TIC1001—Introduction to Computing and Programming
National University of Singapore
School of Continuing and Lifelong Education

# Midterm Test

3 October 2019 **Time allowed:** 1 hour 30 mins

| **Student No:** | **S** | **O** | **L** | **U** | **T** | **I** | **O** | **N** | **S** |
|---|---|---|---|---|---|---|---|---|---|

**Instructions (please read carefully):**

1. Write down your **student number** on the question paper. DO NOT WRITE YOUR NAME ON THE ANSWER SET!
2. This is **an open-book test**. While you are allowed to bring any physical materials and notes, with the exception of a **non-programmable calculator**, no other electronic devices such as tablets and laptops are allowed.
3. This paper comprises **THREE (3) questions** and **EIGHT (8) pages**. The time allowed for solving this test is **1 hour 30 mins**.
4. The maximum score of this test is **40 marks**. The marks for each question is given in square brackets beside the question number.
5. Note that the marks and order of the questions do not necessarily correspond to the level of difficulty of the question.
6. All questions must be answered correctly for the maximum score to be attained.
7. The pages marked "scratch paper" in the question set may be used as scratch paper.
8. You are allowed to un-staple the sheets while you solve the questions. Please make sure you staple them back in the right order at the end of the test.
9. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red colour, please).

# ALL THE BEST!

**EXAMINER'S USE ONLY**

| Question | Marks |
|---|---|
| Q1 | / 12 |
| Q2 | / 20 |
| Q3 | / 8 |
| **Total** | / 40 |

This page is intentionally left blank.

It may be used as scratch paper.

# Question 1: C/C++ Expressions [12 marks]

There are several parts to this question which are to be answered independently and separately. Each part consists of a fragment of C/C++ code. Write the exact output produced by the code in **the answer box**. If an error occurs, or it enters an infinite loop, state and explain why.

You may show workings **outside the answer box** in the space beside the code. Partial marks may be awarded for workings if the final answer is wrong.

Assume that all appropriate preprocessor directives e.g., `#include <iostream>`, etc. have already been defined.

**A.** [4 marks]

```
double f(int x, double y) {
    x = (int)(x / y);
    return x * y;
}

void main() {
    int x = 10;
    double y = 2.7;
    cout << f(f(x, y), y) << endl;
}
```

```
5.4
```

**B.** [4 marks]

```
int a = 42;
if (a % 7)  // Note the lack of braces
    cout << "Best ";
    a /= 7;
if (a < 7)
    cout << "of ";
if (a)
    cout << "both ";
else
    cout << "worlds";
```

Boolean:Anything that is not 0 is true

Only run the first line of If,
so will still run a/=7

```
of both
```

**C.** [4 marks]

```
int i = 0, j = 0;
for (int i = 0; i < 10; i += 2) {
    printf("%d,", i);
    if (i % 3) {
        continue;
    } else if (i < j) {
        break;
    } else {
        j += 1;
    }
}
printf("%d,%d\n", i, j);
```

```
0,2,4,6,8,0,2
```

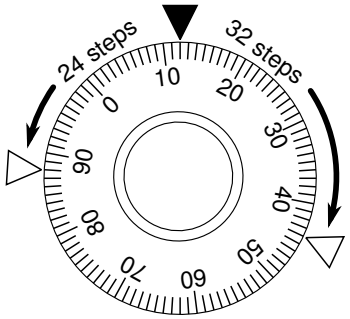
# Question 2: Combination Locks [20 marks]

**Instruction: In view of function abstraction, you must reuse function defined in earlier parts whenever possible in other parts of the question.**

A combination lock is a type of locking device in which a sequence of symbols, usually numbers, is used to open the lock. The sequence may be entered using a single or multiple rotating dials on the lock.

**A. [Warm up]** A single dial combination lock has a dial with $n$ positions, numbered 0 to $n-1$, arranged along a circular path. Because the circular arrangement wraps around both ends, there are two distances between any two numbers: one counting anti-clockwise, and the other counting clockwise.

For example, the illustration on the right shows a lock with $n = 100$ (numbers 0–99). The smaller distance between the numbers 12 and 88 is 24 steps going anti-clockwise, and between 12 and 44 is 32 steps going clockwise. In both cases, going the opposite direction would result in a longer distance.




Implement a function `int distance(int n, int from, int to)` which takes in three non-negative integers, where `n` is the number of positions on the dial. The function returns the smaller distance between the positions `from` and `to`.

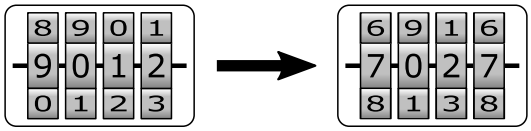
You may assume that the inputs will always be valid, i.e., the positions given will be valid positions on the dial.

Note: The standard functions `abs`, `min` and `max` are available for use. [5 marks]

```
int distance(int n, int from, int to) {
    int d1 = abs(from - to);
    return min(n-d1, d1);
}
```

**Multi-dial Combination Locks** have several dials arranged in a line. Each dial has the digits 0–9 arranged in a circular manner with both ends wrapping around. A marking line indicates the digits that makes up the current combination.

