EE1003 Introduction to Computer I

Programming Assignment 2
X-Linked Rings Puzzle

授課教師：陳聿廣教授

學生：王佑恩

學號：108601205

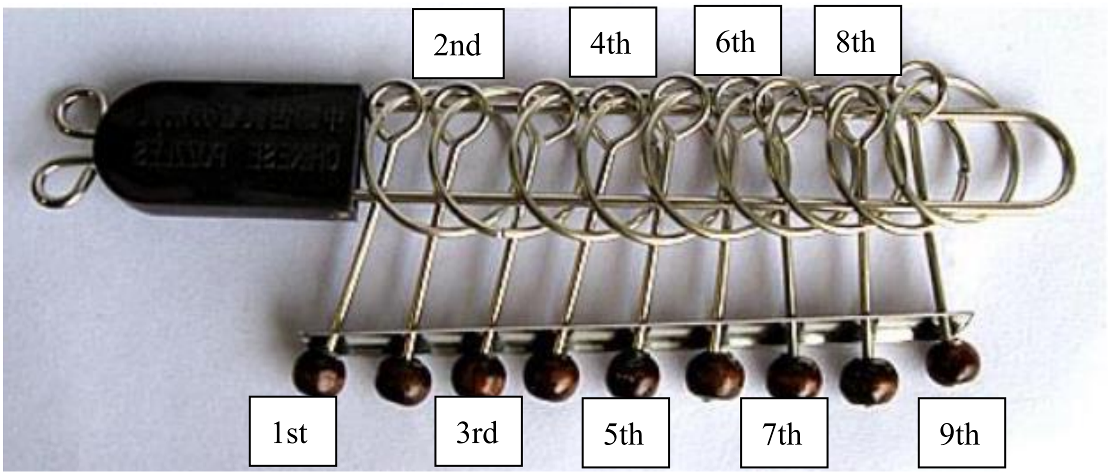
系級：電機三 A

2022.11.16

一、 題目簡介

九連環是一款著名的兒童益智遊戲，很多人小時候花了很多心血去解決它。此次 PA 透過對九連環解法的解析，找出解開九連環的規則，並將其設計為一套程式。

此程式能根據使用者輸入的環數以及連環的狀態（1 為未解開，環仍在劍上；0 為已解開，環跟劍已分開），顯示出解開所有連環的步驟，引導使用者一步步解開連環。



圖一、九連環與本程式對環的序數

二、 程式敘述

1. 程式運作分析

由 Reference[1]得知，解開多連環的過程中，所需要的兩個規則是 R-rule 與 S-rule。

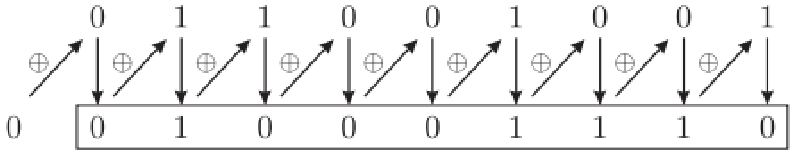
R-rule
- Turn the ring down：當最右邊的環（離劍柄最遠的環）為1，則將其轉換為0
- Turn the ring on：當最右邊的環（離劍柄最遠的環）為0，則將其轉換為1

S-rule
- Turn the ring down ：從最外側的環往內，當遇到兩個連續的1時，將較內側的1轉為0
- Turn the ring on：從最外側的環往內，當遇到01並排（0在靠內側，1在靠外側）時，將較內側的0轉為1

在解環的過程中，需要從最外層的環（離劍柄最遠的環）開始解起。首先先根據使用者輸入之環的狀態，組合成一串只包含 0 與 1 的一維陣列。

為了方便描述，在本篇報告中以"S0"表示上述一維陣列，例如當使用者輸入之狀態的陣列為{1, 0, 1, 0, 1, 1,…}，則 $S0 = 101011\ldots_2$，由左往右的數字編號為 1th, 2nd, 3rd, 4th, …。

由 Reference[1]得知，對於任意狀態 x，恰需要 f(x)步才能解出。設狀態 S0 = 011001001_2，f(x)計算方式如下：



以 0 對狀態 S0 的 1th 數字做 XOR，接著將所得的值對 2rd 數字做 XOR，重複步驟依序往下做，即得 f(S0) = 010001110_2，解開狀態 S0（即 S0 = 000000000_2）所需的步數為 142 步。

解環時，若解環的操作不符合規律，則無法將環順利解開，我們可以從 f(S0)為奇數或偶數，判斷解環時需先操作 R-rule 還是 S-rule。當 f(S0)為奇數時，需先操作 R-rule；為偶數時，需先操作 S-rule。

最後交替操作 R-rule 與 S-rule，即可將環全數解開。

2. Pseudocode

```
Function Declaration:  決定此程式所需的所有 function
Prompt the user to enter how many X-Linked Ring want to solve
Prompt the user to enter the rings state from inside to outside

透過某個 function 來將使用者輸入的 state 存成 array
透過某個 function 顯示對 X-Linked Ring 操作一次 R-rule
透過某個 function 顯示對 X-Linked Ring 操作一次 S-rule

透過某個 function 開始解環
    判斷要先做 R-rule 還是 S-rule
    依照 R-S-R-S-...或是 S-R-S-R-...的順序進行解環
    顯示所需的步驟並離開程式
```

3. 程式參數簡介

根據題意，我們需要宣告多個參數：

Parameter	Data Type	Meaning	Range
rings	int	使用者輸入的環數	0 ~ 2^32-1
i	int	作為計數用的變數，初始值為 1	1 ~ rings
steps	int	統計解開環所需的步驟數	1~2^31-1
total	int	計算所有環狀態總和的數值，用以判斷是否將環全數解開	0 ~ 2^31-1

arraySize	const int	依照使用者輸入的環數，限制各個 array 的大小	rings
state_all	int []	存入使用者輸入之各個環的狀態	0, 1
f	int []	Reference[1]所提到的 Finite Automaton 函數 f	0, 1
run_R_rule_once_array	int []	運作一次 R-rule 後各環的狀態	0, 1
run_S_rule_once_array	int []	運作一次 S-rule 後各環的狀態	0, 1

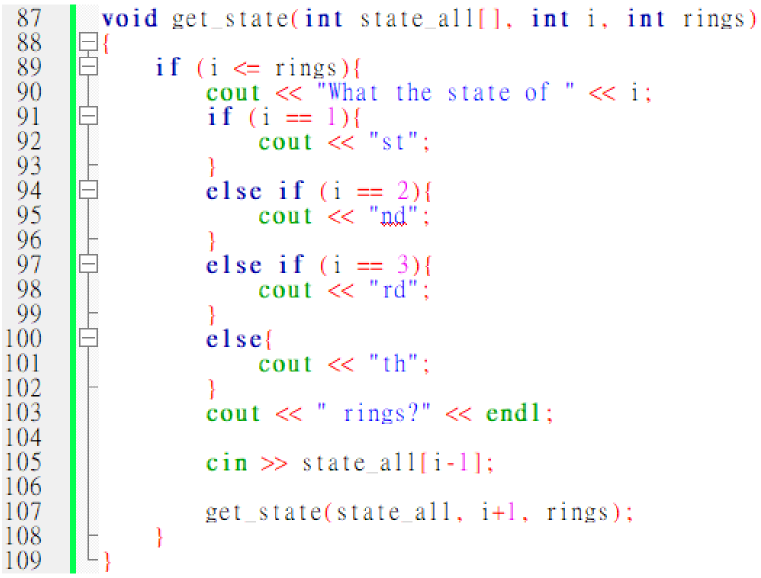
4. 程式內 function 簡介

依照所發想的 Pseudocode，我們需要做出多個 function：

Fuction	Meaning	Prototype
get_state	將使用者輸入的環數存入 array state_all	void get_state(int [], int, int)
show_state	顯示全部環的狀態	void show_state(int [], int, int)
array_assign	因在程式中會多次使用 array state_all，為了避免在程式中更改到最初環的 state，用此 recursion 進行 array 的 assign	void array_assign(int [], int[], int, int)
do_XOR	根據 Reference[1]為了找出 Finite Automaton 函數 f 所做的 XOR，為了提高程式的 Readability，而將 XOR 的過程另設為一個 function	int do_XOR(int, int, int)
Finite_Automation	根據 Reference[1]找出 Finite Automaton 函數	void Finite_Automation(int [], int, int)
run_R_rule_once	運作一次 R-rule	void run_R_rule_once(int [], int)
R_rule	在開始解環時，所使用到的 R-rule	void R_rule(int [], int)
run_S_rule_once	運作一次 S-rule	void run_S_rule_once(int [], int)
S_rule	在開始解環時，所使用到的 S-rule	void S_rule(int [], int)
success_solve	判斷是否將環全數解開	int success_solve(int [], int, int, int)
start_solve	開始解環所用的 recursion	void start_solve(int [], int [], int, int, int, int)

