Project 2 Unix Utilities

Running the program(s):

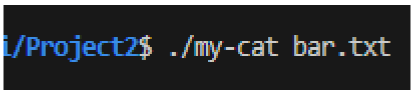
1. Go to the right directory wiht the programs, my-cat.c, my-grep.c, my-zip.c, my-unzip.c
2. Compile the programs:
    a. Gcc -o my-cat my-cat.c -Wall -Werror
    b. Gcc -o my-grep my-grep.c -Wall -Werror
    c. Gcc -o my-zip my-zip.c -Wall -Werror
    d. Gcc -o my-unzip my-unzip.c -Wall -Werror
3. Then run:
    a. ./my-cat
    b. ./my-grep
    c. ./my-zip
    d. ./my-unzip

Different inputs and outputs of the program:

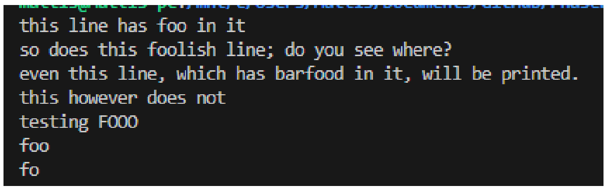
1. ./My-cat [file…]

The my cat program prints the contents of file(s) into your stdout. You can have any amount of files as arguments:

One input:

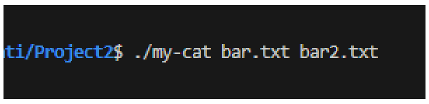


```
i/Project2$ ./my-cat bar.txt
```

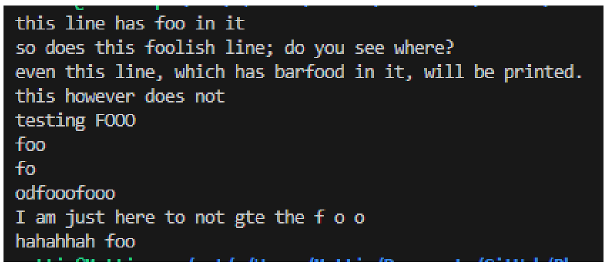


```
this line has foo in it
so does this foolish line; do you see where?
even this line, which has barfood in it, will be printed.
this however does not
testing FOOO
foo
fo
```

Two or more inputs inputs:

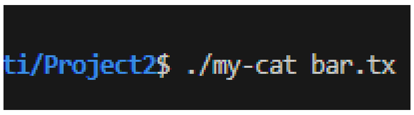


```
ti/Project2$ ./my-cat bar.txt bar2.txt
```

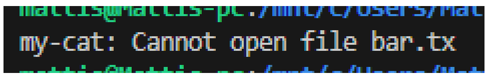


```
this line has foo in it
so does this foolish line; do you see where?
even this line, which has barfood in it, will be printed.
this however does not
testing FOOO
foo
fo
odfooofooo
I am just here to not gte the f o o
hahahhah foo
```

Error cases:



```
ti/Project2$ ./my-cat bar.tx
```



```
my-cat: Cannot open file bar.tx
```

Working principle:

Veeery simple. Open file(s) one by one → read each line and print it → close file and free memory

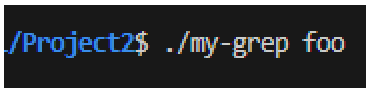
2. ./my-grep searchTerm [file(s)]

The grep program searches a term of your choosing from an input file or stdin depending on the amount of arguments:

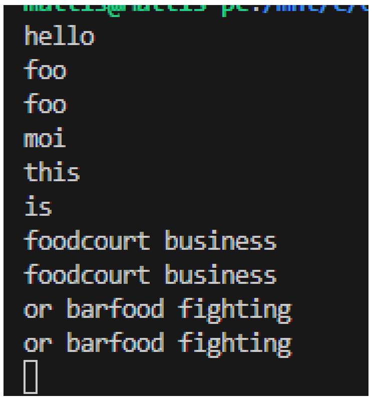
Only search term:

The input is stdin if only the searchterm is given. The program prints the line as soon

as it has the search term in it, so the duplicate lines are printed out. Press CTRL D or CTRL C to exit

