INPUT ARGUMENTS

case: → call a function from here

- r = ✗ -- raw = ✗ (-- raw - r) → Implicitly default value "true" in this case
- b = ✗ -- buckets = ✗ -- buckets -b → Implicitly default values "intelligent" value finding (?)
- c = ✗ -- csv = ✗ -- csv -c → put it into a *_log_stat.csv
- o = ✗ -- out = ✗ -- out -o → 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 → if non-"=" found
- i = ✗ -- ifs = ✗ NO IMPLICIT VALUE FOR JUST THE EXISTENCE OF THIS FLAG
-- match = strict/loose (?) then assign the defaults? Yup(?)
- v = ✗ -- value-format = ✗ ALSO, THERE IS NO IMPLICIT VALUE JUST FOR TE EXISTENCE OF THIS FLAG

INPUT ARGUMENTS C++

OpA class should be separated from application logic of flag handling...

GetFlagValue() → just return flag value (empty string if flag does not exist or ~~is~~ has empty value, basically if '=' is not present

IsFlagPresent() → returns if flag is present in the flags thing?

Basically returns everything after '=' character...

Make it work with '-' '--' and just the flag name...

We know that flag will begin with '-'

match substring

~~OpA~~ strip of '-' and '--' from searched String & only requirement is starting with '-'
and match substring...?
if first char is - rem -
rem -
if then first is - rem -

--- na vrnivi false
- a - na vrnivi false

IsFlagPresent()

flag
Requirements: '-' is the first char

searched String
We can enter: - flag -- flag flag And it should return 'true'

But the -- curl - is equivalent (probably no)

