作業一: 程式執行時間複雜度分析

問題說明:

為了解決一個問題而設計程式時, 分析該演算法的執行時間複雜度是個很重要的評估依據。例如線性時間的演算法通常要比二次方時間的演算法受歡迎, 因為執行程式需要的時間在比較大的 *n* 時線性比二次方少很多。

通常問題的大小 *n* 可以決定演算法的執行時間, 例如 *n* 是被排序的數字個數, 或是多邊形的點的數目, 數字的位元數等等。由於要算出一個演算法相對於 *n* 的執行時間公式不是很容易,對於一般的程式來說是不太可能做到的, 但是如果我們只考慮非常簡單的程式, 自動替它計算運算時間複雜度就是可行的了。這個作業中考慮的程式是根據下面的規則 (BNF格式, 例如 C程式的BNF語法) 所建立的, 其中 ***number*** 及 ***float-number*** 是大於等於零的十進位整數及浮點數。

程式語法定義

1. ***Program*** ::= BEGIN ***Statementlist*** END
2. ***Statementlist*** ::= ***Statement*** | ***Statement Statementlist***
3. ***Statement*** ::= ***LOOP-Statement*** | ***OP-Statement***
4. ***LOOP-Statement*** ::= ***LOOP-Header Statementlist*** END
5. ***LOOP-Header*** ::= LOOP ***number*** | LOOP n
6. ***OP-Statement*** ::= OP ***float-number***

上面這六個語法中 BEGIN, END, LOOP, LOOP, n, OP 為關鍵字, 符合上面的語法描述的組合可以定義出一種語言, 舉例說明如下:

```
BEGIN
  OP 1
  OP 2
  OP 3
END
```

這個程式符合上面的語法, 解釋方法如下:

1. 首先由第*1*個語法可以看到: 程式就是一個*Program*, 由 BEGIN, END 以及包在中間的敘述串列 (*Statementlist*) 組成, 這個例子裡 *Statementlist* 就是指 OP 1, OP 2, 和 OP 3 三個敘述
2. 由第*2*個語法可以看到: 每一個敘述串列 (*Statementlist*) 可以是一個單一敘述 (*Statement*)或是一個敘述 (*Statement*)再串接一個敘述串列 (*Statementlist*), 這個例子裡 OP 1是單一敘述, OP 2 和 OP 3 是一個敘述串列 (*Statementlist*), 進一步再運用語法*2*可以把 OP 2 看成一個單一敘述, OP 3看成是是一個敘述串列 (*Statementlist*), 最後再運用一次語法*2*把OP 3 看程式一個單一敘述
3. 由第*3*個語法可以看到:每一個單一敘述(*Statement*)要不是迴圈敘述(*LOOP-Statement*), 就是運算敘述(*OP-Statement*), 此例中OP 1, OP 2, 或是 OP 3都是運算敘述
4. 由語法*6*可以看到每一個運算敘述(*OP-Statement*)都由關鍵字OP後面接一個浮點數來表示

程式語法解說

只要整個程式的每一部分都可以用這六個語法一層一層的描述, 這個程式就符合語法

再看另外一個例子

```
BEGIN
  LOOP n
    OP 1
  END
  OP 2
END
```

這個程式也符合上面的語法, 解釋方法如下:

1. 首先由第*1*個語法可以看到: 程式就是一個 *Program*, 由 BEGIN, END 以及包在中間的敘述串列 (*Statementlist*) 組成, 這個例子裡 *Statementlist* 就是指 OP 1, LOOP n, OP 2 和 END 這幾個敘述
2. 由第*2*個語法可以看到: 每一個敘述串列 (*Statementlist*) 可以是一個單一敘述 (*Statement*)或是一個敘述 (*Statement*)再串接一個敘述串列 (*Statementlist*), 這個例子裡 LOOP n OP 1 END是一個迴圈敘述, OP 2 是敘述串列 (*Statementlist*), 進一步再運用語法*2*可以把 OP 2 看成一個單一敘述
3. 由第*3*個語法可以看到:每一個單一敘述(*Statement*)要不是迴圈敘述(*LOOP-Statement*), 就是運算敘述(*OP-Statement*), 此例中LOOP n OP 1 END 是迴圈敘述, OP 2 是運算敘述
4. 由語法*4*可以看到每一個迴圈敘述(*LOOP-Statement*)都包括迴圈標頭(*LOOP-Header*), 敘述串列 (*Statementlist*), 以及關鍵字 END組成
5. 由語法*5*可以看到每一個迴圈標頭要不是 LOOP *number* 就是LOOP n 兩種, 此例中是後者
4. 由語法*6*可以看到OP 2這一個運算敘述(*OP-Statement*)是由關鍵字OP後面接浮點數 2 來表示

