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：　图灵 2301

学　号：　U202311239

姓　名：　刘星佳

指导教师：　王多强

2024 年 9 月 24 日

一、实验目的与要求

⑴ 熟练掌握程序开发的基本方法，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

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

pack_student_bytebybyte 函数中，对于 name 和 remark 这样的字符串，就直接遍历整个字符串并将字符逐一加到 buf 指针之后。而对于 short/float 型变量（以 short 型变量 age 为例），我们首先取得当前待压缩 student 结构体中 age 的首地址，并将其转换为 char *类型（这样就可以逐字节获取该 short 变量存储的内容），将该指针右移 sizeof(short)-1 次就可以获取 age 变量每字节的内容并将其拷贝到 buf 指针后面。float 型变量 score 同理。

pack_student_whole 函数中，对于字符串变量直接使用 strcpy 拷贝到 buf 之后，对于 short/float 型变量则使用 memcpy 函数，同样获取变量首地址并转换称 char *类型，将之后的 sizeof(short)/sizeof(float)字节直接用 memcpy 拷贝到 buf 指针之后即可。

在 restore_student 函数中，其实就是类似于 pack_student_bytebybyte 的逆向操作，先定位每个结构体变量展开成字节存储之后在 buf 中的首地址，然后获得对应结构体变量的首地址，将 buf 中向后 sizeof(变量)长度的字节依次拷贝到结构体中，即可复原 student 结构体。

输出的 message 前 20 个字节内容如下：

```
➜ lab1 git:(main) ✗ gcc task1.c -o task1 && ./task1 < task1data.txt
old_s:
name: lxj               age: 19         score: 39.000000                remark:abcde
name: rigel             age: 30         score: 233.330002               remark:islxj
name: abaaba            age: 2          score: 1.162000         remark:beizhu
name: adsf              age: 3          score: 34.549999                remark:bz
name: fas               age: 3342               score: 0.000000         remark:sd
message first 20th:
6C 78 6A 00 13 00 00 00 1C 42 61 62 63 64 65 00 72 69 67 65
new_s:
name: lxj               age: 19         score: 39.000000                remark:abcde
name: rigel             age: 30         score: 233.330002               remark:islxj
name: abaaba            age: 2          score: 1.162000         remark:beizhu
name: adsf              age: 3          score: 34.549999                remark:bz
name: fas               age: 3342               score: 0.000000         remark:sd
```

调试时观察到的前 20 个字节：

```
(gdb) x/20xb message
0x7fffffffd350: 0x6c    0x78    0x6a    0x00    0x13    0x00    0x00    0x00
0x7fffffffd358: 0x1c    0x42    0x61    0x62    0x63    0x64    0x65    0x00
0x7fffffffd360: 0x72    0x69    0x67    0x65
```

两种方式展示的 message 前 20 字节是一致的。

第 0 个学生的 score 是 39，以 float 形式存储时的编码应当为 0|10000100|00111000000000000000000。由图可知内存中存储的 score 字节分别为 0x00,0x00,0x1c,0x42，转换成大端序为 0x42,0x1c,0x00,0x00，与上述编码完全符合。

student 结构体中，首先存储占 8 个字节的 name 字符串，紧接着存储占 2 个字节的 short 类型，接下来有 2 个字节是空字节，然后才是占 4 个字节的 float 类型，最后紧接着 char[]类型字符串。

（2）任务 2 的算法思想、运行结果等记录

absVal 首先检查 x 的第 31 位（符号位），如果该位为 1，则 x 为负数，通过(~x)+1 的方式将 x 变为相反数，否则 x 不变。

negate 将正数变成负数其实就是将原码变为负数补码，即(~x)+1。而这一操作可逆，所以可以同样用于负数变成正数。

bitAnd 先考虑单独的一位要怎么实现即可（位运算中每位独立，扩展到整个数一定是对的），要想让只有两个数都是 1 时返回 1，考虑到或运算只有两个数都是 0 时返回 0 具有对称性，就可以先将两个数取反再或运算，这时候如果两个数都是 1 就会返回 0，取反一下即可，也就是~((~x) | (~y))。

bitOr 与 bitAnd 同理，返回~((~x) & (~y))。

bitXor 可以借助之前已经实现好的 bitAnd 和 bitOr，同样只考虑一位，只要(~x)&y 和 x&(~y)有一个是 1,就说明 x≠y，需要返回 1，因此最后使用 or 运算即可，即 bitOr(bitAnd(~x, y), bitAnd(x, ~y))。