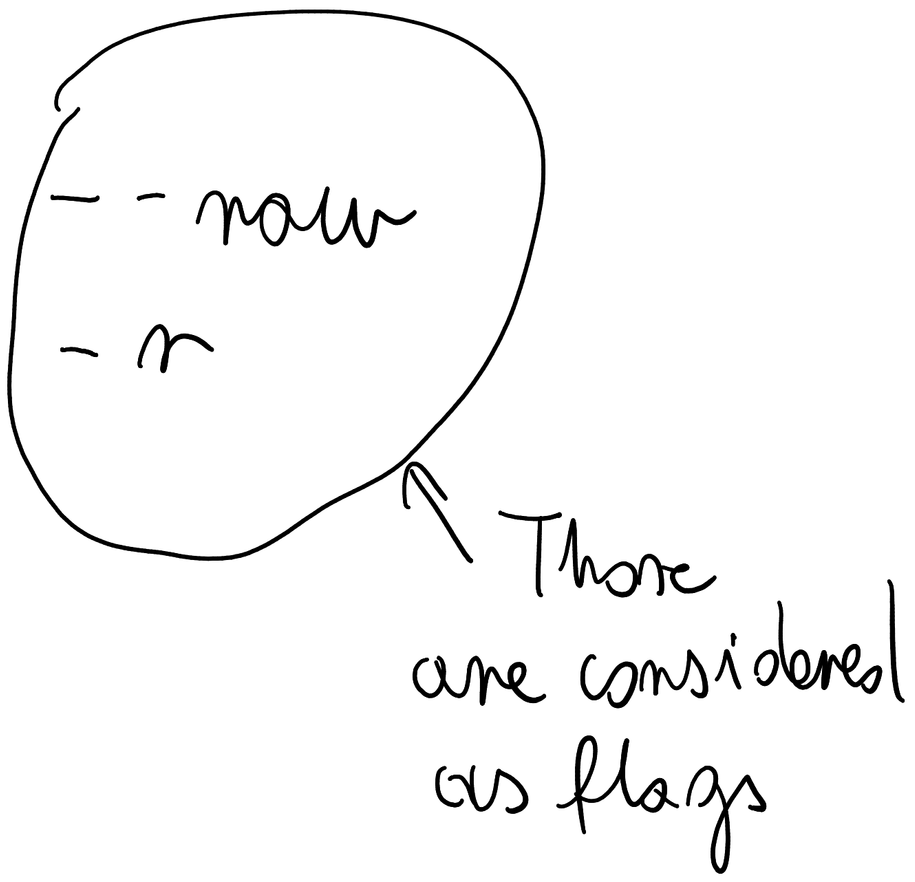
- - e
0 1 2

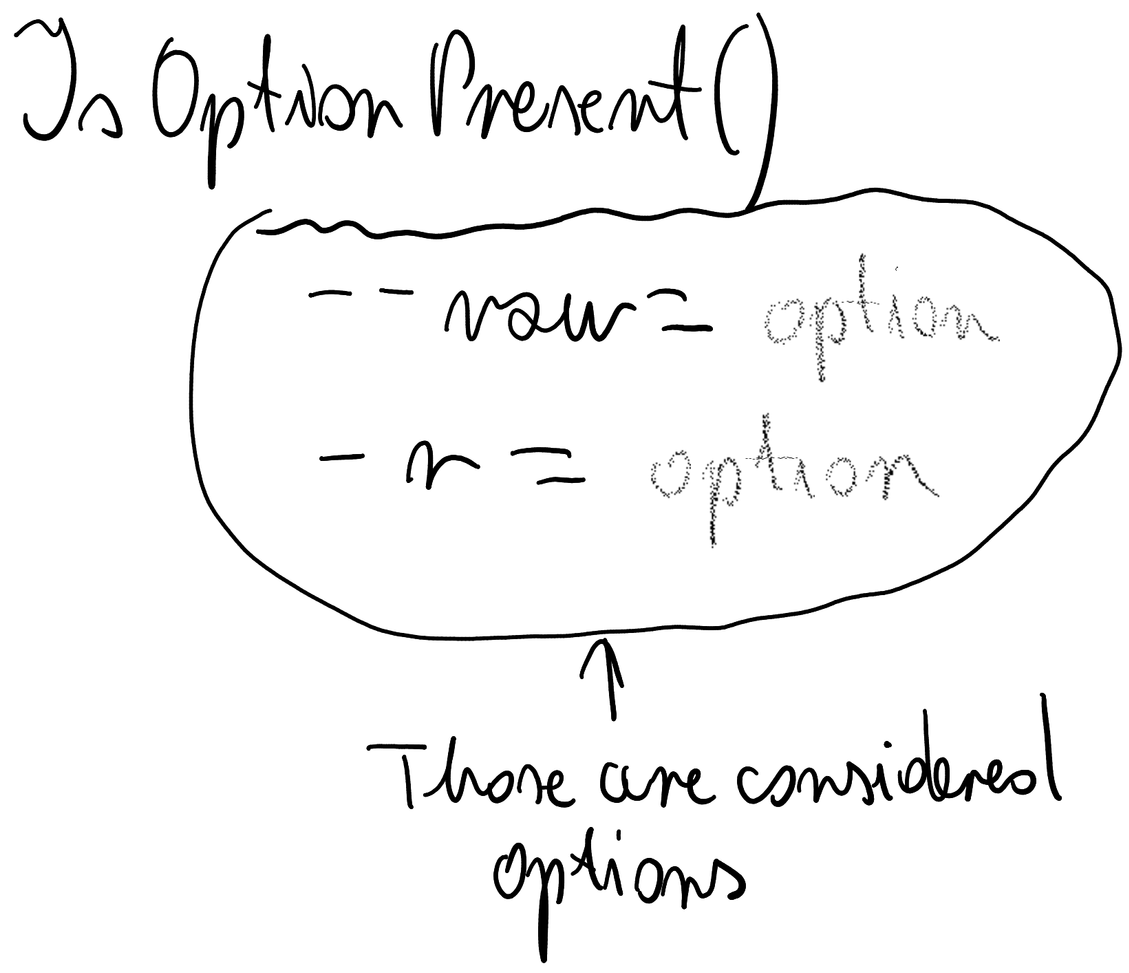
~~[redacted]~~ Two cases?
searchedString begins with '-'

Yeah, but then we have the -- flag option...

Need to mimic the Bash approach?
If --raw or -r then flag exists and overrides the default init flag-present default...

One case
first_first_of('-') and compare?
AND
position of '-' is not longer than two
THEN
Just compare the strings...
Problem with -something
-- something





Yeah, THAT NEEDS TO BE THE FOCUS ALONG WITH IsFlagPresent() REVIEW...

