

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
FINAL ASSESSMENT FOR
Semester 1 AY2021/2022

CS1010 Programming Methodology

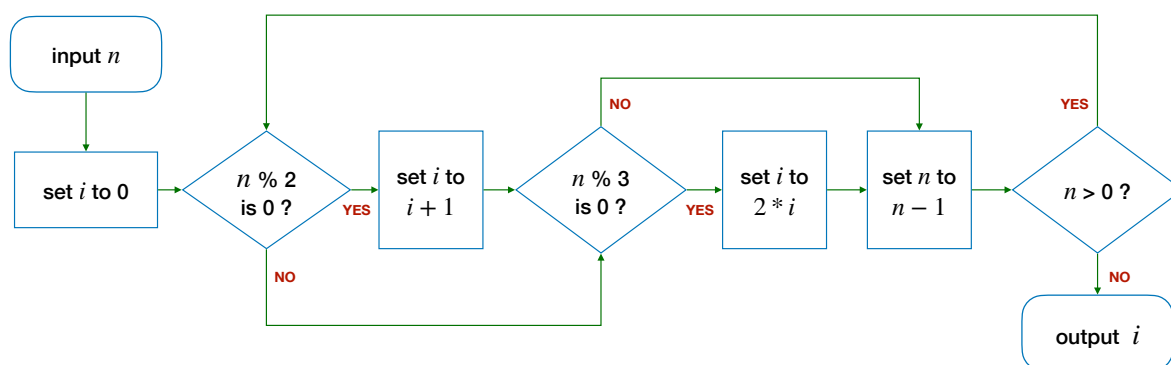
November 2021

Time Allowed 120 Minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains 19 questions and comprises 26 printed pages, including this page.
2. Write all your answers in the answer sheet provided.
3. The total marks for this assessment is 60. Answer **ALL** questions.
4. This is an **OPEN BOOK** assessment.
5. You can assume that in all the code given, no overflow nor underflow will occur during execution. In addition, you can assume that all given inputs are valid and fits within the specified variable type.
6. State any additional assumption that you make.
7. Please write your student number only. Do not write your name.

1. (3 points) Consider the flowchart below:



Which of the following function implements the flowchart above correctly? (Select all correct options)

A. `long foo(long n) {
 long i = 0;
 do {
 if (n % 2 == 0) {
 i += 1;
 } else if (n % 3 == 0) {
 i *= 2;
 }
 n -= 1;
 } while (n > 0);
 return i;
 }`

B. `long foo(long n) {
 long i = 0;
 do {
 if (n % 2 == 0) {
 i += 1;
 }
 if (n % 3 == 0) {
 i *= 2;
 }
 n -= 1;
 } while (n > 0);
 return i;
 }`

C. `long foo(long n) {
 long i = 0;
 do {
 if (n % 2 == 0) {
 i += 1;
 }
 if (n % 3 == 0) {
 i *= 2;
 }
 n -= 1;
 } while (n > 0);
 return i;
 }`

- ```

 }
D. long foo(long n) {
 long i = 0;
 while (n > 0) {
 if (n % 2 == 0) {
 i += 1;
 } else if (n % 3 == 0) {
 i *= 2;
 }
 n -= 1;
 }
 return i;
}

E. long foo(long n) {
 long i = 0;
 while (n > 0) {
 if (n % 2 == 0) {
 i += 1;
 }
 if (n % 3 == 0) {
 i *= 2;
 }
 n -= 1;
 }
 return i;
}

F. long foo(long n) {
 long i = 0;
 while (n > 0) {
 if (n % 2 == 0) {
 i += 1;
 }
 if (n % 3 == 0) {
 i *= 2;
 }
 n -= 1;
 }
 return i;
}

G. None of the above

```

**Solution: C.**

There are two parts that we test here: (a) whether students know how to convert a loop from a flowchart into code, and (b) whether students know how to convert a branch from a flowchart into code. The loop here is a do-while loop, since it runs the body at least once. The branch is actually two separate if statements, since it checks both conditions, regardless of whether the first condition is true or false.

Note that Option A only checks the second condition if the first condition is false. Option B only checks the second condition if the first condition is true.

161 students (about 66%) of students got this correct.

2. (3 points) Suppose `m` and `n` are `long` variables holding positive integers. Which of the following expressions return `true` if and only if `m` and `n` have the same number of digits (excluding leading zeros)?

