

9 條件式敘述、邏輯運算式與關係運算子

- 邏輯運算式
- if敘述
- switch敘述
- 條件運算式

9.1 邏輯運算式(Logical Expression)

所謂的邏輯運算式(logical expression)亦稱為布林運算式(boolean expression)但其運算結果為只有true與false兩種可能的布林值(boolean value)C語言將數值0視為false並將其它所有非0的數值視為true有三種運算子與邏輯運算式相關：關係運算子、相等運算子與邏輯運算子，分述如下：

邏輯運算與布林值是由著名數學家 [George Boole](#) 所提出，是當代電腦科學的基礎。

9.1.1 關係運算子/Relational Operator

關係運算子是一個binary運算子，用以判斷兩個運算元(數值、函式、變數或運算式)之間的關係，其可能的關係有：大於、小於、等於、或不等於C語言提供以下的關係運算子，如[table 1](#)

符號	範例	意義
>	$a > b$	a 是否大於 b
<	$a < b$	a 是否小於 b
>=	$a \geq b$	a 是否大於或等於 b
<=	$a \leq b$	a 是否小於或等於 b

Tab. 1: Relational Operators

雖然我們已經提過C語言將數值0視為false並將其它所有非0的數值視為true但關係運算子會以數值0代表運算結果為false以數值1代表true還要注意關係運算子較算術運算子的優先順序低，所以像是 $x + y < i - j$ 等同於 $(x + y) < (i - j)$ 在C語言中 $x < y < z$ 等同於 $(x < y) < z$ 因為關係運算子為左關聯。假設 $x=1, y=3, z=5$ $x < y < z \Rightarrow (x < y) < z \Rightarrow 1 < z \Rightarrow 1$

9.1.2 相等運算子/Equality Operator

相等運算子是一個binary運算子，用以判斷兩個運算元(數值、函式、變數或運算式)之值是否相等C語言提供以下的關係運算子，如[table 2](#)

符號	範例	意義
==	$a == b$	a 是否等於 b
!=	$a != b$	a 是否不等於 b

Tab. 2: Equality Operators

同樣地，相等運算子會以數值0代表運算結果為false以數值1代表true還要注意相等運算子與關係運算子

一樣，其優先順序都較算術運算子來得低。

是 `==`，不是 `=`。千萬不要將比較兩數是否相等的 `==` 寫成 `=`，這實在是一個常常會遇到的錯誤!建議您以後如果遇到程式執行結果錯誤，但找不出任何問題時，試試檢查一下所有的 `=` 與 `==`，有很高的機會可以改正您的程式。

9.1.3 邏輯運算子/Logical Operator

邏輯運算子共有以下三種，如table 3

符號	意義	unary or binary
!	NOT	unary
&&	AND	binary
	OR	binary

Tab. 3: Logical Operators

其運算結果請參考table 4的真值表：

X	Y	NOT X	X AND Y	X OR Y
0	0	1	0	0
0	1		0	1
1	0	0	0	1
1	1		1	1

Tab. 4: Truth Table

假設變數score代表c語言的修課成績，以下的邏輯運算即為檢查成績是否介於0~100：

```
((score >= 0) && (score <= 100))
```

9.1.4 優先順序(Precedence of Operators)

我們將目前為止介紹過的運算子之優先順序整理如table 5(表中是以優先權高至低依序列示)：

運算子	符號
一元運算子	+(正)、-(負) [] ++ [] -- [] !(NOT)
算術運算子(乘除)	* [] / [] %
算術運算子(加減)	+ [] -
關係運算子	> [] < [] > [] <
相等運算子	== [] !=
邏輯運算子	&& []
條件運算子	?:
指定運算子	= [] * [] / [] % [] + [] - [] =

Tab. 5: 各運算子的優先順序(由高至低)

9.2 IF敘述

當我們在程式寫作時，某些功能可能是要視情況來決定是否要加以執行的。在C語言中，提供一個if敘述，可以做到依特定條件成立與否，來決定該執行哪些程式碼。if的語法如下：

```
if ( expression ) statement
```

