

Shell 其实就是一种脚本语言，也是一个可以用来连接内核和用户的软件，我们编写完源码后不用编译，直接运行源码即可。

bash, a general purpose scripting language.

command [arguments]

interactive and batch / 交互式和批处理

Gnuplot (在Linux下基于Qt的一套绘图工具)

https://blog.csdn.net/weixin\_42014622/article/details/82962634

LaTeX（LTEX，音译“拉泰赫”）是一种基于ΤΕΧ的排版系统

https://blog.csdn.net/pipisorry/article/details/54571521

关于shell 和 bash 之间的关系

<https://blog.csdn.net/weixin_42432281/article/details/88392219>

|  |  |  |
| --- | --- | --- |
| Platform | Linux | Windows |
| 字符界面模式 | shell | Command Prompt |
| 命令行 | bash | command |
| 批处理模式 | batch mode | 批处理 bat |

commands: aliases, functions, built-ins, keywords, and executables.

Text, letter

Description automatically generated

**Here document** 的使用总结

https://blog.csdn.net/liumiaocn/article/details/86715953

Sudo apt install / upgrade/update \*\*\*\*

Sudo apt-get install \*\*\*\*\*

确定型随机数是一个伪随机数，因为给定一相相同的种子seed，它就会生成相同的随机数，所以这种随机是假的，称为伪随机数。

如果想得到不同的随机序列，需要确保每次使用不同的种子，生成不同的序列。

Mersenne Twister算法译为马特赛特旋转演算法，是伪随机数发生器之一，其主要作用是生成伪随机数。

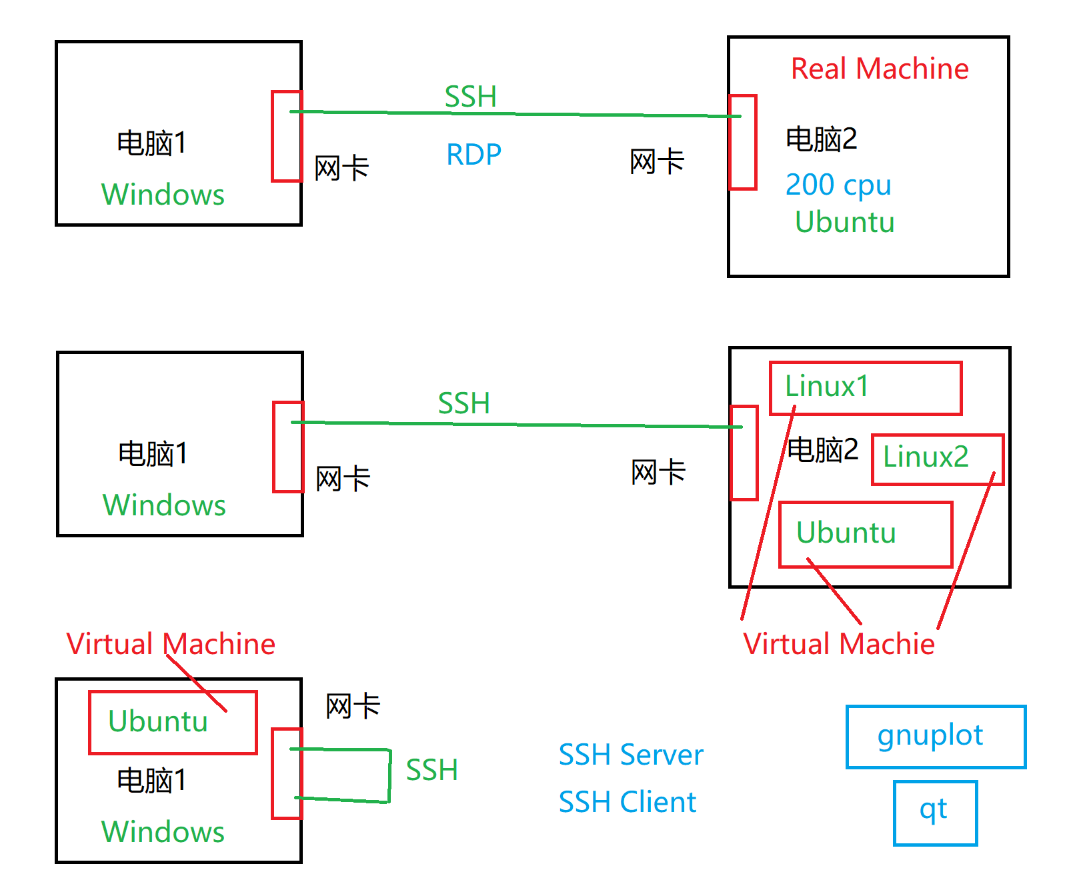
由于bash是不支持浮点型的计算，bash利用expr只能计算整数的基本运算。目前如果想要在bash中计算浮点型，只能通过bc或者awk来帮忙。

