##### 2.6.4、安裝 C、C++、Fortran 語言編譯器（gcc、g++、gfortran）

Linux系統GCC編譯器官網：<https://gcc.gnu.org/>

mingw-w64編譯器官網：<http://www.mingw-w64.org/doku.php>

mingw-w64編譯器官網下載：<https://osdn.net/projects/mingw/releases/>

mingw-w64編譯器中科大鏡像下載：<http://mirrors.ustc.edu.cn/msys2/mingw/x86_64/>

日本北陸先端科學技術大學院大學SourceForge鏡像：<https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/>

MSYS系統官網：<http://mingw.org/wiki/msys>

MSYS系統中科大鏡像下載：<http://mirrors.ustc.edu.cn/msys2/msys/>

日本北陸先端科學技術大學院大學SourceForge鏡像：<https://sourceforge.net/projects/mingw/files/MSYS/Base/msys-core/>

MSYS2系統官網：<https://www.msys2.org/>

MSYS2系統中科大鏡像源：<https://lug.ustc.edu.cn/wiki/mirrors/help/msys2/>

MSYS2系統清華鏡像源：<https://mirrors.tuna.tsinghua.edu.cn/help/msys2/>

日本北陸先端科學技術大學院大學SourceForge鏡像：<https://sourceforge.net/projects/msys2/files/>

GSL庫官網：[http://www.gnu.org/software/gsl/#navigation](http://www.gnu.org/software/gsl/" \l "navigation)

GSL庫下載ftp服務器：<http://mirrors.nju.edu.cn/gnu/gsl/>

南京大學開源鏡像站：<http://mirrors.nju.edu.cn/gnu/gsl/>

官方手冊：<https://www.gnu.org/software/gsl/doc/html/>

適用於32位arm-v7架構處理器（CPU）上的Android 5.0系統的Termux v0.76模擬器Linux-Ubuntu-20.04-LTS-rootfs -armhf子系統；

Operating system :

Google-Pixel-7 Android-11 Termux-0.118 Ubuntu-22.04-LTS-rootfs Arm64-AArch64 MSM8998-Snapdragon835-Qualcomm®-Kryo™-280

gcc (aarch64-linux-gnu Ubuntu 22.04.1) 11.3.0

Operating system: Window10 x86\_64 Inter(R)-Core(TM)-m3-6Y30

gcc (x86\_64-posix-seh-rev0, Built by MinGW-w64 project) 8.1.0

因爲後邊安裝 Jupyter-Notebook 、Cython 、 Rcpp 、ggplot2 等軟件時，安裝相關依賴環境，會用到 C 語言的 gcc 編譯器和C++語言的 g++ 編譯器，所以最好最先配置好C語言的 gcc 、g++ 編譯器，然後再安裝其它軟件；其中在後面安裝Jupyter-Notebook的R語言支持内核（IRkernel.r）時，需要用到R語言中的「devtools」擴展包從Github網站下載安裝R支持内核（IRkernel.r）包，而R語言安裝「devtools」擴展包時又需要用到gfortran編譯器，因此最好也把Fortran語言的gfortran編譯器配置好；

###### ⑴、Linux系統使用 apt install 工具安裝 gcc 編譯器

安裝前先更新apt工具清單到最新版，運行命令：

root@localhost:~# apt update

Ubuntu 控制臺運行命令安裝 C 語言 gcc 編譯器：

root@localhost:~# apt install gcc

安裝完成後，運行命令查看 gcc 版本，同時驗證是否安裝成功：

root@localhost:~# gcc -v

如下編譯示例，創建一個示例C語言源文件，假設命名爲「Hello.c」,可以使用NaNo文檔編輯器直接創建，Ubuntu控制臺運行命令：

root@localhost:~# nano Hello.c

NaNo文檔編輯器創建文件「Hello.c」後，會直接打開「Hello.c」文件，輸入如下内容：

#include <stdio.h>

int main(){

printf("Ubuntu gcc say Hello World");

return 0;

}

輸入完畢後，同時按下「Ctrl」+「o」鍵保存文件，再同時按下「Ctrl」+「x」鍵退出文檔編輯器，返回到Ubuntu控制臺調用gcc編譯器，將「Hello.c」文本文件編譯成爲可執行文件「hello」：

root@localhost:~# gcc Hello.c -o hello

編譯完成後，會生成一個可執行的「hello」文件，Ubuntu控制臺輸入命令：

root@localhost:~# ./hello

可以運行這個新生成的可執行的「hello」文件，控制臺返回信息如下：

Ubuntu gcc say Hello World

Ubuntu 控制臺運行命令安裝 C++ 語言 g++ 編譯器：

root@localhost:~# apt install g++

安裝完成後，運行命令查看 g++ 版本，同時驗證是否安裝成功：

root@localhost:~# g++ -v

如下編譯示例，創建一個示例C++語言源文件，假設命名爲「Hello.cpp」,可以使用NaNo文檔編輯器直接創建，Ubuntu控制臺運行命令：

root@localhost:~# nano Hello.cpp

NaNo文檔編輯器創建文件「Hello.cpp」後，會直接打開「Hello.cpp」文件，輸入如下内容：

#include <stdio.h>

int main(){

printf("Ubuntu g++ say Hello World")

return 0;

}

輸入完畢後，同時按下「Ctrl」+「o」鍵保存文件，再同時按下「Ctrl」+「x」鍵退出文檔編輯器，返回到Ubuntu控制臺調用gcc編譯器，將「Hello.c」文本文件編譯成爲可執行文件「hello」：

root@localhost:~# gcc Hello.cpp -o hello

編譯完成後，會生成一個可執行的「hello」文件，Ubuntu控制臺輸入命令：

root@localhost:~# ./hello

可以運行這個新生成的可執行的「hello」文件，控制臺返回信息如下：

Ubuntu g++ say Hello World

也可以同時編譯多個文件，比如創建三個示例源文件，首先創建一個頭文件「speaker.h」，Ubuntu控制臺運行命令：

root@localhost:~# nano speaker.h

NaNo文檔編輯器創建文件「speaker.h」後，會直接打開頭文件「speaker.h」，輸入如下内容：

class speaker {

public:

void sayHello(const char \*);

};

輸入完畢後，同時按下「Ctrl」+「o」鍵保存文件，再同時按下「Ctrl」+「x」鍵退出文檔編輯器，返回到Ubuntu控制臺，創建文件「speaker.cpp」：

root@localhost:~# nano speaker.cpp

NaNo文檔編輯器創建文件「speaker.cpp」後，會直接打開「speaker.cpp」文件，輸入如下内容：

#include "speaker.h"

#include <iostream>

using namespace std;

void speaker::sayHello(const char \*str) {

cout << "Hello " << str << "\n";

}

輸入完畢後，同時按下「Ctrl」+「o」鍵保存文件，再同時按下「Ctrl」+「x」鍵退出文檔編輯器，返回到Ubuntu控制臺，創建文件「testspeaker.cpp」：

root@localhost:~# nano testspeaker.cpp

NaNo文檔編輯器創建文件「testspeaker.cpp」後，會直接打開「testspeaker.cpp」文件，輸入如下内容：

#include "speaker.h"

int main(int argc,char \*argv[]) {

speaker speak; //定義對象speak

speak.sayHello("world"); //調用成員函數sayHello

return 0;

}

輸入完畢後，同時按下「Ctrl」+「o」鍵保存文件，再同時按下「Ctrl」+「x」鍵退出文檔編輯器，返回到Ubuntu控制臺調用g++編譯器，將「speaker.h」、「speaker.cpp」、「testspeaker.cpp」文本文件編譯成爲可執行文件「testspeaker」：

root@localhost:~# g++ testspeaker.cpp speaker.cpp -o testspeaker

編譯完成後，會生成一個可執行的「testspeaker」文件，Ubuntu控制臺輸入命令運行這個新生成的可執行的「testspeaker」文件：

root@localhost:~# ./testspeaker

控制臺返回信息如下：

Hello World

Ubuntu 控制臺運行命令安裝 Fortran 語言 gfortran 編譯器：

root@localhost:~# apt install gfortran

安裝完成後，運行命令查看 gcc 版本，同時驗證是否安裝成功：

root@localhost:~# gfortran -v

能夠顯示 gcc、g++、gfortran 的版本號，沒有報錯就表示安裝成功。

Windows下C語言使用GCC編寫和調用動態鏈接庫dll

① 首先新建一個文件夾作爲項目空間

文件夾名就是項目名稱，注意文件夾名字中不要包含中文，所在路徑名中也不要包含中文，例如可以在桌面新建一個名爲DllTest的文件夾「C:\Users\china\Desktop\DllTest\」；

② 使用gcc編譯動態鏈接庫foo.dll文件, 供給另一個可執行程序DllTest.exe調用，執行DllTest.exe程序在控制臺換行打印輸出：foo.dll 字符；

1)、首先在這個「C:\Users\china\Desktop\DllTest\」文件夾（項目空間）中新建一個文本文檔，作爲被調用的動態鏈接庫的頭文件，將擴展名改爲「.h」，假設將這個文檔命名為「foo.h」，在文檔中寫入如下内容：

「

#ifndef FOO\_H

#define FOO\_H

void foo();

#endif

」

2)、然後在這個「C:\Users\china\Desktop\DllTest\」文件夾（項目空間）中再新建一個文本文檔，作爲被調用的動態鏈接庫的源代碼文件，將擴展名改爲「.c」，假設將這個文檔命名為「foo.c」，在文檔中寫入如下内容：

「

#include "foo.h"

#include <stdio.h>

void foo(){

printf("foo.dll\n");

}

」

3)、打開Window控制臺cmd（注意如果是Window10系統，最好鼠標右鍵單擊，然後選擇以管理員身份運行cmd控制臺），進入項目空間文件夾中：

C:\Users\china> cd c:\Users\china\Desktop\DllTest\

然後在cmd控制臺輸入如下命令編譯dll動態鏈接庫：

C:\Users\china\Desktop\DllTest> gcc --share foo.c -o foo.dll

等待編譯完成，可以看到「C:\Users\china\Desktop\DllTest\」文件夾（項目空間）中已經多了一個「foo.dll」動態鏈接庫文件；

③ 使用gcc編譯一個可執行的main.exe程序,調用剛才編譯成功的foo.dll動態鏈接庫文件；

1)、在這個「C:\Users\china\Desktop\DllTest\」文件夾（項目空間）中再新建一個文本文檔，作爲可執行程序DllTest.exe的源代碼文件，將擴展名改爲「.c」，假設將這個文檔命名為「main.c」，在文檔中寫入如下内容：

「

#include "foo.h"

#include <stdio.h>

int main(){

foo();

}

」

2)、打開Window控制臺cmd（注意如果是Window10系統，最好鼠標右鍵單擊，然後選擇以管理員身份運行cmd控制臺），進入項目空間文件夾中：

C:\Users\china> cd c:\Users\china\Desktop\DllTest\

然後在cmd控制臺輸入如下命令編譯dll動態鏈接庫：

C:\Users\china\Desktop\DllTest> gcc main.c foo.dll -o DllTest.exe

等待編譯完成，可以看到「C:\Users\china\Desktop\DllTest\」文件夾（項目空間）中已經多了一個「DllText.exe」可執行程序；

在cmd控制臺輸入如下命令按回車鍵「Enter」測試執行「DllText.exe」可執行程序：

C:\Users\china\Desktop\DllTest> DllText.exe

這時可見cmd控制臺打印出：foo.dll 字符，可見調用foo.dll成功；

###### ⑵、Linux系統手動下載 gcc 編譯器二進制可執行文件安裝配置

適用於32位arm-v7架構處理器（CPU）上的Android 5.0系統的Termux v0.76模擬器Linux-Ubuntu-20.04-LTS-rootfs -armhf子系統；

Operating system :

Google-Pixel-7 Android-11 Termux-0.118 Ubuntu-22.04-LTS-rootfs Arm64-AArch64 MSM8998-Snapdragon835-Qualcomm®-Kryo™-280

gcc (aarch64-linux-gnu Ubuntu 22.04.1) 11.3.0

###### ⑶、Windows10系統配置MinGW-w64編譯器和MSYS工具鏈

MinGW-w64編譯器官網：<http://www.mingw-w64.org/doku.php>

MinGW-w64編譯器官網下載：<https://osdn.net/projects/mingw/releases/>

MinGW-w64編譯器中科大鏡像下載：<http://mirrors.ustc.edu.cn/msys2/mingw/x86_64/>

日本北陸先端科學技術大學院大學SourceForge鏡像：<https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/>

MSYS系統官網：<http://mingw.org/wiki/msys>

MSYS系統中科大鏡像下載：<http://mirrors.ustc.edu.cn/msys2/msys/>

日本北陸先端科學技術大學院大學SourceForge鏡像：<https://sourceforge.net/projects/mingw/files/MSYS/Base/msys-core/>

Operating system: Window10 x86\_64 Inter(R)-Core(TM)-m3-6Y30

gcc (x86\_64-posix-seh-rev0, Built by MinGW-w64 project) 8.1.0

①、Windows10系統安裝配置MinGW-w64編譯器

開源下載站：<https://sourceforge.net/projects/mingw-w64/files/>

1)、下載 MinGW-w64 編譯器二進制可執行文件包

可以在開源軟件下載站sourceforge下載：

https://sourceforge.net/projects/mingw-w64/files/

但是國内限制了登錄這個網站，網速很慢，可以在中科大鏡像源下載：

http://mirrors.ustc.edu.cn/msys2/mingw/x86\_64/

如果可以，最好使用瀏覽器，在sourceforge網站下載官方標準版；

2)、或者同時按下「win」+「s」鍵打開搜索框，在搜索框中輸入「cmd」，鼠標左鍵單擊或右鍵菜單選擇「使用管理員身份運行」，啓動 cmd 控制臺命令行對話窗口；

然後在 cmd 控制臺使用 wget 命令獲取：

C:\> wget https://sourceforge.net/projects/mingw-w64/files/Toolchains%20targetting%20Win64/Personal%20Builds/mingw-builds/8.1.0/threads-posix/seh/x86\_64-8.1.0-release-posix-seh-rt\_v6-rev0.7z

等待下載完畢，會得到一個「x86\_64-8.1.0-release-posix-seh-rt\_v6-rev0.7z」壓縮包；

3)、在 cmd 控制臺使用 tar 命令解壓縮獲取到的壓縮包：

C:\> tar -xzvf x86\_64-8.1.0-release-posix-seh-rt\_v6-rev0.7z -C C:\

解壓完畢，在Windows10系統的「C:\」盤，會得到一個「mingw64」文件夾；

4)、將mingw64的可執行文件路徑「C:\mingw64\bin\」追加寫入到Windows10系統環境變量Path中，臨時對 Windows10 系統環境變量（Path）追加寫入，可以使用命令：

C:\> set Path=%Path%;C:\mingw64\bin\

臨時寫入的環境變量，退出 cmd 控制臺後即會失效，永久寫入 Windows10 系統環境變量（Path），可以使用命令：

C:\> setx "Path" "%Path%;C:\mingw64\bin\" /m

注意，對於Window10系統的 Path 變量中内容的修改，是追加，而不是替換，只在文件的末尾增加一行代碼即可，不要改動文件中原有的内容；

5)、至此在Windows10系統安裝Mingw-w64 編譯器即已配置成功，可以在 cmd 控制臺運行命令：

C:\> gcc --version

查看版本號驗證配置是否成功，使用命令：

C:\> gcc --help

查看幫助；

注意，因爲後面使用 MSYS 系統調用 MinGW-w64 編譯器編譯開源C代碼包的時候，MSYS 工具鏈會使用「make」命令，而 MinGW-w64 編譯器只有「mingw32-make」命令，因此需要將前面 MinGW-w64 編譯器自定義安裝路徑下的「mingw32-make.exe」文件，在同目錄複製一份，然後將文件名修改為「make.exe」以供 MSYS 工具鏈調用，例如，將自定義配置的 MinGW-w64 編譯器「bin」路徑下的「C:\mingw64\bin\mingw32-make.exe」文件複製一份改名為「C:\mingw64\bin\make.exe」保存；

②、Windows10系統安裝配置MSYS工具鏈

官網：<http://www.mingw.org/wiki/msys>

MSYS是Minimal MYStem的縮寫，它是一個開源的Shell系統，是Bourne Shell終端到Windows的輕量級的移植，包含了diff、sed、awk、sort等工具，主要用來配合MinGW進行Linux軟件的跨平臺移植開發；

MSYS DTK 是一個附加的開發工具包，包括Perl、autoconf、automake、libtool等工具，安裝配置到與MSYS共一個目錄裏即可，這樣配置完就可以使用了；

1)、下載 MSYS 工具鏈二進制可執行文件安裝包

MSYS 工具鏈基礎包安裝包，可以在開源軟件下載站sourceforge下載：

<https://sourceforge.net/projects/mingw/files/MSYS/Base/msys-core/msys-1.0.11/MSYS-1.0.11.exe/download?use_mirror=jaist>

MSYS 工具鏈基礎包綠色免安裝版，可以在開源軟件下載站sourceforge下載：

<https://sourceforge.net/projects/mingw/files/MSYS/Base/msys-core/>

MSYS 工具鏈附加開發工具箱二進制安裝包，可以在開源軟件下載站sourceforge下載：

<https://sourceforge.net/projects/mingw/files/Other/Unsupported/MSYS/msysDTK/msysDTK-1.0.1/msysDTK-1.0.1.exe/download?use_mirror=deac-riga>

但是國内限制了登錄這個網站，網速很慢，可以在中科大鏡像源下載：

<http://mirrors.ustc.edu.cn/msys2/msys/>

如果可以，最好使用瀏覽器，在sourceforge網站下載官方標準版；

2)、或者同時按下「win」+「s」鍵打開搜索框，在搜索框中輸入「cmd」，鼠標左鍵單擊或右鍵菜單選擇「使用管理員身份運行」，啓動 cmd 控制臺命令行對話窗口，然後在 cmd 控制臺使用 wget 命令獲取，下載 MSYS 工具鏈基礎包綠色免安裝版：

C:\> wget https://sourceforge.net/projects/mingw/files/MSYS/Base/msys-core/msys-1.0.11/msysCORE-1.0.11-bin.tar.gz/download