isTmax 可以发现最大的正数为 01111...111，取反后与原数+1 相等。但这只是必要条件并不充分，因为-1 也满足上述条件。所以在此基础上判断 x+1 不等于 0 即可。

bitCount 采用类似于倍增的方法，首先截取 x 中的奇数位（可以通过 x&0x55555555 获得），然后与剩下的偶数位右移一位对齐并相加。得到的数 x’满足 x 的第 2k+1 位和第 2k 位组成的二进制数中 1 的个数恰好是 x’的第 2k+1 位和第 2k 位表示的二进制数；以此类推，截取 x 中模 4 余 0、1 的位（可以通过 x&0x33333333 获得），然后与剩下的位右移两位对齐后相加，得到的 x’’ 满足 x 的第 4k+3 位到 4k 位组成的二进制数中 1 的个数恰好是 x’’ 的第 4k+3 位到第 4k 位表示的二进制数。进行 5 次以上操作后，x’’’’’’ 表示的数即为所求的 bitCount。这一做法正确性的基

础建立在对于任意一个二进制数 x，其 1 的个数的二进制数表示不会超过 x 所用的二进制位的表示极限。运算次数为 20 次。

bitMask 先构造一个从 highbit 到最低位全位 1 的二进制数，然后通过右移 lowbit 位再左移 lowbit 位将其低 lowbit 位全置为 0。运算次数为 7 次。

addOK 先考虑除了 x 和 y 向最高位的进位数 ad(ad=0,1)，设 x,y 的最高位分别为 xh,yh，那么根据有符号数的表示性质，当且仅当 xh=yh=1 且 ad=0（向下溢出）和 xh=yh=0 且 ad=1（向上溢出）时会发生溢出，运算次数 16 次。

byteSwap 考虑先获取 x 的第 n 和 m 个字节表示的数（如(x>>(n<<3))&255），然后将 x 的第 n 和 m 个字节都置为 0（用异或操作），再把这两个字节分别装在第 m 和 n 个字节上（用异或和或都可）。运算次数 14 次。

程序中还写出了每个函数的最简易实现方式，并与每个正常实现的函数通过单元测试进行 assert 比较，证明了代码中的函数实现正确性。

四、体会

在完成数据压缩与解压的编程任务中，我深刻体会到结构体和内存管理的重要性。压缩和解压过程让我认识到不同的字节序和数据存储方式对程序运行的影响。同时，通过 GDB 调试，我能够直观地观察内存数据，验证程序的正确性。这一过程不仅提升了我的编程能力，还加深了我对数据处理和计算机系统底层机制的理解，为今后的学习打下了坚实的基础。

在编写位运算程序的任务中，我对有符号和无符号整数以及 IEEE 标准浮点数在计算机中的表示方法有了更深入的理解，能够清晰地理解原码补码移码的概念，并且这些实验有的题目难度很大（如 bitCount）激发了我长时间的思考，在思考中我的思维得到了锻炼，也收获了更深层的知识。

