

一、 操作手冊

【程式功能說明】

使用 Quine McClusky (或 Petrick's Method) 找出 10 變數函式之 Minimum SOP 所有可能解。

【程式使用流程】

1. 將欲讀取的檔案名稱設為 input.txt，欲寫入的檔案名稱則設為 output.txt，並與其他檔案放置在同一資料夾中。

input.txt 格式：

將 10 變數函式的 minterm 和 don't care term 以下方範例之格式表示。

***Don't care term 請寫在 () 內**

範例：

0

1

16

17

128

(341)

512

640

1023

2. 開啟 simulator.exe，執行畫面如下：
(請依照指示操作以繼續下一步)

```
c:\users\user\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
Combine terms that have the same '-' and differ one in the number of 1's.
Check off the combined terms.

=====Round0=====
0000000000 0,
-----
0000000001 1,
0000010000 16,
0010000000 128,
1000000000 512,
-----
0000010001 17,
1010000000 640,
-----
d 0101010101 341,
-----
1111111111 1023,
-----

Press ENTER to see the next step...
```

```
c:\users\user\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe

=====Round1=====
v 0000000000 0,
-----
v 0000000001 1,
v 0000010000 16,
v 0010000000 128,
v 1000000000 512,
-----
0000010001 17,
1010000000 640,
-----
d 0101010101 341,
-----
1111111111 1023,
-----
<NEW GROUP>
000000000- 0,1,
00000-0000 0,16,
00-0000000 0,128,
-000000000 0,512,

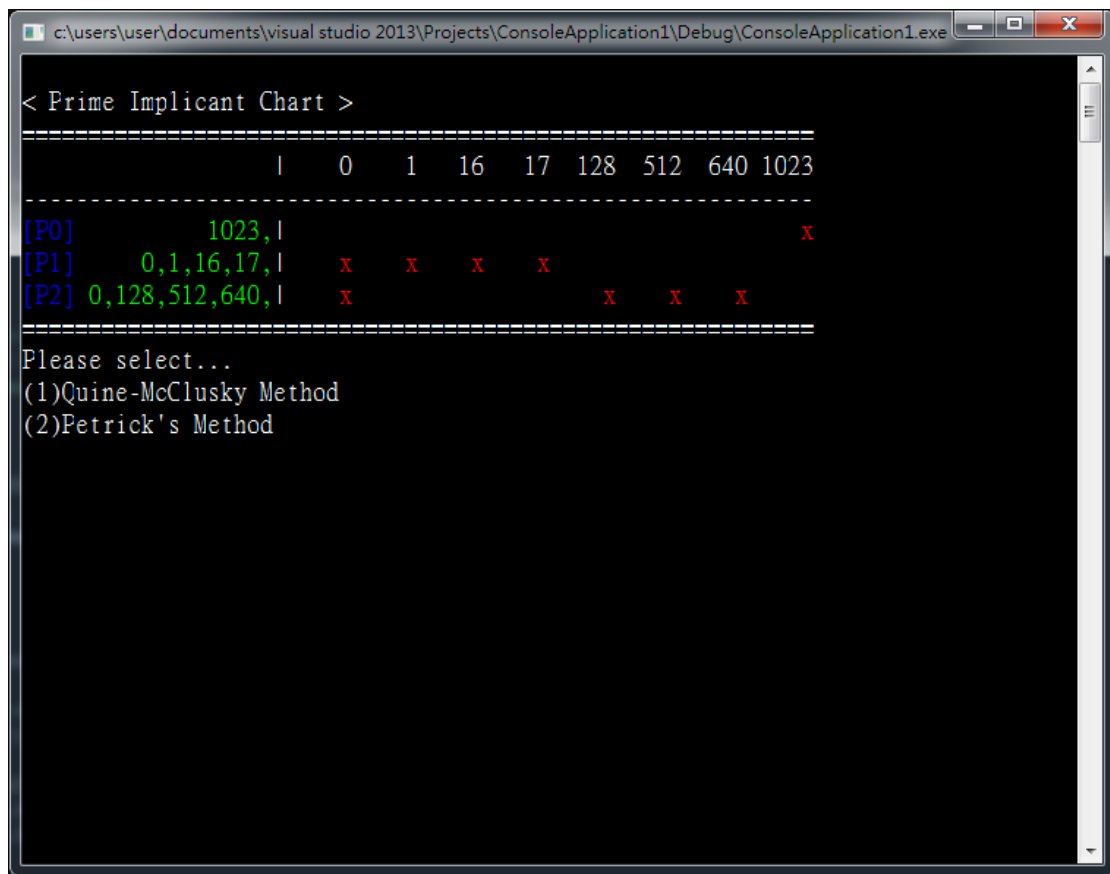
Press ENTER to see the next step...
```