等待下載完畢，會得到一個「msysCORE-1.0.11-bin.tar.gz」的壓縮包；在 cmd 控制臺使用 tar 命令解壓縮獲取到的壓縮包：

C:\> tar -xzvf msysCORE-1.0.11-bin.tar.gz -C C:\mingw64\msys\

解壓完畢，在Windows10系統的「C:\mingw64\msys\」目錄下，會得到一個文件夾「msysCORE-1.0.11-bin」，將這個文件夾改名為「1.0」；在剛才配置的「C:\mingw64\msys\」文件夾下得到一個「C:\mingw64\msys\1.0\」文件夾；再將msys-1.0.11的批處理文件（.bat）路徑「C:\mingw64\msys\1.0\」追加寫入到Windows10系統環境變量Path中，臨時對 Windows10 系統環境變量（Path）追加寫入，可以使用命令：

C:\> set Path=%Path%;C:\mingw64\msys\1.0\

臨時寫入的環境變量，退出 cmd 控制臺後即會失效，永久寫入 Windows10 系統環境變量（Path），可以使用命令：

C:\> setx "Path" "%Path%;C:\mingw64\msys\1.0\" /m

注意，對於Window10系統的 Path 變量中内容的修改，是追加，而不是替換，只在文件的末尾增加一行代碼即可，不要改動文件中原有的内容；

注意，因爲使用 MSYS 系統調用 MinGW-w64 編譯器編譯開源C代碼包的時候，MSYS 工具鏈會使用「make.exe」命令，而MinGW-w64編譯器只有「mingw32-make.exe」命令，因此需要將前面 MinGW-w64 編譯器自定義安裝路徑下的「mingw32-make.exe」文件，在同目錄複製一份，然後將文件名修改為「make.exe」以供 MSYS 工具鏈調用，例如，將上面自定義配置的 MinGW-w64 編譯器「bin」路徑下的「C:\mingw64\bin\mingw32-make.exe」文件複製一份改名為「C:\mingw64\bin\make.exe」保存；

3)、或者下載 MSYS 工具鏈基礎包二進制安裝文件：

C:\> wget https://sourceforge.net/projects/mingw/files/MSYS/Base/msys-core/msys-1.0.11/MSYS-1.0.11.exe/download?use\_mirror=jaist

等待下載完畢，會得到一個「MSYS-1.0.11.exe」的Windows系統二進制可執行安裝文件；再下載 MSYS 工具鏈附加開發工具箱二進制安裝文件：

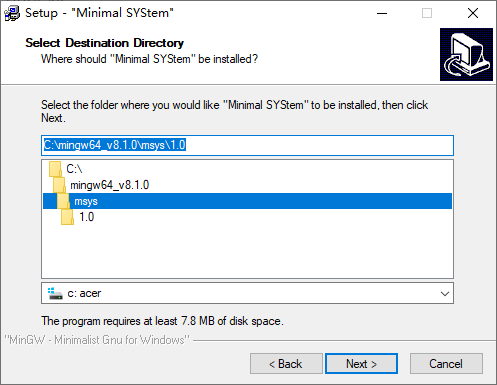
C:\> wget https://sourceforge.net/projects/mingw/files/Other/Unsupported/MSYS/msysDTK/msysDTK-1.0.1/msysDTK-1.0.1.exe/download?use\_mirror=deac-riga

等待下載完畢，會得到一個「msysDTK-1.0.1.exe」的Windows系統二進制可執行安裝文件；

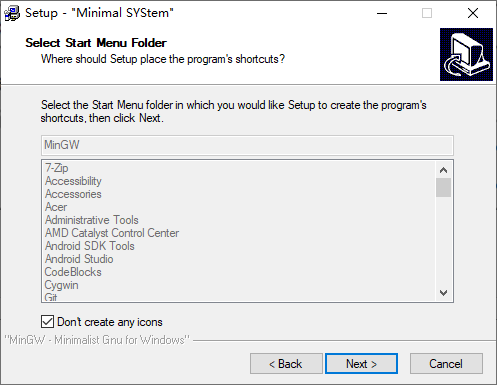
4)、在 cmd 控制臺或者鼠標左鍵雙擊，運行「MSYS-1.0.11.exe」安裝文件：

C:\> MSYS-1.0.11.exe

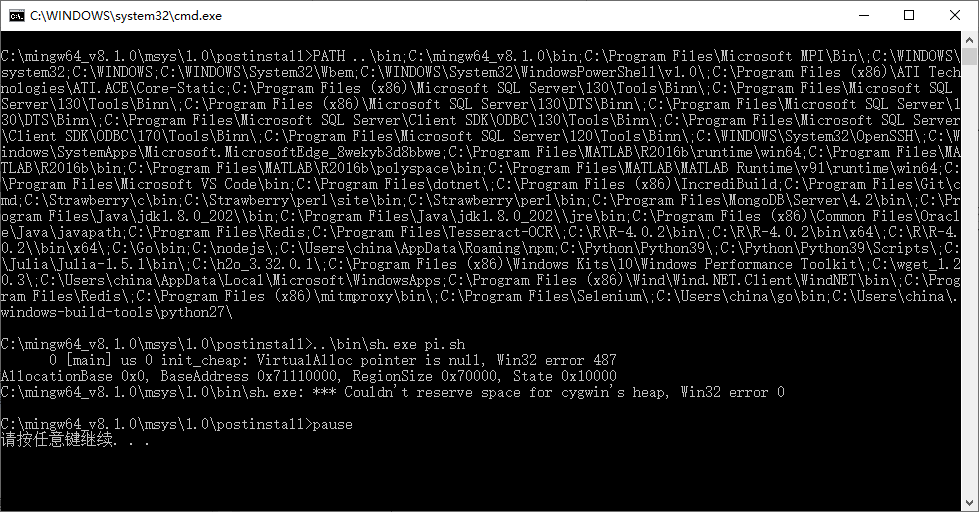
啓動 MSYS 工具鏈的安裝程序，注意安裝路徑，可以選擇在 Mingw-w64 編譯器的路徑下新建一個名為「..\msys\1.0\」的文件夾；



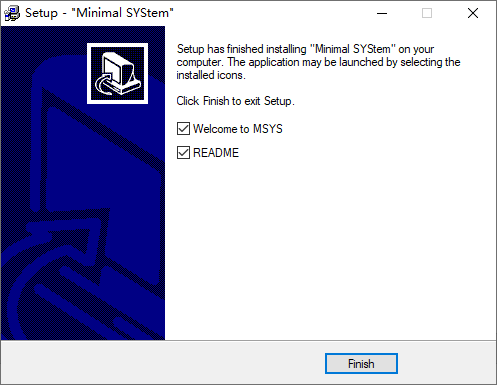
注意，可以勾選「Don’t create any icons」選項，不修改開始菜單：



等待安裝完畢：



安裝完畢，點擊「Finish」結束；



鼠標左鍵雙擊「C:\mingw64\msys\1.0\msys.bat」批處理文件，打開MSYS對話窗：



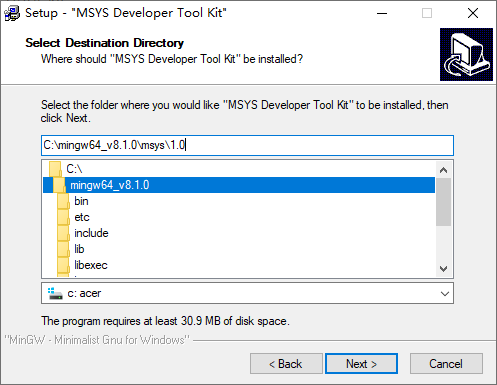
MSYS系統的根目錄是「/」，對應於Windows10系統自定義配置的MSYS工具鏈的安裝目錄，例如在這裏就是Windows10系統的「C:\mingw64\_v8.1.0\msys\1.0\」目錄；MSYS系統啓動時的默認起始目錄是「/home/Admin/」，例如在這裏對應的，就是Windows10系統的「C:\mingw64\_v8.1.0\msys\1.0\home\Admin\」目錄；需要注意，因爲MSYS與Windows10是兩個系統，MSYS系統有自己默認的目錄結構，如果使用Windows10系統的窗口鼠標右鍵新建文件夾，有時在MSYS系統中無法識別，MSYS控制臺使用「ls -a」命令會顯示不出來，同樣在MSYS系統中使用「mkdir」命令創建的目錄，有時在Windows10系統的窗口中也會顯示不出來，但是沒關係，進入MSYS系統後使用「ls -a」命令查看，創建的目錄能顯示出來，就表示創建的目錄存在可以在MSYS系統中使用；

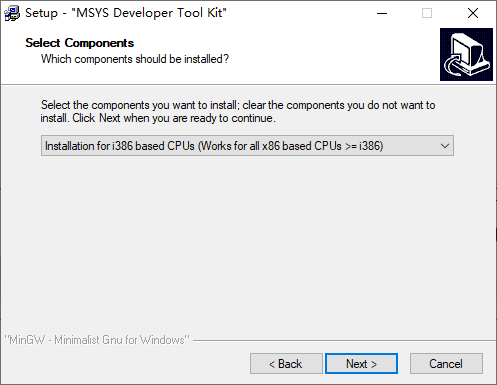
注意，因爲使用 MSYS 系統調用 MinGW-w64 編譯器編譯開源C代碼包的時候，MSYS 工具鏈會使用「make.exe」命令，而MinGW-w64編譯器只有「mingw32-make.exe」命令，因此需要將前面 MinGW-w64 編譯器自定義安裝路徑下的「mingw32-make.exe」文件，在同目錄複製一份，然後將文件名修改為「make.exe」以供 MSYS 工具鏈調用，例如，將上面自定義配置的 MinGW-w64 編譯器「bin」路徑下的「C:\mingw64\bin\mingw32-make.exe」文件複製一份改名為「C:\mingw64\bin\make.exe」保存；

5)、在 cmd 控制臺或者鼠標左鍵雙擊，運行「msysDTK-1.0.1.exe」安裝MSYS developer Tool Kit附加開發工具箱文件：

C:\> msysDTK-1.0.1.exe

啓動 MSYS 工具鏈附加開發工具箱的安裝程序，注意安裝路徑，需要選擇與前面配置的MSYS基礎包安裝路徑相同，比如像前面的「..\msys\1.0\」的文件夾；





等待安裝完畢，即可；

6)、將MSYS系統的啓動批處理文件（.bat）文件路徑「C:\mingw64\msys\1.0\msys.bat」追加寫入到Windows10系統環境變量Path中，臨時對 Windows10 系統環境變量（Path）追加寫入，可以使用命令：

C:\> set Path=%Path%;C:\mingw64\msys\1.0\

臨時寫入的環境變量，退出 cmd 控制臺後即會失效，永久寫入 Windows10 系統環境變量（Path），可以使用命令：

C:\> setx "Path" "%Path%;C:\mingw64\msys\1.0\" /m

注意，對於Window10系統的 Path 變量中内容的修改，是追加，而不是替換，只在文件的末尾增加一行代碼即可，不要改動文件中原有的内容；

5)、修改MSYS的配置文件「../msys/1.0/etc/fstab」，在MSYS中配置MinGW-w64編譯器；

在MSYS的安裝路徑「../msys/1.0/etc/」下，可以看到一個「fstab.sample」文件「../msys/1.0/etc/fstab.sample」，在同目錄下複製一個「fstab.sample」文件，重命名為「fstab」文件「../msys/1.0/etc/fstab」，打開這個「../msys/1.0/etc/fstab」文件，可以看到内容為：

「

#fstab.sample

#This is a sample file for /etc/fstab.

#Currently /etc/fstab is only read during dll initialization.

#I will eventually watch the directory for changes and reread the file.

#The line format is simple in that you give the Win32 path, followed by one or

#more space or tab delimiter, followed by the mount point. Mount points in

#typical UNIX environments must be a physical name on a drive before it can

#actually be used as a mount point. In this implementation the "must exist"

#requirement isn't enforced, however, it will be an aide to such programs as

#find and readline's tab completion if it does exist.

#You can use a # as the first character on the line as a comment indicator.

#Blank lines are ignored.

#Win32\_Path Mount\_Point

c:/mingw /mingw

c:/ActiveState/perl /perl

」

將其中的「c:/mingw /mingw」語句代碼，修改爲已經自定義安裝配置的 Mingw-w64 編譯器路徑：「c:/mingw64 /mingw」，保存退出即可；

6)、修改MSYS的配置文件「../msys/1.0/etc/profile」，配置在MSYS中使用MinGW-w64編譯器時優先查找自定義配置的64位庫文件目錄；

MSYS系統對應的是MinGW編譯器，默認編譯出的是amd32位CPU的Windows系統的可執行文件，MSYS的增强版MSYS2系統對應的是MinGW-w64編譯器，默認編譯出的是amd64位CPU的Windows系統的可執行文件，如果想用MinGW-w64編譯器配MSYS工具鏈編譯C、C++源代碼，可以進行如下配置；

啓動MSYS系統，MSYS系統啓動時的默認起始目錄是「/home/Admin/」，對應於Windows10系統自定義配置的MSYS工具鏈的安裝目錄下的「..\home\Admin\」文件夾，例如在這裏就是Windows10系統的「C:\mingw64\msys\1.0\home\Admin\」目錄；而需要創建的是MSYS系統的「/mingw64/local/」目錄，在這裏對應於Windows10系統就是「C:\mingw64\msys\1.0\mingw64\local\」目錄，需要注意，**是在MSYS系統下使用「mkdir」命令創建「/mingw64/local/」目錄，而不是在Windows10系統的窗口中鼠標右鍵新建文件夾**，因爲MSYS與Windows10是兩個系統，MSYS系統有自己默認的目錄結構，如果使用Windows10系統的窗口新建文件夾，有時在MSYS系統中無法識別，使用「ls -a」命令會顯示不出來是無效的，同樣在MSYS系統中使用「mkdir」命令創建的目錄，有時在Windows10系統的窗口中也會顯示不出來，但是沒關係，進入MSYS系統後使用「ls -a」命令查看，創建的目錄能顯示出來，就表示創建的目錄存在，可以在MSYS系統使用；

在MSYS控制臺使用「mkdir」命令創建「/mingw64/local/」目錄：

Admin@LAPTOP-4ON2LC2R ~

$ mkdir /mingw64/local/

然後在「/mingw64/local/」目錄下自定義分別創建如下64位庫文件目錄：

Admin@LAPTOP-4ON2LC2R ~

$ mkdir /mingw64/local/bin/

$ mkdir /mingw64/local/docs/

$ mkdir /mingw64/local/include/

$ mkdir /mingw64/local/lib/

$ mkdir /mingw64/local/share/

然後修改MSYS的配置文件「../msys/1.0/etc/profile」，在配置文件末尾追加寫入如下幾行代碼：

「

export LDFLAGS="-L/mingw64/local/lib -lpthread $LDFLAGS" # -lpthread required to compile GraphicsMagick.

export CFLAGS="-I/mingw64/local/include $CFLAGS"

export CPPFLAGS="-I/mingw64/local/include $CPPFLAGS"

export CXXFLAGS="-I/mingw64/local/include $CXXFLAGS"

export PATH="/mingw64/local/bin/:$PATH"

」

告訴MSYS中使用MinGW-w64編譯器時優先查找如上自定義配置的64位庫文件目錄；注意配置文件「../msys/1.0/etc/profile」是在末尾追加寫入，不要改動配置文件原有的内容；

然後將編譯時需要用到的64位庫文件放置在「../msys/1.0/mingw64/local/lib/」文件夾中，將頭文件放置在「../msys/1.0/mingw64/local/include/」文件夾中，將動態庫文件放置在「../msys/1.0/mingw64/local/bin/」文件夾中，在MSYS中使用MinGW-w64編譯器編譯C、C++源代碼時，就會優先查找自定義的庫文件目錄；

7)、將MSYS窗口啓動命令添加到Windows10系統任意路徑下的鼠標右鍵菜單中，實現MSYS右鍵打開當前目錄功能；

在Windows10系統中使用MinGW-w64編譯C、C++源代碼時，經常需要在MSYS系統中運行編譯命令，此時就需要在MSYS系統一路cd到project目錄，比較繁瑣，可以增加一個MSYS右鍵功能表，直接在目前的目錄啓動MSYS系統對話窗口，控制臺默認跳轉到當前路徑下；

7.1)、首先修改MSYS的配置文件「../msys/1.0/etc/profile」，打開「../msys/1.0/etc/profile」配置文件，可以看到裏邊内容為：

「

# Copyright (C) 2001, 2002 Earnie Boyd <earnie@users.sf.net>

# This file is part of the Minimal SYStem.

# http://www.mingw.org/msys.shtml

#

# File: profile

# Description: Shell environment initialization script

# Last Revised: 2002.05.04

if [ -z "$MSYSTEM" ]; then

MSYSTEM=MINGW32

fi

# My decision to add a . to the PATH and as the first item in the path list

# is to mimick the Win32 method of finding executables.

#

# I filter the PATH value setting in order to get ready for self hosting the

# MSYS runtime and wanting different paths searched first for files.

if [ $MSYSTEM == MINGW32 ]; then

export PATH=".:/usr/local/bin:/mingw/bin:/bin:$PATH"

else

export PATH=".:/usr/local/bin:/bin:/mingw/bin:$PATH"

fi

if [ -z "$USERNAME" ]; then

LOGNAME="`id -un`"

else

LOGNAME="$USERNAME"

fi

# Set up USER's home directory

if [ -z "$HOME" ]; then

HOME="/home/$LOGNAME"

fi

if [ ! -d "$HOME" ]; then

mkdir -p "$HOME"

cp /etc/inputrc.default "$HOME"/.inputrc

fi

if [ "x$HISTFILE" == "x/.bash\_history" ]; then

HISTFILE=$HOME/.bash\_history

fi

export HOME LOGNAME MSYSTEM HISTFILE

for i in /etc/profile.d/\*.sh ; do

if [ -f $i ]; then

. $i

fi

done

export MAKE\_MODE=unix

export PS1='\[\033]0;$MSYSTEM:\w\007

\033[32m\]\u@\h \[\033[33m\w\033[0m\]

$ '

alias clear=clsb

cd "$HOME"

」

將最後一行的啓動時進入HOME目錄的「cd "$HOME"」代碼注釋掉，修改為「# cd "$HOME"」，保存退出即可；

7.2)、然後再修改Windows10系統的注冊表項，將MSYS啓動文件加入到鼠標右鍵菜單中；

7.2.1)、首先同時按下「win」+「s」鍵打開運行框，輸入「regedit」點擊確認，打開Windows10系統的註冊表編輯器；

7.2.2)、在Windows10系統的注冊表編輯器中定位到「\HKEY\_CLASSES\_ROOT\Directory\Background\shell」路徑，然後在這個位置新建一個「msys」文件夾項，然後在「msys」文件夾下再新創建一個「command」項，將「command」項的預設值改為剛才自定義安裝配置的MSYS系統的安裝路徑下的「"..\msys\1.0\msys.bat"」啓動批處理文件（.bat）；

