使用手册（User Manual）

前言

生成器结构简单使用python os库和shutil库来完成。支持简单的文件或者文件夹的增删，设计初衷主要是为了方便起C/C++工程，特别是嵌入式领域库的种类繁多，文件复杂，只有ARM系列部分的产品有官方的生成器，导致其他芯片特别是国产的芯片的工程创建比较复杂，手动的创建文件夹和工程项目拉取文件这些操作重复又麻烦，在开发库的时候一个个拉文件也比较痛苦。这脚本可以通过几个简单的指令来完成复杂的创建，目前仅支持windows。当然这个脚本比不上Cmake和makefile这些成熟的产品的。也没用替代这些东西的想法，但是C语言的许多工具都没有在创建工程方面做出很好的工具，加之没用包管理所以有时候开发需要拉很长时间的依赖。虽然主流的方式是准备模板工程，但是我觉的还是有一个工具来解决这个问题更好。

当然你用这个批量的创建其他文件也是一样可行的。

> ⚠ **Warning**
>
> 所有的指令都是区分大小写的，仅支持小写，别做规则外的事情，否则会发生意想不到的错误。所有的文件路径未作说明的全部都代表绝对路径，本身并没有对相对路径做支持。我TM就一个人写不了太大的东西，有时间才更新。如果有BUG我会记住修不修，有时间再说吧。

如何使用脚本

安装好python之后，在控制台使用py指令运行脚本即可，开发使用的python本版为3.9

指令列表

- yl：这个指令就是依（yi）赖（lai）的拼音，用来执行写好的CPM文件生成模板工程。
- crt：Create的缩写，用于创建文件和文件夹。
- cd：不用说了吧懂得都懂，打开一个文件夹。
- cpy：copy的缩写，用来复制文件或者文件夹到指定的目录，功能类似yl，但是他们的区别是，yl只能使用文件来自动创建，cpy是在控制台以指令的形式创建的。
- del：删除文件或者文件夹。
- help：获取帮助的指令，暂时没有做。
- q or Q：唯一一个不区分大小写的指令，用于退出程序。

语法

指令：yl （这个最后读）

yl指令使用非常简单，一下是一个示例：

```
yl D:\xxx\xxx.cpm
```

在理解这个指令之前需要说明一个概念叫做cpm，yl指令只能解析后缀名为cpm的文件，你可以编写好cpm文件来描述你的工程创建操作，注意我这里的表述是创建工程的操作而非工程本身。这样做的原因是减少学习成本，只要你学会了后文的所有指令就可以自由编写CPM文件，而且为了减少编写成本提高体验还增加了一些方便的功能。

cpm文件

cpm是**C Program Maker**的缩写，创建文件的时候记住文件名小写，因为我没做大写的判断（

cpm文件的执行方式是从上到下顺序的。

cpm文件的编写非常方便只需要像控制台敲指令一样把指令一条条敲进去就行了，为了方便cpm是支持变量的，但是这个变量的设计只是为了省的写一堆重复的路径，所以功能上这个东西本质就是C语言的宏替换，只不过你可以在声明后的任意地方修改值，因此他确实是个变量。

变量的声明格式很简单：

```
var name = "value"
define name value       //这个也是变量，懒得写常量的处理了，define只是提供一种更简洁的方式，也算是个C语言彩蛋
```

需要注意的是以上的所有方式空格都不能省略，等号前后必须有空格，否则会出BUG，因为解析方式是按空格分割读取的指令，还需要注意的是在使用var声明并初始化的变量需要在value内使用空格应该使用"_"来代替例如:

```
var name = "v_a_l_u_e" 解析后value -> v a l u e
```

变量在声明的时候一定要初始化，因为我没有写未初始化的解析（我这是在帮你们规范编码习惯指正doge）。

变量声明后的使用相对死板，毕竟我不可能写一个真的语言出来是吧。就几百行的代码不可能那么完善，我又不是计算机之神。

变量的使用方式如下：

```
#直接使用变量
var program_path = "D:\xxxx\Progream"   #工程路径
crt $program_path                       #crt创建工程文件夹

#混合其他文本使用变量
var lib = "Core"                        #库路径
cpy D:\program D:\xxxlib\$lib           #复制库到指定文件夹
cpy D:\program D:\xxxlib\$lib$\Src|Inc  #复制库到指定文件夹
```

