作業四: 複雜度分析(II)

- 這個作業接續作業一, 希望你運用這學期學到的物件導向設計方法, 以C++撰寫“簡單演算法的複雜度分析程式”, 完成以後你可以和作業一以程序化方式設計的程式比較, 在短短一個學期裡我們不太容易藉由撰寫大規模的程式來擷取經驗, 所以希望你藉由比較兩種“關鍵性的設計” 來看到物件導向設計的精神
- 基本的問題描述和第一次作業相同, 但是我們稍微擴充一下語法, 把基本的運算式一起加進來

1. ***Program*** ::= BEGIN ***Statementlist*** END
2. ***Statementlist*** ::= ***Statement*** | ***Statement Statementlist***
3. ***Statement*** ::= ***LOOP-Statement*** | ***Expression-Statement***
4. ***LOOP-Statement*** ::= ***LOOP-Header Statementlist*** END
5. ***LOOP-Header*** ::= LOOP ***number*** | LOOP n
6. ***Expression-Statement*** ::= ***Expression*** ;
7. ***Expression*** ::= ***Term*** | ***Expression*** + ***Term***
8. ***Term*** ::= ***Variable*** | ***Term*** * ***Variable***
9. ***Variable*** ::= **a** | **b** | **…** | **z**

程式執行時間複雜度

- 加法敘述 *Expression* 所需要的時間為一單位的加法時間
- 乘法敘述 *Term* 所需要的時間為一單位的乘法時間
- 迴圈敘述 *LOOP-Statement* 所需要的時間基本上是迴圈次數乘上 迴圈內敘述串列所需要的時間, 忽略變數存取的時間, 也忽略迴圈控制變數的加法所需要的時間 (所以 LOOP n a ; END 需要的時間為 0)
- 敘述串列所需要的時間是所有節點敘述所需要的時間總和
- 上述前兩種敘述的時間都和實際執行的CPU有關, 例如下列兩種 CPU
  - CPU1: 一單位的加法時間為2個CPU時脈週期, 乘法為4個CPU時脈週期
  - CPU2: 一單位的加法時間為1個CPU時脈週期, 乘法為5個CPU時脈週期

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, +, -, ; 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
  a + b * c ;
  LOOP 3
  LOOP n
  w * w * z + x * y ;
  END
  a * w ;
  END
  r + r + r + r * s ;
  LOOP n
  a * a * a * a ;
  END
  END
  s * s + s ;
END
	CPU1 (2,4)
	執行時間 = 54 * n^2 + 28 * n^1 + 6
	CPU2 (1,5)
	執行時間 = 63 * n^2 + 29* n^1 + 6

BEGIN a * a * b ; LOOP n LOOP n c + d + e * f ; END END r * s + u * v ; END	CPU1 (2,4) 執行時間 = 8 * n^2 + 18 CPU2 (1,5) 執行時間 = 7 * n^2 + 21

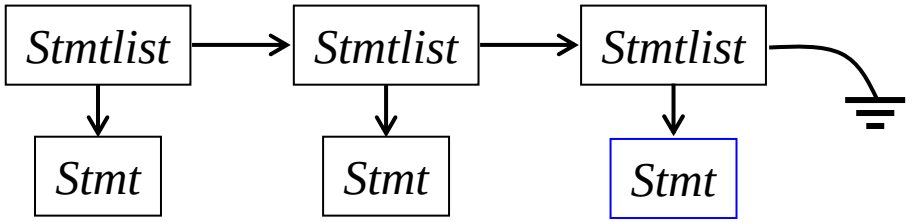
範例程式與基本測試資料

- Complexity06.exe
  請連同下列資料檔案下載到同一個資料匣, 在命令列視窗中執行 complexity06
  - testComp04.dat
  - testComp05.dat
  - testComp06.dat
- Complexity06interactive.exe
  執行此程式可以由鍵盤指定資料檔案名稱
- ComplexityCppUnitTest.exe
  執行此程式可以執行 11 個基本測試

類別設計 (1/8)