(Select all correct options)

- A. `m / n == 0`
- B. `m / n < 10`
- C. `m / n <= 1`
- D. `(m / n <= 1) || (n / m <= 1)`
- E. `(m / n < 10) && (n / m < 10)`
- F. None of the above.

**Solution:** F.

87 students (36%) got this correct.

3. (3 points) Consider the code snippet below.

```
bool in_range = false;
if (x > 10) {
 if (y < 20) {
 in_range = true;
 }
}
if (!in_range) {
 // Line A
}
```

What assertion must be true at Line A?

(Select all correct options)

- A. `(x > 10) && (y < 20)`
- B. `(x > 10) || (y < 20)`
- C. `(x <= 10) || (y >= 20)`
- D. `(x <= 10) && (y >= 20)`
- E. None of the above.

**Solution:** At Line A, we know that `in_range` is false. To set `in_range` to `true`, the only way is when `x > 10` and `y < 20`. Since `in_range` is false, we apply de Morgan's law and get `x <= 10 || y >= 20`. (Option C).

200 students got this correct.

4. (3 points) Consider the function below:

```
void foo(long x, long y) {
 while (x * y > 0 && x != y) {
 x += 1;
 y -= 1;
 if (x > y) {
 return;
 }
 }
 // Line B
}
```

Which of the following is/are correct assertions at Line B?

(Select all correct options)

- A. `x * y > 0 && x != y`
- B. `x * y > 0 || x != y`
- C. `x * y <= 0 && x == y`
- D. `x * y <= 0 || x == y`
- E. `x < y`
- F. `x <= y`

**Solution:** If we can reach Line B, then the loop condition must be false, so applying de Morgan's law, we have `x * y <= 0 || x == y` (Option D).

Note that due to the `return` statement within the loop, some students think that Option F is correct. If `x > y` and `x * y <= 0`, however, then the loop is not entered at all and Option F is not a correct assertion at Line B.

88 students got this correct. (36%)

5. (3 points) Consider the recursive function above.

```
long recurse(long n) {
 if (n == 1) {
 return 1;
 }
 return 2 + recurse(recurse(n - 1));
}
```

What is the output of `recurse(2)` and `recurse(3)`?

Write “no output” if you think the function would run forever (until stack overflow occurs).

**Solution:** `recurse(2)` calls `2 + recurse(recurse(1))`, which gives `2 + recurse(1)`, i.e., 3.

`recurse(3)` calls `2 + recurse(2)`, which gives `2 + recurse(3)`, which leads to an infinite loop.

So answer is 3 and no output.

95% of students (229) got the right answer.

6. (3 points) Each of the following five functions aims to sum up the values in a given array.

Which of the functions could potentially lead to memory errors or returning of incorrect results when an empty array (len is 0) is given?

(Select all correct options)

- A. `long tata(long len, long a[len]) {  
     long sum = 0;  
     for (long i = 0; i <= len - 1; i += 1) {  
         sum += a[i];  
     }  
     return sum;  
}`
- B. `long tete(long len, long a[len]) {  
     long sum = 0;  
     for (long i = len - 1; i >= 0; i -= 1) {  
         sum += a[i];  
     }  
     return sum;  
}`
- C. `long titi(long len, long a[len]) {  
     long sum = a[0];  
     long i = 1;  
     do {  
         sum += a[i];  
         i += 1;  
     } while (i <= len - 1);  
     return sum;  
}`
- D. `long toto(long len, long a[len]) {  
     long sum = 0;  
     long i = 0;  
     do {  
         sum += a[i];  
         i += 1;  
     } while (i <= len - 1);  
     return sum;  
}`
- E. `long tutu(long len, long a[len]) {  
     long sum = 0;  
     long i = 0;  
     while (i <= len - 1) {  
         sum += a[i];  
         i += 1;  
     }  
     return sum;  
}`

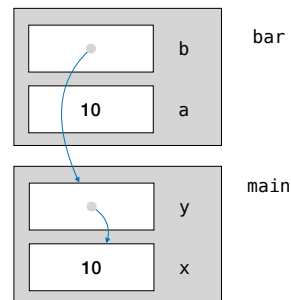
**Solution:** When an array has zero element (e.g., `long *a` without initialization), accessing `a[0]` would lead to an error. A, B, and E all checks `i` before entering the loop, so avoided access to `a[0]`.