〈註〉 d 表 don't care ; v 表 checked off

3. 請依指示輸入，選擇欲使用之解法：

1 → 使用 Quine McClasky Method

2 → 使用 Petrick's Method



```
< Prime Implicant Chart >
=====
      |  0  1  16  17 128 512 640 1023
-----
[P0]      1023, |                                x
[P1]    0,1,16,17, |      x   x   x   x
[P2]  0,128,512,640, |    x                                x   x   x
=====
Please select...
(1)Quine-McClusky Method
(2)Petrick's Method
```

4. 若輸入 1：Quine McClasky，執行畫面如下：

(請依照指示操作以繼續下一步)

```

c:\users\user\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
Minterm1 is only covered by P1.
P1 has to be chosen. Delete P1 and the other minterms it covers.

< Prime Implicant Chart >
=====
      | 128 512 640 1023
-----
[P0]      1023, |
[P2] 0,128,512,640, | x x x
=====

Press ENTER to see the next step...

```

```

c:\users\user\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
      | 1023
-----
[P0]      1023, | x
=====

Press ENTER to see the next step...

Minterm1023 is only covered by P0.
P0 has to be chosen. Delete P0 and the other minterms it covers.

< Prime Implicant Chart >
=====
      |
-----

Press ENTER to see the next step...

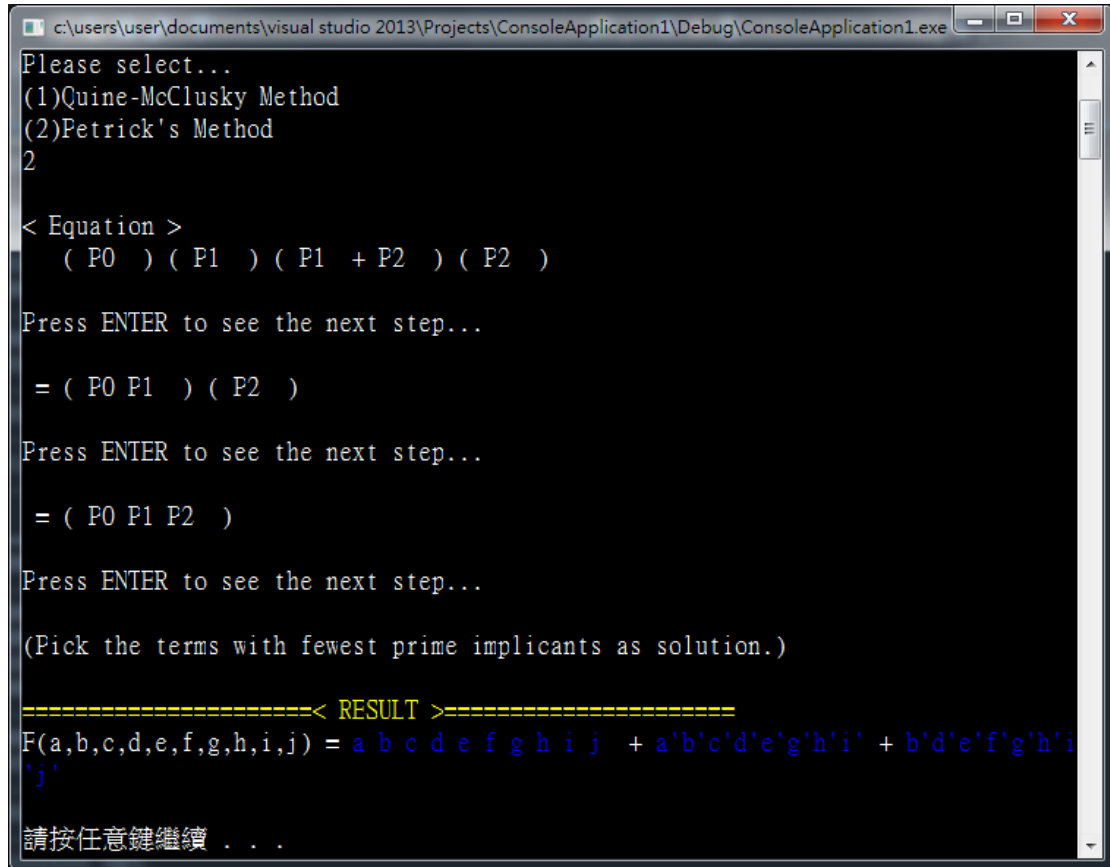
===== < RESULT > =====
F(a,b,c,d,e,f,g,h,i,j) = a b c d e f g h i j + a'b'c'd'e'g'h'i' + b'd'e'f'g'h'i'
'j'

請按任意鍵繼續 . . .

