信息安全基础综合设计实验

复习

课程内容

信息安全基础综合实验

数论基础	伪随机数生成器	对称密码	非对称密码
模指数运算	LCG	RC4	大数运算
素性检测	BBS	DES	RSA加密
乘法逆元	ANSI	SHA-1	RSA签名

Linux简单编程

复习提纲: Linux简单编程

- 编译环境: g++
  - 编译指令: **g++ -o {可执行文件名} {源程序文件名}**
- Makefile文件编写 **(不考)**
  - **Makefile规则**

```
<target>:  <prerequisites>
[tab]      <commands>
```

  - **递归**编译

复习提纲: 数论基础

- 模指数运算
  - ➡ 掌握模指数运算的**分治算法**, 及其**编程实现**
- 素性测试
  - ➡ 掌握**Eratosthenes筛选法**和**Miller-Rabin算法**, 及其**编程实现**
- 乘法逆元
  - ➡ 掌握**扩展欧几里得算法**, 及其**编程实现**

复习提纲: 伪随机数产生器

- 伪随机数概念: **与随机数不可区分, 可重现**
- 线性同余伪随机数生成器
  - ➡ **基本概念**: 递推公式, 参数选择, 周期性,…
  - ➡ 熟练线性同余伪随机数生成器的**编程实现**
- BBS伪随机数生成器
  - ➡ **基本概念**: 递推公式, 参数, 安全性,…
  - ➡ 熟练BBS伪随机数生成器的**编程实现**
- ANSI X9.17伪随机数生成器: 熟练**编程实现**

复习提纲: OpenSSL基础

- 编译: g++ -o {可执行文件名} {源程序文件名} **-lcrypto**
- 大数运算库
  - 头文件: openssl/bn.h
  - 初始化: BN_init()或BN_new()
  - 回收: BN_free()
  - 操作: BN_mod_exp(), BN_mod_inverse(),…
  - 应用: **基于OpenSSL实现数论基础实验和伪随机数产生器实验**

复习提纲: 对称密码

- RC4: 流密码及其特征, 宏观流程,…
- DES: 分组密码及其特征, Feistel网络, 宏观流程,…
  - **基于OpenSSL编程实现:** 头文件openssl/des.h, DES_cblock结构, DES_set_key_checked()和DES_ecb_encrypt()
- SHA-1: 哈希函数及其特征, SHA-1长度, 宏观流程,…
  - **基于OpenSSL编程实现**: 头文件openssl/sha.h, SHA1()

复习提纲: 非对称密码

- RSA算法: 参数, 安全性,…
- RSA加密: 流程
  - ➡ **基于OpenSSL编程实现:** 头文件openssl/rsa.h, RSA结构, RSA_generate_key(), RSA_public_encrypt(), RSA_private_decrypt()
- RSA签名: 数字签名及特征,…
  - ➡ **基于OpenSSL编程实现:** 头文件openssl/rsa.h, RSA结构, RSA_generate_key(), RSA_private_encrypt(), RSA_public_decrypt()

编程参考函数

- 内存处理函数: memcpy(), memset(), memcmp()
  - ➡ 依赖头文件**string.h**

- BIGNUM处理函数: BN_hex2bn(), BN_copy(), BN_print_fp(),…

- 万能查询方法: **man {函数名}**
  - ➡ 当函数名记忆不完整时, 输入函数名前缀后**<tab>补全**

试卷结构

- **选择题 (20%: 2 * 10):** 考察对**基本概念**的理解
- **填空题 (10%: 2 * 5):** 考察对**基本概念**的理解
- **论述题 (20%: 5 + 5 + 10):** 考察对**基本概念**的理解
- **计算题 (10%: 5 * 2)**: 考察对**信息安全算法**的理解
- **编程题 (40%: 20 * 2)**: 考察**编程**能力, **查阅**参考函数能力, 灵活使用**OpenSSL**能力

Linux编程: Library的使用

Library

- **Library:** **编译好**的程序组件，可被不同应用程序**复用**，但又**独立**于应用程序本身的代码
  - ➡ C标准库 (例如stdio.h, math.h, stdlib.h,…), C++ STL,…
  - ➡ 优点: 共享程序组件, 发布API接口