5. 程式 Function 細節

(1) get_state

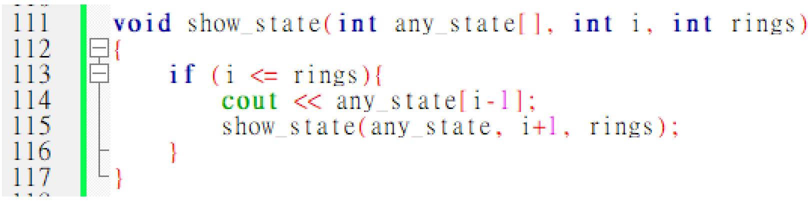


```
void get_state(int state_all[], int i, int rings)
{
    if (i <= rings){
        cout << "What the state of " << i;
        if (i == 1){
            cout << "st";
        }
        else if (i == 2){
            cout << "nd";
        }
        else if (i == 3){
            cout << "rd";
        }
        else{
            cout << "th";
        }
        cout << " rings?" << endl;

        cin >> state_all[i-1];

        get_state(state_all, i+1, rings);
    }
}
```

(a) 根據使用者輸入的環數（變數 rings）不停輸出：「What the state of nth rings?」。使用者輸入的 state 會依序被存入 state_all 這個 array，形成 state_all[rings] = {state 1st, state 2nd, state 3rd, state 4th, …}。直到 i > rings 時停止，使陣列裡數值的個數等於環數 rings。

(b) 因計數變數 i 的初始值為 1，而 array 的數值從 0 開始編號，當使用者輸入第 2 個環狀態時，會將狀態存入 state_all[1]，即使用者輸入的第 i 個 state 會存進 state_all[i-1]。

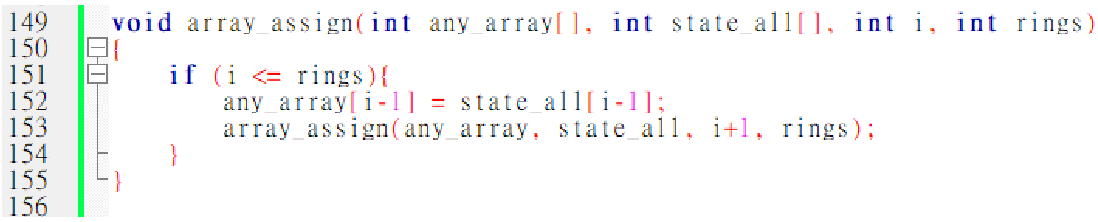
(2) show_state



```
void show_state(int any_state[], int i, int rings)
{
    if (i <= rings){
        cout << any_state[i-1];
        show_state(any_state, i+1, rings);
    }
}
```

(a) 將任意 array 內的值依序印出，若 any_state[3] = {1, 0, 1}，即印出 101。

(3) array_assign



```
void array_assign(int any_array[], int state_all[], int i, int rings)
{
    if (i <= rings){
        any_array[i-1] = state_all[i-1];
        array_assign(any_array, state_all, i+1, rings);
    }
}
```

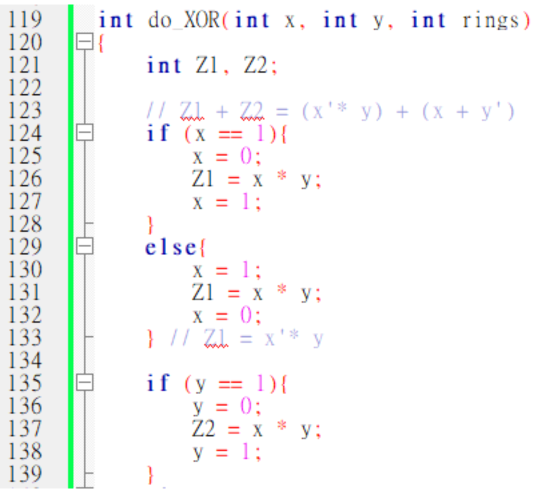
(a) 目的：因為在後續的程式中會多次用到 array state_all，且用途皆有差異，為了避免更動到 array state_all，需要將其 assign 給其他 array。

(b) 將任意 array 內的值依序 assign 到另一個 array，即複製一個不同名稱但內容值相同的 array。

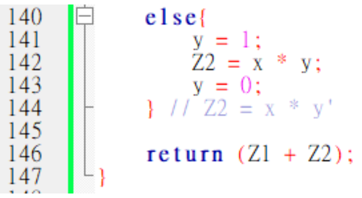
(c) 本程式中所執行的 array assign：

array	**assign to**
state_all	f
	run_R_rule_once_array
	run_S_rule_once_array

(4) do_XOR



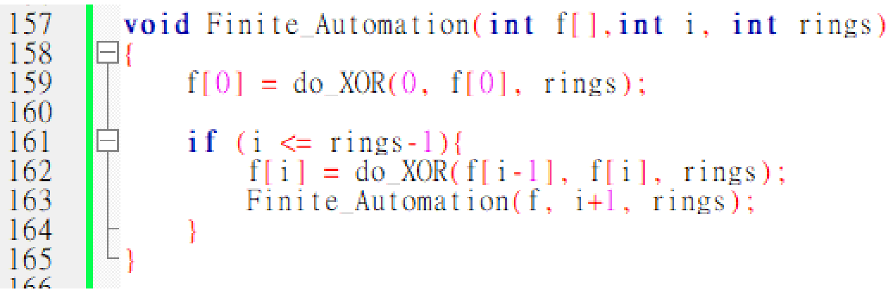
```
int do_XOR(int x, int y, int rings)
{
    int Z1, Z2;

    // Z1 + Z2 = (x'* y) + (x + y')
    if (x == 1){
        x = 0;
        Z1 = x * y;
        x = 1;
    }
    else{
        x = 1;
        Z1 = x * y;
        x = 0;
    } // Z1 = x'* y

    if (y == 1){
        y = 0;
        Z2 = x * y;
        y = 1;
    }
```



```
    else{
        y = 1;
        Z2 = x * y;
        y = 0;
    } // Z2 = x * y'

    return (Z1 + Z2);
}
```

(a) $x \oplus y = (x' * y) + (x * y') = Z1 + Z2$

(5) Finite_Automation



```
void Finite_Automation(int f[],int i, int rings)
{
    f[0] = do_XOR(0, f[0], rings);

    if (i <= rings-1){
        f[i] = do_XOR(f[i-1], f[i], rings);
        Finite_Automation(f, i+1, rings);
    }
}
```

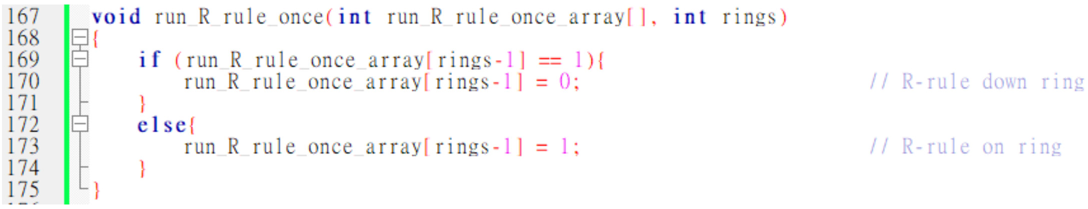
(a) 目的：成功獲取處理好的 array f，表示已經獲得 f(S0)，可用於判斷解環的第一步驟。

(b) argument f[]為透過 function array_assign 將 array state_all assign 給 array f 所形成的 array f。

(c) 因為要先以 0 對狀態 S0 的 state 1st 做 XOR，所以將 0 與 f[0]放進 function do_XOR 內，並將回傳的值 assign 給 f[0]。

(d) 相繼將 f[i-1]與 f[i]當作 function do_XOR 的 argument 並將回傳值 assign 給 f[i]，即可獲得處理好的 array f。

