticos, esto es para no tener que poner "std" todo el rato.

int main(){          // ¡Comenzamos!

   // Imprimimos en el terminal... la bienvenida ;)

   cout<<endl<<endl;
   cout<<"                ╔═╗ ┬ ┬┌─┐ ┌┐ ┌┌─┬┬┬─┐  ╔═╗┬─┐ ┌─┐ ┌─┐ ┌─┬┬
┬┬─┐ ┌─┐ "<<endl;
   cout<<"                ╠═╣ │ │├─┤ │││ │ │ │││  ╠╣ ├┬┘ ├─┤ │  │ │├┬┘ ├ "<<endl;
   cout<<"                ╚═╝ ╙ └─┘┴ ┴┘ ┴ ┴ └─┘┴┴  ╙ ┴└─┴┴└─┘ ┴ └─┘┴└─ └─┘ "<<endl;
   cout<<"
______________________________________________________________________________
"<<endl<<endl;
   cout<<"                              PRESENTA...                         "<<endl;
   cout<<endl;
   cout<<"╔═╗ ┌─┐┬  ┌─┐┬┬┬  ┌─┐  ┌┬┐ ┌─┐  ╔═╗┬   ╔╦╗ ┌─┐ ┌┬┐ ┌─┐ ┌┬┐ ┌─┐
 ┌┬┐ ┌─┐  ╔╦╗ ┌─┐ ┌┐ ┌┌┬┐ ┌─┐ ┌─┐ ┌─┬─┐┬  ┌─┐ "<<endl;
   cout<<"║  ├─┤ │ │ ││ ││  ││├  ╠═╝│ ───  ║║║├  │ │ │││ │  ││├  ║║║ ││││
 │ ├ │  ├─┤ ├┬┘ │  │ │"<<endl;
   cout<<"╚═╝┴ ┴┴└─┘ └─┘ └─┘┴─┘ └─┘ ─┴┘ └─┘  ╩╩ ┴    ╩ ╩╩└─┘ ┴ └─┘ ─┴┘ └─┘
─┴┘ └─┘  ╩ ╩└─┘ ┘ └┘ ┴ └─┘ └─┘┴ ┴┴└─┴─┘ └─┘ "<<endl;

   cout<<endl<<"           ¡Con todos los números aleatorios que quieras! ¡Garantizado!
;)"<<endl<<endl;


cout<<"_______________________________________________________________________________
______________________"<<endl<<endl;
   cout<<"                            ADVERTENCIA                       "<<endl;
   cout<<"Si está usando este código en un compilador online, puede ser que utilizar un número
excesivo"<<endl;
   cout<<" de dardos haga que el cálculo necesite tanto tiempo para ser computado que el
compilador lo"<<endl;
   cout<<"          aborte automáticamente. En ese caso, pruebe un número menor."<<endl;

   ////////////////////////////////////////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////////////////////////////////////

   double N = 12345;        // NÚMERO DE PUNTOS ALEATORIOS A LANZAR. ¡CAMBIAD ESTO!

   ////////////////////////////////////////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
    // Se expulsa tal número por terminal:

    cout<<endl<<"
╔══════════════════════════════════════════════════════════════════════════════════════════
═════════════════════════════════════╗";
    cout<<endl<<"  ║                    Número de 'dardos' a lanzar :   "<<N;
    cout<<endl<<"
╚══════════════════════════════════════════════════════════════════════════════════════════
═════════════════════════════════════╝";

    double r=1;       // RADIO DEL CÍRCULO: Puedes cambiar este número si lo deseas; el tamaño
de la circunferencia no afecta a pi.

    srand((unsigned)time(0));  // Para que el ordenador nos genere los números aleatorios, hay que
darle una semilla. Normalmente
                              // suele cogerse el tiempo del reloj del ordenador en ese momento.

    double x;                 // Defino las coordenadas de cada punto aleatorio. No queremos
almacenarlas; reescribiremos estas variable.
    double y;

    double c=0;               // Defino el número de puntos dentro del círculo (de la porción). Partimos
de 0.

    //int m=1e7;

    int cota=10;              // COTA. NÚMERO DE REPETICIONES DEL MÉTODO. Puedes cambiar
este número si lo deseas.

    double pi_ar[cota];       // Defino el array que voy a llenar de los distintos pi's que obtenga.


    for (int j=0; j<cota; j++) {              // Primer BUCLE. Repetirá Montecarlo "cota" veces.

        for (int i=0; i<N; i++) {              // Segundo BUCLE. En cada vuelta, lanza un dardo.

            x=(double)rand()/(double)RAND_MAX;   // Generamos dos números aleatorios desde 0 a 1.
Nótese que en los siguientes
            y=(double)rand()/(double)RAND_MAX;   // lanzamientos estos números serán reescritos.

            x=x*r;                         // Dilato estos números hasta el radio. Ahora van de 0 a "r". Estas
son las coordenadas
            y=y*r;                         // en las que ha caido un dardo.

            if (x*x+y*y<r*r) {              // Compruebo si el dardo está o no dentro del circulo. Si es así,
c aumentará en uno.
```

```
            c++;
          }
       }                              // FIN Segundo BUCLE

       pi_ar[j]=4*c/N;                    // Calculo el pi generado en esta tanda y lo almaceno.
       c=0;                             // Inicializo a cero para la siguiente tanda de disparos.

   }                                  // FIN Primer BUCLE

   double pi=0;            // Defino pi y el error de pi. Los inicializo a cero por el método para obtener
la media y la SD.
   double err=0;

   for (int j=0; j<cota; j++) {
      pi = pi_ar[j]/cota + pi;               // Hago la media de todos los pi's calculados
   }

   for (int j=0; j<cota; j++) {
      err = err + pow(pi-pi_ar[j],2)/cota;     // Calculo la desviación estándar de los pi's calculados.
Consulta su definición
   }                                  // para más info, pero es sumar estos términos y...

   err = sqrt(err);                        // ... hacer la raiz cuadrada de lo que te salga.


   cout.precision(15); // Estos son el número de digitos que quiero que se expulsen por pantalla.
Puedes aumentarlo si quieres.

   // Sacamos los resultados por pantalla para disfrute del usuario:

   cout<<endl<<"
╔═══════════════════════════════════════════════════════════════════════════════
═══════════════════════════════════════════╗";
   cout<<endl<<"  ║                    "<<"Pi = "<<pi<<"  +/-  "<<err;
   cout<<endl<<"  ║";
   cout<<endl<<"  ║               "<<"o, dicho de otra manera, el valor de pi se encuentra entre";
   cout<<endl<<"  ║                    "<<pi+err<<"   y   "<<pi-err;
   cout<<endl<<"
╚═══════════════════════════════════════════════════════════════════════════════
═══════════════════════════════════════════╝"<<endl<<endl;


   return 0;  // Y hemos terminado. Cerramos el chiringuito.

}
```