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 2202

学　　号：U202215378

姓　　名：冯瑞琦

指导教师：王多强

2024 年 3 月 26 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 数据存放的压缩与解压编程

1.1 算法思想

先初始化数据，使用结构体数组 old_s 存储学生信息，每个学生有姓名、年龄、分数和备注。初始化 old_s 中的数据，第一个学生姓名为自己名字，分数为学号后两位。这里直接给出了 5 个学生的信息，避免了实验过程中的多次输入。为了之后便于对比，先使用循环遍历 old_s 数组，输出每个学生的姓名、年龄、分数和备注信息，输出一遍初始数据。使用 sizeof()函数计算 old_s 数组中存放数据的总长度，以字节为单位，计算压缩前存放数据的长度。

压缩数据：调用 pack_student_bytebybyte()函数压缩前两条数据，调用 pack_student_whole()函数压缩后三条数据。pack_student_bytebybyte()函数逐字节读取学生结构体中的数据，并按照指定的格式存储到 buf 中。pack_student_whole()函数则通过 strcpy()函数一次性读入姓名和备注信息，再将年龄和分数以及剩余的字符数据直接复制到 buf 中。

输出压缩后的数据：调用 coutmessage()函数输出压缩后的数据，以及压缩后存放数据的长度。输出 message 前 20 个字节的 16 进制表示：调用 print_hex()函数输出压缩后的 message 的前 20 个字节的 16 进制表示。

解压数据：调用 restore_student()函数解压压缩后的数据，将数据存储到 new_s 数组中。restore_student()函数根据压缩规则逐个读取压缩后的数据，并将其解压还原成学生结构体。最后输出解压后的结果。

各主要函数的具体算法描述如下：

(1) pack_student_bytebybyte 函数

接收三个参数：student* s 表示待压缩数组的起始地址，int sno 表示待压缩的学生数量，char* buf 表示压缩存储区的首地址。使用指针 p 指向 s，并使用指针 p0 指向 buf，以便逐字节复制数据。循环遍历每个学生的信息：逐字节复制学生姓名 name，直到遇到空字符 ‘\0’ 为止，如果 name 未填满 8 个字节，用零填充。直接复制 age 和 score 的数据，分别占用 2 个字节和 4 个字节。逐字节复制学生备注 remark，直到遇到空字符‘\0’为止，如果 remark 未填满 200 个字节，用零填充。返回值为压缩后的字节数。对于 pack_student_whole 函数，字符数组调用 strcpy 函数直接复制，age、score 则用 p 指针移动到这个数据的首字节，再强制转换把剩下的字节作为整体读入。最后用指向压缩数组末尾的指针减去基址 buf 的值得到字节数。

(2) 对于 restore_student 函数

可以用与压缩函数相反的方法进行解压。具体步骤如下。

接收三个参数：student* s 表示待压缩数组的起始地址，int sno 表示待压缩的学生数量，char* buf 表示压缩存储区的首地址。使用指针 p0 指向 buf，以便复制数据。循环遍历每个学生的信息，使用 strcpy() 函数一次性复制学生姓名 name，并直接复制 age 和 score 的数据，分别占用 2 个字节和 4 个字节。最后，使用 strcpy() 函数一次性复制学生备注 remark。返回值为压缩后的字节数。

