進階電腦系統理論與實作 (2017 Fall)

系別: _____

學號: _____

姓名: _____

考試時間：300 分鐘 / 12 題 / 合計 300 分

可翻閱書籍、也可使用網頁搜尋引擎

禁止使用編譯器或任何程式開發環境

以合法C語言(C99以上的規格)撰寫程式碼，需列出完整程式碼

1. (25points) 1970 年代到 1980 年代末期，Digital Equipment Corporation (DEC)公司出品的 PDP-11 和 VAX 電腦曾極盛一時，Bell Labs 的研究人員也在 DEC PDP-11 硬體上開發知名的 UNIX 作業系統。然而和今日電腦相比，VAX 和 PDP-11 缺乏布林代數的 AND 和 OR 指令，取而代之的是 ”bit-set” (組合語言寫作 bis) 和 ”bit-clear” (組合語言寫作 bic)，bic 指令可表達為 C語言程式:

```
int bic(int x, int m) { return x & ~m; }
```

作用是在 m 為 1 的每個 bit，將輸出數值的 bit 清除為 0。

請嘗試「只用」上述 bic 和 bis 來實作 `(x | y)` 和 `(x ^ y)`，填入下方函式的內容:

```
int compute_or(int x, int y) { return ?; }
int compute_xor(int x, int y) { return ?; }
```

並解釋 DEC 公司為何用 bic 和 bis 指令取代 AND 和 OR 指令。

參考資料: 《Introduction to Computer Data Representation》(作者Peter Fenwick), 第71-73頁) [2.13]

2. (25points) 給定以下工具函式:

```
/* 0xff if X_MSB(x) = 1 else 0x00 */
uint8_t MSB(uint8_t x) { return 0 - (x >> (8 * sizeof(x) - 1)); }

/* 0xff if a >= b else 0x00 */
uint8_t greater_than(uint8_t a, uint8_t b)
{ return ~MSB(a ^ ((a ^ b) | ((a - b) ^ b))); }

/* 0xff if x > 0 else 0x00 */
uint8_t non_zero(uint8_t x) { return ~MSB(~x & (x - 1)); }
```

請實作以下函式，其原型宣告為:

```
uint8_t add(uint8_t a, uint8_t b, uint8_t *carry);
```

回傳 `(a + b)`，並在 overflow 時，設定 `*carry = 1`。善用前述工具函式。這樣的加法處理可用在哪裡？請描述一個具體情境。 (提示: 密碼學和資訊安全領域)

參考資料：Why Constant-Time Crypto?

3. (25points) 延續 **2.** 的成果，考慮以下乘法實作，輸入 a 與 b，將乘法結果的高、低 8 bits 分別保存於 hi 與 lo，程式碼如下：

```
void multiply(uint8_t a, uint8_t b, uint8_t *hi, uint8_t *lo) {
  uint8_t a1 = a >> 4, a0 = a & 0xf;
  uint8_t b1 = b >> 4, b0 = b & 0xf;
  uint8_t z0 = a0 * b0; uint8_t z1 = a1 * b1;
  uint8_t z2, z1carry;
  z2 = add(a0 * b1, a1 * b0, &z1carry);
    /* 未完，請補上 */
}
```

請實作完整程式碼，並說明應用情境。

4. (25points) 考慮以下 C 程式，改寫自 2002 年 FreeBSD 作業系統的程式碼:

```
void *memcpy(void *dst, void *src, size_t n);
#define KSIZE 1024
char kbuf[KSIZE];
/* copy at most maxlen byte(s) from kernel region to user buffer */
int copy_from_kernel(void *usr_dst, int maxlen) {
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(usr_dst, kbuf, len);
    return len;
}
```

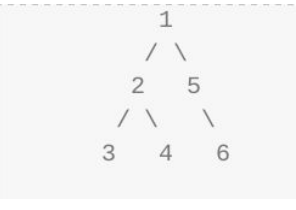
其中 `memcpy` 函式將一段給定長度為n的資料從記憶體某個區域複製到另一個區域 (注意：沒有重疊)，而 `copy_from_kernel` 函式則將作業系統核心維護的資料 (也就是 kernel space) 複製到使用者層級 (user space) 可存取的記憶體區域，對使用者層級來說，大多數核心維護的資料應該不可讀取，因為裡頭可能包含其他用戶層級的資料或者作業系統運作的敏感資訊，不過上述 `kbuf` 裡頭的資料是使用者可讀取的。參數 `maxlen` 是分配給使用者層級的緩衝區長度，這個緩衝區始於 `usr_dst` 指向的地址。

如果某些惡意的開發者或攻擊者嘗試在呼叫 `copy_from_kernel` 函式時，將 < 0 的數值 (也就是負值) 帶入 `maxlen`，會發生什麼事？注意到 `memcpy` 函式的最後一個參數宣告型態為 `size_t`，在 32-bit 硬體環境對應為 unsigned int，而對 64-bit 硬體環境則對應到 unsigned long。換言之，剛提到的 `maxlen` 若用負值帶入 unsigned 整數時，我們將得到一個非常大的正整數。

請描述這樣攻擊情境對 FreeBSD 作業系統的影響，並修改上述程式碼來消除安全漏洞。

參考資料: FreeBSD-SA-02:38.signed-error [59]

