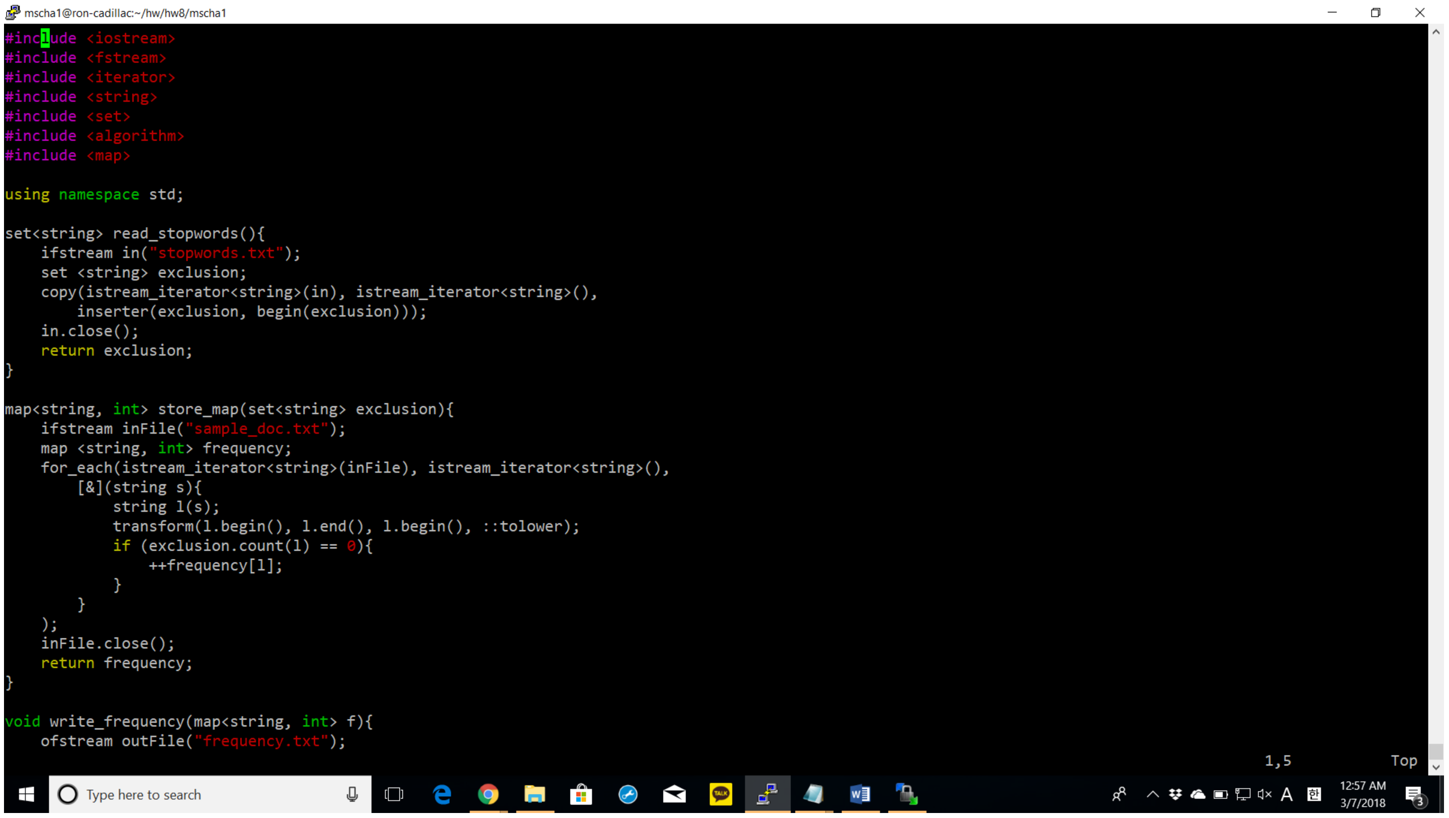
Name: Min Sung Cha ID: 85408485

Include your whole mapset.cpp and process_numbers.cpp for the function analysis. And explain each function in each file.