C blatantly set `sum` to `a[0]`. D uses do-while loop so accesses `a[0]` before checking the value of `i`.



The correct options are C and D.  
180, or 74%, of students got the right answer here.

7. (3 points) Consider the diagram below, which depicts the stack frames of two functions, main and bar. bar is called from main. The arrows depict pointer variables and where they are pointing to.



Which of the following program would lead to the stack frames above at Line C?

Note that some options below might give compilation errors.

(Select all correct options)

A. `void bar(long a, long **b) {`  
     *// Line C*  
`}`

`int main() {`  
     `long x = 10;`  
     `long *y = &x;`  
     `bar(x, &y);`  
`}`

B. `void bar(long a, long **b) {`  
     *// Line C*  
`}`

`int main() {`  
     `long x = 10;`  
     `long *y = &x;`  
     `bar(*y, &y);`  
`}`

C. `void bar(long a, long **b) {`  
     *// Line C*  
`}`

`int main() {`  
     `long x = 10;`  
     `long *y = &x;`  
     `bar(x, &(&x));`  
`}`

D. `void bar(long a, long *b) {`  
     *// Line C*  
`}`

`int main() {`  
     `long x = 10;`  
     `long *y = &x;`

```
 bar(*x, y);
 }
E. void bar(long a, long *b) {
 // Line C
}

int main() {
 long x = 10;
 long *y = &x;
 bar(x, y);
}
```

F. None of the above.

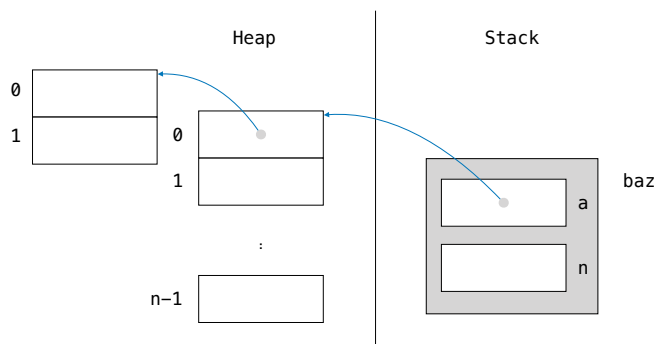
**Solution:** `y` is a pointer to long. `b` points to `y` so `b` is a pointer to pointer to `long`. This excludes D and E as the correct answer.

`&(&x)` is not a valid syntax. So C is out.

A and B are correct. They both pass the address of `y` and the content of `x` into `bar`.

Only 100 (41.3%) students got this correct (surprisingly).

8. (3 points) Consider the diagram below, which depicts the stack and heap of a program. Only the stack frame of the function `baz` is shown. Irrelevant details (such as value of `n`) are omitted. Assume that `malloc` always completes successfully.



Which of the following implementation of `baz` correctly allocates memory and would lead to the stack and heap as depicted in the diagram at Line D of the function?

Note that some options below might give compilation errors.

(Select all correct options)

- A. `void baz(size_t n) {`  
`long a[n];`  
`long b[2];`  
`a[0] = b;`  
`// Line D`  
`}`
- B. `void baz(size_t n) {`  
`long a[n];`  
`a[0] = malloc(2 * sizeof(long));`  
`// Line D`  
`}`
- C. `void baz(size_t n) {`  
`long *a = malloc(n * sizeof(long *));`  
`a[0] = malloc(sizeof(long)*2);`  
`// Line D`  
`}`
- D. `void baz(size_t n) {`  
`long a[n][2];`  
`// Line D`  
`}`
- E. `void baz(size_t n) {`  
`long **a = malloc(n * sizeof(long *));`  
`a[0] = malloc(2 * sizeof(long));`  
`// Line D`  
`}`
- F. None of the above.

**Solution:** VLA is allocated on the stack. This excludes A, B, D. We are left with C and E. C declares `a` to be `long *`. E declares `a` as `long **`. Only E is correct.  
190 (78.5%) students got 3 marks for this.

9. (3 points) Consider the code snippet below.

```
void fred(long n) {
 for (long i = 0; i < n * n; i += n) {
 cs1010_println_long(i);
 }
}
```

How many times will `i` be printed? Express your answer in big-O notation in terms of the input parameter `n`.