Echo $(expr $n1 + $n2)

Echo $(( $n1 + $n2 ))

Touch 创建文件

Vim/vi 文件，也可以创建文件



关于makefile和make:

<https://blog.csdn.net/ZBraveHeart/article/details/123187908>

Makefile文件中的命令有一定规范，一旦该文件编写好以后，在Linux命令行中执行一条make命令即可自动编译整个工程。

Text

Description automatically generated

它表示，为了达成目标，必须1，先有依赖，2，顺序执行下面的命令

Makefile并不会关心命令是如何执行的，仅仅只是会去执行所有定义的命令，和我们平时直接输入命令行是一样的效果。

1、目标即要生成的文件。如果目标文件的更新时间晚于依赖文件更新时间，则说明依赖文件没有改动，目标文件不需要重新编译。否则会进行重新编译并更新目标文件。

2、默认情况下Makefile的第一个目标为终极目标。

3、依赖：即目标文件由哪些文件生成。

4、命令：即通过执行命令由依赖文件生成目标文件。注意每条命令之前必须有一个tab保持缩进，这是语法要求（会有一些编辑工具默认tab为4个空格，会造成Makefile语法错误）。

5、all：Makefile文件默认只生成第一个目标文件即完成编译，但是我们可以通过all 指定所需要生成的目标文件。

**makefile中wildcard的理解**

<https://blog.csdn.net/dianqicyuyan/article/details/123509400>

wildcard 用来明确表示通配符。因为在 Makefile 里，变量实质上就是 C/C++ 中的宏，也就是说，如果一个表达式如 objs = \*.o ，则 objs 的值就是 \*.o ，而不是表示所有的 .o 文件。若果要使用通配符，那么就要使用 wildcard 来声明 \* 这个符号，使 \* 符号具有通配符的功能。

gnu make工具的[Makefile](https://so.csdn.net/so/search?q=Makefile&spm=1001.2101.3001.7020" \t "_blank)中$(SRC:%.c=%.o)的含义：**将SRC变量中所有以.c结尾的文件名替换成对应的以.o结尾的文件名，然后赋回给SRC。**

[makefile](https://so.csdn.net/so/search?q=makefile&spm=1001.2101.3001.7020)**中.PHONY的作用是什么？**