程式執行時間複雜度

前面這個語法定義的程式的執行時間複雜度以下列方法計算：

- 運算敘述 *OP-Statement* 的執行時間就跟它的參數一樣
- 迴圈敘述 *LOOP-Statement* 內部的敘述串列會執行多次:
  有可能會執行常數次 (如果 LOOP 關鍵字後面跟著的參數是常數), 或是執行 *n* 次 (如果 LOOP 關鍵字後面跟著的參數是 *n*), 忽略迴圈控制變數的加法以及比對所需要的時間, 所以空的迴圈的執行時間當成是0 (LOOP *n* END)
- 一個敘述串列 *StatementList* 的執行時間等於構那個敘述串列所有單一敘述 *Statement* 的執行時間的總和

因此程式裡如果有重複執行 *n* 次的迴圈敘述, 執行時間就會跟 *n* 有關係, 是一個 *n* 的多項式

程式輸入與輸出

程式輸入：

空白字元以及換行可能會出現在程式中的任何地方, 但不會出現在關鍵字或是數字之間, 為了簡化起見, 關鍵字一定是正確的, 比如 BEGIN, END, LOOP, n, OP; 迴圈可能有內層的迴圈, 最大深度只會到 10 ; 輸入程式的語法保證一定是正確的。

程式基本輸出：

程式的執行時間, 這會是一個跟 *n* 有關的多項式, 最大的次數會到 10。用平常表示多項式的方法印出來, 格式如下：

執行時間 = *c10*n^10+ ⋯+c2*n^2+c1*n^1+c0*

省略係數是 0 的項次, 係數為 1 者只需要印 *n^k*

如果執行時間是 0, 請印出

執行時間 = 0

由於語法中規定的是浮點數, 所以上面描述中 “係數是 0” 的意思指係數在 $[-10^{-6},10^{-6}]$ 區間中; “係數是 1” 則是指係數在 $[1-10^{-6},1+10^{-6}]$ 區間中

輸入輸出範例

輸入	輸出

BEGIN
  LOOP n
  OP 4
  LOOP 3
  LOOP n
  OP 1
  END
  OP 1.5
  OP 2
  END
  OP 1
  LOOP n
  OP 0.5
  END
  END
  OP 17
END
	執行時間 = 3.5 * n^2 + 15.5 * n + 17
	Postorder: OP-Stmt 4.00, OP-Stmt 1.00, ...
	Inorder: LOOP-Stmt1, OP-Stmt 4.00 ...

BEGIN OP 196 LOOP n LOOP n OP 1 END END OP 401 END	執行時間 = n^2 + 597 Postorder: OP-Stmt 196.00, OP-Stmt 1.00,… Inorder: OP-Stmt 196.00, Stmtlist, …

