請勿攜去
Not to be taken away

香 港 中 文 大 學
The Chinese University of Hong Kong

二 0 一八至一九年度上學期科目考試
Course Examination 1st Term, 2018-19

科目編號及名稱
Course Code & Title : IERG2080 INTRODUCTION TO SYSTEMS PROGRAMMING

時間　　小時　　分鐘
Time allowed : 2 hours 0 minutes

學號　　座號
Student I.D. No. : ..... Seat No. : .....

(Closed-book & notes. Answer ALL questions. Total 100 marks.)

1. C Development Tools (20 marks)

   (a) Give one function/purpose of the C pre-processor. (5 marks)

   (b) Give one function/purpose of the C compiler. (5 marks)

   (c) Give one function/purpose of the C linker. (5 marks)

   (d) Give one function/purpose of GDB. (5 marks)

2. Consider the following codes for computing the factorial of a number x!=x(x-1)(x-2)…(2)(1). (20 marks)

```
#include <stdio.h>

int main(void) {
   int const x = 10;
   int total = 0;
   for (int i=0; i<=x; i++)
      total *= i;
   printf("Value of %i! = %i\n", i, total);
   return 0;
}
```

   (a) What is the purpose of "`#include <stdio.h>`"? (5 marks)

   (b) The above codes have a bug that generates a compile-time error. Identify the bug (2 marks) and propose a fix for the bug. (3 marks)

   (c) After fixing the bug in (b) the codes now compile successfully. However the factorial value printed is always zero. Find the bugs (2 marks) and propose fixes for them (3 marks).

   (d) After the previous bug fixes the program now correctly print the factorial value for 10!=3628800. However if one initializes the variable `x` to 20 then the program prints a negative value for the factorial of 20. Identify the bug (2 marks) and propose a fix (3 marks).

3. Consider the following code fragment and answer the following questions. (20 marks)

```
#include <stdlib.h>

int x;

int *func(void) {
   static int s;
   int y;
   int *pz = malloc(sizeof(int));
   return &y;
}
```

(a) State the storage class (static or automatic) for each of the 5 variables (x, s, y, pz, and *pz). (5 marks)

(b) Where are the memory allocated from (stack, heap, or data segment) for each of the 5 variables (x, s, y, pz, and *pz). (5 marks)

(c) State if the variable is initialized for each of the 5 variables (x, s, y, pz, and *pz). (5 marks)

(d) There is a memory-related bug in the function func(). Identify it (2 marks) and briefly explain why (3 marks).

4. Consider the following code fragment and answer the following questions. (20 marks)

```
#include <stdlib.h>

int *InitArray(int *ptr, size_t size) {
   do {
      ptr[size--] = 0;
   } while (size);
   return ptr;
}

int main(void) {
   int const arraySize = 100;
   int *pAry = malloc(4*arraySize);
   InitArray(pAry, arraySize);
   return 0;
}
```

(a) The way `malloc()` is used is non-portable (i.e., it may not work in a different OS/hardware platform). Explain why (2 marks) and propose a fix (3 marks).

(b) The variable `arraySize` is passed to the function `InitArray()` as the second argument (i.e., `size`) where `size` is modified inside the function. What will be the final value for the variable `size` after the do-while loop ends? (1 mark) Does the final value carry over to `arraySize` in `main()`? (2 marks) Why or why not? (2 marks)

(c) There is an array-out-of-bound access runtime bug in the function `InitArray()`. Does the C language check for array-out-of-bound errors? (1 mark) Identify the bug (1 mark) and propose a fix (3 marks).

(d) Suppose the starting memory address for the array pointed to by `pAry` is equal to `P`. Then what is the memory address for `pAry[i]` where `i` is an `int` and assuming `int` type is 4 bytes? (1 mark)

(e) Consider allocating a 100x200 elements 2D array using `malloc()` as follows:

```
void f() {
   int *p2DAry = malloc(sizeof(int)*100*200);
   int i=10, j=20;
   int *ptr = p2DAry + ?? // incomplete, to calc address of p2DAry[i][j]
}
```

Can one use the 2D array using the syntax `p2DAry[i][j]`? (1 mark) Why or why not? (1 mark)

Complete the line of code for calculating the pointer `ptr` for addressing `p2DAry[i][j]`. (2 marks)

5. Consider the following code fragment and answer the following questions. (20 marks)

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

void StringUpper(char *dest, char *src) {
   while (*src) {
      *dest = toupper(*src); // convert char to upper case
      ++dest; ++src;
   }
   *dest = 0;
}

int main(void) {
   char input[80];
   printf("Please input a string: ");
   scanf("%s", input);
   char output[80];
   StringUpper(output, input);
   printf("In upper case: %s\n", output);
   return 0;
}
```

(a) If you are a user of the above program then propose an input string that will cause it run into runtime error. (2 marks). Propose a way to prevent that (writing out the codes is optional). (3 marks)

(b) The implementation for the function `StringUpper()` has at least two vulnerabilities which can cause the function to run into runtime errors. Identify two of them (4 marks) and modify the codes to prevent them (6 marks).

(c) Memory-related runtime errors are difficult to debug because they do not always result in segmentation fault. Under what condition will the process be terminated by segmentation fault? (2 marks). Give one scenario where accessing an uninitialized pointer may <u>not</u> cause segmentation fault. (3 marks)

- END -