//// random.hpp

#ifndef MY\_RANDOM\_HPP

#define MY\_RANDOM\_HPP

#include <random>

class Rand\_double {

public:

using seed\_type = std::random\_device::result\_type;

Rand\_double(double low, double high): dist{low,high} {}

// draw an integer number

double operator()() { return dist(re); }

// choose new random engine seed

void seed(seed\_type s) { re.seed(s); }

private:

std::default\_random\_engine re;

std::uniform\_real\_distribution<double> dist;

};

#include <iomanip>

#include <iostream>

#include <vector>

#include <math.h>

template<typename T\_>

inline constexpr

T\_ pi\_v{3.141592653589793238462643383279502884L};

inline constexpr double pi = pi\_v<double>;

int main()

{

constexpr double rnd\_min = -1.0, rnd\_max = 1.0;

Rand\_double rnd{rnd\_min, rnd\_max};

std::random\_device rd;

rnd.seed(rd());

std::cout << std::fixed << std::setprecision(3);

/\*

double x1 = rnd();

double y1 = rnd();

std::cout << "Point #1: (" << x1 << ", " << y1 << ")\n";

double x2 = rnd();

double y2 = rnd();

std::cout << "Point #2: (" << x2 << ", " << y2 << ")";

std::cout << std::endl;

\*/

int N = 1000;

int r = 0;

int number\_Ni = 0;

while (r != N) {

double x3 = rnd(); double y3 = rnd();

double d = sqrt(abs(x3)\*abs(x3) + abs(y3) \*abs(y3));

//std::cout << "Point: (" << x3 << ", " << y3 << ")\n";

//std::cout << d;

if (d > -1 && d < 1)

++number\_Ni;

++r;

}

double probability = number\_Ni;

double estimate =( probability \* 4) / N;

std::cout << estimate << std::endl;

double relative\_error = M\_PI - estimate;

double percent\_error = (relative\_error \* 100) / M\_PI;

std::cout << "Relative error: "<< relative\_error << std::endl;

std::cout << "Relative percent error: "<< percent\_error << std::endl;

return 0;

}

// end::lab1-3b[]

#endif /\* MY\_RANDOM\_HPP \*/