- A.  $O(\log n)$
- B.  $O(\log^2 n)$
- C.  $O(n)$
- D.  $O(n^2)$
- E.  $O(n^3)$

**Solution:** C.

This is an easy one. 94%, or 228, of students got 3 marks.

10. (3 points) Consider the code snippet below.

```
void corge(long n) {
 for (long j = 1; j < n; j *= 2) {
 cs1010_println_long(j); // Line E
 }
}

void qux(long n)
{
 long i = 1;
 while (i < n) {
 i *= 2;
 corge(n);
 }
}
```

How many times would Line E be called if `qux` is invoked? Express your answer using the big-O notation.

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

**Solution:** `corge` is called  $O(\log n)$  times. Each call to `corge` invoke Line E  $O(\log n)$  times. So the total number of times is  $O(\log n \times \log n) = O(\log^2 n)$ .

This is a bit harder, but still, 218 students (90%) got the right answer! Good job.

11. (3 points) Consider the following function:

```
long thud(long n) {
 if (n == 1) {
 return 0;
 }
 long sum = 0;
 for (long i = 0; i < n; i += 1) {
 sum += thud(n / 2);
 }
 return sum;
}
```

Let the running time of the function be  $T(n)$ .

Which of the following is the correct recurrence relation expression for  $T(n)$  when  $n > 1$ ?

- A.  $T(n) = 2T(n/2) + 1$
- B.  $T(n) = T(n/2) + 1$
- C.  $T(n) = T(n/2) + n$
- D.  $T(n) = nT(n/2) + 1$
- E.  $T(n) = nT(n/2) + n$
- F. None of the above.

**Solution:** D and E are both correct. 190 students chose one of the correct answers above (78.5%)



12. (3 points) Consider the following recurrence relation:

$$T(n) = T(n/4) + \sqrt{n}$$

and

$$T(1) = 1$$

Express  $T(n)$  in big O-notation.

- A.  $O(n)$
- B.  $O(\log n)$
- C.  $O(4^n)$
- D.  $O(\sqrt{n})$
- E.  $O(\sqrt{n} \log n)$
- F. None of the above.

**Solution:**

$$\begin{aligned} T(n) &= T\left(\frac{n}{4}\right) + \sqrt{n} \\ &= T\left(\frac{n}{16}\right) + \frac{\sqrt{n}}{2} + \sqrt{n} \\ &= T\left(\frac{n}{64}\right) + \frac{\sqrt{n}}{4} + \frac{\sqrt{n}}{2} + \sqrt{n} = .. \end{aligned}$$

So

$$T(n) = T\left(\frac{n}{4^k}\right) + \sum_{i=0}^k \frac{\sqrt{n}}{2^i}$$

When  $k$  is  $\log_4 n$ ,  $T(\frac{n}{4^k})$  is  $O(1)$ , and  $\sum_{i=0}^k \frac{\sqrt{n}}{2^i} < 2\sqrt{n} = O(\sqrt{n})$ .

So answer is D.

90 students (37%) got it right.

13. (3 points) Which of the following program would lead to illegal access of memory?

(Select all correct options)

- A. `int main() {  
    char *s;  
    s[4] = 'x';  
}`
- B. `int main() {  
    char *s = "hello";  
    s[4] = 'x';  
}`
- C. `int main() {  
    char s[5];  
    s[4] = 'x';  
}`
- D. `char *get() {  
    char t[6] = "hello";  
    return t;  
}`  
  
`int main() {  
    char *s = get();  
    s[4] = 'x';  
}`
- E. `char *get() {  
    char *t = malloc(sizeof(char));  
    return t;  
}`  
  
`int main() {  
    char *s = get();  
    s[4] = 'x';  
}`
- F. None of the above

**Solution:** A. Yes. `s` is not initialized.

B. Yes. `s` is pointing to a read-only region of memory.

C. No. `s` is allocated on the stack.

D. Yes. `s` is pointing to a memory that belongs to a stack frame that has been deallocated.

E. Yes. `s` is pointing to a memory that belongs to a heap (which is OK) but consists of only one character. Modifying `s[0]` would be OK, but `s[4]` is not.

