# IERG2080 Spring 2023 - Test #2

30 Mar 2023

Student Name: ____________________ Student ID: ____________________

**Rules:**

1. Time limit: 90 minutes
2. Unless otherwise specified, use C to answer the programming problems in this test.
3. **Assume the size of `int` is 4 bytes.**
4. Answer all questions. Write your answers of Parts I and II in the answer book. For the answers of Part III, you can either write them in the answer book, or submit `.c` files to the Blackboard.
5. Write your name and SID in both the question paper and the answer book.
6. You are not allowed to use any electronic devices during the test except the PC provided and/or your calculator.
7. You have restricted Internet access for browsing some useful webs in the whitelist after setting up the proxy as instructed.

## Proxy Configuration

You may set up the proxy following the instructions below for accessing the (restricted) Internet.

**Step 1:** Open the web browser **Firefox**. Open **Settings**.
**Step 2:** Go to **General → Network Settings → Settings…**
**Step 3:** Select **Manual proxy configuration**.Tick **Also use this proxy for HTTPS**. Enter **137.189.99.132** in the HTTP Proxy textbox, and enter **3128** in the Port textbox. Then, press **OK**.
**Step 4:** Try accessing the Blackboard using Firefox and see if your configuration is fine.
`https://blackboard.cuhk.edu.hk/`

Some useful webs in the whitelist:

- Lecture notes
  `https://blackboard.cuhk.edu.hk/`
- Man pages
  `https://man7.org/linux/man-pages/`
- C standard library reference
  `https://www.tutorialspoint.com/c_standard_library/`

## Note on Visual Studio

For those who want to use Visual Studio, you need to put the following

```
#define _CRT_SECURE_NO_WARNINGS
```

at the beginning of your code, i.e., even before `#include<stdio.h>`
Otherwise, Visual Studio identifies some functions like scanf() as "unsafe" and gives compilation errors.

## Part I: Multiple Choices (30%)

Each question in this part takes 5%.

1. Which of the following encodings support emojis?
   A. ANSI
   **B. Unicode**
   C. Big5
   D. None of the above

Consider the following struct in Problems 2 and 3.

```
#pragma pack(2)
struct temp {
   short s1:1;
   short s2:2;
   short s3:3;
   int i;
};
```

2. What is the size of this struct?
   A. 2 bytes
   B. 4 bytes
   **C. 6 bytes**
   D. 8 bytes

3. Is there any padding byte in this struct? If so, where is the padding byte?
   A. No padding byte
   B. 1 padding byte before `s2`
   C. 1 padding byte before `s3`
   D. **1 padding byte before `i`**

4. Can GCC generate executables for an architecture other than that of the host?
   **A. Yes**
   B. No

5. Is core dump always generated when segmentation fault occurs?
   A. Yes
   **B. No**

6. What is the output of the following Bash commands?

```
i="true or false"
echo ${i: -5}
```

A. true or false
B. true and false
C. true
**D. false**

## Part II: Short Questions (30%)

**Problem 7 [3 marks]**
Suppose we want to compile the C source file “`magic.c`” into an executable “`magic.exe`” (without double quotes in the names) using GCC with the best optimization level for speed. The program uses the APIs from the library named “`curses`”. Although the target users of this program use the same OS and architecture, they may not have installed the library “`curses`”. Write the command that calls GCC to compile the program such that the target users can run the program successfully without installing the library “`curses`” on their own.

```
gcc magic.c -o magic.exe -O3 -lcurses -static
```

**[0.5 mark] gcc**
**[0.5 mark] magic.c**
**[0.5 mark] -o magic.exe**
**[0.5 mark] -O3**
**[0.5 mark] -lcurses OR -l curses**
**[0.5 mark] -static**

**Problem 8 [5 marks]**
Consider the following structs.

```
typedef struct hi *foo, bar;
typedef struct bye good, *bad;
struct hi {
    foo f;
    bad d;
    bar *b;
    int i;
};
struct bye {
    foo f;
    bar b;
    bad d;
    good *g;
    int i;
};
```

