自定函數與副程式

資料來源：http://www.tcgs.tc.edu.tw/~sagit/cpp/q7.htm

一、自定函數

在提到自定函數(Function)之前，我們先來看一個問題，就是要印出 1 到 100 之間所有的質數。假設有一個函數 P(n) 可以判斷 n 是不是質數，若是則傳回 1，否則傳回 0。我們的程式可以寫成：

```
int i;
for (i=1; i<=100; i++)
    if(P(i)) cout << i << endl;
```

但是事實上並沒有一個這樣的函數，因此我們要自己定義這個函數，語法如下：

```
傳回值型態 函數名稱(型態 參數 1, 型態 參數 2, ....)
{
    // 變數宣告
    // 程式碼
    return 傳回值;
}
```

其中 return 語法的功能是讓函數把一個值傳回給呼叫它的程式，也就是說如果函數執行到這一行，後面的程式碼就不會再執行了。而上面函數的寫法是不是和 main() 很像，事實上 main() 本身也是一個函數，只不過 main() 是一個特別的函數，它會在程式開始的時候就自動執行。而要讓 main() 可以使用這個新的函數，要把這個函數定義寫在 main() 之前，整個程式如下：

```
#include <iostream>
using namespace std;
int P(int n)
{
    int i;
    if(n<2) return 0;
    for (i=2; i<n; i++)
        if( (n%i)==0 ) return 0;
    return 1;
}
int main(){
    int i;
    for (i=1; i<=100; i++)
        if (P(i)) cout << i << endl;
    system("PAUSE");
    return 0;
}
```

上面來藍色的部分即是函數 P( ) 的定義。而我們也可以把定義寫在 main() 的後面，不過還是要在 main() 前面加上函數 P( ) 的宣告，即變成：

```
#include <iostream>
using namespace std;
int P(int n);
int main()
{
	int i;
	for (i=1; i<=100; i++)
		if (P(i)) cout << i << endl;
	system("PAUSE");
	return 0;
}
int P(int n)
{
	int i;
	if(n<2) return 0;
	for (i=2; i<n; i++)
		if( (n%i)==0 ) return 0;
	return 1;
}
```

也就是把函數定義的第一行寫上去，後面加一個分號，而函數括號裡的參數，只要寫變數型態即可，變數名稱可以省略，例如：

```
int P(int);
```

函數宣告的功能，在於告訴 C/C++ 的編譯器有一個這樣的函數，它的名稱、參數以及傳回值，編譯器才有辦法處理。而函數定義的同時也就是宣告，在宣告之後才可以使用這些函數。另外，我們程式一開頭的 #include 指令所加入的標頭檔，它裡面也是一大堆的函數宣告，也因此加入這些標頭檔就可以使用這些函數。

至於函數要定義在 main() 之前還是之後，這個見仁見智，寫在前面的優點是不用再宣告一次，寫在後面的優點是 main() 主程式在前面，可以很快看懂這個程式(因為函數只要知道它的功能即可，不用去管它是如何寫出來的)。

上面的例子雖然不一定要用函數的寫法，但是使用自定函數有下列的好處：

1. 讓主程式 main() 更簡潔
2. 可以分工完成各函數後再合併起來

最後，這先把程式分成幾個大步驟，再將這些大步驟分解成數個小步驟的寫法，我們稱之為「由上而下的寫法」(Top-Down)，有助於讓一個複雜的問題簡單化。

二、副程式

有時候，我們只是希望執行某個動作，並不是要它傳回某個數，例如印 M 個空格，或是印 N 個 # 符號，這時候我們可以把它們設計成一個副程式。在其他程式語言(如 BASIC、PASCAL)裡，副程式叫做 Procedure 或 Subroutine，而在 C/C++ 裡，它和函數(Function)是一樣的，差別只在於它不需要傳回值。我們可以設定它的傳回值為 void 即可，而 return 時後面也不需要接任何資料(也不一定要加 return)，例如：

```
void 副程式名稱(型態 參數 1, 型態 參數 2, ....)
{
    // 變數宣告
    // 程式碼
    return;
}
```

當然，你也可以設定傳回的型態為 int，而 return 時用 0 傳回也是可以。上面我們提到的兩個副程式可以寫成：

```
void A(int n)
{
    int i;
    for(i=0; i<n; i++)
        cout << " ";
}
void B(int n)
{
    int i;
    for(i=0; i<n; i++)
        cout << "#";
}
```

這樣，只要呼叫 A(3)、B(5) 即會印出三個空格以及五個 # 符號。我們也可以把這兩個副程式改成一個給它要印的字元和次數的副程式：

```
void C(int n, char c)
{
    int i;
    for(i=0; i<n; i++)
        cout << c;
}
```

這樣，只要呼叫 C(3, ' ')、C(5, '#') 即可達到前面 A(3)、B(5) 的功能。

三、全域變數(Global Variable)與區域變數(Local Variable)

在講解全域變數與區域變數之前，我們先來看一個程式：

```
void P()
{
	int a;
	a=3;
}
int main()
{
	int a;
	a=2;
	P( );
	cout << a << endl;
	system("PAUSE");
	return 0;
}
```

注意上面程式紅色的部分，我們在 main()、P() 兩個函數中同時用到了 a 這個變數名稱，在 main() 中我們先執行了 a=2，接下來呼叫 P() 函數，而 P() 裡面又執行了 a=3，回到 main() 之後我們把 a 印出來，這時候會印出 2 還是 3 呢？答案是 2，因為 main() 裡的 a 和 P() 裡的 a 是不同的變數。

