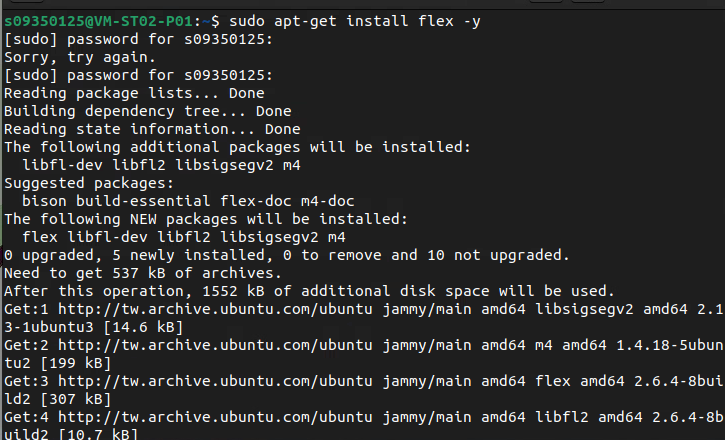
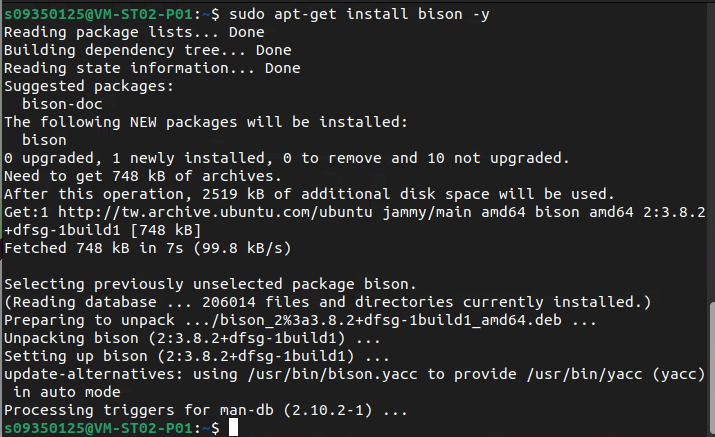
在ubuntu 中安裝lex(flex) 和 yacc (bison)

Lex: sudo -S apt-get install flex -y指令 → 安裝lex(flex)工具

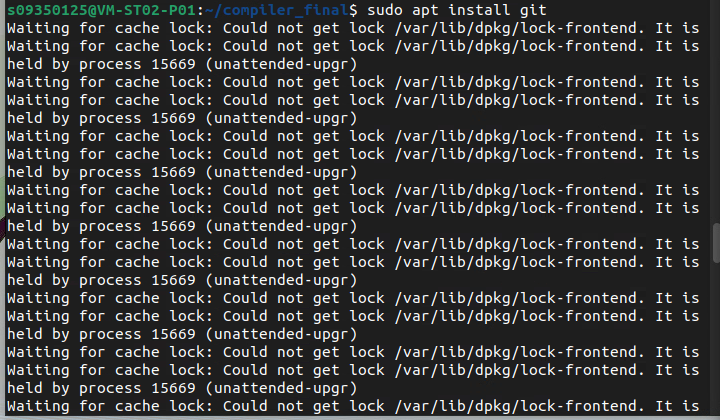


Bison: sudo -S apt-get install bison -y指令 → 安裝yacc(bison)工具



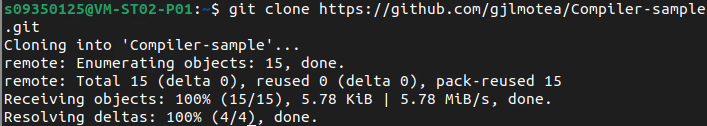
下載git:

