# 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：** 数据的表示


院　　系：计算机科学与技术

专业班级：计算机 202201

学　　号：U202215357

姓　　名：王文涛

指导教师：朱　虹


2024 年 3 月 24 日

## 一、实验目的与要求

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

⑵ 熟悉地址的计算方法、地址的内存转换；

⑶ 熟悉数据的表示形式；

## 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，输入 N 个学生的信息到结构数组 old_s 中。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 20 个字节的内容，并与调试时在内存窗口观察到的 message 的前 20 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各部分的编码，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　编写数据表示的自动评测程序

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

# 三、实验记录及问题回答

## （1）任务 1 的算法思想、运行结果等记录

**算法思想：**

对于函数 pack_student_bytebybyte，要实现按字节压缩，所以将 student 类型指针使用 char 类型强制转换。读取字符串时，读到’\0’为止，并把’\0’存入 message 中，便于解压。读取数字时，按照类型所占字节数读取，读完 short 型数后，跳过两个对齐用的空字节。

对于函数 pack_student_whole，要用一条语句整体写入。对于字符串，使用 strcpy 读取，对于数，直接使用 memcpy 读取。同时记录每次读取的长度，以便确定读取的首地址位置。

对于 restore_student 函数，依然使用 strcpy 读取字符串，使用 memcpy 读取数。每次将总长度 len 减去每次读完第二个字符串的局部总长度，当 len 为 0 时说明已经读完了。

对于问题（5），输出 message 前 20 字节并于内存比较，发现完全一致。

对于问题（6），我的学号后两位为 57，转换为规格化浮点数为 1.11001*2^5，符号位为 0，指数位的移码为 10000100，尾数原码为 11001000000000000000000，转换为 16 进制为 0x42 64 00 00，由于计算机将数据低字节存放到地址高字节，所以内存中应该是 0x00 00 64 42，与内存比较，完全一致。

对于问题（7），结构数组连续存储。在每个结构体中，先是八个字节的字符数组，一个字符占一个字节，最后以’\0’结尾。short 型的数占两个字节，小端存储。float 型的数规格化后，按符号位，指数位移码，尾数位原码存储，也是小端存储。接着是两百个字节的字符数组，后面接着下一个结构体。

**运行结果：**

运行结果如下。

```
解压前数据：
name: wentao
age: 19
score: 57.000000
remark: none0

name: no1
age: 12
score: 34.220001
remark: none1

name: no2
age: 22
score: 12.350000
remark: none2

name: no3
age: 43
score: 66.000000
remark: none3

name: no4
age: 33
score: 123.400002
remark: none4

压缩后长度为83字节

解压后数据：
name: wentao
age: 19
score: 57.000000
remark: none0

name: no1
age: 12
score: 34.220001
remark: none1

name: no2
age: 22
score: 12.350000
remark: none2

name: no3
age: 43
score: 66.000000
remark: none3

name: no4
age: 33
score: 123.400002
remark: none4

缓冲区前20字节： 77 65 6e 74 61 6f 00 13 00 00 00 64 42 6e 6f 6e 65 30 00 6e
```

