華中科技大學

课 程 实 验 报 告

课程名称： 计算机系统基础

实验名称： 数据表示和等效运算

院　　系 ： 计算机科学与技术

专业班级 ： CS2304

学　　号 ： U202315653

姓　　名 ： 郝依凝

指导教师 ： 金良海

2025 年 3 月 13 日

一、实验目的与要求

⑴ 熟练掌握程序开发平台(VS2019/VS2022/VS2023) 的基本用法，包括程序的编译、链接和调试；

(2) 熟悉数据的表示形式；

(3) 熟悉地址的计算方法、地址的内存转换；

(4) 用常见的按位操作（移位、按位与/或/非/异或）等实现运算表达式的等效运算。

二、实验内容

任务 1　数据存放的压缩与解压编程

定义了 结构 student ，以及结构数组变量 old_s[N], new_s[N];　(N=5)

```
struct student {
    char    name[8];
    short    age;
    float    score;
    char    remark[200];   // 备注信息
};
```

编写程序，将 old_s[N] 中的所有信息依次紧凑(压缩)存放到一个字符数组 message 中，然后从 message 解压缩到结构数组 new_s[N]中。打印压缩前(old_s)、解压后(new_s)的结果，以及压缩前、压缩后存放数据的长度。

要求：

（1） 输入的第 0 个人姓名(name)为自己的名字，分数为学号的最后两位；

（2） 编写指定接口的函数完成数据压缩

压缩函数有两个：　int　pack_student_bytebybyte(student* s, int sno, char *buf);

　　int　pack_student_whole(student* s, int sno, char *buf);

s 为待压缩数组的起始地址； sno 为压缩人数； buf 为压缩存储区的首地址；两个函数的返回均是调用函数压缩后的字节数。pack_student_bytebybyte 要求一个字节一个字节的向 buf 中写数据；pack_student_whole 要求对 short、float 字段都只能用一条语句整体写入，用 strcpy 实现串的写入。

（3） 使用指定方式调用压缩函数

old_s 数组的前 N1（N1=2）个记录压缩调用 pack_student_bytebybyte 完成；后 N2（N2==3）个记录压缩调用 pack_student_whole，两种压缩函数都只调用 1 次。

（4） 使用指定的函数完成数据的解压

解压函数的格式：int restore_student(char *buf, int len, student* s);

buf 为压缩区域存储区的首地址；len 为 buf 中存放数据的长度；s 为存放解压数据的结构数组的起始地址；返回解压的人数。解压时不允许使用函数接口之外的信息（即不允许定义其他全局变量）

（5）仿照调试时看到的内存数据，以十六进制的形式，输出 message 的前 40 个字节的内容，并与调试时在内存窗口观察到的 message 的前 40 个字节比较是否一致。

（6）对于第 0 个学生的 score，根据浮点数的编码规则指出其个部分的编码，并与观察到的内存表示比较，验证是否一致。

(7) 指出结构数组中个元素的存放规律，指出字符串数组、short 类型的数、float 型的数的存放规律。

任务 2 编写位运算程序

按照要求完成给定的功能，并**自动判断程序**的运行结果是否正确。（从逻辑电路与门、或门、非门等等角度，实现 CPU 的常见功能。所谓自动判断，即用简单的方式实现指定功能，并判断两个函数的输出是否相同。）

(1) int absVal(int x); 返回 x 的绝对值
仅使用 !、 ~、 &、 ^、 |、 +、 <<、 >>， 运算次数不超过 10 次
判断函数： int absVal_standard(int x) { return (x < 0) ? -x : x;}

(2) int negate(int x); 不使用负号，实现 -x
判断函数： int netgate_standard(int x) { return -x;}

(3) int bitAnd(int x, int y); 仅使用 ~ 和 |，实现 &
判断函数： int bitAnd_standard(int x, int y) { return x & y;}

(4) int bitOr(int x, int y); 仅使用 ~ 和 &，实现 |

(5) int bitXor(int x, int y); 仅使用 ~ 和 &，实现 ^

(6) int isTmax(int x); 判断 x 是否为最大的正整数（7FFFFFFF），
只能使用 !、 ~、 &、 ^、 |、 +

(7) int bitCount(int x); 统计 x 的二进制表示中 1 的个数
只能使用，!~&^|+<<>> ，运算次数不超过 40 次

(8) int bitMask(int highbit, int lowbit); 产生从 lowbit 到 highbit 全为 1，其他位为 0 的数。例如 bitMask(5,3) = 0x38 ；要求只使用 !~&^|+<<>> ；运算次数不超过 16 次。