請參考範例執行程式
以及下面兩頁說明

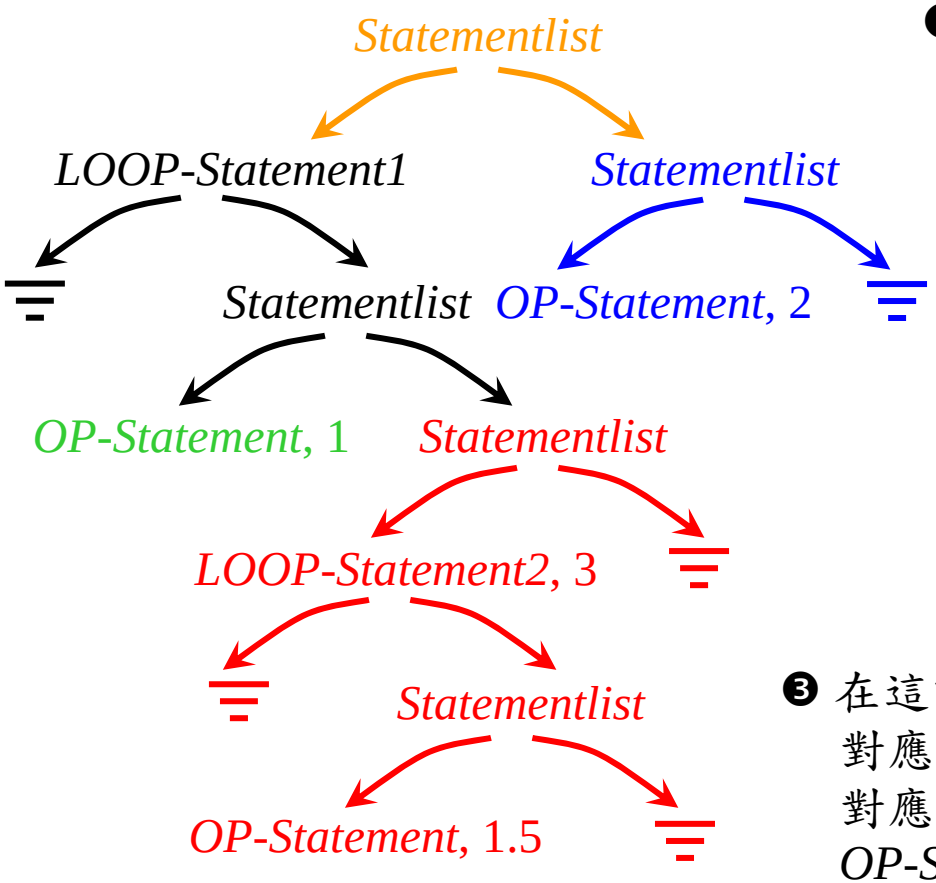
範例程式與基本測試資料

- complexity03.exe
  請在命令列視窗中執行 complexity03 < testComp01.dat
  或是
  執行時請將下列檔案內的程式貼進去 (或是由鍵盤輸入程式)
- testComp01.dat
- testComp02.dat
- testComp03.dat

樹狀資料結構輸出

- 除了上述輸出的 *n* 的多項式之外, 因為我們希望這個作業在學期中以後, 可以和用C++物件導向設計方法的作業四比較, 所以還需要你的程式以 postorder 和 inorder 輸出一個樹狀資料結構
- 以下圖左的程式為例, 你需要讓你的程式自動建立下面的樹狀資料結構

```
BEGIN
  LOOP n
    OP 1
    LOOP 3
      OP 1.5
    END
  END
  OP 2
END
```

