

INPUT ARGUMENTS

case: → call a function from here

- $n = \star$ -- raw = \star '---' \rightarrow implicitly default value
"true" in this case
- $b = \star$ -- buckets = \star '---' buckets \rightarrow implicitly default values
"intelligent" value finding (?)
- $c = \star$ -- CSV = \star '---' CSV \rightarrow put it into a *-log-stat.csv
- $o = \star$ -- out = \star '---' out - o \rightarrow put into a *-log-stat.log

Consider some generalization and separate function for assigning default parameters...
Have the defaults at the beginning? The order should be as follows \rightarrow if non-'=' found
 $-1 = \star$ -- if $n = \star$ ~~NO PRACTICE PLEASE~~ -- match = strict/loose then assign the de-
faults? $\text{impl}()$
FOR THIS REASON
ASSIGN THE SAME SIZES
TO BUCKET COUNT
NOT FOR TEXT LENGTH
OF THIS PARSING

$-v = \star$ -- value-format = \star

PYTHON VISUALIZER

→ for the percentages found
P1 E GRAPH

Support for having the arguments anywhere?

We cannot even export the array from Bash, so gives we are stuck with pickling or some clever way of encoding the in the commandline args, like
it needs to be the same as *

py X -f Vol 1, Vol 2, Vol 3, Vol 4, ... Is that feasible?

Q:

1. How to pass data from Bash to the Python script?

2. What kind of args are accepted by Python (`input/CLI args`)

3. gnm plot (?)

4. Bind C++ version to Python Vmulator and the Bash version...

HOW TO MASS VARIANCE TEST HYPOTHESIS SCRIPT

"errorCode" → py m X Val'1 Vol 2 ... Vol n
number → (map)
of states →
(like error code : 1
errorCode : 2
3 states
errorCode : 4 2

Buckets and logarithmic -- find-gaps = y (?)

-- buckets = x

We have a number of occurrences of the stat name ...

error code *

number of buckets

5 error code: 0

↑
↑
↑
↑
↑
number of name
occurrences $\xrightarrow{x=3}$ stat
value

$\lceil \min, \max \rceil \cdot (\max - \min) / x$

$$5 \quad 15 \quad (15 - 5) / 3 = 10 / 3 = 3, (3) \approx 3, 33$$
$$-10 \quad 20 \quad (20 - (-10)) / 3 = 30 / 3 = 10$$

Bucket Occurrence Value Array $[x] = \text{occur} (?)$

buckets Intervals $[x]$ ~~array of arrays!~~ ~~array of arrays!~~ Bash does not support multi-dim arrays!