當expression成立時，也就是其布林值為true或是其數值不為0時，才會執行後面所接的statement。例如：

```
if (score >= 60) printf("You are pass!\n");
```

上面這行程式的意思是，如果score>=60則印出“You are pass!”。當然，若條件不成立時，後面所接的printf()是不會被執行的。如果條件成立時，想要進行的處理需要一行以上的程式碼該怎麼辦呢？請參考下面的語法：

```
if ( expression ) { statements }
```

我們可以在if敘述後，以一組大括號來將要執行的程式碼包裹起來。這種被包裹起來的程式碼又稱為複合敘述(compound statment)。請參考下面的例子：

```
if (score >= 60)
{
    printf("Your score is %d\n", score);
    printf("You are pass!\n");
}
```

如果我們想判斷的條件不只一個，那又該怎麼辦呢？其實if敘述也是敘述，所以在compound statement中當然可以含有if敘述，請參考下面的程式：

```
if (score >= 60)
{
    printf("Your score is %d\n", score);
    printf("You are pass!\n");

    if(score >= 90)
    {
        printf("You are very excellent!\n");
    }
}
```

良好的縮排(indent)習慣，可以為您的程式碼帶來容易閱讀、維護與除錯等好處。

下面是另一個例子：

```
if (score >= )
{
    if(score <= 100)
    {
```

```
    printf("This score %d is valid!\n", score);  
}  
}
```

不過這個例子，還可以改寫成：

```
if ((score >= )&&(score <=100))  
{  
    printf("This score %d is valid!\n", score);  
}
```

我曾經看過有人把程式這樣寫，雖然我可以瞭解其用意，但這個程式是錯誤的！因為relational operator是左關聯 $0 \leq \text{score} \leq 100 \rightarrow (0 \leq \text{score}) \leq 100$ 假設score為50 $\rightarrow 1 \leq 100 \rightarrow 1 \rightarrow \text{true}$ 但如果score是110，這個運算式的結果還是true

```
if ( <= score <= 100 )  
{  
    printf("This score %d is valid!\n", score);  
}
```

延續上面的例子，若是想要在score超出範圍時，印出錯誤訊息，那又該如何設計呢？請參考下面的程式：

```
if ((score >= )&&(score <=100))  
{  
    printf("This score %d is valid!\n", score);  
}  
  
if (score < 0 || score > 100)  
{  
    printf("Error! The score %d is out of range!\n", score);  
}
```

在這段程式碼中的兩個if敘述，其實是互斥的，也就是當第一個if的條件成立時，第二個if的條件絕不會成立，反之亦然。這種情況可以利用下面的語法，把兩個if敘述整合成一個：

```
if ( expression ) statement else statement
```

```
if ( expression ) { statements } else { statements }
```

else保留字可以再指定一個敘述或是複合敘述，來表明當if條件不成立時，所欲進行的處理。請參考下面的例子：

```
if ((score < 0) || (score > 100))  
{  
    printf("Error! The score %d is out of range!\n", score);  
}  
else  
{  
    printf("This score %d is valid!\n", score);  
}
```

再一次考慮到if敘述也是一種敘述，在else的後面，我們也可以再接一個if敘述，例如下面的例子：

```
if ((score < ) || (score >100))
{
    printf("Error! The score %d is out of range!\n", score);
}
else
{
    if(score>=60)
    {
        printf("You are pass!\n");
    }
}
```

類似的結構延伸，下面的程式碼也是正確的：

```
if ((score < ) || (score >100))
{
    printf("Error! The score %d is out of range!\n", score);
}
else
{
    if(score>=60)
    {
        printf("You are pass!\n");
    }
    else
    {
        printf("You are fail!\n");
    }
}
```

上述的程式碼，也可以利用if敘述及else保留字後面可以接一個敘述(只有一個敘述時，大括號可以省略)，我們可以將部份的大括號去掉，請參考下面的程式碼：

```
if ((score < ) || (score >100))
{
    printf("Error! The score %d is out of range!\n", score);
}
else if(score>=60)
{
    printf("You are pass!\n");
}
else
{
    printf("You are fail!\n");
}
```