```

〈註〉若有多組解，將接續使用 Petrick's Method 找出所有可能解。

5. 若輸入 2：Petrick's Method，執行畫面如下：
(請依照指示操作以繼續下一步)



```
c:\users\user\documents\visual studio 2013\Projects\ConsoleApplication1\Debug\ConsoleApplication1.exe
Please select...
(1)Quine-McClusky Method
(2)Petrick's Method
2

< Equation >
  ( P0 ) ( P1 ) ( P1 + P2 ) ( P2 )

Press ENTER to see the next step...

= ( P0 P1 ) ( P2 )

Press ENTER to see the next step...

= ( P0 P1 P2 )

Press ENTER to see the next step...

(Pick the terms with fewest prime implicants as solution.)

===== < RESULT > =====
F(a,b,c,d,e,f,g,h,i,j) = a b c d e f g h i j + a'b'c'd'e'g'h'i' + b'd'e'f'g'h'i'j
請按任意鍵繼續...
```

6. **output.txt** 中將寫入 Adjacent Group Combination Process，印出 Prime Implicant Chart (未化簡) 和 Minimum SOP 之解。

範例：

```
=====Round0=====
0000000000 0,
-----
0000000001 1,
0000010000 16,
0010000000 128,
1000000000 512,
-----
0000010001 17,
```

```

1010000000 640,
-----
d 0101010101 341,
-----
1111111111 1023,
-----

=====Round1=====
v 0000000000 0,
-----
v 0000000001 1,
v 0000010000 16,
v 0010000000 128,
v 1000000000 512,
-----
0000010001 17,
1010000000 640,
-----
d 0101010101 341,
-----
1111111111 1023,
-----

<NEW GROUP>
0000000000- 0,1,
00000-0000 0,16,
00-0000000 0,128,
-000000000 0,512,

=====Round2=====
v 0000000000 0,
-----
v 0000000001 1,
v 0000010000 16,
v 0010000000 128,
v 1000000000 512,
-----
v 0000010001 17,
v 1010000000 640,

```

```

-----
d 0101010101 341,
-----

1111111111 1023,
-----

0000000000- 0,1,
00000-0000 0,16,
00-0000000 0,128,
-000000000 0,512,
-----

<NEW GROUP>
00000-0001 1,17,
000001000- 16,17,
-010000000 128,640,
10-0000000 512,640,

=====Round3=====
v 0000000000 0,
-----

v 0000000001 1,
v 0000010000 16,
v 0010000000 128,
v 1000000000 512,
-----

v 0000010001 17,
v 1010000000 640,
-----

d 0101010101 341,
-----

1111111111 1023,
-----

v 0000000000- 0,1,
v 00000-0000 0,16,
v 00-0000000 0,128,
v -000000000 0,512,
-----

v 00000-0001 1,17,
v 000001000- 16,17,

```

v -010000000 128,640,

v 10-0000000 512,640,

<NEW GROUP>

00000-000- 0,1,16,17,

-0-0000000 0,128,512,640,

< Prime Implicant Chart >

=====									
		0	1	16	17	128	512	640	1023

[P0]	1023,								x
[P1]	0,1,16,17,	x	x	x	x				
[P2]	0,128,512,640,	x				x	x	x	
=====									

