# Story mode

Welcome to the **Story Mode** of this course. At this point, you must be wondering what is the practical application of the knowledge you have learned so far or you are about to learn. How would this **help** you **build** something? Where is it **applicable** and how do you connect the dots that represent the rest of your knowledge? Well, this is exactly the kind of questions we are going to try and answer while going through the story mode. We are calling it “story mode”, because the idea behind all of this is to use every single piece of knowledge we get from the lectures and apply it in order to make a whole product and all of this is done in small steps. And each one has its own purpose for the whole “story”.

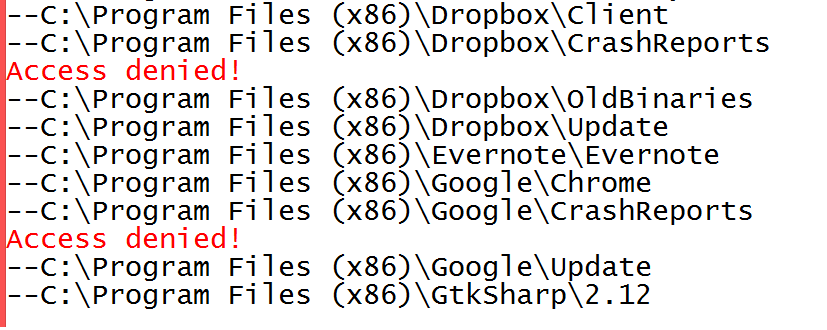
The project we are going to build is a **bash** (**command** **interpreter**), however it has to be with some special functions, in order to be adapted for the needs of **SoftUni**.

This is why we are going to call it **BashSoft.**

Like we mentioned above, the process will essentially consist of many small steps which when put together will result in a real-world application. The very first step we will need to complete is find some way to work with the data currently stored on our computer. After all, if we want a bash-like application to have any real use, we would want to give it some data (files) to work with. To provide that data we first need to retrieve it, i.e. reach all the files we would eventually need. How do we go about this? Even a journey of a million miles begins with a single step, so let’s dive in.

## Stacks and Queues

**First** of all we will probably need a way for traversing the file system, and this is where the [**Queues**](http://www.dotnetperls.com/queue)from the first lecture “Stacks and Queues”come in handy. There we will learn about a very popular algorithm called [**Breadth-first search (or BFS)**](https://en.wikipedia.org/wiki/Breadth-first_search), which in our case will give us the ability to traverse not only one folder but each folder in the current subfolders and then each subfolder in each subfolder of the first subfolder etc.

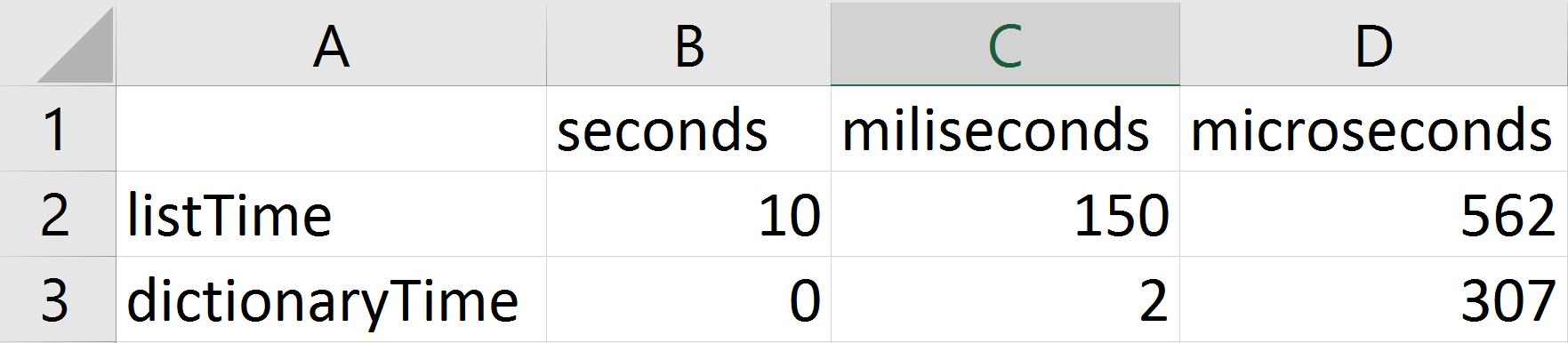


When we have the ability to traverse the file system, we have a lot of options what to do with the data. One way this can become handy is, when we have to deal with direct file manipulation at later stages of our **BashSoft.**

## Sets and Dictionaries

**The next** little step we have to take is to start thinking about the **speed** of our app. You might think “But we have nothing yet, and we worry about speed?” While true, it’s a *very* good idea to use the proper data structures as the foundation for our bash in order to avoid rebuilding the whole thing at later stages.

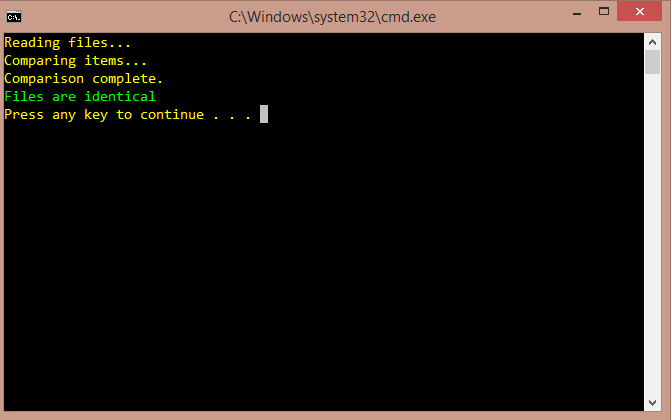
Also, **no one** likes slow programs, so you need to make **fast operations** in your applications and have fast and efficient data structures like **Sets and Dictionaries**. The real strength of using structures like these is probably most obviously shown below



This picture represents what is the speed difference between searching in a List/Dictionary with 50000 values, 50000 times. This is why you might have to figure out how these data structures work in order to write your application to run really fast, because otherwise the students using your software are going to have to wait.