图 1-1 运行结果

将 message 中存的内容与内存比较，完全一致

```
77 65 6e 74 61 6f 00 13 00 00 00 64 42 6e 6f 6e 65 30 00 6e
6f 31 00 0c 00 48 e1 08 42 6e 6f 6e 65 31 00 6e 6f 32 00 16
00 9a 99 45 41 6e 6f 6e 65 32 00 6e 6f 33 00 2b 00 00 00 84
42 6e 6f 6e 65 33 00 6e 6f 34 00 21 00 cd cc f6 42 6e 6f 6e
65 34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

图 1-2 内存视图

**（2）任务 2 的算法思想、运行结果等记录**

**算法思想：**

对于函数 absVal(int x)，先取 x 符号位 sign,然后返回(x + sign) ^ sign，若 x 为正，则 sign 为 0，相当于直接返回 x，若 x 为负，则 sign 为 0xffffffff，相当于 x 减 1 然后取反，返回-x。

对于函数 negate(int x)，直接返回~x + 1，即为相反数。

对于函数 bitAnd(int x, int y)，将与转换为或非表达式，即为~(~x | ~y)。

对于函数 bitOr (int x, int y)，将或转换为与非表达式，即为~(~x & ~y)。

对于函数 bitXor (int x, int y)，将异或转换为与非表达式，x^y=((~x)&y)|(x&(~y)),再将或转换为与非，即为~(~(~x & y) & ~(x & ~y))。

对于函数 isTmax(int x)，要判断 x 是否为 0x7FFFFFFF，用先异或再取非来判断，返回~(x ^ 0x7FFFFFFF)。

对于函数 bitCount(int x)，要在不使用循环的条件下判断 x 中 1 的个数。首先构造三个掩码，mask1=0B01010101010101010101010101010101, mask2=0B00110011001100110011001100110011, mask3=0B00001111000011110000111100001111。先执行 x = (x & mask1) + ((x >> 1) & mask1)，执行后，x 每 2 位的大小表示这 2 位 1 的个数，再执行 x = (x & mask2) + ((x >> 2) & mask2)，执行后 x 每 4 位的大小表示这 4 位 1 的个数，再执行 x = (x & mask3) + ((x >> 4) & mask3)，执行后 x 每 8 位的大小表示这 8 位 1 的个数，接着 x = (x + (x >> 8))，将 x 第 1，3 字节的数字加到 2，4 字节上，接着 x = (x + (x >> 16))，将四个字节上的数字都加到第四个字节上，并且因为数字大小不会到第 7 位，所以只用返回后 6 位，即返回 x & 0x3f。

对于函数 bitMask(int highbit, int lowbit)，先构造二进制全为一的数~0u。然后将它向左移 lowbit 位，与它向左移 highbit 加 1 位的结果异或，即(~0u << lowbit) ^ (~0u << highbit << 1)。

对于函数 addOK(int x, int y)，分别求出 x,y,x+y 的符号位，若 x 与 y 的符号位不同，或 x 与 x+y 的符号位不同则一定没有溢出，反之则溢出。

对于函数 byteSwap (int x, int n, int m)，先将 x 的第 n 个和第 m 个字节取出，然后将 x 中的第 n 个和第 m 个字节删去，将第 m 个字节放在第 n 个字节的位置，将第 n 个字节放在第 m 个字节的位置。

**运行结果：**

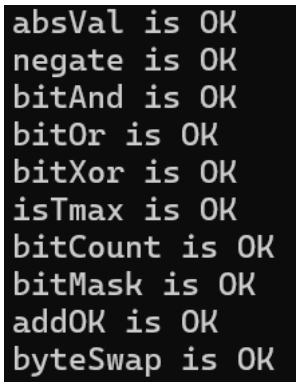


图 2-1 运行结果

# 四、体会

为了完成实验 1，我需要反复在内存中查看数据的存放情况，这让我对计算机中数据的存放方式有了更深的理解，巩固了计算机系统基础的知识。在实现压缩和解压的操作时，我对 c 语言中指针的用法和强制类型转换也更加熟练了。

在完成实验 2 的过程中，我对位操作有了更深的理解，对补码的认识也更加深刻。实现 bitCount 函数时，学习到

了更加高效的方法，受益匪浅。

## 五、源码

任务 1：

```
#include<stdio.h>
#include<string.h>

const int N = 5;
const int N1 = 2;
const int N2 = 3;
char message[500];

struct student {
    char  name[8];
    short  age;
    float  score;
    char  remark[200];
}old_s[N], new_s[N];

int  pack_student_bytebybyte(student* s, int sno, char* buf)
{
    int i = 0;
    for (int k = 0; k < sno; k++)
    {
        char* p = (char*)(s+k);
        char* q = p;
        do {
            buf[i++] = *q;
        } while (*(q++));
        q = p + 8;
        for (int k = 0; k < 2; k++)
            buf[i++] = *(q++);
        q += 2;
        for (int k = 0; k < 4; k++)
            buf[i++] = *(q++);
        do {
            buf[i++] = *q;
        } while (*(q++));
    }
    return i;
}

int  pack_student_whole(student* s, int sno, char* buf)
{
    int len = 0,len1,len2;
```

```
    for (int i = 0; i < sno; i++)
    {
        len1 = strlen(s[i].name)+1;
        strcpy(buf+len, s[i].name);
        memcpy(buf + len + len1, &s[i].age, 2);
        memcpy(buf + len + len1 + 2, &s[i].score, 4);
        len2 = strlen(s[i].remark)+1;
        strcpy(buf + len +len1+6, s[i].remark);
        len +=  len1+ 6 + len2;
    }
    return len;
}

int restore_student(char* buf, int len, student* s)
{
    int num = 0,len1,len2;
    char* p = buf;
    while (len>0)
    {
        len1 = strlen(p)+1;
        strcpy(s[num].name, p);
        p += len1;
        memcpy(&s[num].age, p, 2);
        memcpy(&s[num].score, p + 2, 4);
        p += 6;
        len2 = strlen(p)+1;
        strcpy(s[num].remark, p);
        p += len2;
        len -= len1 + len2 + 6;
        num++;
    }
    return num;
}

void print_message(char* buf)
{
    printf("缓冲区前20字节：");
    for (int i = 0; i < 20; i++)
        printf("% 02x ", (unsigned char)buf[i]);
}

void print_student(student* s,int sno)
{
    for(int i=0;i<sno;i++)
        printf("name: %s\nage: %d\nscore: %f\nremark: %s\n\n", s[i].name, s[i].age, s[i].score,
```

