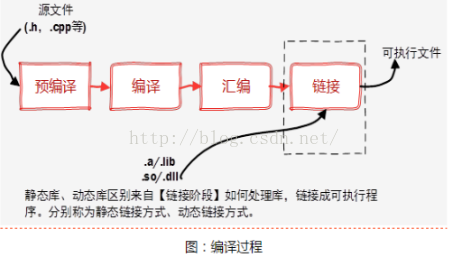
1. 生成可执行程序的过程：



1. 预编译：

处理#预编译指令：#include/#define/#if/#else/#endif…

1. 编译：

首先要检查代码的规范性、是否有语法错误等，以确定代码的实际要做的工作，在检查无误后，把代码翻译成汇编语言。

其中重要的一点，编译阶段确定了程序所需数据（常量、变量、数组等…）的内存分配方式（静态分配和动态分配）。

所有的类型都是由基本数据类型组成的，基本数据类型也是由确定的字节数确定的，因此，在编译阶段程序中所有类型的大小都已经确定。

静态数据区：静态内存空间，其中数据的总大小和初始值在编译时确定，数据在整个程序运行时一直存在；单独编译阶段就有了偏移地址和大小。

栈空间：自动内存空间，其中数据的大小在编译时确定，数据的分配和释放也由编译器在函数进入和退出时插入指令完成，数据生命周期和函数一样。

堆空间：动态（手动）内存空间（地址 + 大小），其中数据（数组）的大小和初始值在运行时确定，数据生命周期不定。

1. 汇编：

将汇编语言文件转为可重定位的二进制文件（目标文件）.o/.obj（每一个cpp对应一个.obj）。此类文件包含代码段、数据段、未解决符号表（在此编译单元中哪些符号的地址找不到，这些符号的实际内存在其他目标文件中）、导出符号表（此目标文件能为其他目标文件提供哪些符号的实例地址）。

实际上还有一个表：地址重定向表，记录了本编译单元内对自身地址的引用记录，大概内容：我这个编译单元中地址为0X0005的地方储存的地址需要重定位！只记录需要重定位的位置，不用使用符号来标识。因为在原本基础上加上目标文件在可执行文件中的首地址就可以实现重定位。

这里的符号（包括变量和函数）：就是编译器会产生的所谓的C++名称修饰，比如函数int Function1(char \*var1,unsigned long);被修饰为?Function@@YGXXZ

其中数据段主要存放静态变量。这就是为什么程序启动时静态变量就有了内存。

通常假如程序要引用了库，那么程序中#include库中的头文件，使用此头文件中的符号，对于使用者都应为未解决符号（除非库头文件中使用内联），需进一步链接。（静态库和动态库的链接方式不同）。

静态库可以理解为就是打包了一个目标文件.o/.obj的集合。与目标文件同级。

假如一个工程只是生成库，那么就没有下面链接的过程。可以理解为：只是将各个目标文件打包成集合。

1. 链接：

分为静态链接和动态链接。

静态链接（个人理解工程生成的目标文件和静态库.a/.lib的链接过程都是静态链接）：

\*链接过程：

首先决定各个目标文件在最终可执行文件里的位置。然后访问逐个所有目标文件的地址重定向表，对其中记录的地址进行重定向（即加上该编译单元实际在可执行文件里的起始地址），至此先解决了自身内部的链接问题。然后遍历所有目标文件的未解决符号表，并且在所有的导出符号表里查找匹配的符号，并在未解决符号表中所记录的位置上填写实际的地址（也要加上拥有该符号定义的编译单元实际在可执行文件里的起始地址）。最后把所有的目标文件中所需的内容复制到可执行文件的相应位置。一个可执行文件产生 。

\*静态链接之后，所需要的内容都被复制打包进可执行文件中。生成可执行文件后，程序就与所有的.o/.obj文件或者静态库文件没有瓜葛。但是使得可执行文件相对很大。这是与动态链接的主要区别。

\*C++程序在执行静态链接时，如果静态库里（或者工程自己的目标文件）的某些方法没有任何地方调用到，最终这些没有被调用到的方法或变量可能将会被丢弃掉，不会被链接到可执行程序中。这样做大大减小生成可执行程序的体积。（要看编译器的行为，有时需要手动设置）。

动态链接（动态库的链接）：

5、目标文件和重定位：

<https://blog.csdn.net/u014774781/article/details/48088221>

1. 静态库（.a/.lib）:
2. 动态库（.so/.dll）: