Name: Min Sung Cha ID: 85408485

Include your whole SetList.h and MapArray.h and also main.cpp for the function analysis. And explain how insertion is done in each of your SetList and MapArray.

SetList.h

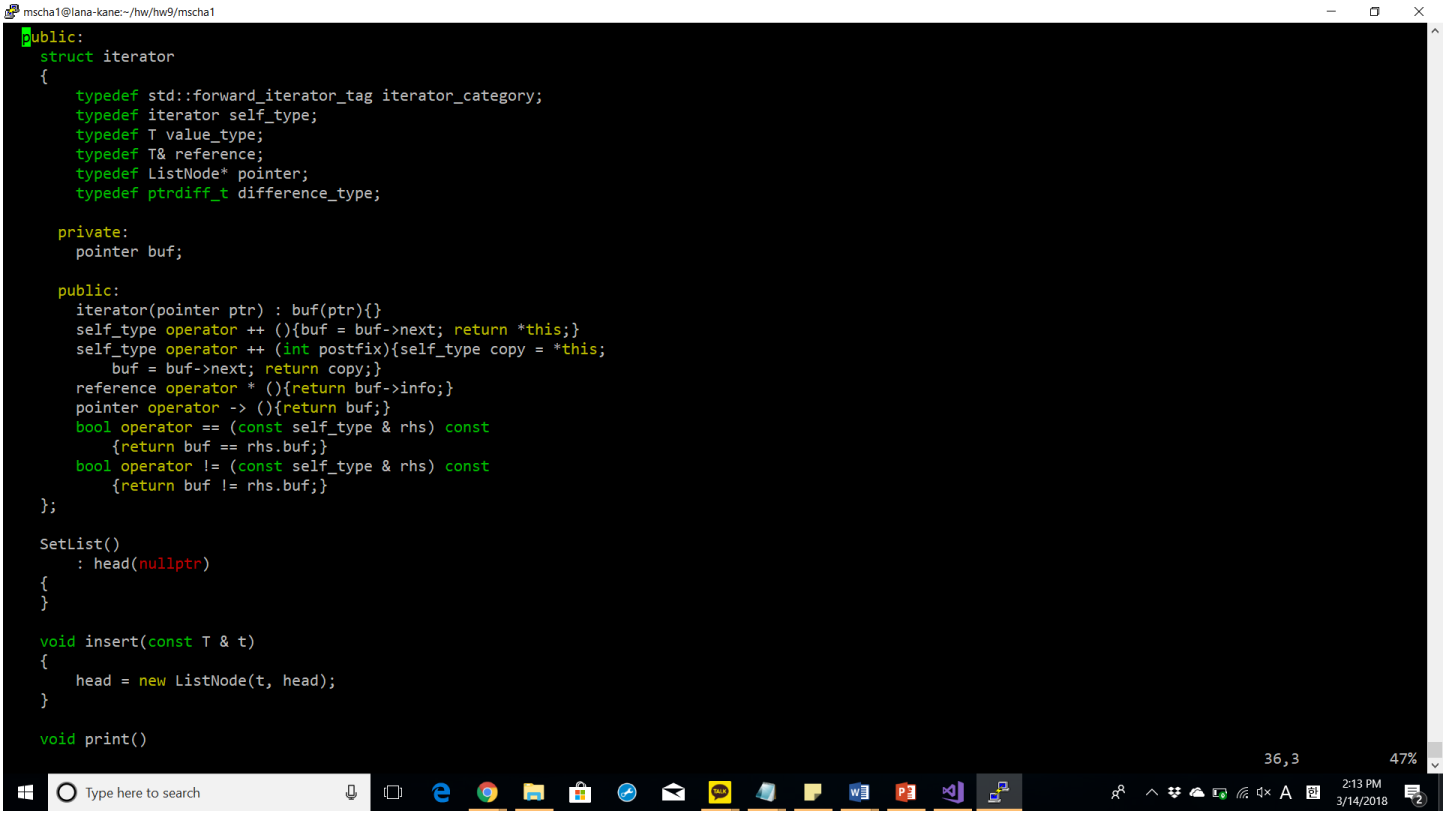
mscha1@lana-kane:~/hw/hw9/mscha1

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

using namespace std;

template
    <typename T>
class SetList
{
  private:
    struct ListNode
    {
        T info;
        ListNode * next;
