NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
FINAL ASSESSMENT FOR
Semester 1 AY2023/2024

CS1010 Programming Methodology

December 2023 Time Allowed 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains 16 questions and comprises 10 printed pages, including this page.
2. Write or shade your answers in the answer sheet provided.
3. Please write and shade your student number in the corresponding box in the answer sheet.
4. The total marks for this assessment is 80. Answer **ALL** questions.
5. This is an **OPEN BOOK** assessment.
6. You can assume that in all the code given: (i) no overflow nor underflow will occur during execution; (ii) all given inputs are valid and fits within the specified variable type; (iii) compile without syntax errors; and (iv) all the required header files are already included.

Part I
Multiple Choice Questions (36 points)

- For each of the questions below, **shade your answer on the answer sheet.** Each question is worth 3 points.
- If multiple answers are equally appropriate, pick one and shade the chosen answer on the answer sheet. Do NOT shade more than one answer per question.
- If none of the answers is appropriate, shade X on the answer sheet.

1. (3 points) Consider a boolean function whose return value depends on three boolean variables, `x`, `y`, and `z`. The *only* three combinations that cause the function to return `true` are listed in the table below:

`x`	`y`	`z`
`true`	`true`	`true`
`true`	`false`	`true`
`false`	`true`	`true`

Which of the following `return` statement correctly implements the boolean function?

A. `return x || y || z;`

B. `return x && y && z;`

C. `return (x || y) && z;`

D. `return (x && y) || z;`

E. `return z;`

Shade X on the answer sheet if none of the choices above is correct.

Solution: C.

`z` must be `true` for the expression to return `true`. On the other hand, either one of `x` and `y` must be true for the expression to return `true`. The correct logical expression that represents the table is therefore `(x || y) && z`.

Almost all students (302/313) got this question correct.

2. (3 points) Consider the function below:

```
long ash(long m, long n) {
  long count = 0;
  while (m < n) {
    count++;
    m++;
    n--;
  }
  return count;
}
```

Suppose that `m` < `n`. Which of the expressions below best describes what the function computes?

A. `(n - m) / 2`

B. `(n - m) / 2 - 1`

C. `(n - m) / 2 + 1`

D. `(n - m + 1) / 2`

E. `(n - m - 1) / 2`

Shade X on the answer sheet if none of the choices above is correct.

Solution: D.

If `n - m` is odd, the function returns `(n - m) / 2 + 1`; else `(n - m) / 2`. The expression `(n - m + 1) / 2` correctly captures both cases.

This is a relatively straightforward question that assesses the students' ability to trace a loop and the understanding of the `/` operator. 73% of students got this question correct.

3. (3 points) Consider the function below:

```
void eng(long m, long n) {
  do {
    m = do_something(m);
    n = do_something(n);
  } while (m != 0 && n > 0);

  if (m != 0 && n > 1) {
    m = 1; // Line W
  } else if (m == 0) {
    n = 1; // Line X
  } else if (n <= 0) {
    n = 0; // Line Y
  } else {
    m = n; // Line Z
  }
}
```

Which of the statements, labeled W, X, Y, and Z will never be executed for any values of `m` and `n`?

A. Line W

B. Line X

C. Line X and Line Y

D. Line Y and Line Z

E. Line W and Line Z

Shade X on the answer sheet if none of the choices above is correct.

Solution: E.

After the while loop, either `m == 0` or `n <= 0` is true. So Line W will never be executed. Since either Line X or Line Y will be entered, the last `else` is redundant. Line Z will never be executed.

Only 54% of students got this question correct. This question assesses if students know about reason about the properties of a piece of code.

4. (3 points) Consider the function `vee` below:

```c
void vee(long h) {
  for (long i = 0; i < h; i += 1) {
    for (long space = 0; space <= i-1; space += 1) {
      putchar(' ');
    }
    putchar('#');
    for (long space = 0; space <= f(i); space += 1) {
      putchar(' ');
    }
    putchar('#');
    for (long space = 0; space <= i-1; space += 1) {
      putchar(' ');
    }
    putchar('\n');
  }
  putchar('\n');
}
```

The function will draw the following patterns, as examples.

