NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
PRACTICAL EXAMINATION II FOR
Semester 1 AY2021/2022

CS1010 Programming Methodology

October 2021 Time Allowed 2 Hours 30 Minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains 5 questions and comprises 12 printed pages, including this page.
2. The total marks for this assessment is 32. Answer **ALL** questions.
3. This is an OPEN BOOK assessment. You are only allowed to refer to any printed or handwritten materials.
4. You can assume that all the given inputs are valid.
5. You can assume that the types `long` and `double` suffice for storing integer values and real values respectively, for the purpose of this examination.
6. Login to the special account given to you. You should see the following in your home directory:
   - The skeleton code `ease.c`, `max.c`, `cycle.c`, `box.c` and `path.c`
   - A file named `Makefile` to automate compilation and testing
   - A file named `test.sh` to invoke the program with its test cases
   - Two directories, `inputs` and `outputs`, within which you can find some sample inputs and outputs
   - The files `.clang-tidy` and `compile_flags.txt` for configuring `clang` and `clang-tidy`.
7. Solve the given programming tasks by editing the given skeleton code. You can leave the files in your home directory and log off after the examination is over. There is no need to submit your code to GitHub.
8. You can run the command `make` to automatically compile, run the tests (if compiled successfully), and run the command `clang-tidy` on your code.
9. Only the code written in `ease.c`, `max.c`, `cycle.c`, `box.c` and `path.c` directly under your home directory will be graded. Make sure that you write your solution in the correct file. Failure to do so would result in 0 marks for the corresponding question.
10. Note that the marking criteria "Correctness" is defined in the broad sense of using the various programming constructs in C (type, function, variable, loops, conditionals, arithmetic expressions, logical expressions) properly – not just producing the correct output.

11. You should write code that is clean, neat, and readable to get the mark allocated to style. You need to show considerable effort in attempting to solve a question to be awarded the style mark.

12. There is a 1 mark deduction for every warning (including repeated warnings) generated by `clang` and `clang-tidy`. Programs that cannot be compiled would receive 0 marks.

Note: The original Practical Exam II is conducted as an online exam. The questions have been reformatted to fit into paper format.

1 Ease (6 marks)

Singapore is imposing some restrictions on our daily activities due to the COVID-19 pandemic. The restrictions will be eased only if the weekly infection growth rate drops below 1.0.

The weekly infection growth rate can be calculated based on the number of new COVID-19 positive cases over a two-week period and is defined as the ratio of the total new infections in the second week of the period over the total new infections in the first week of the period.

For example, in the last two weeks of October 2020, the numbers of daily new infections are given by the following 14 numbers:

```
3 7 4 6 12 8 10
14 5 3 7 7 7 9
```

There are 50 infections in the first week. In the subsequent week, there are 52 infections. The weekly infection growth rate is $52/50 = 1.04$.

In the last two weeks of October 2021, the numbers of daily new infections are given by

```
3058 2553 3994 3862 3439 3637 3598
3383 3174 3277 5324 3432 4248 3112
```

These numbers give the total weekly new infections of 24141 and 25950 respectively. The weekly infection growth rate is 1.0749.

In either of the two examples above, the government cannot ease the current COVID-19 restriction.

Write a program called `ease` that reads from the standard input 14 non-negative integers that correspond to the number of new daily infections over a two-week period. Your program should print, to the standard output, the following information:

- the total weekly new infections of the first week of the given period,
- the total weekly new infections of the second week of the given period,
- the weekly infection growth rate,
- and the answer of whether the COVID-19 restriction can be eased: "yes" if the weekly infection growth rate is less than 1.0; "no" otherwise.

You can assume that the total number of weekly new infections is larger than zero.

Grading Criteria

Correctness	4
Efficiency	0
Memory Management	1
Style	1
Documentation	0

Memory Management: You do not have to handle the case where memory allocation fails, but you should still ensure that your program does not have any memory leaks if it completes successfully, where applicable. If memory management does not apply to your code, you will get this 1 mark for free.

Efficiency: No mark is allocated for efficiency. Your code is expected to run in O(1) time. We may deduct marks if your implementation performs obvious redundant or duplicated work.