(6) run_R_rule_once



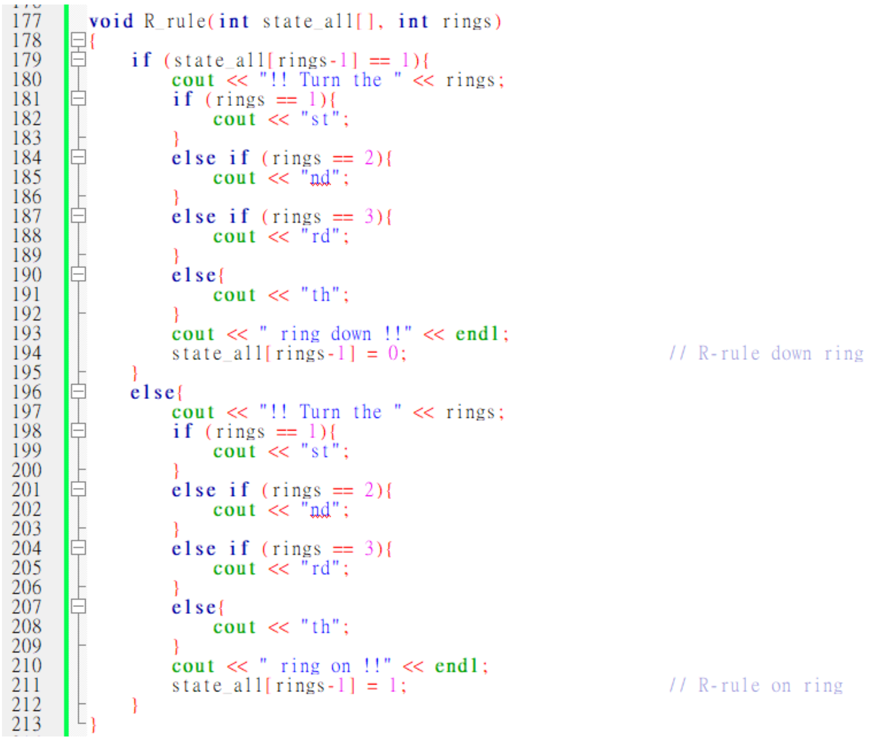
```
void run_R_rule_once(int run_R_rule_once_array[], int rings)
{
    if (run_R_rule_once_array[rings-1] == 1){
        run_R_rule_once_array[rings-1] = 0;                     // R-rule down ring
    }
    else{
        run_R_rule_once_array[rings-1] = 1;                     // R-rule on ring
    }
}
```

(a) 目的：此 function 是為了在執行時，透過 function show_state 印出執行一次 R-rule 的結果，所以特地製作一個 array 和 function 來達成此目的。

(b) argument run_R_rule_once_array[]為透過 function array_assign 將 array state_all assign 給 array run_R_rule_once_array 所形成的 array run_R_rule_once_array。

(c) rings-1 指的是 run_R_rule_once_array 中最右邊的 element，依判斷式將其值做一次 R-rule，並將結果存成新的 run_R_rule_once_array。

(7) R_rule

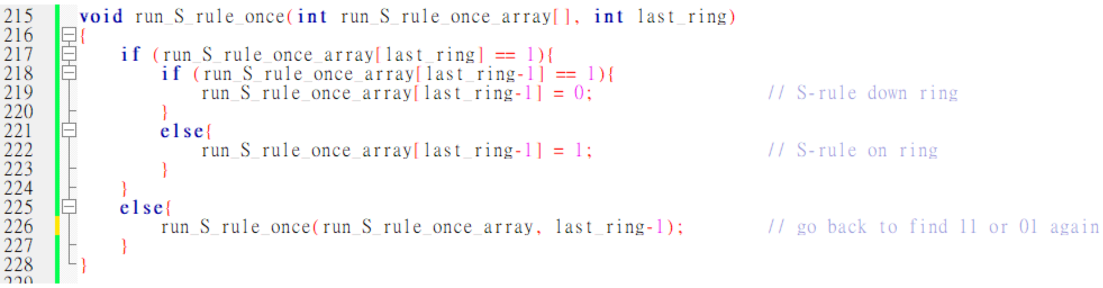


```
void R_rule(int state_all[], int rings)
{
    if (state_all[rings-1] == 1){
        cout << "!! Turn the " << rings;
        if (rings == 1){
            cout << "st";
        }
        else if (rings == 2){
            cout << "nd";
        }
        else if (rings == 3){
            cout << "rd";
        }
        else{
            cout << "th";
        }
        cout << " ring down !!" << endl;
        state_all[rings-1] = 0;                     // R-rule down ring
    }
    else{
        cout << "!! Turn the " << rings;
        if (rings == 1){
            cout << "st";
        }
        else if (rings == 2){
            cout << "nd";
        }
        else if (rings == 3){
            cout << "rd";
        }
        else{
            cout << "th";
        }
        cout << " ring on !!" << endl;
        state_all[rings-1] = 1;                     // R-rule on ring
    }
}
```

(a) 解環時所使用的 R-rule，兼具顯示步驟以及更動 array state_all 的功能。

(b) state_all[ring-1]為 array state_all 最右邊的 element，按照程式運作分析（二、1）所示之 R-rule 規則，執行 turn down/on the ring。

(8) run_S_rule_once



```
void run_S_rule_once(int run_S_rule_once_array[], int last_ring)
{
    if (run_S_rule_once_array[last_ring] == 1){
        if (run_S_rule_once_array[last_ring-1] == 1){
            run_S_rule_once_array[last_ring-1] = 0;                 // S-rule down ring
        }
        else{
            run_S_rule_once_array[last_ring-1] = 1;                 // S-rule on ring
        }
    }
    else{
        run_S_rule_once(run_S_rule_once_array, last_ring-1);        // go back to find 11 or 01 again
    }
}
```

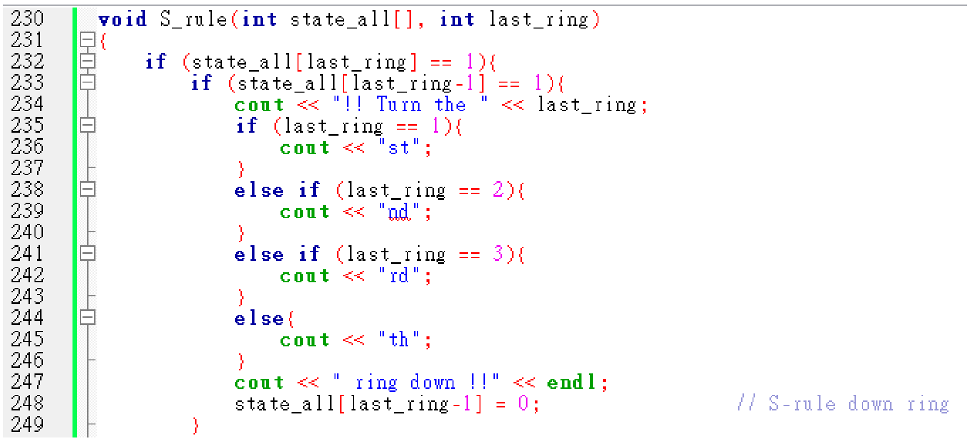
(a) 目的：此 function 是為了在執行時，透過 function show_state 印出執行一次 S-rule 的結果，所以特地製作一個 array 和 function 來達成此目的。

(b) argument run_S_rule_once_array[]為透過 function array_assign 將 array state_all assign 給 array run_S_rule_once_array 所形成的 array run_S_rule_once_array。

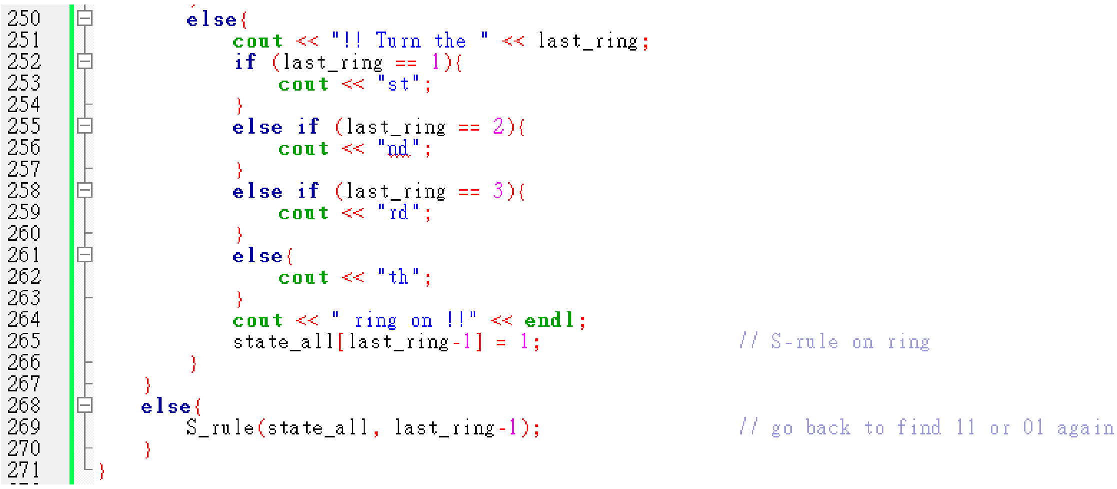
(c) last_ring 特別指的是 run_S_rule_once_array 中從右邊往左數第一個遇到值為 1 的 element，故需要運用 if…else statement 以及 recursion 的方式將此 element 找出。

(d) 而後根據程式運作分析（二、1）所示之 S-rule 規則，對 last_ring 左邊的 element 執行 turn down/on the ring 並將結果存入 run_S_rule_once_array。

(9) S_rule



```
void S_rule(int state_all[], int last_ring)
{
    if (state_all[last_ring] == 1){
        if (state_all[last_ring-1] == 1){
            cout << "!! Turn the " << last_ring;
            if (last_ring == 1){
                cout << "st";
            }
            else if (last_ring == 2){
                cout << "nd";
            }
            else if (last_ring == 3){
                cout << "rd";
            }
            else{
                cout << "th";
            }
            cout << " ring down !!" << endl;
            state_all[last_ring-1] = 0;                     // S-rule down ring
        }
```