9.2.1 IPO程式設計範例

我們以下面這一個例子做為本小節的結尾。

假設在一個售貨系統中，有一個代表銷售總金額的變數`total`，當`total`大於5000時，我們要為客戶打九五折。

現在我們來說明如何完成這個動作：首先是我們要如何判斷`total`是否大於5000？這可以用下面這個敘述來完成：

```
if( total > 5000 )
```

這一個邏輯運算式會傳回true或是false的值。接下來的問題是，如果傳回的值是true，則我們應該將`total`的值乘上0.95：

```
total = total * 0.95;
```

讓我們以IPO分析如下：

- Input Section:
 - 取得`total`
- Process Section:
 - 判斷`total`是否大於5000
 - 若是則`total*=0.95`
- Output Section:
 - 輸出`total`的值

考慮到打95折是以`*0.95`方式進行，則必須注意其結果為浮點數，現在讓我們把Declaration Section也加入IPO分析，並增加更多細節：

- Declaration Section:
 - `float total;`
- Input Section:
 - `printf("Please input the total:");`
 - `scanf("%f", &total);`
- Process Section:
 - `if(total > 5000)`
 - `total*=0.95;`
- Output Section:
 - `printf("Total=%f", total);`

完整的程式碼，很容易就可以依上面的IPO分析來完成：

[1|h total.c](#)

```
#include <stdio.h>

int main()
```

```
{
    // Declaration Section:
    float total;

    // Input Section:
    printf("Please input the total:");
    scanf("%f", &total);

    // Process Section:
    if(total > 5000 )
    {
        total*=0.95;
    }

    // Output Section:
    printf("Total=%f", total);
}
```

9.3 switch 敘述

請參考下面的資料表：

deptID	系所名稱	分機號碼
1	資訊工程系	21201
2	電腦與通訊系	21303
3	電腦與多媒體系	21701

以下的程式讓使用者輸入系所代碼後，依系所印出其分機號碼:

```
int deptID;

scanf("%ld", &deptID);

if( deptID == 1 )
{
    printf("Computer Science and Information Engineering\n");
    printf("Phone: (08)7238700 ext.21201.\n");
}
else if( deptID == 2 )
{
    printf("Computer and Communications\n");
    printf("Phone: (08)7238700 ext.21303.\n");
}
else if( dept == 3 )
{
    printf("Computer and Multimedia\n");
}
```

```
printf("Phone: (08)7238700 ext.21701.\n");
}
else
{
    printf("The value of deptID %d is invalid!\n", deptID);
}
```

上面這個程式具備「給定一個數值(或運算式)，依其值決定該執行的程式碼」的程式結構。同樣的結構C語言提供另一個選擇switch敘述，其語法如下：

```
switch (expression)
{
    case constant-expression: statements
    ...
    case constant-expression: statements
    default: statements
}
```

其執行流程是：先判斷接在switch後面的運算式的值，依照其值去執行對應的case後的敘述群。我們將其語法各個部份分別說明如下：

- switch(expression)
 - 以switch開頭，緊接一個括號與其內的expression。這裡的expression其運算結果必須為整數或字元(預設為int型態，若為char型態也會被自動轉換為int)。其它的浮點數或字串等都不被接受。
- { ... } 最後以一組大括號與其內的敘述定義不同情況下的處理方法。可以接受的敘述包含以下兩種：
 - case constant-expression : statements
 - case → 以case開頭
 - constant-expression → 所謂的constant expression即為運算式，但其運算元僅接受constant值，例如數值1, 2, 3等整數值、或'A', 'B', 'C'等字元型態(會被自動轉換成整數)或運算式 1+1, 1+2, 2+3等皆可；但不允許含有變數，例如x+y是不被接受的。
 - statements → 一行或一行以上的敘述，此部份為選擇性。當前述的expression的運算結果與這裡的constant expression運算結果相同時，則從此處開始往下繼續執行程式碼。
 - default : statements → 若前面的各個case的constant expression都不等同於switch的expression運算結果，那麼程式就跳過前述的各個case直接到default這裡執行剩下的敘述。注意default是選擇性的，也可以不寫。