Documentation: You do not have to write Doxygen documentation for each function. You are still encouraged to comment on your code to help the grader understand your intention.

Sample Runs

```
ooiwt@pe101:~$ ./ease
3 7 4 6 12 8 10
14 5 3 7 7 7 9
50
52
1.0400
no
ooiwt@pe101:~$ ./ease
3058 2553 3994 3862 3439 3637 3598
3383 3174 3277 5324 3432 4248 3112
24141
25950
1.0749
no
ooiwt@pe101:~$ ./ease
2 2 2 2 2 2 2
1 1 1 1 1 1 1
14
7
0.5000
yes
```

Comments:

A significant number of students are able to answer this.

The most common deductions are due to:

- usage of `double` instead of `long` to store integers.
- usage of 14 variables instead of using 1-2 arrays
- usage of 12 addition operations instead of loops

2 Max (10 marks)

Write a program `max` that finds the maximum value from a list of n integers.

The integers can be arbitrarily large. Since the types provided by C can only represent a number up to a certain value, we will represent a number using an arbitrarily long string consisting of characters (of type char) '0' to '9' (note: not integer 0 to 9). If a number is negative, it is prefixed with the negative sign '-'.

Recall that C supports comparison operations on char values as well. To compare the numerical values of two digit character, you can use the standard operators, `<`, `<=`, `>`, `>=`, `==` and `!=`.

You will likely need to use the C standard library function `strlen`, which returns you the number of characters in a string (excluding the terminating `'0'`). You can look up the man page of `strlen` to see how to use this function. You might need casting of the return value to avoid warnings.

Your code must include two functions with the declaration below:

```
bool is_larger(char *s1, char *s2);
char *find_max(size_t k, char** list);
```

The function `is_larger` should return `true` if the integer represented by the string `s1` is larger than the integer represented by the string `s2`, and return `false` otherwise.

The function `find_max` should return the string representing the maximum value among the numbers in the given list of size k.

Your code should read, from the standard input,

- an integer k, which is the number of input integers in the list
- followed by k integers, represented as strings.

The inputs do not have leading zeros (except for the number 0).

Special Constraints

You have previously solved a version of this question where the inputs are stored in `long` variables. Submissions that store the input values in an array of `long` will receive 0 marks (for the entire question).

You are not allowed to call methods from the C string library, except `strlen`.

Grading Criteria

Correctness	4
Efficiency	1
Memory Management	2
Style	1
Documentation	2

Efficiency: Your code is expected to run in $O(mn)$ time, where n is the number of integers in the input and m is the length of the longest integers. In addition, we will deduct marks if your code has

obvious redundant or duplicated work. Your code must be correct or almost correct to receive the efficiency marks.

Memory Management: To obtain full marks for memory management, make sure that your code does not have memory leaks and does not crash, even if memory allocation fails.

Documentation: You should document every function in this question according to the CS1010 documentation requirement using the Doxygen format.

Sample Runs

```
ooiwt@pe101:~$ cat inputs/max.1.in
5
310 29 8 7380 546
ooiwt@pe101:~$ max < inputs/max.1.in
7380

ooiwt@pe101:~$ cat inputs/max.2.in
5
-310 -29 -8 -7380 -546
ooiwt@pe101:~$ max < inputs/max.2.in
-8

ooiwt@pe101:~$ cat inputs/max.4.in
10
876298476519873619827651983746592837465293746519872619876398176352987346520345723049568
812876519827641982736598198176239871345823745029834750298347502398457209847520397465209
128374619827364198273651983746529837465109857029387502938475029384750918725091874509287312
128374619827364198273651989450198976891734019827340198237401987324019283750938475029384712
-917820958703495871098475098347520983475209837450298374502983745029837450928374509283740
-949581730498571098475109834751098745019834750983274502983702938750193874501934785019374512
-884750293847510928475029837450293874502983475029387450298374502983745089273450982741
982374509187245091873450928374502983475029384750293847520938475029384750293847502938401
987236598273645982736459827634598716349587263948576239847562938475629834765928376451
ooiwt@pe101:~$ max < inputs/max.4.in
897029387508753945703570239845710984570394857203948571029847503928457203984752093847520912
```

