NATIONAL UNIVERSITY OF SINGAPORE

CS1010E – PROGRAMMING METHODOLOGY

Sample Exam Paper 1

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. **[Total: 12 marks]**

Write down the output of each of the following program or program fragments.

1.1

```
char c = 'f', f = 'c';
printf("%c\n", c-(c-f)/2);
```

1.2

```
char c = 'f', f = 'c';
char str1[] = {'p', 'e', 'n', '\0', 'c', 'i', '\0'};
char str2[13];

strcpy(str2, "apple");
strcat(str2, str1);

puts(str2);
```

1.3

```
int a[] = {1, 2, 3, 4, 5};
printf("%d\n", a[a[a[a[1]]]]);
```

1.4

```
int a = 7;
double b = 5.99;

int *p;
double *q;

p = &a;
q = &b;

*p = *q;
*q += *p;

printf("%d %.2f\n", a, b);
```

1.5

```
#include <stdio.h>

void what(int r[], int *q);

int main(void) {

  int x[5] = {0, 1, 2, 3};

  what(x, &x[3]);

  printf("%d %d\n", x[3], x[4]);

  return 0;
}

void what(int r[], int *q) {

  int k;

  for (k = 0; k < 5; k++) {
    r[k] += *q;
  }
}
```

1.6

```
int x, y, count;

count = x = 0;

while (x < 10) {
  y = 10 – x;
  while (y > x) {
    count++;
    y--;
  }
  x += 2;
}

printf("count = %d\n", count);
```

1.7

```
#include <stdio.h>

int f(int n);

int main(void) {

  int a, b;

  a = f(6);
  b = f(5);
  printf("%d\n", a-b);

  return 0;
}

int f(int n) {
  if (n == 1) {
    return 2;
  } else if (n == 2) {
    return 3;
  } else {
    return f(n-2) + f(n-1);
  }
}
```

1.8

```
#include <stdio.h>
#define N 5

int main(void) {

  int i, a[N] = {5, 4, 3, 2}, b[N];

  for (i = 0; i < N; i++) {
    b[i] = a[(i+3)%5];
  }

  for (i = 0; i < N; i++) {
    printf("%d ", b[i]);
  }
  printf("\n");

  return 0;
}
```

1.9

```
#include <stdio.h>
#include <string.h>

typedef struct {
  int a;
  char b[10];
} mystruct_t;

void weird(int a, char b[]);

int main(void) {

  mystruct_t mine = {5, "abc"};

  weird(mine.a, mine.b);
  printf("%d %s\n", mine.a, mine.b);

  return 0;
}

void weird(int a, char b[]) {
  a = 10;
  strcpy(b, "xy");
}
```

1.10

```
#include <stdio.h>
#define N 9

int main(void) {
  int a, b;
  for (a = 1; a < N/2; a++) {
    for (b = a; b < N/2; b++) {
      if ( a*a+b*b < N*N/8 ) {
        printf("(%d,%d) ", a, b);
      }
    }
  }
  printf("\n");

  return 0;
}
```

1.11

```
#include <stdio.h>

int functionXYZ(char *, char);

int main(void) {
  char s[] = "abbacadaba";
  printf("%d\n", functionXYZ(s, 'b'));
  return 0;
}

int functionXYZ(char *str, char ch) {
  int i = 0, j = 0;
  while (str[i]) {
    if (str[i] == ch) {
      j++;
    }
    i++;
  }
  return j;
}
```

1.12

```
int a[4][4] = {{2,0,1},{0,1},{1}};
int b[4] = {0};
int i, j;

for (i = 0; i < 4; i++) {
  for (j = 0; j < 4; j++) {
    a[i][j] += a[3-i][3-j];
  }
}

for (i = 0; i < 4; i++) {
  for (j = i; j < 4; j++) {
    b[i] += a[i][j];
  }
}

printf("%d %d %d %d\n", b[0], b[1], b[2], b[3]);
```

Q2. Adam wrote the following function that takes in an array of 5 integers and outputs the largest integer in the array that is less than 50 (assuming its existence). For example, given `list` = {76, 35, 98, 49, 23}, the function returns 49. Adam submitted his code to CodeCrunch and found that his code failed some test cases.

```
int largest(int list[]) {
  int i = 1;
  while (list[i] < 50) {
    i++;
  }
  return list[i];
}
```

a) Provide a test case (i.e. an array) that will alert Adam to the incorrect initialization of `i` to 1.

b) Provide a test case that will highlight the incorrect logic of the loop condition.

c) Provide a test case that will highlight the failure to check for array bound.

Q3. Write a <u>recursive</u> function

`int rdup(int n, int d)`

that accepts a positive integer **n** and a digit **d**. It then returns another integer **n'** in which all occurrence of digit **d** are removed from **n**.

For example, given an integer 10334 and a digit 3*,* **rdup()** returns the transformed integer 104. More examples are given below:

n	**d**	**n'**
980828	8	902
1020	1	20
333	3	0
453	6	453

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

Q4. Write a function

```
void anti_diagonal_matrix(int mtx[N][N], int n)
```

that fills the two-dimensional array **`mtx`** with integers from 1 to **`n*n`** in an anti-diagonal fashion. **`N`** is a pre-defined constant representing the physical size of **`mtx`** and **`n`** is the actual size of **`mtx`** in a sample run (i.e. **`n`** rows and **`n`** columns). You may assume that **`N`** > **`n`**.

Two examples are given below.

1	3	6	10
2	5	9	13
4	8	12	15
7	11	14	16

`n` = 4

1	3	6	10	15
2	5	9	14	19
4	8	13	18	22
7	12	17	21	24
11	16	20	23	25

`n` = 5

Q5. Write a function

```
int shuffle(char s1[], char s2[], char s3[])
```

that checks if **`s3`** is a *shuffle* of **`s1`** and **`s2`**. It returns 1 if so, or 0 otherwise.

`s3` is said to be a shuffle of **`s1`** and **`s2`** if it can be formed by interleaving the characters of **`s1`** and **`s2`** in a way that maintains the left to right ordering of the characters from both strings.

For example, given **`s1 = "abc"`** and **`s2 = "def"`**,

- **`"adbcef"`** is a shuffle of **`s1`** and **`s2`** since it preserves the character ordering from each string.
- **`"defabc"`** is also a shuffle of **`s1`** and **`s2`**.
- But **`"deabc"`**, **`"dafbce"`** and **`"adbceff"`** are not.

Q6.

Write a complete C program to read in a list of health screen readings:

- Each input line represents a reading consisting of 2 numbers: a **score** (of type `double`) indicating the health score, and a **freq** (of type `int`) indicating the number of people with that score. A structure type `reading_t` (comprising two members: **score** and **freq**) should be created to describe readings.
- You may assume that there is at least 1 reading and at most 50 readings.
- The input will end with the reading 0 0, or when 50 readings have been read, whichever occurs earlier.
- You may assume that all readings are positive except the last one (0 0).

As the readings are gathered from various clinics, there might be duplicate scores in the input. You are to determine how many unique scores there are and which score gives the highest combined frequency (assume the uniqueness of such a score).

A sample run is shown below with the user's input highlighted in **bold**.

```
Enter score and frequency:
5.2135 3
3.123  4
2.9    3
0.87   2
2.9    2
8.123  6
3.123  2
7.6    3
2.9    4
0.111  5
0      0
Number of unique scores = 6
Score 2.900 has the highest combined frequency: 9
```

=== END OF PAPER ===