92 students chose the right options. (38%)

14. (3 points) Consider the program below:

```
typedef struct box {
 int x;
} box;

void modify(box *b) {
 b->x = 2;
}

void modify_again(box b) {
 b.x = 3;
}

struct box modify_some_more(box b) {
 b.x = 4;
 return b;
}

int main()
{
 box a;
 a.x = 1;
 modify(&a);
 // Line F
 modify_again(a);
 // Line G
 a = modify_some_more(a);
 // Line H
}
```

What are the values of `a.x` at Lines F to H?

**Solution:** 2, 2, and 4.

`modify` takes in the parameter by reference, so it successfully modifies `a.x` to 2.

`modify_again` takes in the parameter by value, so it is modifying a copy of `a`. After it exits, `a.x` remains 2.

`modify_some_more` takes in a copy of `a`, but it also returns the copy, which is copied into `Ta`. So `a.x` is modified into 4.

I decided to give partial marks for this – 1 mark for each correct answer. 204 students received full 3 marks (84%)

15. (1 point) Consider the function below:

```
long mystery(size_t len, long *a) {
 size_t l = 0;
 size_t r = len - 1;
 while (l != r) {
 if (a[l] < a[r]) {
 r -= 1;
 } else {
 l += 1;
 }
 }
 return a[l];
}
```

What would the function return if the input array `a` is `{50, 40, 20, 30}` and `len` is 4?

**Solution:** 20. Almost everyone 238 (98%) got this one right.

16. (4 points) Consider the snippet taken from the function in the previous question:

```
if (a[l] < a[r]) {
 r -= 1;
 // Line U
} else {
 l += 1;
 // Line V
}
```

What can you assert at Lines U and V based only on the snippet above?

**Solution:** At Line U, `a[l] < a[r + 1]`; At Line V, `a[l - 1] >= a[r]`. 187 students got full 4 marks.

17. (5 points) Let's denote the set of consecutive elements `a[i]`, `a[i+1]`, `a[i+2]`, ... `a[j]` as `a[i..j]`.

We extend the comparison operator `<`, `>`, `<=`, `>=` to denote that an element is less than, greater than, less than or equal to, greater than or equal to, every element in a set respectively.

For example, if `a[3] < a[0]` and `a[3] < a[1]`, we can write `a[3] < a[0..1]` using our new notation.

Furthermore, if `j < i`, then `a[i..j]` is an empty set.

```
long mystery(size_t len, long *a) {
 size_t l = 0;
 size_t r = len - 1;
 // Line W
 while (l != r) {
 // Line X
 if (a[l] < a[r]) {
 r -= 1;
 } else {
```

```

 l += 1;
}
// Line Y
}
// Line Z
return a[l];
}

```

Consider the following assertion:

`a[l] < a[r+1 .. len-1] && a[r] <= a[0 .. l-1]`

Is the assertion above an invariant for the loop in the function above? If yes, explain why by referring to the assertion at Lines W, X, Y, and Z. If not, modify it into the correct invariant and explain your answer.

**Solution:** The assertion is NOT an invariant for the loop.

You can give a counter example. If the input is {40, 20, 50, 30}, after the two iterations, `l` is 1 and `r` is 2. The assertion

`a[l] < a[3 .. 3] && a[2] <= a[0 .. 0]`

is false (since `a[2]` is 50 and `a[0 .. 0]` contains 40) at Line Y.

The correct invariant should be that “the minimum element of the array is in `a[l .. r]`”

This is true at Line W since `a[l..r]` is the whole array.

Assuming it is true at Line X, at Line Y, we know either `a[l] < a[r+1]` and the min is in `a[l..r+1]` or `a[l-1] >= a[r]` and the min is in `a[l-1..r]`.

For the first case, since `a[l] < a[r+1]`, we know `a[r+1]` cannot be the min, so the min is in `a[l..r]`.

For the second case, since `a[l-1] >= a[r]`, we again can safely remove `a[l-1]` from consideration and say the min is in `a[l..r]`. So the invariant holds at Line Y.

At Line Z, we exit the loop after Line X without modifying `l` or `r` so it is true.

Many students stumbled at this. Many have thought the the assertion is a valid invariant or think that it is not a correct invariant but modified it into something that is still wrong.