变量的使用方法非常粗糙，有且必须在变量名的前面加$符号，如果在变量右侧拼接其他的字符需要在另外一侧也添加$符号。前面说了变量的用法非常死板所有不要期望这样使用：

```
#多变量的指令
var program_path = "D:\Program"
var main_path = "D:\xxxLibe"
var src_path = "\core\Src"
var inc_path = "\core\inc"

cpy $program_path $main_path$src_pth    #这种方式是禁止的，错误的在一条指令的每一个由空格分割段内只能出现一个变量
cpy $program_path $main_path$\core\Src  #这种方式是允许的，你可以在每一个段内使用一个变量
```

（吐槽）虽然这还不够好使，但是目前只能开发成这样了，这种字符串解析的小把戏就应该交给AI，人写太浪费时间了。

当然你也应该注意到了这个文件是支持单行注释的，为了照顾习惯注释使用两个符号随意选择，"#"和";"都可以作为注释符号使用，解析的时候会自动丢弃这些符号后面的内容。

因为在执行crt的时候对C语言文件做了特殊处理，为了不修改太多代码，减少工作量的同时不让main.c文件的创建打断这个自动化流程，所以直接搞了一个扩展指令"-inc"，他的作用是描述main.c在个性化时需要手动敲入的内容。一下是一个用例：

```
crt D:\Program\Core\Src main.c -inc <stdio.h> <string.h> "main.h"
```

在编写cpm文件的时候，这个"-inc"指令是必须的否则会打断cpm的自动化流程，后面的内容虽然不建议缺省，但是缺了也应该没啥事。没有重写之前的代码，只是少量增添了一些让他能兼容，所以这个指令你在控制台也可以用。

cpm的文件编写还考虑到了一个需求就是文本过长想要换行怎么办？如果你是C程序员想必不会陌生，没错整个软件的设计逻辑都是基于"define"的扩展，所以你可以在行末输入一个"\"来表示换行，例如：

```
crt D:\Program\Core\Src main.c -inc \
<stdio.h> <string.h> "main.h"
```

不过不能随便换行，例如下面这种就会引发不可预测的错误：

```
crt D:\Program\Core\Src ma\
in.c -inc <stdio.h> <string.h> "main.h"
```

不能打断连续的文本哦~，这样搞 main.c 就会呗分割为 ma 和 in.c 了！

cpm现在还支持混编Makefile和CMakeLists，混编的方式非常简单格式如下：

```
Makefile        #编写makefile
xxxxxx
xxxxxx
end

CMakeLists      #编写CMakeLists
xxxxxx
xxxxxx
end
```

没有功能是完美的，所以请注意一个问题，在混编之前必须保证文件已经创建，为了避免产生意外的错误，在没检测到文件创建的时候是不会处理的。end作为文件的结尾，当然你愿意也可以大写为END。目前仅支持单个创建，本身这个创建器就是针对单个工程设计的，所以你想在不同的文件夹创建不同的Makefile或者CMake恐怕有点困难。不过值得一提的是也不是完全的不可能，因为文件的路径是在创建文件的时候检测，所以可以使用crt指令来创建新的文件，又因为文件是顺序执行的所以可以使用这种方式来创建不同的Makefile和CMakeLists不过我并未测试这种，所以可能会引发BUG。

cpm以上就是cpm的所有内容了。

指令：crt

crt指令可以创建文件和文件夹，对于文件夹一次仅支持单个文件夹，文件可以在一条指令内创建多个，在创建文件时路径可以缺省，但是需要使用`"-f"`来表明创建的是文件，此时创建的文件将目前打开的路径作为默认路径使用。在创建扩展名相同的文件时可以使用`"*extension"`来表明后面文件的扩展名，以此来省略重复的敲扩展名工作。在创建文件时如果使用未创建的路径作为目标，会自动为你创建目录。在创建目录时如果不使用绝对路径会自动在打开的目录下创建。例子如下：