        ListNode(T newInfo, ListNode * newNext = nullptr)
           : info(newInfo), next(newNext)
        {}

        static void deleteList(ListNode * L)
        {
            if (L != nullptr)
            {
                deleteList(L->next);
                delete L;
            }
        }

    };

    ListNode * head;

  public:
    struct iterator
```

1,1 Top

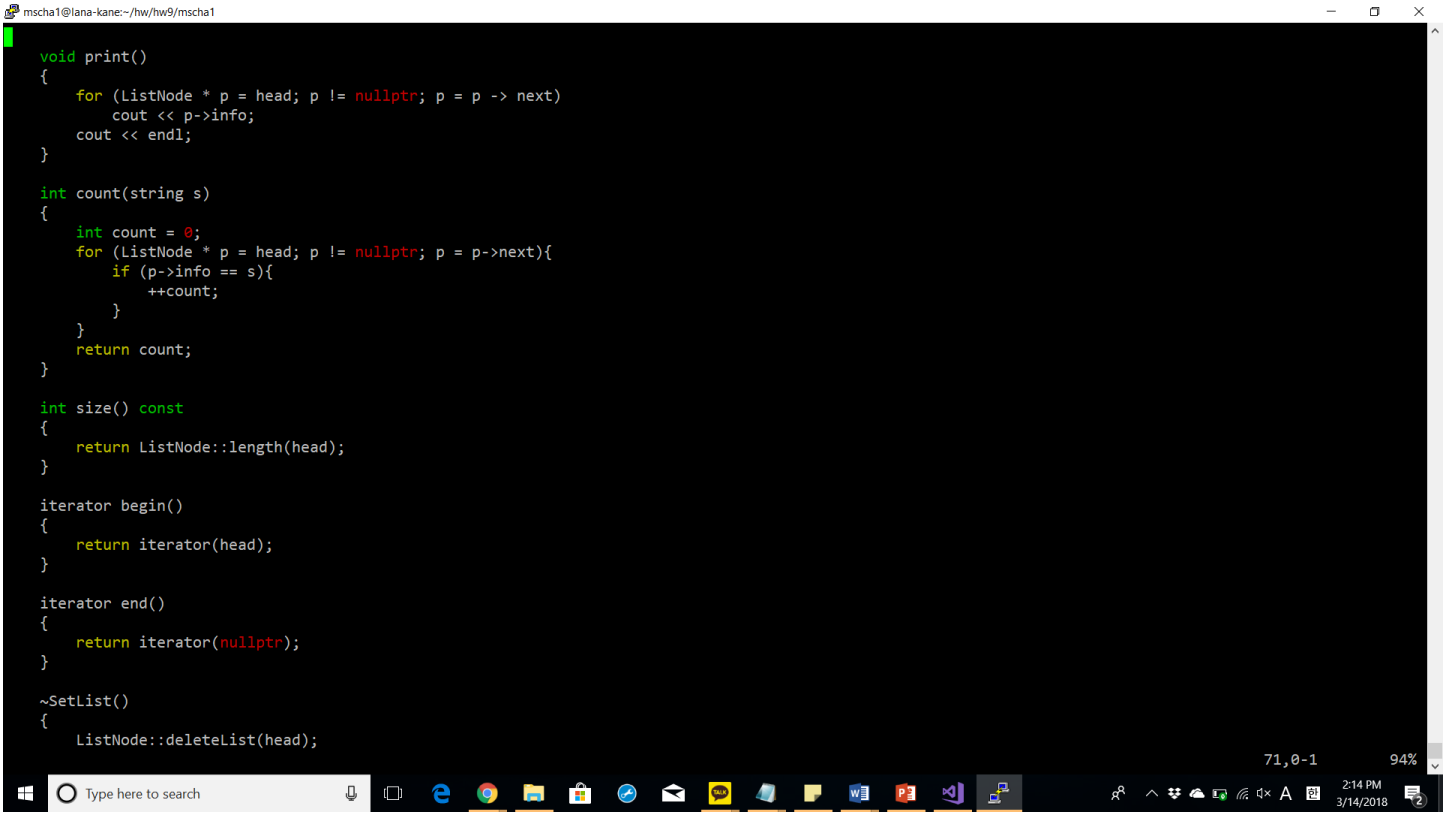
Type here to search 2:13 PM 3/14/2018