Comments:

The function `find_max` is what you have seen since Week 1:

```
char *find_max(size_t n, char** numbers) {
  char *max_so_far = numbers[0];
  for (size_t i = 1; i < n; i += 1) {
    if (is_larger(numbers[i], max_so_far)) {
      max_so_far = numbers[i];
    }
  }
  return max_so_far;
}
```

The only difference is that instead of using `>` operator, we use the function `is_larger`. Surprisingly, many students do not know how to write `find_max` :(A common bug is to compare two consecutive numbers instead of comparing with the max:

```
if (is_larger(numbers[i], numbers[i-1])) {
```

The function `is_larger` tests if you know how to analyze the problem and come up with conditionals that cover all cases. Most of the bugs here are due to students not covering every possible cases when writing `if` checks.

Here is one implementation that is not elegant but it explicitly list down the cases to make it easier to read:

```
bool is_larger(char *s1, char *s2) {
  if (s1[0] == '-' && s2[0] == '-') {
    return is_larger(s2 + 1, s1 + 1);
  }
  if (s1[0] == '-' && s2[0] != '-') {
    return false;
  }
  if (s1[0] != '-' && s2[0] == '-') {
    return true;
  }
  if (strlen(s1) > strlen(s2)) {
    return true;
  }
  if (strlen(s1) < strlen(s2)) {
    return false;
  }
  for (size_t i = 0; i < strlen(s1); i += 1) {
    if (s1[i] > s2[i]) {
      return true;
    }
    if (s1[i] < s2[i]) {
      return false;
    }
  }
  return false;
}
```

A very common bug is that students wrote only one of the two `if`s inside the loop when comparing digit-by-digit.

Furthermore, many of you shot yourselves in the foot, by not reusing `is_larger` to compare two negative numbers. Instead, what you did is to copy paste the code, and either change `true` to `false` and vice versa, or change `>` to `<` and vice versa. This is very error prone and many bugs caught are due to this. It is better to write one correct version of the function and then reuse it.

3 Cycle (9 marks)

Let the function $g(x)$ be $x^2 + x + 1$.

Consider the function $f(n)$ that computes the sum of $g(x)$ for every digit x in the number n. For example,

$$\begin{aligned} f(421) &= g(4) + g(2) + g(1) \\ &= (16 + 4 + 1) + (4 + 2 + 1) + (1 + 1 + 1) \\ &= 31. \end{aligned}$$

If we apply the function f repeatedly on any integer, two things might happen.

First, we might get back the number that we start with. That is,

$$f(f(f(...f(n)))) = n$$

In this case, n is said to be part of a *cycle*. For example,

$$\begin{aligned} f(4) &= 16 + 4 + 1 = 21 \\ f(21) &= (4 + 2 + 1) + (1 + 1 + 1) = 10 \\ f(10) &= (1 + 1 + 1) + (0 + 0 + 1) = 4 \end{aligned}$$

So the sequence

$$4 \to 21 \to 10 \to 4$$

forms a cycle and 4 is part of a cycle.

Second, for a number that is not part of a cycle, after applying the function f repeatedly, we might reach a number that is part of a cycle. For example, consider the sequence below starting with 499.

$$499 \to 203 \to 21 \to 10 \to 4 \to 21$$

The number 21 is part of a cycle. 499 is not part of a cycle, but after applying f twice starting with 499, we reach the number 21 which is part of a cycle. We say that 499 *moves towards a cycle*.

Furthermore, there is a special cycle of size one, called a *fixed point*, where $f(n) = n$. For example,

$$f(34) = (9 + 3 + 1) + (16 + 4 + 1) = 34$$

34 is said to be a fixed point, and any number that ends up with a fixed point after applying f some number of times is said to *move towards a fixed point*. For example, consider the sequence

$$11 \to 6 \to 43 \to 34 \to 34$$

We say that the number 11 moves towards a fixed point.

Write a program `cycle` that reads in from the standard input, a positive number n.

