# 致未来：

来自2017年10月12日的自己

阅读文档是一个体系工程。不可能毫无重点地想要熟悉整个文档。所以，阅读文档的时候，应该注意：

1. 关注目录结构。
2. 有重点地进行查阅和扩展。
3. 文档中遇到不会的单词，应该即可查询，写在草稿纸上，待稍后整理为笔记。

# Introduction

1. There are a lot of functions in the GNU C  
   Library and it’s not realistic to expect that you will be able to remember exactly how to use each and every one of them. It’s more important to become generally familiar with the kinds of facilities that the library provides, so that when you are writing your programs you can recognize when to make use of library functions, and where in this manual you can find more specific information about them.
   1. GNU C Library中有很多的函数，想要准确地记住每一个是不现实的。重要的是要大致熟悉库中所提供的设备类型，这样当我们编程需要库函数的时候，就知道在手册的什么位置可以找到更多具体的信息
2. 文档P925有按字母排序的函数表
3. The header files and library facilities that make up the GNU C Library are a superset of those specified by the ISO C standard
   1. 所以glibc是标准C库的超集
4. The library facilities specified by the POSIX standards are a superset of those required  
   by ISO C;
   1. 所以POSIX也是ISO C的超集
5. POSIX.1 对ISO 的拓展包含：
   1. file system interface primitives (see Chapter 14 [File System Interface], page 390)
   2. device-specific terminal control functions (see Chapter 17 [Low-Level Terminal Interface], page 490)
   3. process control functions (see Chapter 26 [Processes], page 773).
6. POSIX 安全属性~~~暂时不能理解
7. Libraries for use by C programs really consist of two parts: *header files* that define types and macros and declare variables and functions; and the actual library or *archive* that contains the definitions of the variables and functions
   1. C库包含了两个部分：

头文件

实际定义函数的变量的库

1. The purpose of declarations is to allow the compiler to correctly process references to the declared variables and functions
2. A *definition*, on the other hand, actually allocates storage for a variable or says what a function does.
3. 函数的宏定义：并不影响函数的运行，只是用宏来取代实际的函数，程序运行起来会更快#define 函数名 可以取消函数的宏定义，但这一般只是会让程序更慢
4. Feature test macro(测试宏):The exact set of features available when you compile a source file is controlled by which *feature test macros* you define
   1. 不同的C标准有不同的特征，虽然一个标准往往是另一个标准的超集或子集，但是他们之间是相互不兼容的。 因此我们有必要定义测试宏来指定我们用的是哪一个标准的特征
   2. 定义测试宏有两个方法
      1. 在文本最开头用#define定义（具体定义那种类型见于文档）(这种方法比较好，让文件本身定义自己是用什么标准)
      2. 编译的时候用命令行参数指定，比如我一般制定的-std=c99
      3. 可以定义多个测试宏（比如我定义了一个标准，再定义多一个子标准，这样是没有影响的）
      4. 文档中推荐定义：\_GNU\_SOURCE。这样就包含了： ISO C89, ISO C99, POSIX.1,  
         POSIX.2, BSD, SVID, X/Open, LFS, and GNU extensions

# Error Reporting

## Checking for errors

1. 大多数的库函数都有一个返回值来报告程序是否成功执行。但是要知道程序出了什么错误，就需要用到volatile类型变量errno，这个变量定义在errno.h中
2. 但函数出错时，errno被赋予一个非0值（不同错误不同值）。所以在每次使用之前应该初始化error为0。
3. Error的值用宏来定义（见error codes）。

## Error Messages

1. 函数strerror()（包含与string.h） 和perror()（包含于stdio.h）可以根据发生的错误类型自动输出标准的text。
   1. 参数是errno(将checking for errors)，返回值是指向该text的char指针
2. 很多程序运行错误之后都会自动关闭，那么怎么知道是哪一个程序出错呢？
   1. Many programs that don’t read input from the terminal are designed to exit if any  
      system call fails. By convention, the error message from such a program should start with the program’s name, sans directories.。和终端缺乏交互的程序在报告错误的时候，应该在开头加上程序的名字
   2. 变量program\_invocation\_name和program\_invocation\_short\_name储存程序的名字和简写名字（不包含目录），定义在errno.h中
3. 还有很多报告程序错误的函数和变量，自行查阅吧~~~