Data

Structures

and

Algorithms

Assignment

for

En

ter

prise Web

Development students

Programming

Assignment

1:

Unique

Integers

Task

Description:

Using

any

pro

gramming language of

you

r

choice,

read

a

list

of integers

from

an

input

file.

Generate

an

output

file

having

a list of unique integers present in the

input file.

Instructions

1)

Download

sample

data

for

this

assignment

from

this

location

.

Organize

the

code

and

the

sample

input

into

the following locations:

/dsa

/hw01/code/src/

/dsa

/hw01/sample\_inputs/

/dsa/hw

01/sample\_results

However,

you can choos

e to

organize

your code and data in you

r

own

way but

be share to

organize

things properly so that

I

t be eas

y to understand your work.

2)

Read

a

file

that

has

one

integer

on

each

line.

The

integer

can

be

positive

or

negative.

5

14

5

-

9

62

-

1

-

9

-

9

3)

For

each

input

file

in

the

sample

folder,

you

need

to

output

a

result

file

which

contains

a

list of the unique integers in this file. For example, for input file “sample\_input\_02.txt”,

your result should be in sample\_

results

/sample\_input\_02.txt\_results.txt

4)

The

integers

in

result

file

must

be

sorted

in

increasing

order.

5)

There

must

be

one

line

in

result

file

for

each

unique

integer.

6)

For

example,

if

the

input

is

as

shown

in

number

“

2

”

above,

the

result

must

be:

-

9

-

1

5

14

62

Note

that

the

digits

5

and

-

9

appeared

multiple

times

in

the

input

but

have

been

printed

only once.

7)

Few

sample

input

files

and

result

files

are

given

in

the UniqueInt Sample data file

for t

est

purpos

e

.

8)

Your

code

must

also

handle

following

variations

in

the

input

file:

a)

Integers

in

each

line

can

have

a

white

space

before

or

after

them.

A

whitespace

is

limited to one or more tab, and space characters.

b)

If

there

are

any

lines

with

no

inputs

or

white

spaces,

those

lines

must

be

skipped.

See example input file

s.

c)

If

there

are

any

lines

with

two

integers

separated

by

white

space,

those

lines

must

be skipped.

d)

If

there

are

any

lines

that

contain

a

non

-

integer

input,

those

lines

must

be

skipped.

See example input file

•

Non

-

integer

input

includes

alphabets,

punctuation

marks,

non

-

numeric

values,

floating point numbers.

How

to

write

your

code:

1)

Your

code

needs

to

be

implemented

in

the

file

called

UniqueInt

with the appropriate

extension b

ased on

your s

elected

programming

language

,

y

ou can have t

he following

function

in the f

ile

:

UniqueInt::processFile

(std::

string

inputFilePath,

std::

string

outputFilePath)

UniqueInt::readNextItemFromFile

(

FILE

\*

inputFileStream)

2)

You can create other functions/classes

as needed in your

project folder.

Note:

T

he integers in the input file will range from

-

1023 to 1023. A possible solution can be to

implement

a

boolean

array

containing

a

true

value

if

an

integer

is

seen

and

a

false

value

if

the integer is not seen before.

Grading

method:

1)

If

your

code

runs

and

generates

an

output

file

in

the

correct

format

for

each

input

file,

you get submission points.

2)

If

your

method

generates

correct

results

for

each

test

file,

you

get

points

for

correctness.

3)

We

will

review

your

source

code

to

examine

the

internal

documentation

and

award

points for proper use of meaningful internal documentation.

4)

We will measure the memory consumption in Bytes (submitted\_memory). We will take

the

memory

used

by

our

implementation

(our\_memory).

Memory

score

will

be

based

on

the

ratio: (our\_time /submitted\_time) \*

max score for memory. If your memory usage

is

less than our method, you get more points. The maximum points you can get here is

“max

score

for

memory”.

5)

We will measure the run time in milliseconds (submitted\_time). We will take the time

used by our implementation (our\_time). Your run

-

time score will be based on the ratio:

(our\_time

/submitted\_time)

\*

max

score

for

run

-

time.

The

maximum

points

you

can

get

here is “max score for run

-

time”.

6)

Your code

or implementation

sho

uld sh

ow the memory usage and

runtime

o

n each

sample data fi

le you

run your code against.

7)

See table

below

for

max

score on

run

-

time

and

memory