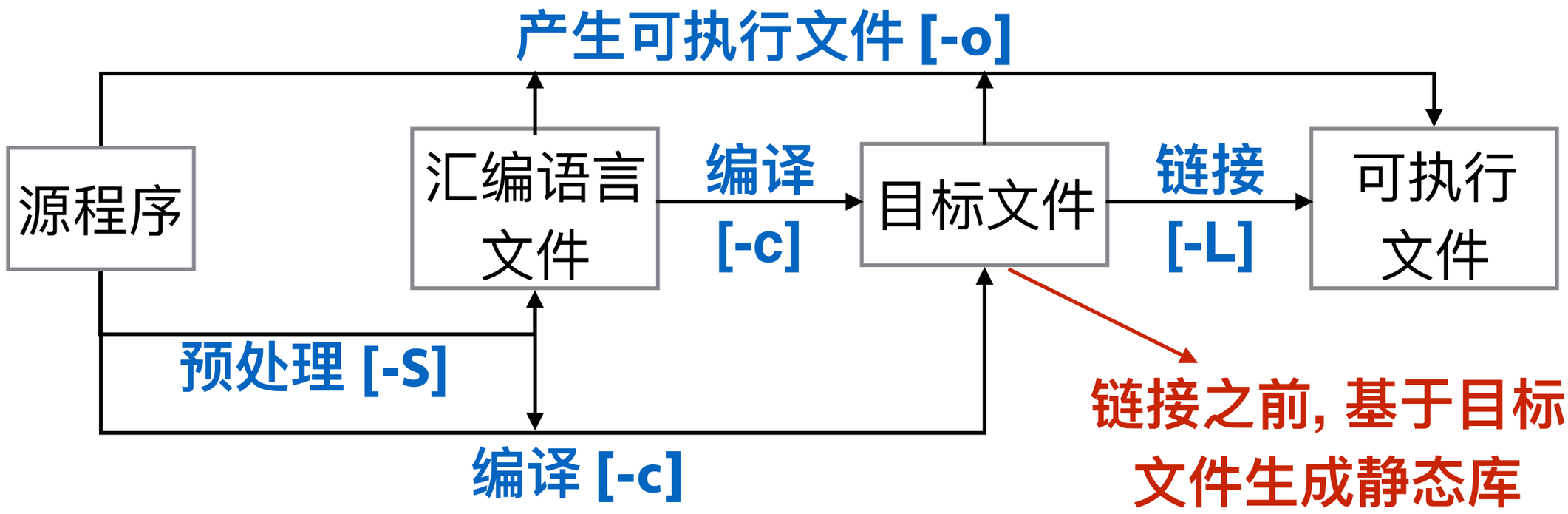
- Linux中的library类型: **动态库 (*.so)** 和**静态库 (*.a)**
  - ➡ 静态库: **链接阶段**与目标文件链接形成可执行文件
  - ➡ 动态库: 程序**运行时**动态加载

Library命名规则

- Library命名以**“lib”**开头, 编译时通过参数**“-l{library名}”**指示需加载的library
  - ➡ 例子: 命令**“gcc -o app src-file.c -lm -lpthread”**为src-file.c加载/usr/lib/XXXX-linux-gnu/lib**m**.so (数学库)和/usr/lib/XXXX-linux-gnu/lib**pthread**.so (线程库)
  - ➡ gcc隐式链接**“-lc”(加载/usr/lib/XXXX-linux-gnu/libc.so)**, g++隐式链接**“-lstdc++”(加载/usr/lib/XXXX-linux-gnu/libstdc++.so)**
    - ✳ 标准输入/输出不需要链接library
  - ➡ 使用OpenSSL须显示链接**“-lcrypto”(加载/usr/lib/XXXX-linux-gnu/libcrypto.so)**

