如何閱讀代碼

1.要養成一個習慣, 經常花時間閱讀別人編寫的高品質代碼.
2.要有選擇地閱讀代碼, 同時, 還要有自己的目標. 您是想學習新的模式|編碼風格|還是滿足某些需求的方法.
3.要注意並重視代碼中特殊的非功能性需求, 這些需求也許會導致特殊的實現風格.
4.在現有的代碼上工作時, 請與作者和維護人員進行必要的協調, 以避免重複勞動或產生厭惡情緒.
5.請將從開放源碼軟件中得到的益處看作是一項貸款, 儘可能地尋找各種方式來回報開放源碼社團.
6.多數情況下, 如果您想要瞭解"別人會如何完成這個功能呢?", 除了閱讀代碼以外, 沒有更好的方法.
7.在尋找bug時, 請從問題的表現形式到問題的根源來分析代碼. 不要沿著不相關的路徑(誤入歧途).
8.我們要充分利用調試器|編譯器給出的警告或輸出的符號代碼|系統調用跟蹤器|數據庫結構化查詢語言的日誌機制|包轉儲工具和Windows的消息偵查程序, 定出的bug的位置.
9.對於那些大型且組織良好的系統, 您只需要最低限度地瞭解它的全部功能, 就能夠對它做出修改.
10.當向系統中增加新功能時, 首先的任務就是找到實現類似特性的代碼, 將它作為待實現功能的模板.
11.從特性的功能描述到代碼的實現, 可以按照字符串消息, 或使用關鍵詞來搜索代碼.
12.在移植代碼或修改接口時, 您可以通過編譯器直接定位出問題涉及的範圍, 從而減少代碼閱讀的工作量.
13.進行重構時, 您從一個能夠正常工作的系統開始做起, 希望確保結束時系統能夠正常工作. 一套恰當的測試用例(test case)可以幫助您滿足此項約束.
14.閱讀代碼尋找重構機會時, 先從系統的構架開始, 然後逐步細化, 能夠獲得最大的效益.
15.代碼的可重用性是一個很誘人, 但難以理解與分離, 可以試著尋找粒度更大一些的包, 甚至其他代碼.
16.在複查軟件系統時, 要注意, 系統是由很多部分組成的, 不僅僅只是執行語句. 還要注意分析以下內容: 文件和目錄結構|生成和配置過程|用戶界面和系統的文檔.
18.可以將軟件複查作為一個學習|講授|援之以手和接受幫助的機會.
++++++++++++++++++++
第二章: 基本編程元素
++++++++++++++++++++
19.第一次分析一個程序時, main是一個好的起始點.
20.層疊if-else if-...-else序列可以看作是由互斥選擇項組成的選擇結構.
21.有時, 要想瞭解程序在某一方面的功能, 運行它可能比閱讀源代碼更為恰當.

22.在分析重要的程序時, 最好首先識別出重要的組成部分.
23.瞭解局部的命名約定, 利用它們來猜測變量和函數的功能用途.
24.當基於猜測修改代碼時, 您應該設計能夠驗證最初假設的過程. 這個過程可能包括用編譯器進行檢查|引入斷言|或者執行適當的測試用例.
25.理解了代碼的某一部分, 可能幫助你理解餘下的代碼.
26.解決困難的代碼要從容易的部分入手.
27.要養成遇到庫元素就去閱讀相關文檔的習慣; 這將會增強您閱讀和編寫代碼的能力.
28.代碼閱讀有許多可選擇的策略: 自底向上和自頂向下的分析|應用試探法和檢查註釋和外部文檔, 應該依據問題的需要嘗試所有這些方法.
29.for (i=0; i<n; i++)形式的循環執行n次; 其他任何形式都要小心.
30.涉及兩項不等測試(其中一項包括相等條件)的比較表達式可以看作是區間成員測試.
31.我們經常可以將表達式應用在樣本數據上, 藉以瞭解它的含義.