deleteList: recursively loops over the linked list and deletes the list nodes until the list node pointer reaches the nullptr.



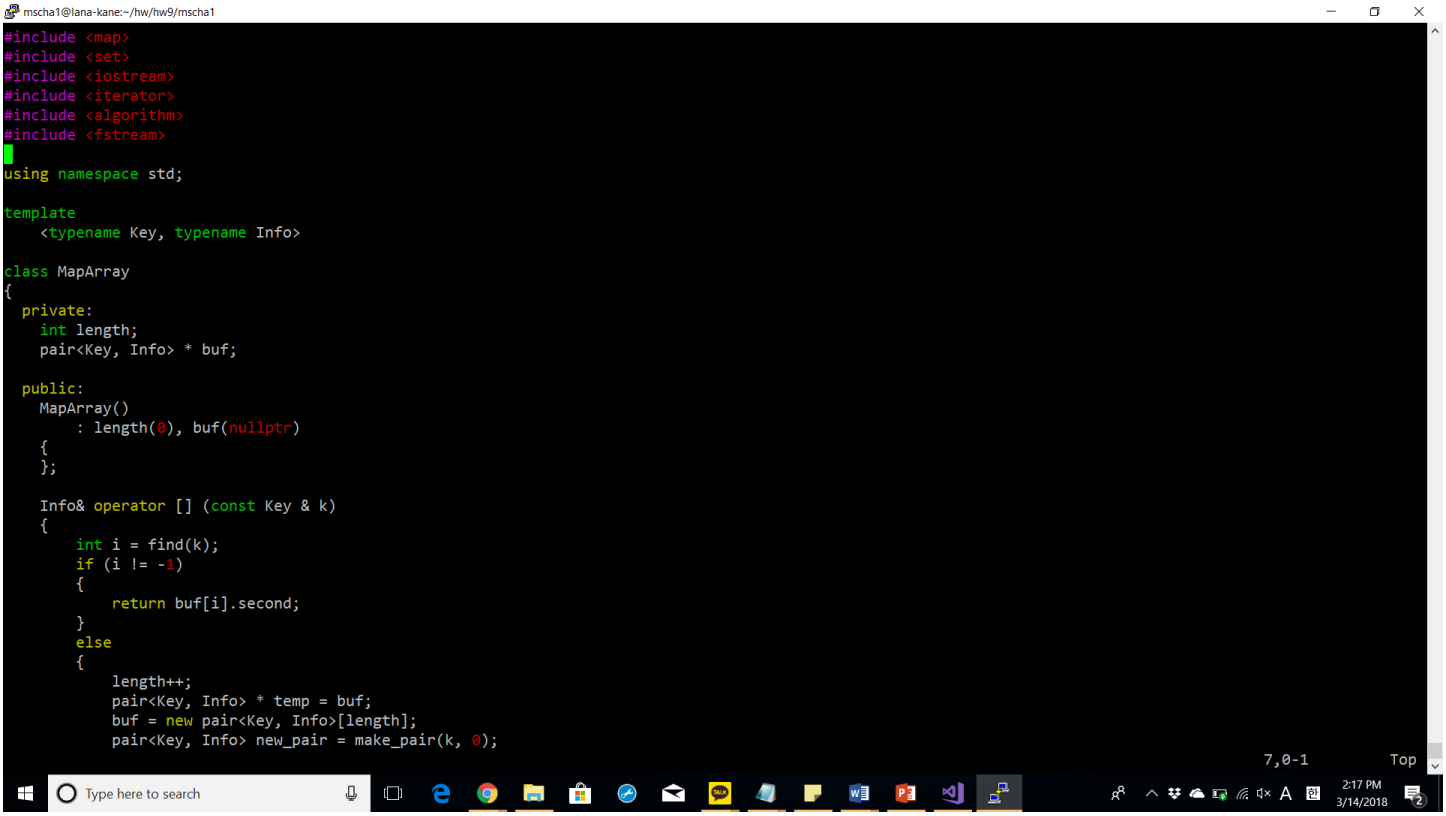
operator ++: list node pointer iterates over the linked list by calling next on the list node.

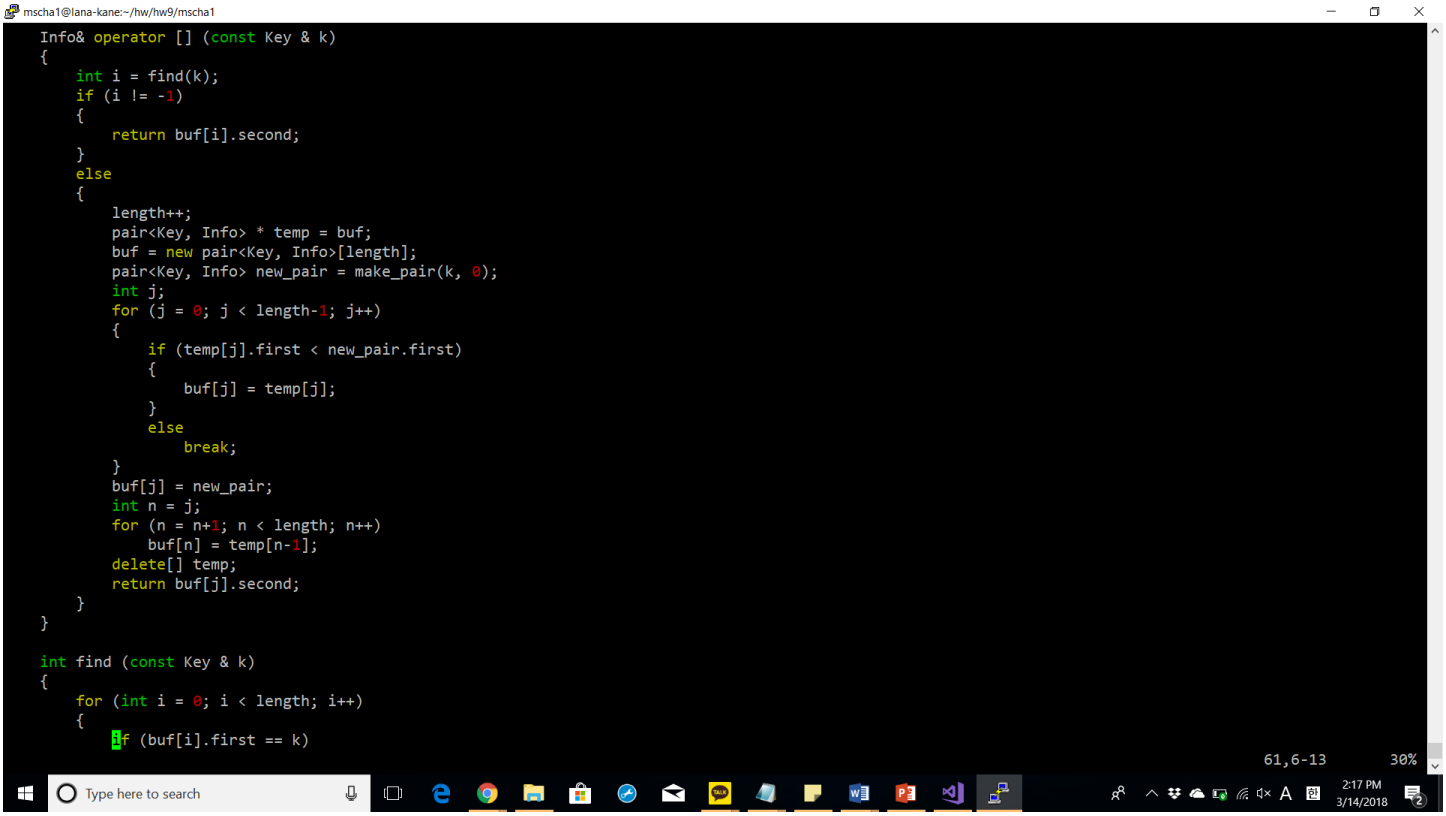
insert: inserts a new list node to the linked list by creating a new list node with the info being the parameter passed in and next being the current head of the linked list.