When h is 3

```
#    #
 #  #
  ##
```

When h is 4

```
#      #
 #    #
  #  #
   ##
```

When h is 5

```
#        #
 #      #
  #    #
   #  #
    ##
```

To achieve the pattern above, what should the function call `f(i, h)` return?

A. `(2 * h)`

B. `(2 * h) - i`

C. `(2 * h) - (2 * i)`

D. `(2 * h) - (2 * i) - 1`

E. `(2 * h) - (2 * i) - 3`

Shade X on the answer sheet if none of the choices above is correct.

Solution: E.

This question tests if students can recognize a pattern and generalize it into an expression. Surprisingly, only slightly more than half of the students managed to solve this.

5. (3 points) Consider the function below:

```
void bus(long *i, long *j) {
  // Find the element in the middle of *i and *j in the array.
  long mid = ________________;
}
```

The function is called with

```
long a[100];
bus(&a[0], &a[99]);
```

The variable `mid` should be initialized to the middle element in the array (i.e., `a[49]`). How should `mid` be initialized?

A. `*((i + j) / 2)`

B. `(*i + *j) / 2`

C. `*(i + j) / 2`

D. `*(i + (j - i) / 2)`

E. `*i + (*j - *i) / 2`

Shade X on the answer sheet if none of the choices above is correct.

Solution: D.

This is a challenging question that assesses if students are aware of how pointer arithmetic operates. Not surprisingly, only 23% of students managed to answer this.

Most students picked A. However, `i` and `j` are not indices to an array, but pointers. The addition of two pointers is not a meaningful arithmetic operation. Pointer arithmetic only supports adding offsets to a pointer and calculating offsets between two pointers (which is what Option D does).

6. (3 points) Consider the function below:

```
void doh(long n) {
  long *a;
  for (long i = 0; i < n; i += 1) {
    a = malloc(2 * sizeof(long));
    if (a == NULL) {
      cs1010_println_string("Not enough memory");
    }
    a[0] = 1;
  }
  free(a);
}
```

Which of the following statements about the function above is true? Assume that `n` > 1.

(i) There is a memory leaks if `malloc` is successful.

(ii) There is an illegal access to memory if `malloc` is successful.

(iii) There is an illegal access to memory if `malloc` is not successful.

A. (i) only
B. (iii) only
C. (i) and (ii) only
D. (i) and (iii) only
E. (ii) and (iii) only

Shade X on the answer sheet if none of the choices above is correct.

Solution: D.

(i) is true. There will be a memory leak if `n > 1` since multiple allocation is made but there is only one `free`.

(iii) is true. If `a` is NULL, then accessing `a[0]` would crash the program.

For Questions 7 and 8, consider the code snippet below, which allocates a 2D array with two rows and `n` columns. We assume that `calloc` does not fail and `n` is a positive number.

```
size_t n = cs1010_read_size_t();

long *canvas[2];
canvas[0] = calloc(2 * n, sizeof(long));
canvas[1] = canvas[0] + n;
canvas[0][n] = 1;  // Line U
free(canvas[0]);
free(canvas[1]);   // Line V
```

7. (3 points) Which of the following statements about Line U is correct?

   A. It sets `canvas[1][0]` to 1

   B. It crashes the program with a heap overflow error.

   C. It crashes the program with a stack overflow error.

   D. It crashes the program with a segmentation fault.

   E. It causes a memory leak.

   Shade X on the answer sheet if none of the choices above is correct.

Solution: A.

The given code snippet allocates a continuous region of memory for two rows of integers. Each row contains `n` integers (indexed from `0` to `n - 1`. Line U attempted to update the element at index `n` of Row 0, i.e., `*(canvas[0] + n)`. This deferences `canvas[1][0]` instead.

About a third of students chose the correct answer.

8. (3 points) Which of the following statements about Line V is correct?

   A. It correctly ensures that there is no memory leak.

   B. It causes a memory leak.

   C. It crashes the program with a stack overflow error.

   D. It crashes the program since the memory pointed to by `canvas[1]` is not at the start of a memory region allocated with `calloc`.