<https://blog.csdn.net/weixin_41969690/article/details/106727385>

.PHONY all clean起到“欺骗”的作用，欺骗make别管别的，让你干嘛你就干嘛的意思。

缺省情况下，makefile中的target是一个文件，上面的定义就是告诉make命令，后面的target不是文件，而只是一个label而已

Makefile有三个非常有用的变量。分别是$@，$^，$<代表的意义分别是：

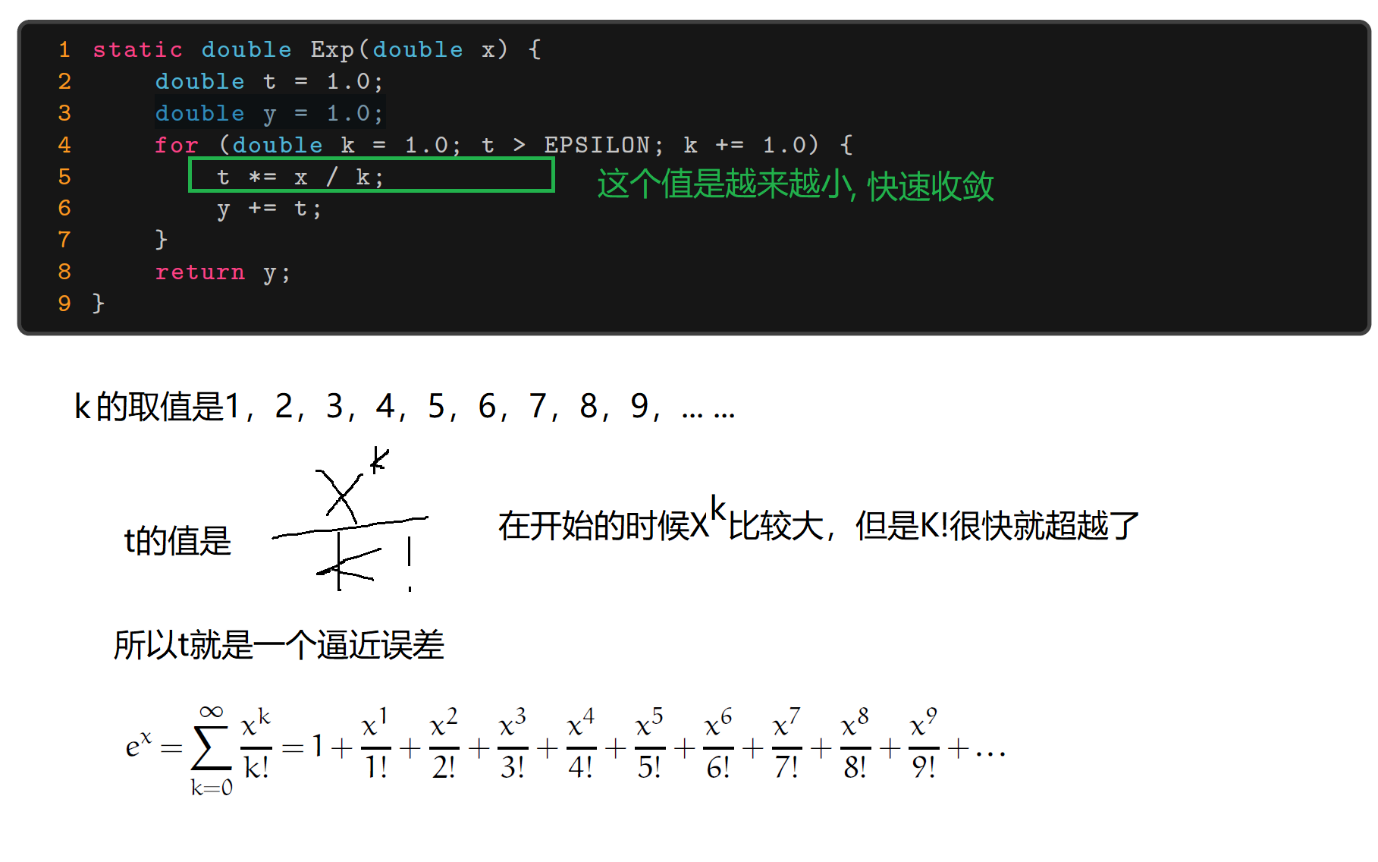
$@--目标文件，$^--所有的依赖文件，$<--第一个依赖文件。

mathlib.o : mathlib.c mathlib-test.c …

$< = mathlib.c

$@ = mathlib.o

$^ = mathlib.c mathlib-test.c …



关于LateX 排版系统

<https://blog.csdn.net/weixin_35805593/article/details/113454369>

documentclass{class} 为配置环境，可大致理解为C++中的 using namespace std;