(3) restore_student() 函数

接收三个参数：char* buf 表示压缩存储区的首地址，int len 表示压缩后的数据长度，student* s 表示存放解压数据的结构数组的起始地址。使用指针 p 指向 buf，以便逐个读取压缩后的数据。循环读取压缩后的数据，直到读取的字节数达到 len。在每一轮循环中，使用 strcpy() 函数从 buf 中读取学生姓名 name，直到遇到空字符 '\0' 为止，并直接读取 age 和 score 的数据，分别占用 2 个字节和 4 个字节。最后，使用 strcpy() 函数从 buf 中读取学生备注 remark，直到遇到空字符 '\0' 为止。返回解压后的学生数量。具体实现如下图。

```
191  int  restore_student(char* buf, int len, student* s)
192  {
193      int cnts = 0;
194      char* p = buf;//遍历message数组
195      while (p - buf < len)//当从message数组中读入的数据不够len
196      {
197          strcpy(s[cnts].name, p);//读入name
198          p += strlen(s[cnts].name) + 1;//p指针移动到age处
199          s[cnts].age = *((short*)p);//读入age
200          p += 2;//p指针移动到score处
201          s[cnts].score = *((float*)p);//读入score
202          p += 4;//p指针移动到remark处
203          strcpy(s[cnts].remark, p);//读入remark
204          p += strlen(s[cnts].remark) + 1;//p指针移动到下一name处
205          cnts++;//增加学生数
206      }
207      return cnts;//返回学生数
208  }
```

(4) coutmessage() 函数：

接收两个参数：char* buf 表示压缩存储区的首地址，int len 表示压缩后的数据长度。使用指针 p 指向 buf，以便逐个读取压缩后的数据。循环读取压缩后的数据，直到读取的字节数达到 len。在每一轮循环中，分别输出学生姓名 name、年龄 age、分数 score 和备注 remark。最后，返回解压后的学生数量。具体实现如下图。

```
210  int  coutmessage(char* buf, int len)
211  {
212      int cnts = 0;
213      char* p = buf;//遍历message数组
214      while (p - buf < len)//当从message数组中读入的数据不够len
215      {
216          printf("%s ", p);
217          p += strlen(p) + 1;//p指针移动到age处
218          printf("%hd ", *((short*)p));
219          p += 2;//p指针移动到score处
220          printf("%f ", *((float*)p));
221          p += 4;//p指针移动到remark处
222          printf("%s\n", p);
223          p += strlen(p) + 1;
224          cnts++;//增加学生数
225      }
226      return cnts;//返回学生数
227  }
```

1.2 运行结果

(1) 压缩数据

压缩前的数据初始化如左图所示，压缩完后如右图所示。

```
name=frq
age=20
score=78.000000
remark=great

name=lh
age=99
score=100.000000
remark=good

name=xm
age=35
score=55.000000
remark=soso

name=cmq
age=19
score=23.000000
remark=excellent

name=wyn
age=34
score=65.000000
remark=wow

压缩前存放数据的长度为1080
```

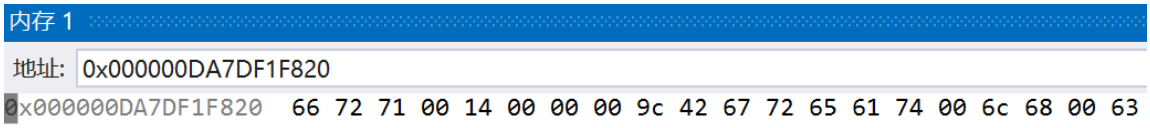
```
frq 20 78.000000 great
lh 99 100.000000 good
xm 35 55.000000 soso
cmq 19 23.000000 excellent
wyn 34 65.000000 wow

压缩后存放数据的长度为78
```

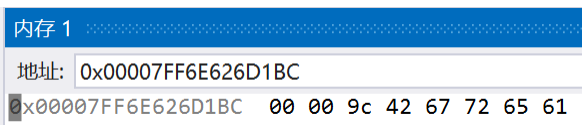
message 前 20 个字节的 16 进制表示如下图所示。

```
message前20字节的16进制表示：
66 72 71 00 14 00 00 00 9C 42 67 72 65 61 74 00 6C 68 00 63
```