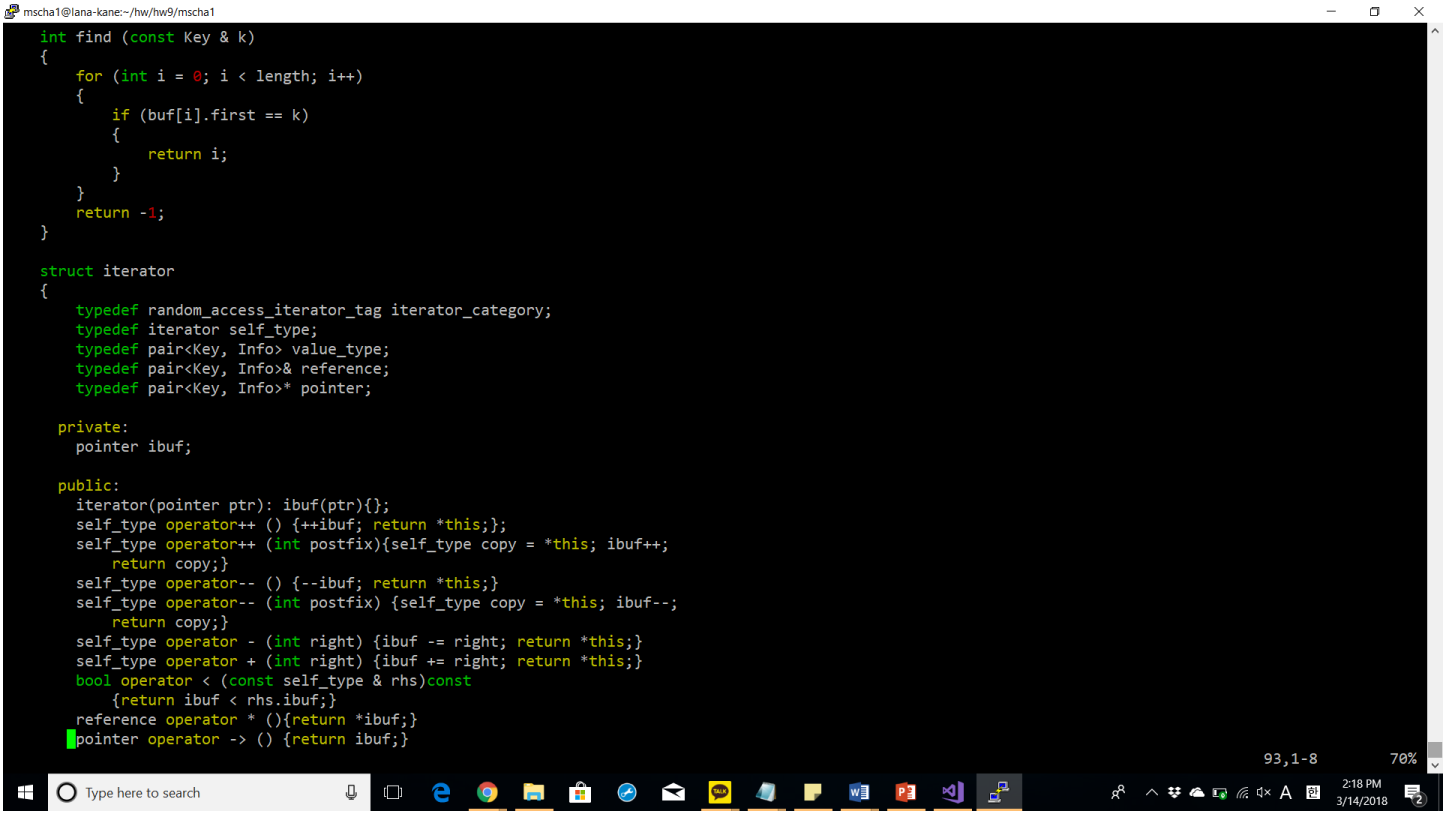
count: iterates over the linked list and if the info of the list node equals the string s parameter the count is increased.