五、源码

实验任务 1、2 的源程序（单倍行距，5 号宋体字）

```
// task1.c
# include <stdio.h>
# include <string.h>

typedef struct student{
    char name[8];
    short age;
    float score;
    char remark[200];
}student;

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

# define N 5

student old_s[N], new_s[N];
```

```
int pack_student_bytebybyte(student *s, int sno, char *buf) {
    char *bufptr = buf;
    for(int i = 0; i < sno; i++) {
        for(int j = 0; s[i].name[j] != '\0'; j++) {
            *bufptr = s[i].name[j];
            bufptr++;
        }
        *bufptr = '\0'; bufptr++;
        char *temptr = (char *)&s[i].age;
        for(int j = 0; j < sizeof(s[i].age); j++) {
            *bufptr = *temptr;
            bufptr++; temptr++;
        }
        temptr = (char *)&s[i].score;
        for(int j = 0; j < sizeof(s[i].score); j++) {
            *bufptr = *temptr;
            bufptr++; temptr++;
        }
        for(int j = 0; s[i].remark[j] != '\0'; j++) {
            *bufptr = s[i].remark[j];
            bufptr++;
        }
        *bufptr = '\0'; bufptr++;
    }
    return bufptr - buf;
}

int pack_student_whole(student *s, int sno, char *buf) {
    char *bufptr = buf;
    for(int i = 0; i < sno; i++) {
        strcpy(bufptr, s[i].name);
        bufptr += strlen(s[i].name);
        *bufptr = '\0'; bufptr++;
        memcpy(bufptr, (char *)&s[i].age, sizeof(s[i].age));
        bufptr += sizeof(s[i].age);
        memcpy(bufptr, (char *)&s[i].score, sizeof(s[i].score));
        bufptr += sizeof(s[i].score);
        strcpy(bufptr, s[i].remark);
        bufptr += strlen(s[i].remark);
        *bufptr = '\0'; bufptr++;
    }
    return bufptr - buf;
}

int restore_student(char *buf, int len, student *s) {
```

```
        char *bufptr = buf;
        student *stuptr = s;
        while(bufptr - buf < len) {
            char *temptr = stuptr->name;
            while(*bufptr != '\0') {
                *temptr = *bufptr;
                temptr++; bufptr++;
            }
            *temptr = *bufptr; bufptr++;
            temptr = (char *)&stuptr->age;
            for(int i = 0; i < sizeof(short); i++) {
                *temptr = *bufptr;
                temptr++; bufptr++;
            }
            temptr = (char *)&stuptr->score;
            for(int i = 0; i < sizeof(float); i++) {
                *temptr = *bufptr;
                temptr++; bufptr++;
            }
            temptr = stuptr->remark;
            while(*bufptr != '\0') {
                *temptr = *bufptr;
                temptr++; bufptr++;
            }
            *temptr = *bufptr; bufptr++;
            stuptr++;
        }
        return stuptr - s;
    }

    int main() {
        char buf[1000];
        for(int i = 0; i < N; i++) {
            scanf("%s%hd%f%s", old_s[i].name, &old_s[i].age, &old_s[i].score,
old_s[i].remark);
        }
        printf("old_s:\n");
        for(int i = 0; i < N; i++) {
            printf("name: %s\t\tage: %hd\t\tscore: %f\t\tremark:%s\n", old_s[i].name,
old_s[i].age, old_s[i].score, old_s[i].remark);
        }
        int buflen1 = pack_student_bytebybyte(old_s, 2, buf);
        int buflen2 = pack_student_whole(old_s + 2, 3, buf + buflen1);
        int stunum = restore_student(buf, buflen1 + buflen2, new_s);
        printf("new_s:\n");
```

```
        for(int i = 0; i < stunum; i++) {
            printf("name: %s\t\tage: %hd\t\tscore: %f\t\tremark:%s\n", new_s[i].name,
new_s[i].age, new_s[i].score, new_s[i].remark);
        }
        return 0;
    }


    //task2.c
    # include <stdio.h>
    # include <stdlib.h>
    # include <time.h>
    # include <assert.h>

    int absVal(int x) {
        if(x >> 31) {
            return (~x) + 1;
        } else {
            return x;
        }
    }

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

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

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

    int bitXor(int x, int y) {
        int v1 = bitAnd(~x, y), v2 = bitAnd(x, ~y);
        return bitOr(v1, v2);
    }

    int isTmax(int x) {
        return bitAnd(!((x + 1) ^ (~x)), !!(x + 1));
    }

    int bitCount(int xx) {
        unsigned int x = xx;
```

```
        unsigned int t1 = x & 0x55555555, t2 = x ^ t1;
        x = t1 + (t2 >> 1);
        t1 = x & 0x33333333; t2 = x ^ t1;
        x = t1 + (t2 >> 2);