usepackage{}为宏包调用，可大致理解为 #include<>等库函数的调用

正文区，以begin{document}打头，且有且只能有一个。这一点又与C++中的main函数如出一辙

latex运用%来做注释

主要用latex来输出数学公式

数学公式的输出有两种形式，一种为行内输出一种为行间输出。

分别用$ $ 与 $$ $$来表示。

latex的一个重大特点是“所见不是所得”，因此我们在命令行中另起一行，生成的pdf文件并不会换行，latex的换行指令为空一行另起。

{} 表示作为一个整体来统一处理

\[ \] 表示另起一行的公式，相当于 $$ $$

c\_m 2^m 表示下标和上标

软件人员必须了解的几个硬件知识

中央处理器CPU，内存Memory，硬盘Hard Disk

C/C++语言必须了解的几个底层知识

内存空间，地址，指针，数组，数组长度/大小

&a[0] = a &a[1] = a+1, 同一个数组内才能这样操作

位数：32位，可访问地址空间为2^32 = 4G，64位可访问地址空间为2^64，

18,446,744,073,709,551,616，理论上很大的一个值，但实际没有用到，足够程序运行

数据结构相关的几个知识

数组Array（可变数组），链表LinkedList（单向链表/双向链表），队列Queue，栈Stack，堆Heap，二叉树 Binary Tree，内存

堆Heap就是一棵完全二叉树，同时满足最大堆或最小堆的

对堆栈的误解

默认情况下，在32 位上运行的线程的最大栈大小为256 KB，64 位上的最大栈大小为512 KB。可以通过系统函数进行修改。

关于程序设计的几点提示：（以asgn2为例子）

1. 读懂需求 （主要是老师提供的asgn2.pdf，特别是Your task部分，一定要读懂每一句的意思）
2. 列出文件，就是在Deliverables节中的内容

Mathlib.h 函数的申明，不能修改

Mathlib.c 在mathlib.h中的函数的实现

Mathlib-test.c main函数必须在这个文件，解析命令行，定义OPTIIONS

Makefile

README.md

DESIGN.pdf

WRITEUP.pdf

1. 根据文件的功能，先实现框架，保证能运行，可测试

比较明确的细节可以先用注释标出

4，再细化每一个文件的功能

细化的原则也是先简单，后细节

5，对复杂的问题进行拆解，分成多个函数，每个函数也是先框架，再细节

6，每细化一个函数，测试一个函数

测试时可以通过输出，断点查看

7，完成步骤2中的所有文件

有些工具必须先准备好，“工欲善其事，必先利其器”，“事半功倍”

比如Visual Studio，Windows和Ubuntu上文件的共享，Ubuntu上的可视化编辑器，等等

VS 2022 Professional key

共享文件夹，修改Ubuntu屏幕大小

<https://blog.csdn.net/Inochigohan/article/details/119791518>

作为CS专业，要善于利用网络资源，比如一些教学视频，在线的小工具

概率论和数理统计

<https://www.bilibili.com/video/BV1Pb411771f/?p=9&spm_id_from=pageDriver&vd_source=342115c1d2b02981c05a987047b37614>

求补码计算器

<http://www.99cankao.com/numbers/twos-complement.php>

二进制计算器

<https://miniwebtool.com/zh-cn/binary-calculator/?number1=0100&operate=1&number2=1010>

理论上的内容，最后用小程序进行验证等

比如Quiz上的题目

关于函数的调用

1，函数的调用可以分为三类：自己的函数调用，系统库函数调用，以及第三方库函数调用

但是本质上是一样的，都包括

a, 函数的申明(declare)，说明函数的名称，参数（包括数量和类型），返回值（类型）

b函数的实现(implement)，就是函数的功能

通常情况下，函数的申明放在.h文件中，函数的实现一般放在.c文件中，而调用者必须include 相应的.h文件，.c文件不用include，但是.c文件必须一起在一个工程里进行link

比如：

rsa.h 和 rsa.c randstate.h 和 randstate.c numtheory.h 和 numtheory.c

如果是自己的函数调用，而且自己定义的函数功能比较单一，只为某个文件中的某个特定功能来实现，可以不用.h文件，而直接和调用者放在同一个.c文件中

比如：

void usage(char\* exec)

void output\_mpz\_t(char\* pre, mpz\_t op)

但其必须放在调用者的前面，这时候，函数的实现，同时也带有函数的申明功能

如果不想把函数的实现放在调用者的前面的话，就必须在调用者之前单独进行函数的申明

【举例说明】

void usage(char\* exec);