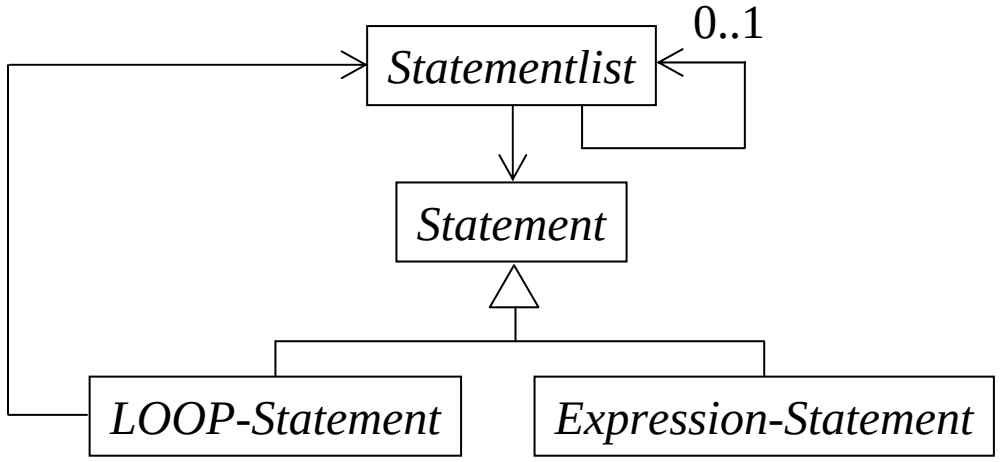
接下來我們看到一連串的類別架構討論, 請耐心讀完, 去想像那種設計代表的意義, 寫程式過程裡和使用者溝通是很重要的, 設計時找到比較適合的架構是很重要的, 不要隨便挑一個就實作

- *Statementlist* 描述一連串敘述構成的串列,如下圖串列中每一個節點指向一個 *Statement*:



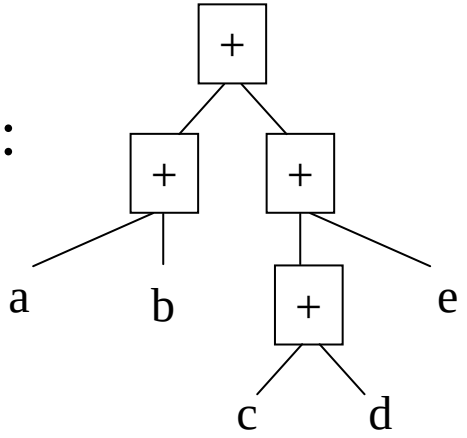
- 上面這種串列的基本*類別設計*如下:

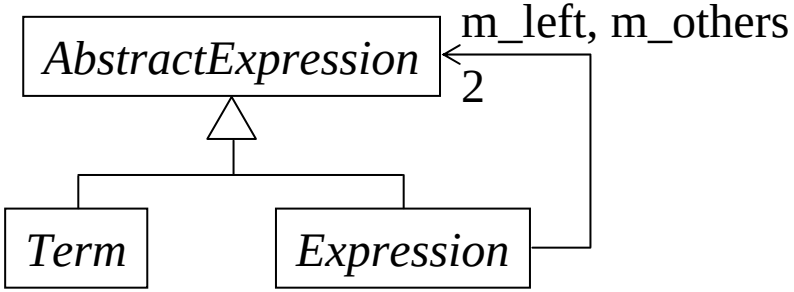
- 敘述 *Statement* 可以是 *LOOP-Statement* 或是 *Expression-Statement*, 需要運用繼承架構設計



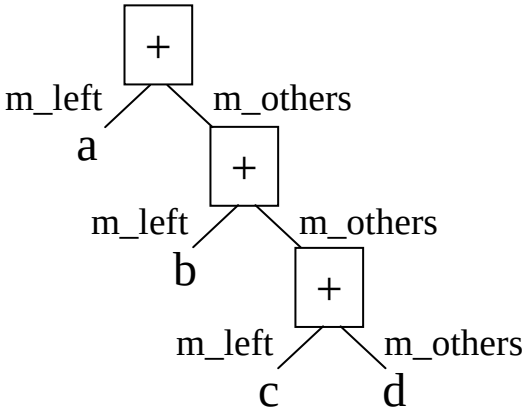
類別設計 (2/8)