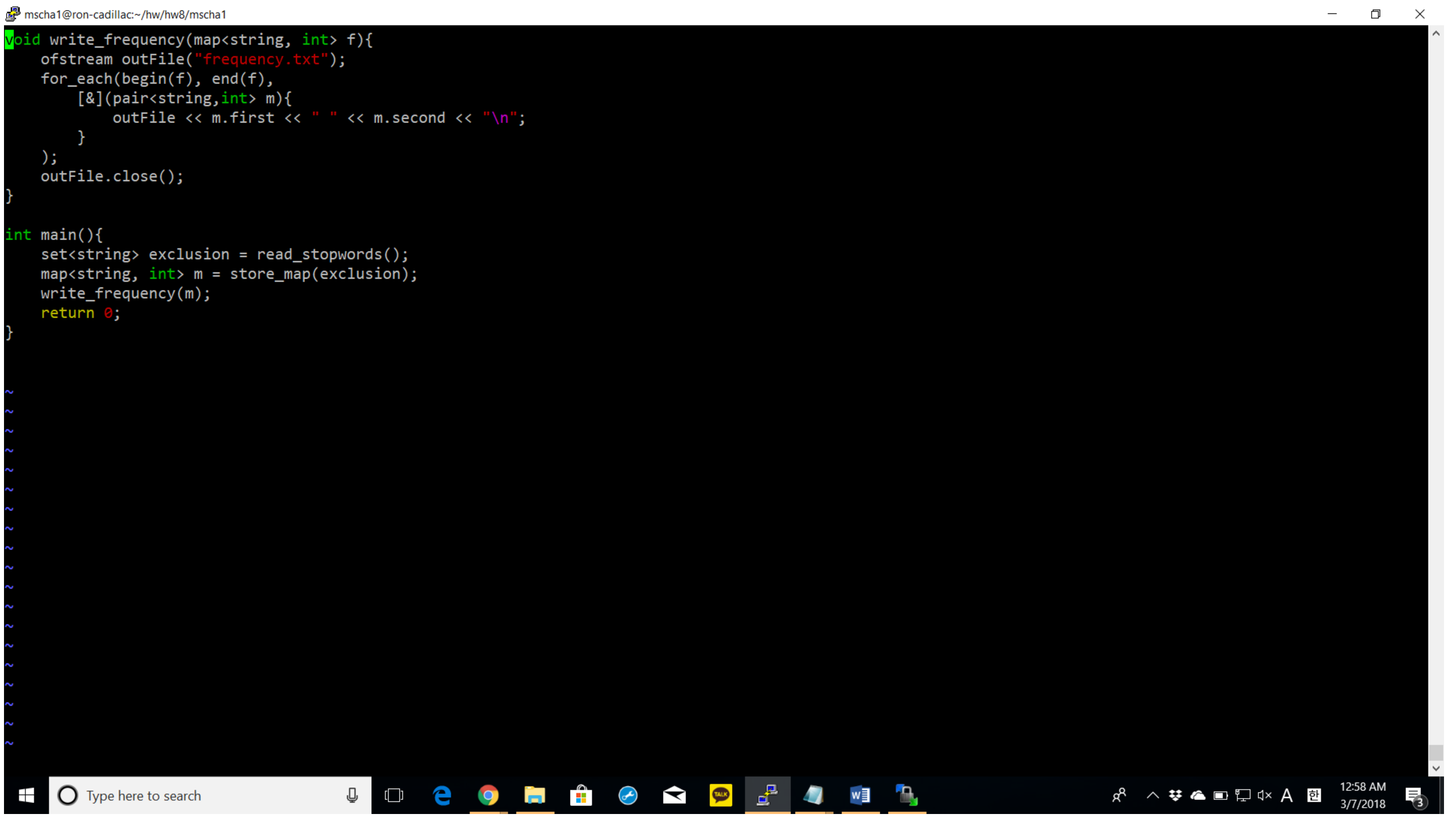
mapset.cpp



```
#include <iostream>
#include <fstream>
#include <iterator>
#include <string>
#include <set>
#include <algorithm>
#include <map>

using namespace std;

set<string> read_stopwords(){
    ifstream in("stopwords.txt");
    set <string> exclusion;
    copy(istream_iterator<string>(in), istream_iterator<string>(),
        inserter(exclusion, begin(exclusion)));
    in.close();
    return exclusion;
}

map<string, int> store_map(set<string> exclusion){
    ifstream inFile("sample_doc.txt");
    map <string, int> frequency;
    for_each(istream_iterator<string>(inFile), istream_iterator<string>(),
        [&](string s){
            string l(s);
            transform(l.begin(), l.end(), l.begin(), ::tolower);
            if (exclusion.count(l) == 0){
                ++frequency[l];
            }
        }
    );
    inFile.close();
    return frequency;
}

void write_frequency(map<string, int> f){
    ofstream outFile("frequency.txt");
                                                                            1,5           Top
```

read_stopwords: this function reads “stopwords.txt” and copies it into set of strings using the algorithm copy. The beginning and end of the file are obtained by istream_iterator<string>(in) and istream_iterator<string>(). The inserter function is used to insert the strings at the back from the beginning.