   E. It crashes the program since the memory pointed to by `canvas[1]` is allocated on the stack.

   Shade X on the answer sheet if none of the choices above is correct.

Solution: D.

Since `canvas[1]` is only a pointer pointing to a region of memory allocated to `canvas[0]`, we are not supposed to free `canvas[1]`. Doing so would cause the program to crash.

Only 45% of the students chose the correct answer.

9. (3 points) Suppose that the running time for the function `find` is $O(n)$, what is the running time of the function below, as a function of n?

```
for (long i = 1; i < n; i += 1) {
  if (n % (i * i) == i) {
    return find(n);
  }
}
```

   A. $O(\log n)$
   B. $O(\sqrt{n})$
   C. $O(n)$
   D. $O(n \log n)$
   E. $O(n^2)$

Shade X on the answer sheet if none of the choices above is correct.

Solution: C.

In the worst case, the program loops $O(n)$ times and then calls `find(n)`. Since `find(n)` runs in $O(n)$ time, the running time of the given loop is $O(n) + O(n) = O(2n) = O(n)$.

31% of the students answered this correctly. A majority of students chose E $O(n^2)$. This is not correct since `find(n)` executes at most once due to the `return` keyword.

10. (3 points) The running time of a recursive algorithm can be characterized by the following recurrence relation:

$$T(n) = \begin{cases} T(n-1) + \log n & \text{if } n > 1 \\ 1, & \text{otherwise} \end{cases}$$

What is $T(n)$?

A. $O(\log n)$
B. $O(\log (n!))$
C. $O(n)$
D. $O(n \log n)$
E. $O(n^2)$

Shade X on the answer sheet if none of the choices above is correct.

Solution: B or D.

$$\begin{aligned} T(n) &= T(n-1) + \log n \\ &= T(n-2) + \log (n-1) + \log n \\ &= T(n-3) + \log (n-2) + \log (n-1) + \log n \\ &\vdots \\ &= T(n-k) + \log (n-k+1) + \ldots + \log (n-1) + \log n \end{aligned}$$

When $n - k = 1$,

$$\begin{aligned} T(n) &= T(1) + \log 2 + \ldots + \log (n-1) + \log n \\ &= 1 + \log (2 \times 3 \times \ldots \times (n-1) \times n) \\ &= 1 + \log(n!) \end{aligned}$$

So B is correct. This is the expected, straightforward, answer.

Option D was included by mistake. The reason why D is also accepted as the answer is more subtle. First, note that:

$$\begin{aligned} \log (n!) &= \log 1 + \log 2 + \ldots + \log (n-1) + \log n \\ &< \log n + \log n + \ldots + \log n + \log n \\ &= n \log n \end{aligned}$$

Furthermore, if we consider only the larger half of the terms on the right-hand side,

$$\begin{aligned} \log (n!) &= \log 1 + \log 2 + \ldots + \log (n-1) + \log n \\ &> \log (n/2) + \log (n/2+1) + \ldots + \log (n-1) + \log n \\ &> \log (n/2) + \log (n/2) + \ldots + \log (n/2) + \log (n/2) \\ &= n/2 \log (n/2) \end{aligned}$$

We have $\frac{n}{2} \log \frac{n}{2} < \log(n!) < n \log n$. Hence, $O(\log(n!)) = O(n \log n)$.

11. (3 points) The final marks of the students in CS1010 are floating point numbers represented as `double` between 0 and 100. The marks were initially sorted in increasing order. Due to regrade requests from students, a small number of students have their marks adjusted and thus, the marks are no longer correctly sorted. The teaching team wants to re-sort the marks again in increasing order. Which of the following algorithms is the most appropriate for re-sorting the final marks after the regrade request?

    A. Counting sort
    B. Insertion sort
    C. Bubble sort
    D. Selection sort
    E. Radix sort

    Shade X on the answer sheet if none of the choices above is correct.

    Solution: B. This is meant as a give-away question but, surprisingly, only slightly less than half of the students got it correct. Insertion sort is fast when it comes to sorting an almost-sorted list.

12. (3 points) Consider the implementation of the bubble sort and selection sort algorithms in class. Suppose we are given an array of size n that is inversely sorted, how many swaps are needed by each of the sorting algorithms?