32.使用De Morgan法則簡化複雜的邏輯表達式.
33.在閱讀邏輯乘表達式時, 問題可以認為正在分析的表達式以左的表達式均為true; 在閱讀邏輯和表達式時, 類似地, 可以認為正在分析的表達式以左的表達式均為false.
34.重新組織您控制的代碼, 使之更為易讀.
35.將使用條件運行符? :的表達式理解為if代碼.
36.不需要為了效率, 犧牲代碼的易讀性.
37.高效的算法和特殊的優化確實有可能使得代碼更為複雜, 從而更難理解, 但這並不意味著使代碼更為緊湊和不易讀會提高它的效率.
38.創造性的代碼佈局可以用來提高代碼的易讀性.
39.我們可以使用空格|臨時變量和括號提高表達式的易讀性.
40.在閱讀您所控制的代碼時, 要養成添加註釋的習慣.
41.我們可以用好的縮進以及對變量名稱的明智選擇, 提高編寫欠佳的程序的易讀性.
42.用diff程序分析程序的修訂歷史時, 如果這段歷史跨越了整體重新縮排, 常常可以通過指定-w選項, 讓diff忽略空白差異, 避免由於更改了縮進層次而引入的噪音.
43.do循環的循環體至少執行一次.
44.執行算術運算時, 當b=2n-1時, 可以將a&b理解為a%(b+1).
45.將a<<n理解為a*k, k=2n.
46.將a>>n理解為a/k, k=2n.
47.每次只分析一個控制結構, 將它的內容看作是一個黑盒.
48.將每個控制結構的控製表達式看作是它所包含代碼的斷言.
49.return, goto, break和continue語句, 還有異常, 都會影響結構化的執行流程. 由於這些語句一般都會終止或重新開始正在進行的循環,因此要單獨推理它們的行為.
50.用複雜循環的變式和不變式, 對循環進行推理.
51.使用保持含義不變的變換重新安排代碼, 簡化代碼的推理工作.
++++++++++++++++++++
第三章: 高級C數據類型
++++++++++++++++++++

52.瞭解特定語言構造所服務的功能之後, 就能夠更好地理解使用它們的代碼.
53.識別並歸類使用指針的理由.
54.在C程序中, 指針一般用來構造鍊式數據結構|動態分配的數據結構|實現引用調用|訪問和迭代數據元素|傳遞數組參數|引用函數|作為其他
值的別名|代表字符串|以及直接訪問系統內存.
55.以引用傳遞的參數可以用來返回函數的結果, 或者避免參數複製帶來的開銷.
56.指向數組元素地址的指針, 可以訪問位於特定索引位置的元素.
57.指向數組元素的指針和相應的數組索引, 作用在二者上的運算具有相同的語義.
58.使用全局或static局部變量的函數大多數情況都不可重入(reentrant).
59.字符指針不同於字符數組.
60.識別和歸類應用結構或共用體的每種理由.
61.C語言中的結構將多個數據元素集合在一起, 使得它們可以作為一個整體來使用, 用來從函數中返回多個數據元素|構造鍊式數據結構|映射數據在硬件設備|網絡鏈接和存儲介質上的組織方式|實現抽像數據類型|以及以面向對象的方式編程.
62.共用體在C程序中主要用於優化存儲空間的利用|實現多態|以及訪問數據不同的內部表達方式.
63.一個指針, 在初始化為指向N個元素的存儲空間之後, 就可以作為N個元素的數組來使用.
64.動態分配的內在塊可以電焊工地釋放, 或在程序結束時釋放, 或由垃圾回收器來完成回收; 在棧上分配的內存塊當分配它的函數退出後釋放.
65.C程序使用typedef聲明促進抽象, 並增強代碼的易讀性, 從而防範可移植性問題, 並模擬C++和Java的類聲明行為.