… …(调用者代码)

void usage(char\* exec) ｛… …｝

2，对于系统库和第三方库，不可能提供.c文件给我们，但是他们一定会提供.h给我们，否则，我们就不知道该如何调用函数，包括函数名，函数参数类型，参数个数，返回值类型等，所以一定会提供.h文件

比如：

gmp.h (第三方库) stdio.h （系统库）

虽然系统库和第三方库不能提供.c文件，但是它们一定会提供由.c所生成的库文件给我们，否则我们就只能通过compile（编译过程，只需要.h文件，就是函数的申明），无法通过link（链接过程，必须需要函数的实现，要么是.c文件，要么是库文件）

|  |  |  |  |
| --- | --- | --- | --- |
|  | 函数的申明 | 函数的实现 |  |
| 自己定义的函数 | .h | .c | .c加入工程中 |
| 第三方库函数 | .h | .c生成的库文件 | 库文件加入工程中 |
| 系统库函数 | .h | .c生成的库文件 | 不用我们加，link会自动加 |

关于库的使用（静态库和动态库）

前面说了，函数的实现是放在.c文件中，实现函数的具体功能，

比如

int mpz\_set\_str (mpz\_ptr, const char \*, int);

如果我不想让别人知道我的具体实现，或者担心别人修改了我的实现代码，导致调用者出错，我们一般不愿意直接把.c文件提供给别人，但是如果你不提供函数的实现，别人就无法link，以至无法run。

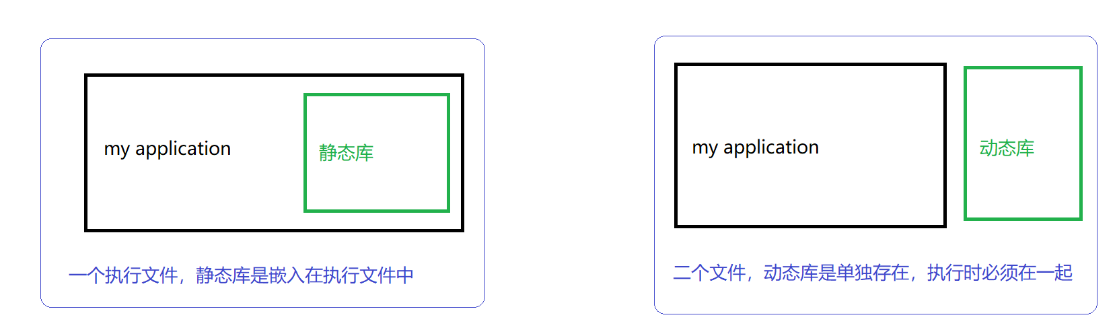
几乎所有的OS都提供了一种机制，保证你不提供源代码，只提供一种叫库的文件（别人无法看到具体的实现，当然也无法进行修改）