- 運算式可以抽象地表示成二元樹狀架構, 例如:
- 如果使用 *Composite* Pattern 來設計

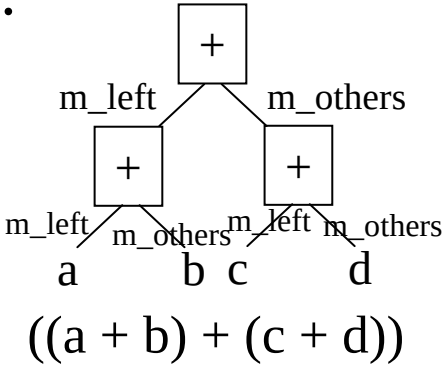




在這種類別架構底下 a + b + c + d 這個運算式的樹狀架構可能有很多種, 例如:



(a + (b + (c + d)))



((a + b) + (c + d))

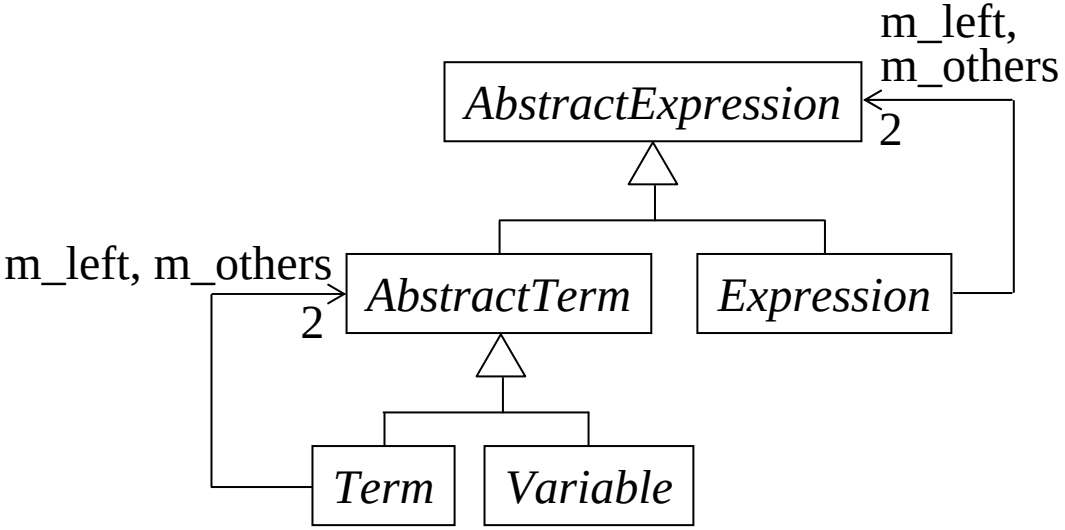
和語法

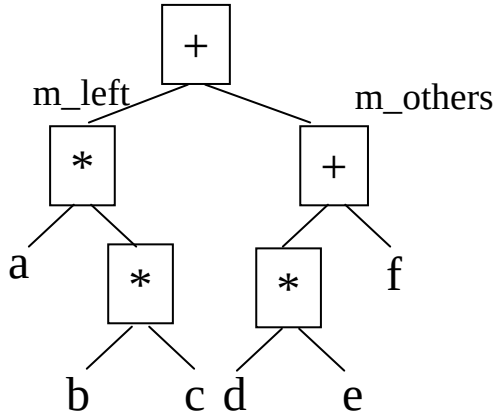
Expression ::= ***Term*** | ***Expression*** + ***Term***

所描述的 (((a+b)+c)+d) 不一樣

類別設計 (3/8)