像上面這樣把變數宣告在函數內部的，我們稱之為區域變數(Local Variable)，它的有效範圍(Scope)只限於它所屬的函數內部，不能跨越另一個函數。而如果我們希望有一個變數可以被多個函數共同使用，則可以把它寫在函數外面，例如：

```
int a=0;
void P()
{
	a=3;
}
int main()
{
	a=2;
	P( );
	cout << a << endl;
	system("PAUSE");
	return 0;
}
```

這種變數我們稱之為全域變數(Global Variable)，它可以被該行變數宣告之後的所有函數使用。而上面程式中的 main() 和 P() 中的 a 是同一個變數，因此印出的即是 3。

而如果我們在函數內部以及外部使用了相同名稱的變數，這時候如此區分區域變數以及全域變數呢？系統會以區域變數為優先，而如果我們要使用全域變數，可以在變數名稱前面加上兩個冒號 ::，例如：

```
int a=1;
int main()
{
	int a=2;
	cout << a << endl; //  區域變數
	cout << ::a << endl; //  全域變數
	system("PAUSE");
	return 0;
}
```

為了避免因為變數名稱相同造成的誤用，一些程式語言的專家建議減少全域變數的使用。然而因為區域變數是使用程式的堆疊，在 Win32 下約只有 2MB 的空間，如果使用到很大的陣列則會造成堆疊使用完畢的問題，因此建議如果要使用超大陣列，請將之宣告成全域變數以避免這個問題。

四、傳值呼叫與傳參照呼叫

前面提到的函數和副程式的寫法，如果我們改變了傳進去的參數值，原本的值並不會改變，這種方式我們稱為「傳值呼叫」，例如：

```
void A(int a)
{
	a=2;
}
int main( )
{
	int a=1;
	A(a);
	cout << a << endl;
	system("PAUSE");
	return 0;
}
```

上面的程式會印出 1 而不是 2。而如果我們希望函數或副程式可以改變原本傳進去的參數值，則可以在參數前面加上 & 符號，例如：

```
void A(int &a){
	a=2;
}
int main( ){
	int a=1;
	A(a);
	cout << a << endl;
	system("PAUSE");
	return 0;
}
```

這時候印出來的就會變成 2 了。這種方式，我們稱為「傳參照呼叫」。我們可以在同一個函數傳進去的不同參數各別設定是傳值或是傳參照，例如：

```
void B(int a, int b, int &c)
{
    ...
}
```

上面的函數(副程式) B 的參數 a、b 都是傳值呼叫，而 c 是傳參照呼叫。要注意的是，如果是傳值呼叫，呼叫時可以給它一個常數、變數或運算式，例如：A(0)、A(a)、A(a+2) ；但是如果是傳參照呼叫，則一定要是一個變數才可以，例如： B(a) 。

五、練習題

1. Q100：The 3n + 1 problem
2. Q488：Triangle Wave
3. Q10018：Reverse and Add

Q100: The 3*n* + 1 problem

Background

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

The Problem

Consider the following algorithm:
1. input n
2. print n
3. if n = 1 then STOP
   4. if n is odd then n <- 3n+1
   5. else n <- n/2
6. GOTO 2

Given the input 22, the following sequence of numbers will be printed 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that 0 < n < 1,000,000 (and, in fact, for many more numbers than this.)

Given an input n, it is possible to determine the number of numbers printed (including the 1). For a given n this is called the cycle-length of n. In the example above, the cycle length of 22 is 16.

For any two numbers i and j you are to determine the maximum cycle length over all numbers between i and j.

The Input

The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than 1,000,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including i and j.

You can assume that no operation overflows a 32-bit integer.

The Output

For each pair of input integers i and j you should output i, j, and the maximum cycle length for integers between and including i and j. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers i and j must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input

1 10

100 200

201 210

900 1000

Sample Output

1 10 20

100 200 125

201 210 89

900 1000 174

488 - Triangle Wave

In this problem you are to generate a triangular wave form according to a specified pair of Amplitude and Frequency.

Input and Output

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is **followed by a blank line**, and there is also a blank line between two consecutive inputs.

Each input set will contain two integers, each on a **separate line.** The first integer is the Amplitude; the second integer is the Frequency.

For each test case, the output must follow the description below. The outputs of two consecutive cases will be **separated by a blank line**.

For the output of your program, you will be printing wave forms each separated by a blank line. The total number of wave forms equals the Frequency, and the horizontal ``height'' of each wave equals the Amplitude. The Amplitude will never be greater than nine.

The waveform itself should be filled with integers on each line which indicate the ``height'' of that line.

NOTE: There is a blank line after each separate waveform, excluding the last one.

Sample Input

```
1

3
2
```

Sample Output

```
1
22
333
22
1

1
22
333
22
1
```

Q10018: Reverse and Add

The Problem

The "reverse and add" method is simple: choose a number, reverse its digits and add it to the original. If the sum is not a palindrome (which means, it is not the same number from left to right and right to left), repeat this procedure.

For example:
195 Initial number
591

786
687

1473
3741

5214
4125

9339 Resulting palindrome

In this particular case the palindrome 9339 appeared after the 4th addition. This method leads to palindromes in a few step for almost all of the integers. But there are interesting exceptions. 196 is the first number for which no palindrome has been found. It is not proven though, that there is no such a palindrome.

Task :

You must write a program that give the resulting palindrome and the number of iterations (additions) to compute the palindrome.

You might assume that all tests data on this problem:
- will have an answer ,
- will be computable with less than 1000 iterations (additions),
- will yield a palindrome that is not greater than 4,294,967,295.

The Input

The first line will have a number N with the number of test cases, the next N lines will have a number P to compute its palindrome.

The Output

For each of the N tests you will have to write a line with the following data : minimum number of iterations (additions) to get to the palindrome and the resulting palindrome itself separated by one space.

Sample Input

```
3
195
265
750
```

Sample Output

```
4 9339
5 45254
3 6666
```