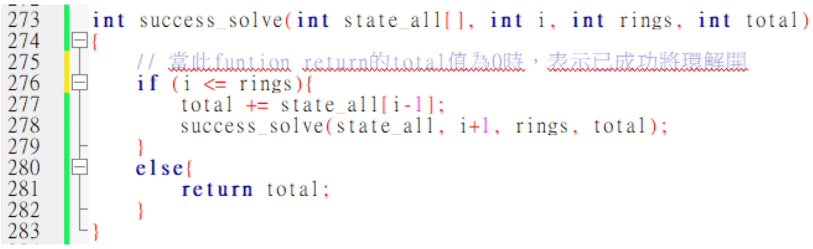
```
        else{
            cout << "!! Turn the " << last_ring;
            if (last_ring == 1){
                cout << "st";
            }
            else if (last_ring == 2){
                cout << "nd";
            }
            else if (last_ring == 3){
                cout << "rd";
            }
            else{
                cout << "th";
            }
            cout << " ring on !!" << endl;
            state_all[last_ring-1] = 1;                 // S-rule on ring
        }
    }
    else{
        S_rule(state_all, last_ring-1);                 // go back to find 11 or 01 again
    }
}
```

(a) 解環時所使用的 S-rule，兼具顯示步驟以及更動 array state_all 的功能。

(b) last_ring 特別指的是 run_S_rule_once_array 中從右邊往左數第一個遇到值為 1 的 element，故需要運用 if…else statement 以及 recursion 的方式將此 element 找出。

(c) 而後根據程式運作分析（二、1）所示之 S-rule 規則，對 last_ring 左邊的 element 執行 turn down/on the ring。

(10) success_solve

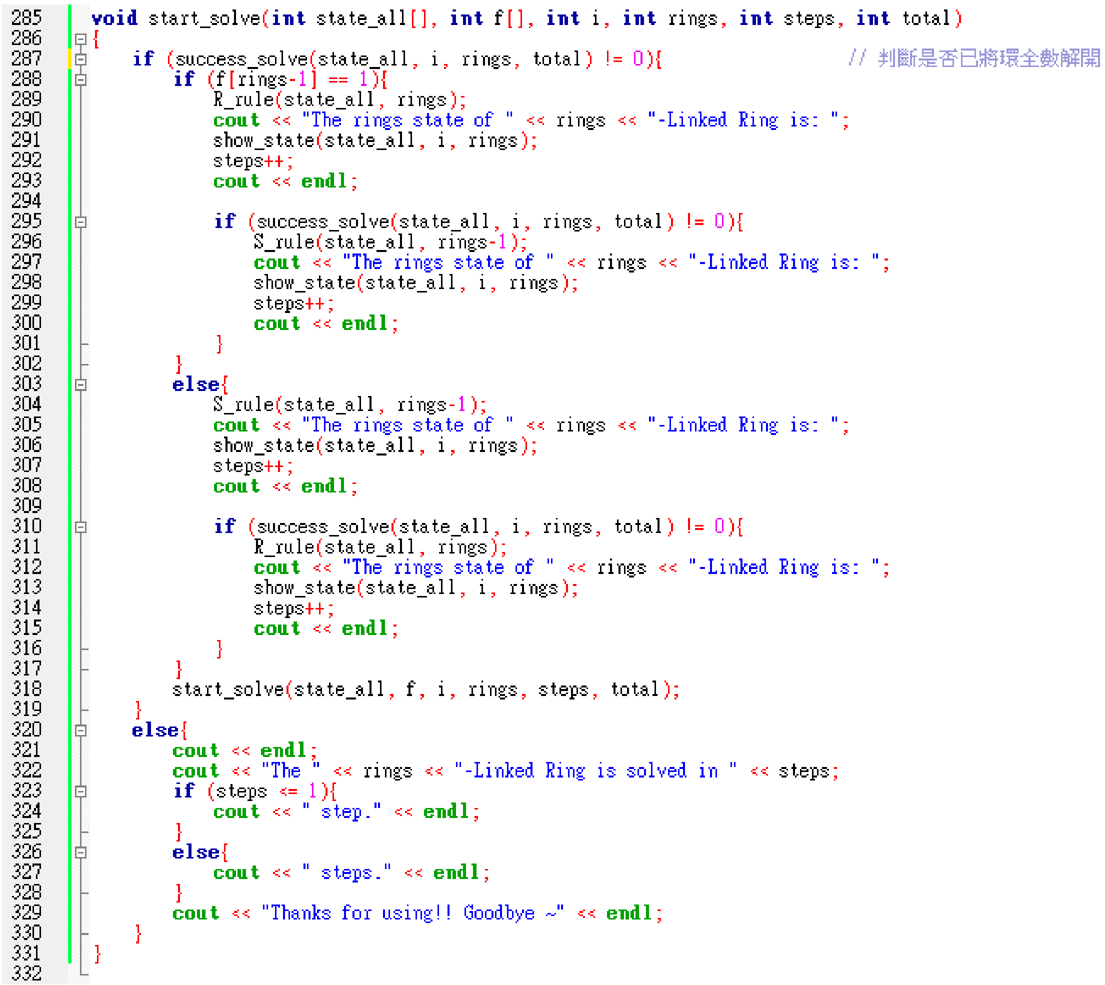


```
int success_solve(int state_all[], int i, int rings, int total)
{
    // 當此funtion return的total值為0時，表示已成功將環解開
    if (i <= rings){
        total += state_all[i-1];
        success_solve(state_all, i+1, rings, total);
    }
    else{
        return total;
    }
}
```

(a) 目的：判斷是否成功將環全數解開。

(b) 將 array state_all 當下的狀態放進此 function 判斷 array 內所有值的加總（total）。若未全數解開，則 total > 0；當所有環已成功解開時，array 內的值皆為 0，表示 S0 = 000…，而 array state_all 內所有 element 的總和為 0（total = 0）。

(11) start_solve



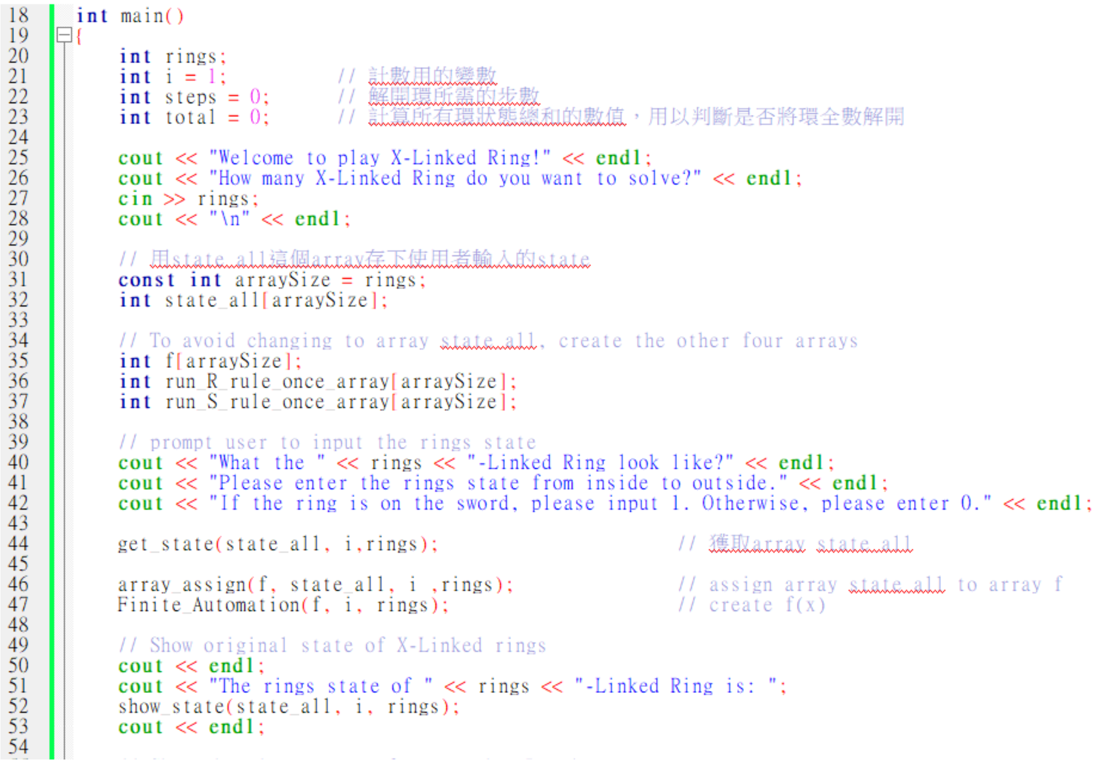
```
285  void start_solve(int state_all[], int f[], int i, int rings, int steps, int total)
286  {
287      if (success_solve(state_all, i, rings, total) != 0){             // 判斷是否已將環全數解開
288          if (f[rings-1] == 1){
289              R_rule(state_all, rings);
290              cout << "The rings state of " << rings << "-Linked Ring is: ";
291              show_state(state_all, i, rings);
292              steps++;
293              cout << endl;
294  
295              if (success_solve(state_all, i, rings, total) != 0){
296                  S_rule(state_all, rings-1);
297                  cout << "The rings state of " << rings << "-Linked Ring is: ";
298                  show_state(state_all, i, rings);
299                  steps++;
300                  cout << endl;
301              }
302          }
303          else{
304              S_rule(state_all, rings-1);
305              cout << "The rings state of " << rings << "-Linked Ring is: ";
306              show_state(state_all, i, rings);
307              steps++;
308              cout << endl;
309  
310              if (success_solve(state_all, i, rings, total) != 0){
311                  R_rule(state_all, rings);
312                  cout << "The rings state of " << rings << "-Linked Ring is: ";
313                  show_state(state_all, i, rings);
314                  steps++;
315                  cout << endl;
316              }
317          }
318          start_solve(state_all, f, i, rings, steps, total);
319      }
320      else{
321          cout << endl;
322          cout << "The " << rings << "-Linked Ring is solved in " << steps;
323          if (steps <= 1){
324              cout << " step." << endl;
325          }
326          else{
327              cout << " steps." << endl;
328          }
329          cout << "Thanks for using!! Goodbye ~" << endl;
330      }
331  }
332  
```