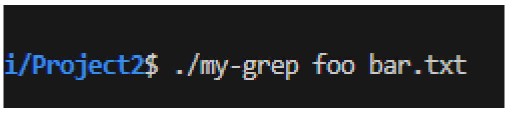


```
/Project2$ ./my-grep foo
```

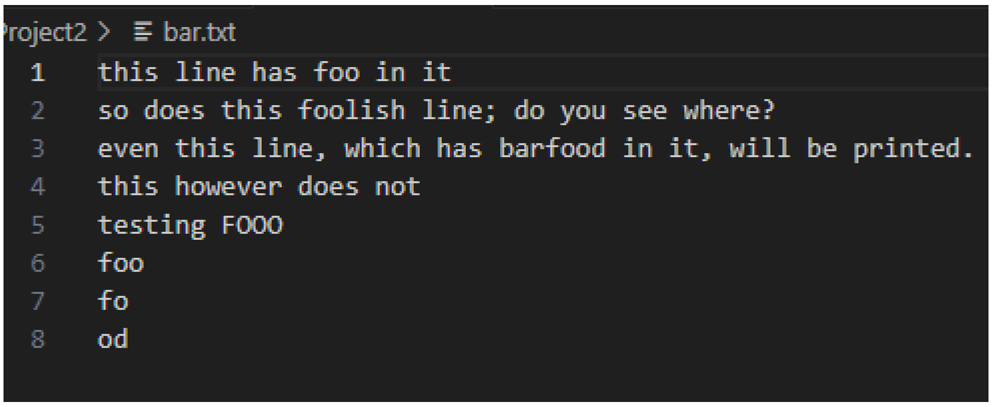


```
hello
foo
foo
moi
this
is
foodcourt business
foodcourt business
or barfood fighting
or barfood fighting
```

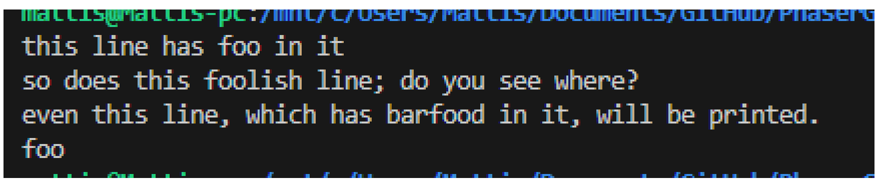
File is given:



```
i/Project2$ ./my-grep foo bar.txt
```



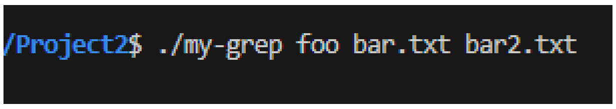
```
roject2 > bar.txt
this line has foo in it
so does this foolish line; do you see where?
even this line, which has barfood in it, will be printed.
this however does not
testing FOOO
foo
fo
od
```



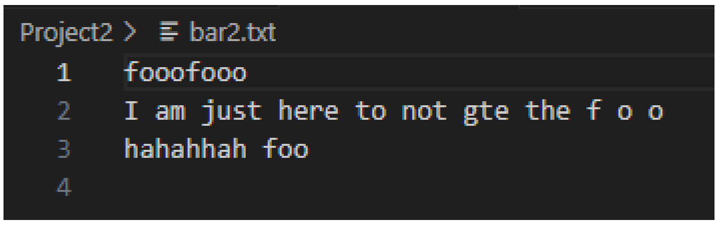
```
mattis@mattis-pc:/mnt/c/Users/Mattis/Documents/GitHub/PhaserG
this line has foo in it
so does this foolish line; do you see where?
even this line, which has barfood in it, will be printed.
foo
```

File(s) are given:

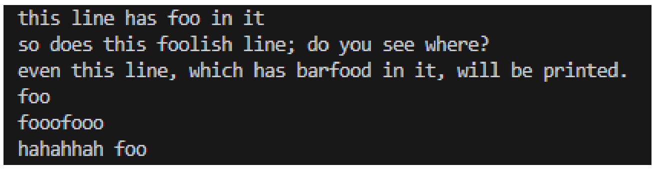
(bar.txt can be seen above)



```
/Project2$ ./my-grep foo bar.txt bar2.txt
```



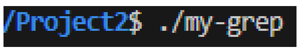
```
Project2 > bar2.txt
fooofooo
I am just here to not gte the f o o
hahahhah foo
```



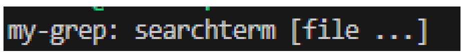
```
this line has foo in it
so does this foolish line; do you see where?
even this line, which has barfood in it, will be printed.
foo
fooofooo
hahahhah foo
```

Error cases:

No search term

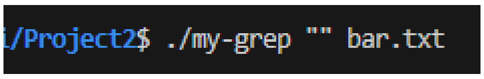


```
/Project2$ ./my-grep
```

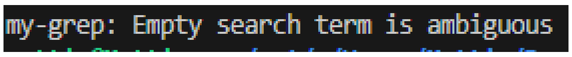


```
my-grep: searchterm [file ...]
```

Empty search term

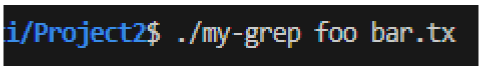


```
i/Project2$ ./my-grep "" bar.txt
```

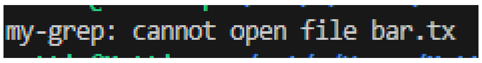


```
my-grep: Empty search term is ambiguous
```

Cannot open file



```
i/Project2$ ./my-grep foo bar.tx
```



```
my-grep: cannot open file bar.tx
```

Working principle:

I reused a lot of code from the my-cat program like reading the files and going through lines. On this program however also reading each letter was needed.

So imagine we have the searchterm argument in argv and the line from a file. We compare the letter in the line of text to the arguments first letter. If they are a match, we add 1 to a variable check, which now moves to the next letter in the argument. Then we check against the next letter in the line and so forth until the amount of letters checked as correct in the line match the lenght of the searchTerm.

3. ./~~my-zip file1 [file2 …]~~ →
./my-zip file1 [file2 …] > filename.z
AND
./my-unzip filename.z [filename2.z …]

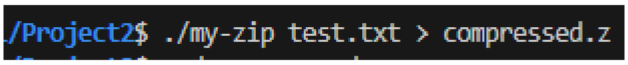
For this program I could use a lot of the code from the my-grep. So this does almost the same thing as grep in the sense that it reads all the letters from each line. However, I would like to mention that I have probably done this task incorrectly as i made the program work with the command **./my-zip file1 [file2 …] > filename.z** instead of just **./my-zip file1 [file2 …]**. I am not going to be changing this anymore so this will be my answer. The program does work us instructed with only the command being different and the output being the .z file.

So how the command works is that it can read any amount of files and then it compresses them into the .z file which the user has specified.

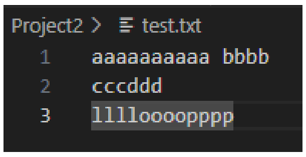
To look inside the .z file, just run ”xxd filename.z”, which then prints the file in binary format that a human can read.

A file is given:

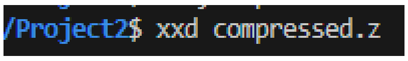
Zipping:



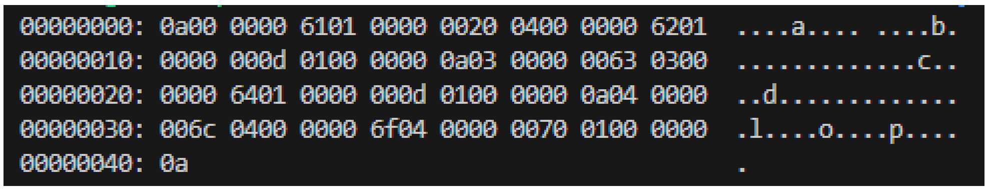
```
/Project2$ ./my-zip test.txt > compressed.z
```



```
Project2 > test.txt
aaaaaaaaaa bbbb
cccddd
lllloooopppp
```

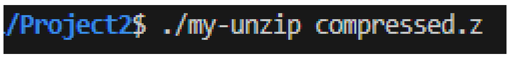


```
/Project2$ xxd compressed.z
```

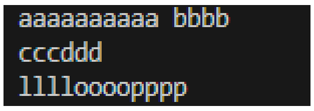


```
00000000: 0a00 0000 6101 0000 0020 0400 0000 6201  ....a.... ....b.
00000010: 0000 000d 0100 0000 0a03 0000 0063 0300  .............c..
00000020: 0000 6401 0000 000d 0100 0000 0a04 0000  ..d.............
00000030: 006c 0400 0000 6f04 0000 0070 0100 0000  .l....o....p....
00000040: 0a                                       .
```

Unzipping:



```
/Project2$ ./my-unzip compressed.z
```

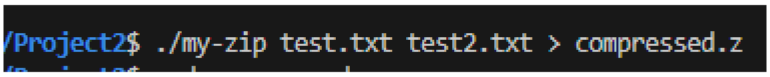


```
aaaaaaaaaa bbbb
cccddd
lllloooopppp
```