7.2.3)、添加右鍵菜單的顯示圖標，在Windows10系統的注冊表編輯器中定位到剛才新建的「msys」文件夾項「\HKEY\_CLASSES\_ROOT\Directory\Background\shell\msys」，新建字串值，命名為「icon」，預設值設置為剛才自定義安裝配置的MSYS系統的安裝路徑下的「"..\msys\1.0\msys.ico"」圖文件（.ico）；

保存退出，重啓或「F5」刷新，即可生效，鼠標右鍵即可顯示「MSYS」項；

###### ⑷、配置環境變量

Linux系統對於使用默認選項，安裝在「/usr/local/」文件夾下的可執行文件，不需要再人爲配置環境變量，Linux系統的環境變量（PATH）會默認搜索「/usr/local/bin/」路徑，所以安裝在默認文件夾「/usr/local/」下的可執行文件，在任意控制臺任意路徑都可以直接運行；

對於選擇自定義安裝路徑的情況，首先要確定 gcc 編譯器的自定義安裝目錄，可以在 gcc 編譯器的自定義安裝文件夾下，控制臺運行：

root@localhost:/usr/local/gcc/# pwd

控制臺會顯示出所在文件夾的絕對路徑，記下這個路徑，將這個安裝路徑下的可執行文件保存文件夾「bin」的絕對路徑，在後面追加寫入環境變量（PATH）中即可；如果安裝過程是按照上面的示例路徑，那麽控制臺應該會顯示：/usr/local/gcc/ 信息；

對於Ubuntu系統，臨時修改環境變量（PATH），可以在控制臺使用如下命令：

root@localhost:~# export PATH=$PATH:/usr/local/gcc/bin/

需要注意，Ubuntu系統控制臺使用 export 命令修改環境變量，只能在當前控制臺對話窗有效，退出重啓後，即失效；

永久修改環境變量，需要修改配置文件，對於 ubuntu20.04 root 環境，有兩個文件可以配置環境變數，其中「/etc/profile」文件是對所有用戶生效的；「/HOME/.profile」文件是對當前使用者生效的，使用NaNo文檔編輯器打開「/etc/profile」文件：

root@localhost:~# nano /etc/profile

假設 gcc 編譯器是自定義安裝在「/usr/local/gcc/」文件夾中，gcc 編譯器的可執行文件保存在「/usr/local/gcc/bin/」文件夾中，那麽就在配置文件「/etc/profile」的最後新增一行，追加寫入如下語句：

export PATH=$PATH:/usr/local/gcc/bin/

修改完畢后，同時按「Ctrl」+「o」鍵保存修改，按「Ctrl」+「x」鍵退出NaNo軟件返回到Ubuntu控制臺，執行「source」命令：

root@localhost:~# source /etc/profile

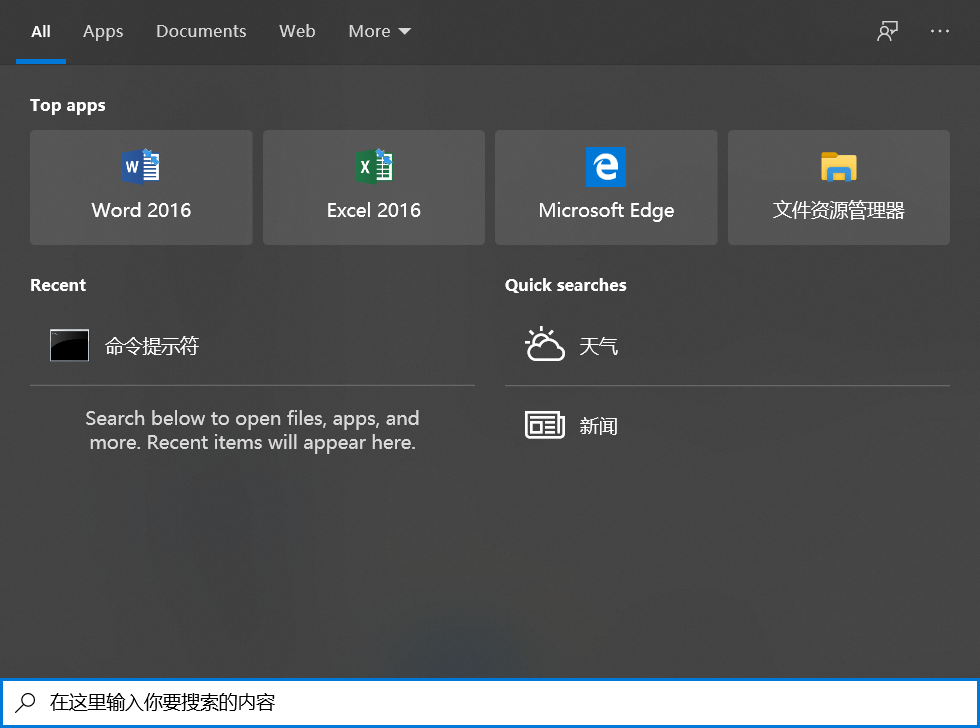
環境變量立刻生效，查看配置是否已經成功，在任意目錄下運行命令：

root@localhost:~# gcc

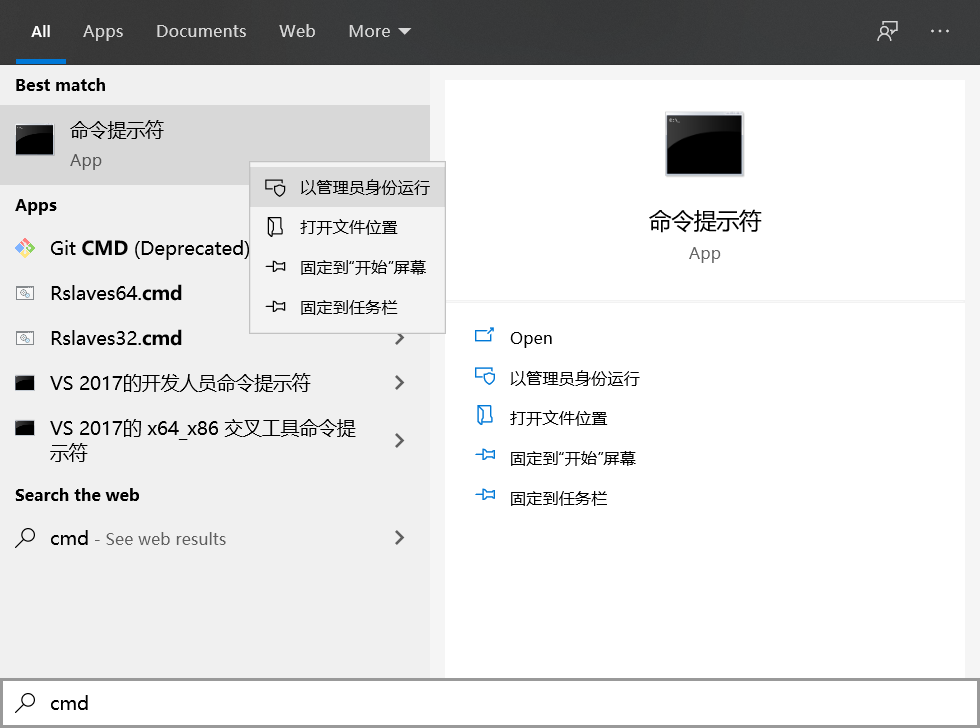
控制臺顯示 gcc 編譯器信息及版本號，則表示環境變量已經配置生效；

對於Window10系統，修改環境變量（Path）可以使用控制臺（cmd）命令行模式；

首先以管理員身份打開cmd控制臺窗口，在鍵盤同時按下「win」+「s」鍵，打開命令檢索輸入框：



在輸入框中輸入「cmd」三個字符，然後在彈出的檢索到的處，鼠標右鍵單機「cmd」可執行文件圖標，選擇「以管理員身份運行」選項，以管理員身份打開「cmd」控制臺：



然後在控制臺中參考如下操作；

Windows 系統臨時修改環境變量（Path）：

C:\> set Path=%Path%;C:\gcc\bin\

Windows 系統永久修改環境變量（Path）：

C:\> setx /m "Path" "%Path%;C:\gcc\bin\"

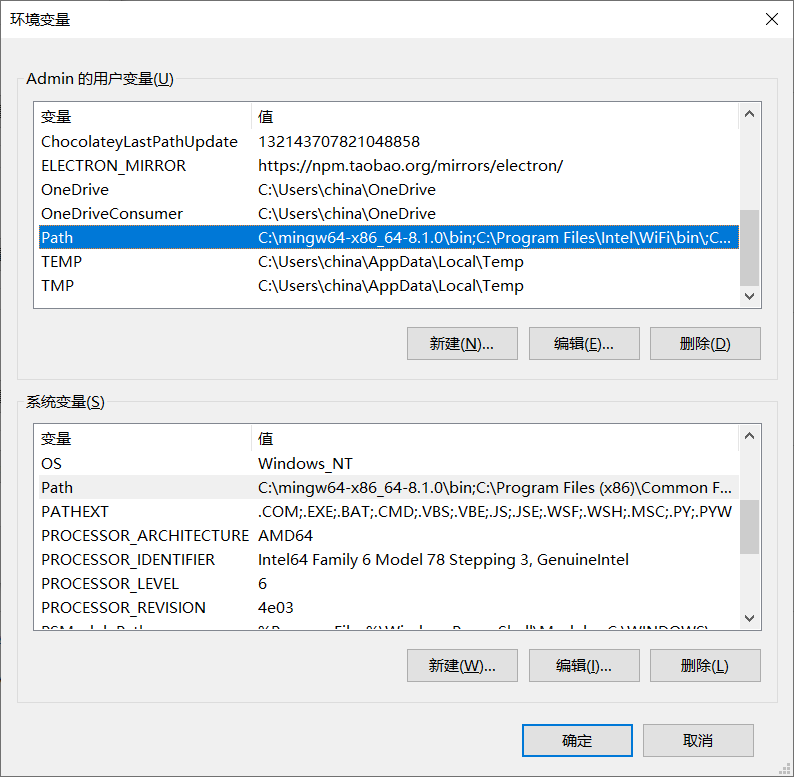
注意如果需要使用 setx 命令添加多個變量，要一次添加完，盡量不要對一個環境變量反復調用 setx 過程進行修改；

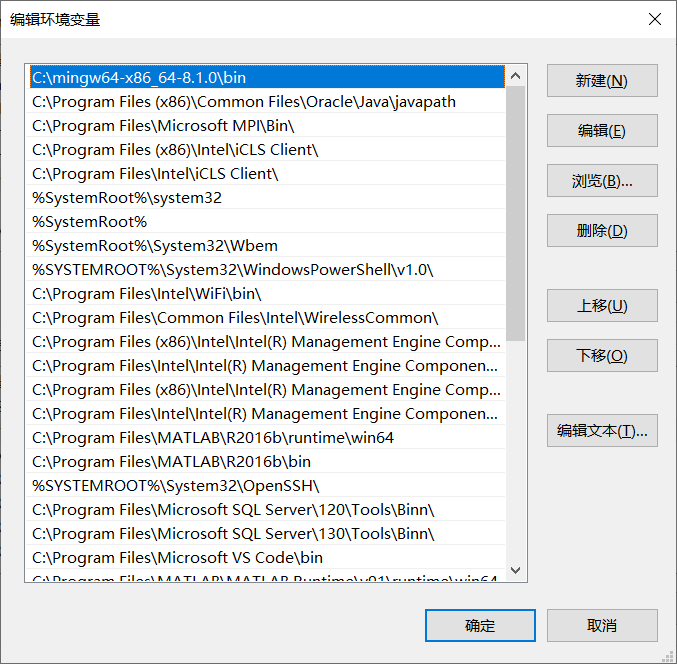
也可以使用窗口對話框修改：











###### ⑸、在 Linux 系統上 GCC 編譯器基本使用教程與範例

①、編譯與連結 C 程式

假設寫好一個 C 程式，常程式碼儲存在 hello.c 這個檔案中，而其內容如下：

「

// hello.c

#include <stdio.h>

int main() {

printf("Hello, world!\n");

return 0;

}

」

若要將 C 的原始碼編譯成執行檔，可以執行 gcc 並指定 C 語言的原始碼檔案：

「

# 編譯 C 程式

root@localhost:~# gcc hello.c

」

GCC 預設會執行編譯與連結，直接產生一個可以執行的程式，輸出至 a.out 這個檔案，完成編譯之後，即可執行這個程式：

「

# 執行編譯好的程式

root@localhost:~# ./a.out

」

控制臺會打印出：Hello, world!

若要指定輸出的執行檔名稱，可以加上 -o 參數，並指定輸出的檔案名稱：

「

# 編譯 C 程式，指定輸出檔名

root@localhost:~# gcc -o hello hello.c

# 執行編譯好的程式

root@localhost:~# ./hello

」

控制臺會打印出：Hello, world!

②、編譯與連結 C++ 程式

假設寫好一個 C++ 的程式儲存在 hello.cpp，內容如下：

「

// hello.cpp

#include <iostream>

using namespace std;

int main() {

cout << "Hello, world!" << endl;

return 0;

}

」

若要編譯 C++ 的程式，可以使用 g++ 這個編譯程式，並指定 C++ 的原始碼檔案：

「

# 編譯 C++ 程式

root@localhost:~# g++ hello.cpp

」

g++ 跟 gcc 非常類似，預設的輸出也是 a.out 這個檔案：

「

# 執行編譯好的程式

root@localhost:~# ./a.out

」

控制臺會打印出：Hello, world!

若要指定輸出的執行檔名稱，可以加上 -o 參數，並指定輸出的檔案名稱：

「

# 編譯 C++ 程式，指定輸出檔名

root@localhost:~# g++ -o hello hello.cpp

# 執行編譯好的程式

root@localhost:~# ./hello

」

控制臺會打印出：Hello, world!

③、GCC 常用參數

GCC 可用的參數不勝枚舉，這裡僅介紹一些常用的參數用法，以及實際的應用範例。

1)、除錯用參數

假設寫了一個 C 語言的程式如下：

「

// bug.c

#include <stdio.h>

int main() {

int v; // 未初始化

printf("v = %d\n", v);

return 0;

}

」

其中宣告了一個變數 v，但是卻在沒有定義的狀況下直接使用它，向這樣的程式通常都是有問題的，但是按照一般的編譯方式來說，編譯器並不會顯示錯誤，如果在開發過程想要讓編譯器協助偵測這類的問題，可以加上 -Wall 參數，讓編譯器顯示所有的警告訊息。

而通常在開發與除錯的階段，都會使用 gdb 來除錯，所以也會加上 -g 參數（如果沒加的話，就無法用 gdb 除錯）。

所以在開發階段，會很常用到像這樣的指令來編譯程式：

「

# 顯示警告訊息、加入除錯資訊

root@localhost:~# g++ -Wall -g -o hello hello.cpp

」

2)、定義巨集（Macro）

在開發中大型的程式時，開發者通常會使用一些巨集來處理各種的問題，例如除錯時會需要輸出詳細的訊息，但正式版的程式又不需要，這時候就可以這樣寫：

「

// macro.c

#include <stdio.h>

int main() {

printf("Hello, world!\n");

#ifdef DEBUG

printf("DEBUG is defined.\n");

#endif

return 0;

}

」

這段程式中，用 #ifdef 判斷 DEBUG 這個變數是否有被定義，若有被定義的話，就加入輸出除錯的程式碼，這樣一來就可以透過 DEBUG 的定義，來決定是否要產生有除錯訊息的執行檔。

若要定義 DEBUG 這個變數，可將定義直接加在程式碼中：

「

// 定義 DEBUG

#define DEBUG

」

也可以直接透過 GCC 的 -D 參數來定義：

「

# 定義 DEBUG

root@localhost:~# gcc -DDEBUG macro.c

」

兩種方式的效果相同，不過使用 GCC 的參數來定義這種時常需要調整的變數，通常會方便很多。

3)、只編譯不連結

GCC 預設會將 C 的原始碼編譯並連結，產生執行檔，若想讓編譯器只進行編譯、不要連結，可加上 -c 參數，這樣就會建立一個 object 檔：

「

# 僅編譯、不連結，建立 obj 檔案

root@localhost:~# gcc -c hello.c

」

執行這行之後，就會產生一個 hello.o 檔案，後續若要進行連結，就可以使用這個 object 檔：

「

# 連結產生執行檔

root@localhost:~# gcc -o hello hello.o

」

通常在大型的專案中，都會將編譯與連結兩個動作拆開，以下是一個典型的例子：

「

# 編譯個別 C 檔案

root@localhost:~# gcc -c a.c

root@localhost:~# gcc -c b.c

root@localhost:~# gcc -c c.c

# 連結

root@localhost:~# gcc -o myapp a.o b.o c.o

」

這樣作除了可讓程式碼方便管理之外，也可以加快編譯的速度，假設我們在開發過程中，更改了 b.c 的內容，在重新編譯時就只要編譯 b.c，然後即可進連結，省去重新編譯 a.c 與 c.c 的時間。

4)、最佳化

GCC 可以根據 CPU 的架構，進行最佳化處理，編譯出效能更好的執行檔，可用的選項有 -O（跟 -O1 相同）、-O2 與 -O3，數字越高代表最佳化的程度越高，許多專案在編譯正式版的時候，都會使用 -O2 進行最佳化。

「

# 進行最佳化

root@localhost:~# gcc -O2 -o hello hello.c

」

5)、標頭檔與函式庫路徑

在編譯程式時，編譯器需要許多標頭檔（\*.h 檔案）來編譯原始碼，而連結的時候則會需要一些函式庫（\*.a、\*.so 等）才能進行連結，但是編譯器只會自動引入一些系統預設的檔案，在大型專案中開發者會需要指定許多額外的標頭檔與函式庫位置，這樣才能讓編譯器順利編譯與連結。

編譯器會在 include 路徑中，搜尋 C 程式碼中以 #include 所引入的標頭檔，如果需要指定額外的搜尋路徑，可以使用 -I 參數增加搜尋路徑，假設我們有一些標頭檔放在 /home/gtwang/include 目錄下，而要新增這個路徑就可以這樣寫：