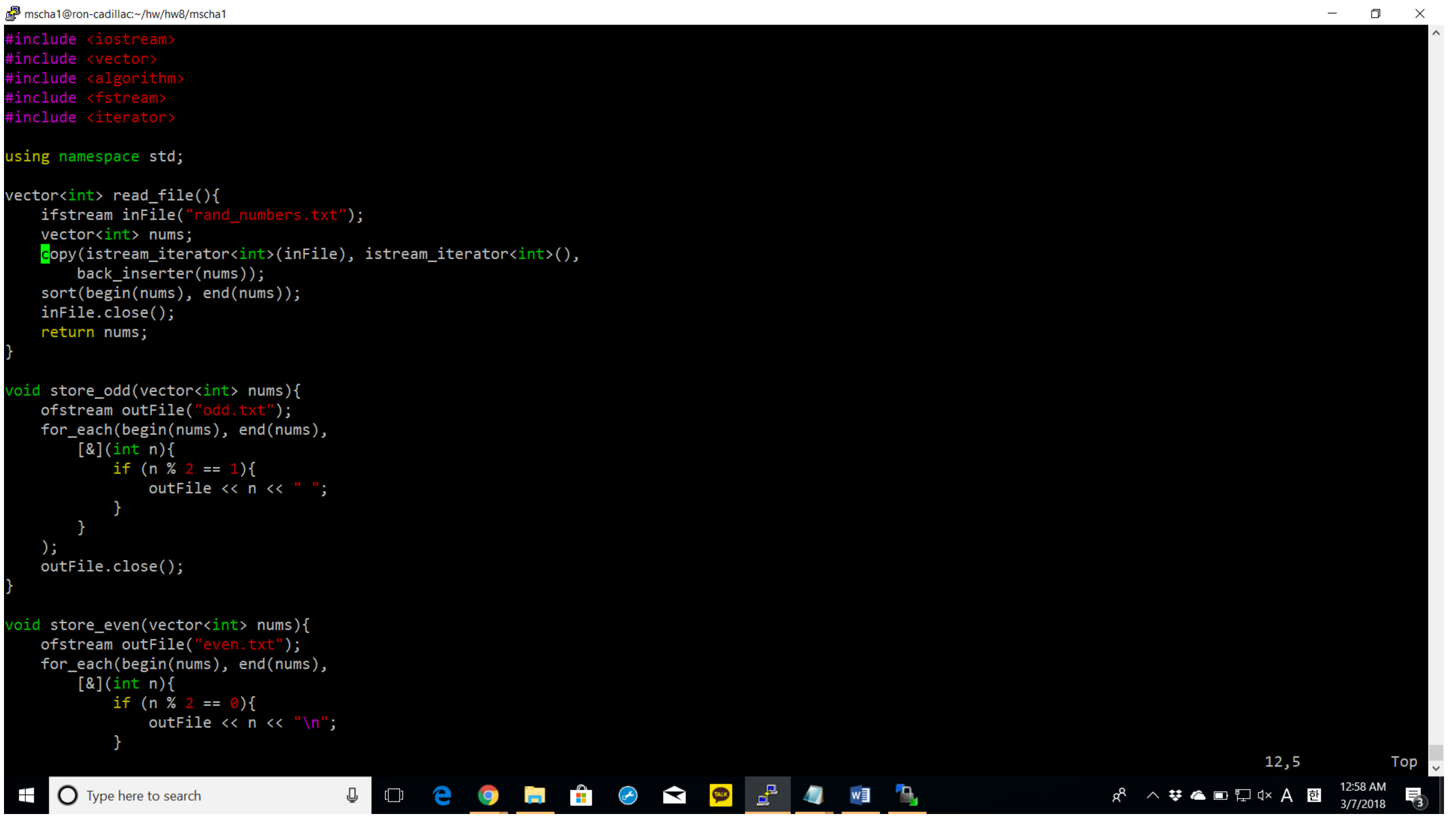
store_map: reads “sample_doc.txt” and creates a map called frequency. It loops over every element in the file by using the algorithm for_each. For_each takes in a lambda which takes in a string as a parameter, duplicates it, changes to lowercase by using transform and checks if the count of the string is zero in the exclusion set (set creates with the words in stopwords.txt). If zero, the corresponding int info of the key is incremented by 1.



```
void write_frequency(map<string, int> f){
    ofstream outFile("frequency.txt");
    for_each(begin(f), end(f),
        [&](pair<string,int> m){
            outFile << m.first << " " << m.second << "\n";
        }
    );
    outFile.close();
}

int main(){
    set<string> exclusion = read_stopwords();
    map<string, int> m = store_map(exclusion);
    write_frequency(m);
    return 0;
}
```

write_frequency: takes in a map as a parameter and creates the file “frequency.txt”. loops through the map by using for_each algorithm which takes a lambda as a parameter. The lambda takes in a pair of string and int as a parameter and writes to the file by using the ostream operator <<.

