File splitter

This assignment will give your practice implementing working with C file input/output, working with files, C/C++ mixing.

Incomplete interface (doesn't allow C/C++ mixing)

```
#ifndef SPLITTER_H
#define SPLITTER_H
enum {E_BAD_SOURCE=1, E_BAD_DESTINATION, E_NO_MEMORY, E_NO_ACTION, E_SMALL_SIZE};

int SplitFile(char const *  filename,  char const * output, size_t size);
int JoinFiles(char** filenames, int num_files, char const * output);
#endif
```

Goal: create file splitter/ file joiner functionality that allows the user to

- split a file into pieces of a given size – imaging a situation when you need to transfer 1Gb file named "`filename`" from one computer to another using only 128Mb=134217728 bytes flash drive. A call
  `SplitFile( "file2split", "chunk_", 134217728)`
  opens file "`file2split`" and creates files `chunk_0001, chunk_0002, ...` all chunks except possibly the last one have size `134217728` bytes.
- After copying files from the flash drives to a harddrive of the destination computer, combine the files into a single file – in the above example merging all chunks into the original file.
  Assuming you have an array `chunks` containing C-style strings { `"chunk_0001", "chunk_0002", "chunk_0002" }`
  `JoinFiles( chunks, 3, "combined_file" );`
  merges files "chunk_0001", "chunk_0002", "chunk_0003" into a file named "combined_file". The size of "combined_file" should be exactly the sum of the sizes of "chunk_0001", "chunk_0002", "chunk_0003".
- **Drivers** - "`driver.c`" and "`driver.cpp`" which accept command line arguments
  - command line switches:
    - -j join
    - -s split, followed by the desired chunk size
    - -i input filename(s)
    - -o
      - output filname when used with -j
      - chunk prefix when used with -s
- **Makefile:**
  - gcc_c – GNU C compiler (used for grading)
  - gcc_cpp – GNU C++ compiler (used for grading)
  - gcc_c_cpp – GNU mixed C/C++ compiler (used for grading)
  - msc_c – MicroSoft C compiler
  - msc_cpp – MicroSoft C++ compiler
  - msc_c_cpp – MicroSoft mixed C/C++ compiler
  - 0,1,2 – simple online run-time tests (run make 0, make 1, make 2. If filee difference0 (or 1,2) is created – you have errors.
  - mem0,mem1,mem2 – online run-time memory tests (run make mem0 ....).
  - Examples:

- To compile use
  `make gcc_c`
  Note that the name of the executable will be the <name of the target>.exe.
- To run:
  ```
  ./gcc_c.exe -s 100 -i 120-byte-file -o ddd_
  ./gcc_c.exe -j -o combinedfile -i chunk1 chunk2
  ```

Implementaions notes and requirements:

1. assume binary input and output and use `fread/fwrite` (original file and reconstructed should be identical to a single byte, including new line characters)
2. **in function SplitFile use read buffer size set to the minimum of the desired chunk size and 4K=4096 .**
3. buffer should be allocated dynamically using malloc,
4. I/O functions like `fseek, rewind`, etc should **not** be used.
5. Return codes:
   A) E_BAD_SOURCE – if input file(s) cannot be open
   B) E_BAD_DESTINATION – if output file(s) cannot be open
   C) E_NO_MEMORY – if malloc fails
   D) E_NO_ACTION – used in driver only
   E) E_SMALL_SIZE – currently not used
6. chunk suffix is a 4 digit number padded with 0's, see provided implementation in `splitter.c.`
7. operating system will expand wildcards for you, so that if you type in the command line
   `./prog.exe -j -o collected -i chunk*`
   and the current folder contains files chunk1, chunk2, chunk3, chunk4, then `argv` array will contain the following:
   ```
   ./prog.exe
   -j
   -o
   collected
   -i
   chunk1
   chunk2
   chunk3
   chunk4
   ```
   see sample implementation file.
8. make sure your code compiles with all targets (see Makefile)
9. make sure you do not have memory errors or leaks
10. Doxygen comments are required
11. Make sure you can split and combine binary and text files.
12. Online I only run simple tests. When grading there will be more tests - it is your responsibility to think what kind additional testing is appropriate.

Usefull tools: (*nix,Cygwin – but there are Windows equivalents)

- "od" -- octal dump (Cygwin's coreutils package), which may help you to examine the produced file. Usage

  ```
  od -x filename
  ```

  output

  ```
  0000000 6923 6e66 6564 2066 4946 454c 414e 454d
  0000020 485f 0a0d
  0000024
  ```

  where the left column contains addresses, all other columns are ASCII codes in hexadecimal format corresponding to characters in the file, note that "6923" represents 2 characters – 23 and 69 (**in this order!!!**) which are '#' and 'i'. Also note that the file was in Windows text format, since we can clearly see "0a0d " in the end, which is Windows CRLF (carriage return "0d ", line feed "0a") line terminator.
- "diff" – see a link on class web-page for more information.

  Very basic usage:

  ```
  diff file1 file2
  ```

  output will look like

  ```
  2c2
  < a
  ---
  > b
  ```

  showing that there is a difference in line 2, which is 'a' in the first file and 'b' in the second. (I used 2 files with the same first line and second line in both files was a single character)

  ```
  diff file1 file2 –strip-trailing-cr
  ```

  "--strip-trailing-cr" flag makes "diff" to NOT pay attention to the newline style (do not use this flag for this assignment, newlines have to be preserved)

  If there is a single line of output from "diff" - there is(are) differences, and you should figure out were exactly they are. Use

  ```
  diff file1 file2 -y
  ```

  "-y" flag makes "diff" to display the output in two columns, marking lines that differ with a bar '|'.

  ```
  1                           1
  a                     |     b
  ```

  Hint: 2 columns will not fit into a terminal, so you should either resize the terminal (click icon "C:\", then Properties->Layout->Window Size), or redirect output into a file (add " > difference.txt" in the end of diff line, and then open file "difference.txt" in any editor)

Submission:

submit electronically 2 files:

```
splitter.c
splitter.h
```