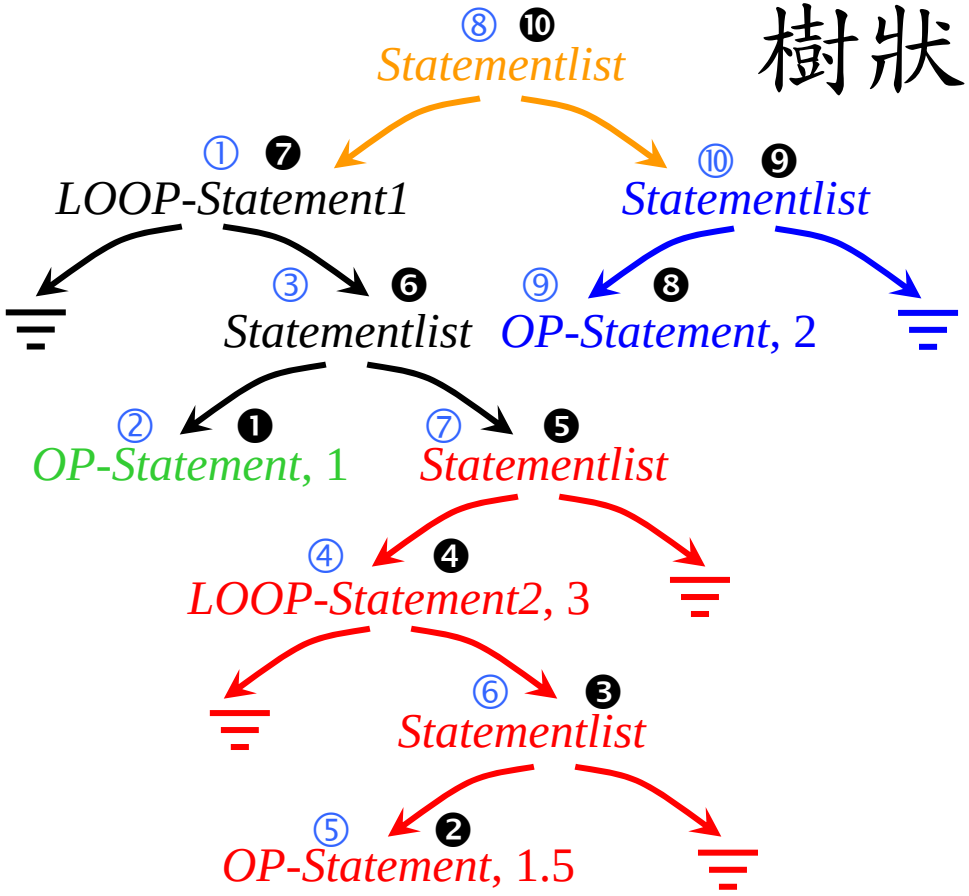


❶ 這是一個二元樹, 每一個節點包括下列兩個資料欄位
1. 節點型態 node type
2. 對應參數 parameter

❷ node type 有下列四種, 請以整數或是 enum 型態表示: 1. *Statementlist*
2. *LOOP-Statement1*
3. *LOOP-Statement2*
4. *OP-Statement*

❸ 在這四種節點裡面, 前兩種沒有對應的參數, *LOOP-Statement2* 對應的 parameter 是一個整數 *OP-Statement* 對應的parameter 是一個浮點數

樹狀資料結構輸出




```
BEGIN
  LOOP n
    OP 1
    LOOP 3
      OP 1.5
    END
  END
  OP 2
END
```

(縮寫一下: *Statement* => *Stmt*)

以上面這個例子來說, postorder 輸出的順序是黑色數字❶~❿標示的順序: 請依序印出

Postorder: *OP-Stmt* 1.00, *OP-Stmt* 1.50, *Stmtlist*, *LOOP-Stmt2 3*, *Stmtlist*, *Stmtlist*, *LOOP-Stmt1*, *OP-Stmt* 2.00, *Stmtlist*, *Stmtlist*

Inorder輸出的順序是藍色數字①~ ⑩標示的順序: 請依序印出

Inorder: *LOOP-Stmt1*, *OP-Stmt* 1.00, *Stmtlist*, *LOOP-Stmt2* 3, *OP-Stmt* 1.50, *Stmtlist*, *Stmtlist*, *Stmtlist*, *OP-Stmt* 2.00, *Stmtlist*

其他程式要求

- 請以 C 語言撰寫, 確定Visual C++ 2010 可以正確編譯執行
- 多項式請定義一個結構儲存其係數以及次數, 請撰寫四個獨立函式完成多項式的加法, 多項式乘 n, 多項式乘常數,以及多項式列印並且置於 polynomial.cpp 檔案中
- 前頁的樹狀資料結構請定義一個節點結構, 樹狀資料結構的inorder, postorder 巡訪,列印,以及記憶體釋放請撰寫函式置於 parsetree.cpp 檔案中
- 語法的解析請針對每一個語法撰寫一個函式來完成, 這些函式請置於 syntax.cpp 檔案中
- 請撰寫函式完成鍵盤資料讀取, 置於 io.cpp 檔案中
- main 函式請置於 main.cpp 檔案中
- 請以 memory_leak.h 及 memory_leak.cpp 檢測程式是否有記憶體未釋放
- 變數以及函數請適當命名, 不可使用全域變數
- 程式繳交時間, 104/03/19 (四) 21:00