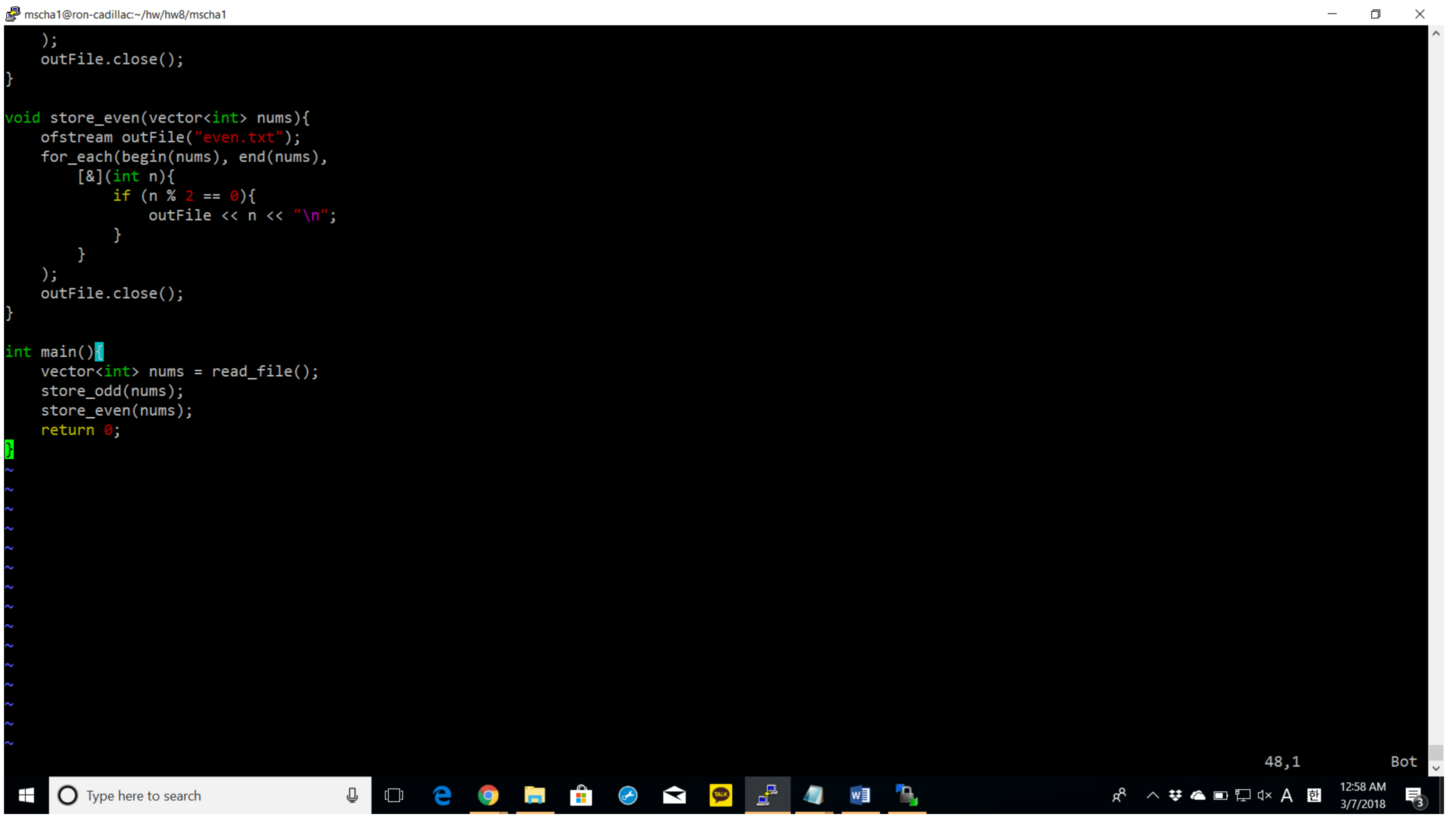
process_numbers.cpp



```
#include <iostream>
#include <vector>
#include <algorithm>
#include <fstream>
#include <iterator>

using namespace std;

vector<int> read_file(){
    ifstream inFile("rand_numbers.txt");
    vector<int> nums;
    copy(istream_iterator<int>(inFile), istream_iterator<int>(),
        back_inserter(nums));
    sort(begin(nums), end(nums));
    inFile.close();
    return nums;
}

void store_odd(vector<int> nums){
    ofstream outFile("odd.txt");
    for_each(begin(nums), end(nums),
        [&](int n){
            if (n % 2 == 1){
                outFile << n << " ";
            }
        }
    );
    outFile.close();
}

void store_even(vector<int> nums){
    ofstream outFile("even.txt");
    for_each(begin(nums), end(nums),
        [&](int n){
            if (n % 2 == 0){
                outFile << n << "\n";
            }
                                                                12,5          Top
```

read_file: it reads rand_numbers.txt and copies all the integers in the file to the integer vector by back inserting them. Then, the vector of integers is sorted by using the sort algorithm.

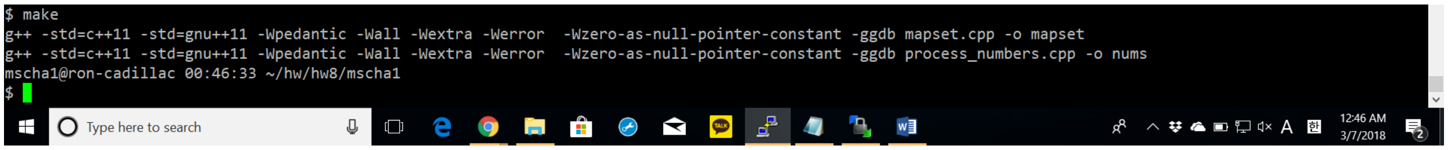
store_odd: creates file “odd.txt” and loops through the vector integer using the for_each loop which takes a lambda as a parameter. The lambda takes in every integer in the vector and determines if it is odd by calculating if the modulus is 1. Then, if odd, the integer is written in the file with a whitespace at the end.


mscha1@ron-cadillac:~/hw/hw8/mscha1

```
    );
    outFile.close();
}

void store_even(vector<int> nums){
    ofstream outFile("even.txt");
    for_each(begin(nums), end(nums),
        [&](int n){
            if (n % 2 == 0){
                outFile << n << "\n";
            }
        }
    );
    outFile.close();
}

int main(){
    vector<int> nums = read_file();
    store_odd(nums);
    store_even(nums);
    return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                                                                    48,1          Bot
```

store_even: creates a file named “even.txt” and loops through the vector of integers using for_each algorithm which takes a lambda as a parameter. The lambda has an formal parameter of int which is used to calculate if the modulus is 0. If even, it is written onto the file with a newline at the end.