静态库

- 静态库生成阶段



- 静态库生成命令

```
# 编译产生.o目标文件
ar -r {静态库名}.a {目标文件名}.o
```

Library搜索路径

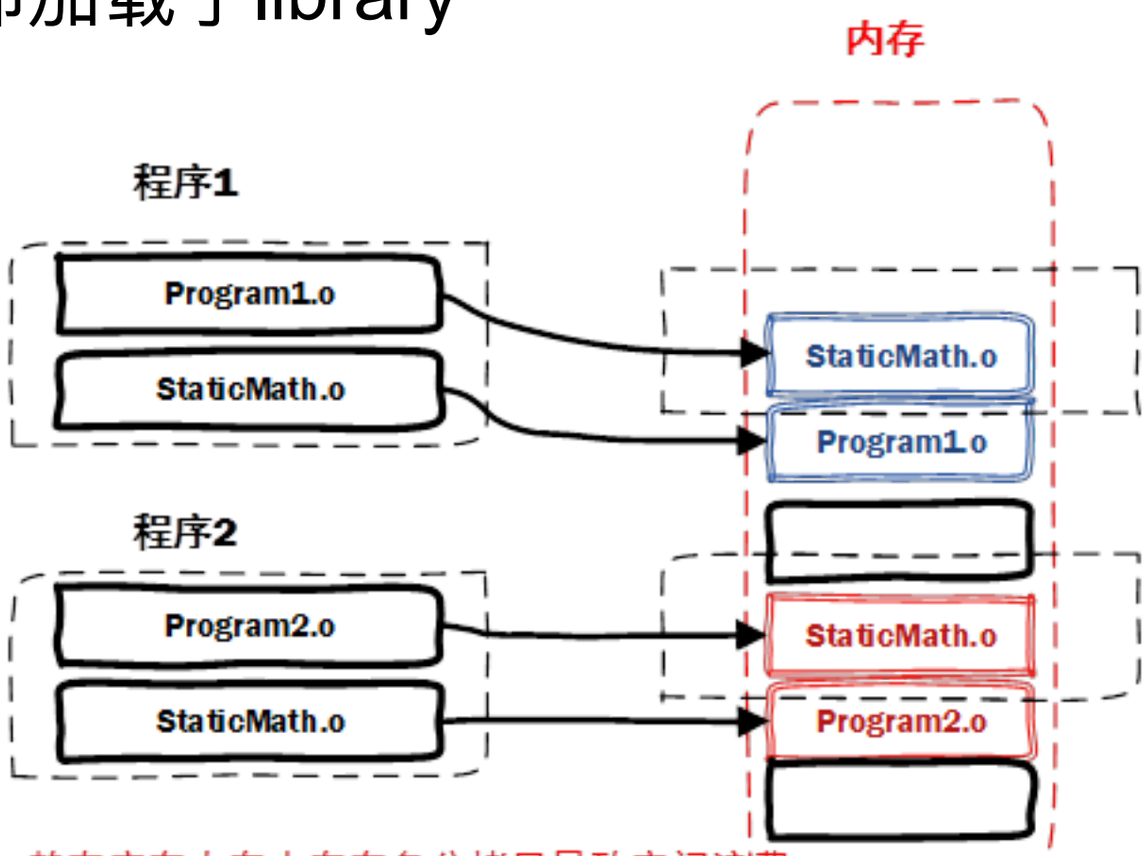
- 存放在**库搜索路径** (例如/usr/lib/等) 下的library才能被编译程序找到
  - ➡ 若library未存放在库搜索路径下, 须加入参数**“-L{library路径}”**显式指定library位置

- 存放在**头文件搜索路径**下的library头文件才能被编译程序找到
  - ➡ 若library头文件未存放在头文件搜索路径下, 须加入参数**“-I{library头文件路径}”**显式指定头文件位置