The program should then print to the standard output the sequence of integers (one per line) after applying f repeatedly (i.e., prints $f(n)$, $f(f(n))$, $f(f(f(n)))$, etc) until a number is repeated or a number is detected to be a fixed point. In the last line of the output, print:

- the string "cycle" if n is part of a cycle;
- the string "fixed point" if n is a fixed point;
- the string "towards cycle" if n is not part of a cycle but it moves towards a cycle;
- the string "towards fixed point" if n is not a fixed point but it moves towards a fixed point.

It is known that the largest number that is part of a cycle is 148.

Grading Criteria

Correctness	4
Efficiency	3
Memory Management	1
Style	1
Documentation	0

Memory Management: You do not have to handle the case where memory allocation fails, but you should still ensure that your program does not have any memory leaks if it completes successfully, where applicable.

Efficiency: Your code should take $O(k)$ time, where k is the number of steps needed to reach a fixed point or to detect a cycle. We may also deduct marks if your implementation performs obvious redundant or duplicated work. Your solution must be correct or almost correct to receive the efficiency mark.

Documentation: You do not have to write Doxygen documentation for each function. You are still encouraged to comment on your code to help the grader understand your intention.

Sample Runs

```
ooiwt@pe101:~$ cycle
4
21
10
4
cycle
ooiwt@pe101:~$ cycle
499
203
21
10
4
21
towards cycle
```

```
ooiwt@pe101:~$ cycle
34
34
fixed point
ooiwt@pe101:~$ cycle
11
6
43
34
34
towards fixed point
```

Comments:

The goal of this question is to see if students how to use an array as a lookup table to reduce the computation. In this case, you should use an array to note down which number has occured before, to avoid an additional loop. This is needed to keep the running time within $O(k)$.

Here is an example of how it can be done:

```
long check_cycle(long n)
{
  bool seen[LARGEST_CYCLE+1];
  for (size_t i = 2; i <= LARGEST_CYCLE; i += 1) {
    seen[i] = false;
  }

  if (n <= LARGEST_CYCLE) {
    seen[n] = true;
  }
  long prev = n;
  long next = f(prev);
  if (prev == next) {
    return FIXED_POINT;
  }
  do {
    prev = next;
    if (prev <= LARGEST_CYCLE) {
      seen[prev] = true;
    }
    next = f(prev);
  } while (prev != next && (next > LARGEST_CYCLE || !seen[next]));

  // { prev == next || seen[next] && next <= LARGEST_CYCLE }

  if (next == prev) {
    return TOWARDS_FIXED_POINT;
  }
  // { seen[next] }
  if (next != n) {
    return TOWARDS_CYCLE;
  }
  return CYCLE;
}
```

Your solution should demonstrate this idea closely to receive the 3 efficiency marks. Some students do this, but instead use a `long` variable. I deduct one mark since a `bool` suffices.

A common error by students is that instead of remember which number has occurred before, they store the list of all numbers generated. There are two issues here.

First, to check if there is a cycle, you need to loop through every number in the list to look for duplicate, this is already $O(k)$, so the total run time would be $O(k^2)$. You would loose the 3 efficiency marks if you do this.

Second, you have no information about what is the maximum number of steps taken, so you don't know how big the array should be. Students come up with all sorts of magic numbers (from 50 to 10000) as the size of the arrays. But these are all arbitrary numbers. A few students did it the hard way – reallocating the array whenever they are out of space – this just adds complexity to the code. Note that 148 is not the maximum number of steps, but rather the maximum number that appear in the cycle.

4 Box (10 marks)

Ash helped Professor Oak to prepare some test cases for a CS1010 assignment. Each test input consists of a canvas, represented by a rectangular area filled with `'.'`, and a box, represented by a rectangular area filled with `'#'`.

The top left corners of the canvas and the box coincide. The box always fits within the canvas.

Ash wrote the following code to generate the test cases. His program takes in four integers, corresponding to the width and height of the canvas, and the width and height of the box, respectively.