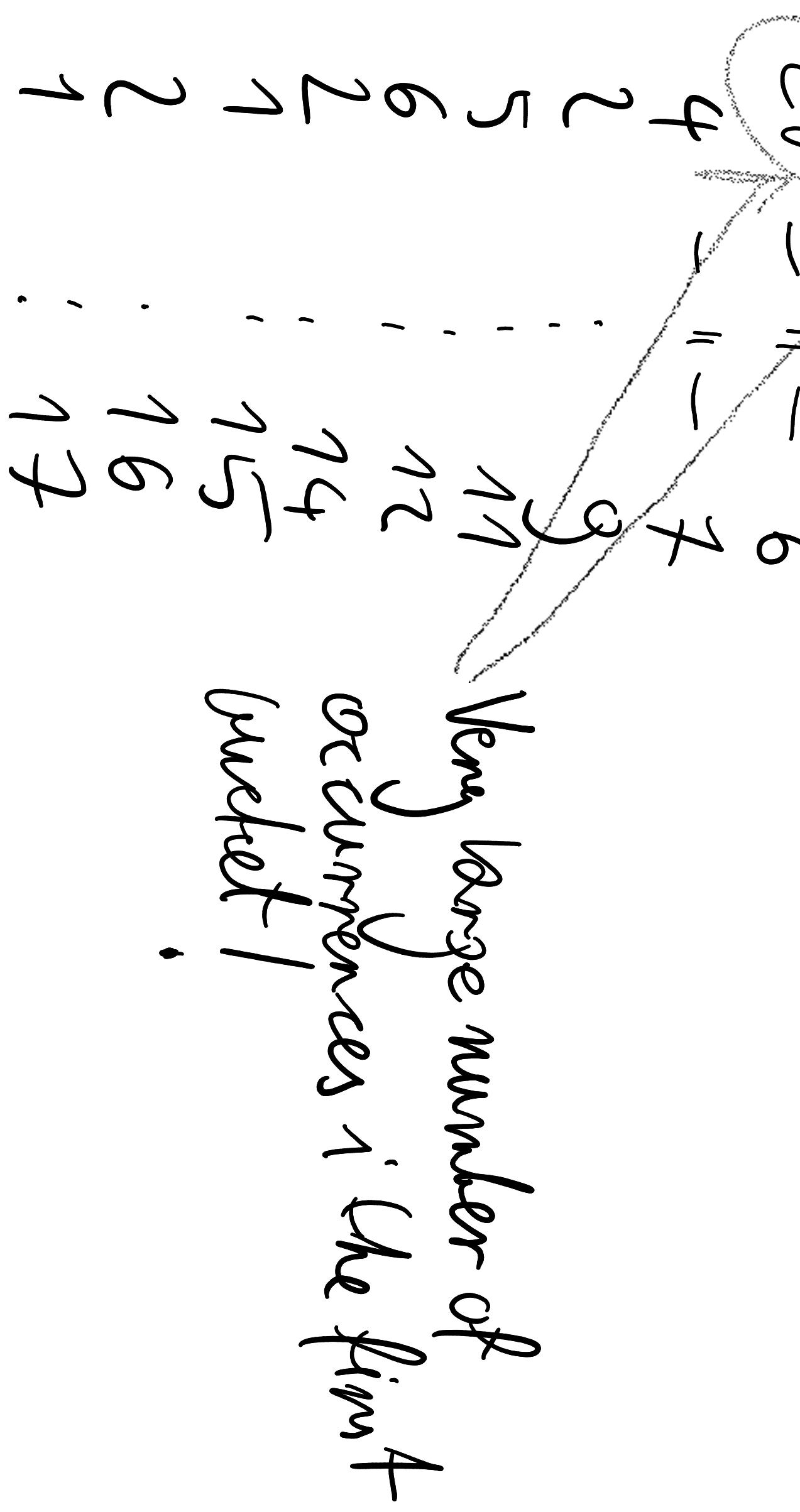
Yeah, we need to find some other way to store this information.

Buckets

We can: very many values of $start$, no need to put them into buckets, example:

1 symbol 1 bucket \rightarrow two buckets!

$$\text{size of each bucket} \rightarrow (17 - 1)/2 = 16/l = 8$$



STAT AMBIGUITY PROBLEM.

error code → will be found by grep

error code derived → because of the '*' in grep that would also be found but won't be counted to specific stat percentage share

Maybe we use '*' and find more stats?

Find ambiguities?

if we match for `stat[0-9a-zA-Z]` then

there is ambiguity → print the found matches...

Current regex in: `stat.*[0-9]+`

Have stat-value separation? Have a default separator? IFS?

C++ BASED ENGINE

Just do a MVP minimum

Just open the file, getLine() use std::regex to search for the parameter/stat... .

How to read the files? Maybe getLine() or something else? Have a SW unit that would align and level-out to the features of Bash and C++?

Remove globals from Bash and mimic all the Bash functions in C++

Analyse the equivalence of grep -Eo and so on...

Get the whole file into the memory? Or push it into a file? (stat1 | stat2 / anotherStat)

How to process the extracted Stat?

```
grep | sort | uniq -c | group(?)
```

getLine → regex - match → see if unique value (store in

std::set) → put into set and increment or only increment

Construct a regex that would match! Or "statNames and counts do just one pass over the file? Would it be feasible? Probably, because the different process it would be impossible to have the same unit function for both Bash AND C++.

C++ BASED ENGINE

If errorCode : 0

Object-Oriented approach ?