5. (25points) 給定以下 C 程式碼和提示圖片，解說程式碼的目的 (如有錯誤，則指出)，並用非遞迴的方式重寫



```
static struct TreeNode *_flatten(struct TreeNode *node){
      if (!node) return NULL;
      if (node->right) node->right = _flatten(node->right);
      if (node->left) {
            struct TreeNode *tmp = node->right;
            node->right = _flatten(node->left);
            struct TreeNode *t = node->right;
            while (t->right) t = t->right;
            t->right = tmp; node->left = NULL;
      }
      return node;
}
void flatten(struct TreeNode *node) { _flatten(node); }
```

注意：需要充分處理 memory leaks，並考慮到執行於現代計算機架構的效率

6. (25points) 在不使用 `sizeof` 運算子的前提下，考慮以下判斷 `int` 是否為 32-bit 寬度的 C 程式碼:

```
#include <stdbool.h>
inline bool is_int_32bit_width() {
    int msb = 1 << 31;
    int beyond_msb = 1 << 32;
    return msb && !beyond_msb;
}
```

在 32-bit 架構的機器上，透過 gcc 編譯程式碼會得到以下錯誤訊息:

```
 warning: left shift count >= width of type [-Wshift-count-overflow]
      int beyond_msb = 1 << 32;
```

請思考以下並提出對應的程式碼修正:

(a) 程式碼哪裡沒依循 C 語言規範？修改程式碼以在 int 至少為 32-bit 機器上運作
(b) 修改程式碼，得以在 int 至少為 16-bit 的任意機器上可運作

7. (25points) 給定以下 awk 程式來實作 Hoare 的 select 演算法:

```
function select(l, u, k, i, m) {
    if (l < u) {
        swap(l, l + int((u - l + 1) * rand())
        m = l
        for (i = l + 1; i <= u; i++)
            if (x[i] < x[l])
                swap(++m, i)
        swap(l, m)
        if (m < k) select(m + 1, u, k)
```

```
        else if (m > k) select(l, m - 1, k)
    }
}
```

程式作用為找出 x[1..n] 裡頭第 k 小的元素，請用 C 語言重寫，仍為遞迴呼叫，並且沿用原本的 tail recursion 技巧。另外也列出非遞迴的實作程式碼並嘗試比較效能差異。

8. (25points) 考慮以下二元搜尋的實作:

```
int binarysearch(datatype *x, datatype t, size_t n) {
    /* Lower and upper limits and middle test value */
    int L = 0, u = n - 1, m;

    while (l <= u) {
        m = (L + u) / 2;
        if (x[m] < t) {
            L = m + 1;
        } else if (x[m] == t) {
            return m;
        } else {
            u = m - 1;
        }
    }
    return -1;
}
```

其功能是自 `x[]` 找出等於等於 t 的單元。注意到 `m = (l + u) / 2` 可能會遇到 overflow，請回答以下:

(a) 可改寫為 `m = l / 2 + u / 2` 嗎？若不行，請說明;

(b) 若無法確認 `u` 永遠會大於等於 `L`，該怎麼避免 overflow 而計算 `(L+u)/2` 呢？

9. (25points) 給定一個 singly-linked list 結構:

```
typedef struct _List {
      struct _List *next;
      int val;
} List;
```

下列`swap`函式的作用是交換存在list中兩個節點(不直接修改內含資料`val`)，指出程式碼的不足或錯誤之處，予以重寫行為正確的版本出來，並指出應用情境。

```
void swap(List *head, List *a, List *b)
{
      List *curr = head;
      List *prevA = NULL; List *prevB = NULL;
      while (curr != a) {
            prevA = curr; curr = curr->next;
      }
      curr = head;
      while (curr != b) {
            prevB = curr; curr = curr->next;
      }
      if (prevA && prevB) {
            prevA->next = b; prevB->next = a;
            List *tmp = a->next;
            a->next = b->next;
      }
}
```

提示: function prototype, assert, error handling, head

10. (25points) 考慮以下程式碼:

```
int32_t func(uint32_t i) {
    i--;
    i |= i >> 1;
    i |= i >> 2;
    i |= i >> 4;
    i |= i >> 8;
    i |= i >> 16;
    i++;
    return i;
}
```

請用正值帶入 `func` 函式，觀察輸出並解說其功能 (不是逐行解說，而是具體功能和應用情境)。

提示: 影像處理, 訊號與系統

11. (25points) 考慮以下字串處理相關程式碼:

```
#include <stdint.h>
size_t strfunc(const char *s) {
    int len = 0;
    while (1) {
        uint32_t x = *(uint32_t *) s;
        if ((x & 0xFF) == 0) return len;
        if ((x & 0xFF00) == 0) return len + 1;
        if ((x & 0xFF0000) == 0) return len + 2;
        if ((x & 0xFF000000) == 0) return len + 3;
        s += 4, len += 4;
    }
}
```

請觀察輸出並解說其功能 (不是逐行解說)，應提及相較於逐一 byte 處理的益處，注意，本程式可能無法在所有硬體平台運作，請解釋。並針對 64-bit 硬體架構提出相似的最佳化程式碼。

提示: alignment

12. (25points) 在一個32-bit little-endian處理器上，以下C程式會得到什麼結果呢？請列出lval, aval, uval的內含值，並解釋為何如此。

```
signed int lval = 0xFEDCBA98 << 32;
signed int aval = 0xFEDCBA98 >> 36;
unsigned int uval = 0xFEDCBA98u >> 40;
```

又，在32-bit RISC上，上述每行C程式對應到幾道組合語言指令呢？(可用ARM或MIPS舉例)