66.可以將typedef聲明理解成變量定義:變量的名稱就是類型的名稱;變量的類型就是與該名稱對應的類型. ++++++++++++++++

第四章: C數據結構
++++++++++++++++
67.根據底層的抽像數據類型理解顯式的數據結構操作.
68.C語言中, 一般使用內建的數組類型實現向量, 不再對底層實現進行抽象.
69.N個元素的數組可以被序列for (i=0; i<N; i++)完全處理; 所有其他變體都應該引起警惕.
70.表達式sizeof(x)總會得到用memset或memcpy處理數組x(不是指針)所需的正確字節數.
71.區間一般用區間內的第一個元素和區間後的第一個元素來表示.
72.不對稱區間中元素的數目等於高位邊界與低位邊界的差.
73.當不對稱區間的高位邊界等於低位邊界時, 區間為空.
74.不對稱區間中的低位邊界代表區間的第一個元素; 高位邊界代表區間外的第一個元素.
75.結構的數組常常表示由記錄和字段組成的表.
76.指向結構的指針常常表示訪問底層記錄和字段的游標.

77.動態分配的矩陣一般存儲為指向數組列的指針或指向元素指針的指針; 這兩種類型都可以按照二維數組進行訪問.
78.以數組形式存儲的動態分配矩陣, 用自定義訪問函數定位它們的元素.
79.抽像數據類型為底層實現元素的使用(或誤用)方式提供一種信心的量度.
80.數組用從0開始的順序整數為鍵, 組織查找表.
81.數組經常用來對控制結構進行高效編碼, 簡化程序的邏輯.
82.通過在數組中每個位置存儲一個數據元素和一個函數指針(指向處理數據元素的函數), 可以將代碼與數據關聯起來.
83.數組可以通過存儲供程序內的抽像機(abstract machine)或虛擬機(virtual machine)使用的數據或代碼, 控製程序的運作.
84.可以將表達式sizeof(x) / sizeof(x[0])理解為數組x中元素的個數.
85.如果結構中含有指向結構自身|名為next的元素, 一般說來, 該結構定義的是單向鍊錶的結點.
86.指向鍊錶結點的持久性(如全局|靜態或在堆上分配)指針常常表示鍊錶的頭部.
87.包含指向自身的next和prev指針的結構可能是雙向鍊錶的結點.
88.理解複雜數據結構的指針操作可以將數據元素畫為方框|指針畫為箭頭.
89.遞歸數據結構經常用遞歸算法來處理.
90.重要的數據結構操作算法一般用函數參數或模板參數來參數化.
91.圖的結點常常順序地存儲在數組中, 鏈接到鍊錶中, 或通過圖的邊鏈接起來.
92.圖中的邊一般不是隱式地通過指針, 就是顯式地作為獨立的結構來表示.
93.圖的邊經常存儲為動態分配的數組或鍊錶, 在這兩種情況下, 邊都錨定在圖的結點上.
94.在無向圖中, 表達數據時應該將所有的結點看作是等同的, 類似地, 進行處理任務的代碼也不應該基於它們的方向來區分邊.
95.在非連通圖中, 執行遍歷代碼應該能夠接通孤立的子圖.
96.處理包含迴路的圖時, 遍歷代碼應該避免在處理圖的迴路進入循環.
97.複雜的圖結構中, 可能隱藏著其他類型的獨立結構.
++++++++++++++++++
第五章: 高級控制流程
++++++++++++++++++
98.採用遞歸定義的算法和數據結構經常用遞歸的函數定義來實現.
99.推理遞歸函數時, 要從基準落伍測試開始, 並認證每次遞歸調用如何逐漸接近非遞歸基準範例代碼.
100.簡單的語言常常使用一系列遵循該語言語法結構的函數進行語法分析.
101.推理互遞歸函數時, 要基於底層概念的遞歸定義.
102.尾遞歸調用等同於一個回到函數開始處的循環.
103.將throws子句從方法的定義中移除, 然後運行Java編譯器對類的源代碼進行編譯, 就可以容易地找到那些可能隱式地生成異常的方法.