The first line of the output contains the width and height of the canvas. The remaining lines show the box and the canvas.

```
#include "cs1010.h"
int main()
{
  long canvas_w = cs1010_read_long();
  long canvas_h = cs1010_read_long();
  long box_w = cs1010_read_long();
  long box_h = cs1010_read_long();
  cs1010_print_long(canvas_w);
  cs1010_print_string(" ");
  cs1010_println_long(canvas_h);
  for (long row = 0; row < canvas_h; row += 1) {
    for (long col = 0; col < canvas_w; col += 1) {
      if (row < box_h && col < box_w) {
        putchar('#');
      } else {
        putchar('.');
      }
    }
    cs1010_println_string("");
  }
}
```

For example, if the input is "10 7 6 4", then Ash's code above generates the following test case:

```
10 7
######....
######....
######....
######....
..........
..........
..........
```

After he generated thousands of test cases, he passed them to Professor Oak. The professor, however, was not impressed. He said, "You should have printed the width and height of the box too!"

But Ash did not record down the boxes' width and height when he generated the test cases. The width and height of each box, however, can be retrieved by reading and processing the test cases that he generated. Ash, therefore, decided that instead of re-generating all the test cases, he would write a second program that reads in each test case and finds the width and height of each box.

For example, if the test case is:

```
10 7
######....
######....
######....
######....
..........
..........
..........
```

Then, his second program should output "6 4".

Ash could write a program that solves this in $O(m+n)$ time quickly, where m and n are the width and height of the canvas. But, to impress the professor, Ash decided to solve it in $O(logm + logn)$ time. He asked you for help.

Help Ash by writing a program `box` that reads the following from the standard input:

- two positive integers m and n, that correspond to the width and height of the canvas;
- the next n lines of the input consist of strings that represent the box and the canvas

Your program should write to the output, the width, and height of the box, on the same line separated by a space.

Grading Criteria

Correctness	3
Efficiency	5
Memory Management	1
Style	1
Documentation	0

Efficiency: Your code should take O(log m + log n) time. An O(m + n) solution is trivial and will receive 0 marks for efficiency. Any solution slower than O(log m + log n) but faster than O(m + n) will receive partial marks for efficiency. Your solution must be correct or almost correct to receive the efficiency mark.

Memory Management: You do not have to handle the case where memory allocation fails, but you should still ensure that your program does not have any memory leaks if it completes successfully, where applicable.

Documentation: You do not have to write Doxygen documentation for each function. You are still encouraged to comment on your code to help the grader understand your intention.

Sample Runs

```
ooiwt@pe101:~$ box
10 7
######....
######....
######....
######....
```

```
..........
..........
..........
6 4
ooiwt@pe101:~$ box
1 1
#
1 1
ooiwt@pe101:~$ box
2 2
#.
..
1 1
```

Comments:

Many students demonstrated that they know how to apply binary search to solve this problem. Students who solved it with binary search receives 5 marks for efficiency, even if there are bugs. Common bugs involve accessing `mid + 1` or `mid - 1`, which may be out of the bound of the arrays.

Here is one way to write the binary search algorithm. This version keeps narrowing the range until there is only one element left, which is either the first `.` or the edge of the canvas.

```
size_t search_row(char **row, size_t start, size_t end)
{
  if (end == start) {
    return start;
  }
  size_t mid = (start + end)/2;
  if (row[0][mid] == '#') {
    return search_row(row, mid+1, end);
  }
  return search_row(row, start, mid);
}
```

Some students solve it with two trivial loops, giving a

$$O(m+n)$$

solution.

5 Path (10 marks)

One of the important operations in contact tracing during a pandemic is to determine the path of transmission: how did the virus go from person A to person B? Does it go from A to B directly, or through one or more intermediate individuals?

Suppose we are given the information on who is in close contact with who. Given a source A and a destination B, we want to print out all possible paths of transmissions from A to B.

We assume that "close contact" is a symmetric relation. If A is a close contact of B, then B is a close contact of A too. Because of this, we can represent the contact traces among n people as a lower triangular matrix (using a jagged 2D array). A proper type to store in each element of the matrix is bool. To simplify our life, however, we store each element of the matrix as a char, with '1' representing a close contact, '0' otherwise. The contact traces for n people is thus an array of n strings, each string containing characters of '0' and '1' only. The first row of the matrix is a string of length one; the second row is of length two; the third row, length three, etc. The last character of each string (i.e., the diagonal of the matrix) is 1 since everyone has contact with him/herself.