「

# 新增標頭檔搜尋路徑

root@localhost:~# gcc -I/home/gtwang/include -o hello hello.c

」

由於標頭檔的名稱都寫在 C 程式碼中，所以檔名都是已知的，只需要告訴編譯器路徑即可進行編譯。

在連結時期所需要函式庫名稱與路徑，可以透過 GCC 的 -l 與 -L 參數來指定。在 Linux 中假設有一個 libsum.a 函式庫放在 /home/gtwang/lib 目錄下，若要將其納入連結，則可以這樣寫：

「

# 納入指定的函式庫

root@localhost:~# gcc -lsum -L/home/gtwang/lib -o hello hello.c

」

###### ⑹、使用 gcc 自製 C/C++ 靜態、共享與動態載入函式庫教程

在開發不同的程式時，如果有一些程式碼是重複會用到的話，就可以將其製作成獨立的函式庫，不僅維護上更方便，也可以方便其他專案或是開發者使用，以下介紹以 gcc 還有 ar 等工具製作靜態、共享與動態載入函式庫的步驟。

①、範例程式

這裡我們用一個很簡單的 C 語言函數來做示範，假設我們有一個計算總和的 sum 函數，不同的專案或開發者都會需要用到它，sum.c 的內容如下：

「

#include "sum.h"

double sum(double a, double b) {

return a + b;

}

」

其對應的標頭檔（header file）sum.h 內容如下：

「

#ifndef \_\_SUM\_H\_\_

#define \_\_SUM\_H\_\_

double sum(double a, double b);

#endif

」

這是一個簡單的 main.c 使用範例，其內容如下：

「

#include <stdio.h>

#include "sum.h"

int main(int argc, char \*argv[]) {

double a = 2.6, b = 4.2, c;

c = sum(a, b);

printf("%.1f + %.1f = %.1f\n", a, b, c);

return 0;

}

」

若就一般正常的編譯程序，我們會使用 gcc 直接編譯所有的 C 檔案，連結後產生執行檔：

「

root@localhost:~# gcc -o main main.c sum.c

」

編譯完成後，執行之：

「

root@localhost:~# ./main

」

控制臺顯示：2.6 + 4.2 = 6.8

如果 sum.c 的內容只有在這一個專案中會用到，那麼這樣直接編譯的做法是最簡單又省事的。

如果有其他的專案也會用到 sum.c 裡面函數的話，最直接的作法就是把 sum.c 與 sum.h 兩個原始碼檔案複製一份到新的專案中，然後也用同樣的編譯方式如法泡製，但是這樣的缺點有兩個，一個就是每次都要重新編譯 sum.c 的內容，比較量費時間，另外就是未來若要持續維護 sum.c 的內容時，就需要同時修改好多個副本，既容易出錯也很麻煩。

另外如果 sum.c 的程式碼有機密性的問題，也就是說只想讓別人用，但是不想給原始碼的時候，也沒辦法直接用這種方式處理。

②、靜態函式庫

靜態函式庫（static library）就是由一些物件檔案（object files）所構成的封裝檔，通常其檔案名稱都會以 lib 開頭，而副檔名則為 .a。

使用靜態連結函式庫的好處就是所有的程式都包裝在執行檔中，不會因為缺少函式庫的檔案而不能執行，不過缺點就是這樣的執行檔大小會比較大，而如果函式庫有更新的話，整個執行檔也要跟著重新編譯。

以下我們示範將 sum.c 的內容製作成一個靜態函式庫的步驟。

1、編譯 sum.c 的程式碼：

「

root@localhost:~# gcc -c -o sum.o sum.c

」

2、使用 ar 指令將 sum.o 打包成 libsum.a 這個靜態連結函式庫：

「

root@localhost:~# ar -rcs libsum.a sum.o

」

3、使用 libsum.a 這個靜態連結函式庫：

「

root@localhost:~# gcc main.c -L. -lsum -o main\_static

」

也可以直接把 libsum.a 這個靜態連結函式庫檔案放進去編譯：

「

root@localhost:~# gcc main.c libsum.a -o main\_static

」

這樣就可以將 main.c 這個程式編譯成 main\_static 這個執行檔，而在編譯的過程中就不需要 sum.c 這個原始碼的檔案。

使用靜態連結函式庫所編譯出來的執行檔可以獨立執行，不需要原本的 libsum.a 檔：

「

root@localhost:~# ldd main\_static

」

控制臺會打印出如下信息：

linux-vdso.so.1 => (0x00007fffb0d1a000)

libc.so.6 => /lib64/libc.so.6 (0x00007f0411e46000)

/lib64/ld-linux-x86-64.so.2 (0x00007f0412220000)

③、共享函式庫

共享函式庫（shared library）是在程式實際開始執行時，才會被載入的函式庫，執行檔本身與共享函式庫是分離的，這樣可以讓執行檔的大小比較小，而且未來共享函式庫在更新之後，執行檔也不需要重新編譯，而缺點則是執行檔在執行時就會需要共享函式庫的檔案，如果缺少了共享函式庫的檔案，就會無法執行。

共享函式庫的檔案通常也是以 lib 開頭，但是其副檔名則為 .so，以下是建立共享函式庫的步驟。

1、編譯時加入 -fPIC 參數，產生共享函式庫所需要的 position independant code：

「

root@localhost:~# gcc -c -fPIC -o sum.o sum.c

」

2、使用 gcc 建立共享函式庫：

「

root@localhost:~# gcc -shared -Wl,-soname,libsum.so.1 -o libsum.so.1.0.0 sum.o

」

這裡的 -Wl 是用來將一些參數設定傳給連結器（linker），所以之後的 -soname 等參數就是傳給連結器的參數。

-soname 指定為 libsum.so.1 是代表函式庫的名稱，以 lib 開頭，接著是自己取的名稱，最後加上 .so 與 version 版本號碼，這一個 version 版本號碼所代表的是函示庫的介面版本，如果介面有改變時就會增加 version 版本號碼，以維護相容性的問題。

而最後產生的實際檔案名稱也跟 soname 類似，不過後面多了 minor 與 release 版本號碼，中間的 minor 號碼是用於標示新增加的介面，而最後面的 release 號碼則是用於程式內容的修正（介面不變的情況）。

如果程式使用到這個共享函式庫，則在執行時就會依據 soname 所指定的名稱來尋找函式庫的檔案，如果想要看共享函式庫的 soname 屬性，可以使用 objdump 指令：

「

root@localhost:~# objdump -p libsum.so.1.0.0 | grep SONAME

」

控制臺會打印出：SONAME libsum.so.1

建立好共享函式庫之後，要建立一個不含版本號碼的 .so 連結檔，gcc 在連結時所需要的函式庫檔案是這一個：

「

root@localhost:~# ln -s libsum.so.1.0.0 libsum.so

」

另外再建立一個執行時要用的連結檔：

「

root@localhost:~# ln -s libsum.so.1.0.0 libsum.so.1

」

使用共享函式庫來編譯執行檔：

「

root@localhost:~# gcc main.c -L. -lsum -o main\_dynamic

」

或是這樣編譯也可以：

「

root@localhost:~# gcc main.c libsum.so -o main\_dynamic

」

編譯完成後，要執行時需要指定 LD\_LIBRARY\_PATH：

「

root@localhost:~# LD\_LIBRARY\_PATH=. ./main\_dynamic

」

庫文件的内容是：2.6 + 4.2 = 6.8

這一個 main\_dynamic 執行檔在執行時，會需要 libsum.so.1 這一個共享函式庫檔案：

「

root@localhost:~# LD\_LIBRARY\_PATH=. ldd main\_dynamic

」

庫文件的内容是：

linux-vdso.so.1 => (0x00007ffc5f1f7000)

libsum.so.1 => ./libsum.so.1 (0x00007efd0148d000)

libc.so.6 => /lib64/libc.so.6 (0x00007efd010b3000)

/lib64/ld-linux-x86-64.so.2 (0x00007efd01690000)

④、動態載入函式庫（.so/.dll）

動態載入函式庫（dynamically loaded library）就類似 Windows 的 dll 檔，是等到程式真正要用到時才會載入函式庫，其實作方式是透過 DL 函式庫配合一般的共享函式庫來處理的，以下是將之前製作的 libsum.so.1 共享函式庫改為動態載入的範例。

「

#include <stdlib.h>

#include <stdio.h>

#include <dlfcn.h>

int main(int argc, char \*\*argv) {

void \*handle;

double (\*sum)(double, double);

char \*error;

// 動態開啟共享函式庫

handle = dlopen ("libsum.so.1", RTLD\_LAZY);

if (!handle) {

fputs (dlerror(), stderr);

exit(1);

}

// 取得 sum 函數的位址

sum = dlsym(handle, "sum");

if ((error = dlerror()) != NULL) {

fputs(error, stderr);

exit(1);

}

// 使用 sum 函數

double a = 2.6, b = 4.2, c;

c = sum(a, b);

printf("%.1f + %.1f = %.1f\n", a, b, c);

// 關閉共享函式庫

dlclose(handle);

return 0;

}

」

將這段程式碼儲存為 main\_dl.c 之後，按照一般的方式進行編譯：

「

root@localhost:~# gcc main\_dl.c -ldl -o main\_dl

」

在使用上跟共享函式庫差不多，一樣要指定 LD\_LIBRARY\_PATH：

「

root@localhost:~# LD\_LIBRARY\_PATH=. ./main\_dl

」

庫文件内容為：2.6 + 4.2 = 6.8

表面上看起來跟一般的共享函式庫類似，不過這種動態載入函式庫方式與共享函式庫有很大的不同，共享函式庫是在程式一開始執行時就要載入（不管實際上有沒有使用到），而動態載入函式庫的做法則是可以在真正需要用到時才載入（如果沒用到就可以不需要載入）。

###### ⑺、C、C++ 語言應用工程示例

①、使用函數：FILE \*popen(const char \*command, const char \*type)，調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）

首先，編寫譯程式設計 C 語言代碼檔（c2exe.c），使用函數：FILE \*popen(const char \*command, const char \*type)，調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）。

然後，在 Windows10 x86\_64系統，使用 MinGW-w64 編譯器，編譯程式設計 C 語言代碼檔（.c），生成二進制可執行檔（.exe）。

最後，再啓動編譯生成的二進制可執行檔（.exe），調用運行外部二進制可執行檔（.exe）。

Window10 x86\_64 系統 cmd 控制臺命令列編譯指令：

C:\Criss> C:\MinGW64\bin\gcc.exe C:/Criss/c/c2exe.c -o C:/Criss/c2exe.exe

Window10 x86\_64 系統 cmd 控制臺命令列顯示中文字符指令;

C:\Criss> chcp 65001

Window10 x86\_64 系統 cmd 控制臺命令列運行指令：

C:\Criss> C:/Criss/c2exe.exe configFile=C:/Criss/config.txt isBlock=true

程式設計 C 語言代碼檔「c2exe.c」的內容如下：

「

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Title: C call executable(.exe) v20161211

// Explain: C startup julia server

// Author: 趙健

// E-mail: 283640621@qq.com

// Telephont number: +86 18604537694

// E-mail: chinaorcaz@gmail.com

// Date: 歲在丙申

// Operating system: Window10 x86\_64 Inter(R)-Core(TM)-m3-6Y30

// Compiler: gcc (x86\_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0

// Operating system: Google-Pixel-2 Android-11 Termux-0.118 Ubuntu-22.04-LTS-rootfs Arm64 MSM8998-Snapdragon835-Qualcomm®-Kryo™-280

// Compiler: gcc (aarch64-linux-gnu Ubuntu 22.04.1) 11.3.0

// Executable: Julia server

// 使用説明：

// Windows x86\_64 系統 cmd 控制臺命令列編譯指令：

// C:\Criss> C:\MinGW64\bin\gcc.exe C:/Criss/c/c2exe.c -o C:/Criss/c2exe.exe

// Windows x86\_64 系統 cmd 控制臺命令列顯示中文字符指令;

// C:\Criss> chcp 65001

// Windows x86\_64 系統 cmd 控制臺命令列運行指令：

// C:\Criss> C:/Criss/c2exe.exe configFile=C:/Criss/config.txt executableFile=C:/Criss/Julia/Julia-1.9.3/bin/julia.exe interpreterFile=-p,4,--project=C:/Criss/jl/ scriptFile=C:/Criss/jl/application.jl configInstructions=interface\_Function=file\_Monitor,webPath=C:/Criss/html/,host=::0,port=10001,key=username:password,number\_Worker\_threads=1,isConcurrencyHierarchy=Tasks,is\_monitor=false,time\_sleep=0.02,monitor\_dir=C:/Criss/Intermediary/,monitor\_file=C:/Criss/Intermediary/intermediary\_write\_C.txt,output\_dir=C:/Criss/Intermediary/,output\_file=C:/Criss/Intermediary/intermediary\_write\_Julia.txt,temp\_cache\_IO\_data\_dir=C:/Criss/temp/ isBlock=true

// Android-Termux-Ubuntu Arm64 系統 bash 控制臺命令列編譯指令：

// root@localhost:~# /bin/gcc /home/Criss/c/c2exe.c -o /home/Criss/c2exe.exe

// Android-Termux-Ubuntu Arm64 系統 bash 控制臺命令列運行指令;

// root@localhost:~# /home/Criss/c2exe.exe configFile=/home/Criss/config.txt executableFile=/home/Criss/Julia/Julia-1.9.3/bin/julia.exe interpreterFile=-p,4,--project=/home/Criss/jl/ scriptFile=/home/Criss/jl/application.jl configInstructions=interface\_Function=file\_Monitor,webPath=/home/Criss/html/,host=::0,port=10001,key=username:password,number\_Worker\_threads=1,isConcurrencyHierarchy=Tasks,is\_monitor=false,time\_sleep=0.02,monitor\_dir=/home/Criss/Intermediary/,monitor\_file=/home/Criss/Intermediary/intermediary\_write\_C.txt,output\_dir=/home/Criss/Intermediary/,output\_file=/home/Criss/Intermediary/intermediary\_write\_Julia.txt,temp\_cache\_IO\_data\_dir=/home/Criss/temp/ isBlock=true

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <signal.h>

#include <libgen.h>

#include <sys/types.h>

#include <sys/time.h>

#include <sys/stat.h>

char \*NoteVersion = "C call executable(.exe) v20161211\ngcc (x86\_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0\nWindows10 x86\_64 Inter(R)-Core(TM)-m3-6Y30\nC startup julia server\nJulia server\n283640621@qq.com\n+8618604537694\n弘毅\n歲在丙申\n";

char \*NoteHelp = "C call executable(.exe) v20161211\ngcc (x86\_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0\nWindows10 x86\_64 Inter(R)-Core(TM)-m3-6Y30\nC startup julia server\nJulia server\n283640621@qq.com\n+8618604537694\n弘毅\n歲在丙申\n\n--help -h ? --version -v\n\nconfigFile=C:/Criss/config.txt\n\nisBlock=true\n\nexecutableFile=C:/Criss/Julia/Julia-1.9.3/bin/julia.exe\n\ninterpreterFile=-p,4,--project=C:/Criss/jl/\n\nscriptFile=C:/Criss/jl/application.jl\n\nconfigInstructions=interface\_Function=file\_Monitor,webPath=C:/Criss/html/,host=::0,port=10001,key=username:password,number\_Worker\_threads=1,isConcurrencyHierarchy=Tasks,is\_monitor=false,time\_sleep=0.02,monitor\_dir=C:/Criss/Intermediary/,monitor\_file=C:/Criss/Intermediary/intermediary\_write\_C.txt,output\_dir=C:/Criss/Intermediary/,output\_file=C:/Criss/Intermediary/intermediary\_write\_Julia.txt,temp\_cache\_IO\_data\_dir=C:/Criss/temp/,isMonitorThreadsOrProcesses=0,isDoTasksOrThreads=Tasks,to\_executable=C:/Criss/c2exe.exe,to\_script=configFile=C:/Criss/config.txt\n\nexecutableFile=C:/Criss/py/Scripts/python.exe\n\nexecutableFile=C:/Criss/Python/Python311/python.exe\n\nscriptFile=C:/Criss/py/application.py\n\nconfigInstructions=interface\_Function=file\_Monitor,webPath=C:/Criss/html/,host=::0,port=10001,key=username:password,number\_Worker\_threads=1,isConcurrencyHierarchy=Tasks,is\_monitor=False,time\_sleep=0.02,monitor\_dir=C:/Criss/Intermediary/,monitor\_file=C:/Criss/Intermediary/intermediary\_write\_C.txt,output\_dir=C:/Criss/Intermediary/,output\_file=C:/Criss/Intermediary/intermediary\_write\_Python.txt,temp\_cache\_IO\_data\_dir=C:/Criss/temp/,is\_Monitor\_Concurrent=Multi-Threading,to\_executable=C:/Criss/c2exe.exe,to\_script=configFile=C:/Criss/config.txt\n\nexecutableFile=C:/Criss/Nodejs/node.exe\n\nscriptFile=C:/Criss/js/application.js\n\nconfigInstructions=interface\_Function=file\_Monitor,webPath=C:/Criss/html/,host=::0,port=10001,Key=username:password,number\_cluster\_Workers=0,is\_monitor=FALSE,Sys\_sleep=0.2,monitor\_dir=C:/Criss/Intermediary/,monitor\_file=C:/Criss/Intermediary/intermediary\_write\_C.txt,output\_dir=C:/Criss/Intermediary/,output\_file=C:/Criss/Intermediary/intermediary\_write\_Nodejs.txt,temp\_cache\_IO\_data\_dir=C:/Criss/temp/,number\_Worker\_process=0,to\_executable=C:/Criss/c2exe.exe,to\_script=configFile=C:/Criss/config.txt\n";

#define BUFFER\_LEN 1024 // 定義緩衝區最多不得超過 1024 個字符;

// 控制臺傳參，直接使用 gcc 的：main() 函數參數，獲取從控制臺傳入的參數，注意 C 語言的 main() 函數的參數是兩個，第一個是參數的數量（參數數組的長度），第二個是參數的數組;

int main(int argc, char \*argv[]) {

// 判斷控制臺命令行是否有傳入參數，若有傳入參數，則繼續判斷是否爲：查詢信息指令;

if (argc > 1) {

// 若控制臺命令列傳入參數爲："--version" 或 "-v" 時，則在控制臺命令列輸出版本信息;