- 如果運用兩層的 *Composite* Pattern, 加法和乘法的運算式可以整合設計成





a*b*c + d*e + f

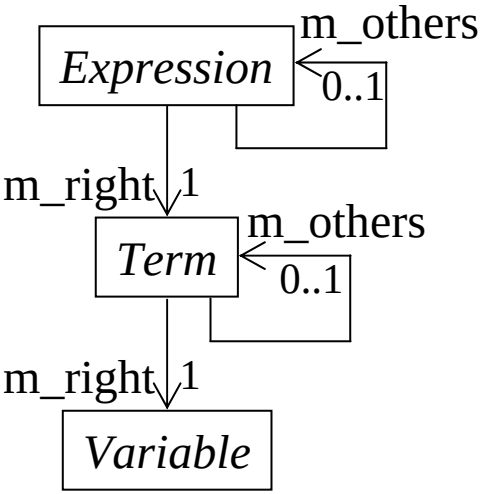
問題和前一頁是一樣的 – 如此設計出來的樹狀架構無法限制成和語法描述的樹狀架構一樣 (沒有辦法依照正確的結合律來剖析一個運算式)

Expression ::= ***Term*** | ***Expression*** + ***Term***

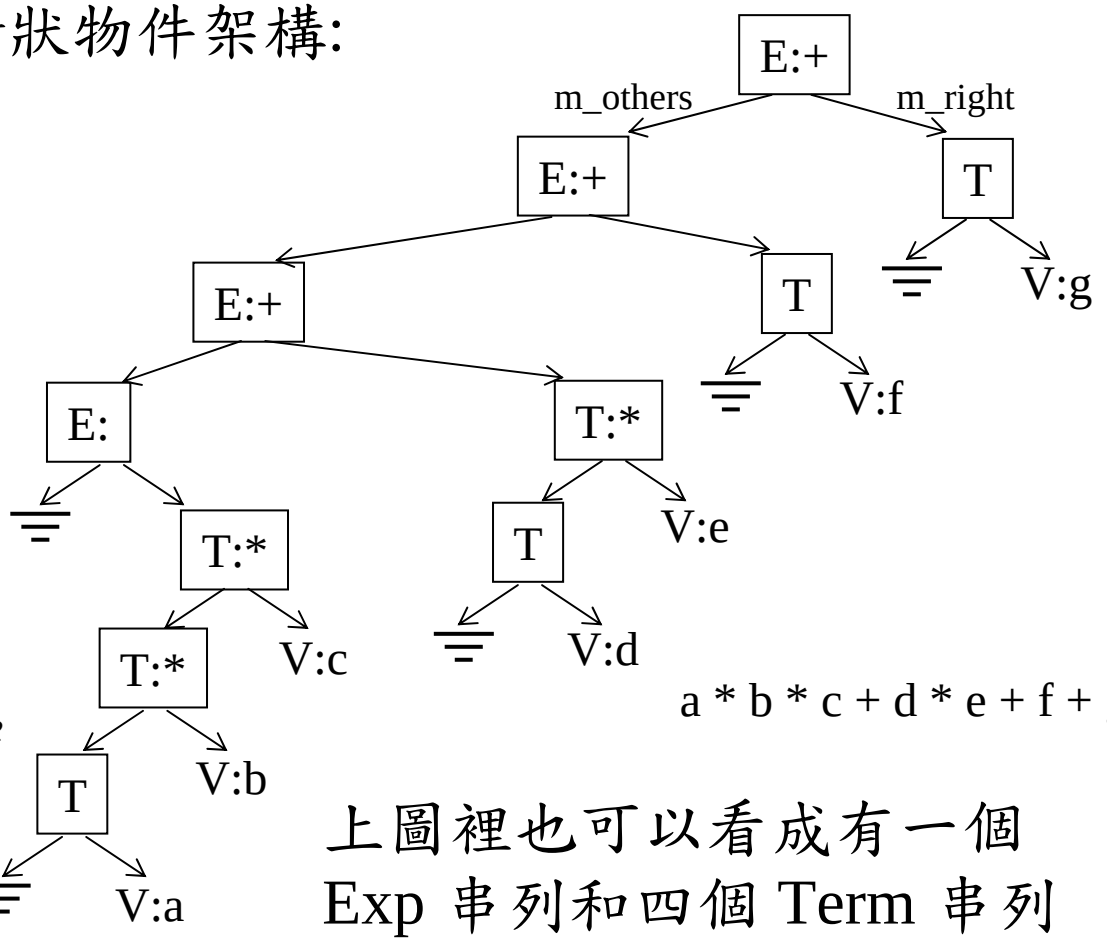
Term ::= ***Variable*** | ***Term*** * ***Variable***