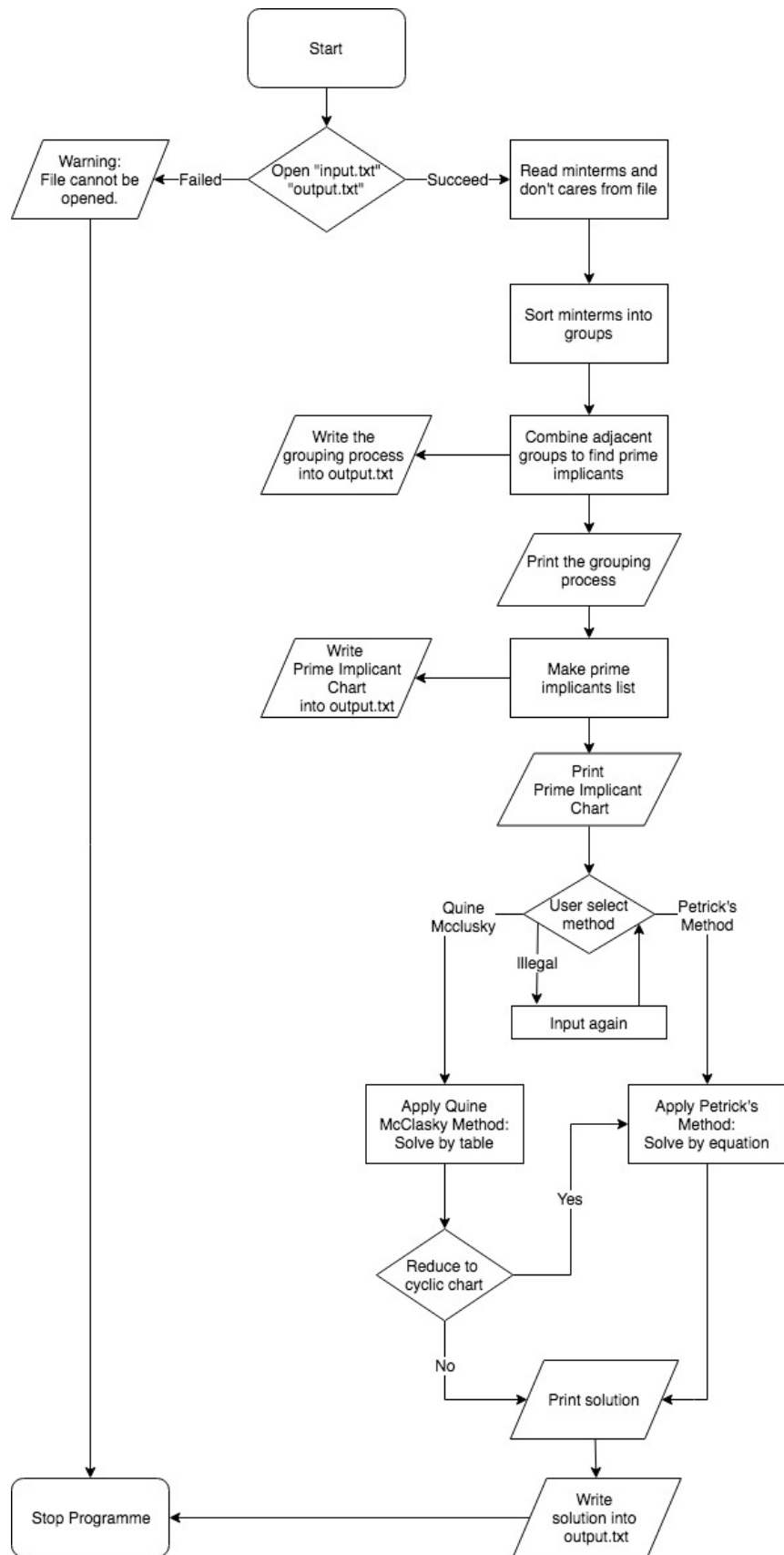
< RESULT >

F(a,b,c,d,e,f,g,h,i,j) = a b c d e f g h i j +

