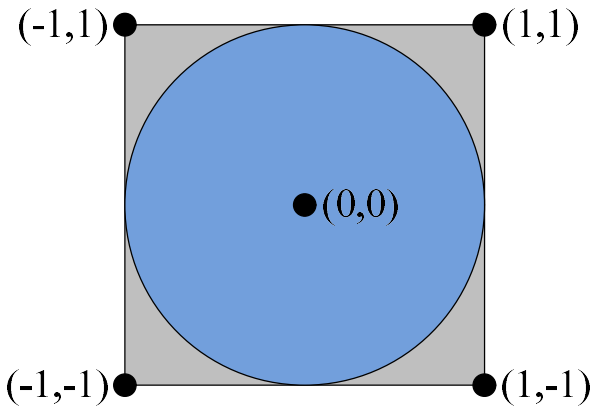
CSCI423/501 Programming Assignment 4

Part A

An interesting way of calculating π is to use a technique known as ***Monte Carlo***, which involves randomization. This technique works as follows: Suppose you have a circle inscribed within a square, as shown in the following figure (Assume that the radius of this circle is 1.) First, generate a series of random points as simple (x, y) coordinates. These points must fall within the Cartesian coordinates that bound the square. Of the total number of random points that are generated, some will occur within the circle. Next, estimate π by performing the following calculation:

$$\pi = 4 \times (\textit{number of points in circle}) / (\textit{total number of points})$$

Write a multithreaded C program of this algorithm that creates a separate thread to generate a number of random points. The thread will count the number of points that occur within the circle and store that result in a global variable. When this thread has exited, the parent thread will calculate and output the estimated value of π. It is worth experimenting with the number of random points generated. As a general rule, the greater the number of points, the closer the approximation to π. So the total number of points generated by the created thread should be no less than 50,000,000.



Hint: You may use the following function to generate a double precision random number between 0 and 1:

```
double random_double()
{
    return random() / ((double)RAND_MAX + 1);
}
```

You may use the following formula to determine if random point (x, y) occurs within

the circle:

```
( sqrt(x*x + y*y) < 1.0 )
```

where the formula gives true if random point (x, y) occurs within the circle, otherwise it gives false.

In addition, you need to include the "`stdlib.h`" and "`math.h`" in your code to use the provided function and formula. When you compile your code, you need to add additional parameter "`-lpthread` " and "`-lm` " as the following example:

```
gcc YourCode.c -lpthread -lm
```

Part B

Now we extend the Part A so that you create several threads, each of which generates random points and determines if the points fall within the circle. Each thread will have to update the global count of all points that fall within the circle. Protect against race conditions on updates to the shared global variable by using mutex locks. The number of created threads will be provided from the command line. Perform necessary error checking to ensure that a positive integer is passed on the command line. The total number of points generated by all the created threads should be no less than 50,000,000. Following are some running examples, assuming the compiled program named b.out:

```
./b.out
Usage: ./b.out <number of threads>

./b.out 10 20
Usage: ./b.out <number of threads>

./b.out -3
<number of threads> should be a positive integer

./b.out 10
Pi = 3.141446
```