(a) 運用 function success_solve 回傳的 total 值判斷是否成功將環全數解開。若尚未成功（total ≠ 0），則進行解環；若已成功（total = 0），則跳至第 320 行將步數與結束的訊息印出。

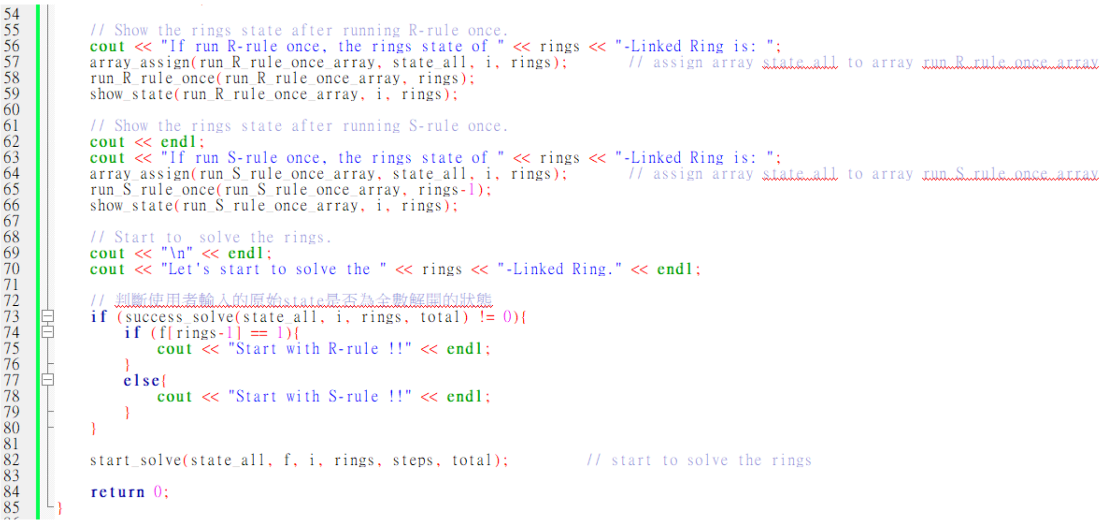
(b) 第 288 與 303 行：當 array f 最右邊的 element 為 1 時，表示 $f(S0)_2$ 為奇數，需要先操作 R-rule；為 0 時，表示 $f(S0)_2$ 為偶數，需要先操作 S-rule。且無論哪種情形發生，每次執行完 R-rule 或 S-rule 一次後，都需要判斷是否已成功將全部的環解開，再執行後續步驟。

(c) 第 323 行：當步數 step 為小於或等於 1 時，step 為單數，需印出"step"；而步數大於 1 時，step 為複數，需印出"steps"。

7. 主程式運作（main function）



```
18  int main()
19  {
20      int rings;
21      int i = 1;          // 計數用的變數
22      int steps = 0;      // 解開環所需的步數
23      int total = 0;      // 計算所有環狀態總和的數值，用以判斷是否將環全數解開
24
25      cout << "Welcome to play X-Linked Ring!" << endl;
26      cout << "How many X-Linked Ring do you want to solve?" << endl;
27      cin >> rings;
28      cout << "\n" << endl;
29
30      // 用state_all這個array存下使用者輸入的state
31      const int arraySize = rings;
32      int state_all[arraySize];
33
34      // To avoid changing to array state_all, create the other four arrays
35      int f[arraySize];
36      int run_R_rule_once_array[arraySize];
37      int run_S_rule_once_array[arraySize];
38
39      // prompt user to input the rings state
40      cout << "What the " << rings << "-Linked Ring look like?" << endl;
41      cout << "Please enter the rings state from inside to outside." << endl;
42      cout << "If the ring is on the sword, please input 1. Otherwise, please enter 0." << endl;
43
44      get_state(state_all, i,rings);                      // 獲取array state_all
45
46      array_assign(f, state_all, i ,rings);               // assign array state_all to array f
47      Finite_Automation(f, i, rings);                     // create f(x)
48
49      // Show original state of X-Linked rings
50      cout << endl;
51      cout << "The rings state of " << rings << "-Linked Ring is: ";
52      show_state(state_all, i, rings);
53      cout << endl;
54
```



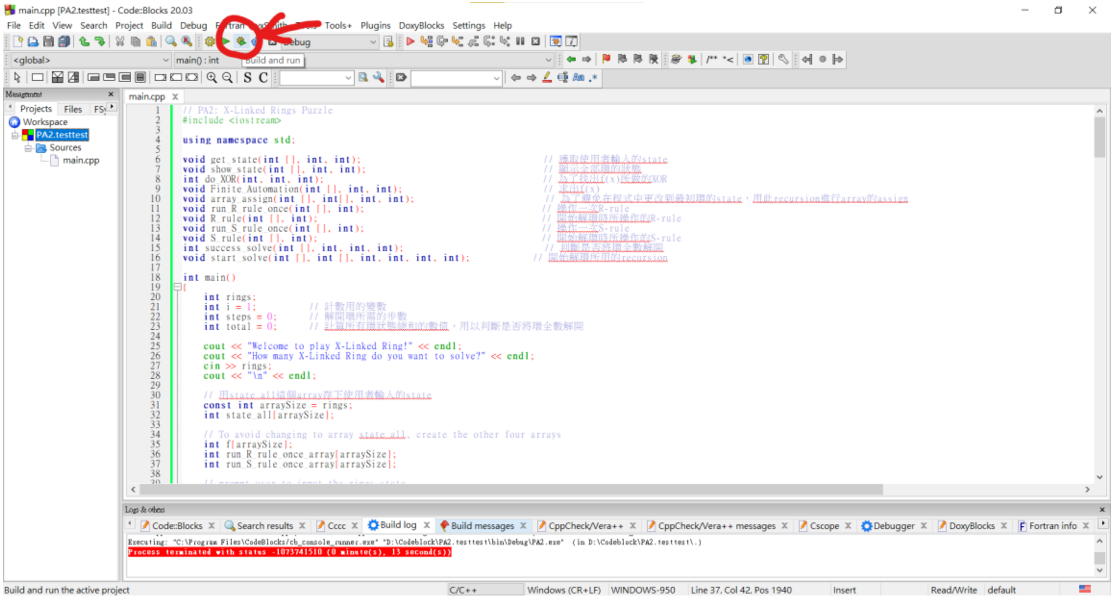
```
54
55      // Show the rings state after running R-rule once.
56      cout << "If run R-rule once, the rings state of " << rings << "-Linked Ring is: ";
57      array_assign(run_R_rule_once_array, state_all, i, rings);      // assign array state_all to array run_R_rule_once_array
58      run_R_rule_once(run_R_rule_once_array, rings);
59      show_state(run_R_rule_once_array, i, rings);
60
61      // Show the rings state after running S-rule once.
62      cout << endl;
63      cout << "If run S-rule once, the rings state of " << rings << "-Linked Ring is: ";
64      array_assign(run_S_rule_once_array, state_all, i, rings);      // assign array state_all to array run_S_rule_once_array
65      run_S_rule_once(run_S_rule_once_array, rings-1);
66      show_state(run_S_rule_once_array, i, rings);
67
68      // Start to  solve the rings.
69      cout << "\n" << endl;
70      cout << "Let's start to solve the " << rings << "-Linked Ring." << endl;
71
72      // 判斷使用者輸入的原始state是否為全數解開的狀態
73      if (success_solve(state_all, i, rings, total) != 0){
74          if (f[rings-1] == 1){
75              cout << "Start with R-rule !!" << endl;
76          }
77          else{
78              cout << "Start with S-rule !!" << endl;
79          }
80      }
81
82      start_solve(state_all, f, i, rings, steps, total);          // start to solve the rings
83
84      return 0;
85  }
```

(a) 第 46、57 與 64 行：將 array state_all 分別 assign 給 array f、array run_R_rule_once_array 與 run_S_rule_once_array。

(b) 第 74 行：當使用者輸入的環狀態已為成功解開的狀態（S0 = 000…），則不印出"Start with R-rule !!"或"Start with S-rule !!"