调试时在内存窗口观察到的 message 的前 20 个字节如下图所示。经比较，两者完全一致。

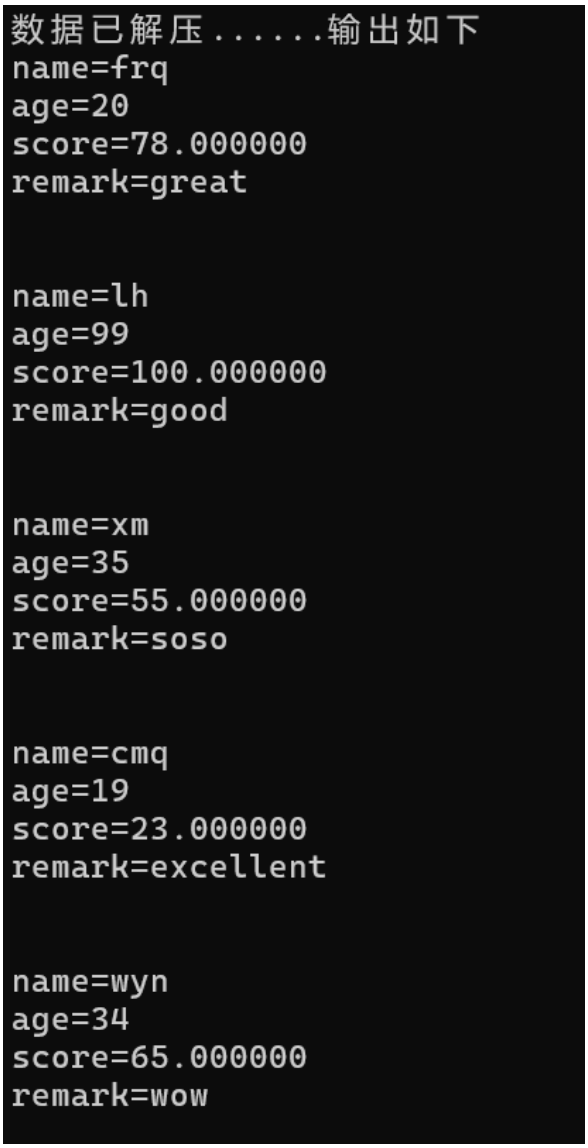


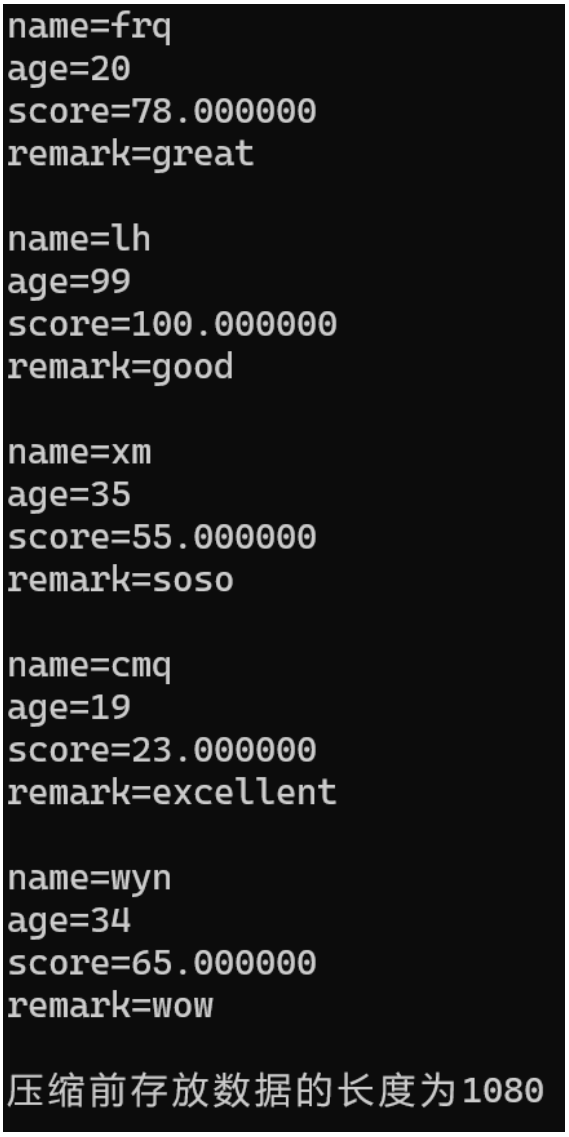
第 0 个学生的 score 为学号后两位，即 78.000000。根据浮点数的编码规则，78 为正数，因此符号位为 0。先将 78 转换为二进制，得到 1001110。然后规格化为 1.001110 * 2^6。指数部分为 6，因为偏移值为 127（单精度浮点数的偏移值），所以实际指数为 6 + 127 = 133，即 10000101。尾数部分：由规格化后的二进制小数部分组成，即 001110，最终 16 进制表示应为 429c0000H，在实际计算机中存放方式不同，因此 42 和 9c 位置交换，另外可能由于精度不同，尾数更低位有一些不同。与观察到的内存表示（如下图）比较，结果基本一致，最终输出结果一致。



