# PRG155 期末專題：家庭保全系統 — 詳細解題報告

Home Security System（DeviceAlpha 開發板模擬）

繳交期限：2026 年 7 月 30 日實驗課開始前（需現場簡報） | 繳交檔案：`finalproject.c` | 配分：15 分

## 1 題目說明

本期末專題要求使用 **DeviceAlpha** 開發板撰寫一支 C 程式，模擬一套可運作的**家庭保全系統（Home Security System）**。專題整合按鈕（PB1–PB4）、LED 指示燈、蜂鳴器、溫度感測器與七段顯示器，展示真實嵌入式系統的核心原理。程式運作流程規定如下：

1. **啟動驗證**：程式開始時提示使用者輸入字元 `'Y'` 才能啟動，大小寫皆可（`'Y'` 或 `'y'`）。輸入其他字元時必須持續重新提示，直到收到有效回覆為止。
2. **初始化**：啟動後初始化開發板上所有元件，並立刻在**七段顯示器**顯示目前**攝氏溫度**，讓使用者觀察環境狀況；短暫延遲後清除顯示器，系統進入**待機（解除警戒 disarmed）**狀態，等待按鈕輸入。
3. **PB1 — 布防（Arm）**：按下 PB1 啟動布防程序。七段顯示器顯示 **3 秒倒數**，倒數期間**黃色 LED 閃爍**表示系統準備布防；倒數到 0 時**綠色 LED** 亮起，確認系統已布防（armed）、可偵測動作。
4. **PB2 — 動作偵測（Motion）**：系統在布防狀態下按 PB2 模擬「偵測到動作」事件，觸發**警報狀態**：**紅色 LED** 亮起、**蜂鳴器**響起表示有人入侵。警報持續作用，直到使用者解除警戒。
5. **PB3 — 解除警戒（Disarm）**：按 PB3 立即重置系統：關閉所有 LED、蜂鳴器靜音、清除七段顯示器，系統回到解除警戒狀態，可再次進行下一輪操作。（選做：可加入密碼序列解除警戒以增加複雜度。）
6. **PB4 — 離開程式（Exit）**：按 PB4 結束程式；但若系統在**布防中**或**警報作用中**，按 PB4 只會顯示提醒訊息，要求使用者先解除警戒。

**C 語言硬性要求（評分重點）**

程式必須示範以下 C 語言結構的使用：

- **自訂函式**（user-defined functions）
- **迴圈**（至少兩種不同型態，例如 `for` 與 `while`）
- **決策結構**（`if` 與 `switch` 敘述）
- 所有使用者輸入都必須做**輸入驗證**，確保程式運作可靠

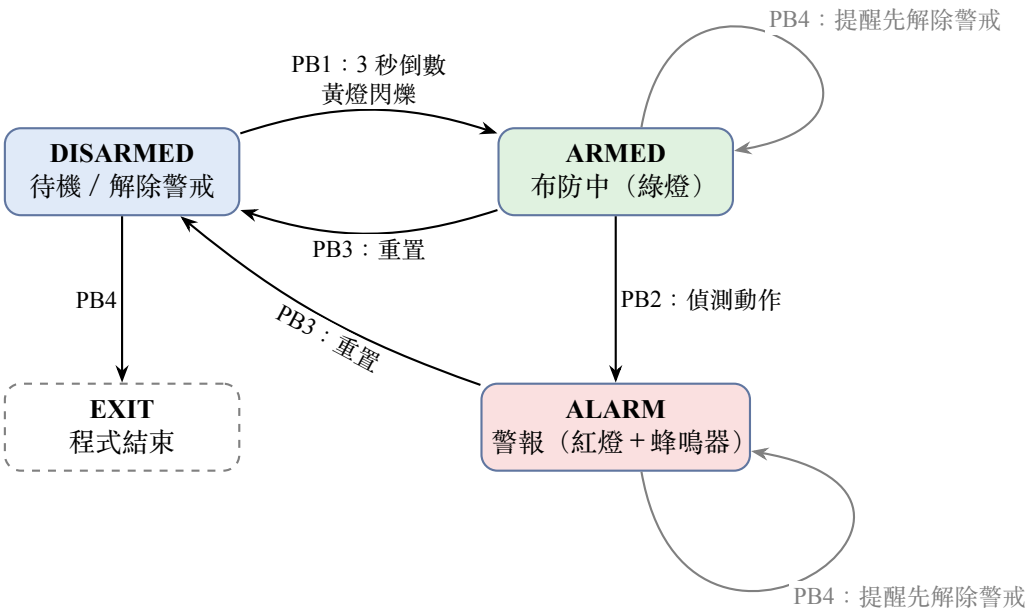
**注意**：需現場簡報並回答關於程式碼的問題；若無法解釋自己的程式碼、展現對其功能的完整理解，專題成績為零分。檔名必須是 `finalproject.c`（其他副檔名一律零分）。

