Assignment 4

- Submit all the deliverables to CourSys: https://courses.cs.sfu.ca/
- 10% penalty per day late; assignments not accepted after 2 days late (20% penalty).
- This assignment may be done individually or in pairs. Do not show non-group members your code, do not copy code found online or from previous course offerings, and do not post questions about the assignment online. Direct all questions to the instructor and TA.
- See the marking guide for details on how each part will be marked.

1. Simplified Hard Drive Description

Hard drives are the primary non-volatile storage in a computer. This means that they retain their stored information when the computer is turned off. The performance of a hard drive is an important factor in the performance of a computer. In this assignment, you will create a program to calculate some hard drive performance values.

Standard (non SSD) hard drive have a stack of disks which are used to store data, but for simplicity, we'll only consider a single disk. This disk (Figure 1) has circular tracks which are divided into sectors. The hard drive has a read head (head), which is similar to the arm and needle on record-players; it moves across the disk and reads information as the disk spins beneath it.

There are a number of factors which govern the performance of a hard drive:

1. **Seek Time:** The head must move from the track it was at to the new track to read. For this the arm must move the head either in towards the centre of the disk, or out towards the outside of the disk to position the head over the correct track.

2. **Rotation Delay:** Once the head is in position over the track, the disk must then rotate so that the desired sector is underneath the head. Note that the hard drive is continually spinning at between about 4,000 and 15,000 RPM (revolutions per minute).

3. **Read Time:** The disk must finally rotate through the sector, allowing the head to read all data in the sector. This is similar to the rotation delay, except that during this time the head is actively transferring data.

However, this configuration is complicated by the fact that Figure 1’s depiction of the disk is slightly over-simplified. That figure shows that each track has the identical number of sectors; however, that this makes sectors on the outer tracks longer than sectors on the inner track. To make better use of the hard drive the outer tracks have more sectors than inner tracks, as shown in

Figure 1: Over simplified structure of a single hard drive disk. Yellow ring shows a track, and the blue section is a sector. Image source How Stuff Works. Does not show exact track/sector counts for this assignment.
Figure 2.

For our purposes, we’ll number tracks and sectors as follows\(^1\):

- **Track**: Inner most track is numbered 0.
- **Sectors**: Numbered starting from 0. Sector 0 is on the inner-most track; the last sector is on the outer most disk.

\(^1\) This numbering is a simplification; actual hard drives have their tracks and sectors numbered differently

---

### 2. Performance Calculator

Create a C++ program named `hardDrivePerf.cpp`. Create the following global constant:

- Total number of tracks: an integer, set to 22.

For each function to create, if it says “**pass in**” or “**accept**”, it means a **parameter**.

#### 2.1 Track Functions

Create functions which do the following:

1. Calculate the number of sectors for a specific track.
   - Pass in a track number (integer) and return the number of sectors in that track.
   - A track’s *Track Group* identifies which tracks have the same number of sectors. The track group can be calculated as the track number divided by 4 (rounded down).
   - The number of sectors per track is:
     
     \[
     \text{[# Sectors]} = m \times x + b
     \]
     - \(m\): Additional sectors per *Track Group* = 2
     - \(x\): Track’s group number
     - \(b\): Base sectors per track = 8
   - The function is shown in Figure 3.
   - Hint: Use named constants for each of the constants listed here.

\(^1\) This numbering is a simplification; actual hard drives have their tracks and sectors numbered differently

---

Figure 2: Structure of a hard drive disk showing outer tracks as having more sectors than inner tracks. Image source PC Guide. Does not show exact track/sector counts for this assignment.

Figure 3: Plot showing number of sectors per track.
2 Calculates the total number of sectors on the disk.
   ◦ No parameters to accept; returns the total number of sectors (integer).
   ◦ Hint: Loop through all possible tracks, adding up how many sectors there are on each
     track. To get the sectors on a specific track, call the function you just created (above).

3 Calculate a track number from a sector number.
   ◦ Accept a sector number (integer); return the track number (integer).
   ◦ Hint: Loop through all possible tracks summing up the number of sectors seen. Once
     the sum exceeds the sector number, you have found the track.
   ◦ For example, the first few tracks each have 8 sectors. Imagine we are looking for
     sector 24.
     ▪ Check track 0: Sum = 8, and since 8 is not > 24, keep looking.
     ▪ Check track 1: Sum = 16, and since 16 is not > 24, keep looking.
     ▪ Check track 2: Sum = 24, and since 24 is not > 24, keep looking.
     ▪ Check track 3: Sum = 32, and since 32 is > 24, done: answer is track 3.