(2) 解压数据

解压后的数据再次输出，如下图（左）所示，与解压前数据（下图右）比较，结果一致。





通过实验，总结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 型的数的存放规律如下。

结构数组中每个结构体元素占用的字节数取决于结构体内部成员的数据类型及其对齐方式。结构体数组的地址计算方法与一般数组相同，即第一个元素的地址加上索引乘以结构体的大小。结构体内成员的地址计算遵循结构体内存布局规则，一般按照成员的定义顺序依次存放。

char 类型将字符与 ASCII 码对应，实际上相当于一字节的整数。字符串数组占用的字节数为字符串的长度加上一个额外的字节来存储字符串结束符‘\0’，未初始化的部分存储不确定的初值。

short 类型占用两个字节，在计算机内表示为二进制数。但是内存中为了对齐，两个变量之间还是会间隔 4 个字节。

float 类型采用 IEEE754 标准，有符号位，指数位和尾数位，其中尾数位能够表示一定精度内的 0~2 的数，指数位的计算方式想当于一个 unsigned 二进制整数值减去一个偏移量。

数据存储单元的地址表达形式和地址计算方法：

一般而言，相邻定义数据时其地址的差值为 32 个字节，当定义的数据长度超过 4 字节则差值为 36 字节，数组取地址时取到的是其第一个元素的地址。地址计算方法下标为 i 的元素的地址等于首个元素的地址加上 i*数组元素类型对应的字节数，当定义结构体时，会有特殊的对齐原则。

2. 任务 2　编写数据表示的自动评测程序

2.1 算法思想

(1) 绝对值函数 absVal(int x)

通过位运算来实现。首先根据 x 的符号位生成一个掩码 mask，然后使用异或运算符（^）来反转 x 的所有位，接着加上 mask，最后加上 ~mask + 1 来处理 x 为负数时的情况。

(2) 取反函数 negate(int x)

直接使用了二进制的按位取反（~）和加 1 的操作，模拟了负数的表示。

(3) 按位与函数 bitAnd(int x, int y):

利用了按位取反（~）和按位或（|）操作来实现按位与的功能。通过对 x 和 y 分别取反后再取反的方式，实现了按位与的效果。

(4) 按位或函数 bitOr(int x, int y)

也使用了按位取反和按位与操作，但是是对 x 和 y 分别取反后再取反，模拟了按位或。

(5) 按位异或函数 bitXor(int x, int y)

使用了按位与、按位或和按位取反操作来实现异或功能，通过对 x 和 y 的取反操作以及对结果再次取反的方式，实现了按位异或。

(6) 判断最大正整数函数 isTmax(int x)

通过判断 x+1 和~x 是否相等，以及 x 是否不为零，来判断 x 是否为最大的正整数。

(7) 统计二进制表示中 1 的个数函数 bitCount(int x)

不断地将 x 与 1 进行按位与操作，然后右移一位，直到 x 变为 0，同时统计按位与结果为 1 的次数来计算二进制表示中 1 的个数。

(8) 生成特定位数为 1 的掩码函数 bitMask(int highbit, int lowbit)

对全 1 的二进制数左移 highbit+1 位，然后减去 lowbit 位全 1 的二进制数得到结果，这样就生成了从 lowbit 到 highbit 位全为 1，其他位为 0 的数。

