Name: Xiaowei Tan
Partner:

General Problem Description

Write a C++ program, called average, that reads a series of numbers from a file, whose name is specified as an argument, then prints the average of those integers.

Additional Problem Specifics

The program should work with any specified input file which may have any number of numbers and any other characters inside it.

Sample Input

numbers.dat
10.0
1.0
8.0
5.0
4.0
3.0
6.0
2.0
7.0
9.0

Proposed Algorithm

Description:

Declare a double variable average to denote the average, and a long variable count to denote the number of numbers that have been calculated.
Every time the program will take a single line of text from the input file and check whether it is a number. If it is not a number, skip it and continue running. Otherwise, it uses the following formula to calculate the latest average:

$$average_{i+1} = average_i \div (i+1) \times i + num_{i+1} \div (i+1)$$

And then it increases count by 1.

Correctness:

The formula could be deduced from the following formula:

$$\begin{aligned} average_{i+1} &= \frac{\sum_1^{i+1} num}{i+1} \\ &= (average_i \times i + num_{i+1}) \div (i+1) \\ &= average_i \div (i+1) \times i + num_{i+1} \div (i+1) \end{aligned}$$

At the same time, the program will skip all the non-numeric texts so that the count of numbers is correct.
What's more, everytime when it conducts the calculation, it will do the division at first in order to avoid the overflow of the average.
Therefore the result is correct.

Time Complexity:

O(n)

Space Complexity:
O(1)

C++ Implementation of Algorithm

```
ifstream fin(filename.c_str());
string s;
while(getline(fin,s)){
    if(isNum(s)){
        average = atof(s.c_str())/(count+1)+average/(count+1)*count;
        count++;
    }
}
```

Advantages/Disadvantages of Your Algorithm and Any Other Comments

Advantages:

It only takes one line everytime, so compared with the algorithm which takes the whole file into memory, it will save considerable memory definitely.

On the other hand, it can avoid overflow when dealing with oceans of numbers.

Test Cases

- TestCase1.dat which contains ten numbers
    - output we expect (want)

      The average of the 10 numbers in file TestCase1.dat is 5.5
    - output our algorithm produces

      The average of the 10 numbers in file TestCase1.dat is 5.5

- TestCase2.dat which is an empty file
    - output we expect (want)

      The average of the 0 numbers in file TestCase2.dat is 0
    - output our algorithm produces

      The average of the 0 numbers in file TestCase2.dat is 0

- TestCase3.dat which contains 5 numbers and 5 blank lines mixed
    - output we expect (want)

      The average of the 5 numbers in file TestCase3.dat is 21.602
    - output our algorithm produces