(9) int addOK(int x, int y); 当 x+y 会产生溢出时返回 1，否则返回 0
仅使用 !、 ~、 &、 ^、 |、 +、 <<、 >>， 运算次数不超过 20 次

(10) int byteSwap(int x, int n, int m); 将 x 的第 n 个字节与第 m 个字节交换，返回交换后的结果。 n、m 的取值在 0~3 之间。
例：byteSwap(0x12345678, 1, 3) = 0x56341278
byteSwap(0xDEADBEEF, 0, 2) = 0xDEEFBEAD
仅使用 !、 ~、 &、 ^、 |、 +、 <<、 >>， 运算次数不超过 25 次

(11) int bang(int x) 当 x =0 时，返回 1; 其他情况返回 0。实现逻辑非(!)
例：bang(3) = 0； bang(0) =1;
仅使用 ~、 &、 ^、 | 、+、 <<、 >>，运算次数不超过 12 次
提示：只有当 x=0 时，x 与 -x 的最高二进制位会同时为 0。

(12) int bitParity(int x) 当 x 有奇数个二进制位 0，返回 1；否则返回 0
例：bitParity (5) = 0； bitParity (7) = 1;
仅使用 ！、~、 &、 ^、 | 、+ 、<< 、>> ，运算次数不超过 20 次。
提示：只有当 x 的高字与低字的对应位（第 0 位对应第 16 位，第 1 位对应第 17 位，依次类推）同时为 1，则出现了成对的二进制位 1，此时，可以将对应的二进制位置为 0，不会影响二进制位 1 的个数的奇偶性判断。

三、实验记录及问题回答

（1）任务 1

1. 算法思想

（1）压缩函数 int pack_student_bytebybyte(student* s, int sno, char *buf);

循环逐个读取 student 结构体类型数组内容

对于数组中每个 student 的每一项进行逐个字节压缩

name 压缩：将 char 类型不为‘\0’字符逐个读取写入目标位置，内容写入结束后将结尾‘\0’写入以指示 name 结束；写入过程中逐位计数。

age 压缩：short 类型为 2 字节，将 2 字节写入目标位置；写入过程中逐位计数。

score 压缩：float 类型为 4 字节，将 4 字节写入目标位置；写入过程中逐位计数。

remark 压缩：将 char 类型不为‘\0’字符逐个读取写入目标位置，内容写入结束后将结尾‘\0’写入以指示 remark 结束；写入过程中逐位计数。

（2）压缩函数 int pack_student_whole(student* s, int sno, char *buf);

循环逐个读取 student 结构体类型数组内容

对于数组中每个 student 的每一项进行整体压缩

name 压缩：用 strcpy 函数将 name 写入到目标位置，用 strlen 函数计算下一次写入的目标位置，由于 strlen 不包括‘\0’，因此需移动字节数为 strlen（name）+1；同时用 strlen（name）+1 计数。

age 压缩：将目标地址强制转换为 short 类型指针，一步即可存入 age，指针移动 2 字节；计数增加 2。

score 压缩：将目标地址强制转换为 float 类型指针，一步即可存入 score，指针移动 4 字节；计数增加 4。

remark 压缩：用 strcpy 函数将 remark 写入到目标位置，用 strlen 函数计算下一次写入的目标位置，由于 strlen 不包括‘\0’，因此需移动字节数为 strlen（remark）+1；同时用 strlen（remark）+1 计数。

（3）解压函数 int restore_student(char *buf, int len, student* s);

标记读取结束位置，用于结束循环

循环逐个写入 student 结构体类型数组内容

name 解压：用 strcpy 函数函数将 name 写入到数组，用 strlen 函数计算下一次读取位置，由于 strlen 不包括‘\0’，因此需移动字节数为 strlen（name）+1。

age 解压：将读取位置强制转换为 short 类型指针，写入数组，读取位置移动 2 字节。

score 解压：将读取位置强制转换为 float 类型指针，写入数组，读取位置移动 4 字节。

remark 解压：用 strcpy 函数函数将 remark 写入到数组，用 strlen 函数计算下一次读取位置，由于 strlen 不包括‘\0’，因此需移动字节数为 strlen（remark）+1；此时一位 student 信息录取结束，人数 num 增加 1。

