NATIONAL UNIVERSITY OF SINGAPORE

CS1010E – PROGRAMMING METHODOLOGY

Sample Exam Paper 2

Please DO NOT upload questions and answers onto the Internet.

Time allowed: 2 hours

INSTRUCTIONS TO CANDIDATES

1. This sample assessment paper contains **SIX** questions and comprises **NINE** printed pages, including this page.
2. This is an **OPEN BOOK** assessment. You may bring in any paper material.
3. Calculators are allowed, but not laptops or other electronic devices.
4. You are suggested to use **pencil** to write your programs. **Pen** is preferred for other questions.

Q1. Multiple Choice Questions (MCQs)

You may assume that all necessary header files have been included.

Q1.1 What is printed out by the following program fragment?

```
char c = 'b', d = 'H';
int a = d - toupper(c);
printf("%c %d\n", c, a);
```

A. `B -16`

B. `B 6`

C. `b -16`

D. `b 6`

E. It will give compile-time error.

Q1.2 What is the largest value that could be returned by the following function?

```
int func(int key) {

  int index, arr[] = {2, 4, 6, 8, 10, 12, 14};

  for (index = 0; index < 7; index++) {
    if (arr[index] > key) {
      break;
    }
  }

  return index;
}
```

A. 6

B. 7

C. 8

D. 14

E. None of the above

Q1.3 Which of the following lists will take the most number of passes to complete sorting in ascending order, using the Selection Sort algorithm given in the lecture notes?

A. `int list[] = {3, 4, 1, 2};`

B. `int list[] = {4, 1, 2, 3};`

C. `int list[] = {1, 2, 3, 4};`

D. `int list[] = {4, 3, 2, 1};`

E. The above 4 lists will take an equal number of passes for sorting.

Q1.4 What is the output of the following program fragment?

```
char s1[7] = "ace", s2[] = "bdf";
strcat(s1, s2);
printf("%s\n", s1);
```

A. **acebdf**
B. **bdface**
C. **ace**
D. It will give compile-time error.
E. It will give run-time error.

Q1.5 What is the output of the following program fragment?

```
int i = 0, sum = 0;

do {
  switch (i%4) {
    case 1:
      sum += i;
      break;
    case 3:
      sum -= i;
      break;
    default:
      sum *= 2;
  }
  i++;
} while (i <= 5);

printf("%d %d\n", i, sum);
```

A. **4 3**
B. **5 -2**
C. **6 -4**
D. **6 3**
E. None of the above

Q1.6 Suppose a text file "numbers.txt" contains the following values:

```
2 3
7
-5 2 8
```

What is printed out by the following program fragment?

```
FILE *infile;
int i, a, b, sum = 0;

infile = fopen("numbers.txt", "r");
for (i = 0; i < 3; i++) {
  fscanf(infile, "%d %d", &a, &b);
  sum += -a + b;
}
printf("%d\n", sum);
```

A. `0`

B. `17`

C. `-5`

D. It will give compilation error.

E. It will give run-time error.

Q1.7 What does the following function return, assuming both **x** and **k** are positive integers?

```
int what(int x, int k) {

  int i;
  for (i = 0; i < k; i++) {
    x *= 2;
  }

  return x;
}
```

A. k^x

B. $2^k * x$

C. $2^{k+1} * x$

D. $2 * x^k$

E. None of the above

Q1.8 What does the following function return?

```
int mystery(int x, int y) {
  if (x == 0) {
    return y;
  } else if (x < 0) {
    ++x;
    --y;
    return mystery(x, y);
  } else {
    --x;
    ++y;
    return mystery(x, y);
  }
}
```

A. It returns the value of **y**.

B. It returns the value of **x–y**.

C. It returns the value of **x+y**.

D. It returns the value of **x*y**.

E. It will give compile-time error.

Q1.9 What is printed out by the following code fragment?

```
double num = (double) rand() / RAND_MAX * 6 + 1;
printf("%d\n", (int) num * 2 + 1);
```

A. It always prints 3

B. It prints either 3 or 13

C. It prints either 3 or 15

D. It prints a random odd integer in the range [3, 13] (both inclusive)

E. It prints a random odd integer in the range [3, 15] (both inclusive)

Q1.10 What is the output of the following program?

```
#include <stdio.h>

int func(int [], int);

int main(void) {
  int a[] = {1, 2, 2, 1, 2, 2, 1}, i;

  for (i = 4; i <= 7; i++) {
    printf("%d ", func(a, i));
  }
  printf("\n");

  return 0;
}

int func(int arr[], int size) {
  int i;
  for (i = 0; i < size/2; i++) {
    if (arr[i] != arr[size-i-1]) {
      return 0;
    }
  }
  return 1;
}
```

A. **0 1 0 1**

B. **0 1 1 0**

C. **1 0 0 1**

D. **1 0 1 1**

E. **1 1 0 1**

Q2. An infinite sequence of white and black balls is arranged in the following pattern.

There is (are):

- 1 black ball between the 1st and the 2nd white balls.
- 2 black balls between the 2nd and the 3rd white balls.
- 3 black balls between the 3rd and the 4th white balls.
- …
- k black balls between the kth and the (k+1)th white balls.

The example below shows the first 16 balls in the sequence.

Write a function

```
int check_colour(int n)
```

that returns an integer representing the colour of the **n**th ball in the sequence (0 for black and 1 for white).

