

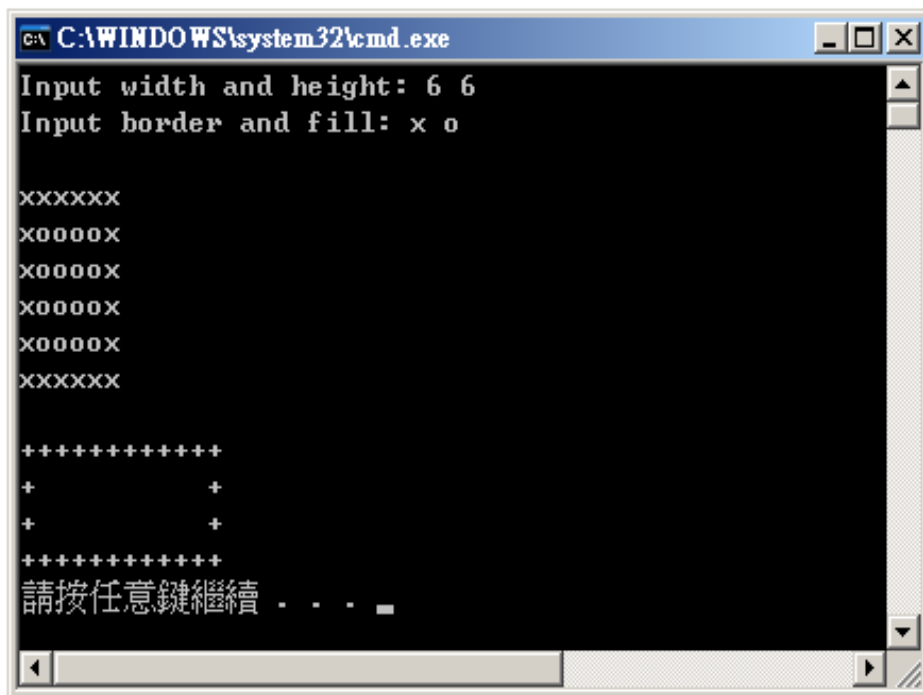
Homework Assignment No. 4

(Total 60% + 30% Bonus, 每題 30%, 任挑兩題做, 有時間再做 Bonus)

1. (30%): Write a function `draw_box` with the following declaration:

```
void draw_box(int width, int height, char border='+',  
char fill=' ');
```

The input parameters are the width dimension, height dimension, border and fill characters for a rectangular box. You then write a program that calls the function and draws two boxes. The first box will be drawn as specified by the user inputs; the second will be twice as wide, two rows shorter, and use the `draw_box` function's default border and fill characters, '+' and ' ', respectively. Below is a sample run:



2. (30%) Write a program that converts rectangular coordinates to polar and vice versa. To convert polar coordinates (r, θ) to rectangular coordinates (x, y) and vice versa, use the following equations and function declaration:

$$x = r \cos \theta$$

$$y = r \sin \theta$$

```
| void PolarToRectangular(double r, double t, double& x, double& y);
```

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \tan^{-1} \frac{y}{x}$$

```
| void RectangularToPolar(double x, double y, double& r, double& t);
```

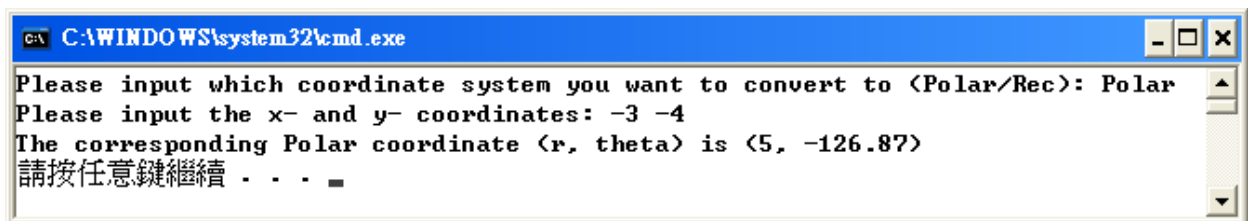
Important: some descriptions on math library and atan2

The `atan2()` function from the math library (`cmath`) calculates the angle (in radius) from the `x` and `y` values:

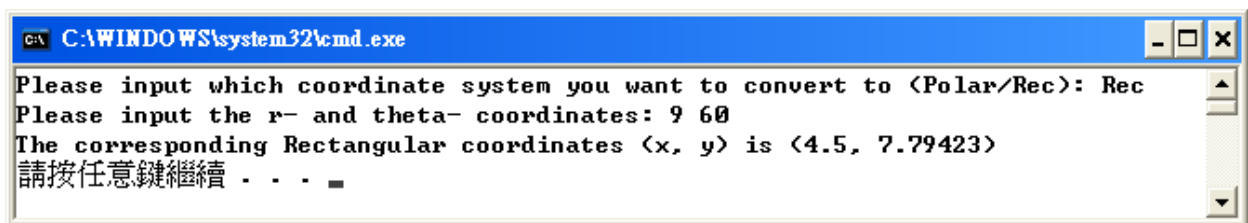
```
| angle = std::atan2(y, x)
```

There is also an `atan()` function, but it doesn't distinguish between angles 180 degrees apart. That uncertainty is not desirable.

Below are typical runs:



```
C:\WINDOWS\system32\cmd.exe
Please input which coordinate system you want to convert to <Polar/Rec>: Polar
Please input the x- and y- coordinates: -3 -4
The corresponding Polar coordinate <r, theta> is <5, -126.87>
請按任意鍵繼續 . . .
```



```
C:\WINDOWS\system32\cmd.exe
Please input which coordinate system you want to convert to <Polar/Rec>: Rec
Please input the r- and theta- coordinates: 9 60
The corresponding Rectangular coordinates <x, y> is <4.5, 7.79423>
請按任意鍵繼續 . . .
```

3. (30%) We can implement a home-made square root using a simple Newton iteration.
Given $a > 0$, a simple Newton iteration to find \sqrt{a} is:

$$x_{k+1} = \frac{1}{2} \left(x_k + \frac{a}{x_k} \right) \quad x_0 = a$$

Implement this home-made square root function with the following function declaration:

```
| void sqrt(double a, double tol)
```

in which `tol` is the tolerance for convergence ($\frac{|x_{k+1} - x_k|}{|x_{k+1}|} < \text{tol}$). Print the iteration number

$k+1$ and x_{k+1} . Below is a sample run:

```
Please input the number which you want to find square root and the tolerance:
2 0.001
1: 1.5 0.333333
2: 1.41667 0.0588235
3: 1.41422 0.0017331
4: 1.41421 1.50182e-06
Please input the number which you want to find square root and the tolerance:
2 0.01
1: 1.5 0.333333
2: 1.41667 0.0588235
3: 1.41422 0.0017331
Please input the number which you want to find square root and the tolerance:
5 0.0001
1: 3 0.666667
2: 2.33333 0.285714
3: 2.2381 0.0425532
4: 2.23607 0.000906208
5: 2.23607 4.10606e-07
Please input the number which you want to find square root and the tolerance:
```

HW Grading Policy:

1. You should consider about exception handling, e.g. error input, file opening fail, etc.
請注意所有例外狀況的處理，例如：錯誤的符號字串輸入、檔案開啟失敗等。
2. The coding style includes your output format.
輸出資料的格式將納入格式評分。
3. If your code is not compilable, your score in this problem is zero (including coding style).
若程式無法編譯，則該題以零分計算。(包含格式分數)
4. Your program will be tested with other data which is not the same as provided samples.
除了題目所提供的範例測試資料以外，作業程式碼將以額外的測試資料進行測試。

- Coding Style (20%): 編碼格式分數

1. format
整體形式與輸出資料的格式
2. comments
註解
3. readability
可讀性

4. variables naming

變數命名方式

5. typesetting

型別設定

- Functionality (80%): 功能性分數

1. run-time performance:

執行時的表現

1) samples not passed -> x

範例測資錯誤 => 此部分零分

2) samples passed but some tests failed -> partial

範例測資通過但是部分測資失敗 => 部份給分

3) samples and tests all passed

範例測資與所有測資通過 => 此部分滿分

3. excellent method++

綜合以上，又以能展現解決問題的巧思尤佳。