2. 运行结果

图 1 任务 1 运行结果

将信息输入，存入数组 old_s，经过压缩至 message 数组再解压至 new_s 输出结果

压缩至 message 后以十六进制的形式，输出 message 的前 40 个字节的内容

3. 观察记录问题的解答

（1）仿照调试时看到的内存数据，以十六进制的形式，输出 message 的前 40 个字节的内容，并与调试时在内存窗口观察到的 message 的前 40 个字节比较是否一致。

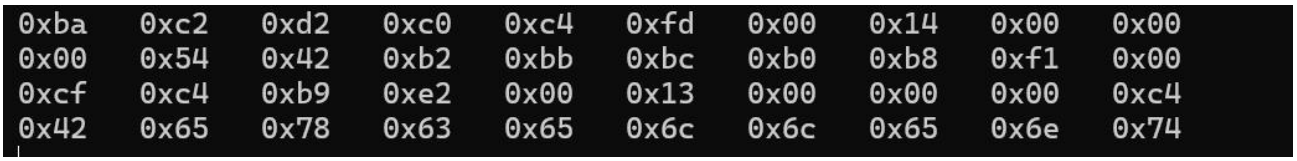

图 2 message 前 40 个字节

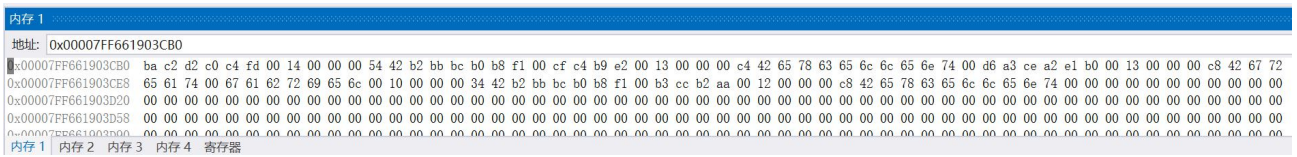

图 3 内存窗口观察 message 的字节

以十六进制的形式输出 message 的前 40 个字节的内容调试时在内存窗口观察到的 message 的前 40 个字节一致。其中 0xba~0xfd 为第 0 个学生 name，0x00 标志 name 结束；0x14~0x00 为第 0 个学生 age，类型为 short，2 字节，小端存储可得年龄为 20；0x00~0x42wei 第 0 个学生 score，类型为 float，4 字节，小端存储可得 0100 0010 0101 0100 0000 0000 0000 0000 根据浮点数的编码规则可得 53。

（2）对于第 0 个学生的 score，根据浮点数的编码规则指出其个部分的编码，并与观察到的内存表示比较，验证是否一致。

第 0 个学生 score 为 53

编码应为 0x42540000：将 53 转化为二进制 110101，根据浮点数的编码规则转化为 1.10101000000000000000000*2^5，正数符号位为 0，指数移码为 127+5=132（二进制 10000100），尾数 23 位为 10101000000000000000000，即 0100 0010 0101 0100 0000 0000 0000 0000，转化为 16 进制应为 0x42540000。由于小端存储，在内存中应为 0x00 0x00 0x54 0x42。

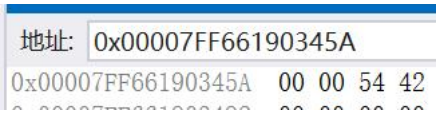

图 4 old_s[0].score

二进制为 0100 0010 0101 0100 0000 0000 0000 0000，根据浮点数的编码规则拆分 0 10000100 10101000000000000000000，即符号位为 0-正数，指数移码为 10000100，指数为 132-127=5，尾数为 10101000000000000000000，即得出浮点数 1.10101000000000000000000*2^5=53。