For example:

- Function call **`check_colour(1)`** returns 1 (colour of the 1st ball).
- Function call **`check_colour(2)`** returns 0 (colour of the 2nd ball).
- Function call **`check_colour(6)`** returns 1 (colour of the 6th ball).

Q3. Write a function

```
int count_substring(char str[], char c1, char c2)
```

that counts the number of substrings in **`str`** that begins with character **`c1`** and ends with character **`c2`**. You may assume that **`c1`** is different from **`c2`**.

For example, if **`str`** is **`"10-31S0K"`**, **`c1`** is **`'1'`** and **`c2`** is **`'0'`**, there are 3 substrings that begin with **`'1'`** and ends with **`'0'`**, namely: **`"10"`**, **`"10-31S0"`**, and **`"1S0"`**.

Q4. The *Collatz conjecture* states that given any positive integer n, if it is **odd** we multiply it by 3 and add 1, or if it is **even** we divide it by 2. If we repeat this process, n will eventually reach 1 where the process stops. The number of steps taken for n to reach 1 is called the cycle-length of *n*. By definition, the cycle-length of 1 is 0.

For example, given n = 6, the sequence obtained is:

6 → 3 → 10 → 5 → 16 → 8 → 4 → 2 → 1

and the cycle-length of 6 is thus 8.

Write a recursive function

```
int count_cycle_length(int n)
```

to compute and return the cycle-length of **n**, assuming that **n** is a positive integer.

You are **not** allowed to use any loop constructs (*for*, *while* or *do-while*) in this question.

Q5. You are given a square-shaped map with square cells, where the length of each side of a cell is 1. The value of a cell is either 0 (for sea) or 1 (for land). Two cells of both 1s are part of the same island if they are adjacent to each other either vertically or horizontally.

For simplicity, you may assume that the maximum size of the map is 10 and there is exactly one island in a given map.

Below are two example maps in which islands are highlighted.

0	0	**1**	**1**	**1**
0	0	**1**	**1**	**1**
0	0	**1**	**1**	**1**
0	0	0	0	0
0	0	0	0	0

Area of sea = 16
Perimeter of island = 12

0	0	**1**	0	0
0	**1**	**1**	**1**	**1**
0	**1**	**1**	**1**	0
0	0	**1**	**1**	0
0	0	0	0	0

Area of sea = 15
Perimeter of island = 16

(a) Write a function **`int compute_sea_area(int map[10][10], int size)`** that computes the area of the sea in a given **`map`** of **`size`** rows and **`size`** columns.

(b) Write a function **`int compute_island_perimeter(int map[10][10], int size)`** that computes the perimeter of the island in a given **`map`** of **`size`** rows and **`size`** columns.

Q6. Write a C program to model a deck of playing cards. A deck consists of 52 cards, each card containing a **rank** and a **suit**. The ranks are (from lowest to highest) 2, …, 9, 10, Jack, Queen, King and Ace, represented by the characters '2', …, '9', 'T', 'J', 'Q', 'K' and 'A' respectively. The suits are (from lowest to highest) Clubs, Diamonds, Hearts and Spades, represented by the characters 'C', 'D', 'H' and 'S' respectively.

(a) Define a structure called **`card_t`** to represent a playing card, and declare an array of cards, called **`deck`**, to represent the deck of cards.

(b) Write a function **`init_deck(card_t deck[])`** to initialize the deck, i.e. create the deck of cards. The deck should be created starting with all cards of the lowest suit, and for cards of the same suit, in increasing order of their ranks. Hence, the first card is the 2 of Clubs and the last card is the Ace of Spades.

After the **`init_deck()`** function is called, the **`deck`** array should contain the following cards (with each line showing 13 cards):

```
C2 C3 C4 C5 C6 C7 C8 C9 CT CJ CQ CK CA
D2 D3 D4 D5 D6 D7 D8 D9 DT DJ DQ DK DA
H2 H3 H4 H5 H6 H7 H8 H9 HT HJ HQ HK HA
S2 S3 S4 S5 S6 S7 S8 S9 ST SJ SQ SK SA
```

(c) Write a function called **`shuffle_deck(card_t deck[])`** to shuffle the **`deck`** by writing a loop with 52 iterations. In iteration *i* we generate a random integer *r* in the range [0, 51] and swap deck[*r*] with deck[*i*].

For example, after shuffling the **`deck`** array might look something like this (i.e. the cards are now randomly ordered):

```
D9 C2 H9 S6 C8 D3 H2 DJ S3 D7 S8 H3 HQ
SJ CA DQ C4 D4 H4 HT CJ C3 HJ D8 H7 SK
C7 S4 DT D2 SQ H6 DA H8 ST C5 S7 S9 DK
D6 S2 CT HA C6 SA D5 CK C9 S5 CQ H5 HK
```

A sample run is shown below.

```
The original deck:
C2 C3 C4 C5 C6 C7 C8 C9 CT CJ CQ CK CA
D2 D3 D4 D5 D6 D7 D8 D9 DT DJ DQ DK DA
H2 H3 H4 H5 H6 H7 H8 H9 HT HJ HQ HK HA
S2 S3 S4 S5 S6 S7 S8 S9 ST SJ SQ SK SA

The shuffled deck:
DA D5 CA S7 D8 H6 S2 C7 HA D3 D4 HQ DT
SK H9 D7 H2 H4 S5 H8 CT C5 H5 HT SA C9
SQ C3 H7 S9 S8 D2 S4 SJ HK CJ D6 C6 DQ
C2 D9 CK S6 C4 ST CQ S3 C8 DK DJ HJ H3
```

=== END OF PAPER ===