(9) 检查加法溢出函数 addOK(int x, int y)

通过对 x 和 y 相加得到 sum，然后检查 sum 与 x、sum 与 y 的异或结果的符号位来判断是否溢出。

(10) 字节交换函数 byteSwap(int x, int n, int m)

通过将 x 右移 n*m 位，然后与掩码进行按位与操作，得到要交换的两个字节的值，然后将这两个字节交换位置，最后将结果合并返回。

2.2 运行结果

运行结果如下图所示，设计了标准函数自动判断结果是否正确。下图中所有函数结果都与标准函数结果一致，与任务说明中给出的例子及结果也相同，说明功能均已成功实现。

```
1. absVal(-1) = 1,
   absVal_standard(-1) = 1
2. negate(2) = -2,
   negate_standard(2) = -2
3. bitAnd(12, 6) = 4,
   bitAnd_standard(12, 6) = 4
4. bitOr(5, 3) = 7,
   bitOr_standard(5, 3) = 7
5. bitXor(5, 3) = 6,
  bitXor_standard(5, 3) = 6
6. isTmax(0x7FFFFFFF) = 1,
   isTmax_standard(0x7FFFFFFF) = 1
7. bitCount(15) = 4,
   bitCount_standard(15) = 4
8. bitMask(5, 3) = 38,
   bitMask_standard(5, 3) = 38
9. addOK(0x7FFFFFFF, 1) = 0,
   addOK_standard(0x7FFFFFFF, 1) = 0
10. byteSwap(0x12345678, 1, 3) = 56341278,
    byteSwap(0xDEADBEEF, 0, 2) = DEEFBEAD
```

四、体会

完成任务 1 的过程中，我学到了许多关于结构体和内存操作的知识。在这个任务中，我需要实现对学生结构体的逐字节压缩和整体压缩，并且能够将压缩后的数据恢复成原来的结构体。这要求我对结构体在内存中的存储方式有深入的了解。

在实现逐字节压缩函数时，我需要逐个字节地读取结构体的各个成员，并且要考虑到成员之间的内存对齐问题，这让我更加深入地理解了结构体内存布局的细节。而在整句读入压缩函数中，我利用了 strcpy 函数，将整个字符串一次性地拷贝进缓冲区，这种方法更加简洁高效。

在实现解压函数时，我需要根据压缩后的字节流逐个读取出结构体的各个成员，并且要正确地将字节流中的二进制数据转换为对应的数据类型，这让我对于指针操作和类型转换有了更深入的认识。

在测试和输出函数中，我需要仔细处理字节流的长度和结尾标志，以确保正确地输出压缩和解压后的数据。通过使用 VS，我学会了在调试中查看变量在内存中的地址和值，能够很方便地观察比较内存值和自己函数中输出的结果是否一致，判断功能的正确性。

完成任务 1 后，我对于结构体、内存布局、指针操作和类型转换有了更深入的理解。

在实现任务 2 的功能时，我深入学习了 C 语言中的位操作原理和技巧。通过实现绝对值、取反、与、或、异或等基本位操作函数，并与标准函数进行对比，我加深了对位操作的理解。在这个过程中，我学会了使用位操作符（如&、|、^、<<、>>）来实现各种功能，同时也注意到了特殊情况的处理，比如绝对值函数中对负数的处理以及加法函数中溢出的处理。通过与标准函数的对比验证，我不仅能够确认自己的实现是否正确，还可以发现潜在的错误或改进空间。通过位操作实现功能常常比直接用标准函数繁琐，但是能够通过单单几个运算符就实现同样的功能，也体现了位操作符的地位重要，应用范围广泛，可以实现强大功能。