```
s[i].remark);
}
int main()
{
    int len1,len2;
    freopen("1.txt", "r", stdin);
    for (int i = 0; i < N; i++)
        scanf("%s\n%hd\n%f\n%s", &old_s[i].name, &old_s[i].age, &old_s[i].score,
&old_s[i].remark);
    printf("解压前数据:\n");
    print_student(old_s, N);
    len1 = pack_student_bytebybyte(old_s, N1, message);
    len2 = pack_student_whole(old_s+ N1, N2, message+len1);
    restore_student(message, len1 + len2, new_s);
    printf("压缩后长度为%d字节\n\n", len1 + len2);
    printf("解压后数据:\n");
    print_student(new_s, N);
    print_message(message);
    return 0;
}
```

任务 2：

```
#include<stdio.h>

int absVal(int x)
{
    int sign = x >> 31;
    return (x + sign) ^ sign;
}
int absVal_standard(int x) { return (x < 0) ? -x : x; }

int negate(int x) { return ~x + 1; }
int netgate_standard(int x) { return -x; }

int bitAnd(int x, int y) { return ~(~x | ~y); }
int bitAnd_standard(int x, int y) { return x & y; }

int bitOr(int x, int y) { return ~(~x & ~y); }
int bitOr_standard(int x, int y) { return x | y; }

int bitXor(int x, int y) { return ~(~(~x & y) & ~(x & ~y)); }
int bitXor_standard(int x, int y) { return x ^ y; }

int isTmax(int x) { return ~(x ^ 0x7FFFFFFF); }
int isTmax_standard(int x) { return (x== 0x7FFFFFFF); }
```

```
int bitCount(int x)
{
    x = (x & 0x55555555) + ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = (x & 0x0f0f0f0f) + ((x >> 4) & 0x0f0f0f0f);
    x = (x + (x >> 8));
    x = (x + (x >> 16));
    return x & 0x3f;
}
int bitCount_standard(int x)
{
    int count = 0;
    for (int i = 0; i < 32; i++)
        count += (x >> i) & 1;
    return count;
}

int bitMask(int highbit, int lowbit) { return (~0u << lowbit) ^ (~0u << highbit << 1); }
int bitMask_standard(int highbit, int lowbit)
{
    int result = 1u << lowbit;
    for (int i = lowbit + 1; i <= highbit; ++i)
        result |= 1u << 1;
    return result;
}

int addOK(int x, int y)
{
    int x_sign = x >> 31, y_sign = y >> 31, sum_sign = (x + y) >> 31;
    return (x_sign ^ y_sign) | (x_sign ^ sum_sign);
}
int addOK_standard(int x, int y)
{
    int sum = x + y;
    return (x > 0 && y > 0 && sum < 0) || (x < 0 && y < 0 && sum > 0);
}

int byteSwap(int x, int n, int m)
{
    unsigned int nb = 0xff << ((4 - n) * 8);
    unsigned int mb = 0xff << ((4 - m) * 8);
    return x & ~(nb + mb) | (x & nb) << (n - 1) * 8 >> (m - 1) * 8 | (x & mb) << (m - 1) * 8 >>
(n - 1) * 8;
}
int byteSwap_standard(int x, int n, int m)
```

```
{
    char* p = (char*)&x, temp;
    temp = p[n];
    p[n] = p[m];
    p[m] = temp;
    return x;
}

int main()
{
    int a = -5, b = 3;
    if(absVal(a) == absVal(a))
        printf("absVal is OK\n");
    if (negate(a) == negate(a))
        printf("negate is OK\n");
    if (bitAnd(a,b) == bitAnd(a, b))
        printf("bitAnd is OK\n");
    if (bitOr(a, b) == bitOr(a, b))
        printf("bitOr is OK\n");
    if (bitXor(a, b) == bitXor(a, b))
        printf("bitXor is OK\n");
    if (isTmax(a) == isTmax(a))
        printf("isTmax is OK\n");
    if (bitCount(a) == bitCount(a))
        printf("bitCount is OK\n");
    if (bitMask(1, 3) == bitMask(1, 3))
        printf("bitMask is OK\n");
    if (addOK(a, b) == addOK(a, b))
        printf("addOK is OK\n");
    if (byteSwap(0x12345678, 1, 3) == byteSwap_standard(0x12345678, 1, 3))
        printf("byteSwap is OK\n");
}
```