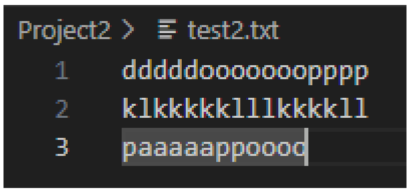
File(s) are given:

Zipping:

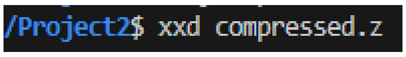
Test.txt can be seen above



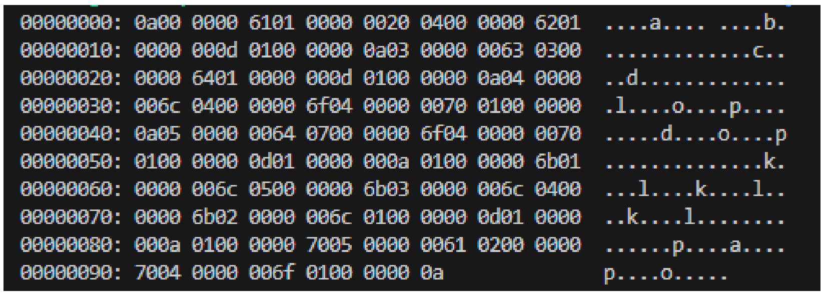
```
/Project2$ ./my-zip test.txt test2.txt > compressed.z
```



```
Project2 > test2.txt
dddddoooooooopppp
klkkkkklllkkkkll
paaaaappoooo
```

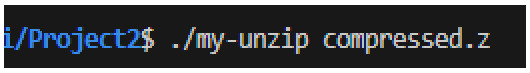


```
/Project2$ xxd compressed.z
```

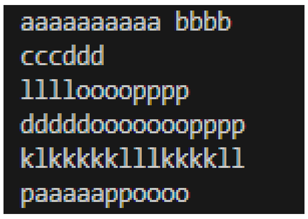


```
00000000: 0a00 0000 6101 0000 0020 0400 0000 6201  ....a.... ....b.
00000010: 0000 000d 0100 0000 0a03 0000 0063 0300  .............c..
00000020: 0000 6401 0000 000d 0100 0000 0a04 0000  ..d.............
00000030: 006c 0400 0000 6f04 0000 0070 0100 0000  .l....o....p....
00000040: 0a05 0000 0064 0700 0000 6f04 0000 0070  .....d....o....p
00000050: 0100 0000 0d01 0000 000a 0100 0000 6b01  ..............k.
00000060: 0000 006c 0500 0000 6b03 0000 006c 0400  ...l....k....l..
00000070: 0000 6b02 0000 006c 0100 0000 0d01 0000  ..k....l........
00000080: 000a 0100 0000 7005 0000 0061 0200 0000  ......p....a....
00000090: 7004 0000 006f 0100 0000 0a              p....o.....
```

Unzipping:

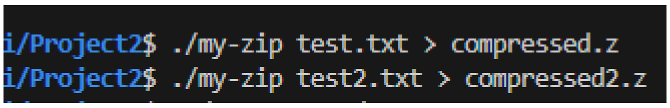


```
i/Project2$ ./my-unzip compressed.z
```

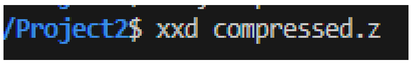


```
aaaaaaaaaa bbbb
cccddd
llllooopppp
dddddoooooopppp
klkkkkklllkkkkll
paaaaappoooo
```

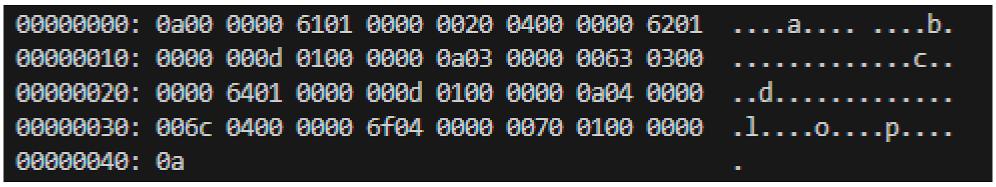
Unzipping multiple files:



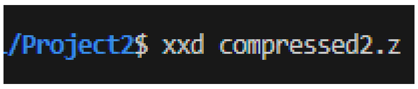
```
i/Project2$ ./my-zip test.txt > compressed.z
i/Project2$ ./my-zip test2.txt > compressed2.z
```