## 2 問題分析

### 2.1 核心觀念：有限狀態機（Finite State Machine）

保全系統本質上是一個**有限狀態機**：系統任一時刻只會處於三種狀態之一，按鈕事件觸發狀態轉移。這正是題目要求 `switch` 敘述的最佳應用場景——用 `switch (currentState)` 分派各狀

態的行為。




## 2.2 硬體裝置與其角色

| 裝置 | 作用 | 何時啟動 | 對應狀態 |
|---|---|---|---|
| 黃色 LED | 布防倒數指示（閃爍） | PB1 按下後的 3 秒倒數 | 轉移期間 |
| 綠色 LED | 已布防指示 | 倒數結束 | ARMED |
| 紅色 LED | 入侵警報指示 | 布防中按 PB2 | ALARM |
| 蜂鳴器 | 入侵警報聲響 | 與紅色 LED 同時 | ALARM |
| 七段顯示器 | 顯示環境溫度、倒數數字 | 初始化、布防倒數 | — |
| 溫度感測器 | 顯示啟動時的環境溫度 | 系統初始化 | — |
| PB1–PB4 | 布防 / 觸發 / 解除 / 離開 | 主迴圈輪詢 | 事件來源 |

## 2.3 關鍵設計決策

1. **主迴圈＋輪詢（polling）架構**：嵌入式系統的典型寫法。`while (systemRunning)` 主迴圈裡不斷用 `GetPushButtonState()` 輪詢按鈕，再交給 `switch` 依目前狀態決定反應。每輪加 `Wait(50)` 作為簡單的防彈跳（debounce）與降低輪詢頻率。
2. **狀態常數用 `#define` 命名**：`STATE_DISARMED`、`STATE_ARMED`、`STATE_ALARM`，可讀性遠勝魔術數字，簡報答詢時也更容易說明。
3. **「一次按壓只處理一次」**：偵測到 `PRESSED` 後先呼叫 `waitForButtonRelease()` 等按鈕放開，否則同一次按壓會在下一輪迴圈被重複讀到（例如 PB1 按一次卻布防兩次）。

4. 啟動輸入驗證：`getchar()` 讀入第一個字元後，把該行剩餘字元全部讀掉（清空輸入緩衝區），再判斷是否為 `'Y'` / `'y'`。這樣輸入 `"Yes"`、`"yes"` 也能正確啟動，輸入 `"no"`、數字、空白行則重新提示。
5. **PB4 的條件式行為**：同一顆按鈕在不同狀態有不同意義——DISARMED 時離開程式；ARMED / ALARM 時只印提醒訊息。這由狀態機自然實現：每個 `case` 裡對 PB4 寫不同的處理。
6. **五個自訂函式分工**：`waitForStartCommand`（啟動驗證）、`initializeSystem`（初始化 + 顯示溫度）、`waitForButtonRelease`（等待放開）、`runArmingCountdown`（倒數 + 黃燈閃爍）、`activateAlarm` 與 `resetSystem`（警報開 / 全部重置）。`main` 只負責狀態機骨架，結構清晰。

**如何同時滿足所有 C 語言要求**

- **自訂函式**：共 6 個（含帶參數的 `waitForButtonRelease(int button)`）。
- **兩種迴圈**：`while`（主迴圈、輸入驗證、等待放開）與 `for`（3 秒倒數、黃燈閃爍的巢狀迴圈）。
- **if 與 switch**：`switch (currentState)` 分派狀態，各 `case` 內用 `if / else if` 判斷按鈕。
- **輸入驗證**：啟動字元只接受 `'Y'` / `'y'`，其餘輸入清空緩衝區後重新提示。