補充說明

- 看了那麼多的說明, 也許有一種有看沒有懂的感覺, 不要過於擔心, 請趕快提出你的問題, 程式作業不是考試, 也不是要找你麻煩, 只是給你一個目標, 希望你運用你所有資源去整體地學習, 快速地增進你的程式設計能力
- 這個練習裡面主要運用的是程式設計和資料結構, 以及一開始在實習課裡練習的多檔案與記憶體遺失測試, 你不需要先看很多C++或是物件導向的東西; 也許你覺得程式設計和資料結構都不熟悉那怎麼辦...時間還夠, 也有同學、助教、老師可以問, 只要你開始寫, 就有機會可以針對你所需要的知識提出問題, 針對這個作業所需要的去了解, 就足夠完成這個作業了
- 如果一下子看所有的要求覺得太複雜, 那麼可以簡化它, 例如只有單一一個

  語法6: *OP-Statement* ::= OP *float-number*

  這個語法告訴你如果輸入的程式是 “OP 3.5”, 就是符合語法的, 你寫一個函式process_opstatement(), 由鍵盤讀取空格或是換列字元分隔的程式字串, 會先讀到 “OP”, 和常數字串“OP”比對確認無誤以後, 再由鍵盤讀取 3.5 這個浮點數, 此時你可以輸出

  執行時間 = 3.5

  接下來請配置一個節點, 標示節點的型態是 *OP-Statement*, 把 3.5 記錄在結構裡面, 這個就是完整的樹狀資料結構了, 因為只有一條語法, 所以並不允許連續的兩個OP敘述, 最後把這個節點列印出來就完成了