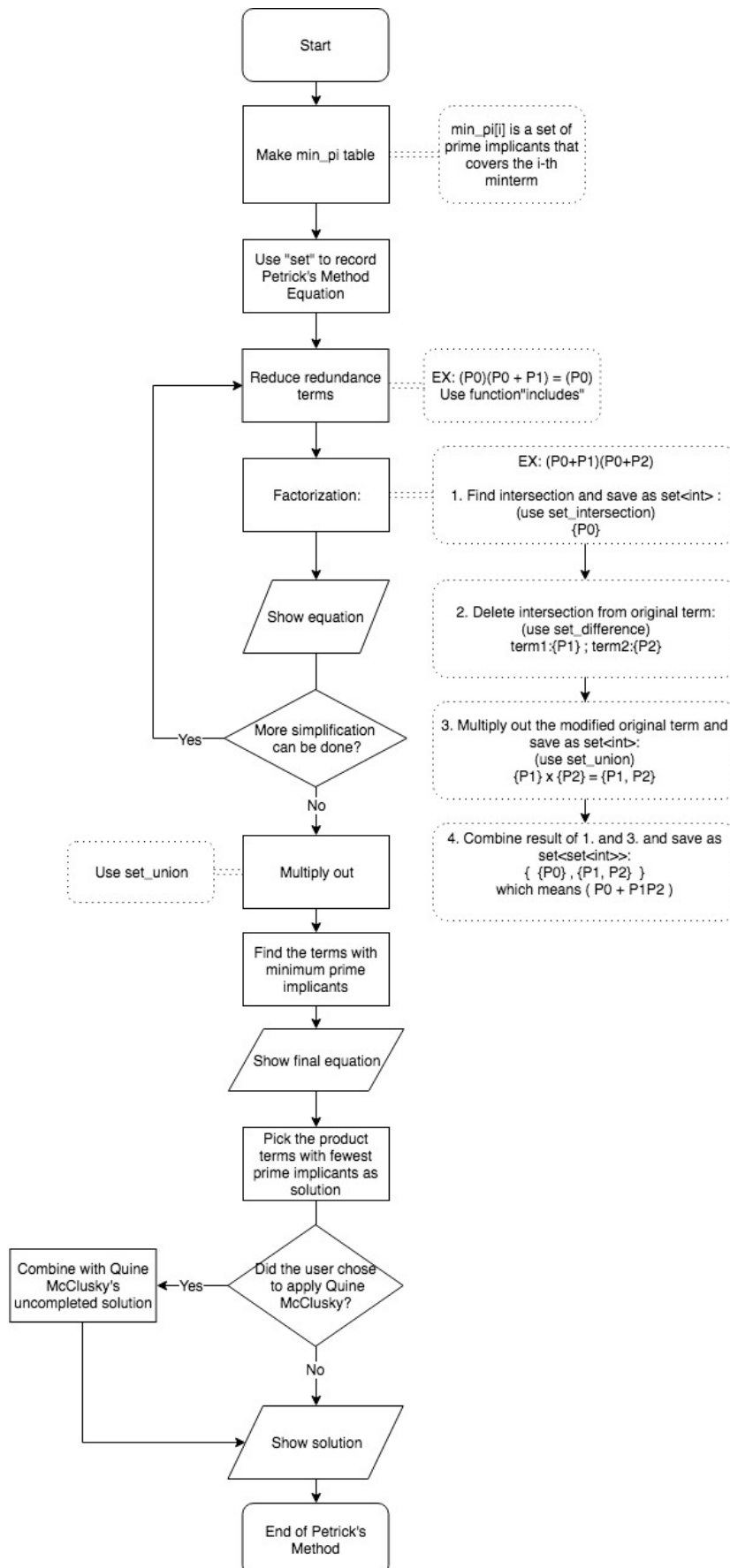
a'b'c'd'e'g'h'i' + b'd'e'f'g'h'i'j'