# 3 逐步解題過程

## 3.1 步驟 1：啟動驗證函式 `waitForStartCommand`

用 `while` 迴圈持續提示，直到輸入 `'Y'` 或 `'y'`。重點是**清空輸入緩衝區**：使用者可能輸入一整行（如 `"yes"` 或 `"abc"`），只取第一個字元、其餘讀掉，才不會影響下一輪判斷：

```
while (!validStart)
{
    printf("Enter 'Y' to start the Home Security System: ");
    startCharacter = (char)getchar();

    if (startCharacter != '\n')             /* 清掉該行剩餘字元 */
    {
        extraCharacter = getchar();
        while (extraCharacter != '\n' && extraCharacter != EOF)
            extraCharacter = getchar();
    }

    if (startCharacter == 'Y' || startCharacter == 'y')
        validStart = 1;
    else
        printf("Invalid input! You must enter 'Y' or 'y' to start.\n");
}
```

## 3.2 步驟 2：初始化函式 `initializeSystem`

關閉所有 LED 與蜂鳴器（確保起始狀態乾淨），讀取溫度並以 2 位小數顯示在七段顯示器；停留 2 秒讓使用者觀察後清除顯示器，進入待機狀態：

```
SetLEDState(YELLOW_LED, OFF);
SetLEDState(GREEN_LED, OFF);
SetLEDState(RED_LED, OFF);
SetBuzzerState(BUZZER, OFF);

currentTemperature = GetTemperature(TEMPERATURE_SENSOR);
printf("Current temperature: %.2f degrees Celsius\n", currentTemperature);
DisplayDoubleValue(currentTemperature, 2);
Wait(2000);
ClearSevenSegmentDisplay();
```

### 3.3 步驟 3：主迴圈與狀態機

`main` 的核心是 `while` 主迴圈包住 `switch` 狀態機。以 DISARMED 狀態為例：

```
case STATE_DISARMED:
    if (GetPushButtonState(PB1) == PRESSED)
    {
        waitForButtonRelease(PB1);       /* 一次按壓只處理一次 */
        runArmingCountdown();            /* 3 秒倒數 + 黃燈閃爍 */
        currentState = STATE_ARMED;
    }
    else if (GetPushButtonState(PB4) == PRESSED)
    {
        waitForButtonRelease(PB4);
        printf("Exiting the Home Security System. Goodbye!\n");
        systemRunning = 0;               /* 結束主迴圈 */
    }
    break;
```

ARMED 狀態多了 PB2（觸發警報）與 PB3（解除）的判斷，且 PB4 改為印出提醒；ALARM 狀態只接受 PB3（解除）與 PB4（提醒），完整程式碼見第 4 節。

### 3.4 步驟 4：布防倒數 `runArmingCountdown`（for 迴圈）

外層 `for` 從 3 倒數到 1，每一秒內層 `for` 讓黃燈閃爍兩次（4 段 250 ms 恰好等於 1 秒），同時七段顯示器顯示目前秒數；倒數結束顯示 0、點亮綠燈：

```
for (countdownSecond = 3; countdownSecond >= 1; countdownSecond--)
{
    DisplayIntValue(countdownSecond);
    for (blinkNumber = 0; blinkNumber < 2; blinkNumber++)
    {
        SetLEDState(YELLOW_LED, ON);
        Wait(250);
        SetLEDState(YELLOW_LED, OFF);
        Wait(250);
    }
}
DisplayIntValue(0);
SetLEDState(GREEN_LED, ON);              /* 系統已布防 */
```

### 3.5 步驟 5：警報與重置

`activateAlarm()`：綠燈滅、紅燈亮、蜂鳴器響——警報會一直持續，因為狀態機停在`STATE_ALARM`，只有 PB3 能離開這個狀態。
`resetSystem()`：關閉所有 LED、蜂鳴器靜音、清除七段顯示器，回到 DISARMED。兩個轉移（ARMED → DISARMED、ALARM → DISARMED）共用同一個函式，避免重複程式碼。

