OS Homework 1 Problem 3

We are tasked with writing a program that takes input files one at a time and concatenates them into an output file. These input files and the output file are specified as arguments to the program.

For the sake of demonstration, the following files and their contents listed below will be used in the program. If the content cell entry has quotes around it, that means the text enclosed within the quotes is what is actually in the file. Otherwise, the content cell entry is merely describing what's in the respective file:

Filename	Content
Input1.txt	“helloworld"
Input2.txt	“testing123"
rinput1.txt	10000 Random bytes generated from /dev/urandom
rinput2.txt	10000 Random bytes generated from /dev/urandom
-	“this file is named with a "-””

The source code is contained in `kitty.c` .No makefile was used to generate the executable. Instead, the following command was used:

```
$ gcc kitty.c -o kittyC
```

To run the program, enter arguments as assigned in the problem sheet:

```
kitty [-o outfile] infile1 [..infile2..]
```

TEST CASES

Notes about testing procedure:

Assignment sheet requests that we print messages to stderr after read/write and for errors. My program prints stderr to console, which may not be intended. My previous experience with Cooper program submissions requested that all prints to console be suppressed, but I left it on in this case.

Assignment sheet states that std_in is to be read until the EOF character. To facilitate testing, I pressed Enter instead of Ctrl+D, which inserted an extra '\n' character at the end of each std_in input. This is a minor difference and should not impact the program's function when Ctrl+D is used instead.

Assignment sheet states that upon an error to exit the program with -1. However, with online research, I found that the macro EXIT_FAILLURE is the preferred returned value upon failure of a program. The difference is that EXIT_FAILURE is equal to 1, not -1. I know I did not follow exact project specifications in this aspect, but since my method adheres to indusry standards I think it is a negligible difference.

I was only able to replicate a open system call error with my current knowledge. However, code for detecting read, write, close, and lseek errors are implemented in the program.

Test case 1

Input:

```
$ kitty input1.txt input2.txt
```

Output:

```
kitty transferred 10 bytes from file input1.txt with 1 read call(s)
and 1 write call(s)


kitty transferred 10 bytes from file input2.txt with 1 read call(s)
and 1 write call(s)

helloworldtesting123
```

Test case 2

Input:

```
$ kitty -o output1.txt input1.txt input2.txt
```

Output:

```
kitty transferred 10 bytes from file input1.txt with 1 read call(s)
and 1 write call(s)
```

```
kitty transferred 10 bytes from file input2.txt with 1 read call(s)
and 1 write call(s)
```

Contents of output1.txt:

```
helloworldtesting123
```

Test case 3

Input:

```
$ kitty input1.txt - input2.txt
```

Output:

```
kitty transferred 10 bytes from file input1.txt with 1 read call(s)
and 1 write call(s)
```

```
I am typing into stdin here
```

```
kitty transferred 28 bytes from file <standard input> with 1 read
call(s) and 1 write call(s)
```

```
kitty transferred 10 bytes from file input2.txt with 1 read call(s)
and 1 write call(s)
```

```
helloworldI am typing into stdin here
```

```
testing123
```

Test case 4

Input:

```
$ kitty -o output1.txt - - input1.txt
```

Output:

```
I am typing into stdin here for the first time
```

```
kitty transferred 47 bytes from file <standard input> with 1 read
call(s) and 1 write call(s)
```

```
I am typing into stdin here for the second time
```

```
kitty transferred 48 bytes from file <standard input> with 1 read call(s) and 1 write call(s)
```

```
kitty transferred 10 bytes from file input1.txt with 1 read call(s) and 1 write call(s)
```

Contents of output1.txt:

```
I am typing into stdin here for the first time

I am typing into stdin here for the second time

helloworld
```

Test case 5

Input:

```
$ kitty - ./-
```

Output:

```
I am typing into stdin here
```

```
kitty transferred 28 bytes from file <standard input> with 1 read call(s) and 1 write call(s)
```

```
kitty transferred 29 bytes from file ./- with 1 read call(s) and 1 write call(s)

I am typing into stdin here

this file is named with a "-"
```

Test case 6

Input:

```
$ kitty input1.txt input2.txt -o output1.txt
```

Output:

```
kitty transferred 10 bytes from file input1.txt with 1 read call(s)
and 1 write call(s)
```

```
kitty transferred 10 bytes from file input2.txt with 1 read call(s)
and 1 write call(s)
```

Contents of output1.txt:

```
helloworldtesting123
```

Test case 7

Input:

```
kitty -o output1.txt rinput1.txt rinput2.txt - - ./-
```

Output:

```
kitty transferred 10000 bytes from file rinput1.txt with 3 read
call(s) and 3 write call(s)
```

```
Warning: input file rinput1.txt is of a binary file type
```

```
kitty transferred 10000 bytes from file rinput2.txt with 3 read
call(s) and 3 write call(s)
```

```
Warning: input file rinput2.txt is of a binary file type
```

```
typing into stdin for the first time
```

```
kitty transferred 37 bytes from file <standard input> with 1 read
call(s) and 1 write call(s)
```

```
typing into stdin for the second time
```

```
kitty transferred 38 bytes from file <standard input> with 1 read
call(s) and 1 write call(s)
```

```
kitty transferred 29 bytes from file ./- with 1 read call(s) and 1
write call(s)
```

Content of output1.txt:

Output has been omitted due to length of file, but this can be recreated for furhter examination

Test case 8 (comparing cat and kitty for random binary files)

kitty command:

Input:

```
$ kitty -o output1.txt rinput1.txt rinput2.txt
```

Output:

```
kitty transferred 10000 bytes from file rinput1.txt with 3 read
call(s) and 3 write call(s)
```

```
Warning: input file rinput1.txt is of a binary file type
```

```
kitty transferred 10000 bytes from file rinput2.txt with 3 read
call(s) and 3 write call(s)
```

```
Warning: input file rinput2.txt is of a binary file type
```

Checksum of output1.txt:

```
$ md5sum output1.txt
```

```
e8276082830ec827e84478e2c7e68ce1  output1.txt
```

cat command:

Input:

```
$ cat rinput1.txt rinput2.txt > output2.txt
```

Output:

None

Checksum of output2.txt:

```
md5sum output2.txt
```

```
e8276082830ec827e84478e2c7e68ce1  output2.txt
```

As you can see by comparing the checksums, these output files are identical.

Error handling test cases

Test case 1 (input3.txt does not exist):

Input:

```
$ kitty -o output1.txt input1.txt input2.txt input3.txt
```

Output:

```
kitty transferred 10 bytes from file input1.txt with 1 read call(s)
and 1 write call(s)
```

```
kitty transferred 10 bytes from file input2.txt with 1 read call(s)
and 1 write call(s)
```

```
Could not open input file input3.txt correctly: No such file or
directory
```

Contents of output1.txt:

```
helloworldtesting123
```