MapArray.h

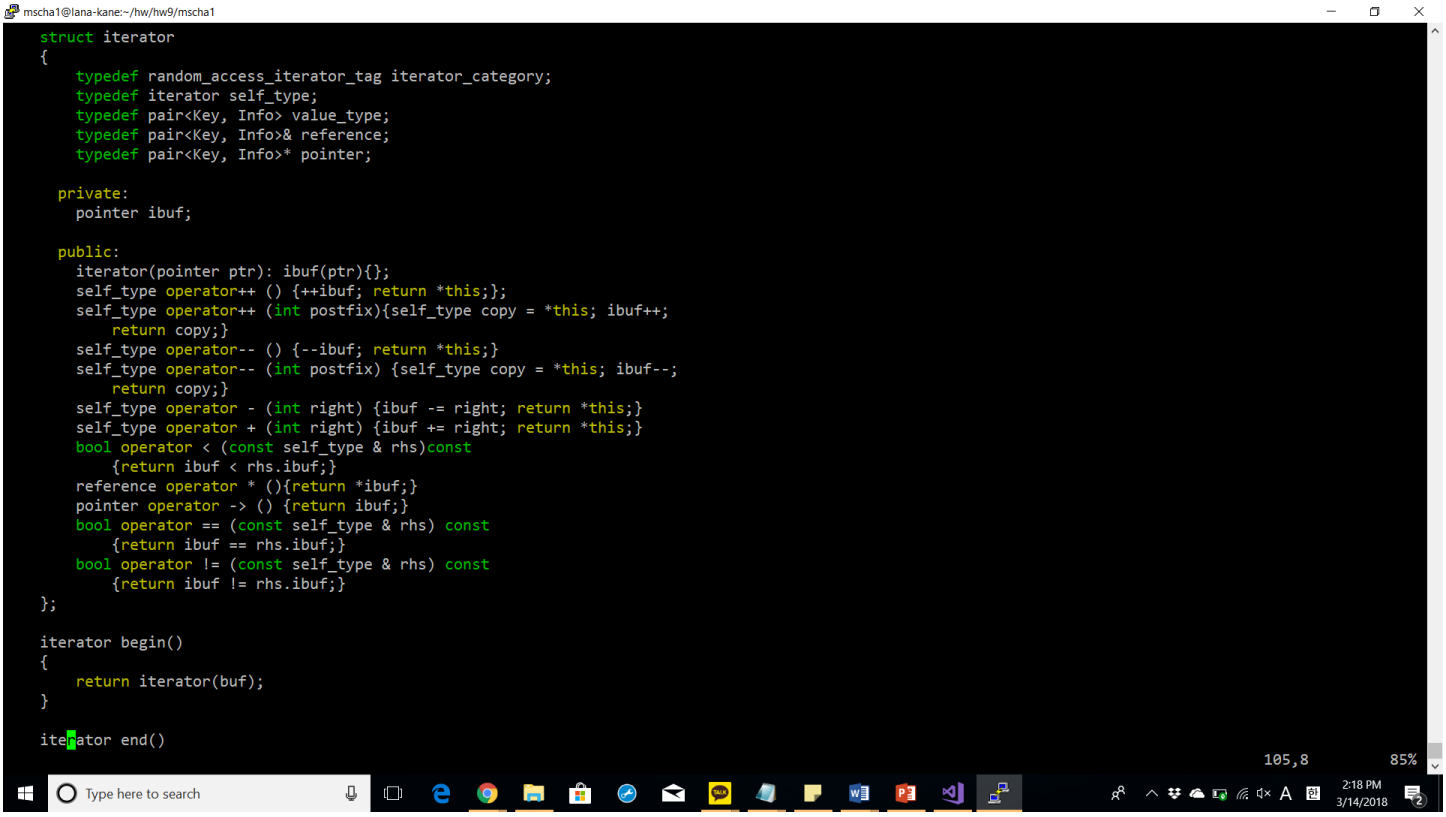


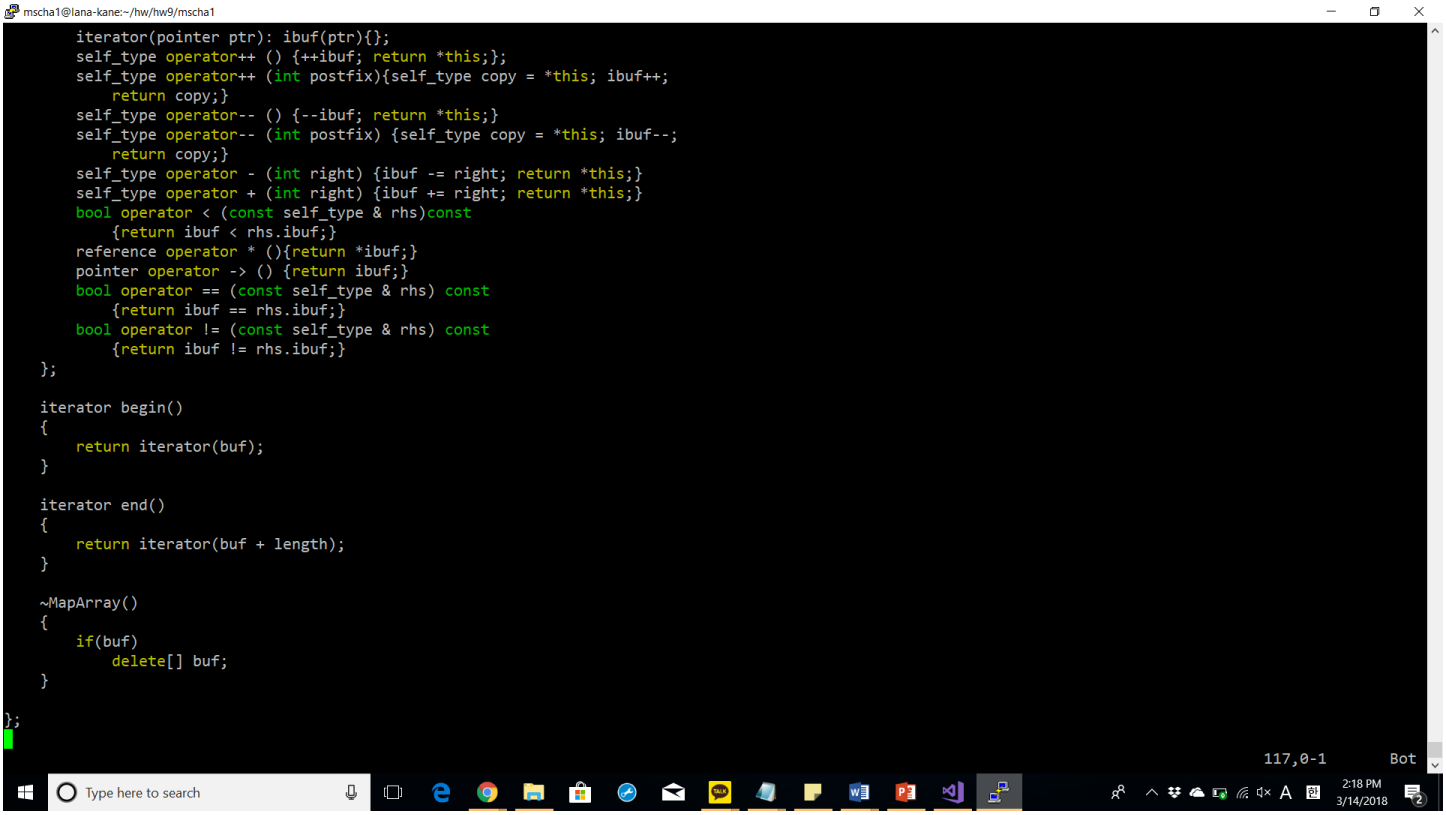


operator []: if the key is found in the MapArray, then the reference of info is returned. Else, the length is increased, the