104.在多處理器計算機上運行的代碼常常圍繞進程或線程進行組織.
105.工作群並行模型用於在多個處理器間分配工作, 或者創建一個任務池, 然後將大量需

要處理標準化的工作進行分配.
106.基於線程的管理者/工人並行模型一般將耗時的或阻塞的操作分配給工人子任務, 從而維護中心任務的響應性.
107.基於進程的管理者/工人並行模型一般用來重用現有的程序, 或用定義良好的接口組織和分離粗粒度的系統模塊.
108.基於流水線的並行處理中, 每個任務都接收到一些輸入, 對它們進行一些處理, 並將生成的輸出傳遞給下一個任務, 進行不同的處理.
109.競爭條件很難捉摸, 相關的代碼常常會將競爭條件擴散到多個函數或模塊; 因而, 很難隔離由於競爭條件導致的問題.
110.對於出現在信號處理器中的數據結構操作代碼和庫調用要保持高度警惕.
111.在閱讀包含宏的代碼時, 要注意, 宏既非函數, 也非語句.
112.do…while(0)塊中的宏等同於控制塊中的語句.
113.宏可以訪問在它的使用點可見的所有局部變量.
114.宏調用可改變參數的值
115.基於宏的標記拼接能夠創建新的標記符.

++++++++++++++++++
第六章: 應對大型項目
++++++++++++++++++
116.我們可以通過瀏覽項目的源代碼樹—包含項目源代碼的層次目錄結構, 來分析一個項目的組織方式. 源碼樹常常能夠反映出項目在構架和軟件過程上的結構.
117.應用程序的源代碼樹經常是該應用程序的部署結構的鏡像.
118.不要被龐大的源代碼集合嚇倒; 它們一般比小型的專門項目組織得更出色.
119.當您首次接觸一個大型項目時, 要花一些時間來熟悉項目的目錄樹結構.
120.項目的源代碼遠不只是編譯後可以獲得可執行程序的計算機語言指令; 一個項目的源碼樹一般還包括規格說明|最終用戶和開發人員文檔|測試腳本|多媒體資源|編譯工具|例子|本地化文件|修訂歷史|安裝過程和許可信息.
121.大型項目的編譯過程一般聲明性地借助依賴關係來說明. 依賴關係由工具程序, 如make及其派生程序, 轉換成具體的編譯行動.
122.大型項目中, 製作文件常常由配置步驟動態地生成; 在分析製作文件之前, 需要先執行項目特定的配置.
123.檢查大型編譯過程的各個步驟時, 可以使用make程序的-n開關進行預演.
124.修訂控制系統提供從儲存庫中獲取源代碼最新版本的方式.
125.可以使用相關的命令, 顯示可執行文件中的修訂標識關鍵字, 從而將可執行文件與它的源代碼匹配起來.
126.使用修訂日誌中出現的bug跟蹤系統內的編號, 可以在bug跟蹤系統的數據庫中找到有關的問題的說明.
127.可以使用修訂控制系統的版本儲存庫, 找出特定的變更是如何實現的.
128.定製編譯工具用在軟件開發過程的許多方面, 包括配置|編譯過程管理|代碼的生成|

測試和文檔編制.
129.程序的調試輸出可以幫助我們理解程序控制流程和數據元素的關鍵部分.
130.跟蹤語句所在的地點一般也是算法運行的重要部分.
131.可以用斷言來檢驗算法運作的步驟|函數接收的參數|程序的控制流程|底層硬件的屬性和測試用例的結果.
132.可以使用對算法進行檢驗的斷言來證實您對算法運作的理解, 或將它作為推理的起點.
133.對函數參數和結果的斷言經常記錄了函數的前置條件和後置條件.
134.我們可以將測試整個函數的斷言作為每個給定函數的規格說明.
135.測試用例可以部分地代替函數規格說明.
136.可以使用測試用例的輸入數據對源代碼序列進行預演.