总体来说，通过本次实验，我掌握了程序开发平台(VS2019)的基本用法，包括程序的编译、链接和调试，熟悉了地址的计算方法、地址的内存转换，还熟悉数据的表示形式。

五、源码

实验任务 1、2、3 的源程序（单倍行距，5 号宋体字）

1. 任务 1 数据存放的压缩与解压编程

```
// 计算机系统实验task1.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

#pragma warning(disable:4996)
#include<stdio.h>
#include<string.h>
struct student {
    char  name[8];
    short age;
    float score;
    char  remark[200];  // 备注信息
};//student结构占用字节多   13*16+8=216 8+4(2)+4+200
student  old_s[5];    // old_s[0].name 为自己的姓名，score为学号后两位;
student  new_s[5];

int  pack_student_bytebybyte(student* s, int sno, char* buf);//逐字节压缩
int  pack_student_whole(student* s, int sno, char* buf);//整句读入，要求使用strcpy
int  restore_student(char* buf, int len, student* s);//修复函数
int  coutmessage(char* buf, int len);//输出函数

void print_hex(char* data, int len) {//输出message前20个字节的16进制表示
    int i;
    printf("message前20字节的16进制表示:\n");
    for (i = 0; i < len; i++) {
        printf("%02X ", (unsigned char)data[i]);
    }
    printf("\n\n");
}

int main()
{
    /*初始化*/
    int cnts = 0;
    char message[505];
    strcpy(old_s[0].name, "frq");
    old_s[0].age = 20;
    old_s[0].score = 78;
    strcpy(old_s[0].remark, "great");
    strcpy(old_s[1].name, "lh");
    old_s[1].age = 99;
```

```
    old_s[1].score = 100;
    strcpy(old_s[1].remark, "good");
    strcpy(old_s[2].name, "xm");
    old_s[2].age = 35;
    old_s[2].score = 55;
    strcpy(old_s[2].remark, "soso");
    strcpy(old_s[3].name, "cmq");
    old_s[3].age = 19;
    old_s[3].score = 23;
    strcpy(old_s[3].remark, "excellent");
    strcpy(old_s[4].name, "wyn");
    old_s[4].age = 34;
    old_s[4].score = 65;
    strcpy(old_s[4].remark, "wow");
    //这里直接给出结构体的数据，避免了实验过程中多次的输入

    while (cnts < 5)
    {
        printf("name=%s\n", old_s[cnts].name);
        printf("age=%hd\n", old_s[cnts].age);
        printf("score=%f\n", old_s[cnts].score);
        printf("remark=%s\n", old_s[cnts].remark);
        printf("\n");
        cnts++;
    }//输出压缩前的结果

    cnts = 0;
    printf("压缩前存放数据的长度为%u\n\n", sizeof(student) * 5);//输出压缩前存放数据的长
度

    int cntbuf = pack_student_bytebybyte(old_s, 2, message);//用第一种压缩函数压缩第一条
数据

    cntbuf += pack_student_whole(old_s + 2, 3, message + cntbuf);//用第二种压缩函数压缩
后两条数据

    coutmessage(message, cntbuf);//输出压缩后的数据
    printf("\n压缩后存放数据的长度为%d\n\n", cntbuf);//输出压缩后存放数据的长度
    print_hex(message, 20);
    int sno = restore_student(message, cntbuf, new_s);//用解压函数解压数据
    printf("数据已解压.....输出如下\n");
    while (cnts < sno)
    {
        printf("name=%s\n", new_s[cnts].name);
        printf("age=%hd\n", new_s[cnts].age);
```

```
        printf("score=%f\n", new_s[cnts].score);
        printf("remark=%s\n", new_s[cnts].remark);
        printf("\n\n");
        cnts++;
    }//输出解压后的结果
    return 0;
}

int  pack_student_bytebybyte(student* s, int sno, char* buf)
{
    int cnts = 0, cntname = 0, cntage = 0, cntscore = 0, cntremark = 0, cntbuf = 0;
    char* p = (char*)s;
    char* p0 = buf;