Assume all pointers are set up probably so that they can be dereferenced without segmentation faults. Suppose we start with a variable `hello` of type `struct hi`. Fill in the blanks so that the member `i` at the end can be accessed.

a) `hello.d->g->b.b->i`
b) `(&hello)->f->b->d->f->i`

**Each blank 0.5 mark**

**Problem 9**
Consider the following C code.

```
#include <stdio.h>

int main(void)
{
    int x = 1, y = 2, z = 0;
    printf("%d\n", x or y and z);
    return 0;
}
```

This code cannot be compiled using GCC in Linux.

a) **[1 mark]** Describe how to debug the code if you are allowed to modify the code in `main()` only.

   **Replace or into || and replace and into &&**
   **`printf("%d\n", x || y && z);`**
   **Each replacement 0.5 mark**

b) **[1 mark]** Describe how to debug the code if you are NOT allowed to modify the code in `main()`.

   **[1 mark]**
   **`#include <iso646.h>`**

   **OR**

   **[each 0.5 mark]**
   **`#define or ||`**
   **`#define and &&`**

c) **[1 mark]** What is the output of the program after debugging?

   **1**

**Problem 10**

One main difference between the usage of `mbtowc()` and `mbrtowc()` is that the latter function requires a state variable pointer in its argument.

a) **[1 mark]** As the algorithm applied in both functions is the same, the former function also requires a state variable. Why does the former function not require a state variable pointer as an argument?

**Any answer mentioning that**
**mbtowc() has a (static) state variable inside the function**
**OR**
**mbtowc() records the state inside it**

b) **[1 mark]** Why does the latter function require a state variable pointer instead of a state variable instance?

**Any answer describing that**
**The function needs to change the value of the state variable (so it needs the address of the instance)**

**Problem 11**

Consider storing the complex number $3.14159 + 2.71828i$ in a complex variable in C using double-precision floating-point numbers.

a) **[1 mark]** What is the size of this complex variable?

**16 bytes [-0.5 if wrong/missing unit]**

b) Write the bytes stored in this variable in hexadecimal if the machine is using

i) **[2 marks]** big endian.

```
40 09 21 F9 F0 1B 86 6E 40 05 BF 09 95 AA F7 90
```

ii) **[2 marks]** little endian.

```
6E 86 1B F0 F9 21 09 40 90 F7 AA 95 09 BF 05 40
```

**Each byte 0.125 mark**
**Accept no space between bytes**
**Accept writing 0x in the prefix, or subscript 16 at the end**

**Problem 12**
**Download the file `cafe.c` from Blackboard.** This file can be compiled by Visual Studio in Windows, but not by GCC in Linux. Below is the hexdump of the file.

```
00000000: fffe 2300 6900 6e00 6300 6c00 7500 6400  ..#.i.n.c.l.u.d.
00000010: 6500 2000 3c00 7300 7400 6400 6900 6f00  e. .<.s.t.d.i.o.
00000020: 2e00 6800 3e00 0d00 0a00 0d00 0a00 6900  ..h.>.........i.
00000030: 6e00 7400 2000 6d00 6100 6900 6e00 2800  n.t. .m.a.i.n.(.
00000040: 7600 6f00 6900 6400 2900 0d00 0a00 7b00  v.o.i.d.).....{.
00000050: 0d00 0a00 0900 7000 7500 7400 7300 2800  ......p.u.t.s.(.
00000060: 2200 6300 6100 6600 e900 2200 2900 3b00  ".c.a.f...".).;.
00000070: 0d00 0a00 0900 7200 6500 7400 7500 7200  ......r.e.t.u.r.
00000080: 6e00 2000 3000 3b00 0d00 0a00 7d00 0d00  n. .0.;.....}...
00000090: 0a00                                     ..
```

a) **[1 mark]** What is the encoding used in this file?
   - Hints: Open the file using Notepad.

   **UTF-16 LE**

b) **[1 mark]** Why can Visual Studio and Notepad identify the encoding of the file?
   - Hints: All information about encoding is stored in the content of the file.