（3）指出结构数组中个元素的存放规律，指出字符串数组、short 类型的数、float 型的数的存放规律。

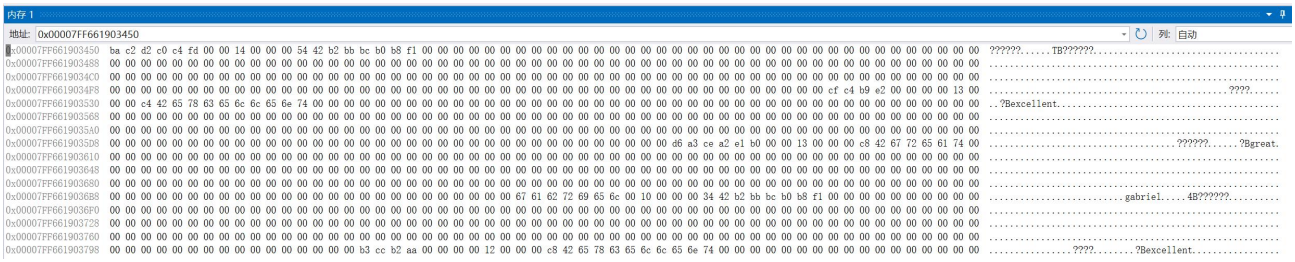

图 5 old_s 数组

结构数组中每个元素占用 214 字节，依次存放 char 类型数组 name（占用 8 字节）、short 类型 age、float 类型 score、char 类型数组 remark（占用 200 字节），元素逐个紧邻存放。

字符串数组中每个元素占用 1 字节，元素逐个紧邻存放。

short 类型数占用 2 字节，小端存储。

float 类型数占用 4 字节，小端存储，根据 IEEE 754 单精度浮点数的标准（1 位符号位，8 位指数位，23 位尾数位）转换为 2 进制存储。

（2）任务 2

1. 算法思想

（1）int absVal(int x)

若 x 为负数，x 绝对值应为-x，即 x 按位取反后加一；若 x 为正数，x 绝对值应保持不变。

x 为负数右移 31 位变为 0xffffffff，x 为正数右移 31 位变为 0x00000000；1 异或效果为取反，0 异或效果为保持原有数据不变，则可以用 x 右移 31 位结果异或 x 达到负数取反、正数不变效果；用 x 右移 31 位结果并 1 达到负数加一、正数不变效果。

（2）int negate(int x)

x 转换为-x 方式为 x 按位取反后加一

（3）int bitAnd(int x, int y)

数学原理：x∧y=¬（¬x∨¬y）

（4）int bitOr(int x, int y)

数学原理：x∨y=¬（¬x∧¬y）

（5）int bitXor(int x, int y)

数学原理：x⊕y=¬（¬x∧¬y）∧¬（x∧y）

（6）int isTmax(int x)

检查 x 与 0x7fffffff 是否相同，通过异或得到若二者不同结果不为 0，相同结果为 0；根据要求对结果取非即可。

（7）int bitCount(int x)

分组得到结果后汇总。

进行 2 位分组，对 2 位相加即可获得 2 位一组中的 1 个数；进行 4 位分组，对 4 位中 2 个 2 位相加即可获得 4 位一组中的 1 个数；进行 8 位分组，对 8 位中 2 个 4 位相加即可获得 8 位一组中的 1 个数；进行 16 位分组，对 16 位中的 2 个 8 位相加即可获得 16 位一组中的 1 个数；进行 32 位分组，对 32 位中 2 个 16 位相加即可获得 32 位一组中的 1 个数，即 int 类型数据二进制形式 1 的个数。

（8）int bitMask(int highbit, int lowbit)

取到 32 位 1 的数据左移 highbit+1 位后按位取反，得到 0~highbit 位为 1 其余为 0 的数据；将数据右移 lowbit 位后左移 lowbit 位即可令 0~lowbit-1 位变为 0，其余位不变；达到除 lowbit~highbit 位为 1 其余为 0。

（9）int addOK(int x, int y)

溢出时为 x 与 y 同号且 x、y 与 x+y 异号

保存符号位：x、y、x+y 均右移 31 位得到 32 位数据均与其符号位相同

判断符号位：通过按位异或判断 x、y 右移后数据是否相同（相同为 0，不同为 0xffffffff），对结果取反得到不同为 0，相同为 0xffffffff；通过按位异或判断 x、 x+y 右移后数据是否相同（相同为 0，不同为 0xffffffff）

两条件满足情况：二者结果并得到当 x、y 右移后数据相同且 x、x+y 右移后数据不同时（即满足 x 与 y 同号且 x、y 与 x+y 异号）结果为 0xffffffff，其余条件结果为 0