二、 程式撰寫流程（已省略繁瑣微小的細節）

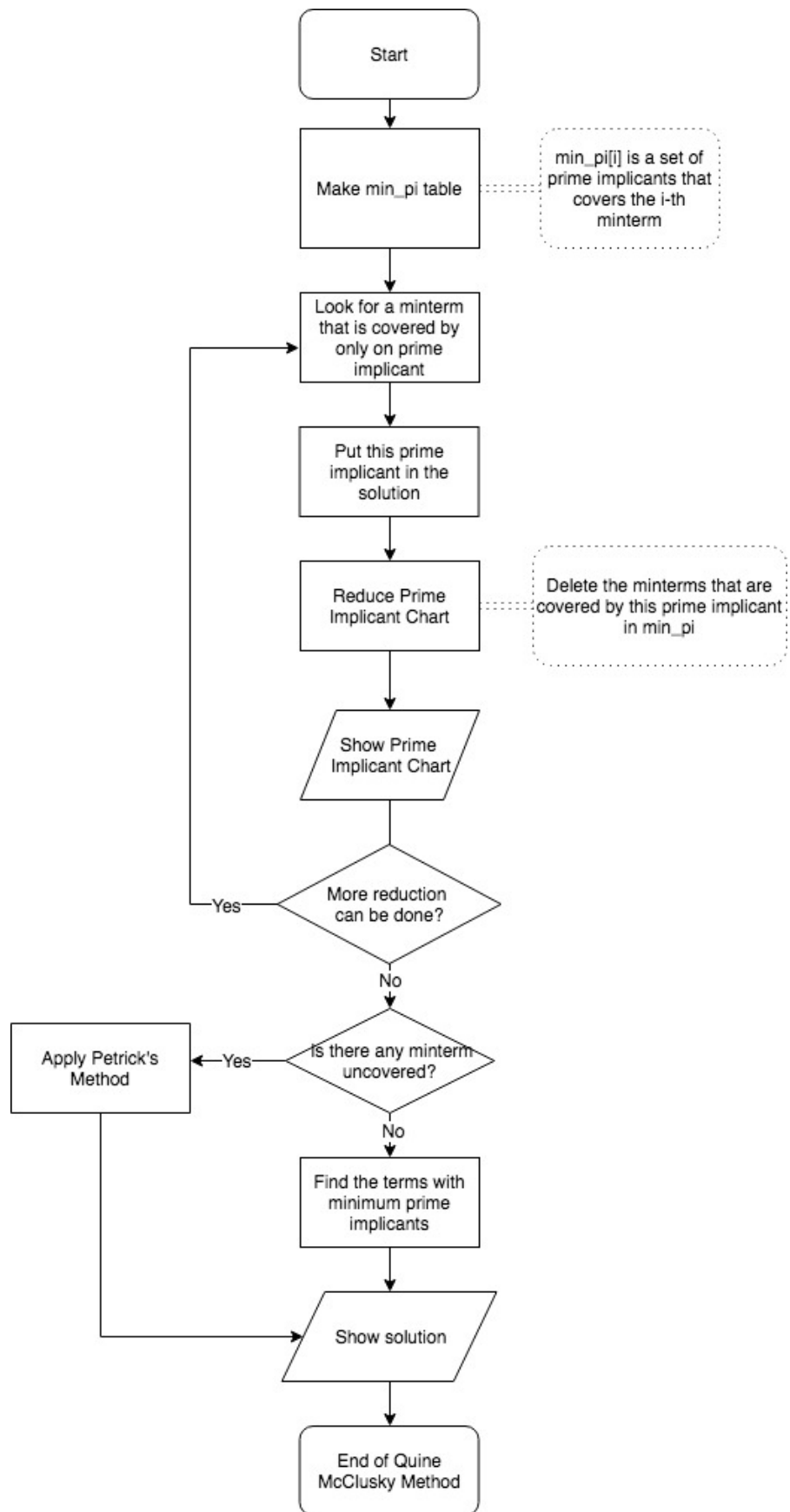
1. 主程式 Source.cpp



2. Petrick's Method



3. Quine McClasky Method



三、 作業心得

由於前次作業中，我就採用了 Quine McClusky (以下簡稱 QM) 和 Petrick's Method (以下簡稱 PM)，因此一開始以為只需對 input 讀取方法做修正和擴增變數數目即可。但修改 input 的部分之後，我發現之前所使用的方法若要擴增到 10 變數，在修改上相當麻煩。雖然只要能做出 Prime Implicant Chart，後續的 QM 和 PM 的程式碼不用改太多，但進行 Adjacent Group Combination 的前置作業卻需要大幅改寫。而且先前特別設計一個 class 來處理這個步驟，現在回頭看發現有些繁冗，所以我決定將這一部份的程式碼全部刪除並重新設計。