To change the combination, each digit dial has to be moved independently up or down one step at a time. For example, using a 4-dial lock as illustrated below, to get from combination 9012 to 7027, the leftmost dial has to be moved 2 steps down, the dial second from the right has to be moved 1 step up, and the rightmost dial has to be moved 5 steps in either direction. The dial second from the left need not be moved. In total, it took $2+1+5=8$ steps to change the combination.




**B.** Given two lock combinations, we want to obtain the minimum number of steps needed to change the combination from one to another.

Implement the function `int num_steps(int c1, int c2)` which takes two non-negative integers that represents the combinations of a multi-dial lock, and returns the minimum number of steps required to change from one to another.

Note that leading zeros are omitted in the integer representation of the combination. E.g., combination `000001` will just be the integer `1`.

Though you may assume that the inputs will not overflow the `int` value range, you should not assume anything about the number of dials on the lock. [5 marks]

```
int num_steps(int c1, int c2) {
   int steps = 0;
   while (c1 > 0 or c2 > 0) {
       steps += distance(10, c1%10, c2%10);
       c1 /= 10;
       c2 /= 10;
   }
   return steps;
}
```

**C.** The function `inc` takes in two integer inputs: `combi` and `pos`, where `combi` represents the current combination of a multi-dial lock. The function modified combination by incrementing the dial indicated by `pos` by one step. The right-most dial (the least-significant digit) is position `1`.

For example, suppose `combi` $\rightarrow$ `12345` and `pos` $\rightarrow$ `2`, then the call function `inc` with these two arguments will result in `combi` $\rightarrow$ `12355`. Further calling `inc` with `combi` and `pos` $\rightarrow$ `5` will result in `combi` $\rightarrow$ `22355`.

Provide an implementation for the function `inc`. You are to decide on the appropriate return type and parameters for the function. You may assume the `pow(b, e)` function from the math library is available for use. [5 marks]

```
void inc(int &combi, int pos) {
    pos = pow(10, pos-1);
    int digit = (combi / pos) % 10;
    if (digit == 9) {
        combi -= 9 * pos;
    } else {
        combi += pos;
    }
}
```

**D.** Implement the function `int advance(int combi, int steps)` that takes a combination as an input, along with the number of steps to increment the dials, and returns a new combination. The position of each digit in `steps` corresponds to the dial in the combination.

For example, if the `combi` $\rightarrow$ 9012 and `steps` $\rightarrow$ 8015, then the function `advance` will return a combination with the dials advanced, from left to right, by 8, 0, 1 and 5 steps, respectively. Thus, it will return 7027.

*Hint: You may wish to define a helper function to increment a dial by n positions.* [5 marks]

```
int advance(int combi, int steps) {
    int new_combi = combi;
    for (int pos = 1; combi > 0; pos++) {
        inc_n(new_combi, pos, n);  // helper function
        clicks /= 10;
        combi /= 10;
    }
    return new_combi;
}

void inc_n(int &combi, int pos, int n) {
    for (int i = 0; i < clicks % 10; i++)
        inc(new_combi, pos);
}
```

# Question 3: Computing Topics [8 marks]

## A. Computer Organisation and Compilation

Indicate whether each of the statements are True or False by **circling** the respective letter. Each answer carries equal weightage. [2 marks]

i. In a von Neumann architecture, program code must be moved from memory into the CPU using the BUS. [ (T) / F ]

ii. In a load-store architecture, operands are loaded directly from the memory into the ALU and the result is directly stored back into memory. [ T / (F) ]

iii. All high-level programming languages has to be compiled into the native ISA before it can be executed. [ T / (F) ]

iv. The linking process allows code from other sources to be incorporated into a program. [ (T) / F ]

## B. Data Representation

Answer the following Multiple Choice Questions by **circling** the appropriate option. Each question carries equal weightage. [2 marks]

i. The word size of a computer refers to the size of its registers and pointers. What is the approximate maximum addressable memory size on a computer with 16-bit word size?

*Note: kilo = 1,000; mega = 1,000 kilo; giga = 1,000 mega*

**A)** 256 bytes **B)** (≈65 kilobytes) **C)** ≈16 megabytes **D)** ≈4.3 gigabytes

ii. A microcontroller uses 12-bits to store its integer type using 2's complement. What is the range that the integer can represent?

**A)** −4069 to 4096 **B)** −2048 to 2048 **C)** (−2048 to 2047) **D)** −2047 to 2048

## C. Number Bases

Write your answer in the respective boxes. Each question carries equal weightage. [4 marks]

i. Express the decimal number $47_{10}$ in base-3 (ternary). ..... $1202_3$

ii. Express the base-5 number $1010_5$ in decimal (base-10) ..... $130_{10}$

iii. Express the base-3 number $2112010_3$ in base-9. ..... $2463_9$

iv. Express the hexadecimal (base-16) number $DAD_{16}$ in base-4. ..... $312231_4$

— END OF PAPER —