current buf is stored in a temporary pair pointer, and a new array of pairs is constructed. It loops over temp for all the values smaller than the key. The key is inserted. Then, the rest of the temp is copied to the new buf. At the end, temp is deleted and the info of the new pair is returned.

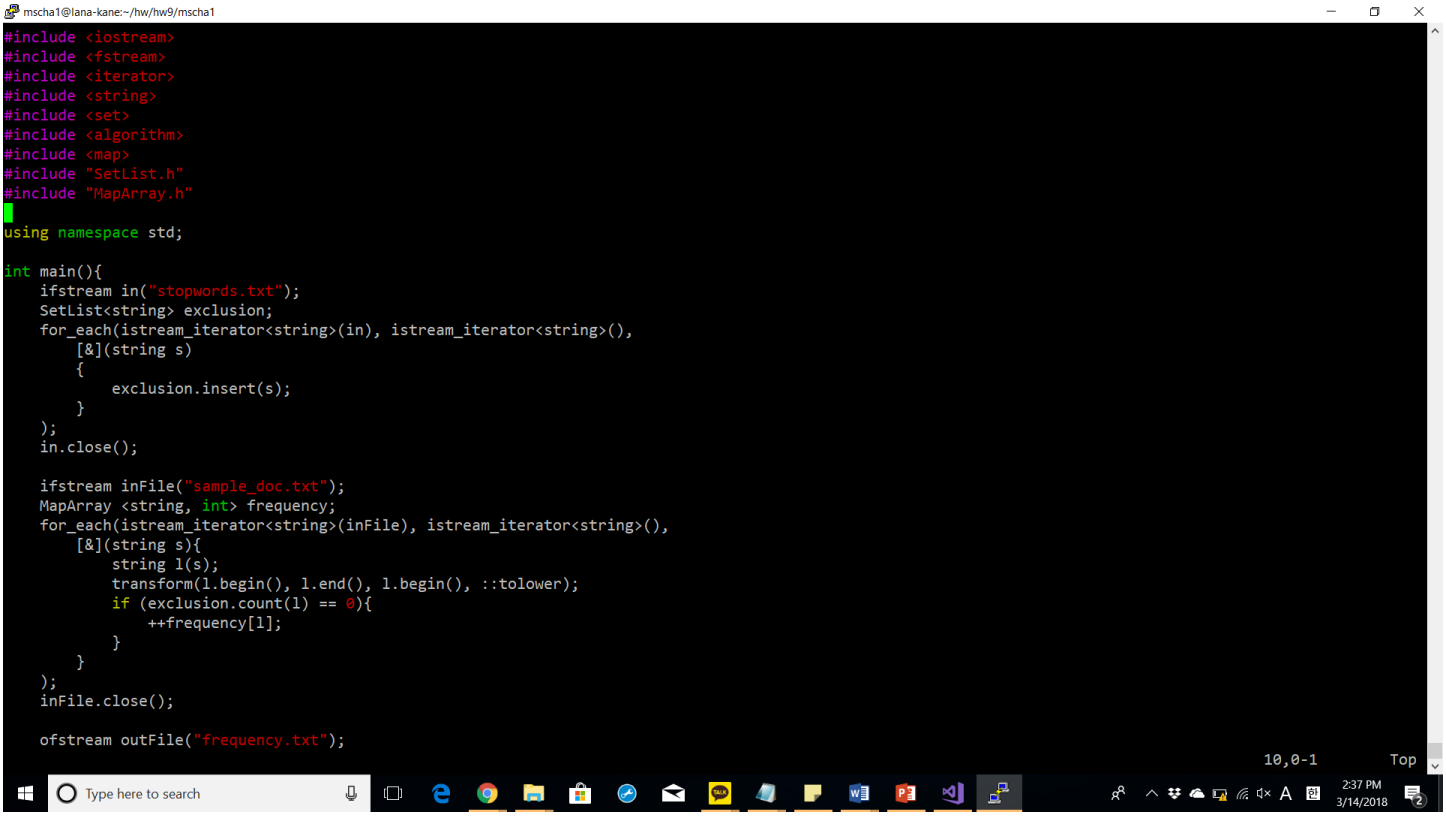


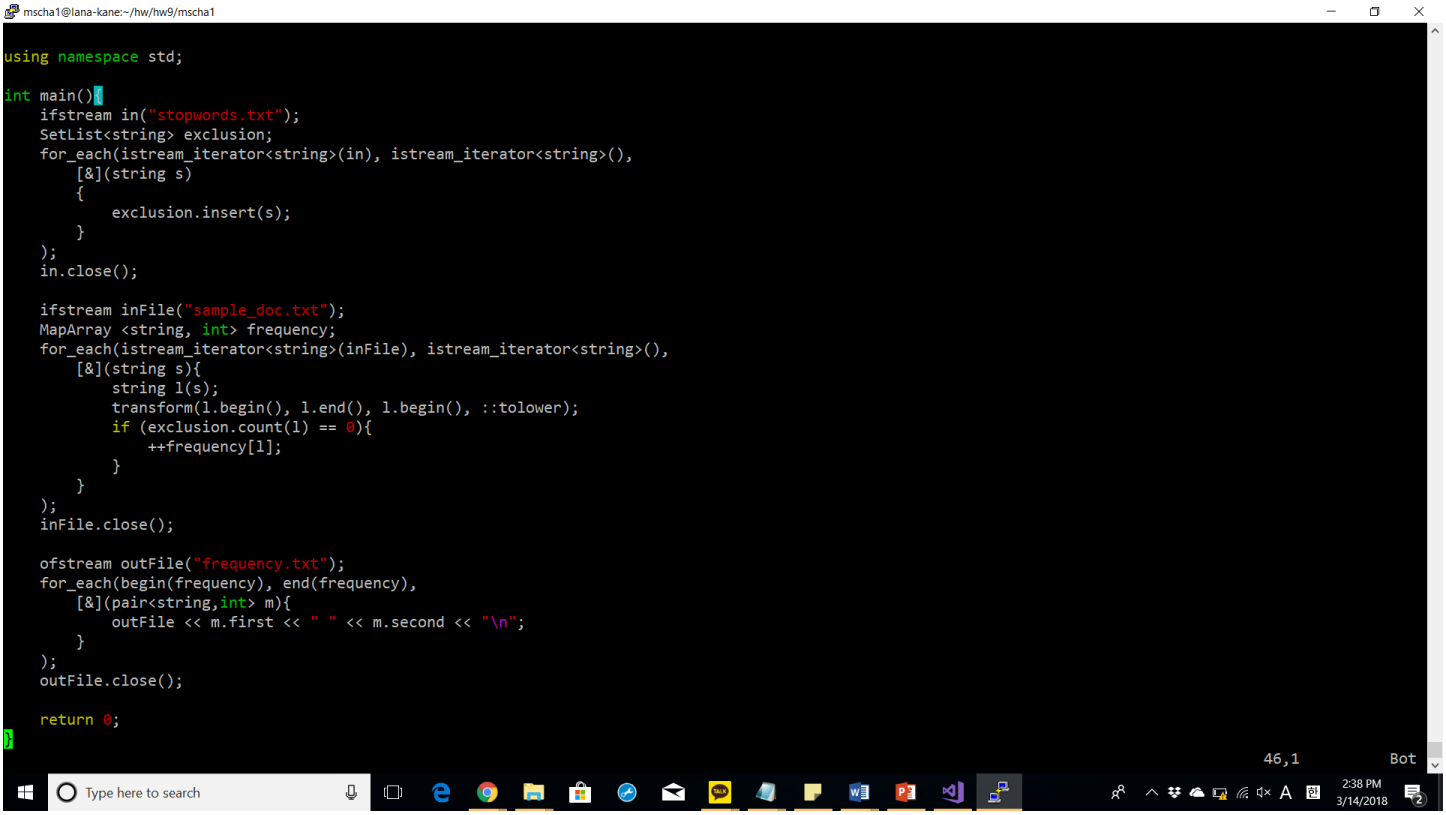
Find: returns the index if the key is found in the array, else -1 is returned.