   **Any answer mentioning FFFE**
   **The encoding is represented by the bytes FFFE at the beginning of the file.**

c) **[1 mark]** The unicode code point of é is 0x00E9. However, as shown in the hexdump, this character is represented by the bytes 0xE900. Why?

   **Any answer mentioning little endian**
   **The encoding is little endian, so the byte ordering is reversed.**

d) **[1 mark]** At the end of the file, the hexdump shows the bytes 0x0D000A00. What do they mean?

   **Any answer mentioning \r\n OR newline OR CRLF**

**Problem 13**
Given a multibyte character 0xF09F8D8C in UTF-8 encoding.

a) **[2 marks]** Which of the following data types can be used to store this character? Choose all valid answers.

   `unsigned char`     `wchar_t`     `char16_t`     `char32_t`

   **Every correct answer 1 mark**
   **Every wrong answer -1 mark**

b) **[4 marks]** What is the Unicode code point (in hexadecimal) of this character?

**1F34C**
**Accept 0x or U+ as prefix, or subscript 16 at the end**

**If answer not correct, then:**
**[1 mark] Correctly write down the bits of the UTF-8 characters**
**[1 mark] Correctly extract the bits for forming the Unicode code point (can earn 1 more mark if didn't write down the above step)**
**[1 mark] Form meaningful bytes (maybe wrongly aligning to the left side, or reversing the bytes somehow)**

c) **[2 marks]** Can we use \u to type this character using its Unicode code point in a string literal? If yes, write down the escape sequence. If not, explain the reason.

**No [1 mark]**
**Because \u only support 4 hexadecimal digits [1 mark]**
Remark: Each \u is converted to the corresponding encoding during compilation, so writing 2 \u won't work if the encoding is UTF-8 as they are converted individually

## Part III: C Programming (40%)

**Problem 14 [20 marks]**
Transmitting electric power from a power station to the city is the backbone of power systems. Although the loss of power during transmission is reduced by applying a high voltage to the transmission line, the loss effect is still significant when the distance is long. Let $V_S$ and $I_S$ be the sending voltage and sending current respectively, and $V_R$ and $I_R$ be the receiving voltage and receiving current respectively. Their relation can be modeled as

$$\begin{pmatrix} V_S \\ I_S \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} V_R \\ I_R \end{pmatrix}$$

with $A = D = cosh(\gamma x)$, $B = Z_c sinh(\gamma x)$, and $C = \frac{1}{Z_c} sinh(\gamma x)$, where $x$ is the length of the transmission line, $\gamma$ is the propagation constant, and $Z_c$ is the characteristic impedance. These two parameters can be expressed as $\gamma = \sqrt{zy}$ and $Z_c = \sqrt{z/y}$ , where $y$ is the series impedance per unit length, and $z$ is the shunt admittance per unit length.

**Task:** Your task is to calculate the sending voltage and sending current using the readings of receiving voltage and receiving current. Write a C program that reads $x$, $z$, $y$, $V_R$ and $I_R$ from stdin, and prints $V_S$ and $I_S$ in stdout up to 6 decimal places. Assume all inputs are valid, where

$x$ is a positive real number;
$z$, $y$ are non-zero complex numbers in standard form, i.e., $a + jb$ where $j = \sqrt{-1}$ ;
$V_R$, $I_R$ are non-zero complex numbers in polar form (in degree), i.e., $r\angle\theta°$

For the output, both $V_S$ and $I_S$ are complex numbers in polar form (in degree).

| | Example 1 | Example 2 |
|---|---|---|
| Values | $x = 500$<br>$z = 0.03 + j0.35$<br>$y = j4.44 \times 10^{-6}$<br>$V_R = 124130$<br>$I_R = 335.67$ | $x = 370.14$<br>$z = 0.16029 + j0.82772$<br>$y = j5.105 \times 10^{-6}$<br>$V_R = 215000\angle 30°$<br>$I_R = 332.84\angle 26.26°$ |
| Input | `500`<br>`0.03 0.35`<br>`0 4.44e-6`<br>`124130 0`<br>`335.67 0` | `370.14`<br>`0.16029 0.82772`<br>`0 5.105e-6`<br>`215000 30`<br>`332.84 26.26` |
| Output | `119623.575021∠28.442415°`<br>`377.892748∠44.167610°` | `205122.549260∠59.956633°`<br>`437.362455∠87.604419°` |