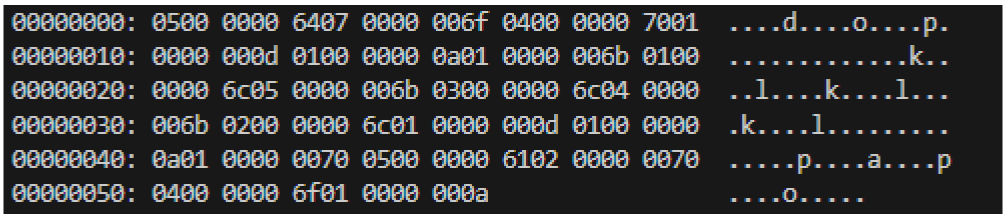
```
/Project2$ xxd compressed.z
```



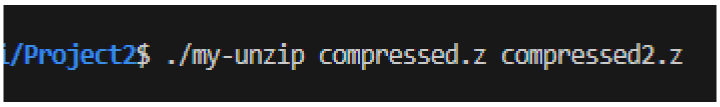
```
00000000: 0a00 0000 6101 0000 0020 0400 0000 6201  ....a.... ....b.
00000010: 0000 000d 0100 0000 0a03 0000 0063 0300  .............c..
00000020: 0000 6401 0000 000d 0100 0000 0a04 0000  ..d.............
00000030: 006c 0400 0000 6f04 0000 0070 0100 0000  .l....o....p....
00000040: 0a                                       .
```



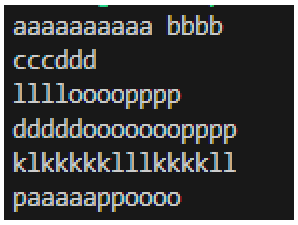
```
/Project2$ xxd compressed2.z
```



```
00000000: 0500 0000 6407 0000 006f 0400 0000 7001  ....d....o....p.
00000010: 0000 000d 0100 0000 0a01 0000 006b 0100  .............k..
00000020: 0000 6c05 0000 006b 0300 0000 6c04 0000  ..l....k....l...
00000030: 006b 0200 0000 6c01 0000 000d 0100 0000  .k....l.........
00000040: 0a01 0000 0070 0500 0000 6102 0000 0070  .....p....a....p
00000050: 0400 0000 6f01 0000 000a                 ....o.....
```

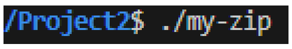


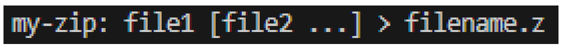
```
i/Project2$ ./my-unzip compressed.z compressed2.z
```

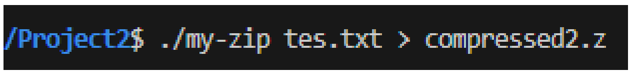


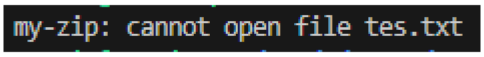
```
aaaaaaaaaa bbbb
cccddd
llllooopppp
dddddoooooopppp
klkkkkklllkkkkll
paaaaappoooo
```

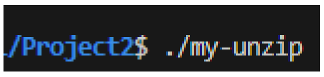
Error cases:

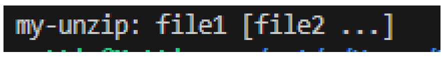

```
/Project2$ ./my-zip
```


```
my-zip: file1 [file2 ...] > filename.z
```


```
/Project2$ ./my-zip tes.txt > compressed2.z
```


```
my-zip: cannot open file tes.txt
```


```
/Project2$ ./my-unzip
```


```
my-unzip: file1 [file2 ...]
```

*note: the my-zip program should have more error cases like not having the command ”>” or the ”.z” file, but that checking would turn out to be almost as long as my current file so I just hope this is enough. The program should operate as instructed in the details, but with a little different command

Working principle:

My-zip:

So the zipping as said in the description of this project works similarly ot the grepping. The only major change is that it counts how many letters there are of the same kind in a row. If the letter changes, we use fwrite with sizeof(int) to write the amount of letters with 4 bytes, and then using fwrite wth sizeof(char) to write the character with 1 byte. We just go through all the lines and letters and adding these to the compressed file.

My-Unzip:

The unzip just does almost the same, by going through each file as binary and each line and then checking with fread sizeof(int) getting the 4 bytes which translate to a number and then using the fread to read the next byte as a character with sizeof(char). Now that we have the lenght, we just loop through printing with putchar as many times as the lenght is.