    while (cnts < sno)
    {
        //读入name数组
        cntname = 0;
        while (cntname < 8){
            if (*p) {
                *p0 = *p;
                cntname++;
                cntbuf++;
                p++;
                p0++;
            }
            else{
                *p0 = 0;
                cntbuf++;
                p += (8 - cntname);
                p0++;
                break;
            }
        }
        //读入short
        cntage = 0;
        while (cntage < 2)
        {
            *p0 = *p;
            cntage++;
            cntbuf++;
            p++;
            p0++;
        }
        p += 2;
        //读入float
```

```
        cntscore = 0;
        while (cntscore < 4)
        {
            *p0 = *p;
            cntscore++;
            cntbuf++;
            p++;
            p0++;
        }
        //读入remark数组
        cntremark = 0;
        while (cntremark < 200)
        {
            if (*p){
                *p0 = *p;
                cntremark++;
                cntbuf++;
                p++;
                p0++;
            }
            else{
                *p0 = 0;
                cntbuf++;
                p += (200 - cntremark);
                p0++;
                break;
            }
        }
        cnts++;
    }
    return cntbuf;//返回压缩后的字节数
}
int  pack_student_whole(student* s, int sno, char* buf)
{
    int cnts = 0;
    char* p0 = buf;
    char* p = NULL;
    while (cnts < sno)
    {
        p = s[cnts].name;
        strcpy(p0, p);
        p0 += (strlen(p) + 1);//读入name数组

        p = (char*)&s[cnts].age;
        *((short*)p0) = *((short*)p);
```

```
        p0 += 2;//读入age

        p = (char*)&s[cnts].score;
        *((float*)p0) = *((float*)p);
        p0 += 4;//读入score

        p = s[cnts].remark;
        strcpy(p0, p);
        p0 += (strlen(p) + 1);//读入remark数组

        cnts++;
    }
    return p0 - buf;

}
int  restore_student(char* buf, int len, student* s)
{
    int cnts = 0;
    char* p = buf;//遍历message数组
    while (p - buf < len)//当从message数组中读入的数据不够len
    {
        strcpy(s[cnts].name, p);//读入name
        p += strlen(s[cnts].name) + 1;//p指针移动到age处
        s[cnts].age = *((short*)p);//读入age
        p += 2;//p指针移动到score处
        s[cnts].score = *((float*)p);//读入score
        p += 4;//p指针移动到remark处
        strcpy(s[cnts].remark, p);//读入remark
        p += strlen(s[cnts].remark) + 1;//p指针移动到下一name处
        cnts++;//增加学生数
    }
    return cnts;//返回学生数
}

int  coutmessage(char* buf, int len)
{
    int cnts = 0;
    char* p = buf;//遍历message数组
    while (p - buf < len)//当从message数组中读入的数据不够len
    {
        printf("%s ", p);
        p += strlen(p) + 1;//p指针移动到age处
        printf("%hd ", *((short*)p));
        p += 2;//p指针移动到score处
        printf("%f ", *((float*)p));
```

```
            p += 4;//p指针移动到remark处
            printf("%s\n", p);
            p += strlen(p) + 1;
            cnts++;//增加学生数
        }
        return cnts;//返回学生数
    }
```