最终编译指令: g++ -o {可执行文件} {源程序} [-I库头文件路径] [-L库路径] -l{库名}

静态库的缺点

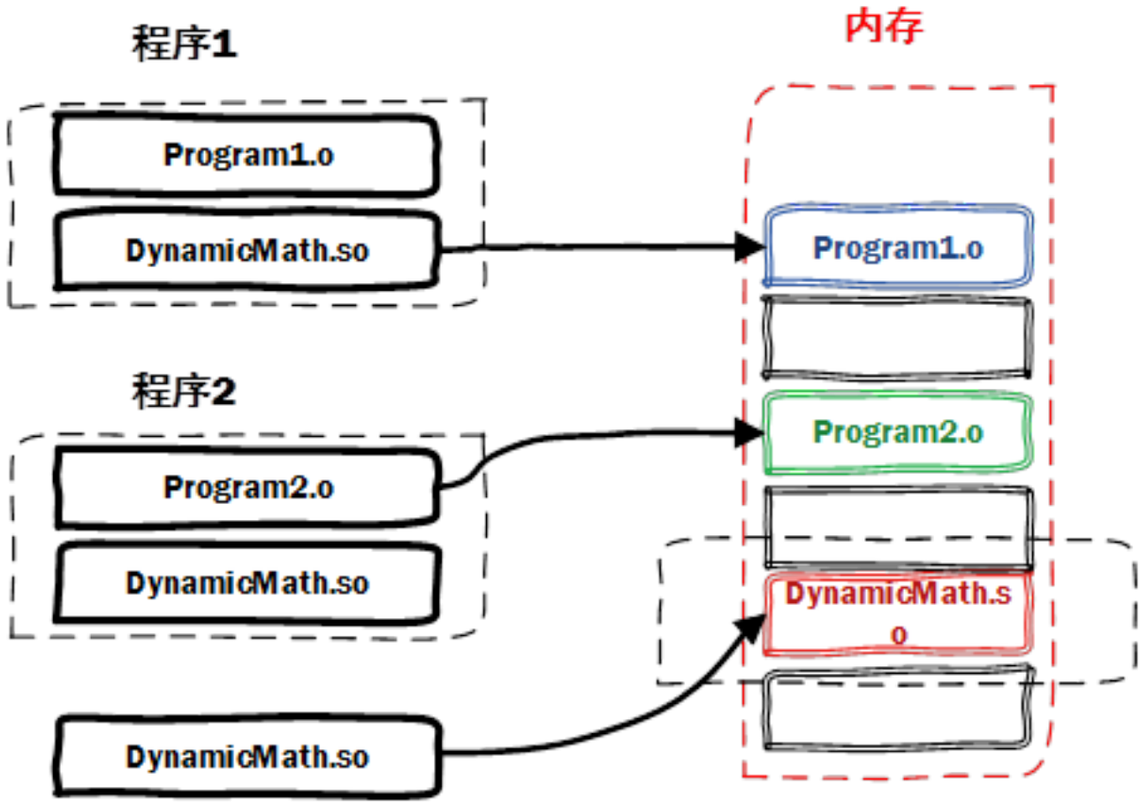
- 产生的可执行文件较大
  - ➡ 链接时, 即加载了library



静态库在内存中存在多分拷贝导致空间浪费。
假如，静态库占用**1M**内存，有**2000**个这样的程序，将占用近**2GB**的空间~~~~~

动态库特点

- 运行时加载
  - 多个(运行的)程序**共享**, 节省内存空间



动态库在内存中只存在一份拷贝，避免了静态库浪费空间的问题。

动态库

- 生成目标文件: g++ **-fPIC** -c {源程序}
  - ➡ **-fPIC**: 创建与地址无关的编译程序, 利于共享
- 链接生成动态库: g++ **-shared** -o lib{库名}.so {目标文件}
  - ➡ **-shared**: 指定生成动态链接库
  - ➡ 两条命令合并: g++ **-fPIC -shared** -o lib{库名}.so {源程序}

动态库的使用

- 编译: g++ -o {可执行文件} {源程序} [-I库头文件路径] [-L库路径] -l{库名}
  - ➡ 执行时报错: **error while loading shared libraries**
    - ✳ 动态链接器找不到动态库
  - ➡ **解决方法: 在环境变量LD_LIBRARY_PATH中加入动态库路径**

```
export LD_LIBRARY_PATH = $LD_LIBRARY_PATH:{动态库路径}
```

    - ✳ 放入**~/.bash_profile**以自动启动
  - ➡ 解决方法: 将动态库放入搜索路径(/lib/, /usr/lib/)中…