Data recovery in Visual Basic II

Topics

- file i/o,
- binary data, and
- file representation and file signatures.

Instructions

5. Copy and paste the folder `ImageRecovery_Part4`. Rename the copy `ImageRecovery_Part5`. Launch the VS Express 2013 software and open the project `ImageRecovery_Part5`.

   Take a moment to reflect on your project as it stands (i.e., at the end of Part4). It currently reads the 512 byte blocks of a file in sequence and checks whether or not each block demarks the beginning of a JPEG. If so, it outputs the JPEG number associated with that block as well as the block number itself.

   Ultimately, our goal is to output the blocks associated with each JPEG into separates files, effectively *recovering* the JPEGs that were lost. So, the next step in this process is to create empty files into which we will eventually put the blocks belonging to each JPEG.

   Creating an empty file in Visual Basic is fairly straightforward. All that is required is to open a file that does not already exist, and it will be created. The only catch is that, as in Part1, the file should be opened in `OpenMode.Binary`. Whereas in Part1 we opened `card.raw` with this mode so that we could read its bytes directly, now we are opening the JPEG files with this mode so that we can put the blocks directly into them.

   So, update the action for `btnRecover` so that every time a block that demarks the beginning of a JPEG is input from `card.raw`, an empty file is created that will eventually represent the JPEG to which the block belongs. Along with this, update the output of your project so that it outputs a more meaningful message. Your project's output should have the following format:

```
Recovering 01.jpeg...
Recovering 02.jpeg...
Recovering 03.jpeg...
Recovering 04.jpeg...
Recovering 05.jpeg...
-----
5 JPEGs were recovered.
```

   Note: Up to this point, when dealing with files, we have used channel 1 exclusively to open files. However, in this problem, we will need to open multiple files simultaneously (the input file and each of the JPEG files). As a result, we will need to employ more than one channel. My suggestion to you is to reserve channel 1 for the input channel, just as we have done in the past. For the output files (i.e., the JPEG files), you can use channel numbers starting from 2. *[Hint: this would be a good time to declare another variable to keep track of the next file channel to use.]*

   Remember, once a file has been opened in Visual Basic, it is very important that we close it. This is especially true for files to which we are outputting data, since forgetting to close the file may result in some data not being saved to the file. To close all open files, Visual Basic allows the the `FileClose()` function to be used without any input, so:

```
FileClose()
```

   instead of

```
FileClose(1)
```

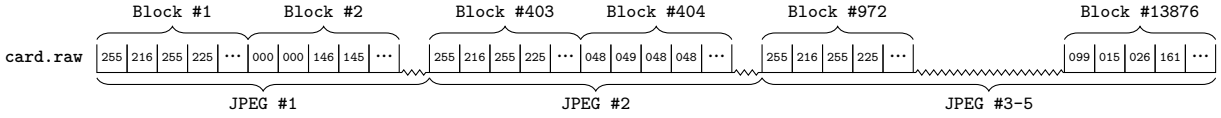
Checkpoint 5 (80/100)

- ☐ project creates 5 empty files named ##.jpeg starting from 01
- ☐ project produces correctly formatted output according to the example above

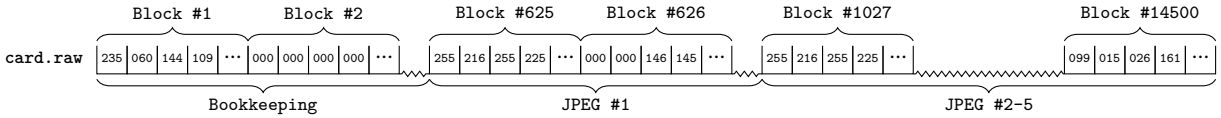
6. Copy and paste the folder `ImageRecovery_Part5`. Rename the copy `ImageRecovery_Part6`. Launch the VS Express 2013 software and open the project `ImageRecovery_Part6`.

   Run your project `ImageRecovery_Part4`. Which block does the first JPEG begin at? Not Block #1. This means that `card.raw` contains a bunch of data before the JPEGs begin. This data belongs to the FAT filesystem and should be ignored by our project.