The marking scheme for this question is:

- Identify that the given invariant is wrong and give the right invariant (2 marks)
- Explain why the given invariant is wrong (1 mark)
- Explain why the modified invariant is right (2 marks)

Many students who answer “No” gave the following as an alternative invariant:

`a[l] < a[r+1 .. len-1] || a[r] <= a[0 .. l-1]`

This is a valid invariant. Some of you who wrote this, however, failed to argue rigorously why it is an invariant. Particular, assuming that it is true at Line X, why is it true at Line Y? You need to show how this invariant changes as `r` decreases or `l` increases, and combine it with the answers of Q16 to derive that the invariant is unchanged at Line Y. You don’t get full marks if there are jump in your logic in the derivation.

An alternative to the invariant I mentioned above is:

```
min(a[r], a[l]) < a[r+1 .. len-1] && min(a[r], a[l]) <= a[0 .. l-1]
```

This is probably the minimal correct change from the invariant given in the question.

18. (2 points) Using the invariant from the previous question, explain why the function `mystery` always returns the minimum element from the given array.

**Solution:** When we exit the loop, we have `l == r`. So there is only one element in the set `a[l..r]`. According to the invariant at Line Z, this must be the minimum element.

Given how badly students do in Q15-18, I am feeling generous and I give full 2 marks to students who correctly derive that the function returns the minimum element based on the wrong invariant

```
a[l] < a[r+1 .. len-1] && a[r] <= a[0 .. l-1]
```

As long as students showed that they are aware that `l == r` and they substitute this into the incorrect invariant above to show `a[l]` and `a[l]` is less than every other elements, then they get 2 marks.

Students who gave the invariant as

```
a[l] < a[r+1 .. len-1] || a[r] <= a[0 .. l-1]
```

however, mostly get 0 for Q18, since this invariant, even though is correct, is too weak and not useful enough to help us derive the desired behavior of the function. Many students wrote something along the line of the following:

“since `l` and `r` are the same, we have `a[l] < a[l + 1 .. len-1]` AND `a[l] <= a[0..l-1]` and so `a[l]` is smaller than every other element in the array.”

The `||` OR condition magically becomes an AND condition! You can’t just anyhow change the logical operator just to prove what you are asked to prove.

19. (6 points) In this question, we want to write a function that print out all possible string as the result of interleaving characters from two given strings. The relative order of the characters from the same string must be preserved in the interleaved string. For instance, if the input strings are `ab` and `123`, then our function should print:

```
ab123
a1b23
a12b3
a123b
1ab23
1a2b3
1a23b
12ab3
12a3b
123ab
```

Part of the code of `interleave` has been given:

```
#include "cs1010.h"

void interleave(char *s, char *t, char *out, size_t last) {
 if (s[0] == '\0') {
 out[last] = '\0';
 cs1010_print_string(out);
 cs1010_println_string(t);
 return;
 }

 if (t[0] == '\0') {
 out[last] = '\0';
 cs1010_print_string(out);
 cs1010_println_string(s);
 return;
 }

 // COMPLETE THE CODE HERE

}
```

The function `interleave` is a recursive function that takes in two strings `s` and `t`. The string `out` is a string that stores an interleaving of characters from `s` and `t`. You may assume that `out` has been properly allocated. The 4th parameter `last` stores the total number of characters from either `s` or `t` (or both) that has been interleaved and stored in `out`.

The base case has been written for you. Here, we reach the base case if either `s` or `t` is an empty string. Since we are out of characters to interleave, we simply print what we constructed so far (out) followed by the remaining characters from the other string.

The first call to the function `interleave` looks like this:

```
interleave(s, t, out, 0);
```

Complete the implementation of the function `interleave`. There should be at most four lines of code. Write `;` in the blank below if your solution has less than four lines and you want to leave the blanks empty. You may print out the interleaving strings in any order (not necessary the same order as the example above).

Hint: if `s` is a string, then `s+1` is the rest of `s` without the first character.

**Solution:**

```
out[last] = s[0];
interleave(s + 1, t, out, last + 1);
out[last] = t[0];
interleave(s, t + 1, out, last + 1);
```

Students did better than I expected here. Bravo!! 73 students scored full marks. 37 students get 5 marks.

Partial marks are given as long as students have the recursion structure: the answer contains two `interleave` calls but may have other bugs (such as initializing `out` incorrectly, mixed up `+1` with other increments, etc) . Students who included loops, branching, 1/3/4 `interleave` calls, are given 0 marks.



END OF PAPER

This page is intentionally left blank.