	Bubble sort	Selection sort
A.	$O(n^2)$	$O(n^2)$
B.	$O(n^2)$	$O(n)$
C.	$O(n)$	$O(n^2)$
D.	$O(n)$	$O(n)$
E.	$O(1)$	$O(n)$

Shade X on the answer sheet if none of the choices above is correct.

Solution: B. This is another give-away question but only 25% of the students got it right.

For bubble sort, the first pass makes $n - 1$ swaps, the second pass makes $n - 2$ swaps, etc. To sort an inversely sorted array, $O(n^2)$ swaps are needed.

For selection sort, each pass only makes one swap (to swap the largest element to its position). So $O(n)$ swaps are needed.

Part II

Short Questions (44 points)

Answer all questions in the space provided on the answer sheet. Be succinct and write neatly.

13. (6 points) Consider the function below, which calculates the number of digits in a positive integer n.

```
long ndigits(long n) {
  if (n < 10) {
    return 1;
  }
  return 1 + ndigits(n / 10);
}
```

Through formulating a recurrence relation, express the running time of the function using the Big-O notation as a function of n. Show your workings.

Solution: The recurrence relation is the following:

- $T(n) = T(n/10) + 1$ (1 mark)
- $T(1) = 1$ (1 mark)

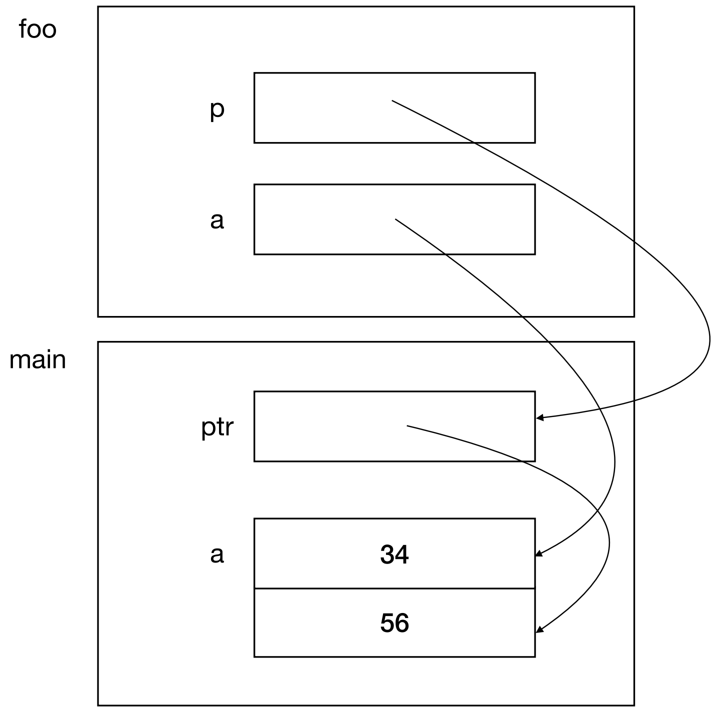
Expanding and observing the pattern, we have $T(n) = T(n/10^k) + k$ (2 marks).
When $\frac{n}{10^k} = 1, k = \log n$ (1 mark).
So $T(n) = T(1) + \log n = O(\log n)$ (1 mark).

14. (10 points) Consider the program below.

```
void foo(long a[], long **p) {
  *p = a;
  (*p) += 1;
  // Line N
}

int main()
{
  long a[2] = {34, 56};
  long *ptr;
  foo(a, &ptr);
  cs1010_println_long(*ptr);
}
```

(a) (8 points) Draw the content of the call stack when the execution reaches Line N using the notations similar to what has been used in CS1010. Label all your stack frames, variables, and values on the call stack. You may use arrows to denote pointers, instead of using the actual memory address.



Solution:
Marking scheme:

- 1 mark for the stack frames with correct function names
- 1 mark for drawing the stack vertically in the correct order
- 1 mark for `a[0]` in main
- 1 mark for `a[1]` in main
- 1 mark for `a` in `foo` correctly pointing to `a[0]`
- 1 mark for `p` in `foo` correctly pointing to `ptr`
- 2 marks for ptr in main correctly pointing to `a[1]`

(b) (2 points) What would the program print to the standard output?