類別設計 (4/8)

- 所以還是要重新考量用 *串列* 來設計運算式



此設計中每一個運算式都對應下面的樹狀物件架構:



Expression ::= ***Term*** |
Expression + ***Term***

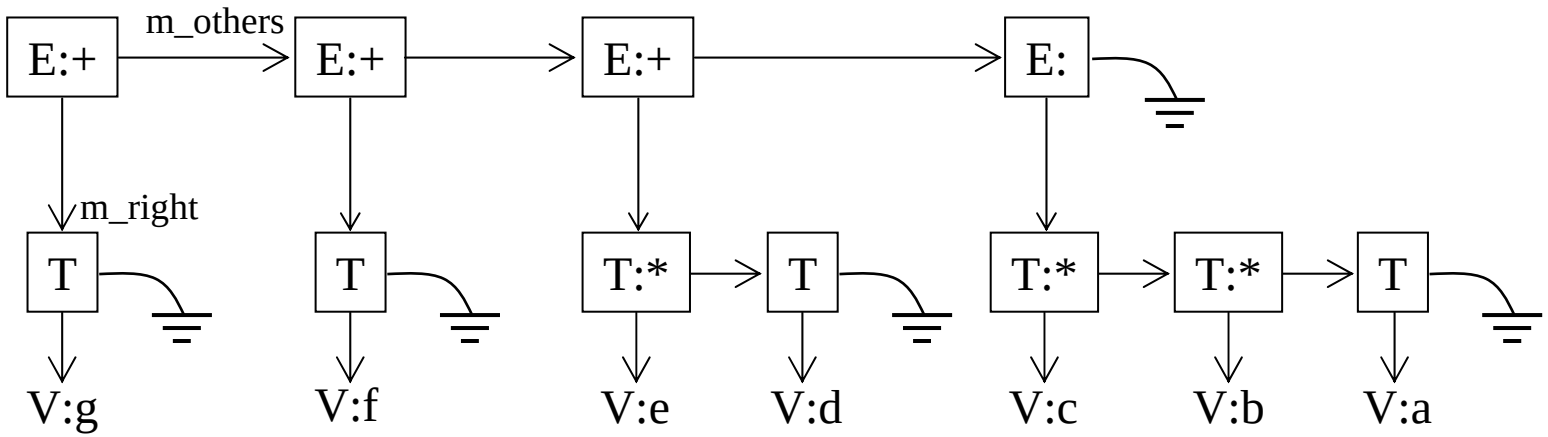
Term ::= ***Variable*** | ***Term*** * ***Variable***

a * b * c + d * e + f + g

上圖裡也可以看成有一個 Exp 串列和四個 Term 串列

類別設計 (5/8)