++++++++++++++++++++
第七章: 編碼規範和約定
++++++++++++++++++++
137.瞭解了給定代碼庫所遵循的文件組織方式後, 就能更有效率地瀏覽它的源代碼.
138.閱讀代碼時, 首先要確保您的編輯器或優美打印程序的tab設置, 與代碼遵循的風格規範一致.
139.可以使用代碼塊的縮進, 快速地掌握代碼的總體結構.
140.對編排不一致的代碼, 應該立即給予足夠的警惕.
141.分析代碼時, 對標記為XXX, FIXME和TODO的代碼序列要格外注意: 錯誤可能就潛伏在其中.
142.常量使用大寫字母命名, 單詞用下劃線分隔.
143.在遵循Java編碼規範的程序中, 包名(package name)總是從一個頂級的域名開始(例如, org, com), 類名和接口名由大寫字母開始, 方法和變量名由小寫字母開始.
144.用戶界面控件名稱之前的匈牙利記法的前綴類型標記可以幫助我們確定它的作用.
145.不同的編程規範對可移植構造的構成有不同的主張.
146.在審查代碼的可移植性, 或以某種給定的編碼規範作為指南時, 要注意瞭解規範對可移植性需求的界定與限制.
147.如果GUI功能都使用相應的編程結構來實現, 則通過代碼審查可以輕易地驗證給定用戶界面的規格說明是否被正確地採用.
148.瞭解項目編譯過程的組織方式與自動化方式之後, 我們就能夠快速地閱讀與理解對應的編譯規則.
149.當檢查系統的發布過程時, 常常可以將相應發行格式的需求作為基準.
++++++++++++
第八章： 文檔
++++++++++++
150.閱讀代碼時, 應該儘可能地利用任何能夠得到的文檔.
151.閱讀一小時代碼所得到的信息只不過相當於閱讀一分鍾文檔.
152.使用系統的規格說明文檔, 瞭解所閱讀代碼的運行環境.

153.軟件需求規格說明是閱讀和評估代碼的基準.
154.可以將系統的設計規格說明作為認知代碼結構的路線圖, 閱讀具體代碼的指引.
155.測試規格說明文檔為我們提供可以用來對代碼進行預演的數據.
156.在接觸一個未知系統時, 功能性的描述和用戶指南可以提供重要的背景信息,從而更好地理解閱讀的代碼所處的上下文.
157.從用戶參考手冊中, 我們可以快速地獲取, 應用程序在外觀與邏輯上的背景知識, 從管理員手冊中可以得知代碼的接口|文件格式和錯誤消息的詳細信息.
158.利用文檔可以快捷地獲取系統的概況, 瞭解提供特定特性的代碼.
159.文檔經常能夠反映和提示出系統的底層結構.
160.文檔有助於理解複雜的算法和數據結構.
161.算法的文字描述能夠使不透明(晦澀, 難以理解)的代碼變得可以理解.
162.文檔常常能夠闡明源代碼中標識符的含義.
163.文檔能夠提供非功能性需求背後的理論基礎.
164.文檔還會說明內部編程接口.
165.由於文檔很少像實際的程序代碼那樣進行測試, 並受人關注, 所以它常常可能存在錯誤|不完整或過時.
166.文檔也提供測試用例, 以及實際應用的例子.
167.文檔常常還會包括已知的實現問題或bug.
168.環境中已知的缺點一般都會記錄在源代碼中.
169.文檔的變更能夠標出那些故障點.
170.對同一段源代碼重複或互相衝突的更改, 常常表示存在根本性的設計缺陷, 從而使得維護人員需要用一系列的修補程序來修復.
171.相似的修復應用到源代碼的不同部分, 常常表示一種易犯的錯誤或疏忽, 它們同樣可能會在其他地方存在.
172.文檔常常會提供不恰當的信息, 誤導我們對源代碼的理解.
173.要警惕那些未歸檔的特性: 將每個實例歸類為合理|疏忽或有害, 相應地決定是否應該修復代碼或文檔.