Solution: 56

15. (22 points) Consider the following function `separate`, which takes an array of integer `a` with `n+1` (`n` > 0) elements as input.

```
void swap(long *a, long i, long j) {
  long temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}

long separate(long *a, long n) {
  long i = -1;
  long j = n + 1;
  long x = a[0];
  while (true) {
    // Line B
    do {
      i += 1;
    } while (a[i] < x);
    // Line C
    do {
      j -= 1;
    } while (a[j] > x);
    // Line D
    if (i < j) {
      swap(a, i, j);
      // Line E
    } else {
      // Line F
      return j;
    }
  }
}
```

Denote `a[i..j]` to be the set { `a[i]`, `a[i+1]`, ... `a[j]` }. If `j` < `i` then the set is empty. We extend the comparison operations `<`, `>`, `<=`, and `>=` to set of elements. For example, we write `a[i..j] < x` if every element in the set `a[i..j]` is less than `x`. We write `a[i..j] <= a[k..m]` if every element in the set `a[i..j]` is less than or equal to every element in `a[k..m]`.

Using this notation, the goal of the function is to rearrange the array elements such as the assertion `a[0..j] <= a[j+1..n]` holds at Line F. We want to show that the function is correct.

(a) (4 points) First, trace through the function if it is invoked with:

```
long a[5] = { 8, 9, 6, 5, 7 };
separate(a, 4);
```

Write the values of variable `i`, `j` and the content of the array `a` every time the execution reaches Line E or Line F.

Solution:

i	j	a
0	4	7 9 6 5 8
1	3	7 5 6 9 8
3	2	7 5 6 9 8

We give 4 marks if all three rows are correct, and at least 1 mark if the first row is correct. We give 3 marks if there is an additional row before or after the above. We deduct 1 mark

for each error in the `i` , `j` or `a` .
61% of students received full marks.

(b) (8 points) The invariant of the while loop is `a[0..i] <= x` and `a[j..n] >= x`. Assume that this assertion holds at Line B at the beginning of the while loop. By deriving the appropriate assertions at Line C and Line D, argue why the assertion is true at Line E.

Solution: `a[0..i] <= x` at Line B. `i` increases until `a[i] >= x`. So, at Line C, we assert `a[0..i-1] <= x` (1 mark) and `a[i] >= x` (1 mark) Similarly, `a[j..n] <= x` at Line B, `j` decreases until `a[j] <= x` . So, at Line D, we have `a[j+1..n] >= x` (1 mark) and `a[j] <= x` (1 mark).

At Line E, we swap `i` and `j` . We have `a[i] <= x` (1 mark) and `a[j] >= x` (1 mark) Combining with `a[0..i-1] <= x` and `a[j+1..n] >= x` , we have the assertion `a[0..i] <= x` and `a[j..n] >= x` (2 marks)

Many students incorrectly assert `a[0..i-1] < x` at Line C and `a[j+1..n] > x` at Line D. This is not correct. At line B, some of the elements in `a[0..i-1]` and `a[j+1..n]` could have the same value as `x` .

For such incorrect assertions, we still award the marks for `a[0..i] <= x` and `a[j..n] >= x` at Line E, since the conclusion still holds.

We do not give marks, however, if the two assertions `a[0..i-1] <= x` and `a[j+1..n] >= x` are missing. Without these, one cannot jump to the conclusion that the invariant holds at Line E.

About a quarter of students scored full marks here, with another quarter of students scoring 6 marks. Well done.

(c) (4 points) Argue why at Line F, we can assert that `i == j || i - 1 == j`.