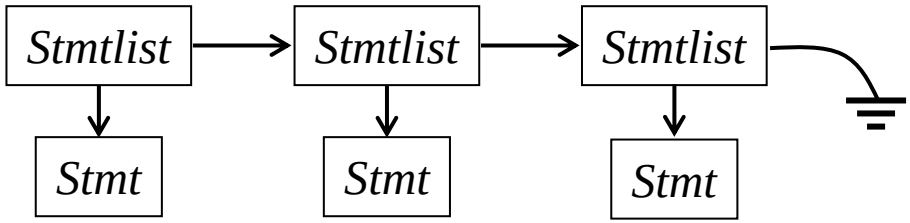
- 用熟悉的串列畫法畫出前一頁的物件圖

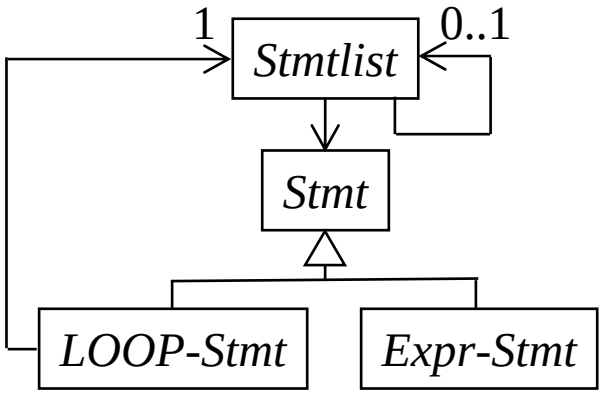


a * b * c + d * e + f + g

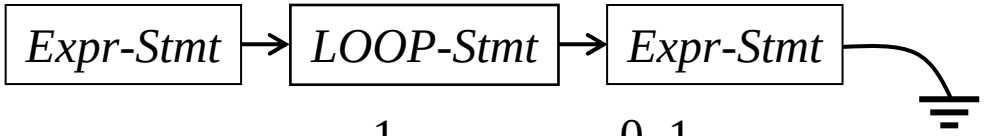
類別設計 (6/8)

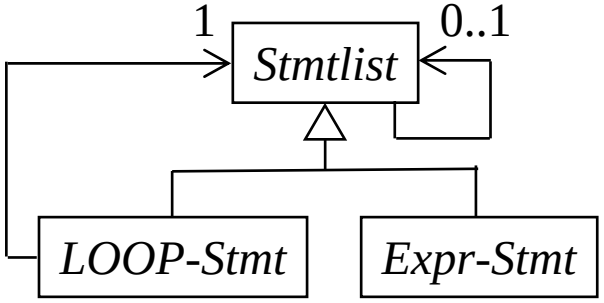
- 前面的 *Stmtlist* 串列裡每一個節點都是一樣的:



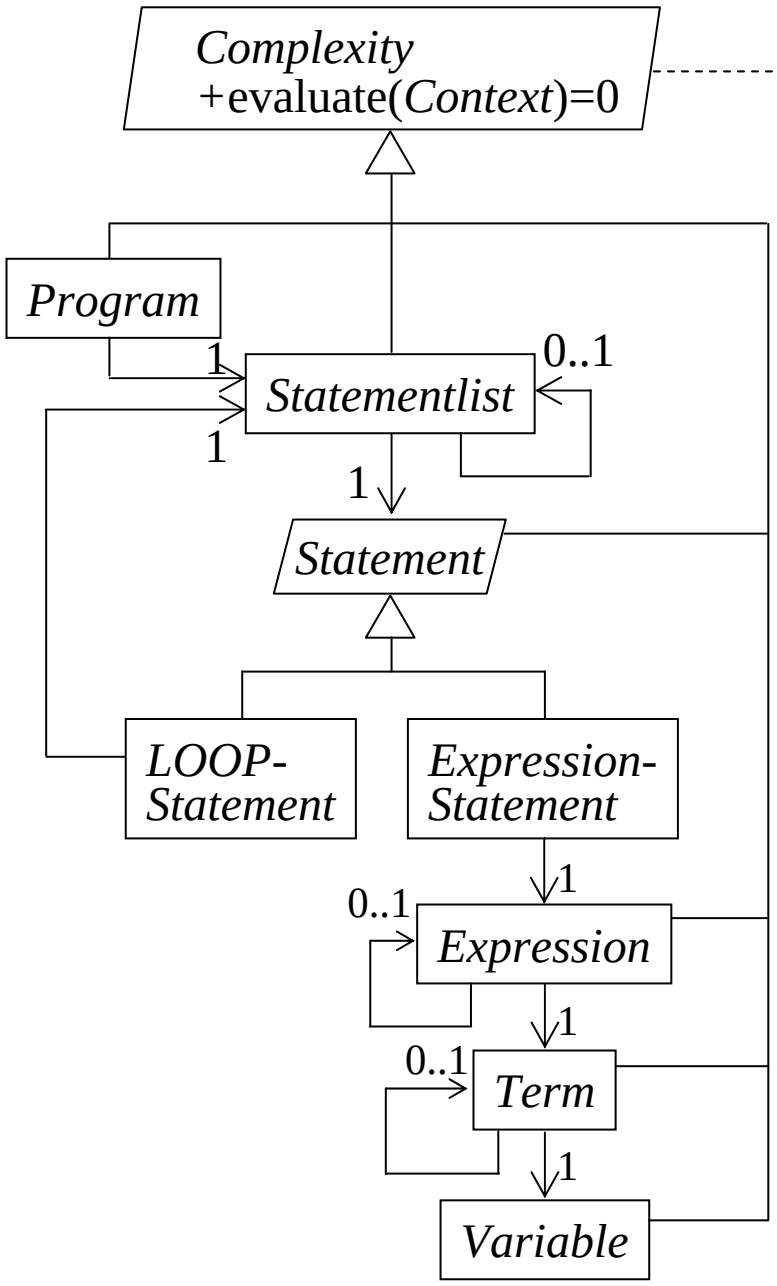


- 如果設計異質的節點, 例如: 可以省去每一節點委託 *Stmt* 類別的架構, 簡化不少, 當然還是需要設計一致的節點界面, 類別架構如右圖:





- 這個架構的缺點是我們讓 *LOOP-Stmt* 和 *Expr-Stmt* 既是可執行的敘述, 又同時繼承了串列節點的功能, 沒有適當地切割開來
- 前一頁的 *Expression* 串列和 *Term* 串列也可以有相同的考量