// 使用函數：strncmp(argv[1], "--version", sizeof(argv[1])) == 0 判斷兩個字符串是否相等;

if ((strncmp(argv[1], "--version", sizeof(argv[1])) == 0) || (strncmp(argv[1], "-v", sizeof(argv[1])) == 0)) {

printf(NoteVersion);

return 0; // 跳出函數，中止運行後續的代碼;

}

// 若控制臺命令列傳入參數爲："--help" 或 "-h" 或 "?" 時，則在控制臺命令列輸出版本信息;

if ((strncmp(argv[1], "--help", sizeof(argv[1])) == 0) || (strncmp(argv[1], "-h", sizeof(argv[1])) == 0) || (strncmp(argv[1], "?", sizeof(argv[1])) == 0)) {

printf(NoteHelp);

return 0; // 跳出函數，中止運行後續的代碼;

}

}

char \*configFile = ""; // 配置文檔的保存路徑參數："C:/Criss/config.txt"

char \*isBlock = ""; // 子進程中的，控制臺命令行程式，運行時，是否阻塞主進程後續代碼的執行;

// isBlock = "true"; // 初始預設值爲："true"，即：阻塞主進程後續代碼的執行;

// printf("isBlock = %s\n", isBlock);

char \*interpreterFile = ""; // 解釋器（Julia、Python、Node.js、Java）的二進制可執行檔啓動時的，配置參數："-p,4,--project=C:/Criss/jl/"

// interpreterFile = ""; // "-p,4,--project=C:/Criss/jl/"; // 初始預設值;

// printf("interpreterFile = %s\n", interpreterFile);

char \*configInstructions = "";// 解釋器（Julia、Python、Node.js、Java）的二進制可執行檔啓動時，讀入的脚步文檔（.jl、.py、.js、.class）的，配置參數："interface\_Function=http\_Server,webPath=C:/Criss/html/,host=::0,port=10001,key=username:password,number\_Worker\_threads=1,isConcurrencyHierarchy=Tasks"

// configInstructions = "interface\_Function=http\_Server,webPath=C:/Criss/html/,host=::0,port=10001,key=username:password,number\_Worker\_threads=1,isConcurrencyHierarchy=Tasks"; // 初始預設值;

// printf("configInstructions = %s\n", configInstructions);

char \*executableFile = ""; // 解釋器（Julia、Python、Node.js、Java）的二進制可執行檔啓動路徑參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"、"C:/Criss/Python/Python311/python.exe"、"C:/Criss/py/Scripts/python.exe"、"C:/Criss/Nodejs/node.exe"

char \*scriptFile = ""; // 解釋器（Julia、Python、Node.js、Java）的二進制可執行檔啓動時，讀入的脚步文檔（.jl、.py、.js、.class）的保存路徑參數："C:/Criss/jl/application.jl"、"C:/Criss/py/application.py"、"C:/Criss/js/application.js"

char \*shellCodeScript = ""; // C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/Python/Python311/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/py/Scripts/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// 1.1、解析預設的配置文檔（C:/Criss/config.txt）的保存路徑參數：configFile;

// 其中：argv[0] 的返回值爲二進制可執行檔完整路徑名稱「/main.exe」;

// printf("The full path to the executable is :\n%s\n", argv[0]);

// printf("當前運行的二進制可執行檔完整路徑爲：\n%s\n", argv[0]); // "C:/Criss/c2exe.exe"

// const int Argument0Length = strlen(argv[0]); // 獲取傳入的第一個參數：argv[0]，即：當前運行的二進制可執行檔完整路徑，的字符串的長度;

// printf("當前運行的二進制可執行檔完整路徑:\n%s\n長度爲：%d 個字符.\n", argv[0], Argument0Length); // 24

// 備份控制臺傳入參數;

// char copyArgv0[Argument0Length + 1];

// strncpy(copyArgv0, argv[0], sizeof(copyArgv0) - 1); // 字符串數組傳值，淺拷貝;

// copyArgv0[sizeof(copyArgv0) - 1] = '\0';

// 備份控制臺傳入參數;

char \*copyArgv0 = strdup(argv[0]); // 函數：strdup(argv[0]) 表示，指針傳值，深拷貝，需要 free(copyArgv0) 釋放内存;

// printf("%s\n", argv[0]); // "C:/Criss/c2exe.exe"

// printf("%s\n", copyArgv0); // "C:/Criss/c2exe.exe"

// const int lengthCopyArgv0 = strlen(copyArgv0);

// printf("%d\n", lengthCopyArgv0); // 42

char \*defaultConfigFileDirectory = ""; // "C:/Criss"

defaultConfigFileDirectory = dirname(copyArgv0);

// printf("預設配置文檔的保存位置爲：\n%s\n", defaultConfigFileDirectory); // "C:/Criss"

const int lengthDefaultConfigFileDirectory = strlen(defaultConfigFileDirectory);

// printf("%d\n", lengthDefaultConfigFileDirectory); // 20

char \*configFileName = "config.txt"; // "C:/Criss/config.txt"

const int lengthConfigFileName = strlen(configFileName);

// printf("%d\n", lengthConfigFileName); // 10

char defaultConfigFilePath[lengthDefaultConfigFileDirectory + 1 + lengthConfigFileName + 1]; // "C:/Criss/config.txt"

snprintf(defaultConfigFilePath, sizeof(defaultConfigFilePath), "%s%s%s", defaultConfigFileDirectory, "/", configFileName);

// printf("預設的配置文檔（config.txt）的保存路徑爲：\n%s\n", defaultConfigFilePath); // "C:/Criss/config.txt"

const int lengthDefaultConfigFilePath = strlen(defaultConfigFilePath);

// printf("%d\n", lengthDefaultConfigFilePath); // 31

defaultConfigFilePath[lengthDefaultConfigFilePath] = '\0';

// printf("預設的配置文檔（config.txt）的保存路徑爲：\n%s\n", defaultConfigFilePath); // "C:/Criss/config.txt"

// const int lengthDefaultConfigFilePath = strlen(defaultConfigFilePath);

// printf("%d\n", lengthDefaultConfigFilePath); // 31

configFile = strdup(defaultConfigFilePath); // 函數：strdup(argv[i]) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// configFile = defaultConfigFilePath;

// printf("預設的配置文檔（config.txt）的保存路徑爲：\n%s\n", configFile); // "C:/Criss/config.txt"

// const int lengthDefaultConfigFilePath = strlen(configFile);

// printf("%d\n", lengthDefaultConfigFilePath); // 31

// // 1.2、解析預設的配置參數值：executableFile、scriptFile、interpreterFile;

// char \*executableFileName = "Julia/Julia-1.9.3/bin/julia.exe"; // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

// const int lengthEexecutableFileName = strlen(executableFileName);

// // printf("%d\n", lengthEexecutableFileName); // 31

// char defaultExecutableFilePath[lengthDefaultConfigFileDirectory + 1 + lengthEexecutableFileName + 1]; // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

// snprintf(defaultExecutableFilePath, sizeof(defaultExecutableFilePath), "%s%s%s", defaultConfigFileDirectory, "/", executableFileName);

// // printf("預設的解釋器二進制可執行檔的啓動路徑爲：\n%s\n", defaultExecutableFilePath); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

// const int lengthDefaultExecutableFilePath = strlen(defaultExecutableFilePath);

// // printf("%d\n", lengthDefaultExecutableFilePath); // 52

// defaultExecutableFilePath[lengthDefaultExecutableFilePath] = '\0';

// // printf("預設的解釋器二進制可執行檔的啓動路徑爲：\n%s\n", defaultExecutableFilePath); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

// // const int lengthDefaultExecutableFilePath = strlen(defaultExecutableFilePath);

// // printf("%d\n", lengthDefaultExecutableFilePath); // 52

// executableFile = strdup(defaultExecutableFilePath); // 函數：strdup(argv[i]) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// // executableFile = defaultExecutableFilePath;

// // printf("預設的解釋器二進制可執行檔（julia.exe）的啓動路徑爲：\n%s\n", executableFile); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

// // const int lengthDefaultExecutableFilePath = strlen(executableFile);

// // printf("%d\n", lengthDefaultExecutableFilePath); // 52

// char \*scriptFileName = "jl/application.jl"; // "C:/Criss/jl/application.jl"

// const int lengthScriptFileName = strlen(scriptFileName);

// // printf("%d\n", lengthScriptFileName); // 56

// char defaultScriptFilePath[lengthDefaultConfigFileDirectory + 1 + lengthScriptFileName + 1]; // "C:/Criss/jl/application.jl"

// snprintf(defaultScriptFilePath, sizeof(defaultScriptFilePath), "%s%s%s", defaultConfigFileDirectory, "/", scriptFileName);

// // printf("預設的解釋器執行的脚本文檔的保存路徑爲：\n%s\n", defaultScriptFilePath); // "C:/Criss/jl/application.jl"

// const int lengthDefaultScriptFilePath = strlen(defaultScriptFilePath);

// // printf("%d\n", lengthDefaultScriptFilePath); // 77

// defaultScriptFilePath[lengthDefaultScriptFilePath] = '\0';

// // printf("預設的解釋器執行的脚本文檔的保存路徑爲：\n%s\n", defaultScriptFilePath); // "C:/Criss/jl/application.jl"

// // const int lengthDefaultScriptFilePath = strlen(defaultScriptFilePath);

// // printf("%d\n", lengthDefaultScriptFilePath); // 77

// scriptFile = strdup(defaultScriptFilePath); // 函數：strdup(argv[i]) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// // scriptFile = defaultScriptFilePath;

// // printf("預設的解釋器執行的脚本文檔（application.jl）的保存路徑爲：\n%s\n", scriptFile); // "C:/Criss/jl/application.jl"

// // const int lengthDefaultScriptFilePath = strlen(scriptFile);

// // printf("%d\n", lengthDefaultScriptFilePath); // 77

// char \*interpreterFileName = "jl/"; // "-p,4,--project=C:/Criss/jl/"

// const int lengthInterpreterFileName = strlen(interpreterFileName);

// // printf("%d\n", lengthInterpreterFileName); // 23

// char defaultInterpreterFilePath[10 + lengthDefaultConfigFileDirectory + 1 + lengthInterpreterFileName + 1]; // "--project=C:/Criss/jl/"

// snprintf(defaultInterpreterFilePath, sizeof(defaultInterpreterFilePath), "%s%s%s%s", "--project=", defaultConfigFileDirectory, "/", interpreterFileName);

// // printf("預設的解釋器啓動時傳入的參數爲：\n%s\n", defaultInterpreterFilePath); // "--project=C:/Criss/jl/"

// const int lengthDefaultInterpreterFilePath = strlen(defaultInterpreterFilePath);

// // printf("%d\n", lengthDefaultInterpreterFilePath); // 54

// defaultInterpreterFilePath[lengthDefaultInterpreterFilePath] = '\0';

// // printf("預設的解釋器啓動時傳入的參數爲：\n%s\n", defaultInterpreterFilePath); // "--project=C:/Criss/jl/"

// // const int lengthDefaultInterpreterFilePath = strlen(defaultInterpreterFilePath);

// // printf("%d\n", lengthDefaultInterpreterFilePath); // 54

// interpreterFile = strdup(defaultInterpreterFilePath); // 函數：strdup(argv[i]) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// // interpreterFile = defaultInterpreterFilePath;

// // printf("預設的解釋器啓動時傳入的參數爲：\n%s\n", interpreterFile); // "--project=C:/Criss/jl/"

// // const int lengthDefaultInterpreterFilePath = strlen(interpreterFile);

// // printf("%d\n", lengthDefaultInterpreterFilePath); // 54

free(copyArgv0);

defaultConfigFileDirectory = NULL;

configFileName = NULL;

// executableFileName = NULL;

// scriptFileName = NULL;

// interpreterFileName = NULL;

// 2、讀取控制臺傳入的配置文檔（C:/Criss/config.txt）的保存路徑參數：configFile;

// 其中：argv[0] 的返回值爲二進制可執行檔完整路徑名稱「/main.exe」;

// printf("The full path to the executable is :\n%s\n", argv[0]);

// printf("Input %d arguments from the console.\n", argc);

// printf("當前運行的二進制可執行檔完整路徑爲：\n%s\n", argv[0]);

// printf("控制臺共傳入 %d 個參數.\n", argc);

// 獲取控制臺傳入的配置文檔：configFile 的完整路徑;

const char \*Delimiter = "=";

for(int i = 1; i < argc; i++) {

// printf("參數 %d: %s\n", i, argv[i]);

// 獲取傳入參數字符串的長度;

// const int ArgumentLength = strlen(argv[i]);

// const int ArgumentLength = 0;

// while (argv[i][ArgumentLength] != '\0') {

// ArgumentLength = ArgumentLength + 1;

// }

// printf("控制臺傳入參數 %s: %s 的長度爲：%d 個字符.\n", i, argv[i], ArgumentLength);

// 備份控制臺傳入參數;

// char copyArgvI[ArgumentLength + 1];

// strncpy(copyArgvI, argv[i], sizeof(copyArgvI)); // 字符串數組傳值，淺拷貝;

// copyArgvI[sizeof(copyArgvI)] = '\0';

// 備份控制臺傳入參數;

char \*copyArgvI = strdup(argv[i]); // 函數：strdup(argv[i]) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// printf("%s\n", argv[i]);

// printf("%s\n", copyArgvI);

const int ArgumentLength = strlen(copyArgvI);

// printf("%d\n", ArgumentLength);

// // 使用：strstr() 函數檢查字符串：haystack 中是否包含字符串：needle，若包含返回第一次出現的地址，若不好含則返回 NULL 值;

// const char \*haystack = "Hello World";

// const char \*needle = "World";

// char \*result = strstr(haystack, needle);

// if (result != NULL) {

// printf("字符串：'%s' 中包含子字符串：'%s' , 位置在：'%s'\n", haystack, needle, result);

// } else {

// printf("字符串：'%s' 中不包含子字符串：'%s'\n", haystack, needle);

// }

// // 若控制臺命令列傳入參數爲："--version" 或 "-v" 時，則在控制臺命令列輸出版本信息;

// // 使用函數：strncmp(copyArgvI, "--version", sizeof(copyArgvI)) == 0 判斷兩個字符串是否相等;

// if ((strncmp(copyArgvI, "--version", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "-v", sizeof(copyArgvI)) == 0)) {

// printf(NoteVersion);

// }

// // 若控制臺命令列傳入參數爲："--help" 或 "-h" 或 "?" 時，則在控制臺命令列輸出版本信息;

// if ((strncmp(copyArgvI, "--help", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "-h", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "?", sizeof(copyArgvI)) == 0)) {

// printf(NoteHelp);

// }

// 若控制臺命令列傳入參數非："--version" 或 "-v" 或 "--help" 或 "-h" 或 "?" 時，則執行如下解析字符串參數;

if (!((strncmp(copyArgvI, "--version", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "-v", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "--help", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "-h", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "?", sizeof(copyArgvI)) == 0))) {

char \*ArgumentKey = "";

char \*ArgumentValue = "";

char \*saveArgumentValue = ""; // 保存函數：ArgumentKey = strtok\_r(copyArgvI, Delimiter, &saveArgumentValue) 分割等號字符（=）之後的第 2 個子字符串，即：ArgumentValue，分割等號字符（=）之後的第 1 個子字符串爲：ArgumentKey;

// const char \*Delimiter = "=";

char \*DelimiterIndex = "";

// 使用：strstr() 函數檢查從控制臺傳入的參數（arguments）數組：argv 的元素中是否包含 "=" 字符，若包含 "=" 字符，返回第一次出現的位置，若不含 "=" 字符，則返回 NULL 值;

// DelimiterIndex = strstr(argv[i], Delimiter);

DelimiterIndex = strstr(copyArgvI, Delimiter);

// printf("參數 %d: %s 中含有的第一個等號字符(%s)的位置在: %s\n", i, copyArgvI, Delimiter, DelimiterIndex);

// printf(Delimiter);

// printf(argv[i]);

// printf(DelimiterIndex);

if (DelimiterIndex != NULL) {

// printf("控制臺傳入參數字符串：'%s' 中包含子字符串（分隔符）：'%s' , 位置在：'%s'\n", argv[i], Delimiter, DelimiterIndex);

// 使用 strtok\_r() 函數分割字符串;

// ArgumentKey = strtok\_r(argv[i], Delimiter, &saveArgumentValue); // 獲取使用字符 "=" 分割後的第一個子字符串，參數名稱;

ArgumentKey = strtok\_r(copyArgvI, Delimiter, &saveArgumentValue); // 獲取使用字符 "=" 分割後的第一個子字符串，參數名稱;

// // 持續向後使用字符 "=" 分割字符串，並持續提取分割後的第一個子字符串;

// while(ArgumentKey != NULL) {

// printf("Input argument: '%s' name: '%s'\n", argv[i], ArgumentKey);

// printf("Input argument: '%s' name: '%s'\n", argv[i], ArgumentKey);

// // 繼續獲取後續被字符 "=" 分割的第一個子字符串，之後對 strtok\_r() 函數的循環調用，第一個參數應使用：NULL 值，表示函數應繼續在之前找到的位置繼續向後查找;

// ArgumentKey = strtok\_r(NULL, Delimiter);

// }

// printf("控制臺傳入參數 %d: 字符串：'%s' 的名稱爲：'%s'\n", i, argv[i], ArgumentKey);

const int ArgumentKeyLength = strlen(ArgumentKey);

// printf("%d\n", ArgumentKeyLength);

ArgumentValue = saveArgumentValue;

// printf("控制臺傳入參數 %d: 字符串：'%s' 的值爲：'%s'\n", i, argv[i], ArgumentValue);

const int ArgumentValueLength = strlen(ArgumentValue);

// printf("%d\n", ArgumentValueLength);

// // 函數：snprintf(buffer, sizeof(buffer), "%s%s%s", ArgumentKey, "=", saveArgumentValue) 表示拼接字符串：ArgumentKey, "=", saveArgumentValue 保存至字符串數組（char buffer[strlen(ArgumentKey) + strlen("=") + strlen(saveArgumentValue) + 1]）：buffer 字符串數組；注意，需要變量：buffer 事先聲明有足夠的長度儲存拼接之後的新字符串，長度需要大於三個字符串長度之和：strlen(ArgumentKey) + strlen("=") + strlen(saveArgumentValue) + 1，用以保存最後一位字符串結束標志：'\0'，否則會内存溢出;

// char buffer[strlen(ArgumentKey) + strlen("=") + strlen(saveArgumentValue) + 1];

// snprintf(buffer, sizeof(buffer), "%s%s%s", ArgumentKey, "=", saveArgumentValue);