        t1 = x & 0x0f0f0f0f; t2 = x ^ t1;
        x = t1 + (t2 >> 4);
        t1 = x & 0x00ff00ff; t2 = x ^ t1;
        x = t1 + (t2 >> 8);
        t1 = x & 0x0000ffff; t2 = x ^ t1;
        x = t1 + (t2 >> 16);
        return x;
    }

    int bitMask(int highbit, int lowbit) {
        if(!(highbit ^ 31)) {
            return 0xffffffff >> lowbit << lowbit;
        } else {
            return ((1u << highbit + 1) + 0xffffffff) >> lowbit << lowbit;
        }
    }

    int addOK(int x, int y) {
        int xh = (x >> 31) & 1, yh = (y >> 31) & 1, xo = x & 0x7fffffff, yo = y &
0x7fffffff, ad = ((xo + yo) >> 31) & 1;
        if(xh & yh & (!ad) | (!xh) & (!yh) & ad) return 1;
        else return 0;
    }

    int byteSwap(int x, int n, int m) {
        int u = n << 3, v = m << 3;
        int x1 = (x >> u) & 255, x2 = (x >> v) & 255;
        return x ^ (x1 << u) ^ (x2 << v) ^ (x1 << v) ^ (x2 << u);
    }

    int absVal_standard(int x)  { return (x < 0) ? -x : x;}

    int netgate_standard(int x)  { return -x;}

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

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

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

    int isTmax_standard(int x) {return x == 0x7fffffff;}
```

```
int bitCount_standard(int x) {
    int ans = 0;
    for(int i = 0; i < 32; i++) ans += (x & 1), x >>= 1;
    return ans;
}

int bitMask_standard(int highbit, int lowbit) {
    unsigned int ans = 0;
    for(int i = lowbit; i <= highbit; i++) ans |= 1u << i;
    return ans;
}

int addOK_standard(int x, int y) {
    long long xl = x, yl = y, sum = xl + yl;
    if(sum > __INT_MAX__ || sum < (int)0x80000000) return 1;
    else return 0;
}

int byteSwap_standard(int x, int n, int m) {
    int tmp = x;
    char *p = (char *)&tmp;
    int a = *(p + n);
    int b = *(p + m);
    *(p + n) = b;
    *(p + m) = a;
    return tmp;
}

int myrand() {
    int x = rand();
    if(rand() % 2) x = -x;
    return x;
}

int main() {
    srand(time(0));

    puts("abs_test...");
    for(int i = 0; i < 10; i++) {
        int x = myrand();
        assert(absVal(x) == absVal_standard(x));
        printf("    %d-th passed\n", i);
    }
    puts("abs_test passed");
```

```
puts("negate_test...");
for(int i = 0; i < 10; i++) {
    int x = myrand();
    assert(negate(x) == netgate_standard(x));
    printf("    %d-th passed\n", i);
}
puts("negate_test passed");

puts("and_test...");
for(int i = 0; i < 10; i++) {
    int x = myrand(), y = myrand();
    assert(bitAnd(x, y) == bitAnd_standard(x, y));
    printf("    %d-th passed\n", i);
}
puts("and_test passed");

puts("or_test...");
for(int i = 0; i < 10; i++) {
    int x = myrand(), y = myrand();
    assert(bitOr(x, y) == bitOr_standard(x, y));
    printf("    %d-th passed\n", i);
}
puts("or_test passed");

puts("xor_test...");
for(int i = 0; i < 10; i++) {
    int x = myrand(), y = myrand();
    assert(bitXor(x, y) == bitXor_standard(x, y));
    printf("    %d-th passed\n", i);
}
puts("xor_test passed");

puts("tmax_test...");
for(int i = 0; i < 10; i++) {
    int x;
    if(i == 0) x = 0x7fffffff;
    else if(i == 1) x = -1;
    else x = myrand();
    assert(isTmax(x) == isTmax_standard(x));
    printf("    %d-th passed\n", i);
}
puts("tmax_test passed");

puts("bitcount_test...");
```

```
    for(int i = 0; i < 10; i++) {
        int x = myrand();
        assert(bitCount(x) == bitCount_standard(x));
        printf("    %d-th passed\n", i);
    }
    puts("bitcount_test passed");

    puts("bitmask_test...");
    for(int i = 0; i < 10; i++) {
        int x = rand() % 32, y = rand() % 32;
        if(x < y) {
            int t;
            t = x; x = y; y = t;
        }
        if(i < 3) x = 31;
        assert(bitMask(x, y) == bitMask_standard(x, y));
        printf("    %d-th passed\n", i);
    }
    puts("bitmask_test passed");

    puts("addok_test...");
    for(int i = 0; i < 10; i++) {
        int x = myrand(), y = myrand();
        assert(addOK(x, y) == addOK_standard(x, y));
        printf("    %d-th passed\n", i);
    }
    puts("addok_test passed");

    puts("byteswap_test...");
    for(int i = 0; i < 10; i++) {
        int x = myrand(), n = rand() % 4, m = rand() % 4;
        assert(byteSwap(x, n, m) == byteSwap_standard(x, n, m));
        printf("    %d-th passed\n", i);
    }
    puts("byteswap_test passed");
    return 0;
}
```