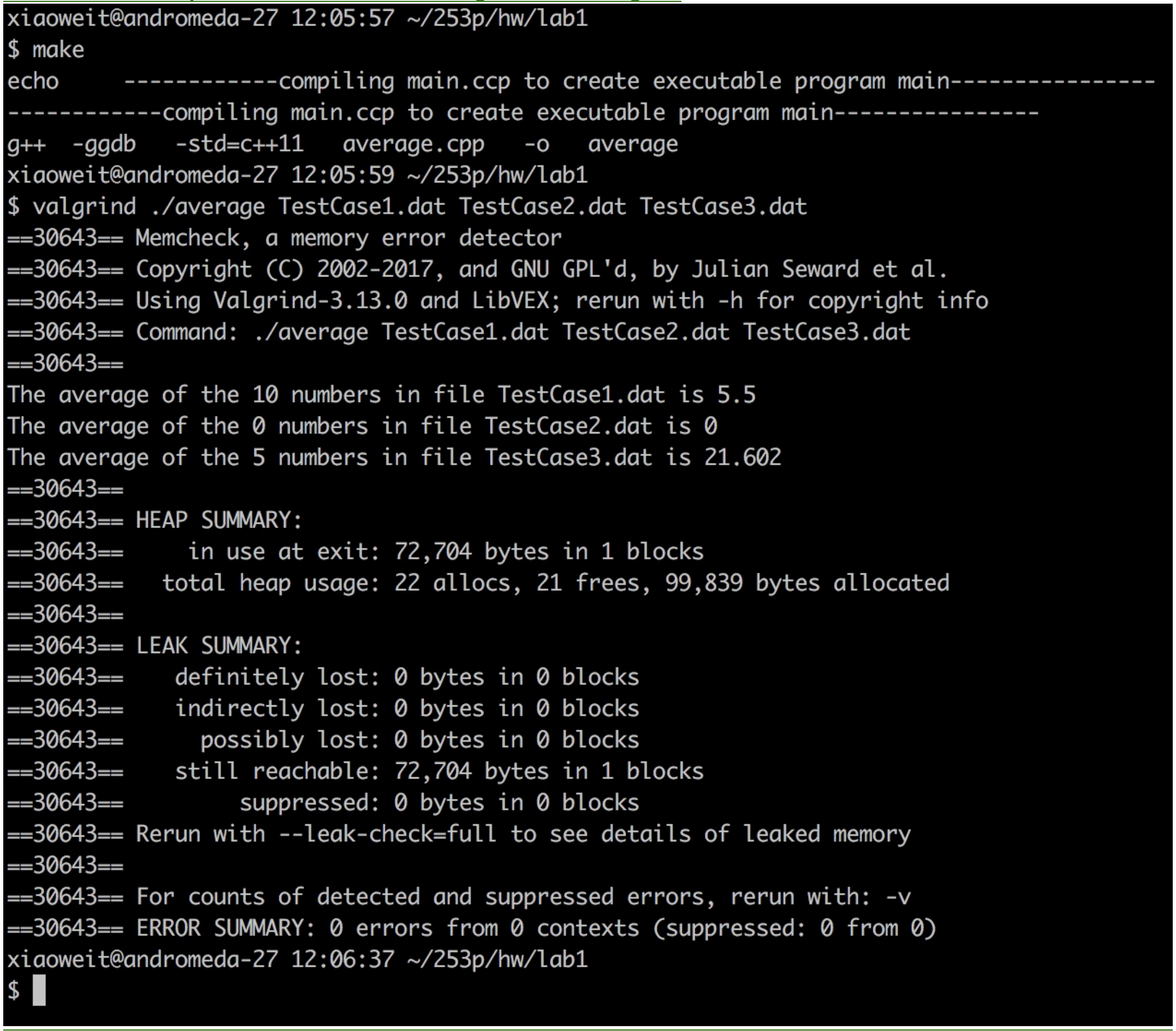
      The average of the 5 numbers in file TestCase3.dat is 21.602

- TestCase4.dat which is provided by instructional staff
    - output we expect (want)

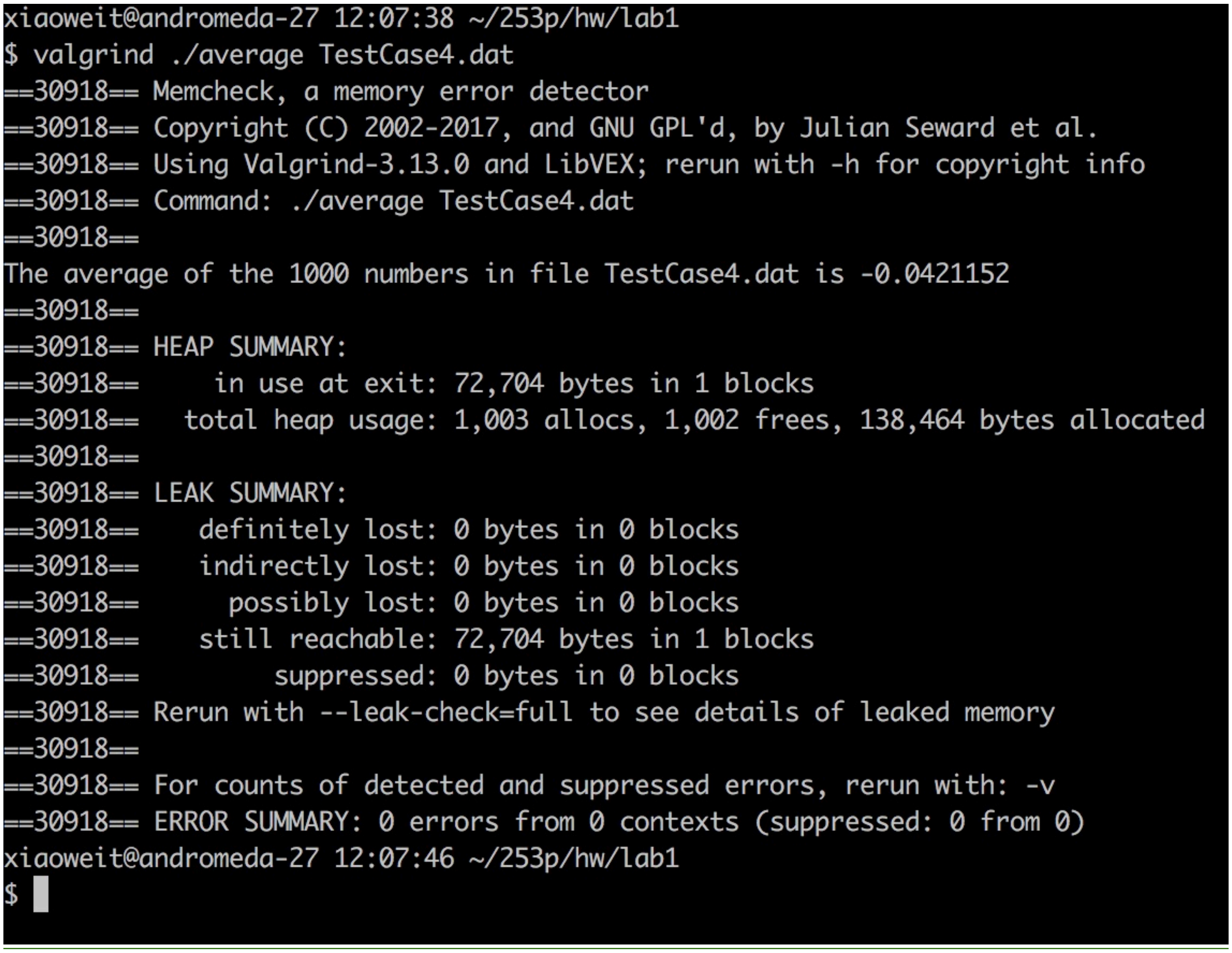
      The average of the 1000 numbers in file TestCase4.dat is -0.0421152
    - output our algorithm produces

      The average of the 1000 numbers in file TestCase4.dat is -0.0421152

Screenshot of Compilation and Execution of Program Under Valgrind



```
xiaoweit@andromeda-27 12:05:57 ~/253p/hw/lab1
$ make
echo     ------------compiling main.ccp to create executable program main----------------
------------compiling main.ccp to create executable program main----------------
g++  -ggdb   -std=c++11   average.cpp   -o   average
xiaoweit@andromeda-27 12:05:59 ~/253p/hw/lab1
$ valgrind ./average TestCase1.dat TestCase2.dat TestCase3.dat
==30643== Memcheck, a memory error detector
==30643== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==30643== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==30643== Command: ./average TestCase1.dat TestCase2.dat TestCase3.dat
==30643==
The average of the 10 numbers in file TestCase1.dat is 5.5
The average of the 0 numbers in file TestCase2.dat is 0
The average of the 5 numbers in file TestCase3.dat is 21.602
==30643==
==30643== HEAP SUMMARY:
==30643==     in use at exit: 72,704 bytes in 1 blocks
==30643==   total heap usage: 22 allocs, 21 frees, 99,839 bytes allocated
==30643==
==30643== LEAK SUMMARY:
==30643==    definitely lost: 0 bytes in 0 blocks
==30643==    indirectly lost: 0 bytes in 0 blocks
==30643==      possibly lost: 0 bytes in 0 blocks
==30643==    still reachable: 72,704 bytes in 1 blocks
==30643==         suppressed: 0 bytes in 0 blocks
==30643== Rerun with --leak-check=full to see details of leaked memory
==30643==
==30643== For counts of detected and suppressed errors, rerun with: -v
==30643== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
xiaoweit@andromeda-27 12:06:37 ~/253p/hw/lab1
$ 
```



```
xiaoweit@andromeda-27 12:07:38 ~/253p/hw/lab1
$ valgrind ./average TestCase4.dat
==30918== Memcheck, a memory error detector
==30918== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==30918== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==30918== Command: ./average TestCase4.dat
==30918==
The average of the 1000 numbers in file TestCase4.dat is -0.0421152
==30918==
==30918== HEAP SUMMARY:
==30918==     in use at exit: 72,704 bytes in 1 blocks
==30918==   total heap usage: 1,003 allocs, 1,002 frees, 138,464 bytes allocated
==30918==
==30918== LEAK SUMMARY:
==30918==    definitely lost: 0 bytes in 0 blocks
==30918==    indirectly lost: 0 bytes in 0 blocks
==30918==      possibly lost: 0 bytes in 0 blocks
==30918==    still reachable: 72,704 bytes in 1 blocks
==30918==         suppressed: 0 bytes in 0 blocks
==30918== Rerun with --leak-check=full to see details of leaked memory
==30918==
==30918== For counts of detected and suppressed errors, rerun with: -v
==30918== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
xiaoweit@andromeda-27 12:07:46 ~/253p/hw/lab1
$
```