调整输出：结果并 1 达到输出要求

（10）int byteSwap(int x, int n, int m)

保存 n 字节位置数据：x 右移 n*8 位后并 0xff（与 1 并数据不变，与 0 并数据清 0）

保存 m 字节位置数据：x 右移 m*8 位后并 0xff（与 1 并数据不变，与 0 并数据清 0）

清空 n、m 字节数据：x 并 0xff 左移 n*8 位后数据，x 并 0xff 左移 m*8 位后数据

填入 n、m 字节数据：保存的 n 字节位置数据左移 m*8 位到达原 m 字节处，保存的 m 字节位置数据左移 n*8 位到达原 n 字节处，得到两个数据或清空后的 x（与 0 或数据不变）

（11）int bang(int x)

只有当 x=0 时，x 与 -x 的最高二进制位会同时为 0。

保存数据符号位：x、-x（按位取反+1 得到）均右移 31 位得到 32 位数据均与其符号位相同

与 0 比较：得到两个 32 位数据进行或运算后与 0 异或，仅当 x、-x 符号位均为 0 时结果为 0

调整输出：结果+1（若为 0xffffffff+1 溢出得到 0，若为 0+1 得到 1）符合输出要求

（12）int bitParity(int x)

16 位一组：分为 2 组，高 16 位右移与低 16 位异或消除同时对应位置为 1，只需判断得到低 16 位 1 个数是否为奇数

8 位一组：分为 2 组，高 8 位右移与低 8 位异或消除同时对应位置为 1，只需判断得到低 8 位 1 个数是否为奇数

4 位一组：分为 2 组，高 4 位右移与低 4 位异或消除同时对应位置为 1，只需判断得到低 4 位 1 个数是否为奇数

2 位一组：分为 2 组，高 2 位右移与低 2 位异或消除同时对应位置为 1，只需判断得到低 2 位 1 个数是否为奇数

1 位一组：分为 2 组，高 1 位右移与低 1 位异或消除同时对应位置为 1，只需判断得到低 1 位 1 个数是否为奇数

调整输出：最低 1 位为结果，将高 31 位置零，即左移 31 位后右移 31 位

2. 运行结果等记录

（1）测试边缘数据

```
int main()
{
    int x = 0x80000000;
    int y = 0x7fffffff;
    int n = 2;
    int m = 3;
    std::cout << "absVal: " << absVal(x) << " " << absVal_standard(x) << std::endl;
    std::cout << "negate: " << negate(x) << " " << netgate_standard(x) << std::endl;
    std::cout << "bitAnd: " << bitAnd(x, y) << " " << bitAnd_standard(x, y) << std::endl;
    std::cout << "bitOr: " << bitOr(x, y) << " " << bitOr_standard(x, y) << std::endl;
    std::cout << "bitXor: " << bitXor(x, y) << " " << bitXor_standard(x, y) << std::endl;
    std::cout << "isTmax0: " << isTmax(x) << " " << isTmax_standard(x) << std::endl;
    std::cout << "isTmax1: " << isTmax(y) << " " << isTmax_standard(y) << std::endl;
    std::cout << "bitCount0: " << bitCount(x) << " " << bitCount_standard(x) << std::endl;
    std::cout << "bitCount1: " << bitCount(y) << " " << bitCount_standard(y) << std::endl;
    std::cout << "bitMask: " << bitMask(m, n) << " " << bitMask_standard(m, n) << std::endl;
    std::cout << "addOK0: " << addOK(x, y) << " " << addOK_standard(x, y) << std::endl;
    std::cout << "addOK1: " << addOK(y, y) << " " << addOK_standard(y, y) << std::endl;
    std::cout << "byteSwap: " << byteSwap(x, n, m) << " " << byteSwap_standard(x, n, m) << std::endl;
    std::cout << "bang0: " << bang(x) << " " << bang_standard(x) << std::endl;
    std::cout << "bang1: " << bang(0) << " " << bang_standard(0) << std::endl;
    std::cout << "bitParity0: " << bitParity(3) << " " << bitParity_standard(3) << std::endl;
    std::cout << "bitParity1: " << bitParity(x) << " " << bitParity_standard(x) << std::endl;
    return 0;
}
```