We assume the virus transmits between close contacts only. We represent each person with an id 0, 1, 2, ..., etc.

Suppose we have the following contact traces. The person with id i has their information stored in Row i and Column i. Recall that if Row i and Column j is 1, it means that Person i is a close contact of Person j.

```
1
01
111
```

The contact traces above indicates that Person 2 is a close contact of Person 0 and Person 1. But Person 0 and Person 1 are not in close contact with each other. Suppose we query how the virus goes from Person 0 to Person 1, there is only one possible path:

$$0 \rightarrow 2 \rightarrow 1$$

As another example, the contact traces below shows a third person, Person 3.

```
1
01
111
1111
```

Person 3 is a close contact of Person 0, Person 1, and Person 2. Now, there are four possible transmission paths from Person 0 to Person 1:

$$0 \rightarrow 2 \rightarrow 1$$

$$0 \rightarrow 2 \rightarrow 3 \rightarrow 1$$

$$0 \rightarrow 3 \rightarrow 1$$

$$0 \to 3 \to 2 \to 1$$

Write a program `path`, that reads from the standard input:

- a positive integer n,
- followed by n lines of strings consisting of '1' or '0' representing the contact traces of these n people,
- followed by two positive integers i and j, representing the ids of a pair of people we are interested in querying.

Print, to the standard output, all possible transmission paths from Person i to Person j. Print one path per line. Each path consists of a sequence of person ids separated by a space. The paths should be printed so that they satisfy the following order: let P_1 and P_2 be two paths, and a_1 and a_2 are the first two people on the transmission path P_1 and P_2 that differ respectively. If $a_1 < a_2$, then P_1 should appear before P_2. For instance, "0 2 1" should appear before "0 2 3 1", since the first two people who differ are 1 (from "0 2 1") and 3 (from "0 2 3 1") and $1 < 3$.

Print nothing if there is no possible transmission path from Person i to Person j.

Grading Criteria

Correctness	5
Efficiency	3
Memory Management	1
Style	1
Documentation	0

Efficiency: Your code should take O(n(n-2)!) time: O((n-2)!) to find all possible paths and an additional O(n) to print each path. Any solution slower than O(n(n-2)!) will receive 0 marks for efficiency. Your solution must be correct or almost correct to receive the efficiency mark.

Memory Management: You do not have to handle the case where memory allocation fails, but you should still ensure that your program does not have any memory leaks if it completes successfully, where applicable.

Documentation: You do not have to write Doxygen documentation for each function. You are still encouraged to comment on your code to help the grader understand your intention.

Sample Runs

```
ooiwt@pe101:~$ cat input/path.1.in
4
1
01
111
1111
0 1
ooiwt@pe101:~$ path < input/path.1.in
0 2 1
```

```
0 2 3 1
0 3 1
0 3 2 1
```

Comments:

A handful of students solved this, but only a few submitted bug-free version.

This question is an combination of `social` and `permutation`. The `social` component includes looking through the jagged array and find connections. The list of all possible paths is actually subset of permutation, but we only continue if there is a connection.

The function to print all paths looks like this:

```
void print_all_paths(size_t n, char **network, bool *visited,
    size_t *path, size_t src, size_t dst, size_t len) {
  if (src == dst) {
    print_one_path(path, len);
    return;
  }
  for (size_t i = 0; i < n; i += 1) {
    if (i != src) {
      if (is_contact(network, src, i) && !visited[i]) {
        path[len] = i;
        visited[i] = true;
        print_all_paths(n, network, visited, path, i, dst, len + 1);
        visited[i] = false;
      }
    }
  }
}
```

Common errors:

- Not using a boolean array to remember which one is visited, but do a search instead. This increases the running time.
- Trying to print the prefix while searching. Just like `stone` and `permutation`, we can't print the prefix until we reach the bottom of recursion. Otherwise, the prefix is only printed once. The solution should "build" the path one by one and then print at the end.

END OF PAPER

This page is intentionally left blank.