   Let's assume for a minute that `card.raw` contained no data before the beginning of the JPEG files, i.e., it looked like



   instead of



   If that were the case, then the algorithm for a recovery program might look something like:

```
1  open the input file
2
3  while not the end of the input file do
4  |   read a block from the input file
5  |
6  |   if the block demarks the beginning of a JPEG file then
7  |   |   create an empty file for the JPEG that this block starts
8  |   end
9  |
10 |   put the block into the last file created in line 7
11 end
12
13 close all open files
```

   So, for simplicity's sake, let's cheat the system for the time being and use the following line to fast-forward through `card.raw` to the point where the JPEGs begin.

```
Seek(1, 319489)
```

   This way we can treat `card.raw` as if it only contained JPEGs, i.e., no bookkeeping information that must be ignored. You should recognize 319489 as the byte where block #625 (i.e., the block where the JPEG #1 begins in the second example above) begins, since the first 624 blocks × 512 bytes per block = 319488 bytes. If we include that line of code immediately after opening `card.raw`, then we can assume that the first block that is input from the file in the loop above will be the first block of JPEG #1.

   Lastly, putting a block of 512 bytes into a file is as easy as reading them from the file. All you need to say is:

```
FilePut(2, b)
```

   You will likely have to change the input to the `FilePut()` function, since not all of your JPEGs are using channel 2. Likewise, if you are not using the variable called `b` to store your blocks, then you must change that also.

So, using the algorithm above and the provided Visual Basic code, update the action for `btnRecover` so that it does not just reate empty files, but also puts the blocks that it inputs into the correct files, thus recovering the lost JPEGs.

Checkpoint 6 (90/100)

- ☐ project recovers all 5 JPEGs, named ##.jpeg starting from 01
- ☐ project produces correctly formatted output according to the example from Part 5

7. Copy and paste the folder `ImageRecovery_Part6`. Rename the copy `ImageRecovery_Part7`. Launch the VS Express 2013 software and open the project `ImageRecovery_Part7`.

   Next, let's take care of that pesky assumption that our loop will start inputting blocks from `card.raw` at the beginning of the first JPEG. In other words, let's remove that magical line of Visual Basic, namely `Seek(1, 319489)`, that makes our assumption correct.

   How does removing this line make the algorithm outlined in Part 6 incorrect? If the first block is not guaranteed to belong to the first JPEG, then what will happen on line 10 for the first block? If you did not remove that line of code yet, do so and then run your project to see what happens. Why do you think this happens?

   [Hint: think about the first time the loop repeats; will the condition of the if statement be true or false? If it is false, then no file will be created, so where will the bytes that were just read be put?]

   Update the action for `btnRecover` so it only calls `FilePut()` after the first JPEG has been encountered, i.e., the first empty file has been created. That way, there will always be a place for the bytes to go.

Checkpoint 7 (95/100)

- ☐ project recovers all 5 JPEGs, named ##.jpeg starting from 01
- ☐ project ignores (not skips) bookkeeping bytes found at the beginning of `card.raw`
- ☐ project produces correctly formatted output according to the example from Part 5

8. Copy and paste the folder `ImageRecovery_Part7`. Rename the copy `ImageRecovery_Part8`. Launch the VS Express 2013 software and open the project `ImageRecovery_Part8`.

   Since this is the last lab dedicated to Visual Basic, let's finish in style. One of the marks of a good programmer is a program that uses no more resources than necessary. In fact, in the early days of computing, programmers went to great lengths to ensure that their programs did not use any more resources than ABSOLUTELY necessary, because resources were much more scarce.

   Unfortunately for you, the program that you have written so far is very wasteful. As written, it uses a different file channel for each of the JPEGs that it recovers. Only once all JPEGs have been recovered does it close any of the file channels. This is VERY wasteful, since at a given time, only one of those files is being written to, i.e., only one JPEG is being recovered at a time.

   Your final Visual Basic task is to improve your project so that it uses only two file channels, namely 1 and 2, for `card.raw` and for the output JPEG that is being recovered, respectfully. This will mean that instead of closing all of the files using a single call to `FileClose()` with no input at the end of the button action, you will need to close the output file each time that you finish recovering a JPEG so that you can reuse the file channel for the next JPEG file to be recovered.

Checkpoint 8 (100/100)

- ☐ project recovers all 5 JPEGs, named ##.jpeg starting from 01
- ☐ project uses only two file channels for input and output
- ☐ project produces correctly formatted output according to the example from Part 5

Submission Instructions

Your `M:\CS130\Labs\Lab08_YourLastName_YourFirstName` folder should contain your solutions to this and Tuesday's lab.

To submit your work, copy this folder and paste it to `N:/Handins/CS130/Lab08_ImageRecovery` PRIOR to midnight tonight. Submissions received after 11:59pm tonight will be considered late and will receive a grade of 0.

You are not allowed to seek help from TAs on this lab outside lab time. TAs have been specifically instructed NOT to provide any help so please refrain from this activity.