計算機程式設計期末專案

系別：四資工系一　　學號：B11215024　　學生姓名：劉柏毅

此專案由內部連結的全域變數作為資料的基礎，並配合函式，對條碼（barcode）進行處理，以下為主要的資料組成：

```
typedef union data_s
{
    struct
    {
        uint8_t b1: 1;
        uint8_t b2: 1;
        uint8_t b3: 1;
        uint8_t b4: 1;
        uint8_t b5: 1;
    };
    char    character;
} data_t;
```

圖 1

如圖 1 所示，以上為條碼的資料框架，其程式中的資料結構使用 5 個 bit 作為基礎，並依此來解碼成對應的字元，每個格式耗費 1 Byte，根據要求 DATA_SIZE 應設為 32：

```
static data_t   data[DATA_SIZE] = {'\0'};
```

```
enum STATUS_FLAG
{
    STATUS_EMPTY_FLAG,
    STATUS_EXIT_FLAG,
    STATUS_BAD_FLAG,
    STATUS_BAD_C_FLAG,
    STATUS_BAD_K_FLAG,
};
```

上述提供了 5 種狀態以供狀態流的使用，分別表示空旗標、跳脫旗標、無效旗標、無效的 C 驗證碼以及無效的 K 驗證碼，以下變數供執行與狀態流使用（狀態碼預設為空旗標）：

```
static uint8_t  step, status;
static uint8_t  status_code = STATUS_EMPTY_FLAG;
```

根據要求，對常數有以下定義：

```
#define CODE_SIZE   150
#define DATA_SIZE   32
#define DATA_WIDTH  5
#define UPPER_WIDTH 150
#define LOWER_WIDTH 29
```

分別表示條碼的最大長度、條碼最大長度對應的資料最大長度、資料寬度、條碼長度上限(與條碼最大長度一致,用於提供程式語意上的方便)及條碼長度下限。

該專案中的執行流分為三大類，並且使用 PROCESS 陣列作為處理順序：

```
#define METHOD_STEP 13

void (*volatile const PROCESS[METHOD_STEP])(void);
```

第一類：處理類函式，主要的程式邏輯，將於後續闡述。

```
void get_length     (void);

void get_code       (void);

void sort_code      (void);

void take_format    (void);

void calibrate_code (void);

void binary_code    (void);

void reverse_code   (void);

void get_data       (void);

void empty_buffer   (void);
```

第二類：檢查類函式，判斷資料是否符合專案所要求。

```
void boundary_check (void);

void split_check    (void);

void C_check        (void);

void K_check        (void);
```

第三類：列印類函式，即輸出資料。

```
void print_code     (void);

void print_code_buf (void);

void print_data     (void);
```

該程式的基本架構使用狀態暫存器（以下稱為狀態流）並配合上述的三大類函式（以下稱為執行流）來實現掃描功能，如圖 2，並且執行流的順序已於 PROCESS 中定義：

```
void (*volatile const PROCESS[METHOD_STEP])(void) =
{
    get_length,     //先取得 m、n。
    boundary_check,    //後判斷 m。
    get_code,       //掃描輸入的條碼。
    sort_code,      //針對條碼值進行排序，以利後續處理。
    take_format,    //取得基準值。
    calibrate_code,    //根據基準值校正資料至無偏差的狀態。
    binary_code,    //將校正過的資料轉成二進制以利處理。
    reverse_code,   //針對資料的 START/STOP 進行判斷與反轉。
    split_check,    //判斷資料是否符合個別字元間的分割之格式。
    get_data,       //針對二進制資料建立資料模型並對其解碼。
    C_check,        //判斷解碼後的資料是否符合檢驗碼 C 的規範。
    K_check,        //判斷解碼後的資料是否符合檢驗碼 K 的規範。
    print_data      //上述程序執行完成無誤則直接輸出資料
};
```

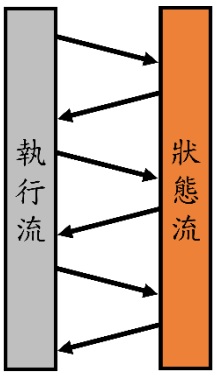

圖 2

由 PROCESS 的定義可知其作為是一個陣列，元素為唯讀（const）易變（volatile）指標（*），該指標指向某個函式，該函式的參數為空（void）且沒有回傳值（void）；主程式由執行流與狀態流共同形成，如下：

```
int main(void)
{
    while(1)
    {
        status_code = STATUS_EMPTY_FLAG;
        for(step = 0; step < METHOD_STEP; ++step)
        {
            PROCESS[step]();
            status = check_status();

            if(status == 1)
            {
                return 0;
            }
            if(status == 2)
            {
                break;
            }
        }
        empty_buffer();
    }
}
```