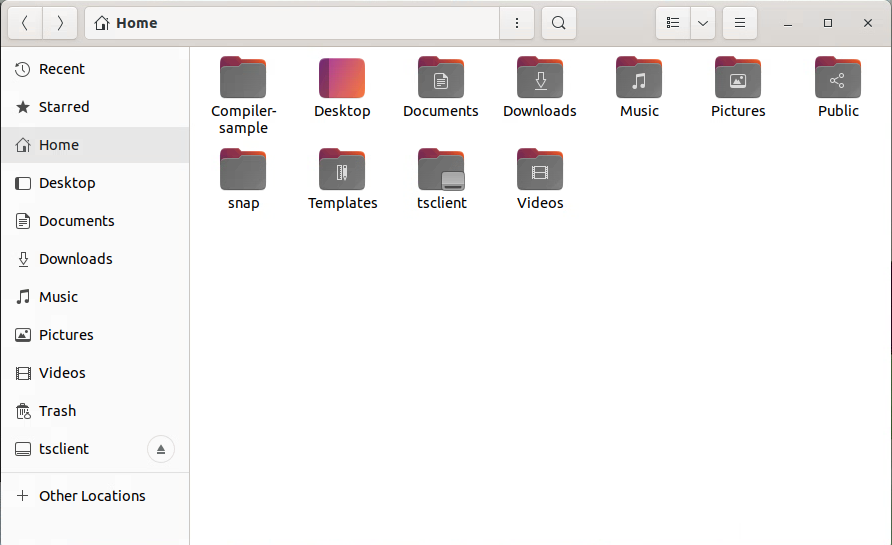
sudo apt install git → 到github 進行下載 Compiler\_sample



下載 <https://github.com/gjlmotea/Compiler-sample>裡面的檔案:

git clone https://github.com/gjlmotea/Compiler-sample.git

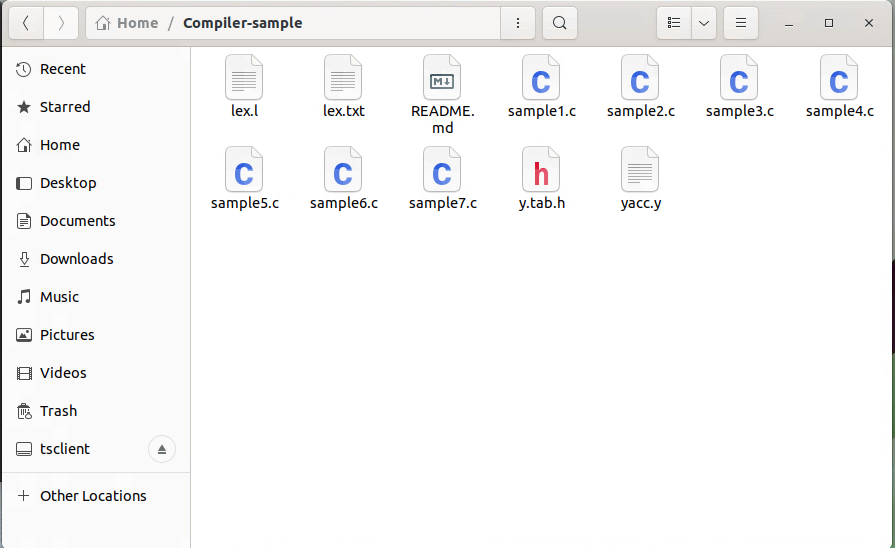




到Compiler-sample 目錄:

cd Compiler-sample → 到下載的資料夾中



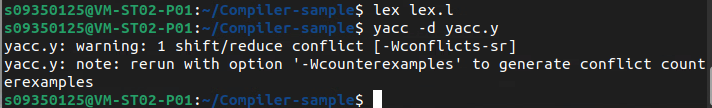


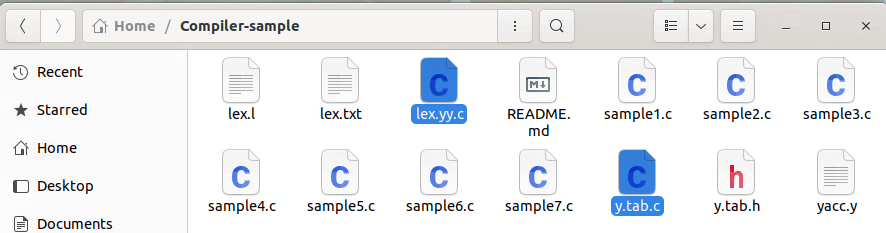
對lex.l使用Lex工具編譯→產生lex.yy.c

lex lex.l → 產生lex.yy.c

對yacc.y使用Yacc工具編譯→產生y.tab.c

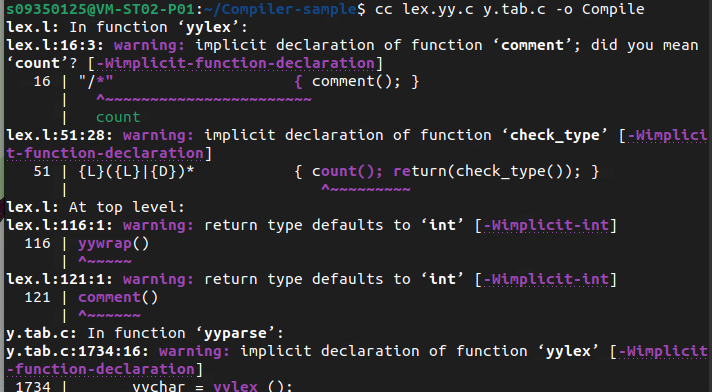
yacc -d yacc.y → 產生y.tab.c

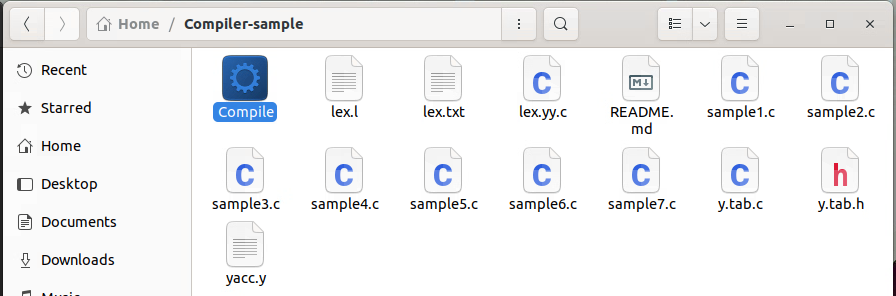




編譯lex.yy.c及y.tab.c兩個檔案，並產生檔名為Compile的可執行檔

cc lex.yy.c y.tab.c -o Compile → lex.yy.c 和 y.tab.c 合併起來，產生 Compiler 執行檔

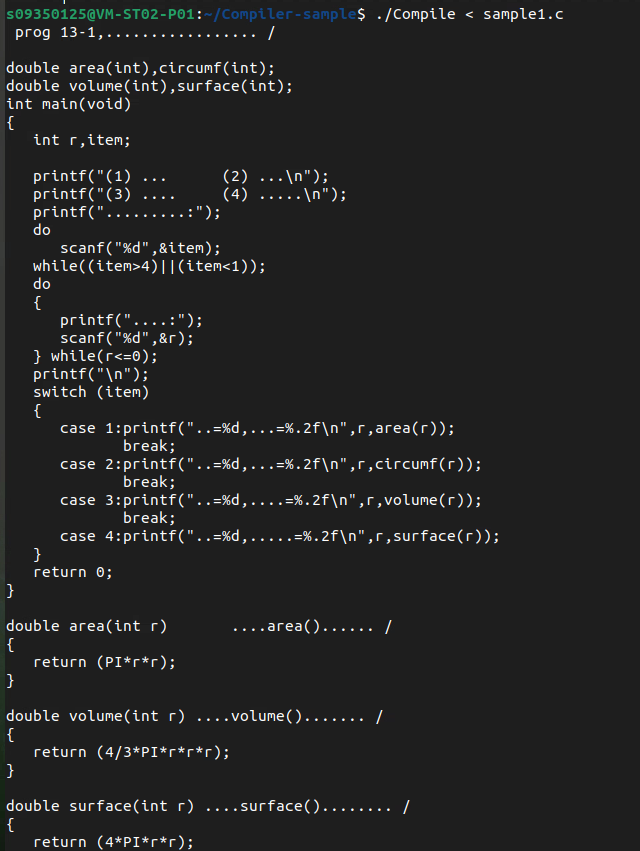


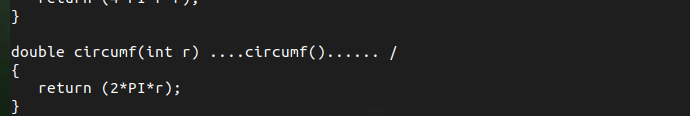


**Sample 測試:**

* Sample1測試

./Compile < sample1.c → 若編譯成功則不會產生 error

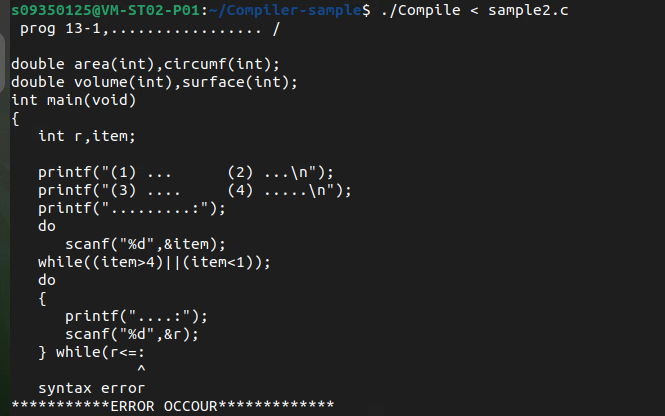
****

****

**\*** Sample1輸出結果: 無語法錯誤

* Sample2測試

