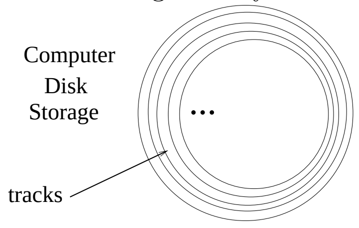
Lab 6 - Using Computer Programs to Solve Problems

Our problem today is to consider a computer disk drive and a sequence of requests (read operations) for information from that disk. Information on a computer disk is stored in a set of concentric tracks on the disk. The track geometry is illustrated below:



A read/write head can move across the face of the disk to position itself above any of the tracks. When a request for information comes in, it is usually for information which is located at some track other than the current position. The disk is mechanical, so it takes some time for a stepping motor to move the head to the correct track. Once at the correct track, the system must also wait for a rotational delay. The requested information is located somewhere along the current track, but we must wait for the rotation of the disk to bring the information under the read/write head. The operation of getting the head correctly positioned so that it can perform a read (or a write) is called a **seek operation**.

Suppose our disk has 1024 tracks numbered 0 to 1023. Let us also assume that requests for information come in a random sequence, requiring at each step to move to a new random position in the range from 0 to 1023. The question we seek to answer in this lab is:

On average, how far (how many tracks) must the read/write head move to satisfy each request ?

To answer this question we will write a C program that simulates the actions of the disk on receiving a sequence of requests. Our C program will generate a random number in the range 0 to 1023 to represent the current position of the read/write head. We then enter a loop in which we generate another random number representing the next track to which the head must go. The distance travelled is the absolute value of the difference between the two track locations. In the loop, we accumulate a total of the distance travelled. At the end, we divide by the number of times through the loop to compute an average distance. A pseudocode algorithm is given below:

```
n = 1000000 ; /* One million. */
i = 0 ; /* i is a loop counter.
total = 0 ; /* total keeps track of the total distance travelled.
current = a random number in the range 0 to 1023 ;
while ( i < n ) {
    next = a new random number in the range 0 to 1023 ;
    total = total + absolute value of ( current - next ) ;
    current = next ; /* Update the current position. */
    i = i + 1 ; /* Count the iteration. */
}
output (total/n) ;
```

Generating a random number

We can generate a random number using a call the the library function `rand()`. Here is a complete example:

```
#include <stdio.h>
#include <stdlib.h>  /* Be sure to include "stdlib.h"   */

int main()
{
    int r, t ;

    r = rand() ;  /* Generates a random number in the range 0 to 32767 */
    printf("%d\n", r ) ;
    /* We use the remainder operator to get a number 0 to 1023.     */
    t = r % 1024 ;    /* Remainder after dividing by 1024.  */
    printf("%d\n", t ) ;
}
```

When I compile and run the program above, I get the numbers 16838 and 454. If I use a loop and repeatedly call `rand()`, and I delete the first "printf()", I get the numbers:

```
454 638 897 107 331 507 482 251 ...
```

Taking an Absolute Value

If I have a positive number, say 55, the absolute value is 55. If I have a negative number, say -37, the absolute value is found by throwing away the minus sign. The absolute value of -37 is 37.

In C, if I have a number `x` that might be negative, I can take its absolute value by changing the sign if it is negative. For example (in C)

```
if ( x < 0 ) {
   x = -x ;
}
```

Computing an Average If we divide two integers (type "int"), an **integer divide** gives us the quotient, but throws away the remainder. For example the C statements:

```
int k ;
k = 7 / 3 ;
```

will give us 2. We can get decimal representation of the ratio `7 / 3` if we use the data type `double`, and use a decimal point in our constants. For example:

```
double k ;
k = 7.0 / 3.0 ;
```

For the project, you will want to make the variable `total` data type `double`. When you print a variable `x` of type double, use the conversion string as follows: `printf( "%lf", x ) ;`