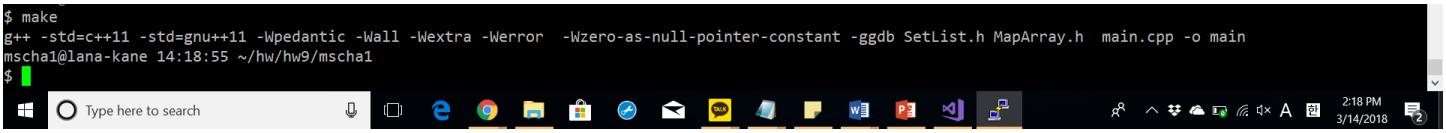

mscha1@lana-kane:~/hw/hw9/mscha1
iterator(pointer ptr): ibuf(ptr){};
self_type operator++ () {++ibuf; return *this;};
self_type operator++ (int postfix){self_type copy = *this; ibuf++;
return copy;}
self_type operator-- () {--ibuf; return *this;}
self_type operator-- (int postfix) {self_type copy = *this; ibuf--;
return copy;}
self_type operator - (int right) {ibuf -= right; return *this;}
self_type operator + (int right) {ibuf += right; return *this;}
bool operator < (const self_type & rhs)const
{return ibuf < rhs.ibuf;}
reference operator * (){return *ibuf;}
pointer operator -> () {return ibuf;}
bool operator == (const self_type & rhs) const
{return ibuf == rhs.ibuf;}
bool operator != (const self_type & rhs) const
{return ibuf != rhs.ibuf;}
};
iterator begin()
{
return iterator(buf);
}
iterator end()
{
return iterator(buf + length);
}
~MapArray()
{
if(buf)
delete[] buf;
}
};
117,0-1
Bot
Type here to search
2:18 PM
3/14/2018

main.cpp

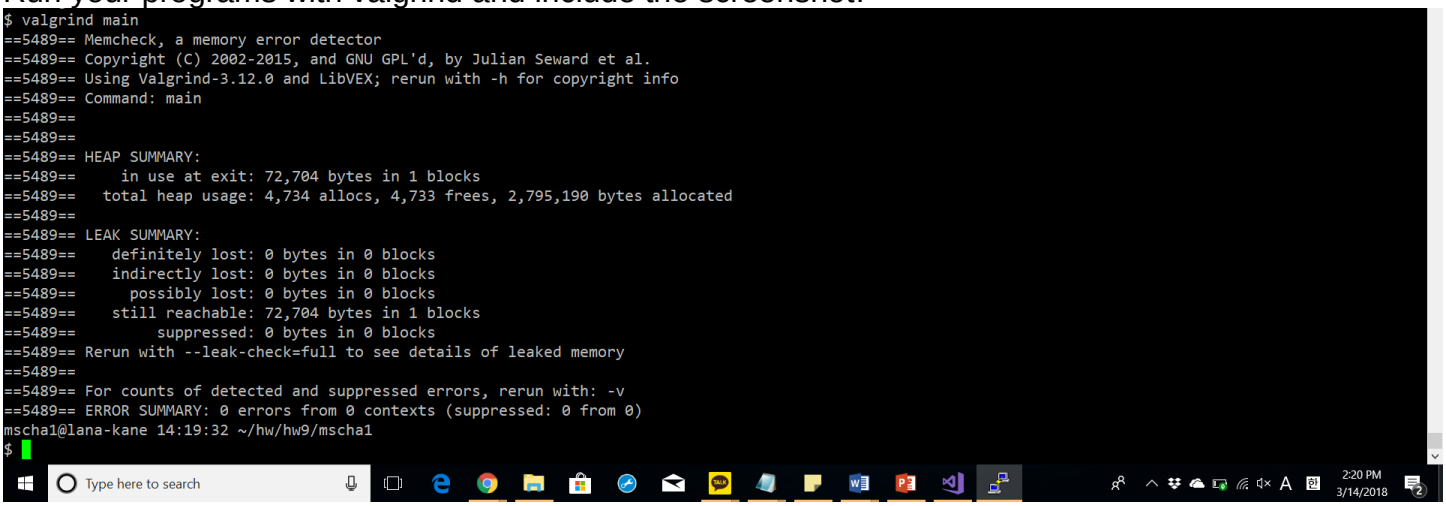

mscha1@lana-kane:~/hw/hw9/mscha1
#include <iostream>
#include <fstream>
#include <iterator>
#include <string>
#include <set>
#include <algorithm>
#include <map>
#include "SetList.h"
#include "MapArray.h"
using namespace std;
int main(){
ifstream in("stopwords.txt");
SetList<string> exclusion;
for_each(istream_iterator<string>(in), istream_iterator<string>(),
[&](string s)
{
exclusion.insert(s);
}
);
in.close();
ifstream inFile("sample_doc.txt");
MapArray <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();
ofstream outFile("frequency.txt");
10,0-1
Top
Type here to search
2:37 PM
3/14/2018


mscha1@lana-kane:~/hw/hw9/mscha1
using namespace std;
int main(){
ifstream in("stopwords.txt");
SetList<string> exclusion;
for_each(istream_iterator<string>(in), istream_iterator<string>(),
[&](string s)
{
exclusion.insert(s);
}
);
in.close();
ifstream inFile("sample_doc.txt");
MapArray <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();
ofstream outFile("frequency.txt");
for_each(begin(frequency), end(frequency),
[&](pair<string,int> m){
outFile << m.first << " " << m.second << "\n";
}
);
outFile.close();
return 0;
}
46,1
Bot
Type here to search
2:38 PM
3/14/2018

show screenshot of compiling for your program with make command



Run your programs with valgrind and include the screenshot.



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

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

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

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