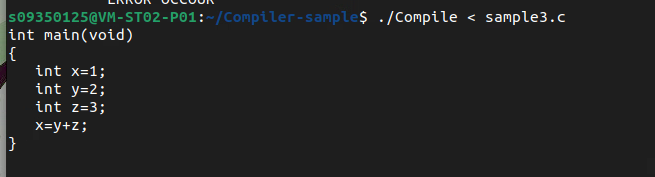
./Compile < sample2.c → 編譯失敗

****

\* Sample2輸出結果: 語法錯誤

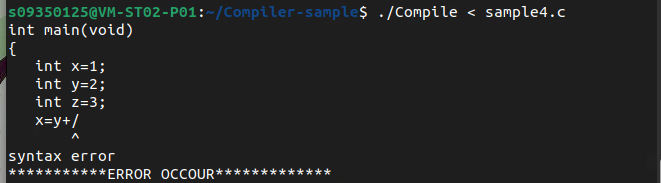
* Sample3測試

./Compile < sample3.c → 編譯成功

****

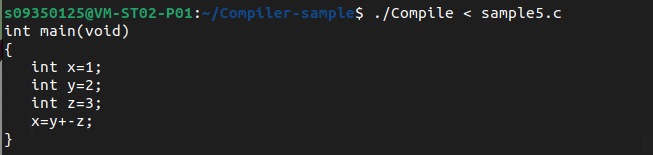
* Sample4測試

./Compile < sample4.c → 編譯失敗

****

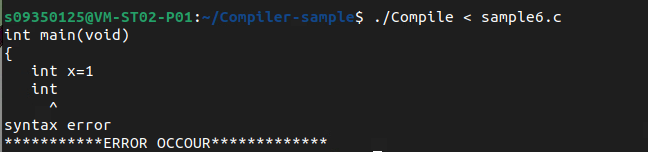
* Sample5測試

./Compile < sample5.c → 編譯成功

****

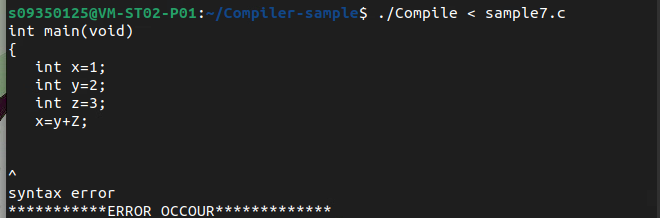
* Sample6測試

./Compile < sample6.c → 編譯失敗



* Sample7測試

./Compile < sample7.c → 編譯失敗

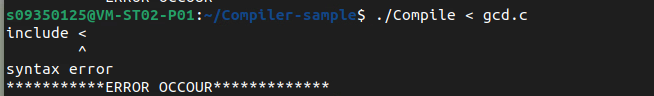


**自己寫的程式 gcd.c ->test**

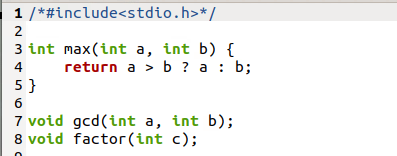
./Compile < gcd.c → 編譯失敗

編譯時發現問題:

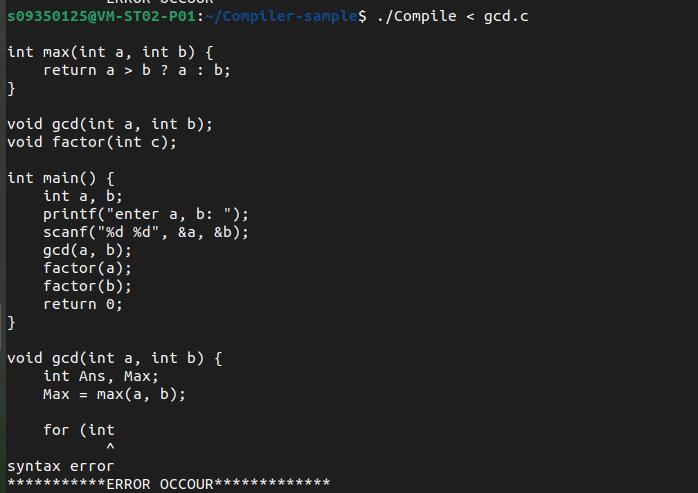
1. 標頭檔無法被編譯



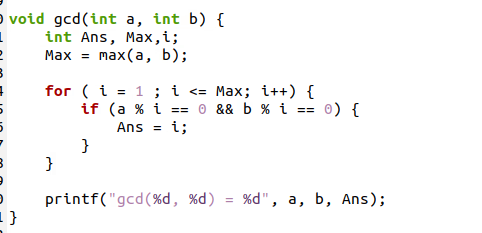
解決方法: 把標頭檔註解



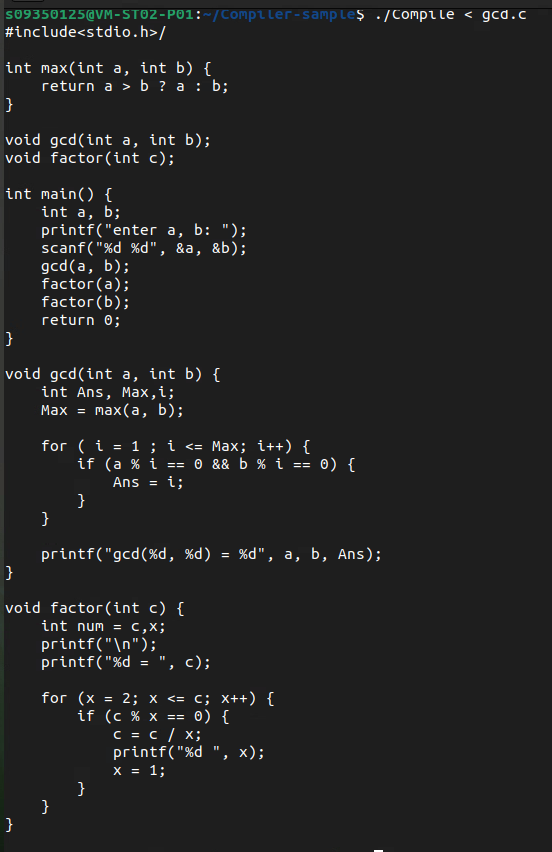