2.1.1 Test the Track Specific Functions
Create a function named testTrackFunctions() which calls the functions you just created
and tests each of them. Carefully test each function as you go because if you miss a bug early on,
it can be very hard to find later on. A sample output file is available on the course website. Your
functions must produce clean-looking results, but they need not match the format of the
results posted online. When you create the test function, make sure you call it from main to
execute it.

• Test the function which calculates the number of sectors in a track by looping through all
  possible tracks (from 0 to the maximum number – 1) and display the number of sectors.
  ◦ Note that testTrackFunctions() must call cout; the sub functions it calls must not call cout
    because they each return a value vs printing a value. This distinction is critical.

• Test the function which calculates the total number of sectors by printing the returned
  value to the screen. Again, it's the test function which prints, not the calculation-function.

• Test the function which converts a sector number to a track number.
  ◦ There are quite a few sectors, so you likely don't want to display the track for all of
    them: display the track number for each of the first 25 sectors, and then print the result
    for every 13th (or so) sector after that.
  ◦ Generally when testing, you often cannot test all possible values, but you have to pick
    good values to test.
**2.2 Seek Time**

Write a function which calculates and returns the seek time of the head. The seek time\(^1\) is calculated as:

\[
\text{SeekTime} = |T_{\text{start}} - T_{\text{end}}| \times r
\]

Where:
- \(T_{\text{Start}}\): The starting track number (between 0 and number of tracks – 1). This is where the read head starts before it begins moving.
- \(T_{\text{End}}\): The ending track number (between 0 and number of tracks – 1). This is where the read head ends up after it has finished moving.
- \(r\): The rate that the head moves. Use 0.002 seconds per track.

The vertical bars in the formula mean absolute value. There is a function \(\text{abs}()\) in the \texttt{cstdlib} header.

\[
\text{int } x = \text{abs}(-10); \quad // \text{Returns +10} \\
\text{int } y = \text{abs}(10); \quad // \text{Returns +10}
\]

**2.2.1 Test the Seek Time Function**

Create a function named \texttt{testSeekTime()} which prints to the screen a full table of all possible seek times. Double check your answers against the sample results posted online. Your table format need not match the posted format exactly, but your output should be well formatted.

Note that the table will likely be quite wide. You may need to resize the output window to handle the output. Try switching the output to the “External Terminal” which should allow you to resize the window and view the table format easier.

**2.3 Rotation Times**

A hard drive's performance depends on its rotations per minute (RPM). Write the following functions:

1. Calculate rotations per second from rotations per minute (i.e., it converts RPM to RPS).
   - Pass in the RPM (an integer).
   - Return the RPS (divide RPM by 60).
   - Be careful about data types: RPM will be an integer, but the RPS will be a floating point value.

2. Calculate the average rotational delay time (in seconds).
   - Pass in the RPM (an integer), and return the rotational delay (floating point value).
   - Rotational delay is, on average, half the time it takes for the disk to rotate once.
   - Therefore, it is:
     \[
     \text{Rotational Delay} = \frac{1}{\text{RPS}} / 2.
     \]
   - Be careful about data types.

3. Calculate the time to read one sector (in units of seconds).
   - Pass in the track number (0 to maximum number of tracks – 1), and the disk RPM.
   - Return the time (in seconds) as a floating point value to read the sector.
   - Time to read one sector = \(\frac{1}{\text{RPS}} / \text{[Number Sectors on Track]}\)
   - You will need to call your other functions to get the needed information.

---

\(^1\) This is a simplification. In reality, the seek time is a non-linear function because the head speeds up as it moves.
2.3.1 Test the Rotation Times Functions
Create a function named `testRotationTimes()` which prints to the screen the rotation delays for some different hard drive RPMs. For each RPM, print:
- The RPM
- The RPS
- The rotation delay.
- The time (seconds) to read one sector off the first track (track 0).
- The time (seconds) to read one sector off the last track (number of tracks - 1).
Your output should be neatly formatted.

2.4 Access Times
Create functions which:
1. Calculate the average time to access and read a specific sector. This is the time for the read head to move from its initial location to the target track, spin the disk to the sector, and then read the sector.
   - Pass in:
     1. The starting track number (where the read head started) (an integer);
     2. The sector number to read (an integer); and
     3. The hard-drive RPM (an integer).
   - Return the sum of: seek time + average rotation delay + sector read time.
   - Hint: sector read time in this formula is the target track number, not the initial track.