三、　How to compile and execute program

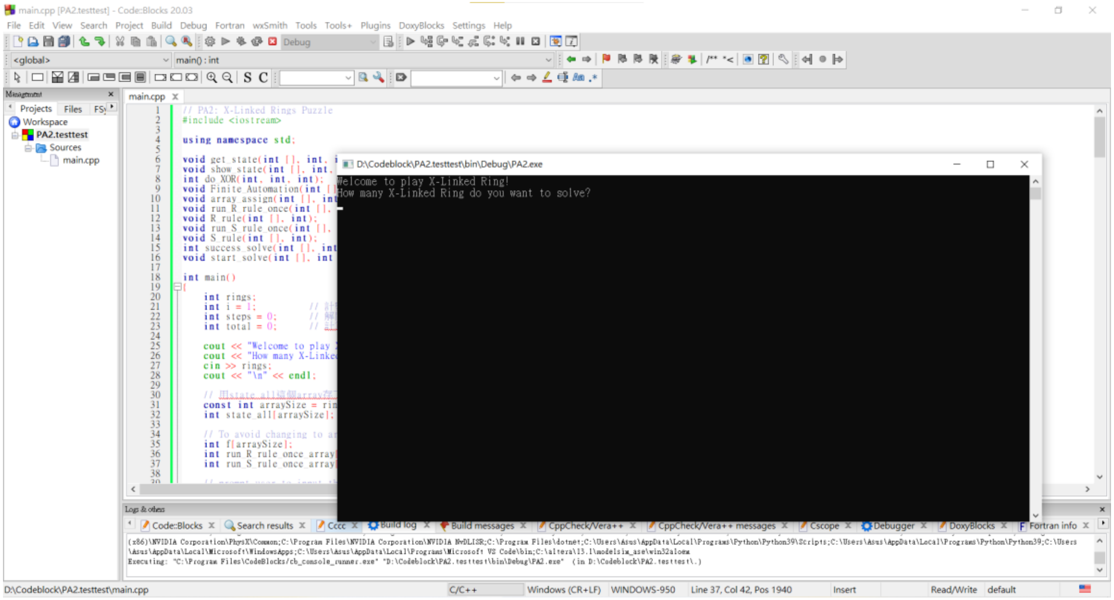
Step1. 點選 Build and run


main.cpp [PA2.testtest] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
main() : int
Build and run
Management
Projects Files FSy
Workspace
PA2.testtest
Sources
main.cpp
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck/Vera++ CppCheck/Vera++ messages Cscope Debugger DoxyBlocks Fortran info
Build and run the active project
C/C++ Windows (CR+LF) WINDOWS-950 Line 37, Col 42, Pos 1940 Insert Read/Write default

```cpp
// PA2: X-Linked Rings Puzzle
#include <iostream>

using namespace std;

void get_state(int [], int, int);                                  // 獲取使用者輸入的state
void show_state(int [], int, int);                                 // 顯示全部環的狀態
int do_XOR(int, int, int);                                         // 為了找出f(x)所做的XOR
void Finite_Automation(int [], int, int);                          // 求出f(x)
void array_assign(int [], int[], int, int);                        // 為了避免在程式中更改到最初環的state，用此recursion進行array的assign
void run_R_rule_once(int [], int);                                 // 操作一次R-rule
void R_rule(int [], int);                                          // 開始解環時所操作的R-rule
void run_S_rule_once(int [], int);                                 // 操作一次S-rule
void S_rule(int [], int);                                          // 開始解環時所操作的S-rule
int success_solve(int [], int, int, int);                          // 判斷是否將環全數解開
void start_solve(int [], int [], int, int, int, int);             // 開始解環所用的recursion

int main()
{
    int rings;
    int i = 1;          // 計數用的變數
    int steps = 0;      // 解開環所需的步數
    int total = 0;      // 計算所有環狀態總和的數值，用以判斷是否將環全數解開

    cout << "Welcome to play X-Linked Ring!" << endl;
    cout << "How many X-Linked Ring do you want to solve?" << endl;
    cin >> rings;
    cout << "\n" << endl;

    // 用state_all這個array存下使用者輸入的state
    const int arraySize = rings;
    int state_all[arraySize];

    // To avoid changing to array state_all, create the other four arrays
    int f[arraySize];
    int run_R_rule_once_array[arraySize];
    int run_S_rule_once_array[arraySize];
```

```
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "D:\Codeblock\PA2.testtest\bin\Debug\PA2.exe"  (in D:\Codeblock\PA2.testtest\.)
Process terminated with status -1073741510 (0 minute(s), 13 second(s))
```

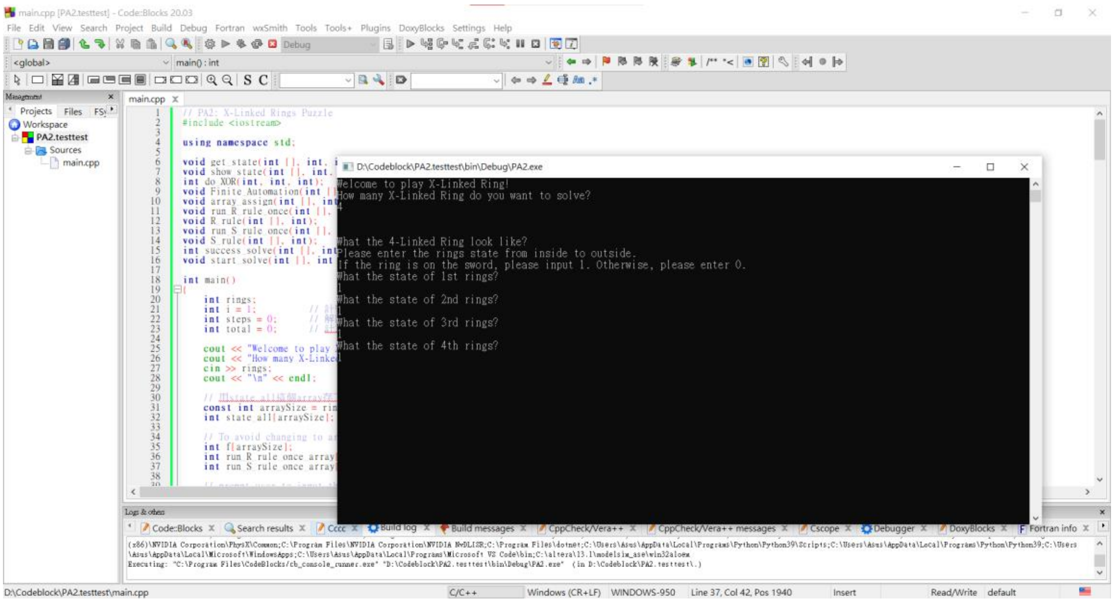
Step2. 依照提示字串輸入數值


main.cpp [PA2.testtest] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
main() : int
Management
Projects Files FSy
Workspace
PA2.testtest
Sources
main.cpp
D:\Codeblock\PA2.testtest\bin\Debug\PA2.exe
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck/Vera++ CppCheck/Vera++ messages Cscope Debugger DoxyBlocks Fortran info
D:\Codeblock\PA2.testtest\main.cpp
C/C++ Windows (CR+LF) WINDOWS-950 Line 37, Col 42, Pos 1940 Insert Read/Write default

```
Welcome to play X-Linked Ring!
How many X-Linked Ring do you want to solve?
```

```
(x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR;C:\Program Files\dotnet;C:\Users\Asus\AppData\Local\Programs\Python\Python39\Scripts;C:\Users\Asus\AppData\Local\Programs\Python\Python39;C:\Users
\Asus\AppData\Local\Microsoft\WindowsApps;C:\Users\Asus\AppData\Local\Programs\Microsoft VS Code\bin;C:\altera\13.1\modelsim_ase\win32aloem
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "D:\Codeblock\PA2.testtest\bin\Debug\PA2.exe"  (in D:\Codeblock\PA2.testtest\.)
```

Step3. 依序輸入提示字串要求的數值


main.cpp [PA2.testtest] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
main() : int
Management
Projects Files FSy
Workspace
PA2.testtest
Sources
main.cpp
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck/Vera++ CppCheck/Vera++ messages Cscope Debugger DoxyBlocks Fortran info
D:\Codeblock\PA2.testtest\main.cpp C/C++ Windows (CR+LF) WINDOWS-950 Line 37, Col 42, Pos 1940 Insert Read/Write default