--OUT should not open a file or dump into some null stream...

GetOptionValue()

More or less works...

if(IsFlagPresent)?

Or just return empty string in both cases?

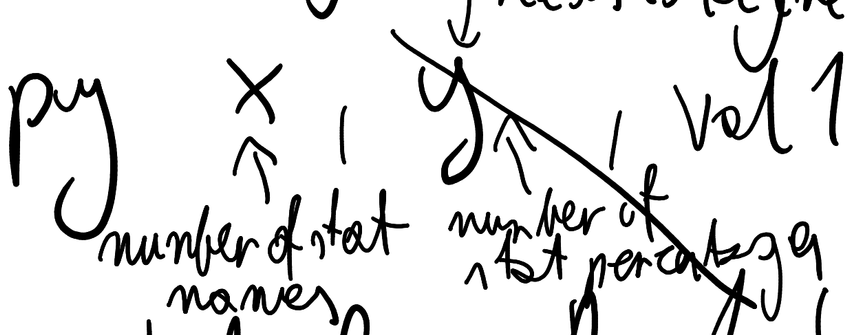
then if --raw= would be treated like no option was provided...

PYTHON VISUALIZER

PIE GRAPH → for the percentages found

Support for having the arguments anywhere?

We cannot even export the array from Bash, so I guess we are stuck with file-proxy or some clever way of encoding that in the commandline args, like



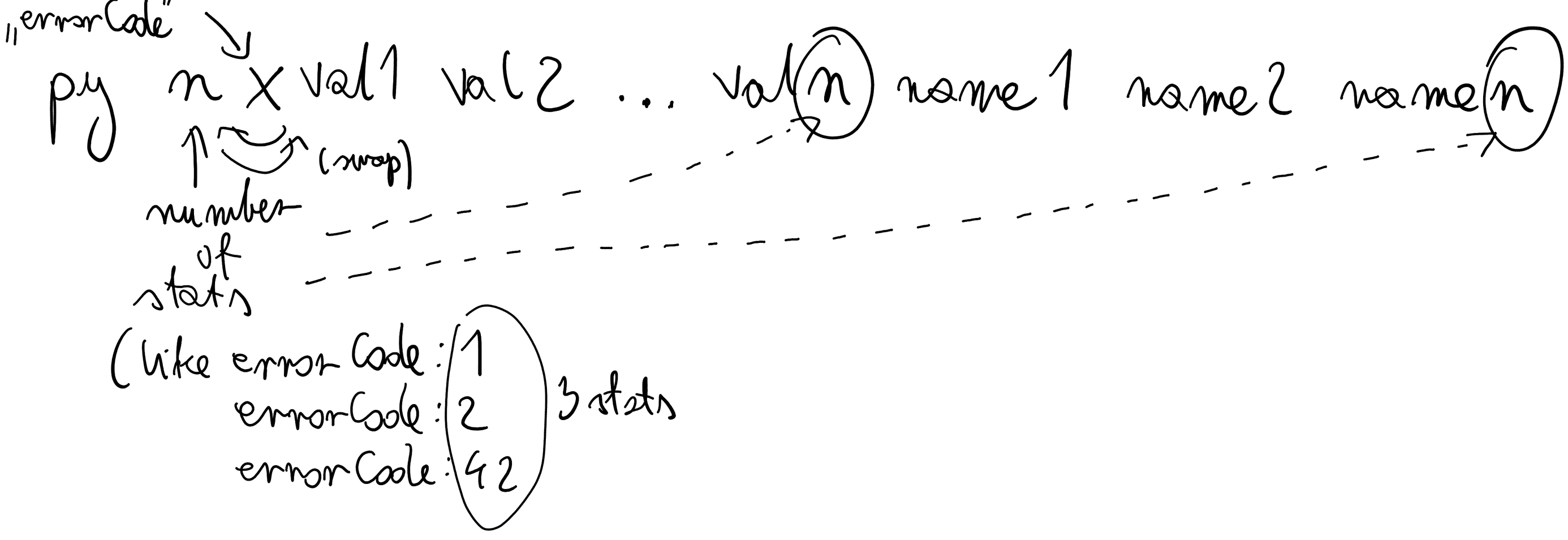
py x y val1, val2, val3, val4, ... Is that feasible?