現在，讓我們使用switch敘述將前述的例子重寫如下：

```
int deptID;

scanf("%ld", &deptID);

switch (deptID )
{
    case 1:
        printf("Computer Science and Information Engineering\n");
        printf("Phone: (08)7238700 ext.21201.\n");
        break;

    case 2:
        printf("Computer and Communications\n");
```



```

        printf("Phone: (08)7238700 ext.21303.\n");
        break;
    case 3:
        printf("Computer and Multimedia\n");
        printf("Phone: (08)7238700 ext.21701.\n");
        break;
    default:
        printf("The value of deptID %d is invalid!\n", deptID);
}

```

請將上述程式碼編輯、編譯並加以執行，看看結果為何？注意，在這個程式中，每個case的敘述後都加了一個break敘述，其作用是讓程式的執行跳離其所屬的程式區塊中，一個程式區塊(block)是一組左右對稱的大括號與其內的敘述所組成。假設deptID的輸入值為2，請參考figure 1，紅色粗線即為程式執行的動線，當deptID等於2時，程式的執行會跳過一部份，直接到case 2:的地方再加以執行，直到遇到break時，則跳出所屬的程式區塊，意即結束了這個switch敘述的執行。

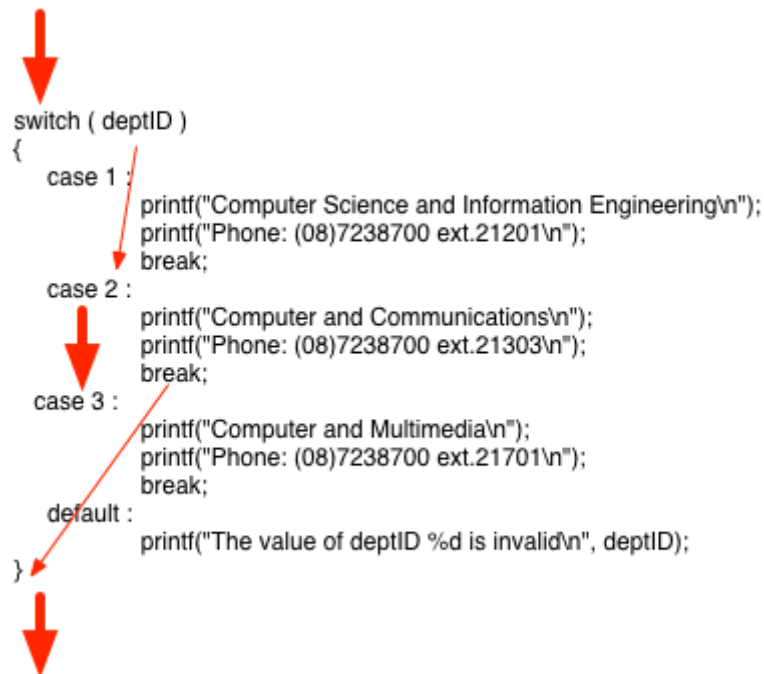


Fig. 1

figure 2則顯示了不使用break的情況下，其程式執行的動線。因為沒有break，所以程式從case 2開始執行後就會一直執行到底，其輸出結果為：

```

2
Computer and Communications
Phone: (08) 7238700 ext.21303.
Computer and Multimedia
Phone: (08) 7238700 ext.21701.
The value of deptID 2 is invalid!