Solution: We have `a[i-1] <= x` and `a[i] >= x` at Line C. If `a[i] == x` , we decrease `j` until `j == i` and stopped. Otherwise, `a[i] > x` and we decrease `j` until `j == i-1` since `a[i-1] <= x` .
You need to make a clear and concise argument, similar to the above to earn 4 marks.
Common mistakes:

- Arguing that, since `i > j` , then `i - 1 == j` . This is wrong – why can't `i - 2 == j` ?
- Arguing that, since `a[i - 1] == a[j]` , then `i - 1 == j` . This is wrong – if `a[2] == a[3]` , then 2 == 3?
- Arguing that, since `a[i] >= x` and `a[j] <= x` , then `i == j` . This is wrong – consider `a[3] = {0, 0, 0}` and `x` is 0. We can have `i == 2` and `j == 0` .
- Arguing that, if `i == j || i - 1 == j` , then the invariant holds. This does not answer the question (it answers Part (d) though).
- In fact, any argument that starts with "if `i == j` " or "if `i - i == j` " is incorrect.
- Any argument that is based on the invariant is incorrect. So far, we only established that the invariant holds at Line E, not Line F.

- Arguing that, since `i` increments by at most 1 and `j` decrements by at most 1, when they meet, the difference between `i` and `j` is at most 1. This is wrong. `i` increments by 1 in a `do-while` loop so it increments by *at least* once. Same for `j` – `j` decrements by at least 1. You need to argue why `j` decrements by at most 1 before the loop exits and reaches Line F.
- Argument that is based on the number of elements n (e.g., whether n is odd or even).

(d) (4 points) Use the invariant of the loop to argue why the assertion `a[0..j] <= a[j+1..n]` holds at Line F.

Solution: At Line D, `a[0..i-1] <= x` and `x <= a[j+1..n]`. At Line F, either `i == j` or `i - 1 == j`.

If `i - 1 == j`, therefore `a[0..j] <= x <= a[j+1..n]` (1 mark) If `i == j`, `a[i] == a[j] == x`, therefore, `a[0..j] <= a[j+1..n]` (3 marks)

The most common mistake is to use the invariant of the loop directly `a[0..i] <= x` and `x <= a[j..n]` without considering the do-while loops. This assertion is true at Line B and Line E, but not Line F (since `a[i]` and `a[j]` are not swapped). No marks are awarded if students start with an incorrect assertion.

(e) (2 points) Express the running time of the function `separate` using big-O notation as a function of n. Explain your answer.

Solution: $O(n)$. It scans through the array once from both sides.

No marks are given if the running time is correct but no/wrong explanation is given.

16. (6 points) The following function generates all possible binary strings (i.e., strings consisting of `'0'` and `'1'` only) of a given length `n`.

```c
void generate(char str[], long n, long k) {
  if (k == n) {
    cs1010_println_string(str);
    return;
  }

  str[k] = '0';             // Line Q
  generate(str, n, k+1);    // Line R
  str[k] = '1';             // Line S
  generate(str, n, k+1);    // Line T
}
```

For example, when called with

```c
char str[4] = { 0 };
generate(str, 3, 0);
```

It generates the following output:

```
000
001
010
011
100
101
110
111
```

The binary output is generated in increasing order of their decimal (base-10) values. In this question, we are interested in generating all possible binary strings such that two consecutive strings differ by at most one bit (one binary digit).

For example, we would like to generate the following when `n` is 2 and 3 respectively.

When `n` is 2

```
00
01
11
10
```

When `n` is 3

```
000
001
011
010
110
111
101
100
```

Assuming that the string `str` has been initialized to a string with `n` `'0'`s, replace the four statements labeled Q, R, S, and T in the function `generate` so that it generates the binary string in the desired order. Use only recursion without loops.

Solution:

```
  generate(str, n, k+1);
  str[k] = (str[k] == '0') ? '1' : '0';
  generate(str, n, k+1);
```

The binary sequence we wish to generate here is called the *gray code* and is often used in hardware due to its guaranteed single-bit switch when counting from one number to the next.

The original recursive calls to `generate` a binary sequence always set/reset the k-th bit to 0, followed by 1, before each recursive call. You can see this effect from the last bit of the sequence, which has the pattern 0101. For gray code, observe that the last bit in the examples above has the pattern 0110. Thus, we `retain` the k-bit before the first recursive call, and *flip* the k-th bit before the second call. This is the only change needed.

There are no partial marks for this question. Either 0 or 6 marks are awarded for the answer. If the answer is correct but there are careless mistakes, we may -1 for each mistake.

11 students received 5-6 marks for this question.

END OF PAPER