// printf("參數 %d: %s\n", i, buffer);

// 判斷是否已經獲取成功使用字符 "=" 分割後的第一個子字符串，參數名稱;

if (ArgumentKey != NULL) {

// 使用函數：strncmp(ArgumentKey, "configFile", sizeof(ArgumentKey)) == 0 判斷兩個字符串是否相等;

if (strncmp(ArgumentKey, "configFile", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// const int ArgumentValueLength = strlen(ArgumentValue);

// // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// for(int g = 0; g < ArgumentValueLength; g++) {

// configFile[g] = ArgumentValue[g];

// }

// configFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// char configFile[ArgumentValueLength + 1];

// strncpy(configFile, ArgumentValue, sizeof(configFile) - 1); // 字符串數組傳值，淺拷貝;

// configFile[sizeof(configFile) - 1] = '\0';

configFile = strdup(ArgumentValue); // 函數：strdup(ArgumentValue) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// printf("%s\n", configFile);

}

// if (strncmp(ArgumentKey, "executableFile", sizeof(ArgumentKey)) == 0) {

// // printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// // const int ArgumentValueLength = strlen(ArgumentValue);

// // // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// // for(int g = 0; g < ArgumentValueLength; g++) {

// // executableFile[g] = ArgumentValue[g];

// // }

// // executableFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// // char executableFile[ArgumentValueLength + 1];

// // strncpy(executableFile, ArgumentValue, sizeof(executableFile) - 1); // 字符串數組傳值，淺拷貝;

// // executableFile[sizeof(executableFile) - 1] = '\0';

// executableFile = strdup(ArgumentValue);

// // printf("%s\n", executableFile);

// }

// if (strncmp(ArgumentKey, "interpreterFile", sizeof(ArgumentKey)) == 0) {

// // printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// interpreterFile = strdup(ArgumentValue);

// // printf("%s\n", interpreterFile);

// // const int ArgumentValueLength = strlen(ArgumentValue);

// // const int ArgumentValueLength = strlen(interpreterFile);

// // 使用：for(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

// for (int j = 0; j < ArgumentValueLength; j++) {

// // printf("%c\n", \*interpreterFile);

// // 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

// if (\*interpreterFile == ',') {

// \*interpreterFile = ' ';

// }

// interpreterFile++; // 每輪循環後，自動向後偏移一位指向;

// }

// // 將字符串指針，重新指向字符串的首地址;

// interpreterFile = interpreterFile - ArgumentValueLength;

// // printf("%s\n", interpreterFile);

// // // 使用：while(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

// // while (\*interpreterFile != '\0') {

// // // printf("%c\n", \*interpreterFile);

// // // 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

// // if (\*interpreterFile == ',') {

// // \*interpreterFile = ' ';

// // }

// // interpreterFile++; // 每輪循環後，自動向後偏移一位指向;

// // }

// // // 將字符串指針，重新指向字符串的首地址;

// // interpreterFile = interpreterFile - ArgumentValueLength;

// // printf("%s\n", interpreterFile);

// }

// if (strncmp(ArgumentKey, "scriptFile", sizeof(ArgumentKey)) == 0) {

// // printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// scriptFile = strdup(ArgumentValue);

// // printf("%s\n", scriptFile);

// }

// if (strncmp(ArgumentKey, "configInstructions", sizeof(ArgumentKey)) == 0) {

// // printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// configInstructions = strdup(ArgumentValue);

// // printf("%s\n", configInstructions);

// // const int ArgumentValueLength = strlen(ArgumentValue);

// // const int ArgumentValueLength = strlen(configInstructions);

// // 使用：for(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

// for (int j = 0; j < ArgumentValueLength; j++) {

// // printf("%c\n", \*configInstructions);

// // 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

// if (\*configInstructions == ',') {

// \*configInstructions = ' ';

// }

// configInstructions++; // 每輪循環後，自動向後偏移一位指向;

// }

// // 將字符串指針，重新指向字符串的首地址;

// configInstructions = configInstructions - ArgumentValueLength;

// // printf("%s\n", configInstructions);

// // // 使用：while(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

// // while (\*configInstructions != '\0') {

// // // printf("%c\n", \*configInstructions);

// // // 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

// // if (\*configInstructions == ',') {

// // \*configInstructions = ' ';

// // }

// // configInstructions++; // 每輪循環後，自動向後偏移一位指向;

// // }

// // // 將字符串指針，重新指向字符串的首地址;

// // configInstructions = configInstructions - ArgumentValueLength;

// // printf("%s\n", configInstructions);

// }

// if (strncmp(ArgumentKey, "isBlock", sizeof(ArgumentKey)) == 0) {

// // printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// isBlock = strdup(ArgumentValue);

// // printf("%s\n", isBlock);

// }

} else {

// printf("控制臺傳入參數字符串：'%s' 中不包含子字符串（分隔符）：'%s'\n", argv[i], Delimiter);

configFile = strdup(ArgumentValue);

// const int ArgumentValueLength = strlen(ArgumentValue);

// // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// for(int g = 0; g < ArgumentValueLength; g++) {

// configFile[g] = ArgumentValue[g];

// }

// configFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// char configFile[ArgumentValueLength + 1];

// strncpy(configFile, ArgumentValue, sizeof(configFile) - 1); // 字符串數組傳值，淺拷貝;

// configFile[sizeof(configFile) - 1] = '\0';

}

} else {

// printf("控制臺傳入參數字符串：'%s' 中不包含子字符串（分隔符）：'%s'\n", argv[i], Delimiter);

configFile = strdup(ArgumentValue);

// const int ArgumentValueLength = strlen(ArgumentValue);

// // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// for(int g = 0; g < ArgumentValueLength; g++) {

// configFile[g] = ArgumentValue[g];

// }

// configFile[ArgumentValueLength] = '\0'; // 字符串末尾添加結束標記;

// char configFile[ArgumentValueLength + 1];

// strncpy(configFile, ArgumentValue, sizeof(configFile)); // 字符串數組傳值，淺拷貝;

// configFile[sizeof(configFile)] = '\0';

}

}

free(copyArgvI);

}

// printf("控制臺傳入的配置文檔（config.txt）完整路徑爲：\n%s\n", configFile);

// 3、讀取配置文檔（C:/Criss/config.txt）中記錄的參數：executableFile, interpreterFile, scriptFile, configInstructions, isBlock;

// printf("配置文檔（config.txt）的保存路徑爲：\n%s\n", configFile); // "C:/Criss/config.txt";

// 讀取配置文檔：configFile 中的參數;

if (strlen(configFile) > 0) {

FILE \*file = fopen(configFile, "rt"); // rt、rb、wt、wb、a、r+、w+、a+;

if (file == NULL) {

// printf("無法打開配置文檔：\nconfigFile = %s\n", configFile);

// // return 1;

}

if (file != NULL) {

// printf("正在使用配置文檔：\nconfigFile = %s\n", configFile); // "C:/Criss/config.txt";

printf("configFile = %s\n", configFile); // "C:/Criss/config.txt";

// // 使用：fread(buffer, size, 1, file) 函數，一次讀入文檔中的全部内容，包含每個橫向列（row）末尾的換行符回車符號（Enter）：'\n';

// fseek(file, 0, SEEK\_END); // 定位文檔指針到文檔末尾;

// long size = ftell(file); // 計算文檔所包含的字符個數長度;

// fseek(file, 0, SEEK\_SET); // 將文檔指針重新移動到文檔的開頭;

// char \*buffer = (char \*)malloc(size + 1); // 動態内存分配，按照上一部識別的文檔大小，申請内存緩衝區存儲文檔内容;

// fread(buffer, size, 1, file); // 讀入文檔中的全部内容到内存緩衝區（buffer）;

// buffer[size] = '\0'; // 在内存緩衝區（buffer）儲存的文檔内容的末尾添加字符串結束字符（'\0'）;

// printf("%s\n", buffer);

// fclose(file); // 關閉文檔;

// free(buffer); // 釋放内存緩衝區（buffer）;

// // 使用：Character = fgetc(file) 函數，逐字符讀入文檔中的内容，包含每個橫向列（row）末尾的換行符回車符號（Enter）：'\n';

// int Character;

// int flag;

// flag = 1;

// while (flag) {

// // 逐字符讀入文檔中的内容;

// Character = fgetc(file); // 從文檔中讀取一個字符;

// // EOF == -1，判斷指針是否已經後移到文檔末尾;

// if (c == EOF) {

// flag = 0;

// break; // 跳出 while(){} 循環;

// }

// printf("%c", Character);

// }

// fclose(file); // 關閉文檔;

// 使用：fgets(buffer, sizeof(buffer), file) 函數，逐個橫向列（row）讀入文檔中的内容，包含每個橫向列（row）末尾的換行符回車符號（Enter）：'\n';

char buffer[BUFFER\_LEN]; // BUFFER\_LEN = 1024 // 代碼首部自定義的常量：BUFFER\_LEN = 1024，靜態申請内存緩衝區（buffer），存儲文檔每一個橫向列（row）中的内容，要求配置文檔：config.txt 中每一個橫向列（row）最多不得超過 1024 個字符;

while (fgets(buffer, sizeof(buffer), file) != NULL) {

// printf("%s\n", buffer);

// 獲取傳入參數字符串的長度;

// const int ArgumentLength = strlen(buffer);

// const int ArgumentLength = 0;

// while (buffer[ArgumentLength] != '\0') {

// ArgumentLength = ArgumentLength + 1;

// }

// printf("控制臺傳入參數 %s: %s 的長度爲：%d 個字符.\n", i, buffer, ArgumentLength);

// 備份控制臺傳入參數;

// char copyArgvI[ArgumentLength + 1];

// strncpy(copyArgvI, buffer, sizeof(copyArgvI) - 1); // 字符串數組傳值，淺拷貝;

// copyArgvI[sizeof(copyArgvI) - 1] = '\0';

// 備份控制臺傳入參數;

char \*copyArgvI = buffer;

// copyArgvI = strdup(buffer); // 函數：strdup(buffer) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// printf("%s\n", buffer);

// printf("%s\n", copyArgvI);

const int ArgumentLength = strlen(copyArgvI);

// printf("%d\n", ArgumentLength);

// if (ArgumentLength > 0 && \*copyArgvI != '\n') {

if (ArgumentLength > 0) {

// // 使用：for(){} 循環，遍歷字符串，查找注釋符號井號字符（'#'），並將之替換爲字符串結束標志符號（'\0'），即從注釋符號井號（#）處截斷字符串;

// printf("%s\n", copyArgvI);

// const int ArgumentLength = strlen(copyArgvI);

// printf("%d\n", strlen(copyArgvI));

// for (int i = 0; i < ArgumentLength; i++) {

// printf("%c\n", \*copyArgvI);

// // 判斷當前指向的字符，是否爲注釋符號井號字符（'#'），若是井號字符（'#'），則將之替換爲字符串結束標志符號（'\0'），若非井號字符（'#'）則保持不變;

// if (\*copyArgvI == '#') {

// \*copyArgvI = '\0';

// }

// copyArgvI++; // 每輪循環後，自動向後偏移一位指向;

// }

// // 將字符串指針，重新指向字符串的首地址;

// copyArgvI = copyArgvI - ArgumentLength;

// printf("%s\n", copyArgvI);

// const int ArgumentLength = strlen(copyArgvI);

// printf("%d\n", strlen(copyArgvI));

// 刪除配置文檔（C:/Criss/config.txt）中，每個橫向列（row）字符串末尾可能存在的換行符回車符號（Enter）：'\n'，代之以字符串結束標志符號：'\0'，字符串長度會縮短 1 位;

// printf("%c ~ %c ~ %c\n", \*copyArgvI, \*(copyArgvI + ArgumentLength - 2), \*(copyArgvI + ArgumentLength)); // \*copyArgvI 爲字符串的首字符，\*(copyArgvI + ArgumentLength - 2) 爲字符串的尾字符，\*(copyArgvI + ArgumentLength - 1) 爲字符串末端的換行符回車符號（Enter）== '\n'，\*(copyArgvI + ArgumentLength) 爲字符串末端的結束標志符號 == '\0' ;

// printf("%c\n", (copyArgvI + ArgumentLength - 1));

if (\*(copyArgvI + ArgumentLength - 1) == '\n') {

\*(copyArgvI + ArgumentLength - 1) = '\0';

} else {

\*(copyArgvI + ArgumentLength) = '\0';

}

// printf("%s\n", copyArgvI);

const int ArgumentLength = strlen(copyArgvI);

// printf("%d\n", ArgumentLength);

// // 使用：strstr() 函數檢查字符串：haystack 中是否包含字符串：needle，若包含返回第一次出現的地址，若不好含則返回 NULL 值;

// const char \*haystack = "Hello World";

// const char \*needle = "World";

// char \*result = strstr(haystack, needle);

// if (result != NULL) {

// printf("字符串：'%s' 中包含子字符串：'%s' , 位置在：'%s'\n", haystack, needle, result);

// } else {

// printf("字符串：'%s' 中不包含子字符串：'%s'\n", haystack, needle);

// }

// 提取配置文檔（C:/Criss/config.txt）中，每個橫向列（row）字符串中，指定的參數值：executableFile, interpreterFile, scriptFile, configInstructions, isBlock;

if (ArgumentLength > 0) {

// printf("%s\n", copyArgvI);

// 判斷當前橫向列（row）字符串的參數，是否被井號（#）注釋掉;

if (\*copyArgvI != '#') {

// printf("%s\n", copyArgvI);

// // 使用：for(){} 循環，遍歷字符串，查找注釋符號井號字符（'#'），並將之替換爲字符串結束標志符號（'\0'），即從注釋符號井號（#）處截斷字符串;

// printf("%s\n", copyArgvI);

// const int ArgumentLength = strlen(copyArgvI);

// printf("%d\n", strlen(copyArgvI));

// for (int i = 0; i < ArgumentLength; i++) {

// printf("%c\n", \*copyArgvI);

// // 判斷當前指向的字符，是否爲注釋符號井號字符（'#'），若是井號字符（'#'），則將之替換爲字符串結束標志符號（'\0'），若非井號字符（'#'）則保持不變;

// if (\*copyArgvI == '#') {

// \*copyArgvI = '\0';

// }

// copyArgvI++; // 每輪循環後，自動向後偏移一位指向;

// }

// // 將字符串指針，重新指向字符串的首地址;

// copyArgvI = copyArgvI - ArgumentLength;

// printf("%s\n", copyArgvI);

// const int ArgumentLength = strlen(copyArgvI);

// printf("%d\n", strlen(copyArgvI));

char \*ArgumentKey = "";

char \*ArgumentValue = "";

char \*saveArgumentValue = ""; // 保存函數：ArgumentKey = strtok\_r(copyArgvI, Delimiter, &saveArgumentValue) 分割等號字符（=）之後的第 2 個子字符串，即：ArgumentValue，分割等號字符（=）之後的第 1 個子字符串爲：ArgumentKey;

// const char \*Delimiter = "=";

char \*DelimiterIndex = "";

// 使用：strstr() 函數檢查從控制臺傳入的參數（arguments）數組：argv 的元素中是否包含 "=" 字符，若包含 "=" 字符，返回第一次出現的位置，若不含 "=" 字符，則返回 NULL 值;

// DelimiterIndex = strstr(argv[i], Delimiter);

DelimiterIndex = strstr(copyArgvI, Delimiter);

// printf("參數 %d: %s 中含有的第一個等號字符(%s)的位置在: %s\n", i, copyArgvI, Delimiter, DelimiterIndex);

// printf(Delimiter);

// printf(argv[i]);

// printf(DelimiterIndex);

if (DelimiterIndex != NULL) {

// printf("控制臺傳入參數字符串：'%s' 中包含子字符串（分隔符）：'%s' , 位置在：'%s'\n", argv[i], Delimiter, DelimiterIndex);

// 使用 strtok\_r() 函數分割字符串;

// ArgumentKey = strtok\_r(argv[i], Delimiter, &saveArgumentValue); // 獲取使用字符 "=" 分割後的第一個子字符串，參數名稱;

ArgumentKey = strtok\_r(copyArgvI, Delimiter, &saveArgumentValue); // 獲取使用字符 "=" 分割後的第一個子字符串，參數名稱;

// // 持續向後使用字符 "=" 分割字符串，並持續提取分割後的第一個子字符串;

// while(ArgumentKey != NULL) {

// printf("Input argument: '%s' name: '%s'\n", argv[i], ArgumentKey);

// printf("Input argument: '%s' name: '%s'\n", argv[i], ArgumentKey);

// // 繼續獲取後續被字符 "=" 分割的第一個子字符串，之後對 strtok\_r() 函數的循環調用，第一個參數應使用：NULL 值，表示函數應繼續在之前找到的位置繼續向後查找;

// ArgumentKey = strtok\_r(NULL, Delimiter);

// }

// printf("控制臺傳入參數字符串的名稱爲：%s\n", ArgumentKey);

const int ArgumentKeyLength = strlen(ArgumentKey);

// printf("%d\n", ArgumentKeyLength);

ArgumentValue = saveArgumentValue;

// printf("控制臺傳入參數字符串的值爲：%s\n", ArgumentValue);

const int ArgumentValueLength = strlen(ArgumentValue);

// printf("%d\n", ArgumentValueLength);

// // 函數：snprintf(buffer, sizeof(buffer), "%s%s%s", ArgumentKey, "=", saveArgumentValue) 表示拼接字符串：ArgumentKey, "=", saveArgumentValue 保存至字符串數組（char buffer[strlen(ArgumentKey) + strlen("=") + strlen(saveArgumentValue) + 1]）：buffer 字符串數組；注意，需要變量：buffer 事先聲明有足夠的長度儲存拼接之後的新字符串，長度需要大於三個字符串長度之和：strlen(ArgumentKey) + strlen("=") + strlen(saveArgumentValue) + 1，用以保存最後一位字符串結束標志：'\0'，否則會内存溢出;

// char buffer[strlen(ArgumentKey) + strlen("=") + strlen(saveArgumentValue) + 1];

// snprintf(buffer, sizeof(buffer), "%s%s%s", ArgumentKey, "=", saveArgumentValue);

// printf("參數 %d: %s\n", i, buffer);

// 判斷是否已經獲取成功使用字符 "=" 分割後的第一個子字符串，參數名稱;

if (ArgumentKey != NULL) {

// // 使用函數：strncmp(ArgumentKey, "configFile", sizeof(ArgumentKey)) == 0 判斷兩個字符串是否相等;

