# Concepts of Parallel and Distributed Systems
# CSCI-251

v1.7

## Project 1: Disk Usage

### 1 Goals

You may be familiar with the disk usage program available in most Linux/Unix operating systems. For those not familiar with the disk usage tool, it is designed to tell you what files and folders are using the space on your system. While it is possible to run it on the entirety of the disk, it is more common to run in a specific directory (such as your home directory) to determine which files/ folders may be causing you to encroach on your given quota.

The following is a subset of the data generated when I run *du -h* (Linux command) in my home directory on the CS machines.

```
211K    ./public_html/2165/csci.541/Labs/HashTable/doc/search
605K    ./public_html/2165/csci.541/Labs/HashTable/doc
614K    ./public_html/2165/csci.541/Labs/HashTable
14M     ./public_html/2165/csci.541/Labs
14M     ./public_html/2165/csci.541
201K    ./public_html/2165/csci.251
14M     ./public_html/2165
```

### 2 Overview

You will be required to write a simplified version of *du* using the .net core framework. This means that it can run in Linux/Mac/Windows equally well. It also means you can develop your program on any of the aforementioned platforms using any editor you like.

Your program will be required to print out the number of files, folders and total size given an input directory . It should also print the number of image files and their total size if image files(s) exist in the directory

In order to make grading easier, you MUST use the .net SDK v7. It is available here: https://dotnet.microsoft.com/download.

### 3 Design

This project must be an Object Oriented application that works on any common platform (Windows/Mac/Linux). You should test your program on multiple OSs, as there may be some nuances that exist between the different Operating Systems with regards to handling files that you have to deal with.

The application itself must be a command line application. It will accept 2 command line arguments, (a character and a path) per the following help:

```
Usage: du [-s] [-d] [-b] <path>
Summarize disk usage of the set of FILES, recursively for directories.
You MUST specify one of the parameters, -s, -d, or -b
-s       Run in single threaded mode
-d       Run in parallel mode (uses all available processors)
-b       Run in both parallel and single threaded mode.
         Runs parallel followed by sequential mode
```

**You must include this help message in your application when invalid command line arguments are given.**

If the user's specification to the disk usage program is in both parallel and sequential mode, you should run the parallel version first. The result to display are not the entire *du* results, instead you should display results that look like the following:

```
> dotnet run -b /Users/jsb/Dropbox
Directory '/Users/jsb/Dropbox':

Parallel Calculated in: 7.5724931s
76,133 folders, 332,707 files, 42,299,411,348 bytes
```

200 image files, 4, 224,4340 bytes

```
Sequential Calculated in: 34.0341592s
76,133 folders, 332,707 files, 42,299,411,348 bytes
```

200 image files, 4, 224,4340 bytes

All you need to do is calculate the time it took to run, the total folders, the total files, the total size, total image files and the total image file size (if they exist in the directory). Display them in the format above.
If there are no image files in the input directory, your program should state that "*no image files found in the directory*".

We are not concerned about the numbers that add up to the totals, as that is usually the bottleneck (displaying the results). We want to compare the parallel and sequential versions, not the output time.

When writing your code, you need not explicitly create threads. Instead, you should write the parallel version using Parallel ForEach statements, while the Sequential version should use a traditional ForEach. Calculating the parallel and sequential version should both be in their own methods, and there will be some nuances you will have to deal with with the arguments to each of the functions you write.

Because you may not have access to all files on a system, you can ignore the files and directories that you do not have access to, in your calculation. In other words, if you attempt to run your program on C:\ on Windows, you should skip over any files you try to read in C:\Windows \System32, which should generate an error when you try to read those files. If you do not handle these conditions explicitly, your program will crash and you will lose points.

## 4 Testing

The application will be tested in both Windows 11, Mac OS 10 and Ubuntu Linux 20.04. Your program should not crash at any time (add exception handling). Be sure to test it in multiple different paths with varied command line arguments. You may also safely ignore any files and directories that can't be read (for example, system/read only files).

Sample outputs are provided for you in text format, so that you can compare outputs for different scenarios to ensure they match. You can find them here:
https://cs.rit.edu/~iige/project_1_sample_run.txt

## 5 Grading

- For full credit, your parallel version must run faster than the sequential version on large data sets (similar to the example provided). Your code should not print approximated time or rounded off time. There must be a clear and distinct difference in time between the versions.
- Some implementations will have a parallel version faster than the sequential version, but this will not be the optimal solution; for full credit, you must implement an optimal solution. It is an exercise for you to find an optimal solution. While you may have a solution that works, you should try other implementations/classes to find out which, if any are faster.
- Your times must also be in the order of magnitude of what I provided in the example. Note, this was run on a Quad Core i5 with an SSD drive.
- Your output must match the output provided, especially the formatting of numbers.
- The counts for parallel and sequential must match.

Each project will be assigned the following points:

- Help message (20 points)
- Parallel mode is faster than Sequential mode - Optimal solution (20 points)
- The parallel Count of files and folders are the same as the Sequential Count (15 points)
- The parallel Count of image files are the same as the Sequential Count (15 points)
- The use of Parallel.ForEach for the parallel method, Foreach for the Sequential method (20 points)
- Exception Handling (10 points)

## 6 Submission

Zip up your solution in a flat file (no folders) called project1.zip. The zip file should contain the following files:

`Program.cs`
`du.csproj`
txt file (optional; only required if a portion of your code was retrieved from Generative AI tools)

I will test your project by running:

```
dotnet run
```

and running the executable that was generated. You should do the same to ensure your submission works. Submit the zipped file to the myCourses dropbox before the due date.