## Files and Directories

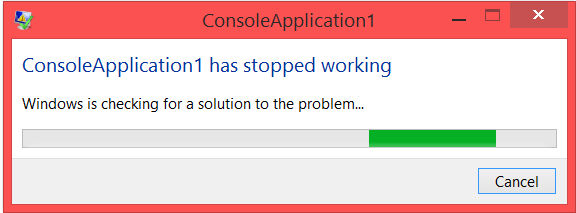
**So far**, we can traverse and **search**/**store** data. Pretty solid foundation for every real-world application. Next, let’s add some custom functionality - reading files, writing in files, creating and deleting files. A pretty important part when we want to do *anything* with the data stored on our computer. Since we have learned how to retrieve and store data, now we will start using it. We will do that by implementing our very own **judge system**. It will also be the very first actually functional part of our bash application.



## Exception Handling

By the time we implement our next functionality, **Exception Handling**, we are going to see how many things can go unexpectedly wrong. If you are writing a bigger application now would be the moment to start “**catching**” such **unexpected** events and **handling** them in such a way that the user is not disgusted of the user experience of our application. An example of an **unexpected** event is when we try to open and read the content of folders and files we don’t have the rights for.

After all, rarely do we feel good when we see this:



## Strings and Text Processing

Next, we are going to apply **string** **manipulation** methods in order to make our command interpreter for the **BashSoft** complete the previous implemented actions, using only commands like “**ls path”** for traversing given folder, **“cmp file1 file2”** to compare two files and so on. We will see how important strings are in the connection with the user.

We are going to implement the operations that we currently have and later implement the ones we’re yet to build.

BASH:

open file in current folder – **open fileName**

make directory in current folder - **mkdir directoryName**

traverse current directory in some depth - **ls depth**

compare content of two files given their absolute path - **cmp path1 path2**

change current directory given a relative path – **changeDirRel relative path or “. .” for returning to the previous folder**

change current directory given an absolute path – **changeDir absolutePath**

read students data base from a given file in the current folder – **readDb fileName**

filter students from given course by some criteria and take some of them - **filter courseName excellent/average/poor take 2/10/42/all**

order students from given course ascending/descending and take some of them – **order courseName ascending/descending take 5/17/23/all**

download file – **download pathOfFile**

download file asinchronously – **downloadAsynch pathOfFile**

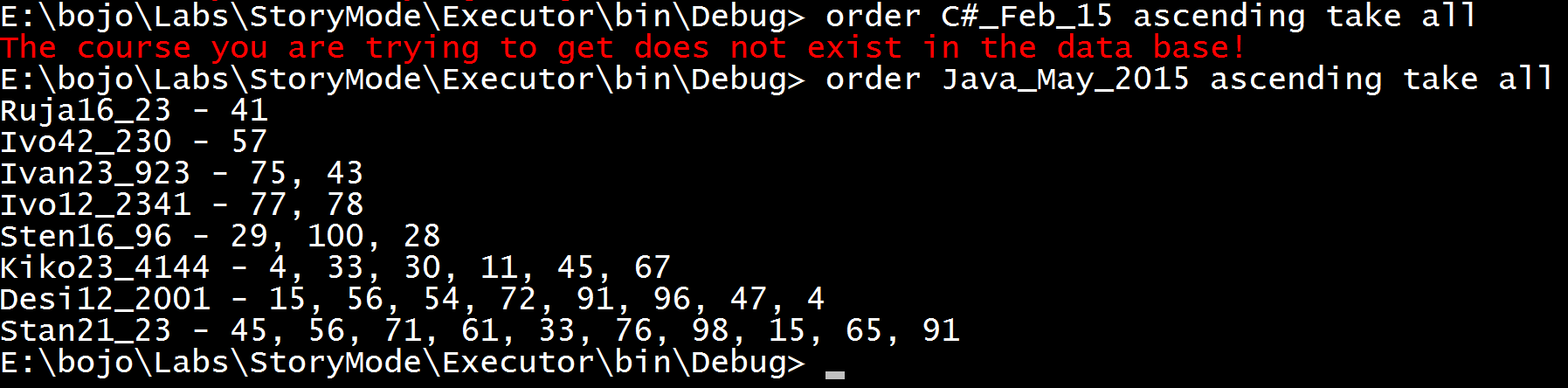
get help – **help**

## Regular expressions

Here we are going to use the power of **regex** to easily **filter** the **input** **data** that is being filled in the data structure, so that we are sure that only entries which **match** a given **format** (and therefore are **valid**) **will** be **inserted**. We will see that it is easier to be done with regex than strings and after this part, we will have a very strict pattern for the possible usernames of students and also the possible names of courses.

## Functional programming

By this time, we should have quite a big project with a lot of functionality. Now it’s time to write some helper **functions**, that help the person who calculates the results for a given exam and we are going to need you to read the database and **filter** some students by given **criteria**, or sort them in **ascending** or **descending** order. In the end, you should have functionality looking pretty much like the following:



## LINQ/Streams API**:**

In this part we are **not** **going** **to** **add** any **new** **functionality**, but we will see how **LINQ/Streams API** **saves** **us** **from** **writing** our own functions for **filtering** **and** **sorting** the **data** **structure** and instead we can use the built-in ones. This way we have **less** **code**, it is **more** **understandable** and makes **reading** and **writing** easier.