2. 任务 2 编写数据表示的自动评测程序

```
#include <stdio.h>
/* 1. 返回 x 的绝对值 */
int absVal(int x) {
    int mask = x >> 31;
    return (x ^ mask) + (~mask + 1);
}
/* 标准函数 */
int absVal_standard(int x) {
    return (x < 0) ? -x : x;
}
/* 2. 实现 -x */
int negate(int x) {
    return ~x + 1;
}
/* 标准函数 */
int negate_standard(int x) {
    return -x;
}
/* 3. 实现 & */
int bitAnd(int x, int y) {
    return ~(~x | ~y);
}
/* 标准函数 */
int bitAnd_standard(int x, int y) {
    return x & y;
}
/* 4. 实现 | */
int bitOr(int x, int y) {
    return ~((~x) & (~y));
}
/* 标准函数 */
int bitOr_standard(int x, int y) {
    return x | y;
}
/* 5. 实现 ^ */
int bitXor(int x, int y) {
    return ~(~(x & ~y) & ~(~x & y));
```

```
}
/* 标准函数 */
int bitXor_standard(int x, int y) {
    return x ^ y;
}
/* 6. 判断x是否为最大的正整数（7FFFFFFF） */
int isTmax(int x) {
    return !((x + 1) ^ (~x)) & !!x;
}
/* 标准函数 */
int isTmax_standard(int x) {
    return x == 0x7FFFFFFF;
}
/* 7. 统计x的二进制表示中 1 的个数 */
int bitCount(int x) {
    int count = 0;
    while (x) {
        count += x & 1;
        x >>= 1;
    }
    return count;
}
/* 标准函数 */
int bitCount_standard(int x) {
    int count = 0;
    while (x) {
        x &= (x - 1);
        count++;
    }
    return count;
}
/* 8. 产生从lowbit 到 highbit 全为1，其他位为0的数 */
int bitMask(int highbit, int lowbit) {
    return (~(~0 << (highbit - lowbit + 1))) << lowbit;
}
/* 标准函数 */
int bitMask_standard(int highbit, int lowbit) {
    return (1 << (highbit + 1)) - (1 << lowbit);
}
/* 9. 当x+y 会产生溢出时返回1，否则返回 0 */
int addOK(int x, int y) {
    int sum = x + y;
    return !((sum ^ x) & (sum ^ y)) >> 31;
}
/* 标准函数 */
```

```
int addOK_standard(int x, int y) {
    int sum = x + y;
    return !(x > 0 && y > 0 && sum <= 0) && !(x < 0 && y < 0 && sum >= 0);
}
/* 10. 将x的第n个字节与第m个字节交换，返回交换后的结果 */
int byteSwap(int x, int n, int m) {
    int mask = 0xFF;
    int byte_n = (x >> (n << 3)) & mask;
    int byte_m = (x >> (m << 3)) & mask;
    x = x & ~(mask << (n << 3)) | (byte_m << (n << 3));
    x = x & ~(mask << (m << 3)) | (byte_n << (m << 3));
    return x;
}
int main() {
    printf("1. absVal(-1) = %d, \n      absVal_standard(-1) = %d\n", absVal(-1),
absVal_standard(-1));
    printf("2. negate(2) = %d, \n       negate_standard(2) = %d\n", negate(2),
negate_standard(2));
    printf("3. bitAnd(12, 6) = %d, \n   bitAnd_standard(12, 6) = %d\n", bitAnd(12, 6),
bitAnd_standard(12, 6));
    printf("4. bitOr(5, 3) = %d, \n     bitOr_standard(5, 3) = %d\n", bitOr(5, 3),
bitOr_standard(5, 3));
    printf("5. bitXor(5, 3) = %d, \n   bitXor_standard(5, 3) = %d\n", bitXor(5, 3),
bitXor_standard(5, 3));
    printf("6. isTmax(0x7FFFFFFF) = %d, \n     isTmax_standard(0x7FFFFFFF) = %d\n",
isTmax(0x7FFFFFFF), isTmax_standard(0x7FFFFFFF));
    printf("7. bitCount(15) = %d, \n    bitCount_standard(15) = %d\n", bitCount(15),
bitCount_standard(15));
    printf("8. bitMask(5, 3) = %X, \n   bitMask_standard(5, 3) = %X\n", bitMask(5, 3),
bitMask_standard(5, 3));
    printf("9. addOK(0x7FFFFFFF, 1) = %d, \n   addOK_standard(0x7FFFFFFF, 1) = %d\n",
addOK(0x7FFFFFFF, 1), addOK_standard(0x7FFFFFFF, 1));
    printf("10. byteSwap(0x12345678, 1, 3) = %X, \n    byteSwap(0xDEADBEEF, 0, 2) = %X\n",
byteSwap(0x12345678, 1, 3), byteSwap(0xDEADBEEF, 0, 2));
    return 0;
}
```