- 不缺省路径创建文件：

```
crt D:\Program\Core\Src -f main.c                    #单文件 -f可省略
crt D:\Program\Core\Src main.c
crt D:\Program\Core\Src -f main.c gpio.c adc.c       #多文件 -f可省略
crt D:\Program\Core\Src main.c gpio.c adc.c
```

- 缺省路径创建文件：

```
crt -f main.c               #-f 不可省略
crt -f main.c gpio.c adc.c  #-f 不可省略
```

- 缺省扩展名创建：

```
crt D:\Program\Core\Src *c main gpio adc
crt D:\Program\Core\Src *c main
```

- 创建文件夹：

```
//虽然crt指令可以不使用绝对路径创建文件夹，但是不建议，因为这个功能有潜在的BUG没有测试，这个功能纯粹是意外
crt D:\Program\Core\Inc
crt core                    #此处假定Program已经创建并打开。
```

> **Note**
>
> 需要注意的是，如果脚本运行时没有任何文件夹打开，系统缺失目录，在第一次创建文件夹的时候一定要使用绝对路径，有一个很方便的功能是，在缺省目录的情况下，第一个被创建的目录会被自动打开。

在开启脚本的时候可以注意一下，提示，控制台界面会显示当前打开的目录：

```
>>  >> crt D:\Program        #开始为空
>> D:\Program >>            #创建之后自动打开，并在两个">>"之间显示
```

crt指令对C工程进行了特化，会自动检测你的文件后缀名，当然这个功能的检测没有做的特别好，除了特殊的文件名"main.c"外，其他的文件仅在使用缺省文件名创建时检测其后缀。

如果检测到了.h文件，会自动写入头文件保护等内容：

```
#ifndef __TEST_H__
#define __TEST_H__

#ifdef __cplusplus
extern "C"{
#endif

#ifdef __cplusplus
}
#endif
#endif /* __TEST_H__ */
```

如果检测到了.c文件会自动为其写入同名.h文件的`#include`，但是如果检测到main.c会自动为您创建模板，并询问您需要加入的库文件，自动加入`#include`。

```
crt D:\Program\Core\Src main.c //也可以使用缺省方式 *c main 最好使用缺省方式
检测到main.c文件，是否需要个性化生成[Y/N] Y
输入库文件:输入库文件:<stdio.h> <string.h> "main.h"

//输出
#include <stdio.h>
#include <string.h>
#include "main.h"

int main()
{
    return 0;
}

```

如果你选择了N则生成空白文件不做任何处理。

指令：cd

这个真没啥好说的cd + 路径就行了，只能作用于文件夹。

```
>>  >>cd D:\Program
//如果有这个文件夹
open >> D:\Program
>> D:\Program >>
//如果没有这个文件夹
E:path error or Folder not found
```

指令：cpy

这个指令真心不好设计，怎么搞都感觉不好使，最后妥协成这样。

以下是一个基本用例：

```
cpy D:\Program D:\xxLib\Core|Device D:\xxLib\Core\Src\xx.c|xx.h
D:\xxLib\Core\Inc D:\xxLib\readme.md
```

这个用例表明了所有可以使用的cpy格式使用"|"符号来连接同级目录或者文件。虽然文件夹和文件都可以复制，但是需要注意的是使用"|"来连接一个目录下的多个文件或者文件夹，必须保证不混合文件和文件夹，例如以下的做法是不行的，可能导致意外的错误：

```
cpy D:\Program D:\xxLib\Core|xx.h
```

这种方式是禁止的，虽然看着很方便，但是我的代码没做这样的处理，大概率会报错，主要是因为复制文件和复制整个文件树需要使用的API不一样。目前结构比较死板，以后那天心情好重构再支持混合。

指令：del

这指令和cd一样，指令名+路径就行了，最好使用绝对路径，相对路径可能导致意外的问题。

用例：

```
del D:\Program          #删除目录
del D:\Program\main.c   #删除文件
```

指令：help（待完善）

这指令目前只做了简单的提示，没有完全做完，主要是整个程序没有完成，再加上我有重构的想法。

但是语法还是有的直接敲help就能看到用法。