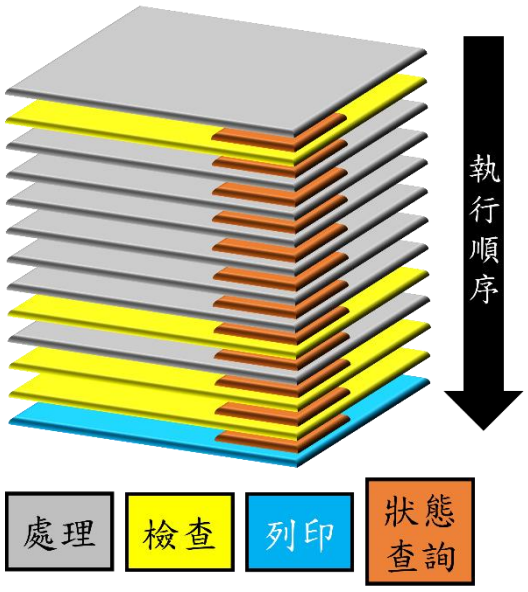

圖 3

如圖 3 所示，執行流中的每一個程序執行完成時，便會進行一次狀態查詢，二者交互執行，若遇到錯誤，則跳出當前執行流，並依據狀態旗標來界定要輸出的結果為何，抑或是直接跳出主程式；以下說明狀態旗標的運作方法：

```
static inline uint8_t check_status(void)
{
    switch(status_code)
    {
    case STATUS_EMPTY_FLAG:
        return 0;
    case STATUS_EXIT_FLAG:
        return 1;
    case STATUS_BAD_FLAG:
        printf("Case %d: bad code\n", count);
        return 2;
    case STATUS_BAD_C_FLAG:
        printf("Case %d: bad C\n", count);
        return 2;
    case STATUS_BAD_K_FLAG:
        printf("Case %d: bad K\n", count);
        return 2;
    default:
        return 0;
    }
}
```

若執行後的狀態仍為空旗標（STATUS_EMPTY_FLAG），則不作任何操作，每次重置執行流時預設狀態即為空旗標；若狀態為 STATUS_EXIT_FLAG，則回傳 1 令主程式直接結束；弱狀態為 STATUS_BAD_???，則輸出對應的錯誤訊息，並回傳 2 令主程式重置執行流。

以下個別介紹其他執行流所需的函式，此處依照類別的不同分成三類進行說明：

一、 列印類函式：

```
void print_code     (void);
void print_code_buf (void);
void print_data     (void);
```

以上三種皆用於輸出資料，根據資料的格式，print_code 可用於輸出任意時候的掃描之資料；print_code_buf 則多用於錯誤調適；print_data 可以直接用於最後解碼成功的資料輸出。

二、 檢查類函式：

```
void boundary_check (void);
void split_check    (void);
void C_check        (void);
void K_check        (void);
```

以上四種皆用於檢查資料，根據資料的要求判斷，boundary_check 判斷輸入的條碼數量是否符合條碼的規範；split_check 判斷條碼的分割條碼是否皆為窄的條碼（亮的條碼以於先前條件即被排除）；C_check 與 K_check 用於判斷資料對應的校正碼是否正確。

三、　處理類函式：以下僅探討該程式的算法核心，並逐個探討。

```
void take_format(void)
{
    for(each = 0; each < m - 1; ++each)
    {
        if((code_buf[each + 1] - code_buf[each]) >
        (bias_upper(code[each + 1]) - bias_lower(code[each + 1])))
            break;
    }

    min = ((code_buf[0] * 2 < code_buf[each + 1])?
              code_buf[0] * 2: code_buf[each + 1]);
    max = ( code_buf[each] * 2  > code_buf[m - 1])?
              code_buf[each] * 2:  code_buf[m - 1];

    wide_bar    = (min + max) / 2;
    narrow_bar  = (min + max) / 4;
}
```

上述程式用於找出窄條碼的基準值，並且先將條碼的數字由小至大排序，則當前條碼的下一個條碼之上下限之差大於可允許的範圍（即下一個條碼與當前條碼的差），則判斷到此結束，取出當前項的下一項*2 與最小的值進行平均即可得到最低的誤差值；並且當前的值*2 與最大的值進行平均可得到最高的誤差值。之後將繼續判斷下一個條碼的上下限之差是否大於可允許範圍，若大於其值，則表示資料具有三種不同的標準，即條碼的資料有誤，將狀態設為 bad。

```
void reverse_code(void)
{
    if(code[0] != code[m - 5] || code[1] != code[m - 4] ||
       code[2] != code[m - 3] || code[3] != code[m - 2] ||
       code[4] != code[m - 1]) {
        status_code = STATUS_BAD_FLAG;
        return;
    }
    if(!code[0] && !code[1] && code[2] && code[3] && !code[4])
        return;
    if(!code[0] && code[1] && code[2] && !code[3] && !code[4])
        for(i = 0, j = m-1; i < j; ++i, --j)
            SWAP(code[i], code[j]);
    return;
    status_code = STATUS_BAD_FLAG;
}
```

上述函式先判斷 START 與 STOP 是否相同，否則直接回傳並設為 bad code；再者，判斷 START/STOP 是否格式是對的，並判斷反序來決定是否要反轉；若上述條件都不符合，則表示格式錯誤，回傳 bad code。

```
void get_data(void)
{
    for(i = 1 + DATA_WIDTH, j = 0; i < m - DATA_WIDTH;
        i += DATA_WIDTH + 1, ++j)
        code_to_data(j, code + i);
    for(each = 0; each < n; ++each)
        if(!data_decode(each)) {
            status_code = STATUS_BAD_FLAG; return;
        }
    n -= 2;
}
```

上述程式即是將條碼對應的二進制之值進行加權，並根據加權值給出對應的解碼，若沒有解碼成功，則回傳 bad code；由於 n 包含了所有除了 START/STOP 的長度，故此處需要以 n-2 先排除 C 與 K 校驗碼。

對於此次專案的感想：

有關要求所敘述的基準值 ±5% 的算法，其具有多種語意，並且各種算法對於結果的影響於特殊情況下甚大，設計算法時，我也對此較為煩惱，這部份是我較為不解的；綜合而言，這次的專題給了我首次感受到程式開發的機會，對於總體專案的設計框架也有所了解，希望在學習完物件導向與軟體工程相關的課程後，再回來看本次的專案仍可以有很大的收穫。