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. gnuplot (?)
4. Bind C++ version to Python Visualizer and the Bash version...

HOW TO PASS VARIABLES TO PYTHON SCRIPT

stat name "errorCode"



BUCKETS and logarithmic

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

errorCode *

5 errorCode: 0

- 5: number of occurrences
- errorCode: stat name
- 0: stat value

x = 3

[min, max] (max - min) / x

5 15 (15 - 5)/3 = 10/3 = 3,(3) ≈ 3,33

-10 20 (20 - (-10))/3 = 30/3 = 10

bucketsOccurrenceValueArray[x] = occurr (?)

bucketsIntervals[x] ↔ array of arrays ?

Bash does not support multi-dim arrays!

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

--find-gaps=y (?)

--buckets=x

- x: number of buckets

buckets vs non-numerics (?)

buckets vs outliers

--ignore-outliers=y

- y: some percentile

BUCKETS

Use case: very many values of stat, so need to put them into buckets, example:

Buckets → two buckets!
size of each bucket → (17-1)/2 = 16/2 = 8

1	symbol	1
2	symbol	2
10	symbol	3
2	"	5
20	"	6
4	"	7
2	⋮	9
5	⋮	11
6	⋮	12
2	⋮	14
1	⋮	15
2	⋮	16
1	⋮	17

Very large number of occurrences i the first bucket!

STAT AMBIGUITY PROBLEM...

errorCode → will be found by grep

errorCodeDerived → because of the ".*" in grep that would also be found but won't be counted to specific stat percentage stats

Maybe use the * and find more stats?

Find ambiguities?

if one matches for(stat[0-9 a-z A-Z])

then

there is ambiguity → print the found matches...

Have a global grep engine that would be more of a global variable, that could be reused AND will be the same across Bash AND C++

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

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

C++ BASED ENGINE

Just do a MVP mirror?

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 be even the features of Bash and C++?
Remove globals from Bash and mimic all the Bash functions in C++?
Analyze 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?

Check which stat matched the group(?)

grep | sort | uniq -c

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 would 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 functions for both Bash AND C++.

C++ BASED ENGINE

Object-oriented approach?

5 errorCode: 0

UNIQUE VALUES FOR A STAT

10 errorCode: 0
2 errorCode: 1
58 errorCode: 2

6 user 23
103 user 53

errorCode

global (?)

value being a key
occurr is the value

pair<string, map<value, occurr>>
(string) (long long)

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

C++ STRUCTURE (MAIN LOOP)

```
main()
{
    LogFile("file-name") log;
    log.findAndProcessStats(
        return the database?
        RVO(?)
```