图 6 测试

（3）测试结果

```
absVal: -2147483648 -2147483648
negate: -2147483648 -2147483648
bitAnd: 0 0
bitOr: -1 -1
bitXor: -1 -1
isTmax0: 0 0
isTmax1: 1 1
bitCount0: 1 1
bitCount1: 31 31
bitMask: 12 12
addOK0: 0 0
addOK1: 1 1
byteSwap: 8388608 8388608
bang0: 0 0
bang1: 1 1
bitParity0: 0 0
bitParity1: 1 1

D:\计算机系统基础实验\计基1_2\x64\Debug\计基1_2.exe (进程 38216)已退出，代码为 0 (0x0)。
按任意键关闭此窗口. . .
```

图 7 测试结果正确

四、体会

本次实验通过手动计算、内存观察和工具验证，将抽象的存储规则具象化，加深了对以下内容的理解：

浮点数的 IEEE 754 编码规则

结构体内存对齐与填充机制

小端序的实际表现

调试工具在底层分析中的应用

这些知识为后续学习内存管理、性能优化及跨平台开发奠定了基础。

本次实验基础性强，知识覆盖全面，通过本次实验我能更好理解数据在计算机内部的存储。

五、源码

1. 任务 1

```
#include<iostream>
#include<string>
using namespace std;
#define N 5  // 学生人数

#pragma pack(1)
struct student {
    char  name[8];
    short  age;
    float  score;
    char  remark[200];  // 备注信息
};

student old_s[N];
student new_s[N];
char message[1250];


void input() {
    for (int i = 0; i < N; i++) {
        cout << "请输入第" << i + 1 << "个学生的姓名：";
        cin >> old_s[i].name;
        cout << "请输入第" << i + 1 << "个学生的年龄：";
        cin >> old_s[i].age;
        cout << "请输入第" << i + 1 << "个学生的成绩：";
        cin >> old_s[i].score;
        cout << "请输入第" << i + 1 << "个学生的备注信息：";
        cin >> old_s[i].remark;
    }
}
```

```
void output() {
    for (int i = 0; i < N; i++) {
        cout << "第" << i + 1 << "个学生的姓名：" << new_s[i].name << endl;
        cout << "第" << i + 1 << "个学生的年龄：" << new_s[i].age << endl;
        cout << "第" << i + 1 << "个学生的成绩：" << new_s[i].score << endl;
        cout << "第" << i + 1 << "个学生的备注信息：" << new_s[i].remark << endl;
    }
}

//s 待压缩数组的起始地址；sno 压缩人数；buf 压缩存储区的首地址；返回调用函数压缩后的字节数
int pack_student_bytebybyte(student* s, int sno, char* buf) {
    int num = 0;
    for (int i = 0; i < sno; i++) {
        for (int namei = 0; namei < 8; namei++) {
            *buf = s[i].name[namei];
            buf++;
            num++;
            if (s[i].name[namei] == '\0') {
                break;
            }
        }

        for (int agei = 0; agei < sizeof(short); agei++) {
            *buf = ((char*)&(s[i].age))[agei];
            buf++;
            num++;
        }

        for (int scorei = 0; scorei < sizeof(float); scorei++) {
            *buf = ((char*)&(s[i].score))[scorei];
            buf++;
            num++;
        }

        for (int remarki = 0; remarki < 200; remarki++) {
            *buf = s[i].remark[remarki];
            buf++;
            num++;
            if (s[i].remark[remarki] == '\0') {
                break;
            }
        }
    }
    return num;
}
```

```
int  pack_student_whole(student* s, int sno, char* buf) {
    int num = 0;
    for (int i = 0; i < sno; i++) {
        strcpy(buf, s[i].name);
        buf += strlen(s[i].name);
        buf++;
        num += strlen(s[i].name);
        num++;

        *((short*)buf) = s[i].age;
        buf += sizeof(short);
        num += sizeof(short);

        *((float*)buf) = s[i].score;
        buf += sizeof(float);
        num += sizeof(float);

        strcpy(buf, s[i].remark);
        buf += strlen(s[i].remark);
        buf++;
        num += strlen(s[i].remark);
        num++;
    }
    return num;
}
```