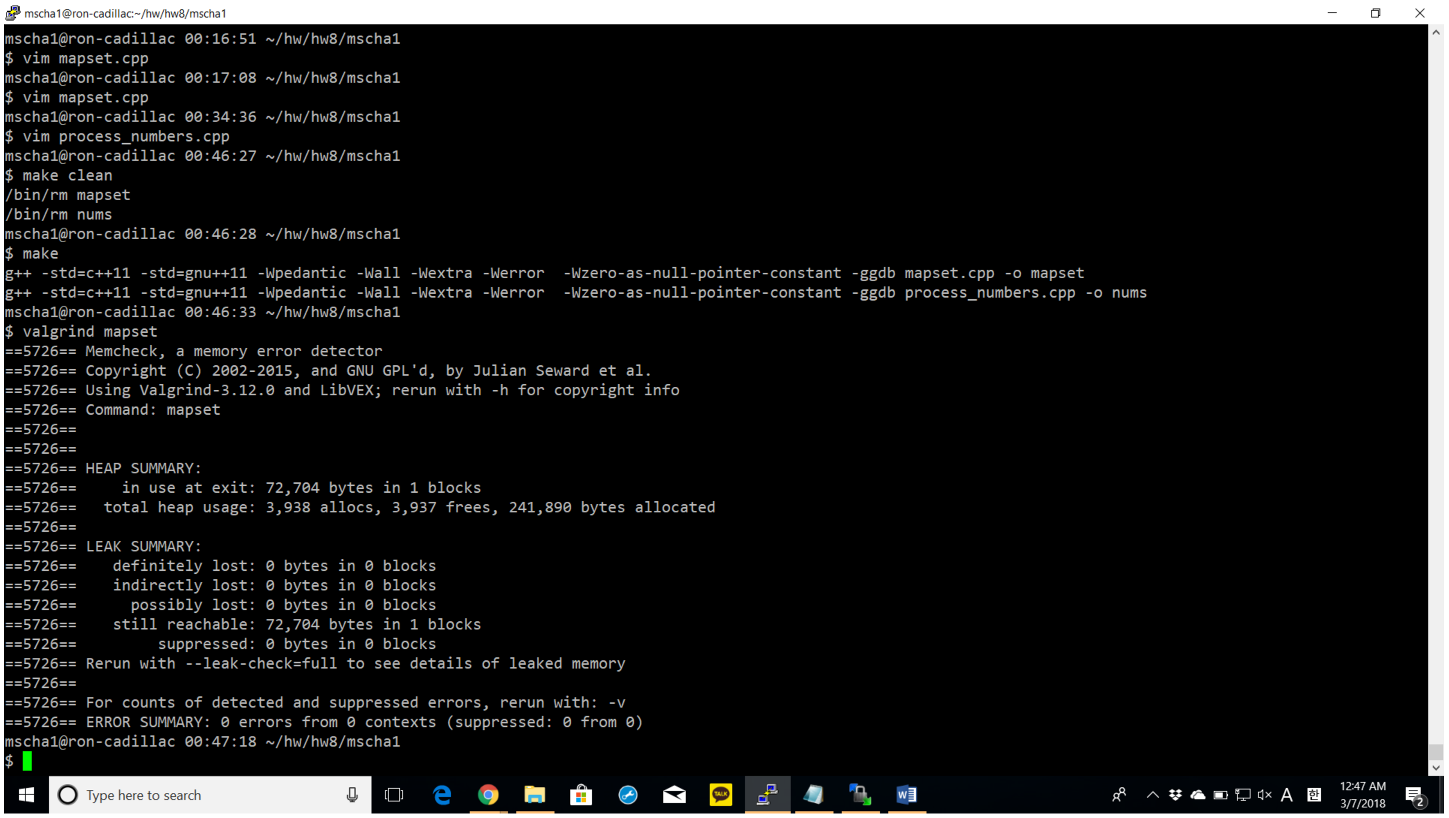
Show screenshot of compiling for both programs with make command



```
$ make
g++ -std=c++11 -std=gnu++11 -Wpedantic -Wall -Wextra -Werror  -Wzero-as-null-pointer-constant -ggdb mapset.cpp -o mapset
g++ -std=c++11 -std=gnu++11 -Wpedantic -Wall -Wextra -Werror  -Wzero-as-null-pointer-constant -ggdb process_numbers.cpp -o nums
mscha1@ron-cadillac 00:46:33 ~/hw/hw8/mscha1
$ 
```