## 4 完整程式碼（finalproject.c）

```c
/*
program: finalproject.c
author: firstName lastName
date: July 30 2026
purpose: Home Security System simulation using the DeviceAlpha board
*/

#include <stdio.h>
#include "deviceAlpha.h"

/* system states */
#define STATE_DISARMED  0
#define STATE_ARMED 1
#define STATE_ALARM 2

/* user-defined function prototypes */
void waitForStartCommand(void);
void initializeSystem(void);
void waitForButtonRelease(int button);
void runArmingCountdown(void);
void activateAlarm(void);
void resetSystem(void);

int main(void)
{
    int currentState = STATE_DISARMED;  /* current state of the system */
    int systemRunning = 1;          /* 1 = keep running, 0 = exit */

    /* step 1: validated start prompt (only 'Y' or 'y' is accepted) */
    waitForStartCommand();

    /* step 2: initialize the board and show the ambient temperature */
    initializeSystem();

    /* step 3: main state machine loop */
    while (systemRunning)
    {
        switch (currentState)
        {
            case STATE_DISARMED:
                if (GetPushButtonState(PB1) == PRESSED)
                {
                    /* start the arming sequence */
                    waitForButtonRelease(PB1);
                    runArmingCountdown();
                    currentState = STATE_ARMED;
                }
```

```
48                  else if (GetPushButtonState(PB4) == PRESSED)
49                  {
50                      /* exit is only allowed while disarmed */
51                      waitForButtonRelease(PB4);
52                      printf("Exiting the Home Security System. Goodbye!\n");
53                      systemRunning = 0;
54                  }
55                  break;
56
57              case STATE_ARMED:
58                  if (GetPushButtonState(PB2) == PRESSED)
59                  {
60                      /* simulated motion detection -> alarm */
61                      waitForButtonRelease(PB2);
62                      printf("Motion detected! ALARM activated!\n");
63                      activateAlarm();
64                      currentState = STATE_ALARM;
65                  }
66                  else if (GetPushButtonState(PB3) == PRESSED)
67                  {
68                      /* disarm before any motion was detected */
69                      waitForButtonRelease(PB3);
70                      resetSystem();
71                      currentState = STATE_DISARMED;
72                  }
73                  else if (GetPushButtonState(PB4) == PRESSED)
74                  {
75                      /* cannot exit while the system is armed */
76                      waitForButtonRelease(PB4);
77                      printf("System is ARMED! Please disarm the system (PB3)
                            before exiting.\n");
78                  }
79                  break;
80
81              case STATE_ALARM:
82                  if (GetPushButtonState(PB3) == PRESSED)
83                  {
84                      /* disarm stops the alarm and resets everything */
85                      waitForButtonRelease(PB3);
86                      resetSystem();
87                      currentState = STATE_DISARMED;
88                  }
89                  else if (GetPushButtonState(PB4) == PRESSED)
90                  {
91                      /* cannot exit while the alarm is active */
92                      waitForButtonRelease(PB4);
93                      printf("ALARM is active! Please disarm the system (PB3)
                            before exiting.\n");
94                  }
95                  break;
96          }
97
98          Wait(50);   /* small pause between button polls (debounce) */
99      }
100
101     return 0;
102 }
103
```

```c
/*
 * waitForStartCommand
 * Keeps prompting until the user enters 'Y' or 'y'.
 * Any other input (wrong letter, number, empty line) is rejected.
 */
void waitForStartCommand(void)
{
    char startCharacter = '\0';
    int extraCharacter;
    int validStart = 0;

    while (!validStart)
    {
        printf("Enter 'Y' to start the Home Security System: ");
        startCharacter = (char)getchar();

        /* clear the rest of the input line */
        if (startCharacter != '\n')
        {
            extraCharacter = getchar();
            while (extraCharacter != '\n' && extraCharacter != EOF)
            {
                extraCharacter = getchar();
            }
        }

        if (startCharacter == 'Y' || startCharacter == 'y')
        {
            validStart = 1;
        }
        else
        {
            printf("Invalid input! You must enter 'Y' or 'y' to start.\n");
        }
    }
}

/*
 * initializeSystem
 * Turns every component off, shows the current temperature on the
 * 7-Segment display for a short time, then clears the display and
 * reports that the system is in the idle (disarmed) state.
 */
void initializeSystem(void)
{
    double currentTemperature;

    SetLEDState(YELLOW_LED, OFF);
    SetLEDState(GREEN_LED, OFF);
    SetLEDState(RED_LED, OFF);
    SetBuzzerState(BUZZER, OFF);

    currentTemperature = GetTemperature(TEMPERATURE_SENSOR);
    printf("Current temperature: %.2f degrees Celsius\n", currentTemperature);
    DisplayDoubleValue(currentTemperature, 2);

    Wait(2000); /* let the user observe the ambient temperature */
    ClearSevenSegmentDisplay();
```

```
162
163     printf("System is DISARMED. PB1 = arm, PB4 = exit.\n");
164 }
165
166 /*
167  * waitForButtonRelease
168  * Busy-waits until the given push button is released, so one physical
169  * press is handled exactly once.
170  */
171 void waitForButtonRelease(int button)
172 {
173     while (GetPushButtonState(button) == PRESSED)
174     {
175         ;   /* wait for the release */
176     }
177 }
178
179 /*
180  * runArmingCountdown
181  * Shows a 3 second countdown on the 7-Segment display while blinking
182  * the YELLOW led, then turns the GREEN led on (system armed).
183  */
184 void runArmingCountdown(void)
185 {
186     int countdownSecond;
187     int blinkNumber;
188
189     printf("Arming the system...\n");
190
191     for (countdownSecond = 3; countdownSecond >= 1; countdownSecond--)
192     {
193         DisplayIntValue(countdownSecond);
194
195         /* two yellow blinks per second (4 x 250 ms = 1 second) */
196         for (blinkNumber = 0; blinkNumber < 2; blinkNumber++)
197         {
198             SetLEDState(YELLOW_LED, ON);
199             Wait(250);
200             SetLEDState(YELLOW_LED, OFF);
201             Wait(250);
202         }
203     }
204
205     DisplayIntValue(0);
206     SetLEDState(GREEN_LED, ON);
207     Wait(500);
208     ClearSevenSegmentDisplay();
209
210     printf("System is ARMED. PB2 = motion event, PB3 = disarm.\n");
211 }
212
213 /*
214  * activateAlarm
215  * Intrusion detected: RED led on and BUZZER on. The alarm stays active
216  * until the user disarms the system with PB3.
217  */
218 void activateAlarm(void)
219 {
```

```
    SetLEDState(GREEN_LED, OFF);
    SetLEDState(RED_LED, ON);
    SetBuzzerState(BUZZER, ON);
}

/*
 * resetSystem
 * Disarms the system: all LEDs off, buzzer silenced, display cleared.
 * The system returns to the idle (disarmed) state.
 */
void resetSystem(void)
{
    SetLEDState(YELLOW_LED, OFF);
    SetLEDState(GREEN_LED, OFF);
    SetLEDState(RED_LED, OFF);
    SetBuzzerState(BUZZER, OFF);
    ClearSevenSegmentDisplay();

    printf("System DISARMED. PB1 = arm, PB4 = exit.\n");
}
```