2. Calculate the average access and read time for all possible situations.
   - Pass in the RPM of the hard drive (an integer).
   - For all possible starting tracks, and for all possible sectors on the whole disk (between 0 and the total number of sectors on the disk), sum up the access and read times (function #1 in this section). Then, divide this sum by the total number of combinations computed.
   - Hint: Loop through all tracks (the starting track). Then use a nested loop that goes through all possible sectors. For each pair, call the above function to compute the time. Count the number of combinations so you can calculate the average value.
   - Be careful about data types.

3. Calculate the average access and read time using a randomized approach.\(^1\)
   - Pass in the RPM of the hard drive.
   - For each of a large number of trials (for example, 1,000,000 trials) pick:
     1. A random starting track (between 0 to the number of tracks – 1), as well as
     2. A random sector to read (between 0 and the maximum number of sectors - 1)
   - Over all your random trials, sum up the access and read time, and divide it by the number of trials. This will give you the average per trial.
   - Since this is a randomized approach, randomize by the timer first.
   - Ideally, the result calculated by this function should be very similar to the result calculated by the non-random approach.

---
\(1\) This is called a Monte Carlo simulation. The idea is to let random sampling pick the data points instead of trying to calculate an answer exactly. Here it's possible to calculate the answer directly, but in many cases it is not as easy to come to an exact solution.
2.4.1 Test the Access Time Functions
Create a function named `testAccessTimes()` which tests:

- Calculating the access and read time for specific combinations. There are a huge number of possible combinations to test, so just test a few.
  - Create nested loops for the RPM, source track, and sector. For each, have it go through just a few of the possible values.
  - For each combination, print out a row in a table to display the read time. See sample output for some ideas on which values to use.

- Loop through different RPMs and display both the average access and the read times for each RPM. Display the value calculated via both the non-random algorithm, and the random algorithm. Compare your results against the sample output.
  - Your randomized times may be slightly different because the values change each time.

2.5 Display Hard Drive Statistics
Write a function which prompts the user to enter an RPM value.

- Ensure the value entered is between 1 and 50000 by looping if the user enters a value that's out of this range. Assume the user enters a valid integer.
- Nothing is passed to the function; it returns the RPM value the user entered.
- Call this function from `main()` and store the returned value in a variable.

Write a function which prints some statistics about the hard drive.

- Pass in the RPM which you read in from the user (above).
- The following shows the combined output for the prompt and display.

```
Enter HD rotations per minute (RPM): 0
Error: Value must be between 1 and 50000.
Enter HD rotations per minute (RPM): -10
Error: Value must be between 1 and 50000.
Enter HD rotations per minute (RPM): 50001
Error: Value must be between 1 and 50000.
Enter HD rotations per minute (RPM): 9919452
Error: Value must be between 1 and 50000.
Enter HD rotations per minute (RPM): 4200

Hard Drive Statistics:
----------------------
Number of tracks:             22
Total number of sectors:      276
Revolutions per minute (RPM): 4200
Revolutions per second (RPS): 70.0
Rotational delay (seconds):   0.007143
Time to read first sector:    0.001786 seconds
Time to read last sector:     0.000794 seconds
Average read time:            0.022944 seconds
Average (random) read time:   0.022869 seconds
```

- Note that most of the values are calculated by the functions you have already written.
- The “Time to read first sector” is reading a sector off track 0, the “Time to read last
sector” is reading a sector off the last track.

3. Deliverables

Your final program should do the following
1. Call each of the test functions described above. The output format need not match the sample output exactly, but it should be well formatted and clear.
2. Ask the user for an RPM.
3. Display the statistics for that RPM.

Constraints
1. You must not have any global variables (global constants are OK).
2. Your calculation functions must not print anything to the screen.
3. If the number of tracks is changed from 22 to something like 42, only one line of code should have to change: the line that defines the constant storing the number of tracks on the hard drive. See sample out for 42 tracks.

Submit the items listed below to the CourSys: https://courses.cs.sfu.ca/
1. hardDrivePerf.cpp

To submit, create a group in CourSys (even if you are working alone). You can do this under this assignment's activity, look for the link in the top right corner. Only one member of your group submits the assignment to CourSys.

Please remember that all submissions will automatically be compared for unexplainable similarities. This comparison will also include similar assignments from previous semesters and programs on the internet. Please review the notes from lecture on the expectations for academic honesty.