//buf 为压缩区域存储区的首地址；len 为 buf 中存放数据的长度；s 为存放解压数据的结构数组的起始地址；返回解压的人数

```
int restore_student(char* buf, int len, student* s) {
    int num = 0;
    char* end = buf + len;

    for (int i = 0; i < N; i++) {
        if (buf >= end) {
            break;
        }
        strcpy(s[i].name, buf);
        buf += strlen(s[i].name);
        buf++;

        s[i].age = *((short*)buf);
        buf += sizeof(short);;

        s[i].score = *((float*)buf);
        buf += sizeof(float);
```

```
            strcpy(s[i].remark, buf);
            buf += strlen(s[i].remark);
            buf++;

            num++;
        }

        return num;
    }

    int main() {
        input();

        int num1 = pack_student_bytebybyte(old_s, 2, message);
        int num2 = pack_student_whole(&(old_s[2]), 3, &(message[num1]));

        //以十六进制的形式，输出 message 的前 40 个字节的内容，并与调试时在内存窗口观察到的
message 的前 40 个字节比较是否一致
        for (int i = 0; i < 40; i++) {
            printf("0x%02hhx   ", message[i]);
        }
        printf("\n");

        int num = num1 + num2;
        int person = restore_student(message, num, new_s);

        output();
        return 0;
    }
```

2. 任务 2

```
#include <iostream>

// 返回 x 的绝对值,仅使用 !、 ~、 & 、^ 、 | 、 + 、 << 、 >> ， 运算次数不超过 10 次
int absVal(int x)
{
    return ((x >> 31) ^ x) + ((x >> 31) & 1);
}
// 判断函数
int absVal_standard(int x) { return (x < 0) ? -x : x; }

// 不使用负号，实现 - x
int negate(int x)
{
    return ~x + 1;
```

```
}
// 判断函数
int netgate_standard(int x) { return -x; }

// 仅使用 ~和 | , 实现 &
int bitAnd(int x, int y)
{
    return ~(~x | ~y);
}
// 判断函数
int bitAnd_standard(int x, int y) { return x & y; }

// 仅使用 ~和 & , 实现 |
int bitOr(int x, int y)
{
    return ~(~x & ~y);
}
// 判断函数
int bitOr_standard(int x, int y) { return x | y; }

// 仅使用 ~和 & , 实现 ^
int bitXor(int x, int y)
{
    return ~(~x & ~y) & ~(x & y);
}
// 判断函数
int bitXor_standard(int x, int y) { return x ^ y; }

// 判断 x 是否为最大的正整数（7FFFFFFF）, 只能使用 !、 ~、 & 、^ 、 | 、 +
int isTmax(int x)
{
    return !(x ^ (0x7fffffff));
}
// 判断函数
int isTmax_standard(int x) { return x == 0x7fffffff; }

// 统计 x 的二进制表示中 1 的个数,只能使用, !~&^| +<< >> , 运算次数不超过 40 次
int bitCount(int x)
{
    x = (x & 0x55555555) + ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = (x & 0x0f0f0f0f) + ((x >> 4) & 0x0f0f0f0f);
    x = (x & 0x00ff00ff) + ((x >> 8) & 0x00ff00ff);
    x = (x & 0x0000ffff) + ((x >> 16) & 0x0000ffff);
    return x;
```

```
}
// 判断函数
int bitCount_standard(int x)
{
    int count = 0;
    for (int i = 0; i < 32; i++)
    {
        if (x & 1)
        {
            count++;
        }
        x >>= 1;
    }
    return count;
}
```

// 产生从 lowbit 到 highbit 全为 1，其他位为 0 的数。例如 bitMask(5, 3) = 0x38 ；要求只使用 !~&^| +<< >> ；运算次数不超过 16 次

```
int bitMask(int highbit, int lowbit)
{
    return (~(~0 << (highbit + 1))) >> lowbit << lowbit;
}
// 判断函数
int bitMask_standard(int highbit, int lowbit)
{
    int result = 0;
    for (int i = 0; i <= highbit - lowbit; i++)
    {
        result = result | 1;
        result = result << 1;
    }
    result = result << lowbit - 1;
    return result;
}
```