174.有時, 文檔在描述系統時, 並非按照已完成的實現, 而是系統應該的樣子或將來的實現.
175.在源代碼文檔中, 單詞gork的意思一般是指」理解」.
176.如果未知的或特殊用法的單詞阻礙了對代碼的理解, 可以試著在文檔的術語表(如果存在的話)|New Hacker's Dictionary[Ray96]|或在Web搜索引擎中查找它們.
177.總是要以批判的態度來看待文檔, 注意非傳統的來源, 比如註釋|標準|出版物|測試用例|郵件列表|新聞組|修訂日誌|問題跟蹤數據庫|營銷材料|源代碼本身.
178.總是要以批判的態度來看待文檔; 由於文檔永遠不會執行, 對文檔的測試和正式複查也很少達到對代碼的同樣水平, 所以文檔常常會誤導讀者, 或者完全錯誤.
179.對於那些有缺陷的代碼, 我們可以從中推斷出它的真實意圖.
180.在閱讀大型系統的文檔時, 首先要熟悉文檔的總體結構和約定.
181.在對付體積龐大的文檔時, 可以使用工具, 或將文本輸出到高品質輸出設備上, 比如

激光打印機, 來提高閱讀的效率.
++++++++++++++++
第九章: 系統構架
++++++++++++++++
182.一個系統可以(在重大的系統中也確實如此)同時出多種不同的構架類型. 以不同的方式檢查同一系統|分析系統的不同部分|或使用不同級別的分解, 都有可能發現不同的構架類型.
183.協同式的應用程序, 或者需要協同訪問共享信息或資源的半自治進程, 一般會採用集中式儲存庫構架.
184.黑板系統使用集中式的儲存庫, 存儲非結構化的鍵/值對, 作為大量不同代碼元件之間的通信集線器.
185.當處理過程可以建模|設計和實現成一系列的數據變換時, 常常會使用數據流(或管道—過濾器)構架.
186.在批量進行自動數據處理的環境中, 經常會採用數據流構架, 在對數據工具提供大量支持的平台上尤其如此.
187.數據流構架的一個明顯徵兆是: 程序中使用臨時文件或流水線(pipeline)在不同進程間進行通信.
188.使用圖示來建模面向對象構架中類的關係.
189.可以將源代碼輸入到建模工具中, 逆向推導出系統的構架.
190.擁有大量同級子系統的系統, 常常按照分層構架進行組織.
191.分層構架一般通過堆疊擁有標準化接口的軟件組件來實現.
192.系統中每個層可以將下面的層看作抽象實體, 並且(只要該層滿足它的需求說明)不關心上面的層如何使用它.
193.層的接口既可以是支持特定概念的互補函數族, 也可以是一系列支持同一抽象接口不同底層實現的可互換函數.
194.用C語言實現的系統, 常常用函數指針的數組, 表達層接口的多路復用操作.
195.用面向對象的語言實現的系統, 使用虛方法調用直接表達對層接口的多嘴復用操作.
196.系統可以使用不同的|獨特的層次分解模型跨各種坐標軸進行組織.
197.使用程序切片技術, 可以將程序中的數據和控制之間依賴關係集中到一起.
198.在並發系統中, 一個單獨的系統組件起到集中式管理器的作用, 負責啟動|停止和協調其他系統進程和任務的執行.
199.許多現實的系統都會博採眾家之長. 當處理此類系統時, 不要徒勞地尋找無所不包的構架圖; 應該將不同構架風格作為獨立但相關的實體
來進行定位|識別並瞭解.
200.狀態變遷圖常常有助於理清狀態機的動作.
201.在處理大量的代碼時, 瞭解將代碼分解成單獨單元的機制極為重要.
202.大多數情況下, 模塊的物理邊界是單個文件|組織到一個目錄中的多個文件或擁有統一前綴的文件的集合.
