南京邮电大学

实 验 报 告

（2025/ 2026 学年 第 二 学期）

课程名称	计算机系统基础I
实验名称	数据表示实验
实验时间	2026 年 6 月 4 日
指导单位	基础教学中心
指导教师	

学生姓名		班级学号	
学院(系)	计算机学院	专　业	软件工程

实 验 报 告

实验名称	炸弹实验			指导教师	
实验类型	设计	实验学时	2	实验时间	

一、 实验目的和要求

实验目的：

通过逆向分析二进制程序（称为“炸弹程序”）的构成和运行逻辑，加深对理论课程中关于程序的机器级表示各方面知识点的理解，以及反汇编、跟踪/分析/调试等常用技
能的掌握。

实验要求

作为实验目标的二进制炸弹“bombs”可执行程序包含了以下 7 个阶段，分别考察对程序的机器级表示以及下面各方面知识点的理解和掌握：

阶段 1：字符串比较（必做）
阶段 2：循环语句（必做）
阶段 3：选择/分支语句（必做）
阶段 4：递归调用（选做）
阶段 5：数组结构（选做）
阶段 6：链表/指针/结构（选做）
阶段 7：递归函数/指针/链（选做）

在每个阶段程序要求输入一个特定字符串，如果输入满足程序代码所指定的字符串，则这个阶段的炸弹被拆除，该阶段即通过，否则炸弹会爆炸，屏幕显示“BOOM!!!”。

为完成炸弹的拆除任务，需要通过反汇编和分析/跟踪（使用 GDB 调试器等工具）程序
每一阶段的机器代码，从中定位和理解程序的主要执行逻辑（关键指令、控制结构等）
和相关数据变量，进而推断拆除炸弹所需的目标字符串。

二、实验环境(实验设备)

硬件：微型计算机

软件：linux 操作系统、Gcc编译套件

三、实验代码及结果

中文五号宋体，英文五号 Times new roman 字体，1.25 倍行距

说明：这部分内容主要包括：

1.截屏实验运行结果，即运行./bomb 学号.txt 结果。

2.截屏显示记分牌上自己的实验成绩。

3.写出各个阶段破译思路和过程。

1、实验结果

```
B23170106@ICS:~/bomb1374$ ./bomb ~/B23170106.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
^CSo you think you can stop the bomb with ctrl-c, do you?
Well...OK. :-)
B23170106@ICS:~/bomb1374$ ./bomb ~/B23170106.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
```

2、实验成绩（boom1374）

Bomb Lab Scoreboard

This page contains the latest information that we have received from your bomb. If your solution is marked **invalid**, this means your bomb reported a solution that didn't actually defuse your bomb.

Last updated: Sat Jun 13 21:33:45 2026 (updated every 30 secs)

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb3	Sun Apr 12 02:18	7	0	70	valid
2	bomb30	Mon Apr 13 21:19	7	0	70	valid
3	bomb95	Thu Apr 16 12:22	7	0	70	valid

四、实验小结（包括问题和解决方法、心得体会、意见与建议等）

中文五号宋体，英文五号Times new roman字体，1.25倍行距

说明：这部分内容主要包括：在编程、调试或测试过程中遇到的问题及解决方法、本次实验的心得体会、进一步改进的设想等。

(一)实验中遇到的主要问题及解决方法

1. bitMask 函数的移位边界问题

在实现 bitMask 时，第一版写法是 (~0) << (highbit + 1)，当 highbit == 31 时会触发 << 32，属于 C 标准的未定义行为，btest 在边界输入上给出了错误结果。解决方法是将单次移位拆成两次：先 << highbit，再 << 1，每次移位都不超过 31 位，行为完全可预测。这让我意识到位运算并不是纯数学，底层的硬件指令（SHL/SAR）有自己的约束边界。

2. isAsciiDigit 的 TMin 边界 bug

最初用 (x + ~0x2F) >> 31 检查 x >= 0x30 的符号位，btest 在 x = 0x80000000（TMin）时报错。原因是 TMin 加 ~0x2F 在 32 位补码里环绕到正数区，符号位翻成 0，误判为合法 ASCII 数字。改用「高半字节是否等于 3」+ 「低半字节是否小于 10」的拆字节方案后，避开了所有算术溢出，btest 通过所有测试用例。

3. floatFloat2Int 遗漏 NaN 和 ±∞ 的特殊处理

最初没有单独判断 exp == 255 的情况，btest 对 NaN 输入报错。回头看 IEEE 754 才注意到阶码全 1（exp == 0xFF）同时代表 ±∞ 和 NaN，真实阶 E = 128 >= 31，刚好被「超出 int 范围」这一条统一捕获，不需要单独写分支。这个规律是 IEEE 754 标准设计层面的内在一致性，理解它之后代码反而变简单了。

（二）实验心得

这次 DataLab 是第一次在比特级别上直接操作数据，整体下来有几个真实的收获。

最深的一个是理解了补码「取反加一」不只是口诀，而是由「加法器需要同时处理正负数」这一工程约束推导出来的必然结果。negate、absVal 的正确写法，以及 absVal 的(x + mask) ^ mask 这个不带分支的绝对值公式，都从这一性质自然推出来。

浮点数部分理解了 IEEE 754 的「字段分工」之后，floatScale2 和 floatFloat2Int 的逻辑几乎是逐字段翻译成代码。之前觉得浮点数很神秘，这次拆开看，符号、阶码、尾数三个字段各自管各自的事，思路清晰了很多。顺带也明白了 JavaScript 里 0.1 + 0.2 !== 0.3 不是 bug，而是二进制下 0.1 本身就是无限循环小数，IEEE 754 在有限位数里做了舍入。

另一个收获是「用位掩码代替分支」的思维方式。题目禁止 if，逼着我去想怎么用算术表达式承载控制流：x >> 31 得到全 0 或全 1 的掩码，再用 & 和 ^ 做条件选择。这种技巧在性能敏感的代码里很常用，现在至少能看懂编译器做分支消除时在干什么。

（三）意见与建议（没有可省略）

无

五.支撑毕业要求指标点
六、指导教师评语 (含学生能力达成度的评价)

成 绩		批阅人		日 期	

评分细则	评分项	优秀	良好	中等	合格	不合格
	遵守实验室规章制度					
	学习态度					
	算法思想准备情况					
	程序设计能力					
	解决问题能力					
	课题功能实现情况					
	算法设计合理性					
	算法效能评价					
	回答问题准确度					
	报告书写认真程度					
	内容详实程度					
	文字表达熟练程度					

	其它评价意见					
	本次实验能力达成评价（总成绩）					