automatically open the file

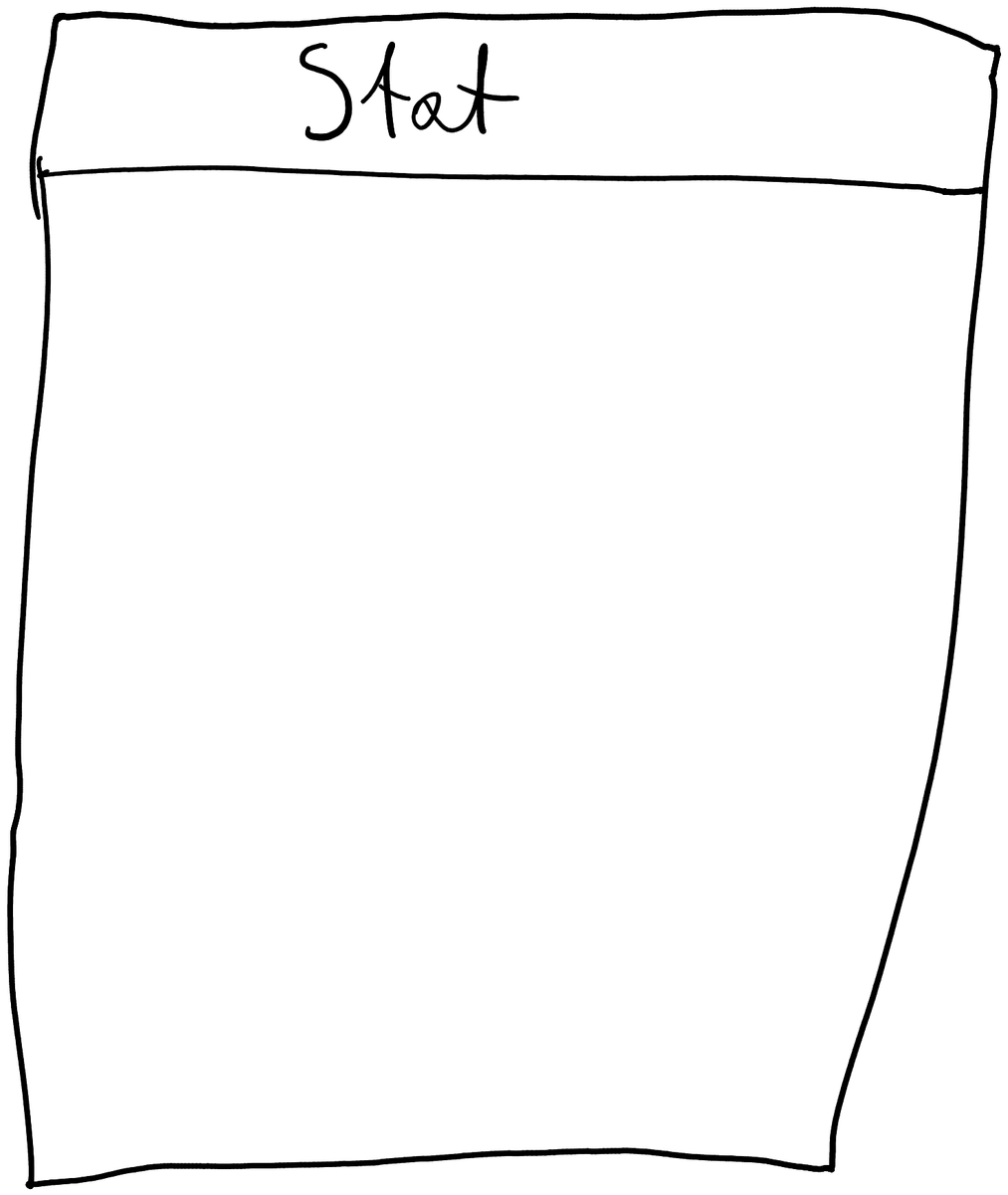
What is there to pass?

) Regex? (stat|lstat|fstat) (ifs) (valueFormat)