203.C中的模塊, 由提供模塊公開接口的頭文件和提供對應實現的源文件組成.

204.對象的構造函數經常用來分配與對象相關的資源, 並初始化對象的狀態. 函數一般用來釋放對像在生命期中佔用的資源.
205.對象方法經常使用類字段來存儲控制所有方法運作的數據(比如查找表或字典)或維護類運作的狀態信息(例如, 賦給每個對像一個標識符的
計數器).
206.在設計良好的類中, 所有的字段都應在聲明為private, 並用公開的訪問方法提供對它們的訪問.
207.在遇到friend聲明時, 要停下來分析一下, 看看繞過類封裝在設計上的理由.
208.可以有節制地用運算符增強特定類的可用性, 但用運算符重載, 將類實現為擁有內建算術類型相關的全部功能的類實體, 是不恰當的.
209.泛型實現不是在編譯期間通過宏替換或語言所支持的功能(比如C++模板和Ada的泛型包)來實現, 就是在運行期間通過使用數據元素的指針和函數的指針|或對象的多態性實現.
210.抽像數據類型經常用來封裝常用的數據組織方案(比如樹|列表或棧), 或者對用戶隱藏數據類型的實現細節.
211.使用庫的目的多種多樣: 重用源代碼或目標代碼, 組織模塊集合, 組織和優化編譯過程, 或是用來實現應用程序各種特性的按需載入.
212.大型的|分佈式的系統經常實現為許多互相協作的進程.
213.對於基於文本的數據儲存庫, 可以通過瀏覽存儲在其中的數據, 破譯出它的結構.
214.可以通過查詢數據字典中的表, 或使用數據庫專有的SQL命令, 比如show table, 來分析關係型數據庫的模式.
215.識別出重用的構架元素後, 可以查找其最初的描述, 瞭解正確地使用這種構架的方式, 以及可能出現的誤用.
216.要詳細分析建立在某種框架之上的應用程序, 行動的最佳路線就是從研究框架自身開始.
217.在閱讀嚮導生成的代碼時, 不要期望太高, 否則您會感到失望.
218.學習幾個基本的設計模式之後, 您會發現, 您查看代碼構架的方式會發生改變: 您的視野和詞彙將會擴展到能夠識別和描述許多通用的形式.
219.頻繁使用的一些模式, 但並不顯式地指出它們的名稱, 這是由於構架性設計的重用經常先於模式的形成.
220.請試著按照底層模式來理解構架, 即使代碼中並沒有明確地提及模式.
221.大多數解釋器都遵循類似的處理構架, 圍繞一個狀態機進行構建, 狀態機的操作依賴於解釋器的當前狀態|程序指令和程序狀態.
222.多數情況下, 參考構架只是為應用程序域指定一種概念性的結構, 具體的實現並非必須遵照這種結構.

++++++++++++++++++
第十章: 代碼閱讀工具
++++++++++++++++++

223.詞彙工具可以高效地在一個大代碼文件中或者跨多個文件查找某種模式.
224.使用程序編輯器和正則表達式查找命令, 瀏覽龐大的源代碼文件.
225.以只讀方式瀏覽源代碼文件.
226.使用正則表達式^function name 可以找出函數的定義.
227.使用正則表達式的字符類, 可以查找名稱遵循特定模式的變量.
228.使用正則表達式的否定字符類, 可以避免非積極匹配.
229.使用正則表達式symbol-1. *symbol-2, 可以查找出現在同一行的符號.
230.使用編輯器的tags 功能, 可以快速地找出實體的定義.
231.可以用特定的tag 創建工具, 增加編輯器的瀏覽功能.
232.使用編輯器的大綱視圖, 可以獲得源代碼結構的鳥瞰圖.
233.使用您的編輯器來檢測源代碼中圓括號|方括號和花括號的匹配.
234.使用grep 跨多個文件查找代碼模式.
235.使用grep 定位符號的聲明|定義和應用.
