编译步骤

编译型语言编程的步骤分为三个步骤：

编辑：使用编辑器编辑代码

编译：生成可执行文件

执行：直接执行可执行文件

其中，编译的步骤分为：

预编译：处理所有（#开头的）预处理语句

编译：将代码编译成汇编文件

汇编：将汇编文件编译成静态库

链接：将静态库链接成可执行文件

预编译语句补充

`__LINE__`、`__FILE__`、`__TIME__`：常用的debug宏，LINE宏用数字打印当前行号，FILE宏和TIME宏用字符串打印所在文件和编译时间。

#error ： 在指定位置报错

#line：修改`__LINE__`、`__FILE__`的提示信息

#pragma pack(4)：强制结构体向4字节对齐（4可以改为其他数字）

多文件项目

当我们的项目大到一定程度的时候，都放到一个文件会大大降低代码的可维护性。所以一般就有了多文件项目。

多文件项目会将每一个文件单独编译成静态库（即前三步），最后通过一个main函数链接成一个可执行文件。

头文件：可以自己定义头文件（.h文件），引用时用双引号：`#include "mytest.h"`。双引号也可以引用默认的头文件，即可以覆盖尖括号的内容，但通常会用双引号和尖括号区分自定义的头文件和系统提供的头文件。

头文件里一般只放声明

头文件为了避免重复引用，需要利用宏来保证引用的唯一性：

```
#ifndef _MYTEST_H_
#define _MYTEST_H_

//头文件内容

#endif //_MYTEST_H_
```

extern关键字：可以通过外部声明的方式，在头文件中声明全局变量和函数。函数声明可以省略extern。

static关键字：可以用static关键字修饰全局变量或者函数，限制该全局变量或者函数只能在本文件使用

main函数传参

main函数允许传最多三个参数：

```
#include <stdio.h>

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

}
```

参数1表示外部参数个数

参数2表示外部参数的每个参数的值

参数3表示环境变量

外部进行脚本调用时，可以通过调用语句后面跟以空格隔开的字符串的方式给main函数传参。

字节序

小端序：指低地址存放数据低位的存储方式，也叫主机字节序。

大端序：指高地址存放数据低位的存储方式，也叫网络字节序。

结构体位段的存储方式是先声明的位段放在较低位。

内存空间分类

栈（stack）空间：由系统自动分配自动释放，一般从上往下增长。大小一般较小，平台做题时以1M作为标准大小，实际使用时不会超过内存的1%。

堆（heap）空间：通过动态内存分配手动申请手动释放，一般从下往上增长。大小一般是总内存大小减去1G。

静态区：存放静态变量和全局变量，全局可见，理论大小无上限。

使用关键字**static**，可以将局部变量定义在静态区，使之成为一个只在局部可见的全局变量。

常量区：存放各种常量，如1、3.14、'a'这类的。

代码段：存放二进制代码。代码一般放在外存储器上，要读取到主存储器上进行执行，为了避免频繁的读取外存操作，故将代码一次读取完放到代码区进行执行。

另一种分类方式：

DATA区：存放可修改的数据，包括栈、堆、已初始化的全局变量。

BSS区：存放未初始化（或者初始化为0）的全局变量。

TEXT区：存放不可修改的数据，包括常量区和代码段。

缓冲区

stdout：输出缓冲区，每个文件都有一个，所有输出语句共用。

每当stdout中的内容输出到屏幕上时，会从缓冲区中清理掉这部分内容。

当前编译器默认清空原则：

1、一句打印语句的调用结束；

2、缓存已满；

3、程序结束。

手动清空语句：`fflush(NULL)`

stdin：输入缓冲区，每个文件都有一个，所有输入语句共用。

输入时，每一个字符会暂时存储在输入缓冲区，回车时提交。

每当stdin中的内容被指定的变量成功读取后，会从缓冲区中清理掉这部分内容。

%d处理方法：

1、先检查是不是不可见字符，如果是，跳过所有不可见字符。

2、先检查第一个字符是不是负号，如果是，置位标记并跳过处理。

3、如果是数字，将所有数字字符处理成对应数字，直到遇到非数字。如果不是数字，则直接判定失败跳出。

4、检查输出和负号标记，处理好数字给对应变量赋值，同时计算赋值成功的元素数量并返回。

当缓存中有异常字符出现，%d无法成功读取时，则%d读取失败，异常字符会依然留在缓冲区内。如果是循环读入，会出现死循环，可以用语句`while(getchar() != '\n');`处理。