// if (strncmp(ArgumentKey, "configFile", sizeof(ArgumentKey)) == 0) {

// // printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// // const int ArgumentValueLength = strlen(ArgumentValue);

// // // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// // for(int g = 0; g < ArgumentValueLength; g++) {

// // configFile[g] = ArgumentValue[g];

// // }

// // configFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// // char configFile[ArgumentValueLength + 1];

// // strncpy(configFile, ArgumentValue, sizeof(configFile) - 1); // 字符串數組傳值，淺拷貝;

// // configFile[sizeof(configFile) - 1] = '\0';

// configFile = strdup(ArgumentValue); // 函數：strdup(ArgumentValue) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// // printf("%s\n", configFile);

// }

if (strncmp(ArgumentKey, "executableFile", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// const int ArgumentValueLength = strlen(ArgumentValue);

// // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// for(int g = 0; g < ArgumentValueLength; g++) {

// executableFile[g] = ArgumentValue[g];

// }

// executableFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// char executableFile[ArgumentValueLength + 1];

// strncpy(executableFile, ArgumentValue, sizeof(executableFile) - 1); // 字符串數組傳值，淺拷貝;

// executableFile[sizeof(executableFile) - 1] = '\0';

executableFile = strdup(ArgumentValue);

// printf("%s\n", executableFile);

}

if (strncmp(ArgumentKey, "interpreterFile", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

interpreterFile = strdup(ArgumentValue);

// printf("%s\n", interpreterFile);

// const int ArgumentValueLength = strlen(ArgumentValue);

// const int ArgumentValueLength = strlen(interpreterFile);

// 使用：for(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

for (int j = 0; j < ArgumentValueLength; j++) {

// printf("%c\n", \*interpreterFile);

// 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

if (\*interpreterFile == ',') {

\*interpreterFile = ' ';

}

interpreterFile++; // 每輪循環後，自動向後偏移一位指向;

}

// 將字符串指針，重新指向字符串的首地址;

interpreterFile = interpreterFile - ArgumentValueLength;

// printf("%s\n", interpreterFile);

// // 使用：while(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

// while (\*interpreterFile != '\0') {

// // printf("%c\n", \*interpreterFile);

// // 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

// if (\*interpreterFile == ',') {

// \*interpreterFile = ' ';

// }

// interpreterFile++; // 每輪循環後，自動向後偏移一位指向;

// }

// // 將字符串指針，重新指向字符串的首地址;

// interpreterFile = interpreterFile - ArgumentValueLength;

// printf("%s\n", interpreterFile);

}

if (strncmp(ArgumentKey, "scriptFile", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

scriptFile = strdup(ArgumentValue);

// printf("%s\n", scriptFile);

}

if (strncmp(ArgumentKey, "configInstructions", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

configInstructions = strdup(ArgumentValue);

// printf("%s\n", configInstructions);

// const int ArgumentValueLength = strlen(ArgumentValue);

// const int ArgumentValueLength = strlen(configInstructions);

// 使用：for(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

for (int j = 0; j < ArgumentValueLength; j++) {

// printf("%c\n", \*configInstructions);

// 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

if (\*configInstructions == ',') {

\*configInstructions = ' ';

}

configInstructions++; // 每輪循環後，自動向後偏移一位指向;

}

// 將字符串指針，重新指向字符串的首地址;

configInstructions = configInstructions - ArgumentValueLength;

// printf("%s\n", configInstructions);

// // 使用：while(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

// while (\*configInstructions != '\0') {

// // printf("%c\n", \*configInstructions);

// // 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

// if (\*configInstructions == ',') {

// \*configInstructions = ' ';

// }

// configInstructions++; // 每輪循環後，自動向後偏移一位指向;

// }

// // 將字符串指針，重新指向字符串的首地址;

// configInstructions = configInstructions - ArgumentValueLength;

// printf("%s\n", configInstructions);

}

if (strncmp(ArgumentKey, "isBlock", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

isBlock = strdup(ArgumentValue);

// printf("%s\n", isBlock);

}

// } else {

// // printf("控制臺傳入參數字符串：'%s' 中不包含子字符串（分隔符）：'%s'\n", argv[i], Delimiter);

// configFile = strdup(ArgumentValue);

// // const int ArgumentValueLength = strlen(ArgumentValue);

// // // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// // for(int g = 0; g < ArgumentValueLength; g++) {

// // configFile[g] = ArgumentValue[g];

// // }

// // configFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// // char configFile[ArgumentValueLength + 1];

// // strncpy(configFile, ArgumentValue, sizeof(configFile) - 1); // 字符串數組傳值，淺拷貝;

// // configFile[sizeof(configFile) - 1] = '\0';

}

// } else {

// // printf("控制臺傳入參數字符串：'%s' 中不包含子字符串（分隔符）：'%s'\n", argv[i], Delimiter);

// configFile = strdup(ArgumentValue);

// // const int ArgumentValueLength = strlen(ArgumentValue);

// // // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// // for(int g = 0; g < ArgumentValueLength; g++) {

// // configFile[g] = ArgumentValue[g];

// // }

// // configFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// // char configFile[ArgumentValueLength + 1];

// // strncpy(configFile, ArgumentValue, sizeof(configFile) - 1); // 字符串數組傳值，淺拷貝;

// // configFile[sizeof(configFile) - 1] = '\0';

}

}

}

}

copyArgvI = NULL; // 清空指針;

memset(buffer, 0, sizeof(buffer)); // 釋放内存緩衝區（buffer）;

}

fclose(file); // 關閉文檔;

}

} else {

// printf("配置文檔的保存路徑參數爲空：\nconfigFile = %s\n", configFile); // 配置文檔的保存路徑參數："C:/Criss/config.txt"

// // printf("configFile = %s\n", configFile); // 配置文檔的保存路徑參數："C:/Criss/config.txt"

// // return 1;

}

// 4、讀取控制臺傳入的其他參數：executableFile, interpreterFile, scriptFile, configInstructions, isBlock;

// 其中：argv[0] 的返回值爲二進制可執行檔完整路徑名稱「/main.exe」;

// printf("The full path to the executable is :\n%s\n", argv[0]);

// printf("Input %d arguments from the console.\n", argc);

// printf("當前運行的二進制可執行檔完整路徑爲：\n%s\n", argv[0]);

// printf("控制臺共傳入 %d 個參數.\n", argc);

// 獲取控制臺傳入的配置文檔：configFile 的完整路徑;

// const char \*Delimiter = "=";

for(int i = 1; i < argc; i++) {

// printf("參數 %d: %s\n", i, argv[i]);

// 獲取傳入參數字符串的長度;

// const int ArgumentLength = strlen(argv[i]);

// const int ArgumentLength = 0;

// while (argv[i][ArgumentLength] != '\0') {

// ArgumentLength = ArgumentLength + 1;

// }

// printf("控制臺傳入參數 %s: %s 的長度爲：%d 個字符.\n", i, argv[i], ArgumentLength);

// 備份控制臺傳入參數;

// char copyArgvI[ArgumentLength + 1];

// strncpy(copyArgvI, argv[i], sizeof(copyArgvI) - 1); // 字符串數組傳值，淺拷貝;

// copyArgvI[sizeof(copyArgvI) - 1] = '\0';

// 備份控制臺傳入參數;

char \*copyArgvI = strdup(argv[i]); // 函數：strdup(argv[i]) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// printf("%s\n", argv[i]);

// printf("%s\n", copyArgvI);

const int ArgumentLength = strlen(copyArgvI);

// printf("%d\n", ArgumentLength);

// // 使用：strstr() 函數檢查字符串：haystack 中是否包含字符串：needle，若包含返回第一次出現的地址，若不好含則返回 NULL 值;

// const char \*haystack = "Hello World";

// const char \*needle = "World";

// char \*result = strstr(haystack, needle);

// if (result != NULL) {

// printf("字符串：'%s' 中包含子字符串：'%s' , 位置在：'%s'\n", haystack, needle, result);

// } else {

// printf("字符串：'%s' 中不包含子字符串：'%s'\n", haystack, needle);

// }

// // 若控制臺命令列傳入參數爲："--version" 或 "-v" 時，則在控制臺命令列輸出版本信息;

// // 使用函數：strncmp(copyArgvI, "--version", sizeof(copyArgvI)) == 0 判斷兩個字符串是否相等;

// if ((strncmp(copyArgvI, "--version", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "-v", sizeof(copyArgvI)) == 0)) {

// printf(NoteVersion);

// }

// // 若控制臺命令列傳入參數爲："--help" 或 "-h" 或 "?" 時，則在控制臺命令列輸出版本信息;

// if ((strncmp(copyArgvI, "--help", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "-h", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "?", sizeof(copyArgvI)) == 0)) {

// printf(NoteHelp);

// }

// 若控制臺命令列傳入參數非："--version" 或 "-v" 或 "--help" 或 "-h" 或 "?" 時，則執行如下解析字符串參數;

if (!((strncmp(copyArgvI, "--version", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "-v", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "--help", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "-h", sizeof(copyArgvI)) == 0) || (strncmp(copyArgvI, "?", sizeof(copyArgvI)) == 0))) {

char \*ArgumentKey = "";

char \*ArgumentValue = "";

char \*saveArgumentValue = ""; // 保存函數：ArgumentKey = strtok\_r(copyArgvI, Delimiter, &saveArgumentValue) 分割等號字符（=）之後的第 2 個子字符串，即：ArgumentValue，分割等號字符（=）之後的第 1 個子字符串爲：ArgumentKey;

// const char \*Delimiter = "=";

char \*DelimiterIndex = "";

// 使用：strstr() 函數檢查從控制臺傳入的參數（arguments）數組：argv 的元素中是否包含 "=" 字符，若包含 "=" 字符，返回第一次出現的位置，若不含 "=" 字符，則返回 NULL 值;

// DelimiterIndex = strstr(argv[i], Delimiter);

DelimiterIndex = strstr(copyArgvI, Delimiter);

// printf("參數 %d: %s 中含有的第一個等號字符(%s)的位置在: %s\n", i, copyArgvI, Delimiter, DelimiterIndex);

// printf(Delimiter);

// printf(argv[i]);

// printf(DelimiterIndex);

if (DelimiterIndex != NULL) {

// printf("控制臺傳入參數字符串：'%s' 中包含子字符串（分隔符）：'%s' , 位置在：'%s'\n", argv[i], Delimiter, DelimiterIndex);

// 使用 strtok\_r() 函數分割字符串;

// ArgumentKey = strtok\_r(argv[i], Delimiter, &saveArgumentValue); // 獲取使用字符 "=" 分割後的第一個子字符串，參數名稱;

ArgumentKey = strtok\_r(copyArgvI, Delimiter, &saveArgumentValue); // 獲取使用字符 "=" 分割後的第一個子字符串，參數名稱;

// // 持續向後使用字符 "=" 分割字符串，並持續提取分割後的第一個子字符串;

// while(ArgumentKey != NULL) {

// printf("Input argument: '%s' name: '%s'\n", argv[i], ArgumentKey);

// printf("Input argument: '%s' name: '%s'\n", argv[i], ArgumentKey);

// // 繼續獲取後續被字符 "=" 分割的第一個子字符串，之後對 strtok\_r() 函數的循環調用，第一個參數應使用：NULL 值，表示函數應繼續在之前找到的位置繼續向後查找;

// ArgumentKey = strtok\_r(NULL, Delimiter);

// }

// printf("控制臺傳入參數 %d: 字符串：'%s' 的名稱爲：'%s'\n", i, argv[i], ArgumentKey);

const int ArgumentKeyLength = strlen(ArgumentKey);

// printf("%d\n", ArgumentKeyLength);

ArgumentValue = saveArgumentValue;

// printf("控制臺傳入參數 %d: 字符串：'%s' 的值爲：'%s'\n", i, argv[i], ArgumentValue);

const int ArgumentValueLength = strlen(ArgumentValue);

// printf("%d\n", ArgumentValueLength);

// // 函數：snprintf(buffer, sizeof(buffer), "%s%s%s", ArgumentKey, "=", saveArgumentValue) 表示拼接字符串：ArgumentKey, "=", saveArgumentValue 保存至字符串數組（char buffer[strlen(ArgumentKey) + strlen("=") + strlen(saveArgumentValue) + 1]）：buffer 字符串數組；注意，需要變量：buffer 事先聲明有足夠的長度儲存拼接之後的新字符串，長度需要大於三個字符串長度之和：strlen(ArgumentKey) + strlen("=") + strlen(saveArgumentValue) + 1，用以保存最後一位字符串結束標志：'\0'，否則會内存溢出;

// char buffer[strlen(ArgumentKey) + strlen("=") + strlen(saveArgumentValue) + 1];

// snprintf(buffer, sizeof(buffer), "%s%s%s", ArgumentKey, "=", saveArgumentValue);

// printf("參數 %d: %s\n", i, buffer);

// 判斷是否已經獲取成功使用字符 "=" 分割後的第一個子字符串，參數名稱;

if (ArgumentKey != NULL) {

// // 使用函數：strncmp(ArgumentKey, "configFile", sizeof(ArgumentKey)) == 0 判斷兩個字符串是否相等;

// if (strncmp(ArgumentKey, "configFile", sizeof(ArgumentKey)) == 0) {

// // printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// // const int ArgumentValueLength = strlen(ArgumentValue);

// // // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// // for(int g = 0; g < ArgumentValueLength; g++) {

// // configFile[g] = ArgumentValue[g];

// // }

// // configFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// // char configFile[ArgumentValueLength + 1];

// // strncpy(configFile, ArgumentValue, sizeof(configFile) - 1); // 字符串數組傳值，淺拷貝;

// // configFile[sizeof(configFile) - 1] = '\0';

// configFile = strdup(ArgumentValue); // 函數：strdup(ArgumentValue) 表示，指針傳值，深拷貝，需要 free(copyArgvI) 釋放内存;

// // printf("%s\n", configFile);

// }

if (strncmp(ArgumentKey, "executableFile", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

// const int ArgumentValueLength = strlen(ArgumentValue);

// // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// for(int g = 0; g < ArgumentValueLength; g++) {

// executableFile[g] = ArgumentValue[g];

// }

// executableFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// char executableFile[ArgumentValueLength + 1];

// strncpy(executableFile, ArgumentValue, sizeof(executableFile) - 1); // 字符串數組傳值，淺拷貝;

// executableFile[sizeof(executableFile) - 1] = '\0';

executableFile = strdup(ArgumentValue);

// printf("%s\n", executableFile);

}

if (strncmp(ArgumentKey, "interpreterFile", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

interpreterFile = strdup(ArgumentValue);

// printf("%s\n", interpreterFile);

// const int ArgumentValueLength = strlen(ArgumentValue);

// const int ArgumentValueLength = strlen(interpreterFile);

// 使用：for(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

for (int j = 0; j < ArgumentValueLength; j++) {

// printf("%c\n", \*interpreterFile);

// 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

if (\*interpreterFile == ',') {

\*interpreterFile = ' ';

}

interpreterFile++; // 每輪循環後，自動向後偏移一位指向;

}

// 將字符串指針，重新指向字符串的首地址;

interpreterFile = interpreterFile - ArgumentValueLength;

// printf("%s\n", interpreterFile);

// // 使用：while(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

// while (\*interpreterFile != '\0') {

// // printf("%c\n", \*interpreterFile);

// // 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

// if (\*interpreterFile == ',') {

// \*interpreterFile = ' ';

// }

// interpreterFile++; // 每輪循環後，自動向後偏移一位指向;

// }

// // 將字符串指針，重新指向字符串的首地址;

// interpreterFile = interpreterFile - ArgumentValueLength;

// printf("%s\n", interpreterFile);

}

if (strncmp(ArgumentKey, "scriptFile", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

scriptFile = strdup(ArgumentValue);

// printf("%s\n", scriptFile);

}

if (strncmp(ArgumentKey, "configInstructions", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

configInstructions = strdup(ArgumentValue);

// printf("%s\n", configInstructions);

// const int ArgumentValueLength = strlen(ArgumentValue);

// const int ArgumentValueLength = strlen(configInstructions);

// 使用：for(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

for (int j = 0; j < ArgumentValueLength; j++) {

// printf("%c\n", \*configInstructions);

// 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

if (\*configInstructions == ',') {

\*configInstructions = ' ';

}

configInstructions++; // 每輪循環後，自動向後偏移一位指向;

}

// 將字符串指針，重新指向字符串的首地址;

configInstructions = configInstructions - ArgumentValueLength;

// printf("%s\n", configInstructions);

// // 使用：while(){} 循環，遍歷字符串，查找逗號字符（','），並將之替換爲空格符號（' '）;

// while (\*configInstructions != '\0') {

// // printf("%c\n", \*configInstructions);

// // 判斷當前指向的字符，是否爲逗號字符（','），若是逗號字符（','），則將之替換爲空格符號（' '），若非逗號字符（','）則保持不變;

// if (\*configInstructions == ',') {

// \*configInstructions = ' ';

// }

// configInstructions++; // 每輪循環後，自動向後偏移一位指向;

// }

// // 將字符串指針，重新指向字符串的首地址;

// configInstructions = configInstructions - ArgumentValueLength;

// printf("%s\n", configInstructions);

}

if (strncmp(ArgumentKey, "isBlock", sizeof(ArgumentKey)) == 0) {

// printf("控制臺傳入參數：'%s' '%s' '%s'\n", ArgumentKey, Delimiter, ArgumentValue);

isBlock = strdup(ArgumentValue);

// printf("%s\n", isBlock);

}

// } else {

// // printf("控制臺傳入參數字符串：'%s' 中不包含子字符串（分隔符）：'%s'\n", argv[i], Delimiter);

// configFile = strdup(ArgumentValue);

// // const int ArgumentValueLength = strlen(ArgumentValue);

// // // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// // for(int g = 0; g < ArgumentValueLength; g++) {

// // configFile[g] = ArgumentValue[g];

// // }

// // configFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// // char configFile[ArgumentValueLength + 1];

// // strncpy(configFile, ArgumentValue, sizeof(configFile) - 1); // 字符串數組傳值，淺拷貝;

// // configFile[sizeof(configFile) - 1] = '\0';

}

// } else {

// // printf("控制臺傳入參數字符串：'%s' 中不包含子字符串（分隔符）：'%s'\n", argv[i], Delimiter);

// configFile = strdup(ArgumentValue);

// // const int ArgumentValueLength = strlen(ArgumentValue);

// // // printf("控制臺傳入參數值(value)的長度爲：%d 個字符.\n", ArgumentValueLength);