Listing 1: finalproject.c 完整原始碼

# 5 如何編譯與執行

## 5.1 在學校的 DeviceAlpha 環境（正式繳交）

1. 在實驗室電腦開啟 DeviceAlpha 開發環境，建立專案並加入 `finalproject.c`，確認專案包含課程提供的官方 `deviceAlpha.h`（本解答附的模擬用標頭檔不要一起繳交）。
2. 若官方函式庫中「清除七段顯示器」的函式名稱與 `ClearSevenSegmentDisplay()` 不同，請改成官方名稱（僅需修改該一處呼叫；若無清除函式，也可以顯示空白或 0 代替）。
3. 編譯執行後依序測試：
   (a) 啟動驗證：先輸入 `n`、`5`、直接按 Enter，確認都被拒絕；輸入 `y` 或 `Y` 成功啟動。
   (b) 觀察七段顯示器顯示目前溫度約 2 秒後清除。
   (c) 按 PB1：顯示器倒數 3、2、1，黃燈閃爍，然後綠燈亮（已布防）。
   (d) 按 PB4：應顯示「請先解除警戒」提醒而不會結束。
   (e) 按 PB2：紅燈亮、蜂鳴器響（警報）；此時按 PB4 同樣只有提醒。
   (f) 按 PB3：全部關閉、回到待機；最後在待機狀態按 PB4 正常結束程式。
4. 確認無誤後，把檔頭註解的姓名與日期改成自己的，將 `finalproject.c` 上傳到 Blackboard 的 Assignments 資料夾，並準備現場簡報（見第 7 節）。

## 5.2 在一般電腦上測試（使用附帶的模擬標頭檔）

沒有實驗板時，可用本解答附的 `deviceAlpha.h` 模擬器。它把 LED / 蜂鳴器動作印成 `[SIM]` 訊息、七段顯示器印成 `[7SEG]` 訊息；按鈕用鍵盤模擬——當程式輪詢按鈕而沒有待處理事件時，模擬器會等待你輸入一行文字：輸入 `1`～`4` 再按 Enter，即代表按下並放開 PB1～PB4。

```
$ gcc -Wall -Wextra -std=c99 -o finalproject finalproject.c
$ ./finalproject
```