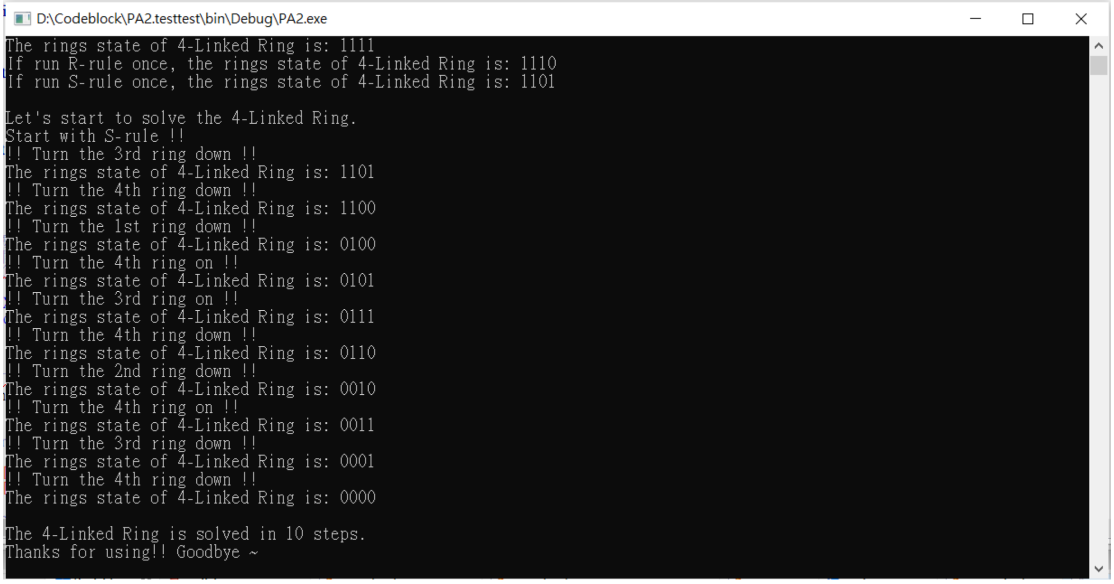
```
// PA2: X-Linked Rings Puzzle
#include <iostream>

using namespace std;

void get_state(int [], int, i
void show_state(int [], int,
int do_XOR(int, int, int);
void Finite_Automation(int [
void array_assign(int [], int
void run_R_rule_once(int [],
void R_rule(int [], int);
void run_S_rule_once(int [],
void S_rule(int [], int);
int success_solve(int [], int
void start_solve(int [], int

int main()
{
    int rings;
    int i = 1;          // 計
    int steps = 0;      // 解
    int total = 0;      // 紀

    cout << "Welcome to play
    cout << "How many X-Linked
    cin >> rings;
    cout << "\n" << endl;

    // 把state_all弄個array存
    const int arraySize = rin
    int state_all[arraySize];

    // To avoid changing to a
    int f[arraySize];
    int run_R_rule_once_array
    int run_S_rule_once_array
```

D:\Codeblock\PA2.testtest\bin\Debug\PA2.exe

```
Welcome to play X-Linked Ring!
How many X-Linked Ring do you want to solve?
4


What the 4-Linked Ring look like?
Please enter the rings state from inside to outside.
If the ring is on the sword, please input 1. Otherwise, please enter 0.
What the state of 1st rings?
1
What the state of 2nd rings?
1
What the state of 3rd rings?
1
What the state of 4th rings?
1
```

```
(x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR;C:\Program Files\dotnet;C:\Users\Asus\AppData\Local\Programs\Python\Python39\Scripts;C:\Users\Asus\AppData\Local\Programs\Python\Python39;C:\Users
\Asus\AppData\Local\Microsoft\WindowsApps;C:\Users\Asus\AppData\Local\Programs\Microsoft VS Code\bin;C:\altera\13.1\modelsim_ase\win32aloem
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "D:\Codeblock\PA2.testtest\bin\Debug\PA2.exe"  (in D:\Codeblock\PA2.testtest\.)
```

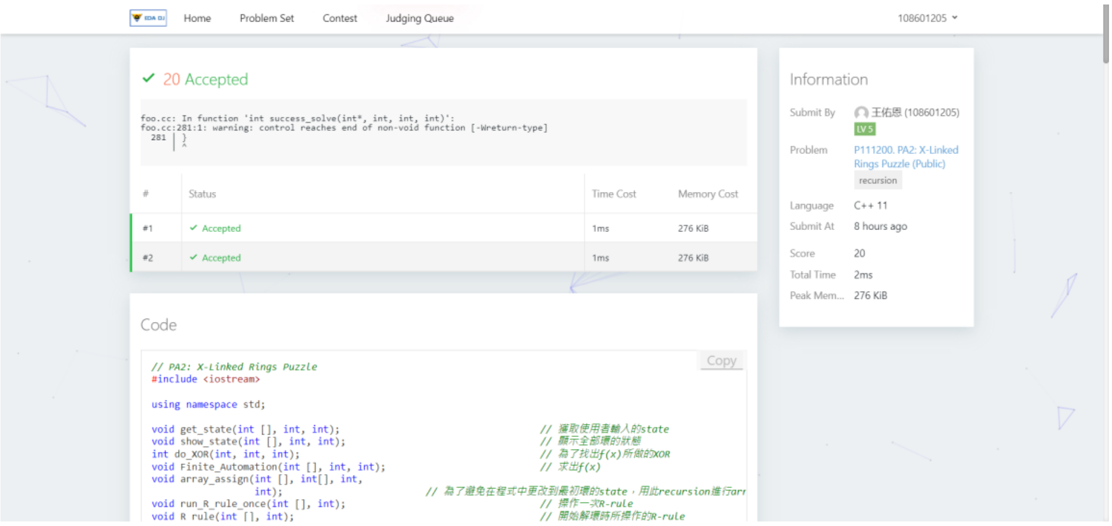
Step4. 得到結果


D:\Codeblock\PA2.testtest\bin\Debug\PA2.exe

```
The rings state of 4-Linked Ring is: 1111
If run R-rule once, the rings state of 4-Linked Ring is: 1110
If run S-rule once, the rings state of 4-Linked Ring is: 1101

Let's start to solve the 4-Linked Ring.
Start with S-rule !!
!! Turn the 3rd ring down !!
The rings state of 4-Linked Ring is: 1101
!! Turn the 4th ring down !!
The rings state of 4-Linked Ring is: 1100
!! Turn the 1st ring down !!
The rings state of 4-Linked Ring is: 0100
!! Turn the 4th ring on !!
The rings state of 4-Linked Ring is: 0101
!! Turn the 3rd ring on !!
The rings state of 4-Linked Ring is: 0111
!! Turn the 4th ring down !!
The rings state of 4-Linked Ring is: 0110
!! Turn the 2nd ring down !!
The rings state of 4-Linked Ring is: 0010
!! Turn the 4th ring on !!
The rings state of 4-Linked Ring is: 0011
!! Turn the 3rd ring down !!
The rings state of 4-Linked Ring is: 0001
!! Turn the 4th ring down !!
The rings state of 4-Linked Ring is: 0000

The 4-Linked Ring is solved in 10 steps.
Thanks for using!! Goodbye ~
```

四、 結果呈現（斜體字為 input）

1. OJ



✓ 20 Accepted

```
foo.cc: In function 'int success_solve(int*, int, int, int)':
foo.cc:281:1: warning: control reaches end of non-void function [-Wreturn-type]
  281 | }
      | ^
```

#	Status	Time Cost	Memory Cost
#1	✓ Accepted	1ms	276 KiB
#2	✓ Accepted	1ms	276 KiB

Information

Submit By	王佑恩 (108601205) LV 5
Problem	P111200. PA2: X-Linked Rings Puzzle (Public) recursion
Language	C++ 11
Submit At	8 hours ago
Score	20
Total Time	2ms
Peak Mem...	276 KiB

Code

```
// PA2: X-Linked Rings Puzzle
#include <iostream>

using namespace std;

void get_state(int [], int, int);                                   // 獲取使用者輸入的state
void show_state(int [], int, int);                                  // 顯示全部環的狀態
int do_XOR(int, int, int);                                          // 為了找出f(x)所做的XOR
void Finite_Automation(int [], int, int);                           // 求出f(x)
void array_assign(int [], int[], int,
                  int);                       // 為了避免在程式中更改到最初環的state，用此recursion進行arr
void run_R_rule_once(int [], int);                                  // 操作一次R-rule
void R rule(int [], int);                                           // 開始解環時所操作的R-rule
```