// 当 x + y 会产生溢出时返回 1，否则返回 0,仅使用 !、 ~、 & 、 ^ 、 | 、 + 、 << 、 >> ， 运算次数不超过 20 次

```
int addOK(int x, int y)
{
    return  (~((x >> 31) ^ (y >> 31)) & ((x >> 31) ^ ((x + y) >> 31))) & 1;
}
// 判断函数
int addOK_standard(int x, int y)
{
    int result = 0;
```

```
    int z = x + y;
    if (x >= 0 && y >= 0 && z < 0)
        result = 1;
    if (x < 0 && y < 0 && z >= 0)
        result = 1;
    return result;
}

// 将 x 的第 n 个字节与第 m 个字节交换，返回交换后的结果。 n、m 的取值在 0~3 之间。仅使用 !、~、 & 、 ^ 、 | 、 + 、 << 、 >> ， 运算次数不超过 25 次
int byteSwap(int x, int n, int m)
{
    int nbyte = (x >> (n << 3)) & 0xff;
    int mbyte = (x >> (m << 3)) & 0xff;
    x = x & ~(0xff << (n << 3)) & ~(0xff << (m << 3));
    x = x | (nbyte << (m << 3)) | (mbyte << (n << 3));
    return x;
}
// 判断函数
int byteSwap_standard(int x, int n, int m)
{
    char* p = (char*)&x;
    char temp = p[n];
    p[n] = p[m];
    p[m] = temp;
    return x;
}

// 当 x=0 时，返回 1; 其他情况返回 0。实现逻辑非(!),仅使用 ~、 & 、^ 、 | 、 + 、 << 、>> ，运算次数不超过 12 次
int bang(int x)
{
    return (((~x + 1) >> 31 | (x >> 31)) ^ 0) + 1;
}
// 判断函数
int bang_standard(int x) { return !x; }

// 当 x 有奇数个二进制位 0，返回 1；否则返回 0,仅使用 !、~、 & 、^ 、 | 、 + 、 << 、 >> ，运算次数不超过 20 次。
int bitParity(int x)
{
    x = x ^ (x >> 16);
    x = x ^ (x >> 8);
    x = x ^ (x >> 4);
    x = x ^ (x >> 2);
```

```
        x = x ^ (x >> 1);
        x = x << 31 >> 31;
        return x & 1;
    }
    // 判断函数
    int bitParity_standard(int x)
    {
        int count = 0;
        for (int i = 0; i < 32; i++)
        {
            if (x & 1)
            {
                count++;
            }
            x >>= 1;
        }
        return count % 2;
    }

    int main()
    {
        int x = 0x80000000;
        int y = 0x7fffffff;
        int n = 2;
        int m = 3;
        std::cout << "absVal: " << absVal(x) << " " << absVal_standard(x) << std::endl;
        std::cout << "negate: " << negate(x) << " " << netgate_standard(x) << std::endl;
        std::cout << "bitAnd: " << bitAnd(x, y) << " " << bitAnd_standard(x, y) << std::endl;
        std::cout << "bitOr: " << bitOr(x, y) << " " << bitOr_standard(x, y) << std::endl;
        std::cout << "bitXor: " << bitXor(x, y) << " " << bitXor_standard(x, y) << std::endl;
        std::cout << "isTmax0: " << isTmax(x) << " " << isTmax_standard(x) << std::endl;
        std::cout << "isTmax1: " << isTmax(y) << " " << isTmax_standard(y) << std::endl;
        std::cout << "bitCount0: " << bitCount(x) << " " << bitCount_standard(x) << std::endl;
        std::cout << "bitCount1: " << bitCount(y) << " " << bitCount_standard(y) << std::endl;
        std::cout << "bitMask: " << bitMask(m, n) << " " << bitMask_standard(m, n) << std::endl;
        std::cout << "addOK0: " << addOK(x, y) << " " << addOK_standard(x, y) << std::endl;
        std::cout << "addOK1: " << addOK(y, y) << " " << addOK_standard(y, y) << std::endl;
        std::cout << "byteSwap: " << byteSwap(x, n, m) << " " << byteSwap_standard(x, n, m) <<
std::endl;
        std::cout << "bang0: " << bang(x) << " " << bang_standard(x) << std::endl;
        std::cout << "bang1: " << bang(0) << " " << bang_standard(0) << std::endl;
        std::cout << "bitParity0: " << bitParity(3) << " " << bitParity_standard(3) << std::endl;
        std::cout << "bitParity1: " << bitParity(x) << " " << bitParity_standard(x) << std::endl;
        return 0;
    }
```