Some useful information:

- $x° = x\pi/180$ radian
- $x$ radian $= (180x/\pi)°$
- $\pi = 4\,tan^{-1}1$
- $r\angle\theta° = r\,cos(\theta\pi/180) + j\,r\,sin(\theta\pi/180)$
- Unicode code point of ∠ is 0x2220
- Unicode code point of ° is 0x00B0

**Each of the following takes 1 mark:**

1. **#include <complex.h>**
2. **Reading the inputs (without merging into complex number)**
3. **Merge the inputs into complex number z**
4. **Merge the inputs into complex number y**
5. **Merge the inputs into complex numbers V_R from the polar form**
6. **Merge the inputs into complex numbers I_R from the polar form**
7. **Calculate gamma (need to use csqrt)**
8. **Calculate Z_c (need to use csqrt)**
9. **Calculate A (need to use ccosh)**
10. **Calculate B (need to use csinh)**
11. **Calculate C (need to use csinh)**
12. **Calculate D (need to use ccosh)**
13. **Calculate V_S**
14. **Calculate I_S**
15. **Extract the modulus of V_S (using cabs or other means) and extract the argument of V_S (using carg or other means)**
16. **Extract the modulus of I_S (using cabs or other means) and extract the argument of I_S (using carg or other means)**
17. **Print the answer of V_S (ok for wrong answer)**

18. **Print the answer of I_S (ok for wrong answer)**
19. **Convert from radian to degree in the output**
20. **Use \u2220 and \u00B0 for printing ∠ and °**

**Problem 15 [20 marks]**
Assume the user input is a positive integer at most 100. Write a C program that prints the pattern following the rules as stated below:

- The number of rows and the number of columns equal to the user input.
- Each row/column is classified as either a special row/column or not.
- Suppose the leftmost column is the 1st column. The (1+2+…+i)-th column in the pattern is the ith special column.
- Suppose the topmost row is the 1st row. The odd rows are the special rows.
- A dot is printed when both row and column are not special.
- A hashtag is printed when either the row or the column is special.
- An at sign is printed when both the row and the column are special.

In each sample run, the underlined italic characters are the inputs from the user.

| Sample Run 1 | Sample Run 2 | Sample Run 3 |
|---|---|---|
| *8*<br>`@#@##@##`<br>`#.#..#..`<br>`@#@##@##`<br>`#.#..#..`<br>`@#@##@##`<br>`#.#..#..`<br>`@#@##@##`<br>`#.#..#..` | *9*<br>`@#@##@###`<br>`#.#..#...`<br>`@#@##@###`<br>`#.#..#...`<br>`@#@##@###`<br>`#.#..#...`<br>`@#@##@###`<br>`#.#..#...`<br>`@#@##@###` | *11*<br>`@#@##@###@#`<br>`#.#..#...#.`<br>`@#@##@###@#`<br>`#.#..#...#.`<br>`@#@##@###@#`<br>`#.#..#...#.`<br>`@#@##@###@#`<br>`#.#..#...#.`<br>`@#@##@###@#`<br>`#.#..#...#.`<br>`@#@##@###@#` |

**Each of the following takes 1 mark unless specified otherwise:**

1. **#include <stdio.h>**
2. **Correct main function**
3. **No printing extra texts, e.g., don't prompt asking the user for input [5 marks]**
4. **Read the input**
5. **Outer loop with correct number of iterations**
6. **Inner loop with correct number of iterations**
7. **Printing # for odd rows (if counts from 1)**
8. **Printing . for even rows (if counts from 1)**
9. **Identify special columns [4 marks]**
   **If hard-coding the columns, then each correct column takes 4/13 mark and each wrong column deduct 0.5 mark. Min 0 mark for this item**
10. **Printing # at special columns**
11. **Printing @ at special rows+columns**
12. **Newline characters**
13. **Overall printing the correct output**