```

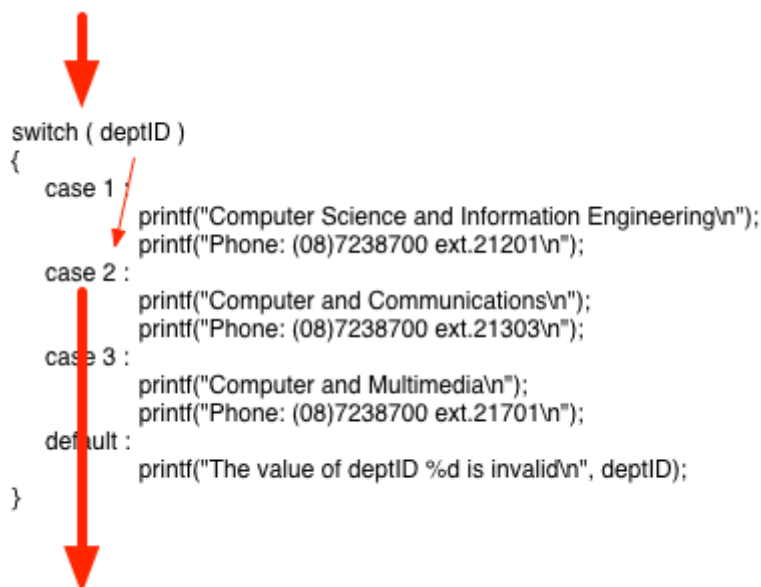


Fig. 2

當程式具有類似處理需求時，使用switch敘述將可以讓程式的架構更為簡單易讀。請再思考看看，還有哪些應用適合使用switch敘述呢？

- 程式提供操作選項，而各選項負責執行不同的功能:

```

switch (choice)
{
    case 'i':
        insert_data();
        break;
    case 'x':
        execute();
        break;
    case 'q':
        exit();
        break;
}

```

- 國小學生週一至週五，每天下課時間不同，有時半天、有時整天：

```

switch (weekday)
{
    case 1:
    case 2:
    case 4:
    case 5:
        printf("After school at 4:00pm\n");
        break;
    case 3:
        printf("After school at 12:00am\n");
}

```

- 設計一程式，輸入一整數N，計算並印出1+2+...+N的結果：

```
int n=, sum=;
scanf("%d",&n);

switch (n)
{
    case 10: sum+=10;
    case 9: sum+=9;
    case 8: sum+=8;
    case 7: sum+=7;
    case 6: sum+=6;
    case 5: sum+=5;
    case 4: sum+=4;
    case 3: sum+=3;
    case 2: sum+=2;
    case 1: sum+=1;
}
printf("Sum=%d\n", sum);
```

9.4 條件運算式(Conditional Expression)

C語言還有提供一種特別的運算式，稱為**條件運算式(conditional expression)**，可依條件決定運算式的傳回值，其語法如下：

```
expression1 ? expression2 : expression3
```

運算式的運算結果expression1的值為true(非0的數值)或false(數值0)而定，當expression1為true時，傳回expression2的值；否則當expression1為false時，傳回expression3的值。事實上，這等同於下面的if敘述：

```
if (expression1)
    result = expression2;
else
    result = expression3;
```

條件式敘述有可能是因為像「如果expression為真則.... 否則....」這樣的述敘，在程式中出現的機會很高的緣故吧！請參考以下的應用：

```
int x=1, y=2, z;
if(x>y)
    z=x;
else
    z=y;
```

上面這段程式碼是令z為x與y兩者中較大的值，如果以條件運算式改寫，則只要寫成

```
z= x>y ? x: y;
```

即可，是不是簡化很多？下面這行程式，假設score為學生成績，則可以簡單地檢查score是否大於100，若超過100則以100分計。

```
score = score > 100 ? 100 : score ;
```

現在讓我們想想下面的運算式在做些什麼呢？

```
x = (x%10) != 0 ? (x-x%10+10) : x;
```

From:

<http://junwu.nptu.edu.tw/dokuwiki/> - Jun Wu的教學網站 國立屏東大學資訊工程學系

Permanent link:

<http://junwu.nptu.edu.tw/dokuwiki/doku.php?id=c:ifcase>

Last update: **2016/05/28 20:19**