separate strings?

```
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 counts for observability...

statName1, statName2, ...

Log Stat? StatRegex? StatRegexEntry? LineClass?

How do the Stat, Regex and LogFile interact?

Entry (construct from the

~~Any int~~

Any interface?

Like a std::string class?

EntryRegex (stat[], ifs, valueformat)

{

the regex itself...

positions of the groups...?

injected from pos(?)

string GetMatch (the position of match)

smatch GetMatches (all by default)

matches

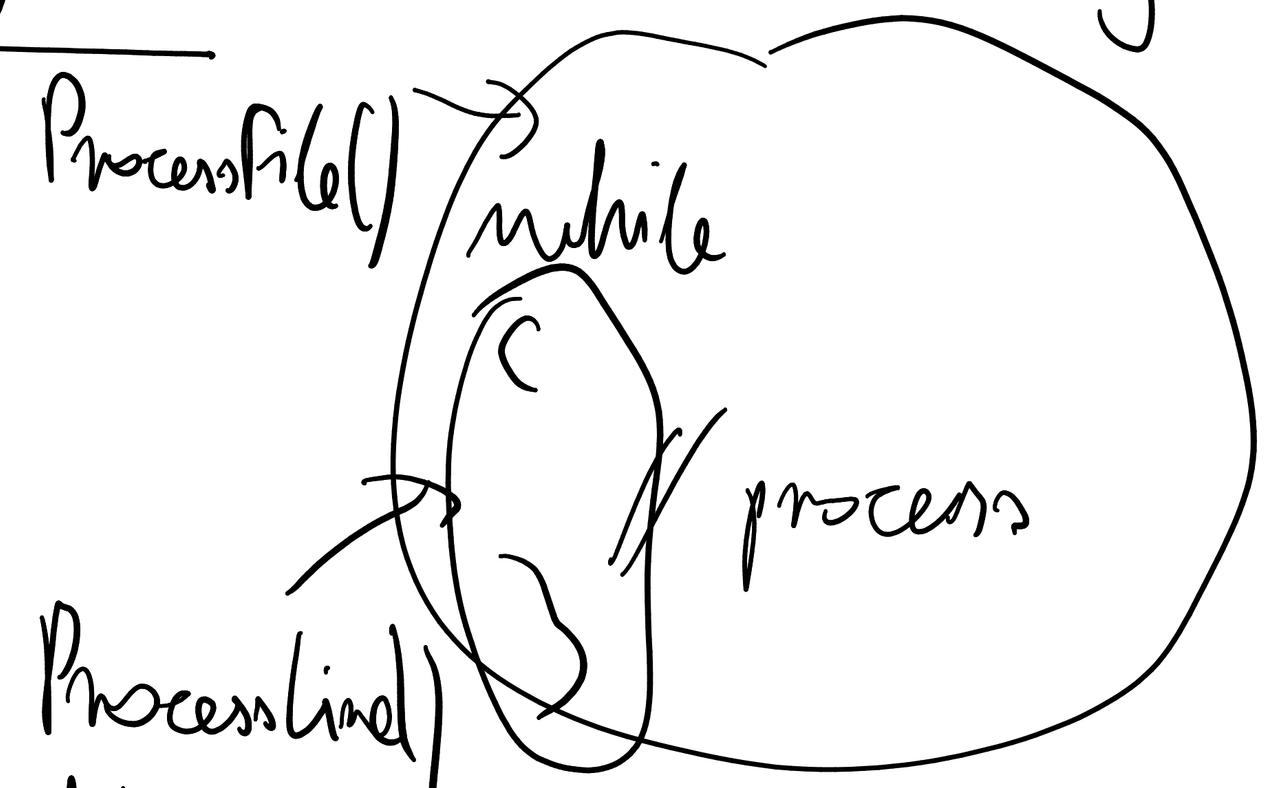
}

consts be extracted and reused widely!

EntryRegex::Search (line) std::string

foo(const x) → char(x)

can it be modified here?



C++ CALCULATING STATISTICS

~~stat~~ vector<Stat> stats

~~GOAL: Match the output!~~

More or less matched - some problems have been detected ...

stat1 + stat2 → returns totalOccurency ...

for (stat in stats) { // Process stat

sumAllOccur (for every value += occurSum

for every value

occurr/occurrSum

Is that even feasible?

Would it be a good design cas? Would we add it

Probably it is not the best idea ...

std::string is the value

std::string - because it doesn't matter ...

(For those specified in readme. ms Yeah, maybe some cases/problem was detected...

Actually the points that are left:

1. ~~Process and print results~~ More or less done
2. ~~Colored string ... (design and dev)~~ Yeah, more or less ...
3. ~~Make the "-" options possible in C++~~ Done
4. Equivalence of Bash and C++ in Opts ...
5. ~~Cleanup ...~~ some done
6. ~~Bind TRACE_ECHO~~
7. Rearrange special fundions

Later:

Make introduction

Test introduction

C++ FILE-LEVEL MODULARISATION

main.cpp
(the main file)

Logfile.cpp

Logfile.hpp

Opt.hpp

Opt.cpp

Stat.cpp

Stat.hpp

Trace.cpp
global Trace
declaration

Trace.hpp
LEAVE and
ENTER macros...

calcTotalShare(statName) → totalOccurrences (MAYBE RENAME?)

(done as "grep -E -o ${statName}${ifsOpt}[0-9]+ ${logFile} | wc -l"

try to parametrize it...

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

Now - reimplement it in C++ (?)

for statName in statNames
    calcTotalShare(statName)

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?
- .so 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 process it with some existing (and better) tools?

TRANSFORM

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

BASH MODULARIZATION