函数库可分为二类：

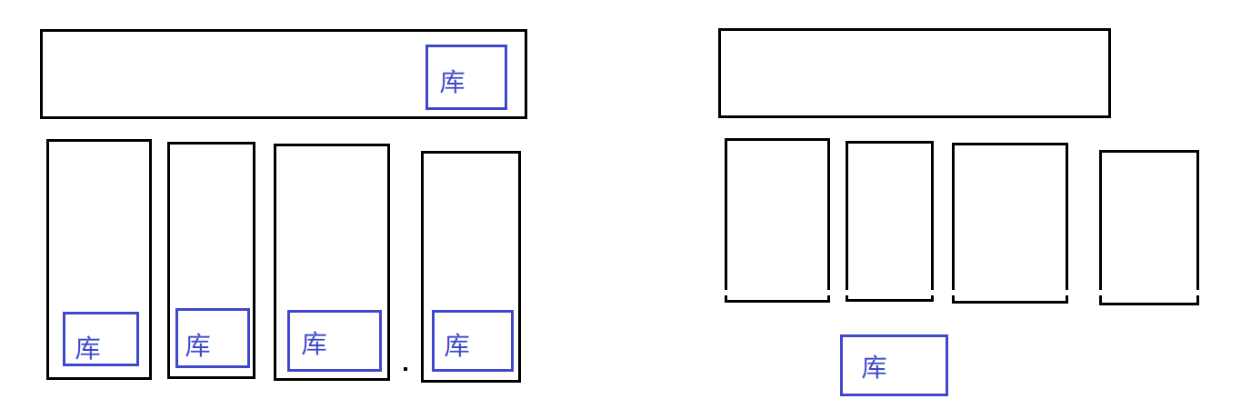
静态库 static library

动态库 dynamic library

虽然它们的存在形式不一样，但它们都是由.c文件编译而成，是包含了函数的具体实现



|  |  |  |
| --- | --- | --- |
|  | 好处 | 坏处 |
| 静态库 | 只有一个可执行文件 | 在多模块情况下，有重复代码，浪费空间 |
| 动态库 | 只有一份代码，节省空间 | 有多个文件，少了任一个都不能执行 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 函数申明 | 函数实现 | 何时调用 | 其它辅助文件 |
| 静态库 | .h | .lib (windows)  .a (unix类) | 在link时 | 无 |
| 动态库 | .h | .dll (windows)  .o (unix类) | 在run时 | 一个小的.lib文件 |

对于静态库，只需要一个包含了所有函数实现代码的.lib文件，.lib文件是比较大的（因为包含了所有的实现代码）

对于动态库，.dll只是在运行时才用得上，那么在link时，它根本不知道函数有没有实现，（注意.h文件只是一个申明，也许是一个假申明），所以它也需要一个.lib文件表明它实现了哪些函数（只是表明实现了，不需要具体的实现代码，具体的实现是在.dll中，所以.lib是很小的）

一个具体的例子，就是我们在assignment5中所用到的第三方库gmp

在windows平台上，我们除了提供gmp.h – 函数的申明

还提供了libgmp-10.dll – 动态库，函数的具体实现

此外，还要提供一个小的配合动态库的libgmp-10.lib，给调用者说明动态库中具体实现了哪些函数

（注意，为了区分真正的静态库，原则上我们不把这个小的.lib称为静态库，只是称为导出函数库，虽然它的后缀也是.lib）

关于gmp库的build

在数论当中，我们要用到大数（其长度远超过当前计算机的64位），gmp就是专门处理大数的一个专用第三方库，它是开源代码的，每个人都可以从它的网站上下到源代码，但是它不提供各个平台上的具体的库，所以就需要我们自己进行build

下面就讨论在Windows上的build（因为Unix和Linux通过package管理工具能够自动下载到）

|  |  |  |
| --- | --- | --- |
| 平台 | gmp |  |
| Unix-like | 包安装 | apt install |
| windows | 1，下载别人build好的 | 简单，快捷 |
| 2，自己build | 麻烦，但是会学到许多知识 |

gmp提供了在Unix-like (类Unix系统，包括Linux，Ubuntu都属于此类)的build，如果想在Windows上进行build，必须安装[MinGW – (Minimalist GNU for Windows)](https://www.baidu.com/link?url=P2K3dd2HS9j2GrIYawjFDKYn0mItXf-_Bt_MyWtPmwIz-Q-WiSCFmal0IPLH2ji46RG3Myrtdp9irrPpn2qsva&wd=&eqid=b8fd954e0011b0c6000000066367ad1c)

此外，还需要安装msys MSYS是一个小型的GNU环境,包括基本的bash,make等等

二者之间的关系见下面link:

https://blog.csdn.net/u012294613/article/details/126460773

事实上，在现在的许多MinGW的安装包中已经包含了msys，对于这类的MinGW，我们就不需要再重复安装msys了，直接可以用msys

win10+编译gmp大数库（亲测可用）

<https://blog.csdn.net/qq_45662588/article/details/122558604>

windows环境下的gmp大数运算库的配置（visual studio 2019 + gmp-6.2.0 + msys

https://blog.csdn.net/qq\_45662588/article/details/122560098

Unix 文件系统，文件描述符（一个小的非负整数）

特别的，文件描述符0，1，2分别赋给stdin, stdout, stderr

Unix使用< 和 > 重定向和管道 | 改变输入/输出的IO

open 返回一个文件描述符

fopen 返回一个文件指针

Hash函数很重要

Table

Description automatically generated

Text

Description automatically generated with low confidence