Run your programs with valgrind and include the screenshot for each of the programs

mapset.cpp



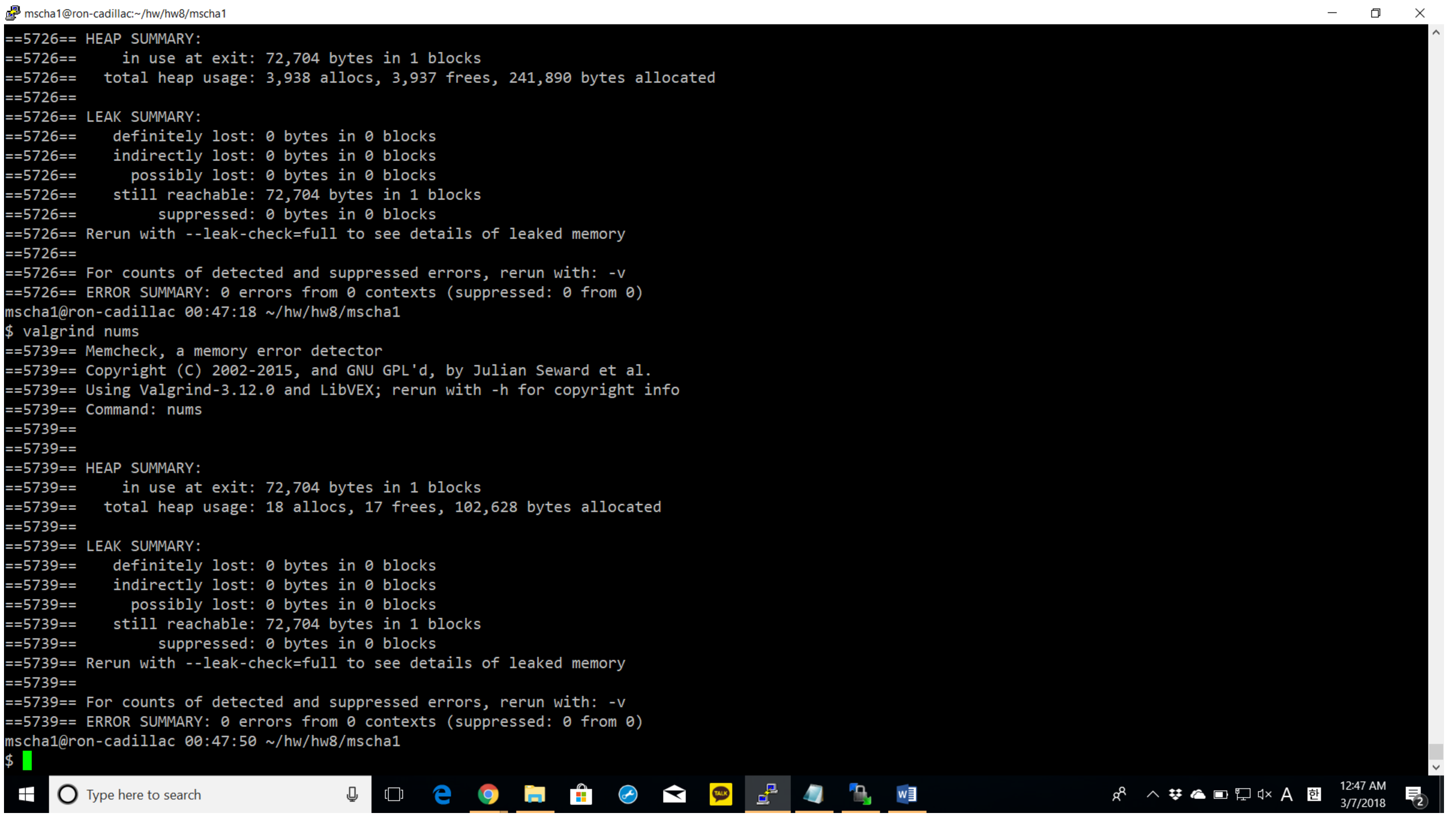
mscha1@ron-cadillac:~/hw/hw8/mscha1

```
mscha1@ron-cadillac 00:16:51 ~/hw/hw8/mscha1
$ vim mapset.cpp
mscha1@ron-cadillac 00:17:08 ~/hw/hw8/mscha1
$ vim mapset.cpp
mscha1@ron-cadillac 00:34:36 ~/hw/hw8/mscha1
$ vim process_numbers.cpp
mscha1@ron-cadillac 00:46:27 ~/hw/hw8/mscha1
$ make clean
/bin/rm mapset
/bin/rm nums
mscha1@ron-cadillac 00:46:28 ~/hw/hw8/mscha1
$ make
g++ -std=c++11 -std=gnu++11 -Wpedantic -Wall -Wextra -Werror  -Wzero-as-null-pointer-constant -ggdb mapset.cpp -o mapset
g++ -std=c++11 -std=gnu++11 -Wpedantic -Wall -Wextra -Werror  -Wzero-as-null-pointer-constant -ggdb process_numbers.cpp -o nums
mscha1@ron-cadillac 00:46:33 ~/hw/hw8/mscha1
$ valgrind mapset
==5726== Memcheck, a memory error detector
==5726== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==5726== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==5726== Command: mapset
==5726==
==5726==
==5726== HEAP SUMMARY:
==5726==     in use at exit: 72,704 bytes in 1 blocks
==5726==   total heap usage: 3,938 allocs, 3,937 frees, 241,890 bytes allocated
==5726==
==5726== LEAK SUMMARY:
==5726==    definitely lost: 0 bytes in 0 blocks
==5726==    indirectly lost: 0 bytes in 0 blocks
==5726==      possibly lost: 0 bytes in 0 blocks
==5726==    still reachable: 72,704 bytes in 1 blocks
==5726==         suppressed: 0 bytes in 0 blocks
==5726== Rerun with --leak-check=full to see details of leaked memory
==5726==
==5726== For counts of detected and suppressed errors, rerun with: -v
==5726== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mscha1@ron-cadillac 00:47:18 ~/hw/hw8/mscha1
$ 
```