236.當您不能精確地表述要查找的內容時, 請使用關鍵單詞的詞幹對程序的源代碼進行查找.
237.用grep 過濾其他工俱生成的輸出, 分離出您要查找的項.
238.將grep 的輸出輸送到其他工具, 使複雜處理任務自動化.
239.通過對grep 的輸出進行流編輯, 重用代碼查找的結果.
240.通過選取與噪音模式不匹配的輸出行(grep-v), 過濾虛假的grep 輸出.
241.使用fgrep 在源代碼中查找字符串列表.
242.查找註釋, 或標識符大小寫不敏感的語言編寫的代碼時, 要使用大小寫不敏感的模式匹配(grep -i).
243.使用grep –n 命令行開關, 可以創建與給定正則表達式匹配的文件和行號的檢查表.
244.可以使用diff 比較文件或程序不同版本之間的差別.
245.在運行diff 命令時, 可以使用diff –b, 使文件比較算法忽略結尾的空格, 用–w 忽略所有空白區域的差異, 用–i 使文件比較對大小寫不敏感.
246.不要對創建自己的代碼閱讀工具心存畏懼.
247.在構建自己的代碼閱讀工具時: 要充分利用現代快速原型語言所提供的能力; 從簡單開始, 根據需要逐漸改進; 使用利用代碼詞彙結構的各種試探法; 要允許一些輸出噪音或寂靜(無關輸出或缺失輸出); 使用其他工具對輸入進行預處理, 或者對輸出進行後期處理.
248.要使編譯器成為您的: 指定恰當級別的編譯器警告, 並小心地評估生成的結果.
249.使用C預處理器理清那些濫用預處理器特性的程序.
250.要徹底地瞭解編譯器如何處理特定的代碼塊, 需要查看生成的符號(彙編)代碼.
251.通過分析相應目標文件中的符號, 可以清晰地瞭解源文件的輸入和輸出.
252.使用源代碼瀏覽器瀏覽大型的代碼集合以及對像類型.
253.要抵制住按照您的編碼規範對外部代碼進行美化的誘惑; 不必要的編排更改會創建不同的代碼, 並妨礙工作的組織.
254.優美打印程序和編輯器語法著色可以使得程序的源代碼為易讀.
255.cdecl 程序可以將難以理解的C和C++類型聲明轉換成純英語(反之亦然).

256.實際運行程序, 往往可以更深刻地理解程序的動作.
257.系統調用|事件和數據包跟蹤程序可以增進對程序動作的理解.
258.執行剖析器可以找出需要著重優化的代碼, 驗證輸入數據的覆蓋性, 以及分析算法的動作.
259.通過檢查從未執行的代碼行, 可以找出測試覆蓋的弱點, 並據此修正測試數據.
260.要探究程序動態動作時的每個細節, 需要在調試器中運作它.
261.將您覺得難以理解的代碼打印到紙上.
262.可以繪製圖示來描繪代碼的動作.
263.可以試著向別人介紹您在閱讀的代碼, 這樣做一般會增進您對代碼的理解.
264.理解複雜的算法或巧妙的數據結構, 要選擇一個安靜的環境, 然後聚精會神地考慮, 不要藉助於任何計算機化或自動化的幫助.
++++++++++++++++++++++
第十一章: 一個完整的例子
++++++++++++++++++++++
265.模仿軟件的功能時, 要依照相似實體的線路(類|函數|模塊). 在相似的現有實體中, 為簡化對源代碼庫的文本查找, 應選取比較罕見的名稱.
266.自動生成的文件常常會在文件的開關有一段註釋, 說明這種情況.
267.如果試圖精確地分析代碼, 一般會陷入數量眾多的類|文件和模塊中, 這些內容會很快將我們淹沒; 因此, 我們必須將需要理解的代碼限定在絕對必需的範圍之內.
268.採用一種廣度優先查找策略, 從多方攻克代碼閱讀中存在的問題, 進到找出克服它們的方法為止.