2. 測試資料 1 (S0 = 11111)

```
Welcome to play X-Linked Ring!
How many X-Linked Ring do you want to solve?
5



What the 5-Linked Ring look like?
Please enter the rings state from inside to outside.
If the ring is on the sword, please input 1. Otherwise, please enter 0.
What the state of 1st rings?
1
What the state of 2nd rings?
1
What the state of 3rd rings?
1
What the state of 4th rings?
1
What the state of 5th rings?
1

The rings state of 5-Linked Ring is: 11111
If run R-rule once, the rings state of 5-Linked Ring is: 11110
If run S-rule once, the rings state of 5-Linked Ring is: 11101
```

```
Let's start to solve the 5-Linked Ring.
Start with R-rule !!
!! Turn the 5th ring down !!
The rings state of 5-Linked Ring is: 11110
!! Turn the 3rd ring down !!
The rings state of 5-Linked Ring is: 11010
!! Turn the 5th ring on !!
The rings state of 5-Linked Ring is: 11011
!! Turn the 4th ring down !!
The rings state of 5-Linked Ring is: 11001
!! Turn the 5th ring down !!
The rings state of 5-Linked Ring is: 11000
!! Turn the 1st ring down !!
The rings state of 5-Linked Ring is: 01000
!! Turn the 5th ring on !!
The rings state of 5-Linked Ring is: 01001
!! Turn the 4th ring on !!
The rings state of 5-Linked Ring is: 01011
!! Turn the 5th ring down !!
The rings state of 5-Linked Ring is: 01010
!! Turn the 3rd ring on !!
The rings state of 5-Linked Ring is: 01110
!! Turn the 5th ring on !!
The rings state of 5-Linked Ring is: 01111
!! Turn the 4th ring down !!
The rings state of 5-Linked Ring is: 01101
!! Turn the 5th ring down !!
The rings state of 5-Linked Ring is: 01100
!! Turn the 2nd ring down !!
The rings state of 5-Linked Ring is: 00100
!! Turn the 5th ring on !!
The rings state of 5-Linked Ring is: 00101
!! Turn the 4th ring on !!
The rings state of 5-Linked Ring is: 00111
!! Turn the 5th ring down !!
The rings state of 5-Linked Ring is: 00110
!! Turn the 3rd ring down !!
```

```
The rings state of 5-Linked Ring is: 00010
!! Turn the 5th ring on !!
The rings state of 5-Linked Ring is: 00011
!! Turn the 4th ring down !!
The rings state of 5-Linked Ring is: 00001
!! Turn the 5th ring down !!
The rings state of 5-Linked Ring is: 00000

The 5-Linked Ring is solved in 21 steps.
Thanks for using!! Goodbye ~
```

2. 測試資料 2 (S0 = 1011)

```
Welcome to play X-Linked Ring!
How many X-Linked Ring do you want to solve?
4



What the 4-Linked Ring look like?
Please enter the rings state from inside to outside.
If the ring is on the sword, please input 1. Otherwise, please enter 0.
What the state of 1st rings?
1
What the state of 2nd rings?
0
What the state of 3rd rings?
1
What the state of 4th rings?
1

The rings state of 4-Linked Ring is: 1011
If run R-rule once, the rings state of 4-Linked Ring is: 1010
If run S-rule once, the rings state of 4-Linked Ring is: 1001

Let's start to solve the 4-Linked Ring.
Start with R-rule !!
!! Turn the 4th ring down !!
The rings state of 4-Linked Ring is: 1010
!! Turn the 2nd ring on !!
```

```
The rings state of 4-Linked Ring is: 1110
!! Turn the 4th ring on !!
The rings state of 4-Linked Ring is: 1111
!! Turn the 3rd ring down !!
The rings state of 4-Linked Ring is: 1101
!! Turn the 4th ring down !!
The rings state of 4-Linked Ring is: 1100
!! Turn the 1st ring down !!
The rings state of 4-Linked Ring is: 0100
!! Turn the 4th ring on !!
The rings state of 4-Linked Ring is: 0101
!! Turn the 3rd ring on !!
The rings state of 4-Linked Ring is: 0111
!! Turn the 4th ring down !!
The rings state of 4-Linked Ring is: 0110
!! Turn the 2nd ring down !!
The rings state of 4-Linked Ring is: 0010
!! Turn the 4th ring on !!
The rings state of 4-Linked Ring is: 0011
!! Turn the 3rd ring down !!
The rings state of 4-Linked Ring is: 0001
!! Turn the 4th ring down !!
The rings state of 4-Linked Ring is: 0000

The 4-Linked Ring is solved in 13 steps.
Thanks for using!! Goodbye ~
```

3. 測試資料 3 (S0 = 00000000000)

```
Welcome to play X-Linked Ring!
How many X-Linked Ring do you want to solve?
11


What the 11-Linked Ring look like?
Please enter the rings state from inside to outside.
If the ring is on the sword, please input 1. Otherwise, please enter 0.
What the state of 1st rings?
0
```

```
What the state of 2nd rings?
0
What the state of 3rd rings?
0
What the state of 4th rings?
0
What the state of 5th rings?
0
What the state of 6th rings?
0
What the state of 7th rings?
0
What the state of 8th rings?
0
What the state of 9th rings?
0
What the state of 10th rings?
0
What the state of 11th rings?
0

The rings state of 11-Linked Ring is: 00000000000
If run R-rule once, the rings state of 11-Linked Ring is: 00000000001
If run S-rule once, the rings state of 11-Linked Ring is: 00000000000

Let's start to solve the 11-Linked Ring.

The 11-Linked Ring is solved in 0 step.
Thanks for using!! Goodbye ~
```

五、 難處發現與解決

1. **問題 1：**

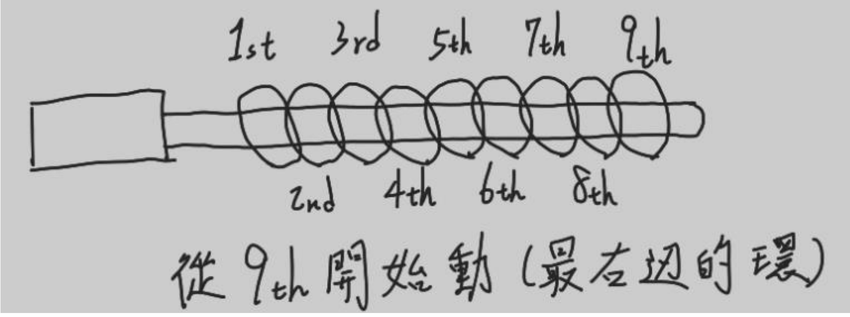
在思考 R-rule 與 S-rule 的運作時，花了很長的時間。把環的順序與解環的過程一起思考，導致被混淆。

問題解決：

後來才發現我是被環的順序與所有提供的參考資料所搞混。在參考資料裡所提供的九連環模擬器，最外側的環其順序是 1st，比較直

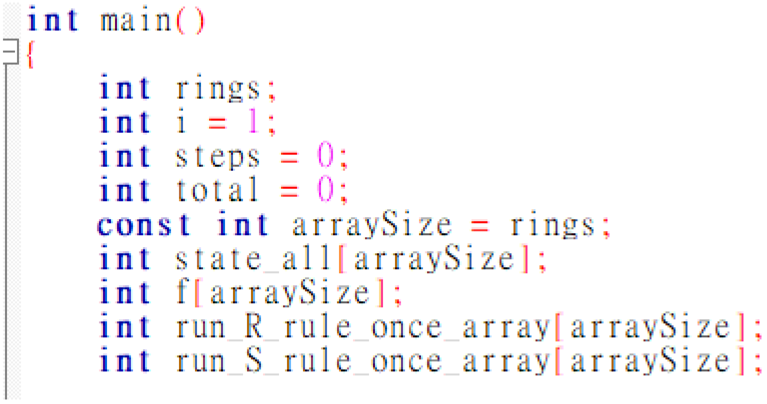
觀。而 PA2 的電子檔，被稱作是 1st 的環反而是最靠內側的環，而 R-rule 要解的「最右邊的環」是最靠外側的環。

後來把圖都畫出來後，很快就了解 R-rule 與 S-rule 的運作。



2. **問題 2：**

Main function 中將變數的宣告統一寫在最開頭，導致 compile 的時候，有些變數呈現亂數。



```
int main()
{
    int rings;
    int i = 1;
    int steps = 0;
    int total = 0;
    const int arraySize = rings;
    int state_all[arraySize];
    int f[arraySize];
    int run_R_rule_once_array[arraySize];
    int run_S_rule_once_array[arraySize];
```

問題解決：

變數的宣告需要看時機，因為 C++是由上到下執行，若是變數之間有相關，且太早宣告變數的話，會造成變數的值沒有成功 assign。

六、 回饋

這次的 PA 運用到相當大量的 recursion，讓我對 recursion 的編寫邏輯，以及 function call function 的邏輯更加的熟悉。此外，花了一天的時間仔細思考程式的編寫後，可以明顯感覺到編寫程式的時間縮短很多，慢慢了解到老師在上課所說的：「花最多的時間思考，較少的時間 coding」。

七、 Reference

[1] 郭君逸,「九連環與格雷碼」數學傳播 38 卷 3 期, pp. 13-24
38302.pdf (sinica.edu.tw)