Value being a key
occur in the value

global (?)

errorCode

10 errorCode : 0

2 error Code : 1

pair < string , map < Value , Occurrence > >

58 error Code : 2



6 when 23

103 when 53

vector of stats ...
push the stats
(.resize to the
number of elements)

C++ STRUCTURE (MAIN LOOP)

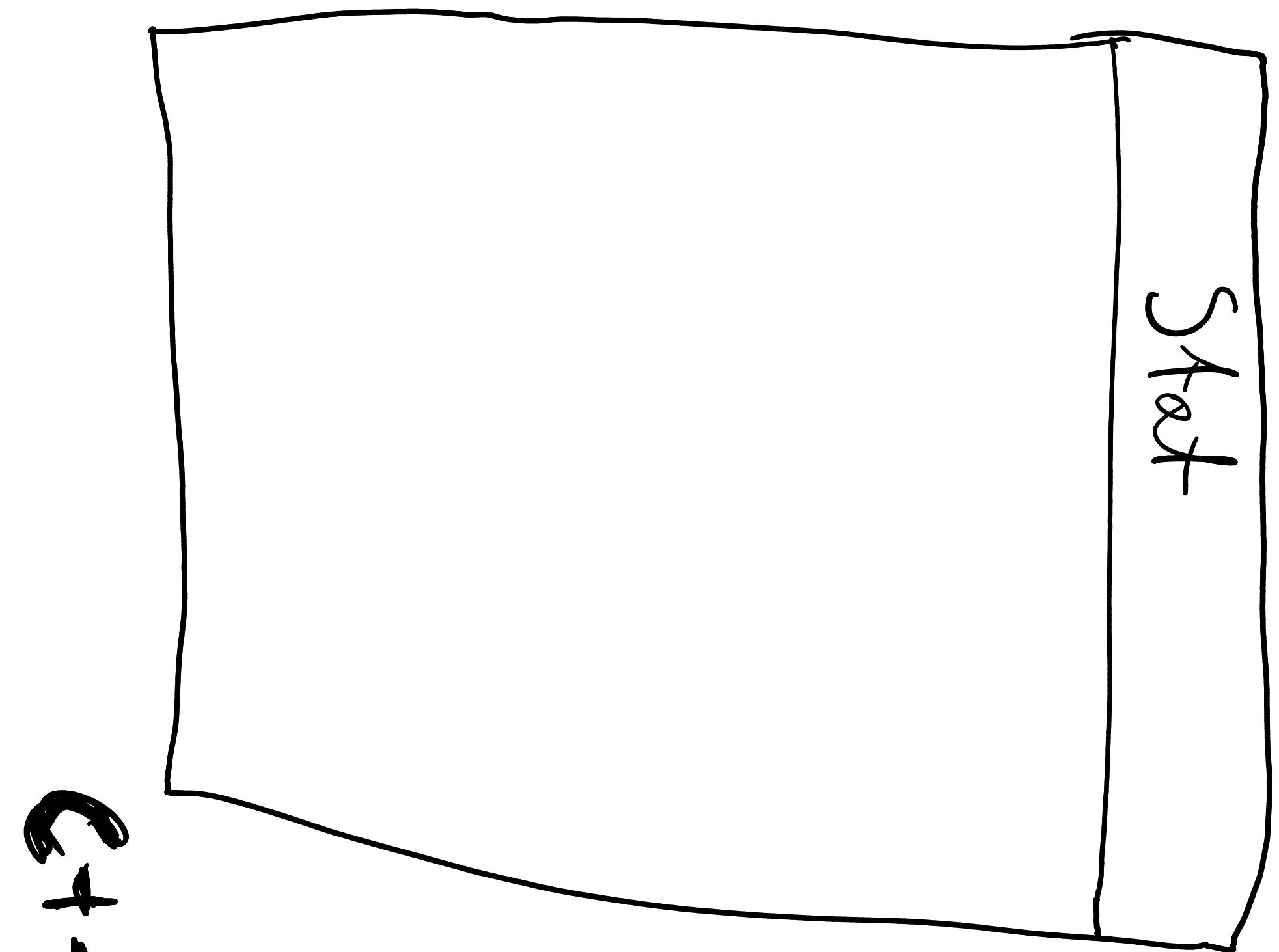
```
main()
{
    Logfile("file-name") >> automatically open the file
    logFindAndProcessStats() What is there to parse?
    return the database?
}

RV0()

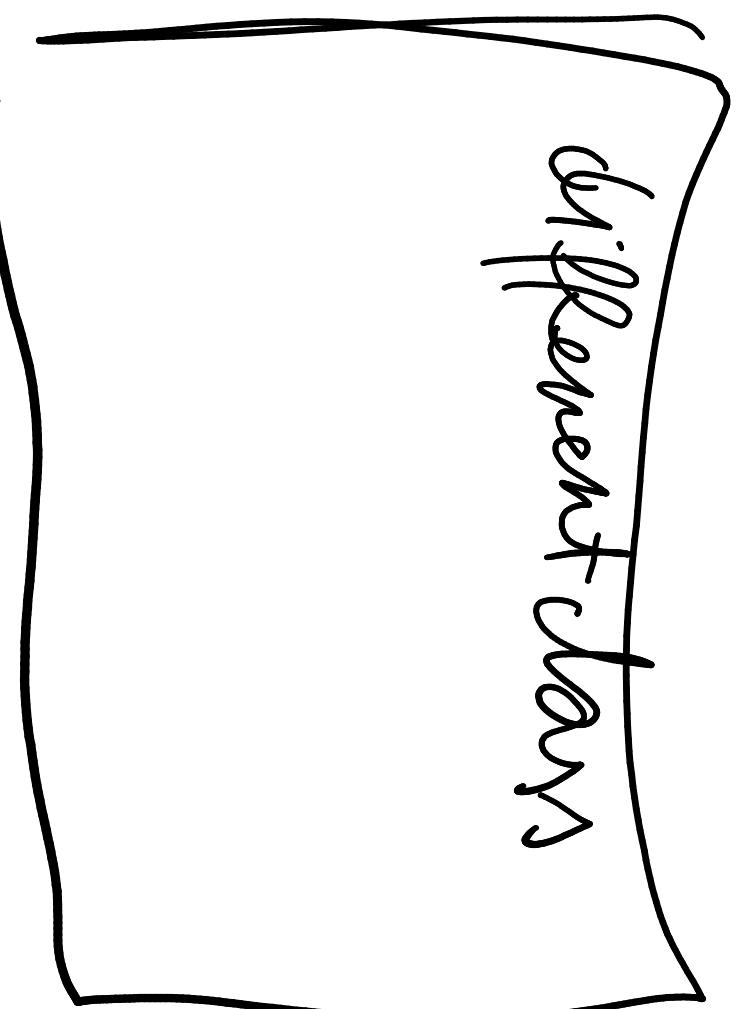
findAndProcessStats()
{
    // Construct regex
    // Do one pass over the file (?)
    while (getline())
    {
        // If new stat was found - then create
        // new stat-object and push to vector
    }
}
```

C++ STRUCTURE (STAT CLASS)

different set of values ...
different number of occurrences of each value ...



C++ STRUCTURE (OPTS) →



Treat all args as stats ...
Do some const for observability ...
statName1, statName2, ...

C++ FILE-LEVEL MODULARISATION



calc TotalShare (statName) → totalOccurrences (MY BE RENAME ?)
(done as "grep -E -o \$statName" if no opt[0-9] + logfile | wc -l")

if we will implement with getline()
then this function would need to
just return previously calculated
value? Would need to be worked in
test?

Now - re-implement it in C++ (?)

for statName in statNames {
 Bind it to test env as soon as possible
 Cross-test both Bash and C++ functions?
 - how to call the C++ functions versus Bash?
 - how to get the return values?
 - do object? load Bash and test-skeleton
 as separate functions?

MORE SOPHISTICATED STATS

Standard deviation? .
Percentiles? .
All of that is not worth it because we can just export it to *.csv file and then proven it with some existing (and better) tools? .

TRANSFORM

Somehow build a module to process the logs and transform them into other form?
Only in C++? How to build it?

BASH MODULARIZATION

Operating on global (?)

All THE FLOATING
PUNCTIONS (?)

INPUT PARAM
PARSER