1. 宣告變數要放在最前面否則會被視為語法錯誤



解決方法: 宣告的值放在上面

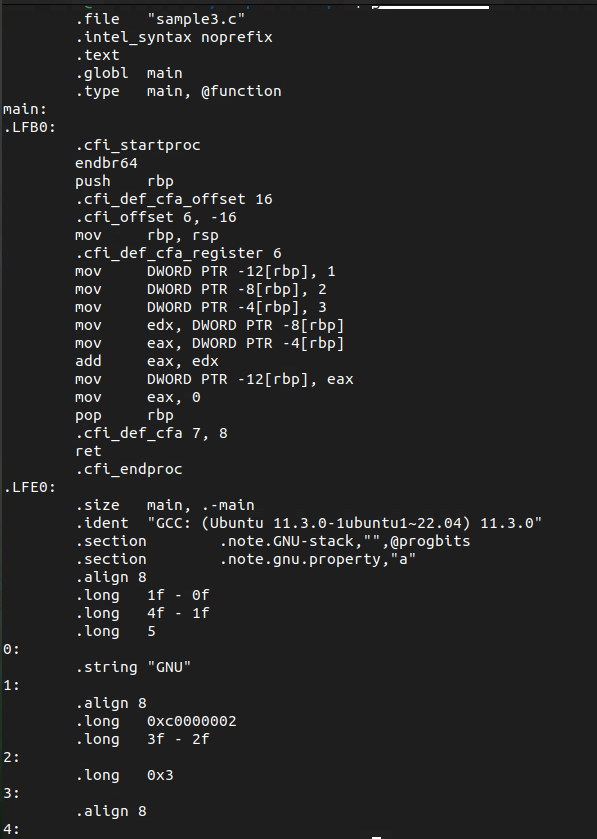


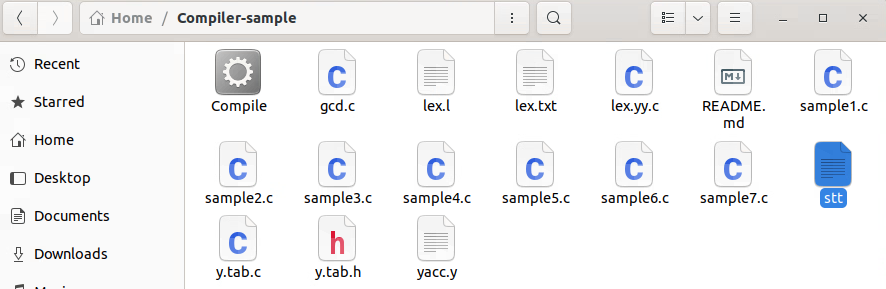
編譯成功:



產生符合intel 語法的組合語言，命名為stt，並顯示產生的stt內容

gcc -S -masm=intel sample3.c -o stt && cat stt

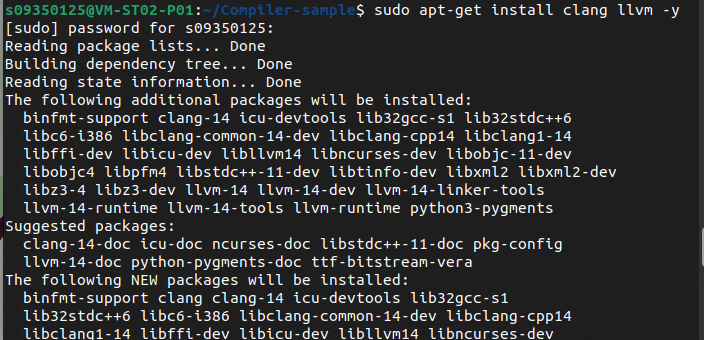




方法二：使用LLVM + Clang產生組合語言

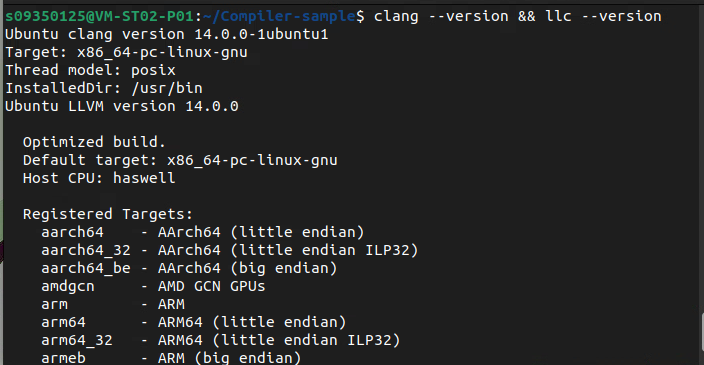
安裝clang與llvm:

sudo apt-get install clang llvm -y



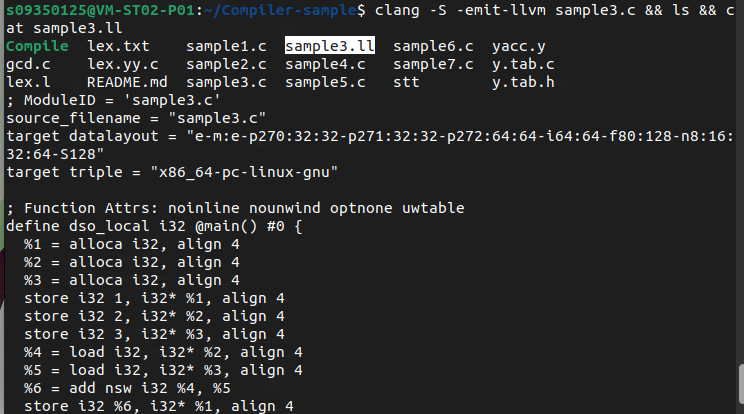
檢查clang與llvm成功安裝的版本

clang --version && llc --version



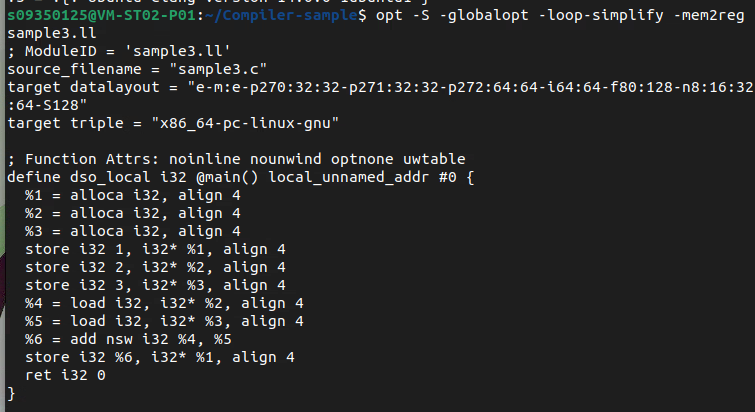
用clang產生sample3.c的IR、ls檢查檔案是否生成、顯示出產生的sample3.ll內容

