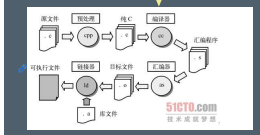
代码到可执行文件

GCC示例



用户编写

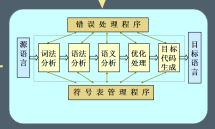
源代码 —预处理 -E→

1. 宏定义指令，如# define Name TokenString,#undef等
2. 条件编译指令，如#ifdef,#ifndef,#else,#elif,#endif,等
3. 头文件包含指令
4. 特殊符号，预编译程序可以识别一些特殊的符号

结果

源语言完整代码

编译：Compile， -S



结果

汇编指令

汇编,-C

翻译成机器语言[二进制]

链接，-O

解决引用

解决内部引用

1. 决定各个文件在可执行文件中位置
2. 对各个文件中的地址重定向表重新计算，得到在可执行文件中的地址表

解决外部引用

3. 遍历所有未解决符号表和导出符号表， 将在未解决符号表中填写在导出符号表里查询到的位置。
4. 将各个文件内容写入到可执行文件中

结果

可执行文件

链接方式

动静，内外

外部

引用地址在导出符号表中

内部

引用地址在重定向地址表中

静态

将代码直接拷贝到程序

动态

将引用的代码放入某个动态链接库的目标文件中
向程序提供需要调用内容地址，程序间接引用

额外的结果

未解决符号表、导出符号表、地址重定向表

未解决符号表

提供了所有在该编译单元里引用
但是定义并不在本编译单元里的符号及其出现的地址

导出符号表

提供了本编译单元具有定义，并且愿意提供给其他编译单元使用的符号及其地址

地址重定向表

提供了本编译单元所有对自身地址的引用的记录

常见问题与语言特性

语言规范

内部与外部

static： 表示当前函数或变量是内部链接， 不导出

内部

外部

extern：这个符号在其他单元定义
需要放到未解决符号表中

默认

1. 函数变量， 默认外部
2. const 变量， 默认内部

函数

1. 函数代码容易变化， 若设置为内部链接
则会被定义在头文件中， 这样每次函数代码变化
包含该头文件的单元都需要重新编译
2. 普通变量和函数若是为内部，则不同文件可以维持不同代码，造成混乱。

const 变量

const 变量只读，不会改变，没有重新编译的代价
取地址和mutual修改变量时仍会出现错误。

内联函数

编译时在使用的地方直接展开【类似于宏】， 不会写入到导出符号表或是导出表， 也不会写入到地址重定向表中，其实没有内部或是外部属性

普通函数

普通函数定义的形式在编译时就是具象化的具体的，可以在编译时直接被编译成二进制代码，产生导出符号。

模板函数： 链接使用时确认

模板函数在定义可见的情况下才能具象化， 成为“具体”的函数。

特殊处理：内部链接

1. 作为外部链接，只见声明不见定义，定义模板的部分无法被直接调用，那么模板便无法具象化，无法编译为二进制代码， 在链接时被寻址到也无法使用。成为无法解析的外部符号。
2. 作为内部链接， 定义可见，那么在编译时便可具象化，被编译成二进制代码， 链接时在内部可以被寻找到并成功调用。

问题

是否正确引用（外部）

未能找到引用

unresolved external link
未能在导出表里找到对应项

重复引用

duplicated external simbols
导出表里出现重复选项

头文件包含定义

1. 头文件若是包含外部定义，则会造成导出表里有多个同样的符号
即表现为： duplicated external simbols
2. 若想要定义，则定义内部链接。

类的静态成员不可以就地初始化

类通常定义在头文件， 若是就地初始化则会出现在头文件中定义外部链接的情况。