// // for(int g = 0; g < ArgumentValueLength; g++) {

// // configFile[g] = ArgumentValue[g];

// // }

// // configFile[ArgumentValueLength + 1] = '\0'; // 字符串末尾添加結束標記;

// // char configFile[ArgumentValueLength + 1];

// // strncpy(configFile, ArgumentValue, sizeof(configFile) - 1); // 字符串數組傳值，淺拷貝;

// // configFile[sizeof(configFile) - 1] = '\0';

}

}

free(copyArgvI);

}

// 5、創建子進程執行調用控制臺 shell 語句運行第三方可執行檔;

// printf("configFile = %s\n", configFile); // "C:/Criss/config.txt"

// const int lengthConfigFile = strlen(configFile);

// printf("%d\n", lengthConfigFile); // 31

// printf("isBlock = %s\n", isBlock); // "true"

const int lengthIsBlock = strlen(isBlock);

// printf("%d\n", lengthIsBlock); // 4

// printf("executableFile = %s\n", executableFile); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"、"C:/Criss/Python/Python311/python.exe"、"C:/Criss/py/Scripts/python.exe"、"C:/Criss/Nodejs/node.exe"

const int lengthExecutableFile = strlen(executableFile);

// printf("%d\n", lengthExecutableFile); // 52

// printf("interpreterFile = %s\n", interpreterFile); // "-p,4,--project=C:/Criss/jl/"

const int lengthInterpreterFile = strlen(interpreterFile);

// printf("%d\n", lengthInterpreterFile); // 54

// printf("scriptFile = %s\n", scriptFile); // "C:/Criss/jl/application.jl"、"C:/Criss/py/application.py"、"C:/Criss/js/application.js"

const int lengthScriptFile = strlen(scriptFile);

// printf("%d\n", lengthScriptFile); // 77

// printf("configInstructions = %s\n", configInstructions); // "interface\_Function=http\_Server,webPath=C:/Criss/html/,host=::0,port=10001,key=username:password,number\_Worker\_threads=1,isConcurrencyHierarchy=Tasks"

const int lengthConfigInstructions = strlen(configInstructions);

// printf("%d\n", lengthConfigInstructions); // 63

if (lengthExecutableFile > 0) {

// 拼接獲取控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數（shellCodeScript）：shellCodeScript = executableFile + " " + interpreterFile + " " + scriptFile + " " + configInstructions

// char defaultShellCodeScript[BUFFER\_LEN]; // 1024

char defaultShellCodeScript[lengthExecutableFile + 1 + lengthInterpreterFile + 1 + lengthScriptFile + 1 + lengthConfigInstructions + 1]; // 255 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

if (lengthScriptFile == 0) {

if (lengthInterpreterFile == 0) {

// char defaultShellCodeScript[lengthExecutableFile + 1]; // 53 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

snprintf(defaultShellCodeScript, sizeof(defaultShellCodeScript), "%s", executableFile); // 52 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 52

defaultShellCodeScript[lengthDefaultShellCodeScript] = '\0';

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

// const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 52

// shellCodeScript = strdup(defaultShellCodeScript); // 函數：strdup(defaultShellCodeScript) 表示，指針傳值，深拷貝，需要 free(shellCodeScript) 釋放内存;

shellCodeScript = defaultShellCodeScript; // C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/Python/Python311/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/py/Scripts/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", shellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"

// const int lengthDefaultShellCodeScript = strlen(shellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 52

}

if (lengthInterpreterFile > 0) {

// char defaultShellCodeScript[lengthExecutableFile + 1 + lengthInterpreterFile + 1]; // 113 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/"

snprintf(defaultShellCodeScript, sizeof(defaultShellCodeScript), "%s%s%s", executableFile, " ", interpreterFile); // 112 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/"

const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 112

defaultShellCodeScript[lengthDefaultShellCodeScript] = '\0';

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/"

// const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 112

// shellCodeScript = strdup(defaultShellCodeScript); // 函數：strdup(defaultShellCodeScript) 表示，指針傳值，深拷貝，需要 free(shellCodeScript) 釋放内存;

shellCodeScript = defaultShellCodeScript; // C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/Python/Python311/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/py/Scripts/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", shellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/"

// const int lengthDefaultShellCodeScript = strlen(shellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 112

}

}

if (lengthScriptFile > 0) {

if ((lengthInterpreterFile == 0) && (lengthConfigInstructions == 0)) {

// char defaultShellCodeScript[lengthExecutableFile + 1 + lengthScriptFile + 1]; // 131 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl"

snprintf(defaultShellCodeScript, sizeof(defaultShellCodeScript), "%s%s%s", executableFile, " ", scriptFile); // 130 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl"

const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 130

defaultShellCodeScript[lengthDefaultShellCodeScript] = '\0';

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl"

// const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 130

// shellCodeScript = strdup(defaultShellCodeScript); // 函數：strdup(defaultShellCodeScript) 表示，指針傳值，深拷貝，需要 free(shellCodeScript) 釋放内存;

shellCodeScript = defaultShellCodeScript; // C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/Python/Python311/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/py/Scripts/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", shellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl"

// const int lengthDefaultShellCodeScript = strlen(shellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 130

}

if ((lengthInterpreterFile > 0) && (lengthConfigInstructions == 0)) {

// char defaultShellCodeScript[lengthExecutableFile + 1 + lengthInterpreterFile + 1 + lengthScriptFile + 1]; // 191 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl"

snprintf(defaultShellCodeScript, sizeof(defaultShellCodeScript), "%s%s%s%s%s", executableFile, " ", interpreterFile, " ", scriptFile); // 190 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl"

const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 190

defaultShellCodeScript[lengthDefaultShellCodeScript] = '\0';

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl"

// const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 190

// shellCodeScript = strdup(defaultShellCodeScript); // 函數：strdup(defaultShellCodeScript) 表示，指針傳值，深拷貝，需要 free(shellCodeScript) 釋放内存;

shellCodeScript = defaultShellCodeScript; // C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/Python/Python311/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/py/Scripts/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", shellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl"

// const int lengthDefaultShellCodeScript = strlen(shellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 190

}

if ((lengthInterpreterFile == 0) && (lengthConfigInstructions > 0)) {

// char defaultShellCodeScript[lengthExecutableFile + 1 + lengthScriptFile + 1 + lengthConfigInstructions + 1]; // 195 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

snprintf(defaultShellCodeScript, sizeof(defaultShellCodeScript), "%s%s%s%s%s", executableFile, " ", scriptFile, " ", configInstructions); // 194 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 194

defaultShellCodeScript[lengthDefaultShellCodeScript] = '\0';

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 194

// shellCodeScript = strdup(defaultShellCodeScript); // 函數：strdup(defaultShellCodeScript) 表示，指針傳值，深拷貝，需要 free(shellCodeScript) 釋放内存;

shellCodeScript = defaultShellCodeScript; // C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/Python/Python311/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/py/Scripts/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", shellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// const int lengthDefaultShellCodeScript = strlen(shellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 194

}

if ((lengthInterpreterFile > 0) && (lengthConfigInstructions > 0)) {

// char defaultShellCodeScript[lengthExecutableFile + 1 + lengthInterpreterFile + 1 + lengthScriptFile + 1 + lengthConfigInstructions + 1]; // 255 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

snprintf(defaultShellCodeScript, sizeof(defaultShellCodeScript), "%s%s%s%s%s%s%s", executableFile, " ", interpreterFile, " ", scriptFile, " ", configInstructions); // 254 // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 254

defaultShellCodeScript[lengthDefaultShellCodeScript] = '\0';

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", defaultShellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// const int lengthDefaultShellCodeScript = strlen(defaultShellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 254

// shellCodeScript = strdup(defaultShellCodeScript); // 函數：strdup(defaultShellCodeScript) 表示，指針傳值，深拷貝，需要 free(shellCodeScript) 釋放内存;

shellCodeScript = defaultShellCodeScript; // C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/Python/Python311/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/py/Scripts/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", shellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// const int lengthDefaultShellCodeScript = strlen(shellCodeScript);

// printf("%d\n", lengthDefaultShellCodeScript); // 254

}

}

// printf("C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數爲：\n%s\n", shellCodeScript); // "C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("%d\n", strlen(shellCodeScript));

// 使用函數：FILE \*popen(const char \*command, const char \*type)，調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）;

char \*IDsubprocess = "";

int pcloseReturn;

char buffer[BUFFER\_LEN]; // BUFFER\_LEN = 1024 // 代碼首部自定義的常量：BUFFER\_LEN = 1024，靜態申請内存緩衝區（buffer），存儲子進程標準輸出通道（stdout）中的返回值的每一個橫向列（row）中的内容，要求子進程標準輸出通道（stdout）中的返回值的每一個橫向列（row）最多不得超過 1024 個字符;

memset(buffer, 0, sizeof(buffer)); // 存緩衝區（buffer）賦初值;

FILE \*fstream = popen(shellCodeScript, "r"); // 取："r" 值表示只讀，取："w" 表示只寫，只能二選一;

if (fstream == NULL) {

printf("調用操作系統（Linux、Window）控制臺命令列（shell、cmd）啓動可執行檔（.exe）失敗.\nshell code script:""\n%s\n", shellCodeScript); // C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/Python/Python311/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/py/Scripts/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

// printf("shellCodeScript = %s\n", shellCodeScript); // C 語言調用操作系統（Linux、Window）控制臺命令列（shell、cmd）運行二進制可執行檔（.exe）的字符串參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe -p 4 --project=C:/Criss/jl/ C:/Criss/jl/application.jl IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/Python/Python311/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"、"C:/Criss/py/Scripts/python.exe C:/Criss/py/application.py IP=::0 port=10001 Is\_multi\_thread=false number\_Worker\_threads=1"

return 1;

}

if (fstream != NULL) {

if ((lengthIsBlock == 0) || (strncmp(isBlock, "true", sizeof(isBlock)) == 0)) {

// 循環讀取子進程程式指令執行的返回值;

while (fgets(buffer, sizeof(buffer), fstream) != NULL) {

printf("%s\n", buffer);

// 從子進程的自定義返回值信息中，提取子進程的 ID 號值;

if (strlen(buffer) > 0) {

// 從：Julia statistical server 自定義的返回值信息中，提取子進程的 ID 號值;

if (strstr(buffer, " listening on") != NULL) {

IDsubprocess = strdup(buffer);

memset(buffer, 0, sizeof(buffer)); // 釋放内存緩衝區（buffer）;

// printf("%s\n", IDsubprocess);

IDsubprocess = strtok(IDsubprocess, " ");

// if ((strstr(IDsubprocess, "process") != NULL) || (strstr(IDsubprocess, "thread") != NULL) || (strstr(IDsubprocess, "task") != NULL)) {

// snprintf(buffer, sizeof(buffer), "%s", IDsubprocess);

// }

while( IDsubprocess != NULL ) {

if ((strstr(IDsubprocess, "process") != NULL) || (strstr(IDsubprocess, "thread") != NULL) || (strstr(IDsubprocess, "task") != NULL)) {

// printf("%s\n", IDsubprocess);

if (strlen(buffer) > 0) {

snprintf(buffer, sizeof(buffer), "%s%s%s", buffer, " ", IDsubprocess);

} else {

snprintf(buffer, sizeof(buffer), "%s", IDsubprocess);

}

IDsubprocess = strtok(NULL, " ");

} else {

break;

}

}

buffer[strlen(buffer)] = '\0';

IDsubprocess = strdup(buffer);

// printf("Subprocess ID: %s\n", IDsubprocess);

}

}

memset(buffer, 0, sizeof(buffer)); // 釋放内存緩衝區（buffer）;

}

}

if (strncmp(isBlock, "false", sizeof(isBlock)) == 0) {

// 只讀取子進程程式指令執行的返回值的第一個橫向列（row）;

if (fgets(buffer, sizeof(buffer), fstream) != NULL) {

// printf("%s\n", buffer);

// 從子進程的自定義返回值的第一個橫向列（row）的信息中，提取子進程的 ID 號值;

if (strlen(buffer) > 0) {

// 從：Julia statistical server 自定義的返回值的第一個橫向列（row）的信息中，提取子進程的 ID 號值;

if (strstr(buffer, " listening on") != NULL) {

IDsubprocess = strdup(buffer);

memset(buffer, 0, sizeof(buffer)); // 釋放内存緩衝區（buffer）;

// printf("%s\n", IDsubprocess);

IDsubprocess = strtok(IDsubprocess, " ");

// if ((strstr(IDsubprocess, "process") != NULL) || (strstr(IDsubprocess, "thread") != NULL) || (strstr(IDsubprocess, "task") != NULL)) {

// snprintf(buffer, sizeof(buffer), "%s", IDsubprocess);

// }

while( IDsubprocess != NULL ) {

if ((strstr(IDsubprocess, "process") != NULL) || (strstr(IDsubprocess, "thread") != NULL) || (strstr(IDsubprocess, "task") != NULL)) {

// printf("%s\n", IDsubprocess);

if (strlen(buffer) > 0) {

snprintf(buffer, sizeof(buffer), "%s%s%s", buffer, " ", IDsubprocess);

} else {

snprintf(buffer, sizeof(buffer), "%s", IDsubprocess);

}

IDsubprocess = strtok(NULL, " ");

} else {

break;

}

}

buffer[strlen(buffer)] = '\0';

IDsubprocess = strdup(buffer);

// printf("Subprocess ID: %s\n", IDsubprocess);

}

}

memset(buffer, 0, sizeof(buffer)); // 釋放内存緩衝區（buffer）;

}

// // 循環讀取子進程程式指令執行的返回值;

// while (fgets(buffer, sizeof(buffer), fstream) != NULL) {

// // printf("%s\n", buffer);

// // 從子進程的自定義返回值信息中，提取子進程的 ID 號值;

// if (strlen(buffer) > 0) {

// // 從：Julia statistical server 自定義的返回值信息中，提取子進程的 ID 號值;

// if (strstr(buffer, " listening on") != NULL) {

// IDsubprocess = strdup(buffer);

// memset(buffer, 0, sizeof(buffer)); // 釋放内存緩衝區（buffer）;

// // printf("%s\n", IDsubprocess);

// IDsubprocess = strtok(IDsubprocess, " ");

// // if ((strstr(IDsubprocess, "process") != NULL) || (strstr(IDsubprocess, "thread") != NULL) || (strstr(IDsubprocess, "task") != NULL)) {

// // snprintf(buffer, sizeof(buffer), "%s", IDsubprocess);

// // }

// while( IDsubprocess != NULL ) {

// if ((strstr(IDsubprocess, "process") != NULL) || (strstr(IDsubprocess, "thread") != NULL) || (strstr(IDsubprocess, "task") != NULL)) {

// // printf("%s\n", IDsubprocess);

// if (strlen(buffer) > 0) {

// snprintf(buffer, sizeof(buffer), "%s%s%s", buffer, " ", IDsubprocess);

// } else {

// snprintf(buffer, sizeof(buffer), "%s", IDsubprocess);

// }

// IDsubprocess = strtok(NULL, " ");

// } else {

// break;

// }

// }

// buffer[strlen(buffer)] = '\0';

// IDsubprocess = strdup(buffer);

// // printf("Subprocess ID: %s\n", IDsubprocess);

// memset(buffer, 0, sizeof(buffer)); // 釋放内存緩衝區（buffer）;

// break;

// }

// }

// memset(buffer, 0, sizeof(buffer)); // 釋放内存緩衝區（buffer）;

// }

}

pcloseReturn = pclose(fstream); // 關閉子進程程式指令執行的返回值的讀取管道;

// printf("pclose return = %d\n", pcloseReturn);

if (pcloseReturn == 1) {

// printf("子進程（%s）程式指令爲空.\n", IDsubprocess);

printf("subprocess program ( %s ) instruction is empty.\n", IDsubprocess);

} else if (pcloseReturn == -1) {

printf("創建子進程（%s）程式指令失敗.\n", IDsubprocess);

} else if (pcloseReturn == 0x7f00) {

printf("子進程（%s）程式指令錯誤無法執行.\n", IDsubprocess);

} else if (pcloseReturn == 0) {

// printf("子進程（%s）程式指令執行完畢.\n", IDsubprocess);

printf("subprocess program ( %s ) instruction completed.\n", IDsubprocess);

// if (WIFEXITED(pcloseReturn)) {

// printf("子進程程式指令執行完畢，返回值爲：%d\n", WEXITSTATUS(pcloseReturn));

// } else if (WIFSIGNALED(pcloseReturn)) {

// printf("子進程程式指令被「中止」信號結束，「中止」信號值爲：%d\n", WTERMSIG(pcloseReturn));

// } else if (WSTOPSIG(pcloseReturn)) {

// printf("子進程程式指令被「暫停」信號暫停，「暫停」信號值爲：%d\n", WSTOPSIG(pcloseReturn));

// }

} else {}

}

if (strlen(IDsubprocess) > 0) { free(IDsubprocess); }

} else {

printf("解釋器（Exempli gratia: Julia、Python、Node.js、Java）可執行檔啓動路徑參數爲空：\nexecutableFile = %s\n", executableFile); // 解釋器（Julia、Python、Node.js、Java）的二進制可執行檔啓動路徑參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"、"C:/Criss/Python/Python311/python.exe"、"C:/Criss/py/Scripts/python.exe"、"C:/Criss/Nodejs/node.exe"

// printf("executableFile = %s\n", executableFile); // 解釋器（Julia、Python、Node.js、Java）的二進制可執行檔啓動路徑參數："C:/Criss/Julia/Julia-1.9.3/bin/julia.exe"、"C:/Criss/Python/Python311/python.exe"、"C:/Criss/py/Scripts/python.exe"、"C:/Criss/Nodejs/node.exe"

return 1;

}

if (strlen(configFile) > 0) {

free(configFile);

// configFile = NULL;

}

if (strlen(isBlock) > 0) {

free(isBlock);

// isBlock = NULL;

}

if (strlen(executableFile) > 0) {

free(executableFile);

// executableFile = NULL;

}

if (strlen(interpreterFile) > 0) {

free(interpreterFile);

// interpreterFile = NULL;

}

if (strlen(scriptFile) > 0) {

free(scriptFile);

// scriptFile = NULL;

}

if (strlen(configInstructions) > 0) {

free(configInstructions);

// configInstructions = NULL;

}

if (strlen(shellCodeScript) > 0) {

free(shellCodeScript);

// shellCodeScript = NULL;

}

return 0;

}

」