

# 信息安全基础综合设计实验

李经纬

# 课程信息

# 自我介绍

- ❖ 姓名: 李经纬
- ❖ 职称: 副教授
- ❖ 研究领域: 信息安全
- ❖ 联系方式: [lijw1987@gmail.com](mailto:lijw1987@gmail.com)
- ❖ 主页: <https://jingwei87.github.io/>

# 课程目标

## ❖ 前置课程

- ✓ “计算机算法与程序设计”：C/C++语言
- ✓ “信息安全数学基础”：数论
- ✓ “密码学”：加密技术, 哈希函数, 数字签名

## ❖ 目标: 建立综合运用密码学和数论基础知识的能力, 以及分析和解决问题的能力

# 课程内容

## ❖ 四个部分

### 信息安全基础综合实验



➔ 鼓励学有余力的同学尝试更多实验内容 (例如[扩展阅读部分](#))

➔ 参考教材: **“信息安全基础综合实验教程”**, 程红蓉等编著, 高等教育出版社, 2012. 02

# 考核方式

- ❖ 平时成绩 **(30%)**: 课堂实验 + 课后实验
  - ➔ 每次课均有课堂实验, **当堂检查+录入分数**, 未能完成, 成绩折扣
  - ➔ 一次课后实验, **期末考试前**提交, 提交**伪代码+源代码+实验报告**
- ❖ 期末成绩 **(70%)**: 程序+试卷
- ❖ 教学辅助 **(+5%)**: 收集作业, 发布通知, 帮助解答问题

# 课堂纪律

- ❖ 平时作业有20%以上未按时完成
- ❖ 实验课缺课学时达到总学时20%以上
- ❖ 无正当理由或未经学院同意, 课堂缺课学时达到总学时的40%以上

以上情况之一, **取消**考试资格 (校教通[2005]106号)

# Linux简单编程



# 编程环境

- ❖ **集成开发环境**隐藏了内部编译, 运行, 调试, 链接等过程, 不利于深度理解和学习
- ❖ 为什么学习Linux
  - ➔ 应用广泛: Android, ChromeOS, 企业服务器
  - ➔ 哲学: 一个程序只实现一个功能, 多个简单程序组合完成复杂任务
- ❖ 编程: **vim/gedit** (编辑器), **g++** (编译器), **gdb** (调试器)
  - ➔ Linux虚拟机环境: virtualbox (VMware) + ubuntu镜像

# VIM简介

- ❖ 誉为编辑器之神, 具备无穷扩展性
- ❖ 三种常用模式:
  - ➔ 普通模式: “Esc”切换, 用于移动光标, 复制粘贴, 查找替换等
  - ➔ 插入模式: 普通模式下“i”或“a”切换, 用于文字编辑
  - ➔ 命令模式: 普通模式下“:”切换, 用于执行命令
- ❖ 练习
  - ➔ “:e {文件名}”: 打开文件
  - ➔ “:wq”: 保存并退出当前文件

# VIM简介：光标移动

- ❖ 仅适用于**普通模式**
- ❖ 基本移动
  - ➔ 上下左右: **“k, j, h, l”** (方向键)
  - ➔ 翻页: **“Ctrl-b”** (Page-Up), **“Ctrl-f”** (Page-Down)
- ❖ 进阶移动
  - ➔ 行移动: **“0”** (行首); **“\$”** (行尾); **“{行号}+G”** (至对应行)
  - ➔ 文件移动: **“gg”** (文件起始); **“G”** (文件末尾)
  - ➔ 单词移动: **“e”** (至单词末尾); **“b”** (至单词起始)

# VIM简介：文件编辑

## ❖ 普通模式进入编辑模式

- ➔ **“i”**: 在光标所在字符**前**开始输入
- ➔ **“a”**: 在光标所在字符**后**开始输入
- ➔ **“o”**: 在光标所在行的**下一行**开始输入

## ❖ 删除(适用普通模式)

- ➔ **“x”**: 删除光标所在字符
- ➔ **“dd”**: 删除光标所在行

# VIM进阶

version 1.1  
April 1st, 05

## vi / vim graphical cheat sheet

**Esc**  
normal mode

~ toggle case	! external filter	@ play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	& repeat is	* next ident	( begin sentence	) end sentence	_ "soft" bol down	+ next line
\ goto mark	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	= auto format
Q ex mode	W next WORD	E end WORD	R replace mode	T back till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	}	end parag.
q record macro	w next word	e end word	r replace char	t till	y yank	u undo	i insert mode	o open below	p paste after	[ misc	]	misc
A append at col	S subst line	D delete to col	F "back" find ch	G eof/ goto ln	H screen top	J join lines	K help	L screen bottom	. ex cmd line	" reg. spec	bol/ goto col	
a append	s subst char	d delete	f find char	g extra ends	h ←	j ↓	k ↑	l →	: repeat	' goto mk. bol	\ not used!	
Z quit	X back-space	C change to col	V visual lines	B prev WORD	N prev (find)	M screen mid!	< un-indent	> indent	? find (rev.)			
Z extra ends	x delete char	c change	v visual mode	b prev word	n next (find)	m set mark	> reverse	. repeat cmd	/ find			

**Legend:**

- motion** moves the cursor, or defines the range for an operator
- command** direct action command, if **red**, it enters insert mode
- operator** requires a motion afterwards, operates between cursor & destination
- extra** special functions, requires extra input

**Q** commands with a dot need a char argument afterwards

bol = beginning of line, col = end of line, mk = mark, yank = copy

words: `quux{foo, bar, baz}`  
WORDS: `quux{foo, bar, baz}`

**Main command line commands ('ex'):**  
 :w (save), :q (quit), :q! (quit w/o saving)  
 :e f (open file f),  
 :%s/x/y/g (replace 'x' by 'y' filewide),  
 :h (help in vim), :mew (new file in vim).

**Other important commands:**  
 CTRL-R: redo (vim),  
 CTRL-F/-B: page up/down,  
 CTRL-K/-Y: scroll line up/down,  
 CTRL-V: block-visual mode (vim only)

**Visual mode:**  
 Move around and type operator to act on selected region (vim only)

**Notes:**

- (1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z, ") (e.g.: "av\$ to copy rest of line to reg 'a')
- (2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)
- (3) duplicate operator to act on current line (dd = delete line, >> = indent line)
- (4) ZZ to save & quit, ZQ to quit w/o saving
- (5) zt: scroll cursor to top, zb: bottom, zz: center
- (6) gg: top of file (vim only), gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to [www.viemu.com](http://www.viemu.com) - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

❖ **vimtutor**: vim学习教程

# Hello World

## ❖ 编写Hello World程序

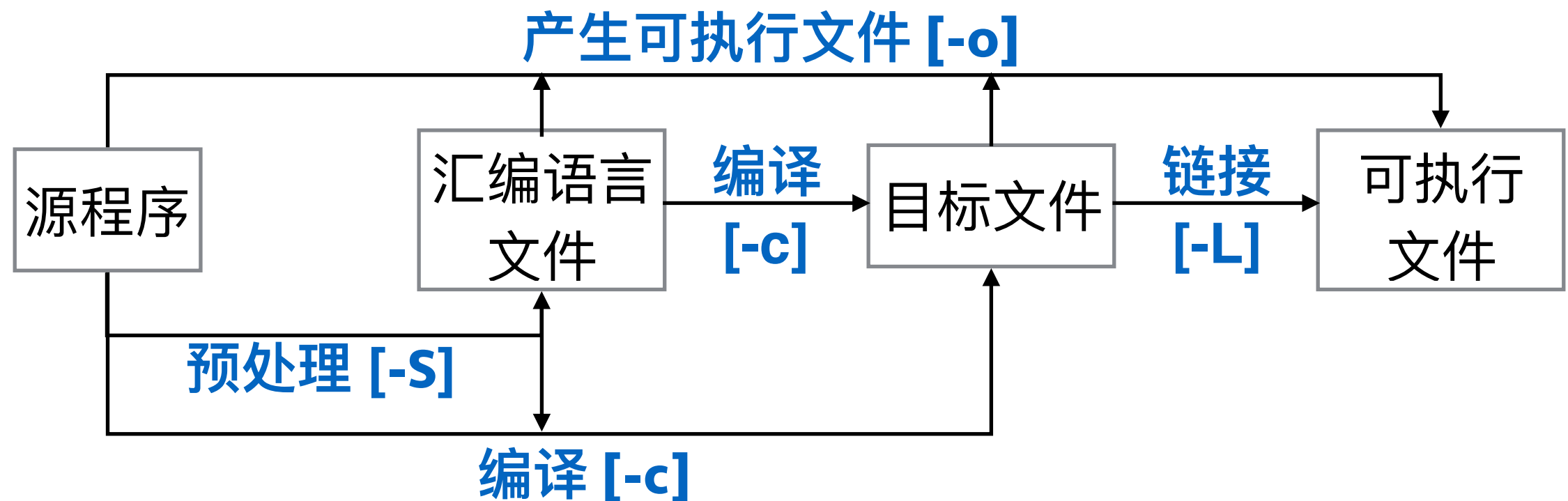
```
1 #include <stdio.h>
2
3 int main (int argc, char *argv[]) {
4     printf("hello world");
5     return 0;
6 }
```

## ❖ 编译(终端内执行): **g++ -o {可执行文件} {源程序}**

```
1 g++ -o hello hello.cc
```

# 编译过程

❖ 简单编译指令: `g++ -o {可执行文件} {源程序}`



➔ “-o”参数封装了预处理、编译、链接过程

❖ 运行程序

```
1 ./hello
```

# Makefile

## ❖ Linux程序编译复杂

- ➔ 耗力: 每个小模块都需要指令来编译
- ➔ 耗时: 修改任意部分导致整个程序重新编译

## ❖ **Makefile**由一系列规则构成, 指明编译顺序

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

- ➔ 省力: **make**编译产生可执行文件
- ➔ 省时: 通过源文件修改时间来判断哪些规则需要重新编译



# Makefile示例I



```
1 hello: hello.cc
2     g++ -o hello hello.cc
3 clean:
4     rm hello
```

- ❖ **make**: 编译产生可执行文件hello
- ❖ **make clean**: 删除可执行文件hello

# Makefile示例II

❖ 三个文件: **number.hh**, **number.cc**, **makefile**

➔ number.hh定义Number类, 及其接口

➔ number.cc实现Number类接口函数

➔ makefile编译/清理工具

I

```
make触发第一条规则
1 SimpleCrypt: number.o
2     g++ -o SimpleCrypt number.o
3 number.o: number.cc
4     g++ -c -o number.o number.cc
5 clean:
6     rm -f *.o
7     rm -f SimpleCrypt
```

make clean触发

# Makefile示例III

## ❖ Program结构

- ➔ src/: 保存源程序文件(\*.h和\*.cc), 主函数所在源文件名须命名为main.cc
- ➔ lib/: 保存动态库和静态库(\*.a和\*.so)

## ❖ Makefile操作

- ➔ make: 编译产生可执行文件; 临时输出文件保存在tmp/
- ➔ make clean: 清空可执行文件和临时输出文件

```
1 # NOTE: ensure main function is included in main.cc
2 EXEC_TARGET = exe
3
4 # compiler & flags
5 CXX = g++
6 CPPFLAGS = -g -O3 -Wall -fno-operator-names -std=c++1y
7 LDFLAGS = -lcrypto -pthread
8
9 # directory structure
10 BUILD_DIR = ./build
11 SRC_DIR = ./src
12 LIB_DIR = ./lib
13 OUTPUT_DIR = ./tmp
14
15 # include & libraries
16 INC_DIRS := $(shell find $(LIB_DIR) -type d -and -name *include)
17 INC_LIBS := $(shell find $(LIB_DIR) -name *.a)
18 INC_LIBS := $(dir $(INC_LIBS))
19 INC_FLAGS := $(addprefix -I,$(INC_LIBS))
20 INC_FLAGS += $(addprefix -I,$(INC_DIRS))
21
22 SRCS := $(shell find $(SRC_DIR) -name *.cpp -or -name *.c -or -name *.cc)
23
24 SRCS_TARGET := $(filter-out %_unittest.cpp %_unittest.c %_unittest.cc,
25 $(SRCS))
26
27 OBJ_TARGET := $(SRCS_TARGET:$(SRC_DIR)/%=$(BUILD_DIR)/%.o)
28
29 SRC_TEST := $(filter-out %main.cpp %main.c %main.cc, $(SRCS))
30 OBJ_TEST := $(SRC_TEST:$(SRC_DIR)/%=$(BUILD_DIR)/%.o)
31
32 all: $(EXEC_TARGET)
33
34 $(EXEC_TARGET): $(OBJ_TARGET)
35     $(CXX) -o $@ $^ $(INC_FLAGS) $(LDFLAGS)
36
37 # C++ SOURCE
38 $(BUILD_DIR)/%.cc.o: $(SRC_DIR)/%.cc
39     $(CXX) $(CPPFLAGS) -c $< -o $@ $(INC_FLAGS) $(LDFLAGS)
40
41 .PHONY: all clean
42
43 clean:
44     $(RM) -r $(BUILD_DIR)
45     $(RM) -r $(OUTPUT_DIR)
46     $(RM) $(EXEC_TARGET) $(EXEC_TEST) $(LOG)
47
48 -include $(DEPS)
49
50 MKDIR_P = mkdir -p
```

# 扩展学习

- ❖ 无插件VIM编程技巧
- ❖ C程序编译过程解析
- ❖ A Simple Makefile Tutorial
- ❖ 使用GDB调试程序 (1, 2, 3, 4, 5, 6, 7)
- ❖ The UNIX Time-sharing System (需操作系统基础)

# Linux编程练习

## ❖ 实现**bitsToInteger**函数和**integerToBits**函数



```
1 int bitsToInteger(unsigned char *bitStr, int length)
2 // bitStr: 输入二进制字符串(例如"100101")
3 // length: 输入二进制字符串的比特位数(例如6)
4 // 返回: bitStr对应的十进制整数
```



```
1 unsigned char *integerToBits(int value, int *pLength)
2 // value: 输入十进制整数value
3 // pLength: 输入指向int型数据的指针(已在程序外为其分配了内存空间)
4 // 返回: value对应二进制字符串, 且pLength指向地址保存字符串比特长度
```