### 5.3 實際執行範例（模擬器輸出節錄）

測試腳本輸入依序為：x（無效）、7（無效）、y（啟動）、1（布防）、4（被拒）、2（警報）、4（被拒）、3（解除）、4（離開）：

```
Enter 'Y' to start the Home Security System: Invalid input! ...
Enter 'Y' to start the Home Security System: Invalid input! ...
Enter 'Y' to start the Home Security System: y
Current temperature: 22.18 degrees Celsius
        [7SEG] 22.18
        [7SEG] (cleared)
System is DISARMED. PB1 = arm, PB4 = exit.
        [SIM] PB1 pressed
Arming the system...
        [7SEG] 3
        [SIM] YELLOW LED ON  ... (blink) ...
        [7SEG] 2  ...  [7SEG] 1  ...
        [7SEG] 0
        [SIM] GREEN LED ON
System is ARMED. PB2 = motion event, PB3 = disarm.
        [SIM] PB4 pressed
System is ARMED! Please disarm the system (PB3) before exiting.
        [SIM] PB2 pressed
Motion detected! ALARM activated!
        [SIM] RED LED ON
        [SIM] BUZZER ON
        [SIM] PB4 pressed
ALARM is active! Please disarm the system (PB3) before exiting.
        [SIM] PB3 pressed
        [SIM] BUZZER OFF
System DISARMED. PB1 = arm, PB4 = exit.
        [SIM] PB4 pressed
Exiting the Home Security System. Goodbye!
```

## 6 常見錯誤與評分要點

1. **沒有等按鈕放開**：偵測到 PRESSED 後立刻處理、不等放開，同一次按壓會被重複讀取多次（最常見的 bug）。務必在每次處理前呼叫 `waitForButtonRelease()`。
2. **啟動驗證不完整**：只判斷 `'Y'` 漏掉 `'y'`，或沒清空輸入緩衝區導致輸入 `"yes"` 之後的字元干擾下一輪讀取。
3. **PB4 無條件結束程式**：題目明確要求布防或警報中按 PB4 只能顯示提醒訊息，直接結束會被扣分。
4. **缺少必要的語言結構**：忘了用 `switch`、或整支程式只用一種迴圈，違反硬性規定。本解法中 `switch` 用於狀態分派、`for` 用於倒數、`while` 用於主迴圈與驗證，天然覆蓋所有要求。
5. **警報自動解除**：警報必須持續到使用者按 PB3，不可用計時自動關閉。
6. **檔名錯誤**：必須是 `finalproject.c`，其他副檔名（`.cpp`、`.txt`）一律零分。
7. **結構與可讀性**：Tab 縮排、描述性 camelCase 命名（`currentState`、`countdownSecond`）、函式各司其職——這些都直接影響評分與簡報表現。

# 7 現場簡報答詢準備（必讀）

本專題必須現場解釋程式碼，無法解釋即為零分。以下是最可能被問到的問題與建議回答方向：

1. 「為什麼用 **switch** 而不是全部用 **if** ?」——系統是有限狀態機，`switch (currentState)` 讓「目前狀態決定行為」的結構一目了然；各狀態內的按鈕判斷才用 `if / else if`。
2. 「**waitForButtonRelease** 的作用是什麼？拿掉會怎樣？」——確保一次實體按壓只被處理一次；拿掉後主迴圈每 50 ms 輪詢一次，同一次按壓會觸發多次動作。
3. 「**倒數的 1 秒是怎麼組成的** ?」——內層 `for` 執行兩次「亮 250 ms + 滅 250 ms」，共 $4 \times 250 = 1000$ ms，同時完成閃爍與計時。
4. 「**輸入 yes 為什麼能啟動？輸入空白行為什麼不會出錯** ?」——只取第一個字元判斷，其餘字元被清緩衝區的 `while` 讀掉；空白行的第一個字元是 `'\n'`，不等於 `'Y'` / `'y'`，直接重新提示。
5. 「**警報狀態下按 PB1 會發生什麼** ?」——什麼都不會發生：`STATE_ALARM` 的 `case` 裡沒有 PB1 的判斷，事件被忽略，這正是狀態機的好處。
6. 「**如何加入選做的密碼解除功能** ?」——把 `STATE_ALARM` 中 PB3 的處理改成要求依序按出特定按鈕序列（例如 PB1-PB2-PB1），用陣列記錄輸入並比對，全對才呼叫 `resetSystem()`。