clang -S -emit-llvm sample3.c && ls && cat sample3.ll



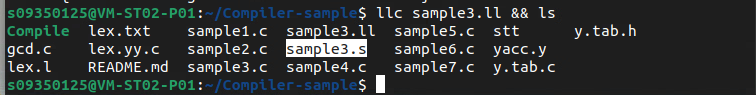
最佳化LLVMMem2Reg 的作用就是分析IR 中可以省略的記憶體操作，並且用register 取代它

opt -S -globalopt -loop-simplify -mem2reg sample3.ll



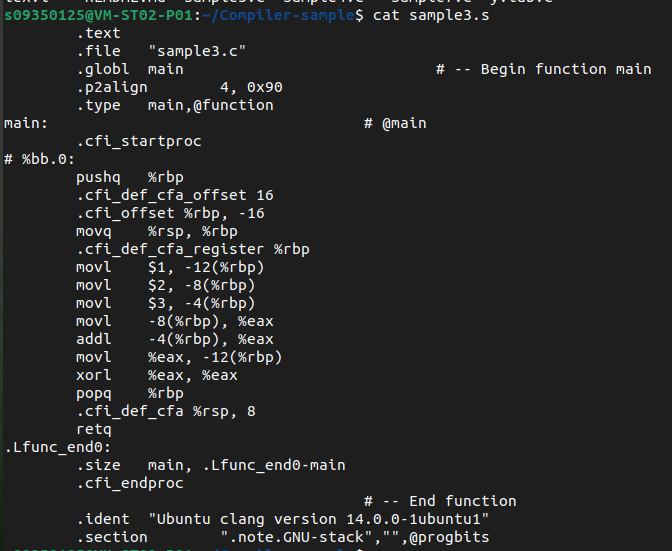
利用llc將IR編譯成組合語言、ls檢查檔案是否生成

llc sample3.ll && ls



顯示出產生的sample3.s內容

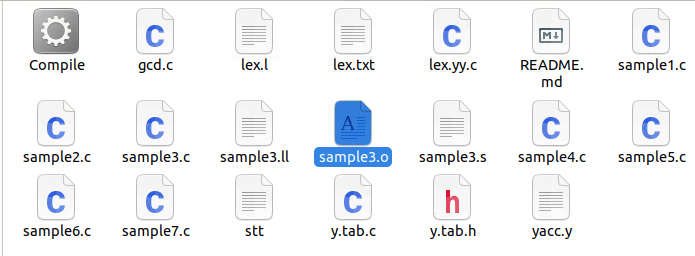
cat sample3.s



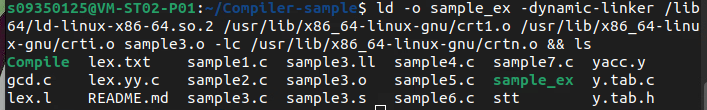
將sample3.s編譯成.o檔

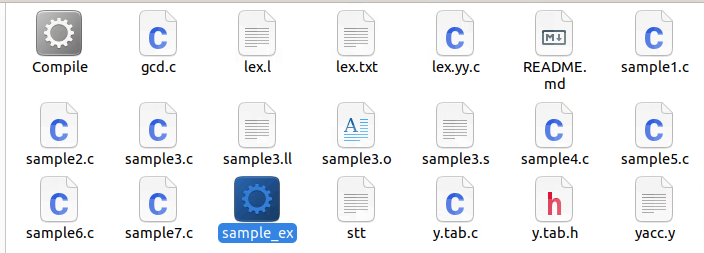
as sample3.s -o sample3.o





用ld串起library產生可執行output命名為sample\_ex、ls檢查檔案是否生成ld -o sample\_ex -dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib/x86\_64-linux-gnu/crt1.o /usr/lib/x86\_64-linux-gnu/crti.o sample3.o -lc /usr/lib/x86\_64-linux-gnu/crtn.o && ls





執行結果

./sample\_ex