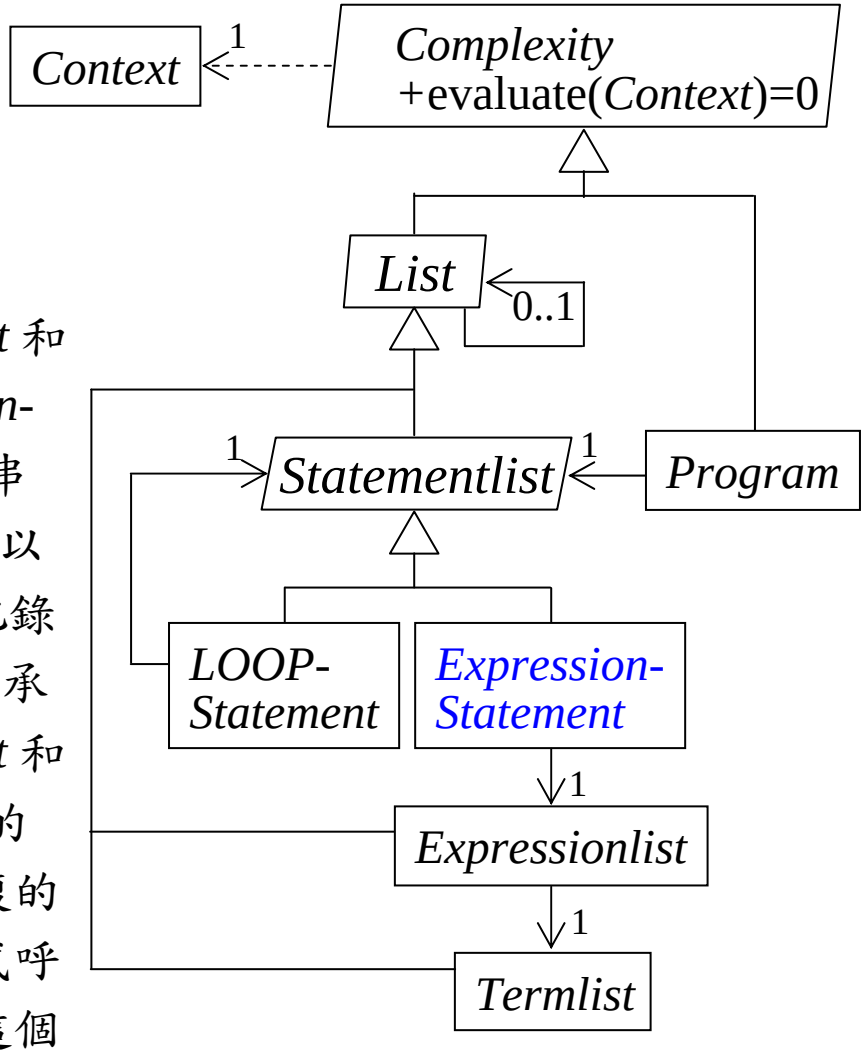


在這個應用裡我們需要估計一個程式的運算複雜度, 基本的運算複雜度和每一個加法、乘法所需要的時間有關係, 這些和處理器的架構有關, *CPUContext* 裡面可以記錄某一個處理器加法和乘法需要的時間, 迴圈敘述和敘述串列所需要的時間基本上是所包含的敘述的執行時間的總和, 所有的類別都繼承*Complexity*抽象類別, 其中的 evaluate() 界面會根據傳入的 *CPUContext* 來計算整個程式的時間複雜度

類別設計 (7/8)

類別設計 (8/8)

如果希望串列的節點是異質的, *Statementlist* 的節點可以是 *LOOP-Statement* 或 *Expression-Statement*, 後者除了代表語法上每一個後面有分號的運算式之外, 也藉由它把 *Statementlist* 和 *Expressionlist* 兩個串列分隔開, 每一個 *Expression-Statement* 類別的物件都指向一個 *Expressionlist* 串列物件, 另外也希望 *Expressionlist* 串列的節點可以是一個 *Termlist* 串列, *Termlist* 串列的節點直接紀錄每一個變數 , 與前一個設計相同所有的類別都繼承 *Complexity* 抽象類別, *Statementlist* , *Expressionlist* 和 *Termlist* 類別繼承一個 *List* 串列抽象類別, 這樣的設計裡你需要注意還是要把各個衍生類別裡重複的功能拉到他們的父類別裡實作，在利用虛擬函式呼叫子類別裡有差異的部份，另外值得討論的是這個類別架構雖然可以實際運作，但是並沒有直接把先前的語法裡面所有的元素實作出來，雖然省略一些類別，但是萬一將來需要修改語法時，困難度應該比前一頁的設計要高出許多



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
- 多項式請定義一個類別, 並且定義其基本的加法，多項式乘 n, 多項式乘常數，以及多項式列印
- 參考前面類別設計的說明 (可以使用不同的設計，不過請解釋設計的目標、架構、與特點)，你的目的在設計描述 “輸入程式” 各個語法架構的類別
- 各個類別應該要有由輸入串流 istream 建構物件的功能，解析語法的功能也就直接寫在這個建構元裡面，當由鍵盤或是測試檔案輸入程式時可以建構出對應的描述各個語法部份的物件
- 請運用 CppUnit 設計單元測試程式碼 (http://squall.cs.ntou.edu.tw/cpp/104spring/lab03/UsingCppUnit.html)，由於在測試的時候我們希望把一些TestCase直接撰寫在測試程式裡面，而不希望由鍵盤或是檔案輸入，我們可以運用istringstream 類別來指定不同的輸入，因為istringstream也是istream的子類別，所以可以用來直接取代鍵盤或是檔案的輸入串流，為了比較清楚了解所建立的資料結構是否正確，可以替每一個類別設計列印的界面函式 print(ostream &) 將內部資料結構對應的程式列印出來

其他程式要求 (cont'd)

- 請嘗試設計下列單元測試:
  1. 測試整體功能的TestCase (基本上就是讀取輸入檔案後，驗證計算出來的複雜度多項式是否如同預期)
  2. 測試所建立表達語法的物件架構是否正確 (這個部份你需要讓測試程式比對由 istream 建構的和手動建構預期的物件架構, 需要替每一個類別設計可以手動建構的建構元以及比對的equal函式), 例如:

```
void ParseTreeTests::testProgramConstrution() {
    istringstream iss1("BEGIN\na + b ;\nEND"), iss2("a + b ;");
    Program program(iss1), expectedProgram(new ExpressionStatement(iss2));
    CPPUNIT_ASSERT(expectedProgram.equal(program));
}
```

  3. 測試多項式類別的基本功能
- 在這個包含 CppUnit 的專案中，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, **104/06/11** (四) 21:00