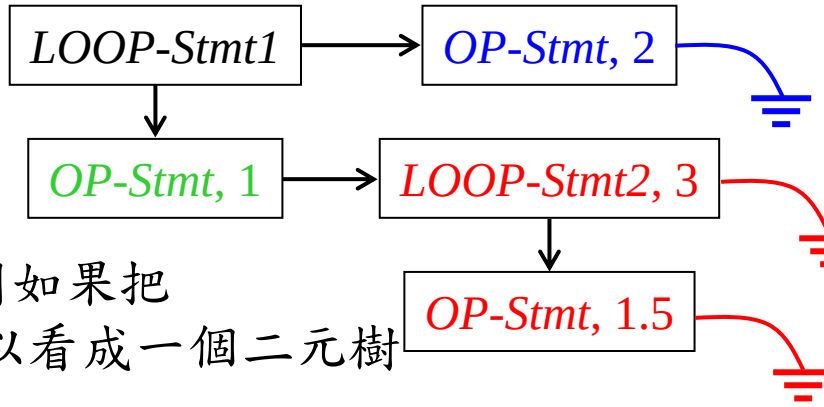
- 接下來你可以多考慮一點, 例如

  1. Program ::= BEGIN ***Statementlist*** END

  2'. Statementlist ::= *OP-**Statement*** | *OP-**Statement Statementlist***

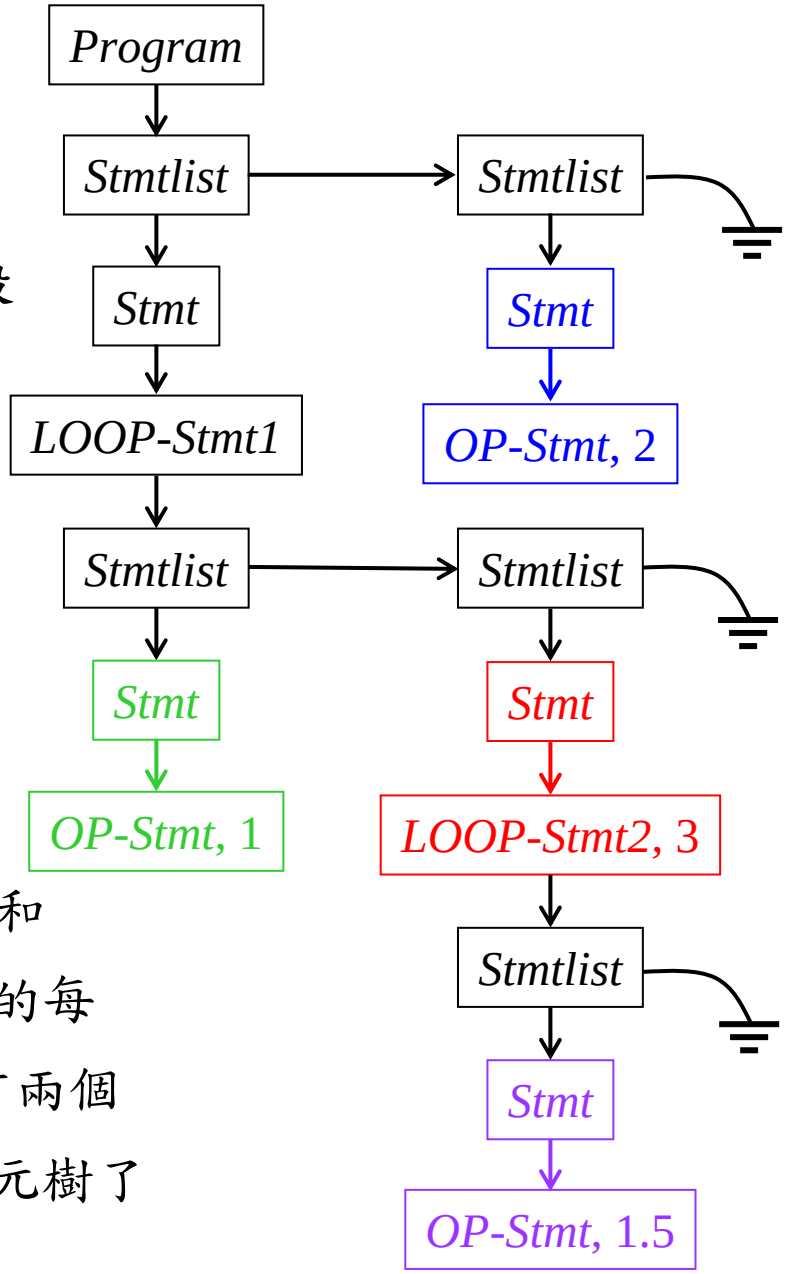
  6. OP-Statement ::= OP *float-number*

- 上面語法 *2'* 和原本的語法 *2* 有一點不同, 暫時不要去看 *LOOP-Statement* 的部份, 假設只有*OP-Statement*, ….
- 先寫一個 process_program() 函式讀入 BEGIN, 比對確認以後, 呼叫 process_statementlist() 函式處理第二條語法, 正確了以後再讀入 END, 比對確認
- process_statementlist() 函式裡面先讀入接下來的輸入, 如果是 OP, 則呼叫 process_opstatement() 來讀取必要的資料, 建立節點, 然後再根據輸入是 OP 還是 END 決定要不要呼叫 process_statementlist()
- 請注意這個語法不允許內容為空的程式 BEGIN END
- 接下來再思考語法*3*, 語法*4*, 語法*5* 處理迴圈敘述的部份…
- 語法*2* 其實按字義來說是串列, 右邊程式可以比較抽象地用下圖表示, 其中*Statementlist* 就是三個橫向的*Statement* 構成的串列, 簡化一下直接用 *LOOP-Stmt* 和*OP-Stmt* 取代語法中的*Statement*,當然這個圖如果把 Statement節點加進來, 還是可以看成一個二元樹




```
BEGIN
  LOOP n
    OP 1
    LOOP 3
      OP 1.5
    END
  END
  OP 2
END
```

- 根據第二頁的語法, 右側程式完整的(沒有簡化過的)語法分析應該如下圖所示




```
BEGIN
  LOOP n
    OP 1
    LOOP 3
      OP 1.5
    END
  END
  OP 2
END
```

這個圖形可以看成是一般化的樹狀圖, 只是這樣畫的時候裡面有兩種內部節點, 一種有兩個子節點, 一種只有一個子節點, 在前面作業要求裡的樹狀圖是稍微簡化過的了, 把 *Program* 和 *Stmt* 節點拿掉, 再讓剩下的每一個內部節點一律都具有兩個子節點, 就成為標準的二元樹了