過程中，我有了驚人的發現——上次的程式碼竟然有 BUG！我在 Combination Process 這部分原先設計的演算法是：

1. 將每個 term 的 1 和 don't care 的 bit 位置分別用 set 存取
2. 取兩相鄰 group 的 term (1 的數目差一個)
3. 若其中一個的 binary code 的 1 所在的 bit 位置為另一個的子集 (代表兩者只差一個 1 位置不同)，且 don't care 也一樣，那麼兩者可以合併
4. 將造一個和 1 較多的 term 一模一樣的新 term，並將比較出的 1 不同的 bit 改成 don't care

但這時就必須考慮 term 中沒有 1 的特例：當 m0 和其他 m1, m2, m4 或 m8 合併，新的 term 仍然沒有 1，所以儲存 1 的 bit 位置的 set 為空集合，永遠都是任何 term 的子集，因此可和任何 term 合併，導致了錯誤的結果。因此我這次使用了 struct 來存取 combined terms，並刪除前述的 set 的比較方法，改成一個個 bit 做比較。其實這是比較直觀的作法，是自己當時將問題複雜化了。

第一份作業就使用 QM 和 PM 是因為我認為這兩種方法較有系統，較適合電腦執行，更重要的是可以擴展變數數目。但那時候設計不周，並沒有實作擴增的部分。所以這次改成讓變數數目變成可隨意更改的參數，確保這支程式不只在 10 變數，甚至是 20 變數時都能使用。另外，我也將化簡流程一步步印出，並加上顏色，希望能讓使用者能更清楚地了解步驟，。

很慶幸第一次作業時，我選擇花費大量時間完成 QM 和 PM，但我相信效能優化還有很大的進步空間，本次作業繳交後，我仍會繼續透過除錯修正，讓演算法變得更有效率。

四、 Source Code 說明

主程式	內容	
Source.cpp	使用 library	
	Library	備註
	#include "OtherFunctions.h"	所使用到的 C++ library 均寫在 OtherFunctions.h
	詳細說明	
	(參考二、1. Source.cpp 流程圖)	

副程式	內容	
OtherFunctions.h	使用 C++ library	
	Library	備註
	#include <iostream>	
	#include <sstream>	printPiChart() 中將 Cmbterms 包含的 minterms 轉為 string 時使用
	#include <fstream>	讀檔、寫檔使用到 ifstream、ofstream
	#include <cmath>	計算總共可能 minterm 數 $2^b \rightarrow \text{pow}(2, b)$
	#include <iomanip>	使用格式化輸出 $\rightarrow \text{setw}(x)$
	#include <algorithm>	取 set 交集 $\rightarrow \text{set_intersection}$ 取 set 差集 $\rightarrow \text{set_difference}$ 取 set 聯集 $\rightarrow \text{set_union}$
	#include <iterator>	使用上方三個函式時需用到 inserter
	#include <string>	
	#include <vector>	
	#include <set>	
	#include <windows.h>	改變字體顏色
	詳細說明	
	將 library、格式化輸出和寫檔函式獨立，以使主程式功能單純化	
	宣告寫檔物件 ofstream write("output.txt", ios::out ios::trunc)	
	定義 struct cmbterm (以下為所含變數與函式)	
	string binary;	儲存二進位數值
	set<int> decimal;	儲存十進位數值
	char type;	儲存 term 為 minterm ('m') 或 don't care ('d')
	bool checked;	記錄是否已被合併

	<code>void print() const{...}</code>	在 Group 的表格中格式化輸出
	<code>void printC() const{...}</code>	以英文字母形式輸出
	<code>void write() const{...}</code>	在 Group 的表格中格式化寫入 output.txt
	<code>void writeC() const{...}</code>	以英文字母形式寫入 output.txt