Type here to search

12:47 AM
3/7/2018

process_numbers.cpp



mscha1@ron-cadillac:~/hw/hw8/mscha1

```
==5726== HEAP SUMMARY:
==5726==     in use at exit: 72,704 bytes in 1 blocks
==5726==   total heap usage: 3,938 allocs, 3,937 frees, 241,890 bytes allocated
==5726==
==5726== LEAK SUMMARY:
==5726==    definitely lost: 0 bytes in 0 blocks
==5726==    indirectly lost: 0 bytes in 0 blocks
==5726==      possibly lost: 0 bytes in 0 blocks
==5726==    still reachable: 72,704 bytes in 1 blocks
==5726==         suppressed: 0 bytes in 0 blocks
==5726== Rerun with --leak-check=full to see details of leaked memory
==5726==
==5726== For counts of detected and suppressed errors, rerun with: -v
==5726== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mscha1@ron-cadillac 00:47:18 ~/hw/hw8/mscha1
$ valgrind nums
==5739== Memcheck, a memory error detector
==5739== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==5739== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==5739== Command: nums
==5739==
==5739==
==5739== HEAP SUMMARY:
==5739==     in use at exit: 72,704 bytes in 1 blocks
==5739==   total heap usage: 18 allocs, 17 frees, 102,628 bytes allocated
==5739==
==5739== LEAK SUMMARY:
==5739==    definitely lost: 0 bytes in 0 blocks
==5739==    indirectly lost: 0 bytes in 0 blocks
==5739==      possibly lost: 0 bytes in 0 blocks
==5739==    still reachable: 72,704 bytes in 1 blocks
==5739==         suppressed: 0 bytes in 0 blocks
==5739== Rerun with --leak-check=full to see details of leaked memory
==5739==
==5739== For counts of detected and suppressed errors, rerun with: -v
==5739== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mscha1@ron-cadillac 00:47:50 ~/hw/hw8/mscha1
$ 
```

Type here to search

12:47 AM
3/7/2018

copy and title your, odd.txt,even.txt,frequency.txt in the report.

odd.txt

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73
75 77 79 81 83 85 87 89 91 93 95 97 99
```

even.txt

```
-98

-96

-94

-92

-90

-88

-86

-84

-82

-80

-78

-76

-74

-72

-70

-68

-66

-64

-62

-60

-58

-56

-54

-52

-50

-48

-46
```

-44
-42
-40
-38
-36
-34
-32
-30
-28
-26
-24
-22
-20
-18
-16
-14
-12
-10
-8
-6
-4
-2
0
2
4
6
8
10
12
14
16
18
20

88

90

92

94

96

98

100

frequency.txt

```
abacus 1
abbreviations 1
abstract 2
abstracted 2
abstraction 1
accredited 1
acknowledges 1
act 1
action 1
actionscript 1
activities 1
actual 1
ad 1
ada 1
adacore 1
adaptations 1
add 1
addition 1
addresses 1
adopted 1
affects 1
air 1
al-jazari 1
algebra 1
algorithms 4
allow 1
allowed 5
allows 1
along 1
also 2
although 1
analogous 1
```

analysis 1

analytical 2

ancient 2

another 1

antikythera 1

application 2

applications 2

architecture 1

areas 2

arithmetic 1

around 2

art 1

artifacts 1

assembly 6

automata 1

automate 1

babbage 1

basic 1

bc 2

became 2

become 1

becoming 1

beginning 1

bernoulli 1

besides 1

beyond 2

binary 1

build 1

building 1

built 1

c 3

calculate 1

calculating 1

calculation 1

calculations 2

calculator 1

calculators 1

calculus 1

calendars 1

call 1

called 1

cams 1

card 3

cards 8

career 2

cases 1

certification 1

challenges 1

changes 1

charles 1

china 1

circa 1

clearance 1

cloth 1

cobol 1

code 6

coding 2

cognitive 1

combines 1

commercial 1

commonly 1

community 1

company 1

compiler 1

complicated 1

computation 1

computational	1
compute	1
computer	11
computers	3
computing	2
conceive	1
conception	1
condition	1
configuration	1
considerably	1
considered	2
consistent	1
consumption	1
control	4
convenient	1
converted	1
core	1
corrections	1
correctness	1
cost	1
countries	3
covers	1
craft	1
crafted	1
created	1
criteria	1
critical	1
cultures	1
cycle	1
data	4
dates	1
debatable	1
debate	3

debugging	1
decisions	1
defined	1
derived	1
design	1
develop	1
developed	7
developing	1
development	5
device	2
devices	2
devising	1
different	10
differs	1
directly	2
discarded	1
discipline	4
domain	1
dozens	1
drive	1
drum	2
drummer	1
due	1
early	1
easier	1
easily	1
easy	1
economic	1
editors	1
efficient	2
efficiently	1
eg	4
elementary	1

elements 1

employed 1

engine 5

engineer 1

engineering 4

engineers 1

enough 1

entered 1

entering 3

entirely 1

entities 1

environments 1

era 2

error 2

europe 1

even 1

every 1

evolvable 2

executable 1

exhaustion 1

existed 1

expertise 1

expressed 1

extent 2

fact 1

faster 1

final 1

find 1

first 4

follow 1

force 1

form 3

formal 1

format 1

forms 1

formula 2

formulation 1

fortran 3

found 1

foundation 2

founded 1

functional 1

fundamental 1

gears 1

general 3

generating 1

geometry 1

giant 1

given 1

goal 1

good 1

governmentally 1

greater 1

greece 1

habitual 1

hardware 2

haskell 1

herman 1

high 2

high-level 3

higher 1

history 1

hole 1

holes 1

hollerith 2

however 2

human	2
humans	1
hypothesis	1
ibm	3
idea	1
ie	1
illegal	1
implementation	5
importing	1
impractical	1
include	3
including	3
increase	1
increasing	1
increasingly	2
incur	1
india	1
industry	1
inexpensive	1
influences	1
initial	1
input	1
instead	1
institution	1
instruction	2
instructions	7
instruments	2
intent	1
invented	5
invention	1
inventions	1
involves	1
jacquard	3

japan	1
java	1
javascript	1
jobs	1
joseph	1
just	1
keypunch	1
knowledge	1
known	2
kurdish	1
labor	1
language	16
languages	11
larger	1
late	2
later	3
leads	1
leaps	1
learn	1
led	1
less	3
let	1
level	2
levers	1
license	1
licensed	1
licensing	1
likely	1
linguistics	1
list	1
lists	1
little	1
locations	1

logic	1
loom	3
looms	1
lovelace	1
low-level	1
lower	1
lunar-to-solar	1
machine	10
machines	4
made	3
maintaining	1
making	1
management	1
manner	1
many	5
marie	1
mathematician	1
maximum	1
may	2
meant	1
measured	1
mechanical	2
mechanism	2
mechanisms	1
mechanized	1
media	1
medieval	1
medium	1
memory	1
method	1
metonic	1
might	2
model	1

modern 4

mostly 1

much 3

musical 1

mutual 1

name 1

nature 1

necessary 1

need 1

neumann 1

new 1

notation 3

number 2

numbers 2

numerical 2

objective-c 1

objects 1

occasionally 1

often 4

olympiads 1

one 4

oneself 1

ongoing 2

operated 1

operating 1

operation 2

operations 3

opportunities 1

opposed 1

order 1

organize 1

original 2

output 2

outsourcing 1

overhead 1

painstakingly 1

panel 1

panels 1

paper 3

part 1

particular 2

particularly 2

parts 1

pass 1

past 1

pasteboard 1

pattern 2

patterns 3

pegs 1

percussion 1

perfectly 1

perform 1

performing 1

perl 1

phase 1

php 1

physically 1

placed 1

playing 1

plugboard 1

popular 2

possibility 1

postulates 1

power 1

practical 1

practices 1

predetermined 1

prior 1

problem 2

process 6

processing 3

produce 2

producing 1

profession 1

professional 1

professions 1

program 6

programmable 2

programmed 1

programmer 4

programmers 5

programming 24

programs 10

progressed 1

prone 1

punch 1

punched 7

punching 1

purpose 1

python 1

read 1

readable 1

rebuilt 1

record 2

recording 1

referred 1

regarded 1

regulated 1

related 1

relations 1

relatively 1

replace 1

represented 1

representing 2

require 1

required 1

requirements 1

requires 1

reserved 1

resources 1

rhythms 1

ruby 1

s 3

sapirwhorf 1

science 2

scientist 1

security 1

see 1

seemed 1

self-governed 1

sequence 3

sequentially 1

series 1

services 1

sets 2

settled 1

several 1

shortened 1

sizes 1

small 1

smalltalk 1

software 9

solution 1

solving 1

sometimes 1

sorter 1

source 5

speaker 1

speakers 1

special 1

specialized 1

specific 2

specified 1

specify 2

specifying 1

speed 2

spoken 1

sql 1

standardized 1

stands 1

states 2

still 3

storage 1

stored 1

strict 1

subject 1

subjects 1

sumeria 1

surrounding 1

susceptible 1

symbolic 1

symbols 1

synthesis 1

system 2

systems 1

tabulating 1

tabulator 2

takes 1

tape 2

target 1

task 2

tasks 1

technical 1

techniques 1

term 2

terminals 1

terms 2

testing 1

tests 1

text 3

theorized 1

things 1

thought 2

thoughts 1

three 1

thus 2

time 2

total 1

tracked 1

translates 1

translation 1

trials 1

trigger 1

turn 1

two 1

type 1

typically 1

typing 1

underlying 1

understanding 1

unit 2

united 2

upon 1

use 5

used 3

uses 2

using 5

usually 4

utilizing 1

various 2

vary 1

verification 1

visual 1

vital 1

von 1

wage 1

way 1

weaves 1

weaving 1

well 1

whether 1

whole 1

whose 2

widely-used 1

will 1

within 1

without 2

wooden 1

world 3

worlds 1

write 1

writing 3

written 1

wrote 1

x 3

y 1

yield 1